



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

GILBERTO AMADO DE AZEVEDO CYSNEIROS FILHO

“FERRAMENTA PARA O SUPORTE DO MAPEAMENTO DA
MODELAGEM ORGANIZACIONAL EM I* PARA UML”

*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

RECIFE, AGOSTO/2001



Pós-Graduação em Ciência da Computação

**“Ferramenta para o Suporte do Mapeamento da
Modelagem Organizacional em *i** para UML”**

Por

Gilberto Amado de Azevedo Cysneiros Filho

Dissertação de Mestrado



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

RECIFE, AGOSTO/2001

Esta dissertação eu dedico à memória de meu pai, umas das pessoas que mais contribuiu na formação do que hoje eu sou e que, certamente, estaria compartilhando intensamente da felicidade que hoje eu tenho em concluir mais este trabalho.

AGRADECIMENTOS

Este trabalho é fruto da cooperação de várias pessoas. Primeiramente, eu gostaria de agradecer a meu orientador, Professor Jaelson Freire Brelaz de Castro, pelo incentivo dado ao meu interesse à pesquisa, desde a época da minha graduação no curso de Ciência da Computação. E pela sua orientação, dicas e o contínuo interesse no meu trabalho sem o qual essa tese não poderia ter sido concretizada.

Em segundo lugar, gostaria de agradecer a Professora Fernanda Alencar, pela oportunidade de dar prosseguimento a seu trabalho de doutorado, como também pelas atividades realizadas em conjunto e pelo seu apoio e orientação.

Ao professores do Centro de Informática, principalmente a Alexandre Vanconcelos, Paulo Borba, Augusto Sampaio, Ana Carolina Salgado e Fernando Fonseca que ministraram as disciplinas que eu cursei no mestrado.

Eu desejo também agradecer a todos os membros do grupo do Laboratório de Engenharia de Requisitos, em especial a Victor Santander, Márcia Seabra, Carina Frota, Ana Elizabeth, Marco Toranzo, Marcelo Barbosa e a Márcio Santos.

Aos colegas e funcionários do Centro de Informática, pela atenção e convivência ao longo das minhas atividades no mestrado.

Ao CNPq, pelo recurso financeiro disponibilizado.

Finalmente, eu gostaria de agradecer à minha família e aos meus amigos pelo amor, paciência e apoio. A minha mãe pela dedicação e compreensão. A Laélia Pumilla que sempre me encorajou a fazer o mestrado e pelo seu companheirismo e amizade.

RESUMO

O sucesso das aplicações de informática principalmente nas áreas de tecnologia de informação depende do bom entendimento do ambiente organizacional. Técnicas de modelagem de requisitos podem ser usadas para ajudar a entender o processo organizacional em termos de objetivos, regras de negócio, tarefas, recursos e o relacionamento entre os seus atores.

Nos últimos anos tem se observado uma crescente influência do paradigma orientado a objeto. Infelizmente a técnica dominante de modelagem orientada a objeto, a UML (Unified Modeling Language), não é bem equipada para representar os requisitos organizacionais. Portanto, nós advogamos o uso da técnica *i** para modelagem organizacional cujo enfoque dado é na descrição dos relacionamentos organizacionais entre os vários atores organizacionais, como também no entendimento das razões para as alternativas escolhidas. Enquanto para especificação dos requisitos funcionais nós confiamos na Unified Modeling Language anotada com restrições em OCL (Object Constraint Language).

As ferramentas de automação para as atividades da Engenharia de Software podem melhorar a produtividade e a qualidade do software e servir como suporte no processo de desenvolvimento de software. Contudo, existem poucas ferramentas que apóiam as atividades da Engenharia de Requisitos que tratam dos requisitos organizacionais. Entre estas ferramentas está o OME (Organization Modeling Environment) que suporta a técnica de modelagem organizacional *i**. Por outro lado, existem diversas ferramentas que suportam a modelagem orientada a objeto de sistemas e entre elas se destaca o Rational Rose pela sua capacidade de integração com outras ferramentas e pela sua posição como líder de mercado.

O resultado principal dessa dissertação foi o desenvolvimento do protótipo da ferramenta GOOD (Goal Into Object Oriented Development) que suporta o mapeamento dos elementos do modelo organizacional em *i** armazenados num repositório de dados em Telos pela ferramenta OME para elementos em UML

através da ferramenta Rational Rose. A ferramenta GOOD consiste de dois componentes: Configurar Mapeamento e Executar Mapeamento.

ABSTRACT

The success of computer applications depends on a good understanding of the organizational environment. Requirement modeling techniques may be used to help to understand the organizational process in terms of goals, business rules, tasks, resources and the relationship between their actors.

In recent years we have observed a growing influence of the object-orientation paradigm. Unfortunately the dominant technique of object oriented modeling UML (Unified Modeling Language) is ill equipped to represent the organizational requirements. So, we advocate the use of the *i** technique to model organizational requirements in terms of the organizational relationships among the several organizational actors, as well as a means for understanding the rationale for the decision making. UML, annotated with restrictions in OCL, is used to specify functional requirements.

Automation tools may support the Software Engineering activities improving the productivity software quality. They may also be used as a support in the software development process. However, there are few tools that support the Requirement Engineering activities that deal with organizational requirements. Among them there is the OME (Organization Modeling Environment) that supports the *i** technique. On the other hand, there are several tools that support the object oriented modeling of systems. Amongst them, we point out the Rational Rose because of its ability to integrate with other tools and its high rank in the market.

The main result of this dissertation was the development of prototype tool named GOOD (Goal Object Oriented Development) which support the mapping of *i** organizational models stored in a Telos repository by the OME tool into UML elements by the Rational Rose tool. The GOOD tool consists of two components: Mapping Configuration and Mapping Execution.

Índice

CAPÍTULO 1 **1**

1	INTRODUÇÃO	1
1.1	MOTIVAÇÕES	2
1.2	OBJETIVOS E ABORDAGEM	5
1.3	ESTRUTURA DA DISSERTAÇÃO	5

CAPÍTULO 2 **8**

	ENGENHARIA DE REQUISITOS	8
2.1	INTRODUÇÃO	9
2.2	VISÃO GERAL DA ENGENHARIA DE REQUISITOS	9
2.3	MOTIVAÇÃO PARA O ESTUDO DA ENGENHARIA DE REQUISITOS	12
2.4	CLASSIFICAÇÃO DE REQUISITOS	14
2.5	O PROCESSO DE ENGENHARIA DE REQUISITOS	18
2.6	CONSIDERAÇÕES FINAIS	27

CAPÍTULO 3 **29**

3	MODELAGEM DE REQUISITOS	29
3.1	INTRODUÇÃO	30
3.2	MODELAGEM ORGANIZACIONAL	31
3.3	A TÉCNICA I*	33
3.4	INTRODUÇÃO À UML	43
3.5	DIAGRAMA DE CLASSES	46
3.6	OCL – OBJECT CONSTRAINT LANGUAGE	49
3.7	REGRAS DE NEGÓCIO COM OCL	52
3.8	DIRETRIZES DE MAPEAMENTO DE I* PARA UML	54
3.9	CONSIDERAÇÕES FINAIS	63

CAPÍTULO 4	65
4 FERRAMENTAS OME E RATIONAL ROSE	65
4.1 INTRODUÇÃO	66
4.2 ORGANIZATION MODELING ENVIRONMENT (OME)	68
4.3 RATIONAL ROSE	80
4.4 CONSIDERAÇÕES FINAIS	88
CAPÍTULO 5	89
5 A FERRAMENTA GOOD	89
5.1 INTRODUÇÃO	90
5.2 CONFIGURANDO O ARQUIVO DE MAPEAMENTO	91
5.3 EXECUTANDO O MAPEAMENTO	101
5.4 CONSIDERAÇÕES FINAIS	103
CAPÍTULO 6	104
6 ESTUDO DE CASO	104
6.1 OBJETIVOS	105
6.2 ESPECIFICAÇÃO DO ESTUDO DE CASO	105
6.3 REQUISITOS INICIAIS EM I*	106
6.4 MAPEAMENTO PARA UML	111
6.5 CONSIDERAÇÕES FINAIS	117
CAPÍTULO 7	118
7 CONCLUSÕES E TRABALHOS FUTUROS	118
7.1 CONTRIBUIÇÕES DO TRABALHO	119
7.2 TRABALHOS RELACIONADOS	121
7.3 TRABALHOS FUTUROS	121
APÊNDICE A	123
INSTALANDO A FERRAMENTA GOOD	123

<u>APÊNDICE B</u>	<u>126</u>
CÓDIGO DO <i>SCRIPT</i> DO COMPONENTE CONFIGURAR MAPEAMENTO	126
<u>APÊNDICE C</u>	<u>131</u>
CÓDIGO DO <i>SCRIPT</i> DO COMPONENTE EXECUTAR MAPEAMENTO	131
<u>APÊNDICE D</u>	<u>156</u>
ARQUIVO TELOS DO FRAMEWORK I*	156
<u>REFERÊNCIAS BIBLIOGRÁFICAS</u>	<u>167</u>

Índice de Figuras

<i>Figura 2.1 Custo Relativo para Correção de um Erro</i>	<i>14</i>
<i>Figura 2.2 - Atividades do Processo da Engenharia de Requisitos no Modelo Espiral.....</i>	<i>19</i>
<i>Figura 3.1 Dependência de Objetivo.....</i>	<i>35</i>
<i>Figura 3.2 Dependência Aberta.....</i>	<i>35</i>
<i>Figura 3.3 Dependência Crítica.....</i>	<i>36</i>
<i>Figura 3.4 Dependência de Tarefa</i>	<i>37</i>
<i>Figura 3.5 Dependência de Recurso.....</i>	<i>37</i>
<i>Figura 3.6 Dependência de Objetivo Soft.....</i>	<i>38</i>
<i>Figura 3.7 Agentes, Posições e Papéis.....</i>	<i>39</i>
<i>Figura 3.8 Meta modelo do SD.....</i>	<i>39</i>
<i>Figura 3.9 Exemplo de um Relacionamento Meio Fim</i>	<i>41</i>
<i>Figura 3.10 Exemplo de uma Decomposição de Tarefa.....</i>	<i>43</i>
<i>Figura 3.11 Meta-modelo dos classificadores de UML</i>	<i>47</i>
<i>Figura 3.12 Regras de Negócio com OCL.....</i>	<i>53</i>
<i>Figura 3.13 Classe Medi@.....</i>	<i>55</i>
<i>Figura 3.14 Mapeamento de uma Tarefa</i>	<i>56</i>
<i>Figura 3.15 Mapeamento de uma sub-Tarefa</i>	<i>57</i>
<i>Figura 3.16 Mapeamento de um Recurso</i>	<i>58</i>
<i>Figura 3.17 Mapeamento de um Objetivo</i>	<i>59</i>
<i>Figura 3.18 Mapeamento de sub-objetivos.....</i>	<i>59</i>
<i>Figura 3.19 Mapeamento de um Objetivo Soft</i>	<i>60</i>
<i>Figura 3.20 Mapeamento de um sub-objetivo Soft</i>	<i>61</i>
<i>Figura 4.1 Diagrama de Arquitetura do OME</i>	<i>70</i>
<i>Figura 4.2 Ator Medi@.....</i>	<i>74</i>
<i>Figura 4.3 Objetivo Comprar Itens de Midia.....</i>	<i>75</i>
<i>Figura 4.4 Tarefa Fazer Pedido.....</i>	<i>76</i>
<i>Figura 4.5 Objetivo Soft Segurança.....</i>	<i>77</i>
<i>Figura 4.6 Recurso Item de Midia.....</i>	<i>78</i>
<i>Figura 4.7 Diagrama de Componentes do Rose.....</i>	<i>84</i>
<i>Figura 4.8 – Modelo do Rose Extensibility Interface</i>	<i>88</i>
<i>Figura 5.1 - Diagrama de Componentes da Integração das ferramentas.....</i>	<i>90</i>
<i>Figura 5.2 Tela de Configuração do Mapeamento.....</i>	<i>91</i>
<i>Figura 5.3 Representação Gráfica do Mapeamento de Ator i*.....</i>	<i>92</i>
<i>Figura 5.4 Representação Gráfica do Mapeamento de uma Tarefa</i>	<i>93</i>
<i>Figura 5.5 Representação Gráfica do Mapeamento de um Recurso</i>	<i>94</i>

<i>Figura 5.6 Representação Gráfica do Mapeamento de um Objetivo</i>	<i>94</i>
<i>Figura 5.7 Representação Gráfica do Mapeamento de um Objetivo</i>	<i>95</i>
<i>Figura 5.8 Representação Gráfica do Mapeamento de uma sub-Tarefa</i>	<i>96</i>
<i>Figura 5.9 Exemplo de um modelo i* para uma sub-Tarefa</i>	<i>96</i>
<i>Figura 5.10 Mapeamento de uma sub-Tarefa quando um Objetivo é mapeado como atributo.....</i>	<i>96</i>
<i>Figura 5.11 Mapeando uma sub-Tarefa quando o Objetivo é um estereótipo de classe.....</i>	<i>97</i>
<i>Figura 5.12 Representação Gráfica do Mapeamento de um sub-Recurso</i>	<i>97</i>
<i>Figura 5.14 Mapeamento de um sub-Recurso quando um Objetivo é mapeado como atributo.....</i>	<i>98</i>
<i>Figura 5.15 Mapeamento de um sub-Recurso quando o Objetivo é um estereótipo de classe.....</i>	<i>98</i>
<i>Figura 5.16 Representação Gráfica do Mapeamento de um sub-Recurso</i>	<i>98</i>
<i>Figura 5.17 Exemplo de um modelo i* para uma sub-Objetivo</i>	<i>99</i>
<i>Figura 5.18 Mapeamento de um sub-Objetivo quando uma tarefa é mapeada como operação.....</i>	<i>99</i>
<i>Figura 5.19 Mapeamento de um sub-Objetivo para Tarefa mapeada como estereótipo de classe.....</i>	<i>99</i>
<i>Figura 5.20 Representação Gráfica do Mapeamento de um sub-Objetivo soft</i>	<i>100</i>
<i>Figura 5.21 Mapeamento de um sub-Objetivo quando uma tarefa é mapeada como operação.....</i>	<i>100</i>
<i>Figura 5.22 Mapeamento de um sub-Objetivo quando uma tarefa é mapeada como operação.....</i>	<i>101</i>
<i>Figura 5.23 Mapeamento de um sub-Objetivo para Tarefa mapeada como estereótipo de classe.....</i>	<i>101</i>
<i>Figura 5.24 Diagrama de Atividades do Componente Executar Mapeamento.....</i>	<i>102</i>
<i>Figura 5.25 – Tela de Localização do Arquivo Telos.....</i>	<i>102</i>
<i>Figura 6.1 Principais Atores e Objetivos.....</i>	<i>107</i>
<i>Figura 6.2 Modelo de Dependência Estratégica da loja Media Shop.....</i>	<i>107</i>
<i>Figura 6.3 Modelo de Razão Estratégica para Media Shop.....</i>	<i>108</i>
<i>Figura 6.4 Modelo de Dependência Estratégica da Media Shop com o ator Medi@.....</i>	<i>109</i>
<i>Figura 6.5 Modelo de Razão Estratégica para o ator Medi@</i>	<i>110</i>
<i>Figura 6.6 Atores da loja de mída mapeados em classes</i>	<i>112</i>
<i>Figura 6.7 Mapeamento das Dependências de Tarefas.....</i>	<i>112</i>
<i>Figura 6.8 Mapeamento das sub Tarefas.....</i>	<i>113</i>
<i>Figura 6.9 Mapeamento dos Recursos.....</i>	<i>113</i>
<i>Figura 6.10 Mapeamento dos Objetivos.....</i>	<i>114</i>
<i>Figura 6.11 Mapeamento dos sub Objetivos.....</i>	<i>114</i>
<i>Figura 6.12 Mapeamento dos Objetivos Soft</i>	<i>115</i>
<i>Figura 6.13 Mapeamento de sub Objetivos Soft.....</i>	<i>115</i>
<i>Figura 6.14 Mapeamento do modelo organizacional da loja Media Shop.....</i>	<i>116</i>

Capítulo 1

Introdução

Este capítulo mostra as principais motivações para realização deste trabalho, os objetivos, a abordagem utilizada e a estrutura da dissertação.

1.1 MOTIVAÇÕES

Os sistemas de software estão cada vez mais presentes no cotidiano das pessoas sendo utilizados nos mais diversos setores, que envolvem desde aplicações triviais até sistemas críticos. Todos os grandes negócios, hoje em dia, fazem algum uso da tecnologia da informação, sendo importante que os seus sistemas sejam projetados para suportá-los.

A velocidade dos avanços tecnológicos, o aumento da complexidade e do tamanho dos sistemas e a ampliação das áreas de atuação da informática geram a necessidade de novos métodos e ferramentas que possam melhorar os atuais padrões de desenvolvimento de software. Em particular, os sistemas de informação requerem técnicas que capturem os objetivos do negócio, os processos de negócio, os relacionamentos entre seus atores e outros elementos do ambiente organizacional, como também requisitos operacionais que devem ser elicitados, analisados, especificados, modelados e gerenciados.

Muito freqüentemente os sistemas de software complexos são entregues atrasados, fora do orçamento previsto e sem atenderem às necessidades reais tanto dos usuários finais como da organização que está pagando pelo sistema. Normalmente estas falhas dos sistemas são conseqüências de problemas com seus requisitos. Um problema comum é que os sistemas de software não suportam as atividades da organização. As principais razões para tais falhas são a falta do entendimento correto da organização pelos desenvolvedores do sistema e também a freqüência das mudanças organizacionais que não podem ser acomodadas pelos sistemas de software existentes (ou seus mantenedores). A modelagem organizacional nos ajuda a entender a estrutura e a dinâmica da organização representando os seus objetivos, processos, recursos, regras de negócio; e a garantir que os clientes, usuários finais e desenvolvedores tenham um entendimento comum da organização. Ela também auxilia na elicitação dos requisitos do sistema, necessários para suportar a organização. Se a especificação de requisitos é baseada num modelo do negócio, há mais garantias que o sistema de informação possa suportar o negócio adequadamente [ERI00].

O propósito da modelagem de negócio é [KRU99]:

- Entender a estrutura e a dinâmica da organização no qual um sistema será implantado.
- Entender os problemas correntes na organização e identificar potenciais de melhoria.
- Garantir que clientes, usuários finais e desenvolvedores tenham um entendimento comum da organização.
- Derivar os requisitos do sistema necessários para suportar a organização.

Um negócio pode ser modelado em três dimensões [PRE02]. A primeira dimensão trata da estrutura organizacional e das funções que são executadas dentro de cada área de negócio definida. A segunda dimensão decompõe as funções do negócio em processos que são responsáveis pela realização da função. A terceira dimensão representa os objetivos e as metas da organização.

A estrutura organizacional pode ser modelada através de gráficos, onde as unidades do negócio são representadas hierarquicamente (ex.: organogramas). No caso dos organogramas, cada caixa representa uma área de negócio da companhia, e pode ser refinada através da hierarquia até unidades que representem pequenos grupos de trabalhos ou até indivíduos. Esse método de documentação é útil para representar como o negócio é construído e organizado.

Os modelos de processos existentes tipicamente descrevem os processos em termos de passos de atividades e fluxos de entidades. Na Ciência da Computação, vários tipos de modelos são usados ou têm sido propostos para modelagem de tipos diferentes de processos, como diagramas de atividades [BOO99], diagramas de fluxo de dados [DEM78], SADT [ROS77] e Redes de Petri [AAL99, DAV93], entre outros.

As técnicas tradicionais de modelagem de sistemas são voltadas apenas para os aspectos funcionais, ou seja, entender o “que” e “como” fazer e não “o porque” [YU94a]. Em particular, a linguagem padrão de modelagem orientada a objeto corrente (Unified

Modeling Language) não é bem equipada para modelagem dos requisitos iniciais (*early requirements*), que são tipicamente informais e normalmente focam nos objetivos organizacionais [YU97a]. Em particular, a UML não pode representar como o sistema atende os objetivos organizacionais, “porque” o sistema é necessário, quais alternativas foram consideradas, qual a implicação das alternativas para as várias partes interessadas (*stakeholders*), e como os seus interesses e preocupações podem ser tratados. UML é mais apropriada para as fases finais da captura de requisitos (*late requirements*), onde ela poderá ser usada para especificar, visualizar, construir e documentar os artefatos de um sistema de software [RUM99].

Portanto, se faz necessário o uso de técnicas mais apropriadas para as fases iniciais dos requisitos, que tratam os aspectos organizacionais, cuja ênfase é dada sobre o entendimento da motivação e das razões que estão por detrás dos requisitos do sistema [YU95a, BOM97, LAM98]. As técnicas orientadas a metas [YU95, DAR93, LAM95, LAM00b, BUB83, KIR94] são mais adequadas para a análise dos requisitos iniciais porque elas focam na definição e exploração das alternativas.

No nosso trabalho adotaremos a técnica do *i** que permite descrever a rede de relacionamentos de dependências entre os atores organizacionais e a representar as razões que estão associadas com cada ator em relação aos relacionamentos de dependência com outros atores.

Nossos trabalhos iniciais visaram à redução da lacuna existente entre a modelagem dos requisitos iniciais e dos requisitos finais [CAS00a, CAS00b, CAS01a, ALE00]. Em particular, fornecemos uma série de diretrizes para o mapeamento de modelos organizacionais em *i** para modelos de classes em UML, que também podem ser anotados com restrições em OCL. Contudo para que essas novas idéias possam ser usadas de uma forma efetiva e sistemática dentro de um processo de desenvolvimento de software é necessário o suporte de ferramentas que apóiem esse processo.

1.2 OBJETIVOS E ABORDAGEM

Nesta dissertação, apresentamos a fundamentação teórica, bem como descrevemos várias diretrizes visando indicar a viabilidade e as vantagens de se integrar no processo de engenharia de requisitos, modelos organizacionais. O principal objetivo desta dissertação é descrever um protótipo de uma ferramenta que suporta o mapeamento de modelos organizacionais em *i** para modelos orientados a objetos de sistemas em UML.

Inicialmente foi feito um estudo da proposta de mapeamento de requisitos organizacionais para UML de Alencar [ALE98a, ALE98b, ALE99a, ALE99b]. Em seguida, foram reavaliadas, as heurísticas utilizadas e feitas algumas modificações para melhorar a automação do processo de mapeamento. Posteriormente, foi feita uma seleção das ferramentas que seriam alvo da integração, baseando-se principalmente nas suas características de integração. Como resultado, o OME (Organization Modeling Environment) [OME02] e o Rational Rose [RAT02] foram escolhidos para servirem como base da integração.

A construção do protótipo foi desenvolvida visando flexibilidade e extensibilidade. A ferramenta foi implementada com o objetivo de permitir ao usuário selecionar quais as diretrizes de mapeamento que serão utilizadas, de acordo com o seu domínio de aplicação. Outro objetivo adotado foi desenvolver o protótipo de forma a facilitar futuras extensões para suportar outras diretrizes de mapeamento e outras técnicas de modelagem organizacional.

1.3 ESTRUTURA DA DISSERTAÇÃO

Além deste capítulo introdutório, esse trabalho consiste de mais seis capítulos, os quais são descritos a seguir.

No Capítulo 2, apresentamos uma visão geral sobre Engenharia de Requisitos. Inicialmente, são tecidas considerações a respeito do que seja requisito (Seção 2.2), a motivação e a importância da Engenharia de Requisitos (Seção 2.3). Em seguida, são

descritos como requisitos podem ser classificados (Seção 2.4) e o processo da Engenharia de Requisitos (Seção 2.5). As considerações finais são tecidas na Seção 2.6.

No Capítulo 3, apresentamos as duas técnicas de modelagem de requisitos utilizadas na proposta: a técnica de modelagem organizacional *i** que é utilizada para modelar os requisitos iniciais (*early requirements*) e o UML que é utilizado para modelar os requisitos finais (*late requirements*). Na Seção 3.1 mostramos a importância da modelagem e apresentamos uma visão geral das técnicas de modelagem organizacional na área de engenharia de requisitos na Seção 3.2. A técnica de modelagem organizacional *i** é discutida na Seção 3.3. Na Seção 3.4 damos uma breve visão de UML onde descrevemos a história do seu desenvolvimento e apresentamos de forma resumida os nove diagramas que fazem parte do padrão. O diagrama de classes e a linguagem de restrições de objeto (OCL) que são utilizadas na proposta para modelagem de requisitos finais são discutidos nas Seções 3.5 e 3.6 respectivamente. Na Seção 3.7 são apresentados alguns exemplos de como aplicar regras de negócio utilizando OCL. Posteriormente na Seção 3.8 apresentamos uma série de diretrizes de mapeamento de requisitos iniciais modelados em *i** para modelos de classes em UML anotados com restrições em OCL. E por fim são tecidas considerações finais na Seção 3.9.

O Capítulo 4 aborda as ferramentas que foram utilizadas como alvo do mapeamento dos requisitos iniciais para requisitos finais. Inicialmente é apresentada uma visão geral (Seção 4.1) da integração de ferramentas, mostrando alguns tipos de classificações, níveis de integração e os benefícios da integração. Na Seção 4.2 é apresentada a ferramenta OME, utilizada para representar os requisitos iniciais em *i** dessa proposta, onde são mostrados suas principais características, arquitetura e o seu mecanismo de representação e armazenamento de dados em Telos. Na Seção 4.3 a ferramenta Rational Rose é descrita, onde são apontados suas características e o seu mecanismo de extensibilidade (o Rational Rose Extensibility). As considerações finais são tecidas na Seção 4.4.

No capítulo 5 é descrito o protótipo da ferramenta GOOD. A Seção 5.1 apresenta uma visão geral da ferramenta e nas Seções 5.2 e 5.3 são mostrados os dois componentes da

ferramenta, o componente Configurar Mapeamento e o componente Executar Mapeamento. Na Seção 5.4 são tecidas as considerações finais.

No Capítulo 6, o estudo de caso de uma loja de venda de itens de mídia é apresentado. Inicialmente são apresentados os objetivos do capítulo (Seção 6.1) e descrito o domínio do problema da loja (Seção 6.2). Baseado na descrição apresentada é desenvolvida a modelagem organizacional em i^* da loja na Seção 6.3. A Seção 6.4 apresenta a aplicação passo a passo das regras de mapeamento mostradas no Capítulo 3 (Seção 3.10) com o uso da ferramenta GOOD. E na Seção 6.5 são tecidas as considerações finais.

O Capítulo 7 apresenta as principais contribuições desse trabalho, os trabalhos relacionados e os trabalhos futuros que podem ser desenvolvidos a partir dessa dissertação.

Capítulo 2

Engenharia de Requisitos

Este capítulo fornece uma visão geral dos conceitos chave da Engenharia de Requisitos, enfatizando a importância da modelagem organizacional para o sucesso de aplicações de sistemas de informação.

2.1 INTRODUÇÃO

A Engenharia de Requisitos tem sido reconhecida como uma fase crítica do processo da Engenharia de Software. Este reconhecimento decorre da descoberta que a maior parte dos problemas, geralmente os mais dispendiosos e de maior impacto negativo no desenvolvimento de software, são originados nas etapas iniciais do desenvolvimento. Estas etapas constituem o processo de engenharia de requisitos, no qual, as principais atividades podem ser definidas como: elicitação, análise, negociação, especificação, gerenciamento e validação de requisitos. Normalmente, falhas na realização destas atividades, resultam em documentos de requisitos inconsistentes, incompletos e conseqüentemente produtos de software de baixa qualidade.

Neste capítulo, apresentamos uma descrição dos elementos mais importantes da Engenharia de Requisitos. Inicialmente, na Seção 2.2 apresentamos uma visão geral da Engenharia de Requisitos, onde são apontadas várias definições para os termos Engenharia de Requisitos e Requisitos. Na Seção 2.3 são mostradas algumas das razões para o estudo da Engenharia de Requisitos. Na Seção 2.4 são discutidos como os requisitos podem ser classificados, na Seção 2.5 descrevemos as principais atividades do processo de Engenharia de Requisitos. Finalmente, na Seção 2.6 são apresentadas as considerações finais do capítulo.

2.2 VISÃO GERAL DA ENGENHARIA DE REQUISITOS

Por se tratar de uma área de pesquisa relativamente recente na literatura, podemos encontrar várias definições da Engenharia de Requisitos. A seguir faremos uma revisão das principais definições.

A Engenharia de Requisitos é a fase do desenvolvimento de sistemas de software responsável pela identificação dos objetivos do sistema pretendido, pela operacionalização de tais objetivos em serviços e restrições e pela atribuição da responsabilidade dos requisitos resultantes para agentes como: humanos, hardware, e software [LAM00a].

Uma outra definição é dada pelo IEEE [IEE84], segundo a qual a Engenharia de Requisitos corresponde ao processo de aquisição, refinamento e verificação das necessidades do cliente para um sistema de software, objetivando-se ter uma especificação completa e correta dos requisitos de software.

Para o projeto STARTS, segundo Hofmann [HOF93], Engenharia de Requisitos compreende o processo pelo qual as intenções e requisitos escritos e falados de seu possuidor são transformados em uma especificação precisa, consistente, não ambígua e completa do comportamento do sistema, incluindo funções, interfaces, desempenho e limitações.

Para Leite[LEI87], a Engenharia de Requisitos pode ser definida como um processo segundo diferentes pontos de vista, no qual "o que" é para fazer é capturado e modelado. Neste processo são usados uma combinação de métodos, ferramentas e atores, sendo produzido, como resultado da modelagem, um documento de requisitos.

Boehm [BOE89] define Engenharia de Requisitos como uma atividade que objetiva desenvolver uma especificação completa, consistente, não ambígua e correta dos requisitos, que sirva, inclusive, de base para um acordo entre as partes envolvidas no processo de desenvolvimento do software, onde se pactue, de forma concisa, "o que" o produto irá fazer. Nesse sentido, a Engenharia de Requisitos seria um processo que requer um envolvimento muito grande entre cliente e desenvolvedores, evitando decisões de projetos durante a definição dos requisitos, ficando, assim, definidas, pelo menos, duas fases bem distintas: o que se tenta alcançar e como projetar o sistema para satisfazer os requisitos. Segundo Hofmann, essa definição parece separar a Engenharia de Requisitos de outras preocupações do desenvolvimento de software [HOF93].

Já segundo Davis [DAV93], a Engenharia de Requisitos corresponde à atividade de entendimento das necessidades do usuário no contexto do problema a ser resolvido, bem como das limitações impostas na solução.

De acordo com Loucopoulos [LOU95], a Engenharia de Requisitos corresponde ao processo sistemático de desenvolvimento de requisitos, através de um processo iterativo, cooperativo de análise do problema, documentação das observações

resultantes, em uma variedade de formatos de representações e checagem da acurácia da compreensão ganha.

Segundo Kotonya e Sommerville [KOT97], Engenharia de Requisitos trata-se de um termo relativamente novo, para cobrir todas as atividades envolvidas na descoberta, documentação e manutenção de um conjunto de requisitos, num sistema baseado em computador. Onde o termo Engenharia implica que técnicas sistemáticas e repetitivas são usadas para assegurar que os requisitos do sistema sejam completos, consistentes e relevantes.

Para Zave [ZAV97], a Engenharia de Requisitos está relacionada com a identificação de metas para serem atingidas pelo sistema a ser desenvolvido, assim como a operacionalização de tais metas em serviços e restrições.

Entre as várias possíveis definições da Engenharia de Requisitos, neste trabalho escolhemos a definição de Zave, pois ela destaca a importância da identificação das metas a serem atingidas.

Outra definição necessária é a de requisitos. Várias definições têm sido usadas para requisitos de software. Elas representam perspectivas diferentes e graus variados de detalhes e precisão. A IEEE [IEE97] define requisito como sendo:

- 1) Uma condição ou uma capacidade de que o usuário necessita, para solucionar um problema ou alcançar um objetivo.
- 2) Uma condição ou uma capacidade que deve ser alcançada ou possuída por um sistema ou componente do sistema, para satisfazer um contrato, um padrão, uma especificação ou outros documentos impostos formalmente.
- 3) Uma representação documentada de uma condição ou capacidade, conforme os itens (1) e (2).

A definição da IEEE cobre a visão do usuário sobre requisitos (comportamento externo do sistema), a visão do desenvolvedor (2) e o conceito fundamental que requisitos devem ser documentados (3).

Uma outra definição sugere que um requisito é uma necessidade do usuário ou uma característica, função ou atributo necessário do sistema que pode ser percebido de uma posição externa daquele sistema [DAV93]. Essa definição enfatiza *o que* o sistema deve conter.

Segundo Kotonya e Sommerville [KOT97], um requisito pode descrever:

- Uma facilidade no nível do usuário; por exemplo, um corretor de gramática e ortografia.
- Uma propriedade muito geral do sistema; por exemplo, o sigilo de informações não autorizadas.
- Uma restrição específica no sistema; por exemplo, o tempo de varredura de um sensor.
- Uma restrição no desenvolvimento do sistema; por exemplo: a linguagem que deverá ser utilizada para o desenvolvimento do sistema.

Uma definição bem simples para requisitos é dada por Macaulay [MAC96], segundo ele requisito é simplesmente algo que o cliente necessita. Já segundo Jackson [JAC95], requisitos são fenômenos ou propriedades do domínio da aplicação que devem ser executados, normalmente são expressos em linguagem natural, diagrama informal ou noutra notação apropriada ao entendimento do cliente e da equipe de desenvolvimento.

2.3 MOTIVAÇÃO PARA O ESTUDO DA ENGENHARIA DE REQUISITOS

Várias são as razões para se estudar e aplicar a Engenharia de Requisitos, que iremos apresentar a seguir. Por exemplo, Frederick Brooks [BRO87] declarou que a parte mais difícil da construção de um sistema de software é decidir precisamente o que construir. Nenhuma outra parte do trabalho conceitual é tão difícil como estabelecer os requisitos técnicos detalhados, incluindo todas as interfaces com as pessoas, máquinas e outros sistemas de software. Nenhuma outra parte do trabalho afeta mais o sistema resultante se feita de forma errada. Nenhuma outra parte é mais difícil de ser retificada depois.

Segundo Kotonya [KOT97] erros nos requisitos são responsáveis:

- Pelo atraso na entrega do sistema e pelo aumento do custo do desenvolvimento do sistema.
- Pela insatisfação do cliente e dos usuários finais com o sistema. Eles podem não usar suas facilidades ou podem ainda descartar por completo o sistema.
- Pela falta de confiabilidade no uso do sistema, devido a erros regulares do sistema e falhas que levam à interrupção da operação normal do sistema.
- Pelo aumento do custo de manutenção e evolução do sistema.

Estudos recentes têm confirmado o problema de requisitos. Uma pesquisa entre 8000 projetos desenvolvidos por 350 companhias dos EUA revelou que um terço dos projetos nunca foram completados, e metade só obtiveram sucesso parcialmente, resultando em sistemas com funcionalidades parciais, orçamentos estourados, e atrasos significantes [STA95]. Quando perguntado sobre a causa de tais falhas, os gerentes executivos identificaram requisitos pobres como a fonte principal de problemas (aproximadamente metade das respostas) - mais especificamente, a falta de envolvimento dos usuários (13%), incompletude dos requisitos (12%), mudança dos requisitos (11%), expectativas não realísticas (6%), e objetivos obscuros (5%). No lado europeu, uma pesquisa entre mais de 3800 organizações em 17 países concluiu de forma semelhante que a maioria dos problemas de software percebidos é na área de especificação de requisitos (>50%) e no gerenciamento de requisitos (50%) [ESI96]. O nível de aceitação dos sistemas de informação comerciais é da ordem de 40%, enquanto que, para sistemas de tempo real, este índice sobe para cerca de 75%. Este fato, em sua maioria, está correlacionado a erros e inadequações na produção do produto final, com falhas ocorridas durante as fases de análise e especificação de requisitos.

Segundo Leite [LEI00], os benefícios da Engenharia de Requisitos são:

- Concordância entre desenvolvedores, clientes e usuário sobre o trabalho a ser feito e quais os critérios de aceitação do sistema.

- Uma base precisa para a estimativa dos recursos (custo, pessoal, prazos, ferramentas e equipamentos).
- Melhoria na usabilidade, manutenibilidade e outras qualidades do sistema.

Do ponto de vista econômico, o custo da correção de erros de requisitos aumenta drasticamente à medida que eles são descobertos no decorrer das fases do desenvolvimento de software, como ilustrado na Figura 2.1. Segundo Pressman [PRE01], o custo de correção de erros na fase de projeto é cerca de três a seis vezes mais alto do que na fase de definição de requisitos. O custo de correção de erros na fase de operação é relativamente cerca de 40 a 1000 vezes mais caro do que o custo da correção de erros na fase de definição dos requisitos.

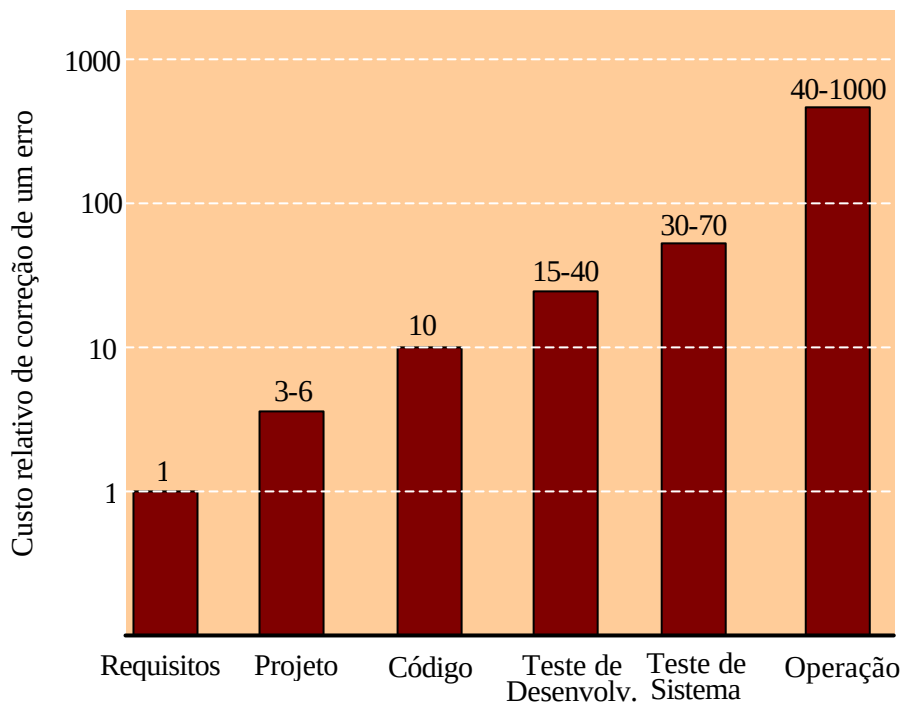


Figura 2.1 Custo Relativo para Correção de um Erro

2.4 CLASSIFICAÇÃO DE REQUISITOS

Os ambientes empresariais atuais são caracterizados por sistemas interligados de propriedades as mais diversas. Assim, os softwares e/ou os sistemas, para essas organizações, precisam ser construídos considerando-se o ambiente no qual eles estarão inseridos, assim como os aspectos relacionados com a estrutura e a política empresarial.

Essa tendência, em se observar aspectos relativos ao ambiente organizacional, já vem se manifestando desde o final dos anos 80 [DOB89, DOB92, DOB94a, DOB94b]. Assim, durante o processo de especificação dos requisitos, surge a necessidade de estabelecer o tipo de requisito de que se está tratando, a fim de melhorar a compreensão das necessidades do cliente, bem como modelar melhor esta necessidade.

De forma geral, podemos categorizar os requisitos, em três classes básicas distintas, mas que podem estar relacionadas: funcionais, não-funcionais e organizacionais [LOU95, KOT97, DAV93]. Alguns autores preferem classificar os requisitos somente segundo os dois primeiros tipos, funcionais e não-funcionais.

Os *requisitos funcionais* dizem respeito à definição das funções que um sistema ou um componente de sistema deve fazer. Eles descrevem as transformações a serem realizadas nas entradas de um sistema ou em um de seus componentes, a fim de que se produzam saídas.

Os *requisitos não-funcionais* dizem respeito a restrições, aspectos de desempenho, interfaces com o usuário, confiabilidade, segurança, manutenibilidade, portabilidade, padrões, e outras propriedades que o sistema deve possuir, bem como aspectos sociais e políticos. Alguns desses requisitos são provavelmente traduzidos em funções (operacionalizados), ao longo do processo de desenvolvimento de software [CHU00].

De forma geral, a diferença entre requisitos funcionais e não-funcionais está no fato dos primeiros descreverem *o que* o sistema deve fazer, enquanto que os outros fixam restrições sobre *como* os requisitos funcionais serão implementados.

Os *requisitos organizacionais* dizem respeito às metas da empresa, suas políticas estratégicas adotadas, os relacionamentos entre os seus atores junto com seus respectivos objetivos.

Encontram-se ainda sugestões de que requisitos possam ser vistos como sendo "o que" o sistema deve fazer, associado com "o porque" deva fazer, em lugar do "como" o sistema deve fazer. Procura-se ampliar a visão tradicional dos requisitos que trata apenas de aspectos funcionais e não-funcionais, com informações organizacionais que abordam a

intencionalidade dos fatos, ou seja, os requisitos organizacionais [YU94a, LOU95, MYL95, YU95b, YU97b].

Segundo essa nova visão, a Engenharia de Requisitos pode ser dita como: *A área do conhecimento preocupada na comunicação com agentes organizacionais, com respeito a suas visões, intenções e atividades relativas às suas necessidades de suporte de computadores, desenvolvimento e manutenção de uma especificação de requisitos adequada a um sistema* [LOU95]. Isso sugere que a Engenharia de Requisitos também inclua problemas e aspectos gerenciais, organizacionais, econômicos, técnicos, sociais e ambientais. Nesse contexto, Bubenko [BUB95] salienta que a própria definição da Engenharia de Requisitos evoluiu ao longo dos anos.

A visão tradicional [LOU95], em geral, esquece aspectos importantes, que influenciam diretamente no sistema que se deseja especificar, tais como:

- Os objetivos do próprio sistema e o relacionamento desses com os objetivos da organização.
- Outras propriedades dos sistemas, como as relacionadas com o desempenho, segurança e restrições.

Esta visão mais ampla sobre os requisitos, serve para caracterizar que deve ser levado em consideração, quando da especificação dos requisitos, o fato de que o sistema que está sendo especificado é apenas um dentre os vários sistemas no ambiente da organização.

Os requisitos organizacionais desempenham papel fundamental no projeto e no desenvolvimento de um sistema. Para Dobson [DOB94b], requisitos organizacionais são aqueles que resultam do sistema pretendido, considerando-se o contexto social no qual ele se encontra inserido. Por exemplo, obrigações e responsabilidades, controle e autonomia, valores e éticas que são normalmente, embutidos na estrutura e na política da empresa, de forma que não são diretamente observados ou facilmente articulados.

A razão principal para se considerar os requisitos organizacionais, está na ajuda à compreensão de interações complexas entre as empresas (organizações) e as pessoas envolvidas. Assim, esta ajuda torna-se essencial, por oferecer meios:

- Para a descrição dos processos da organização.
- Para a definição e estruturação da informação de suporte a estas aplicações.

Embora as muitas classificações propostas tenham muitos pontos em comum, elas diferem de acordo com o ponto de vista dos autores, visto que uma dada classificação pode enfatizar mais um aspecto do que outro. Todavia, a tendência atual aponta para a necessidade de tratar, especificamente, dos requisitos organizacionais[DAR93, MYL92, MYL95, WIE98].

Até pouco tempo, os projetistas e desenvolvedores ignoravam a importância dos aspectos organizacionais para o desenvolvimento de seus produtos. Só mais recentemente é que esses aspectos começaram a serem observados e foram considerados essenciais ao sucesso das implementações. Hoje, existem modelos, que representam componentes intencionais, tais como, razões, motivações e os porquês.

2.4.1 Requisitos Iniciais vs Requisitos Finais

Um recente trabalho na área da Engenharia de Requisitos tem também feito uma importante distinção entre a captura de requisitos iniciais (*early requirements*) e a captura de requisitos finais (*late requirements*) [YU97a]. Há uma necessidade de distinguir entre as fases iniciais de engenharia de requisitos, quando o engenheiro de requisitos está tentando entender um cenário organizacional, das fases finais quando o engenheiro de requisitos formula uma solução. As atividades da fase da captura de requisitos iniciais são tipicamente informais e tratam de questões organizacionais e não funcionais. A ênfase durante esta fase está no entendimento das motivações e das razões que estão por detrás dos requisitos do sistema. As atividades da captura de requisitos finais, por outro lado, focam na completude, consistência e verificação dos requisitos.

Os requisitos iniciais equivalem à definição do espaço de pesquisa (“escopo”) e uma pesquisa entre as alternativas dentro daquele espaço. Enquanto os requisitos finais equivalem a refinar, eliminar ambigüidades e completar a descrição da alternativa escolhida. A análise estruturada e a análise orientada a objeto são boas para tratarem os requisitos finais. Contudo, a análise orientada a metas [YU95, DAR93, LAM95, LAM00b, BUB83, KIR94] é mais apropriada para análise dos requisitos iniciais porque ela foca na definição e exploração do espaço de alternativas (ver Seção 3.3).

2.5 O PROCESSO DE ENGENHARIA DE REQUISITOS

Normalmente, a Engenharia de Requisitos é considerada a atividade inicial do processo de desenvolvimento de software. Contudo os requisitos mudam durante o desenvolvimento e evoluem depois que o sistema já está em operação [LAM00a]. O processo de engenharia de requisitos pode ser considerado um conjunto estruturado de atividades que são seguidas com o objetivo de derivar, validar e manter um documento de requisitos [KOT97].

Existem várias propostas para modelos de processo de engenharia de requisitos. Porém, não existe um processo considerado ideal. Este trabalho adota o modelo proposto por Kotonya e Sommerville [KOT97], que descreve as fases de Elicitação, Análise, Documentação e Validação de Requisitos, como sendo as fases do processo de requisitos. A Figura 2.2 mostra essas atividades representadas num modelo espiral, no qual cada atividade do processo é repetida até que seja tomada a decisão de que o documento de requisitos pode ser aceito. Além disso, as mudanças de requisitos são parte do processo de Gerenciamento de Requisitos. Apesar dessas atividades serem normalmente descritas independentemente e numa ordem particular, na prática, elas consistem de processos iterativos e inter-relacionados que podem cobrir todo o ciclo de vida do desenvolvimento de sistemas de software [NUS00].

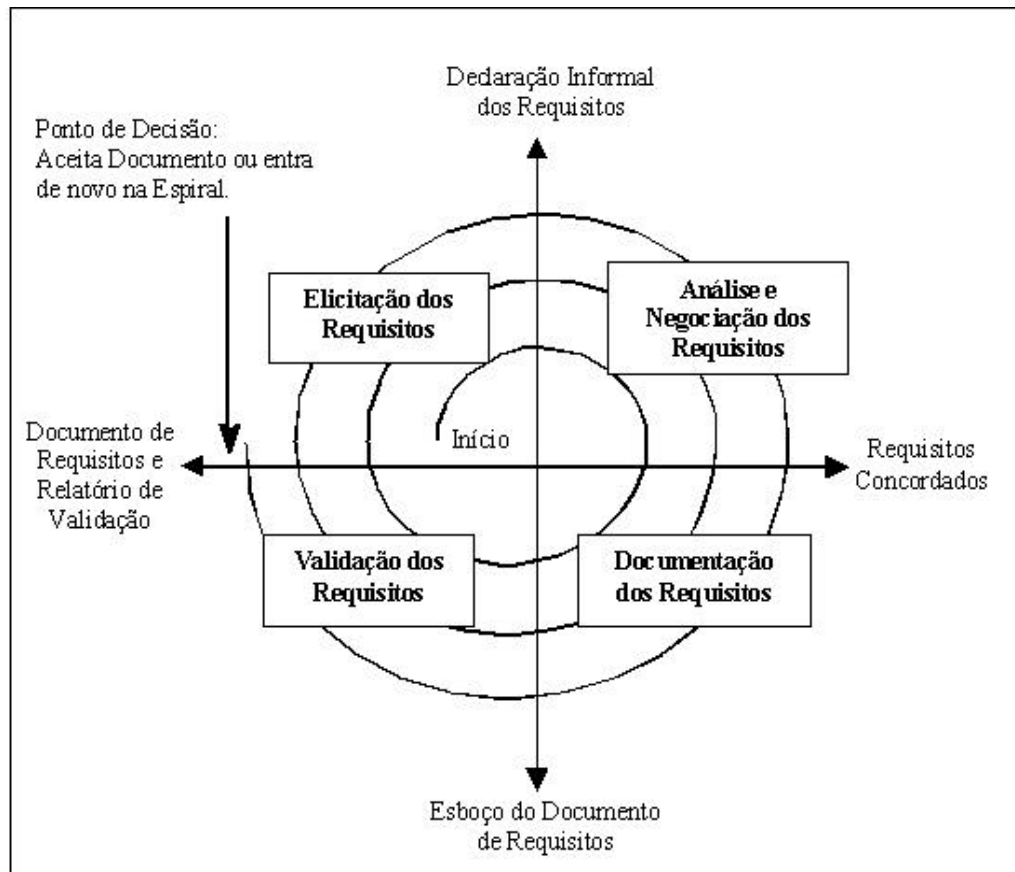


Figura 2.2 - Atividades do Processo da Engenharia de Requisitos no Modelo Espiral

2.5.1 Elicitação de Requisitos

Esta é a primeira atividade no ciclo de vida da Engenharia de Requisitos, que tem como propósito geral obter conhecimento relevante para o problema a ser resolvido. A elicitação de requisitos é um processo de descoberta dos requisitos para um sistema, através da comunicação com clientes, usuários finais e com os demais interessados no

desenvolvimento do sistema. Além de descobrir quais são as necessidades dos usuários, essa atividade também requer uma cuidadosa análise da organização, do domínio de aplicação e dos processos organizacionais [KOT97]. Assim, cabe à elicitação a tarefa de identificar os fatos que compõem os requisitos do sistema, de forma a prover o mais correto entendimento do que é esperado do sistema de software. Outro papel do processo de elicitação é identificar as necessidades das diferentes classes de usuários [SHA99], por exemplo, usuários experientes, ocasionais e iniciantes.

Um dos principais problemas encontrados durante essa fase é a dificuldade de entender as reais necessidades dos usuários [CAS95, NUS00]. Em boa parte dos casos, este fato é decorrente da formação distinta entre analistas e usuários, que têm pontos de vista diferentes sobre o mesmo problema. Outro fator que dificulta o processo de elicitação é que muitas vezes os usuários não têm uma idéia precisa e explícita do sistema a ser desenvolvido, ou mesmo possuem dificuldades em descrever seu conhecimento sobre o domínio do problema. Uma técnica bastante utilizada é o uso de cenários para representar as tarefas que os usuários executam normalmente e aquelas que eles desejam executar [LEI97, SCH98]. A escolha da técnica apropriada para elicitar requisitos depende do tempo e dos recursos disponíveis, assim como do tipo de informação necessária. Algumas das classes de técnicas de elicitação são [NUS00]:

- Técnicas tradicionais – incluem o uso de questionários, entrevistas, análise de documentação existente [WEI96, MAM99].
- Técnicas de elicitação de grupo – são técnicas de dinâmica de grupo com o objetivo de entender de forma mais detalhada as necessidades dos usuários, estão incluídas: brainstorming, sessões JAD e RAD [DAM97].
- Prototipação – é utilizada para elicitar requisitos quando há um alto grau de incerteza ou quando é necessário um rápido feedback dos usuários [DAV92].
- Técnicas de modelagem – fornecem um modelo específico das informações que serão adquiridas, e usa esse modelo para orientar o processo de elicitação. Inclui métodos baseados em metas, como o: KAOS [LAM98], *i** [YU95a] e métodos baseados em cenários como CREWS [MAI98].

- Técnicas cognitivas – incluem uma série de técnicas originalmente desenvolvidas para aquisição de conhecimento para sistemas baseados em conhecimento, alguns exemplos são: análise de protocolo, *laddering*, *card sorting*, *repertory grids* [SHA96].
- Técnicas contextuais – surgiram como uma alternativa para as técnicas tradicionais e cognitivas, inclui técnicas de etnografia e análise social [VIL99].

O processo de eliciação é uma atividade complexa e quase sempre requer uma quantidade considerável de tempo e recursos. Entretanto, ela é fundamental para garantir que o sistema final atenderá às expectativas e necessidades dos usuários.

2.5.2 Análise e Negociação de Requisitos

As atividades de análise e negociação de requisitos são na maioria das vezes executadas de forma paralela ou intercalada. Durante o processo de análise, os requisitos são analisados, a fim de se detectar incompletudes, omissões e redundâncias. Na análise, a preocupação está em descobrir os requisitos que realmente são necessários e que o usuário deseja. A seguir são apresentadas técnicas que podem ser utilizadas no processo de análise.

- ***Lista de Checagem da Análise:*** é uma lista de questões, as quais o analista pode usar para avaliar cada requisito. Cada item serve de guia na avaliação do requisito. No final da checagem, pode obter-se uma lista de problemas associados com os requisitos. Estes problemas podem ser solucionados através de negociação ou se necessário com nova eliciação.
- ***Matrizes de Interação:*** é utilizada para descobrir as interações entre requisitos, apontando possíveis conflitos entre requisitos. A atividade de negociação pode corrigir estes conflitos.
- ***Prototipação:*** os protótipos desenvolvidos na etapa de eliciação de requisitos podem ser aperfeiçoados na etapa de análise, possibilitando uma análise mais

completa dos requisitos. Protótipos facilitam o envolvimento dos diferentes *stakeholders* na elicitação, análise e negociação de requisitos.

Durante o processo de análise de requisitos são indentificados os problemas com os requisitos, tais como: requisitos conflitantes, ambíguos e duplicados. A partir dos requisitos com problemas, os usuários devem discutir, priorizar e negociar esses requisitos até que se obtenha um acordo com possíveis modificações e simplificações. O processo de negociação de requisitos tenta resolver conflitos entre usuários sem necessariamente comprometer a satisfação dos objetivos de cada usuário. Em geral, os modelos de negociação identificam as principais necessidades de cada usuário, onde são estabelecidas prioridades aos requisitos baseados nas necessidades dos usuários. Em seguida, os requisitos são analisados com a finalidade de se tentar garantir que os requisitos mais críticos sejam atendidos. Existem várias técnicas para priorização e negociação de requisitos. Por exemplo, em Karlsson [KAR98], ele descreve um estudo de caso realizado num projeto Ericsson em que duas importantes técnicas para priorização e negociação de requisitos foram comparadas, a técnica AHP [SAA90] e a técnica QFD [HAU88].

Em princípio, a negociação de requisitos deveria ser um processo objetivo. As decisões a serem tomadas em relação aos requisitos do sistema deveriam ser baseadas nas necessidades técnicas e organizacionais [KOT97]. Entretanto, a realidade é normalmente diferente. Negociações raramente são conduzidas usando somente argumentos lógicos e técnicos, elas também são fortemente influenciadas pelas necessidades pessoais dos indivíduos.

2.5.3 Documentação de Requisitos

Depois que os requisitos do sistema são concordados, eles devem ser documentados com um nível apropriado de detalhes. Em geral, é necessário que o documento de requisitos seja entendível por todos os envolvidos no processo da engenharia de requisitos, pois ele servirá como um contrato entre usuários e desenvolvedores.

Segundo Ryan [RYA93], o documento de requisitos é o meio através do qual é possível descrever as restrições quanto às características do produto e quanto ao processo de desenvolvimento, a interface com outras aplicações, a descrição sobre o domínio e as informações de suporte ao conhecimento do problema.

A maioria dos requisitos pode ser documentada através de linguagem natural, na forma de sentenças [LEF99]. Contudo algumas vezes a ambiguidade da linguagem natural causa erros no comportamento do sistema. Isto leva a uso de métodos de especificação mais precisos, como:

- Pseudocódigo,
- Diagramas de máquina de estado,
- Árvores de decisão,
- Diagramas de atividades,
- Modelo de entidade de relacionamento,
- Análise orientada a objetos,
- Análise estruturada,
- Técnicas de modelagem organizacional, como *i**.

No capítulo 3, nós descreveremos as técnicas de modelagem organizacional e a análise orientada a objeto.

2.5.4 Validação de Requisitos

Segundo Loucopoulus [LOU95], o processo de validação de requisitos é definido como a atividade na qual certifica-se que o documento de requisitos é consistente com as necessidades dos usuários. Descrever os requisitos de forma explícita é uma condição necessária não somente para validar os requisitos, mas também para resolver conflitos entre usuários. Em geral, a validação de requisitos é considerada uma atividade difícil

devido à dificuldade de se obter um consenso entre diferentes usuários com objetivos conflitantes [LAM98].

Outro grande desafio durante a validação de requisitos é demonstrar que a especificação dos requisitos do sistema está correta. Enquanto o projeto e a implementação possuem o documento de requisitos para verificar seus resultados, a validação de requisitos não possui nenhuma outra representação que pode ser usada como base. Ou seja, não existe uma forma para demonstrar formalmente que a especificação está correta [KOT97]. Contudo existem várias técnicas que podem ser aplicadas para suportar a validação de requisitos:

- **Revisões dos requisitos:** esta é uma das técnicas mais populares para validar requisitos. Consiste na revisão dos requisitos por um grupo de pessoas que analisam, discutem e apontam caminhos para solucionar os problemas encontrados. A partir dessas descobertas, pode-se adotar medidas que solucionem os conflitos. Problemas típicos encontrados nas revisões incluem: incompletude das descrições dos requisitos, ambigüidade e a não obediência a padrões da organização.
- **Prototipação:** protótipos são desenvolvidos com o intuito de permitir uma melhor representação dos requisitos de um sistema. Enquanto tradicionalmente, o usuário tem de esperar até as etapas finais do processo de desenvolvimento, para visualizar uma versão executável do sistema, com o desenvolvimento de protótipos, os usuários podem ter uma idéia antecipada de como o sistema executável funcionará. É observado que a técnica de prototipação geralmente é usada para ajudar nas atividades de elicitação e análise de requisitos. Contudo, ela também pode ser usada para validar os requisitos. Após o documento de requisitos já ter sido definido e acordado, pode-se aperfeiçoar o protótipo desenvolvido na elicitação e análise e utilizá-lo para validar os requisitos, com um auxílio mais efetivo dos usuários/clientes.
- **Testes de requisitos:** comumente, testes de software são realizados no final do processo de desenvolvimento de software. No entanto, podem ser desenvolvidos

casos de teste para os requisitos, já na fase de validação de requisitos. Pode-se adotar, por exemplo, técnicas baseadas em cenários para elicitare e analisar requisitos e criar casos de teste para os cenários desenvolvidos. Casos de teste podem ser aplicados de forma simulada nos cenários desenvolvidos. Se surgirem dificuldades para criar os casos de teste bem como para executá-los através de simulação, há uma grande probabilidade de existirem problemas com os requisitos. É menos oneroso e traumático descobrir estes problemas na atividade de validação do processo de engenharia de requisitos do que em etapas finais do processo de desenvolvimento de software.

- **Validação de modelos:** geralmente, quando documentamos requisitos, os mesmos são descritos através de linguagem natural e diagramas. Tipicamente, cria-se um documento em linguagem natural descrevendo os requisitos do sistema. Para detalhar melhor o funcionamento do sistema são desenvolvidos modelos do sistema. O desenvolvimento destes modelos está associado à abordagem de desenvolvimento de software adotada. Estes modelos podem ser modelos de fluxo de dados, modelos de dados semânticos ou modelos inerentes ao paradigma de desenvolvimento, por exemplo, orientação a objetos ou métodos formais. Estes modelos precisam ser validados para demonstrar que os mesmos são consistentes e representam os reais requisitos dos usuários.

2.5.5 Gerenciamento de Requisitos

Esta atividade é uma das mais importantes do processo de engenharia de requisitos. Ela tem como objetivo principal o controle e gerenciamento de mudanças nos requisitos [TOR99].

Um dos maiores problemas atuais no desenvolvimento de software reside no fato de que os requisitos de software são modificados, seja por questões de melhoria das funções do sistema, por solicitações do usuário ou devido a correções de erros encontrados. Nestes casos, mudanças podem gerar efeitos colaterais e se propagar negativamente para outras partes do software. O controle dos requisitos descobertos e analisados no processo de engenharia de requisitos é fundamental para que as mudanças sejam adequadamente

tratadas e seus impactos corretamente avaliados. Isto implica necessariamente em poder rastrear os requisitos ao longo de todos os artefatos produzidos no processo de engenharia de software. Este rastreamento de requisitos deve ocorrer em ambos os sentidos, dos requisitos iniciais para os artefatos desenvolvidos e dos artefatos para as fontes que originaram os requisitos.

Gerenciar requisitos significa poder escrever e acompanhar um requisito ao longo de todo o processo de desenvolvimento, enquanto existir. Isto garante documentos de requisitos mais confiáveis e gerenciáveis, aspectos importantes para a obtenção de produtos de software de qualidade.

De forma resumida podemos descrever os objetivos do gerenciamento de requisitos como:

- Gerenciar mudanças para os requisitos acordados;
- Gerenciar o relacionamento entre requisitos;
- Gerenciar as dependências entre o documento de requisitos e os demais documentos produzidos no processo de engenharia de requisitos.

No processo de gerenciamento de requisitos, os requisitos devem ser unicamente identificados. Esta identificação é o meio adotado para poder rastrear e avaliar os impactos advindos de mudanças. Para grandes sistemas, nos quais o número de requisitos a serem gerenciados é muito grande, é necessário que os mesmos sejam armazenados em uma base de dados e sejam registradas as ligações entre os requisitos relacionados. Atualmente, verifica-se que na grande maioria dos sistemas, faz-se necessário o uso de ferramentas CASE que apoiem o processo de gerenciamento de requisitos. Grandes quantidades de requisitos não são gerenciáveis sem a utilização de alguma ferramenta computacional de apoio [WIE99]. Algumas ferramentas existentes atualmente no mercado usadas com este fim são RequisitePro [RAT02], DOORS [TEL02] e Caliber-RM [TEC02].

2.6 CONSIDERAÇÕES FINAIS

Neste capítulo, foram vistos os principais conceitos da Engenharia de Requisitos que é área da Engenharia de Software no qual nosso trabalho visa dar sua contribuição. Nosso trabalho propõe a integração de duas técnicas de modelagem de requisitos: a modelagem organizacional e a modelagem orientada a objeto.

No próximo capítulo, apresentaremos a técnica de modelagem organizacional *i** que foi utilizada na nossa proposta para representar os requisitos iniciais (*early requirements*) e a linguagem de modelagem unificada (UML) para representar os requisitos finais (*late requirements*).

Capítulo 3

Modelagem de Requisitos

Este capítulo fornece uma visão geral das técnicas de modelagem organizacional e das metodologias orientadas a objeto. Em particular, são apresentadas a técnica i e a linguagem de modelagem unificada (UML) enfocando as partes relevantes ao nosso trabalho. Em seguida são mostradas as diretrizes para o mapeamento dos requisitos iniciais (early requirements) em i* para requisitos finais (late requirements) em UML anotados com restrições em OCL.*

3.1 INTRODUÇÃO

Modelos são abstrações que representam a essência de um problema ou estrutura complexa filtrando os detalhes não essenciais, de modo a tornar o problema mais fácil de entender. Abstração é uma capacidade humana fundamental que nos permite tratar com a complexidade. Engenheiros, físicos, artistas têm construído modelos por vários anos para testar seus projetos antes de executá-los. O desenvolvimento de sistemas de software não deve ser uma exceção. Para construir sistemas complexos, o desenvolvedor deve abstrair diferentes visões do sistema, construir modelos usando uma notação precisa, verificar que os modelos satisfazem os requisitos do sistema e gradualmente adicionar detalhes para transformar os modelos em implementação [QUA99].

A modelagem visual é útil para entender o problema, para a comunicação com todos os envolvidos com o projeto (clientes, analistas, projetistas, etc.), modelar a organização, preparar a documentação, e projetar programas e banco de dados. A modelagem promove um melhor entendimento dos requisitos, e sistemas mais fáceis de serem mantidos.

No nosso caso estaremos interessados em modelar os requisitos iniciais e os requisitos detalhados ou finais. Para modelar os requisitos iniciais usaremos o *i**, enquanto para modelagem dos requisitos finais usaremos um subconjunto da UML e OCL.

Neste capítulo apresentaremos várias técnicas de modelagem organizacional na área da Engenharia de Requisitos (Seção 3.2). Em particular detalharemos a técnica *i** na Seção 3.3. Na seção 3.4 descrevemos resumidamente o padrão UML e listamos os nove diagramas que fazem parte do padrão. O diagrama de classes e a linguagem de restrições de objeto (OCL) que são utilizadas na proposta para modelagem de requisitos finais são discutidos nas Seções 3.5 e 3.6 respectivamente. Na Seção 3.7 são apresentados alguns exemplos de como aplicar regras de negócio utilizando OCL. Posteriormente na Seção 3.8 apresentamos uma série de diretrizes de mapeamento de

requisitos iniciais modelados em *i** para modelos de classes em UML anotados com restrições em OCL. As considerações finais são tecidas na Seção 3.9.

3.2 MODELAGEM ORGANIZACIONAL

Várias técnicas têm sido propostas para modelagem organizacional na área da Engenharia de Requisitos. Algumas das principais técnicas são apresentadas a seguir:

- A técnica ORDIT (Organisational Requirements Definition of Information Technology Systems) [ORD93, DOB94b, EAS97], ajuda os participantes das organizações a definirem alternativas técnicas e o futuro organizacional, fornecendo um processo sistemático e aproveitável, que é capaz de suportar gerações de requisitos organizacionais e fornecer métodos e ferramentas associadas que suportem o processo;
- A técnica de Bubenko, apresentada originariamente em [BUB83], destaca áreas de conhecimento da organização. Ela é constituída por cinco submodelos [KIR94] que são elaborados a partir dos objetivos (*Modelo de Objetivos* - MO), dos atores (*Modelo de Atores* - MA), das atividades e do uso (*Modelo de Atividades e Uso* - MAU), dos conceitos (*Modelo de Conceitos* - MC) e dos requisitos (*Modelo de Requisitos do Sistema de Informação* - MRSI);
- A técnica KAOS (Knowledge Acquisition in autOmated Specification) [DAR93, LAM95, LAM00b] é uma técnica para aquisição de requisitos orientados a objetivos, que fornece uma linguagem formal, rica para a captura de requisitos funcionais, não-funcionais e organizacionais, capaz de fornecer facilidades para a descrição de objetivos, agentes, alternativas, eventos, ações, modalidade de existência e responsabilidades de agentes. Trata-se de uma ferramenta para aquisição/elaboração de requisitos direcionada a objetivos, onde é forte o conceito de agentes. Ela explora vários construtores utilizados pelas notações de Análise Orientada a Objetos, cujo projeto final consiste na atribuição de responsabilidades a agentes.

- A técnica de Eriksson-Penker Business Extensions [ERI00] apresenta um conjunto de extensões baseado nos modelos existente do UML. Através do uso de estereótipos, esta técnica representa elementos organizacionais como: objetivos, recursos, processos e regras. O *Eriksson-Penker Business Extensions* sugere um conjunto de diagramas e modelos para modelagem do negócio: o diagrama da visão, o modelo conceitual, o modelo de objetivo, o diagrama de processo, o diagrama de linha de montagem, o diagrama de caso de uso, o modelo de recurso, o modelo da organização, o diagrama de estado, diagramas de interação e diagramas da topologia do sistema.
- O Rational Unified Process [KRU99, JAC99] suporta a modelagem do negócio através de dois modelos que são extensões à UML: o modelo de caso de uso de negócio (*business use-case model*) e o modelo de objetos de negócio (*business-object model*). O modelo de caso de uso de negócio consiste de atores que representam papéis externos ao negócio (ex.: clientes) e os casos de uso de negócio (*business use cases*) que são processos. Um modelo de objeto de negócio é um modelo interior de um negócio que descreve como cada caso de uso de negócio é realizado por um conjunto de trabalhadores (*workers*) que estão usando um conjunto de entidades de negócio (*business entities*). Um trabalhador representa um papel ou um conjunto de papéis num negócio e uma entidade de negócio representa alguma coisa que o trabalhador acessa, inspeciona, manipula, produz ou usa num caso de negócio.
- *i** é uma técnica de modelagem que consiste de dois componentes de modelagem, o Modelo de Dependência Estratégica (SD) e o Modelo de Razão Estratégica (SR). O Modelo de Dependência Estratégica descreve uma rede de relacionamento entre atores. Os atores dependem um do outro para terem seus objetivos alcançados, tarefas executadas e recursos fornecidos. O Modelo de Razão Estratégica permite modelar as razões que estão associadas com cada ator em relação aos relacionamentos de dependência com outros atores. Esses modelos têm sido formalizados usando conceitos intencionais de Inteligência Artificial, tais como objetivo, crença, habilidade e compromisso [COH90].

Nessa dissertação nós iremos utilizar a técnica i^* que será apresentada em detalhe na próxima seção.

3.3 A TÉCNICA i^*

A técnica do i^* ¹ foi proposta por Eric Yu em sua tese de doutorado [YU95a] com o objetivo de fornecer uma técnica de modelagem conceitual mais rica para descrição de processos que envolvem vários participantes. A maioria das outras técnicas de modelagem se concentra nos aspectos comportamentais do processo, deixando de lado as razões ou motivações que estão associadas aos comportamentos. Em geral, elas permitem a descrição dos componentes do sistema (atores) em termos de seu estado, capacidades (processos que podem executar) e comportamento (como e quando os processos são executados), mas não conseguem expressar as razões envolvidas nos processos (o porque). Enquanto, o i^* fornece conceitos para modelar e responder questões como:

- Por que atores executam os processos?
- Quem deseja que eles façam isso?
- Quais são as formas alternativas de executar um processo?
- Por que os atores possuem ou recebem informação?

Esse tipo de entendimento mais profundo sobre os “porquês” ou razões constitui uma parte importante do conhecimento sobre o domínio, contudo não podem ser expressos na maioria dos modelos convencionais que apenas permitem a descrição de entidades e atividades de uma organização e os seus relacionamentos.

Na técnica i^* , as organizações consistem de atores sociais que têm a liberdade de ação, mas dependem um dos outros para terem seus objetivos alcançados, tarefas executadas e recursos fornecidos. A técnica se baseia na premissa que um entendimento maior do

¹ o acrônimo i^* refere-se a “distributed intentionality”

processo pode ser obtido através de uma visão intencional e estratégica. Nessa visão, a unidade central a ser modelada é o ator intencional e estratégico. Um *ator intencional* não simplesmente executa atividades e produz ou consome entidades, mas tem motivações, intenções e razões associadas a suas ações. Os aspectos intencionais de um ator podem ser caracterizados por propriedades como objetivos, crenças, habilidades e compromissos. Um *ator é estratégico* quando não foca apenas seu objetivo imediato, mas quando se preocupa com as implicações de seu relacionamento estrutural com outros atores, por exemplo, as oportunidades e vulnerabilidades que estão presentes numa dada configuração de relacionamentos.

A técnica do *i** tem sido usada no contexto da engenharia de requisitos de sistemas de informação [YU93], reengenharia de processos de negócio [YU94b], modelagem de processos de software [YU94a] e desenvolvimento de sistemas orientados a agentes [CAS00c, CAS01b, CAS01c, MYL00].

A técnica do *i** é composta de dois tipos de modelos: o Modelo de Dependência Estratégica (SD) e o Modelo de Razão Estratégica (SR). O Modelo de Dependência Estratégica fornece uma descrição intencional de um processo em termos de uma rede de relacionamentos de dependência entre atores. O Modelo de Razão Estratégica (SR) fornece uma descrição intencional do processo em termos de elementos do processo e as razões que estão por detrás deles. Os modelos do *i** são formalmente representados através da linguagem de modelagem conceitual *Telos* que será descrita no Capítulo 4.

3.3.1 Modelo de Dependência Estratégica

O Modelo de Dependência Estratégica (SD) consiste de um conjunto de nós e ligações. Cada nó representa um ator e cada ligação entre dois atores indica que um ator depende de um outro ator para algo. O ator dependente é chamado de *dependor*, e o ator sobre quem está a dependência é chamado de *dependee*. O objeto sobre o qual está centrado o relacionamento de dependência é chamado de *dependum*. Por exemplo, na Figura 3.1, temos que o Cliente (*dependor*) depende da loja Media Shop (*dependee*) para alcançar seu objetivo de comprar itens de mídia (*dependum*). Através da dependência, o *dependor* é capaz de alcançar objetivos que ele não conseguiria sozinho ou pelo menos

não facilmente sem o *dependee*. Dessa forma, o *depender* torna-se vulnerável, se o *dependee* falhar em entregar o *dependum*, ou seja, ele poderá ser afetado nas suas habilidades de alcançar seus objetivos.



Figura 3.1 Dependência de Objetivo

Um relacionamento de dependência também pode ser caracterizado pelo grau da dependência. Do lado do *depender*, uma dependência mais forte significa que o *depender* é mais vulnerável, e é provável que ele tome medidas mais fortes para mitigar a vulnerabilidade. Do lado do *dependee*, uma dependência mais forte implica que o *dependee* fará um esforço maior em tentar entregar o *dependum*. O modelo fornece três graus de dependências: Aberta (não comprometido), Comprometido e Crítico.

Uma dependência do tipo Aberta significa que a falha em obter o *dependum* pode afetar os objetivos do *depender*, mas as consequências não serão sérias. A dependência Aberta é representada graficamente pelo símbolo “O”. Na Figura 3.2 temos que o Cliente não será seriamente afetado se seu cadastro não for aprovado imediatamente pelo ator Bank Cty.



Figura 3.2 Dependência Aberta

Numa dependência do tipo Comprometida, os objetivos do *depender* serão significativamente afetados se o *dependum* não for alcançado. Do lado do *dependee*, uma dependência Comprometida significa que o *dependee* tentará fazer o seu melhor possível para entregar o *dependum*. A dependência Comprometida não possui símbolo gráfico, ou seja, ela é o grau de dependência padrão. Ela é exemplificada pela Figura

3.1, onde o Cliente terá os seus objetivos significativamente afetados se a loja Media Shop não conseguir atender o seu objetivo de comprar itens de mídia.

Numa dependência do tipo Crítica, os objetivos do *dependor* serão seriamente afetados se o *dependum* não for alcançado. Ela é representada graficamente por “x”. Por exemplo, na Figura 3.3 o Cliente sofrerá sérias conseqüências se o ator Bank Cty não atender o seu objetivo de segurança, por exemplo, ter o seu cartão clonado.

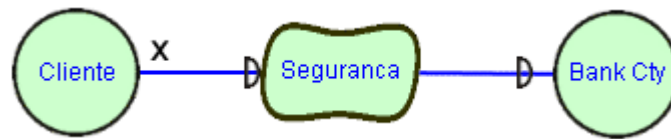


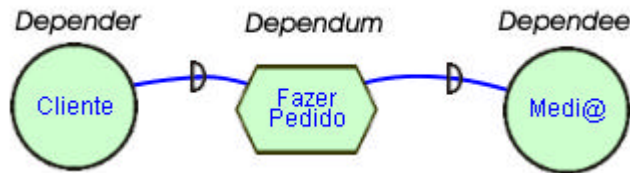
Figura 3.3 Dependência Crítica

A técnica do *i** distingue quatro tipos de dependências:

- Dependência de Objetivo.
- Dependência de Tarefa.
- Dependência de Recurso.
- Dependência de Objetivo Soft.

Na dependência de objetivo, o *dependor* depende do *dependee* para alcançar uma certa meta. Onde as metas representam afirmações sobre estado do mundo que o ator gostaria de alcançar. Elas são condições claramente definidas nos quais a sua satisfação pode ser avaliada objetivamente para verdadeiro ou falso. Como a meta será alcançada não é especificado. Na Figura 3.1, o Cliente tem a meta de comprar Item de Mídia, para tal ele está dependente do ator Media Shop. O cliente está apenas interessado no resultado, ou seja, que seu objetivo seja atingido. Por exemplo, a Media Shop pode atender a meta do Cliente vendendo os itens de mídia através da Internet, por telefone, ou na própria loja. O *dependee* tem a liberdade de escolher como fazê-lo. Com a dependência de objetivo, o *dependor* ganha a habilidade para assumir que a condição ou estado acontecerá, mas fica vulnerável desde que o *dependee* pode falhar em satisfazer aquela condição.

Na dependência de tarefa, o *depender* depende do *dependee* para executar uma atividade. Uma dependência de tarefa especifica como a tarefa é executada, mas não o porque. O *depender* fica vulnerável desde que *dependee* pode falhar em executar a



tarefa. Na Figura 3.4, o Cliente depende do ator Medi@ que representa um sistema de informação, para realizar algum pedido, de acordo com um procedimento predefinido (possivelmente uma das operações disponíveis no sistema computacional).

Figura 3.4 Dependência de Tarefa

Numa dependência de recurso, o *depender* depende do *dependee* para a disponibilidade de uma entidade (física ou informacional). Um exemplo desse tipo de dependência é mostrado na Figura 3.5, em que a loja Media Shop tem a necessidade que o Fornecedor de Mídia entregue mídias (por exemplo, cds, discos, fitas, etc).

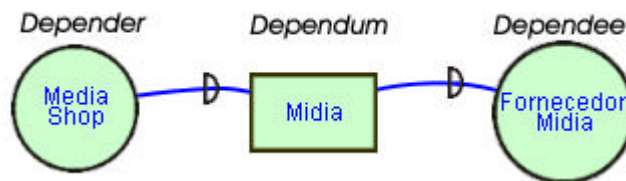


Figura 3.5 Dependência de Recurso

A dependência de objetivo soft é similar à dependência de objeto descrita anteriormente, exceto que é utilizada quando os objetivos não são claramente definidos. Para certos objetivos é difícil decidir de forma clara se eles foram alcançados ou não, porque seu significado não é ainda bem definido ou sua satisfação é definida subjetivamente. Não há concordância a priori entre o *depender* e o *dependee* sobre a realização do objetivo, pois o mesmo envolve aspectos subjetivos que vão sendo esclarecidos, através de métodos que são escolhidos durante a realização do objetivo. A Figura 3.6 ilustra a dependência de objetivo soft entre a loja Media Shop que deseja manter os clientes satisfeitos.

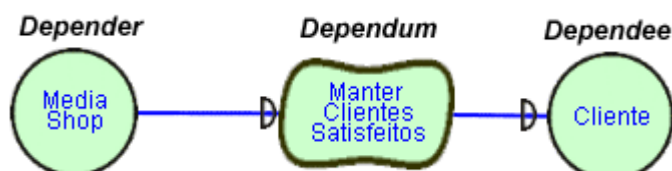


Figura 3.6 Dependência de Objetivo Soft

O modelo também permite o refinamento da noção genérica de ator em papel, agente, ou posição, onde:

- **Papel:** é uma caracterização abstrata do comportamento de um ator social dentro de algum contexto especializado. Suas características são facilmente transferíveis para outros atores. Dependências estão associadas com um papel quando essas dependências se aplicam indiferentemente de quem desempenha o papel. Por exemplo, na Figura 3.7 um gerente poderia desempenhar vários papéis, como o de gerenciar a contabilidade e supervisionar os vendedores.
- **Agente:** é um ator que possui manifestações físicas concretas, tal como uma pessoa, apesar do termo agente poder ser também usado para agentes artificiais (hardware e software). Um agente tem dependências que se aplicam independentemente do papel que ele está desempenhando. Estas características tipicamente não são facilmente transferíveis para outros indivíduos. Por exemplo, na Figura 3.7 um funcionário da loja poderia ser representado como um agente e o funcionário Paulo como uma instância desse agente. Como exemplo de agentes artificiais, temos o sistema de informação Medi@.
- **Posição:** ao nível de abstração é algo intermediário entre um papel e um agente. Ele é um conjunto de papéis que são tipicamente desempenhados por um agente. Ou seja, um agente ocupa uma posição que cobre vários papéis. Por exemplo, na Figura 3.7 o funcionário Paulo é um agente que ocupa a posição de gerente que por sua vez cobre os papéis de gerenciar a contabilidade e de supervisionar os vendedores.

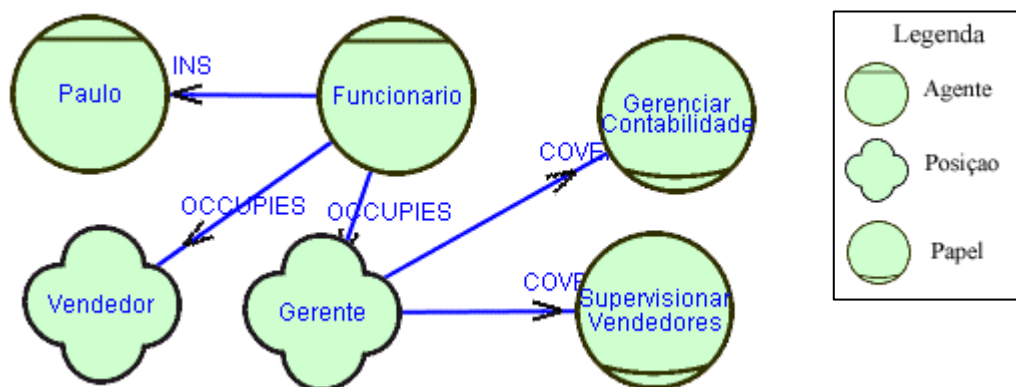


Figura 3.7 Agentes, Posições e Papéis

Um modelo simplificado do Diagrama de Dependência Estratégica (SD) pode ser representado (através de UML) pela figura 3.8, onde temos que um ator pode ser refinado num Agente que pode desempenhar vários papéis e ocupar uma Posição que cobre vários Papéis. Um Ator pode ser relacionado com outro Ator através de um relacionamento de dependência sobre um Dependum que pode ser do tipo Tarefa, Recurso, Objetivo ou Objetivo Soft.

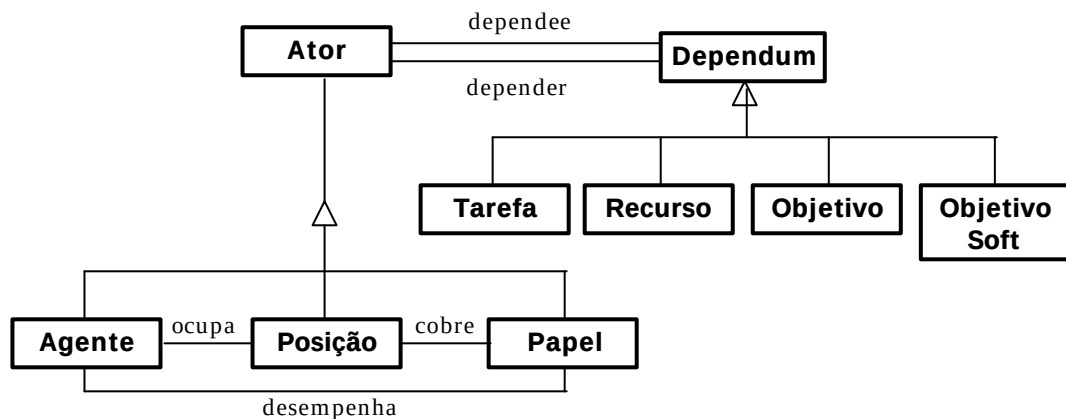


Figura 3.8 Meta modelo do SD

3.3.2 Modelo de Razão Estratégica

No modelo de Razão Estratégica (SR), a razão ou motivação que cada ator tem sobre suas metas e sobre seus relacionamentos com outros atores são descritas.

Observe que o modelo anterior de Dependência Estratégica (SD) pode ser visto como uma abstração do SR. Isto não implica que durante a modelagem do processo, o SR seja usado primeiro e o SD derivado depois. O SD provavelmente será usado primeiro para evitar modelos detalhados nas fases iniciais e detalhes serão adicionados progressivamente para obter o SR.

O modelo SR dá uma descrição das razões que estão associadas a um sistema através da modelagem de seus atores, das metas que eles perseguem e de como eles pretendem alcançar estas metas.

Enquanto o modelo de Dependência Estratégica (SD) modela apenas os relacionamentos externos entre atores, o modelo de Razão Estratégica (SR) permite um

refinamento das descrições dos processos. O modelo de Razão Estratégica (SR) é usado para:

- Descrever os interesses, preocupações e motivações dos participantes (atores) de um processo.
- Possibilitar a avaliação das possíveis alternativas de definição do processo.
- Investigar, mais detalhadamente, as razões existentes por detrás das dependências entre os vários atores.

O modelo SR detalha as tarefas executadas pelos atores e mostra como essas tarefas são relacionadas aos objetivos funcionais e não-funcionais. Ele também mostra maneiras alternativas de atender os objetivos que o ator pode considerar na realização do objetivo.

O Modelo de Razão Estratégica (SR), também consiste de um conjunto de nós e ligações de forma similar ao Modelo de Dependência Estratégica (SD). Há os mesmos quatro tipos de nós definidos no modelo SD que são objetivo, tarefa, recurso e objetivo soft. Contudo, são introduzidos dois novos tipos de ligações, a decomposição de tarefa e a ligação de meio fim.

O modelo SR descreve os relacionamentos intencionais que são internos aos atores, através de relacionamentos meio-fim que relacionam elementos do processo, fornecendo representações explícitas do “porquê”, do “como” e das alternativas.

Uma ligação meio-fim indica (ver Figura 3.9) um relacionamento entre um fim, que pode ser um objetivo a ser alcançado, ou uma tarefa a ser realizada, ou um recurso a ser produzido, ou um objetivo soft a ser satisfeito e um meio para atendê-lo. O meio é normalmente expresso na forma de uma tarefa, já que a semântica de uma tarefa incorpora como fazer alguma coisa. Na notação gráfica a seta aponta do meio para o fim.

Numa ligação onde o fim é um objetivo e o meio é uma tarefa, essa tarefa especifica o “como” alcançar o objetivo (o fim). Da mesma maneira, quando um questionamento do “porquê” é realizado, explora-se descobrir o fim para se explicar a razão da realização

do meio. A Figura 3.9 ilustra o uso de uma ligação meio-fim, onde o objetivo da loja *Media Shop* de *Tratar Pedidos dos Clientes* pode ser alcançado através da tarefa *Pedidos Por Telefone* ou pela tarefa *Pedidos Pela Internet* ou pela tarefa *Pedidos Na Loja*. Numa ligação onde o fim é um recurso e o meio é uma tarefa, a tarefa é usada para garantir a disponibilidade do recurso.

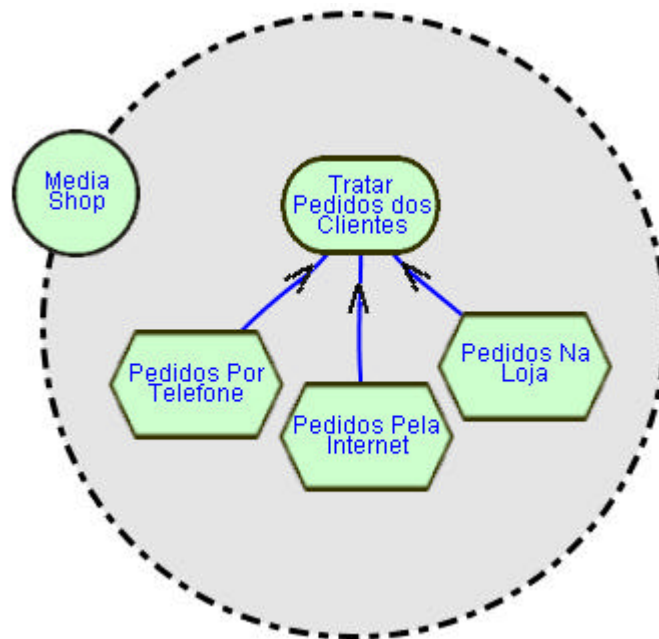


Figura 3.9 Exemplo de um Relacionamento Meio Fim

Uma ligação de decomposição de tarefa é utilizada para modelar uma tarefa em termos de sua decomposição em sub-componentes. Esses sub-componentes podem ser objetivos, tarefas, recursos e/ou objetivos soft. Portanto há quatro tipos de classes de ligações de decomposição de tarefas, que são descritos a seguir:

- **Objetivo-Tarefa:** se uma meta for um componente de uma tarefa, isto significa que a condição representada pela meta é um passo necessário no curso da ação representada pela tarefa. Como esta condição será alcançada é deixado em aberto (ex: será determinado depois ou várias alternativas podem ser consideradas).
- **Tarefa-Tarefa:** se uma tarefa for uma sub tarefa de uma outra, isto significa que a tarefa mais *alta* é decomposta através execução da sub tarefa.

- **Recurso-Tarefa:** se um recurso é um componente de uma tarefa, isso significa que ele é requerido pela tarefa.
- **Objetivo Soft-Tarefa:** se um objetivo soft é um componente de uma tarefa, ele serve como um objetivo de qualidade para aquela tarefa.

A Figura 3.10 exibe um exemplo de uma decomposição da tarefa *Gerenciar Loja da Internet*. A tarefa é inicialmente refinada nos objetivos *Tratar Pesquisa de Item*, *Tratar Pedidos da Internet*, no objetivo soft *Atrair Novos Clientes* e na Tarefa *Produzir Estatísticas*. Ao contrário do relacionamento de meio-fim, onde cada meio representa uma alternativa, na decomposição de tarefa todos os refinamentos terão que ser atendidos ou realizados para a tarefa que está sendo decomposta ter sua realização com sucesso.

A técnica *i** foca nos aspectos intencionais entre os relacionamentos dos atores organizacionais e somente parte destes conceitos são utilizados na proposta. Apenas os principais elementos (atores, recursos, tarefas, objetivos e objetivos soft) e relacionamentos (ligação de dependência, decomposição de tarefa, e ligação de meio-fim) foram tratados.

Nem todos os conceitos da técnica foram apresentados nesta dissertação, entre eles a questão de influência e raciocínio [YU95a].

A técnica *i** descrita nessa seção foi utilizada na nossa proposta para representar os requisitos iniciais, enquanto para os requisitos finais, nós usamos um subconjunto do UML que pode ser anotado com restrições em OCL. Nas seções seguintes apresentaremos o UML e posteriormente OCL.

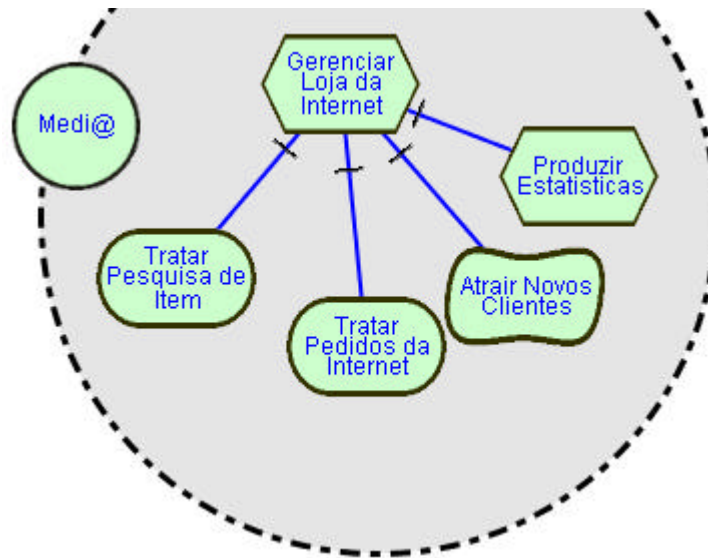


Figura 3.10 Exemplo de uma Decomposição de Tarefa

3.4 INTRODUÇÃO À UML

A abordagem orientada a objeto foi inicialmente proposta no final dos anos 60. Contudo, ela levou quase 20 anos para se tornar largamente adotada [PRE01]. A primeira linguagem orientada a objeto desenvolvida foi Simula-67 que influenciou várias outras linguagens orientadas a objetos posteriores. Contudo, o movimento orientado a objeto, apenas se tornou difundido com o surgimento do Smalltalk no início dos anos 80, seguido por outras linguagens orientadas a objeto como Objective C, C++ e Eiffel. Os primeiros métodos orientados a objeto foram publicados por Shlaer/Mellor [SHL88] e Coad/Yourdon [COA90], seguidos pelos livros de Booch [BOO93], Rumbaugh [RUM91], Wirfs-Brock [WIR90] e Jacobson [JAC92].

Como não havia uma única linguagem de modelagem orientada a objeto padrão, os usuários tinham que escolher entre várias propostas existentes, muitas similares com diferenças secundárias em relação ao poder de expressão global. A maioria das linguagens compartilhava um conjunto de conceitos geralmente aceitos que eram expressos ligeiramente de forma diferente em várias linguagens. Essa falta de concordância desencorajava novos usuários a ingressarem na tecnologia orientada a objeto.

Houve algum esforço inicial na tentativa de unificar conceitos entre os métodos. Um exemplo foi o desenvolvimento do método Fusion por Coleman e seus colegas [COL93], que incluía conceitos de OMT [RUM91], Booch [BOO93] e CRC [WIR90]. Contudo, como este método não envolveu os autores originais, ele deve ser considerado como outro método novo em vez de uma substituição de vários métodos existentes. A primeira tentativa de unificação aconteceu quando Rumbaugh juntou-se a Booch na Rational Software Corporation em 1994. Eles começaram a combinar os conceitos dos métodos do OMT e do Booch, resultando numa primeira proposta do Método Unificado (*Unified Method v. 0.8*) em 1995. Um ano depois Jacobson se associou a eles iniciando o trabalho que resultou no lançamento da versão 0.9 da Linguagem de Modelagem Unificada (UML).

Ao mesmo tempo em que UML estava se tornando um padrão de fato da indústria de software, um grupo internacional de especialistas em modelagem assumiu a responsabilidade de tornar a linguagem um padrão também. O grupo “UML Partners”, representando um consórcio de fabricantes e integradores de sistema, começou a trabalhar com os “Três Amigos” (Booch, Rumbaugh e Jacobson) para propor UML como a linguagem de modelagem padrão para a OMG (Object Management Group). Tal colaboração resultou na UML 1.0. Entre Janeiro e Julho de 1997, o grupo original de parceiros se expandiu e produziu a versão 1.1 de UML que foi adotada pela OMG em 14 de Novembro de 1997.

A manutenção de UML foi então assumida pela RTF (Revision Task Force) da OMG. E até o momento, o padrão já sofreu a atualização da versão 1.2 em Junho de 1998 e no mesmo ano a versão 1.3. foi aprovada. Recentemente, em Setembro de 2001, a OMG liberou a versão 1.4 da especificação de UML.

UML permite desenvolver diversos tipos de diagramas gráficos que representam vários aspectos do sistema. UML tem nove diagramas pré-definidos:

- **Diagrama de classe.** Descreve a estrutura de um sistema. As estruturas são construídas de classes, interfaces, colaborações e seus relacionamentos. Esses diagramas são os mais comuns achados na modelagem de sistemas orientados a objeto.

- **Diagrama de objeto.** Expressa possíveis combinações de objetos de um diagrama de classe específico. Ele é tipicamente usado para exemplificar um diagrama de classe.
- **Diagrama de caso de uso.** Exibe um conjunto de caso de uso, atores e seus relacionamentos. Cada caso de uso é tipicamente definido num texto simples, descrevendo uma parte da funcionalidade total do sistema.
- **Diagrama de estado.** Mostra uma máquina de estado, consistindo de estados, transições, eventos e atividades. Diagramas de estado representam a visão dinâmica de uma classe ou do sistema.
- **Diagrama de atividade.** É um tipo especial de diagrama de estado que mostra o fluxo de atividades dentro de um sistema. Eles são especialmente importantes na modelagem de funções de um sistema e enfatizam o fluxo de controle entre objetos.
- **Diagrama de seqüência.** Mostra uma interação, consistindo de um conjunto de objetos e seus relacionamentos, incluindo mensagens que podem ser trocadas entre eles. O diagrama de seqüência enfatiza a ordenação temporal das mensagens.
- **Diagrama de colaboração.** Mostra uma interação, consistindo de um conjunto de objetos e seus relacionamentos, incluindo mensagens que podem ser trocadas entre eles. O diagrama de colaboração enfatiza a organização estrutural dos objetos que enviam e recebem mensagens. Os diagramas de seqüência e de colaboração são isomórficos.
- **Diagrama de componente.** É um tipo de diagrama utilizado para modelar os aspectos físicos de um sistema orientado a objetos. Um diagrama de componentes mostra a organização e dependências entre um conjunto de componentes. Está relacionado aos diagramas de classe, pois tipicamente os componentes são mapeados para uma ou mais classes, interfaces ou colaborações.
- **Diagrama de implantação.** Mostra a configuração dos nós de processamento em tempo de execução e os componentes neles existentes. Ele pode representar a topologia de hardware sobre o qual sistema executa.

Está relacionado aos diagramas de componentes, pois tipicamente um nó inclui um ou mais componentes.

No nosso trabalho faremos uso de UML durante a especificação dos requisitos finais (*late requirements*), sobretudo dos diagramas de classe. Na próxima seção faremos uma revisão desta notação.

3.5 DIAGRAMA DE CLASSES

Os diagramas de classes mostram a estrutura estática do modelo do sistema. As estruturas são constituídas de classificadores e relacionamentos. Um diagrama de classe é um grafo de classificadores e os seus relacionamentos estáticos. Há vários tipos de classificadores, tais como classes, interfaces e tipo de dados. Para o propósito de implementação existem outros classificadores como subsistemas, componentes, artefatos e nós. A Figura 3.11 mostra os vários tipos de classificadores definidos no meta-modelo de classificadores da especificação versão 1.4 de UML.

Uma classe representa um conceito dentro do sistema que está sendo modelado. As classes possuem estrutura de dados (atributos), comportamento (operação) e relacionamentos com outros elementos. Uma classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica. Um atributo é constituído de alguns elementos, como:

- Nome: o nome do atributo que deve ser único dentro da classe e de sua hierarquia.
- Valor Inicial: uma expressão que especifica o valor daquele atributo que um objeto possui logo após a sua criação. O valor inicial é opcional.
- Escopo: um atributo pode ser de classe ou de instância. Quando o escopo é de classe existe apenas um valor do atributo para todas as instâncias da classe. Quando o escopo é de instância, cada instância da classe (objeto) mantém seu próprio valor do atributo.

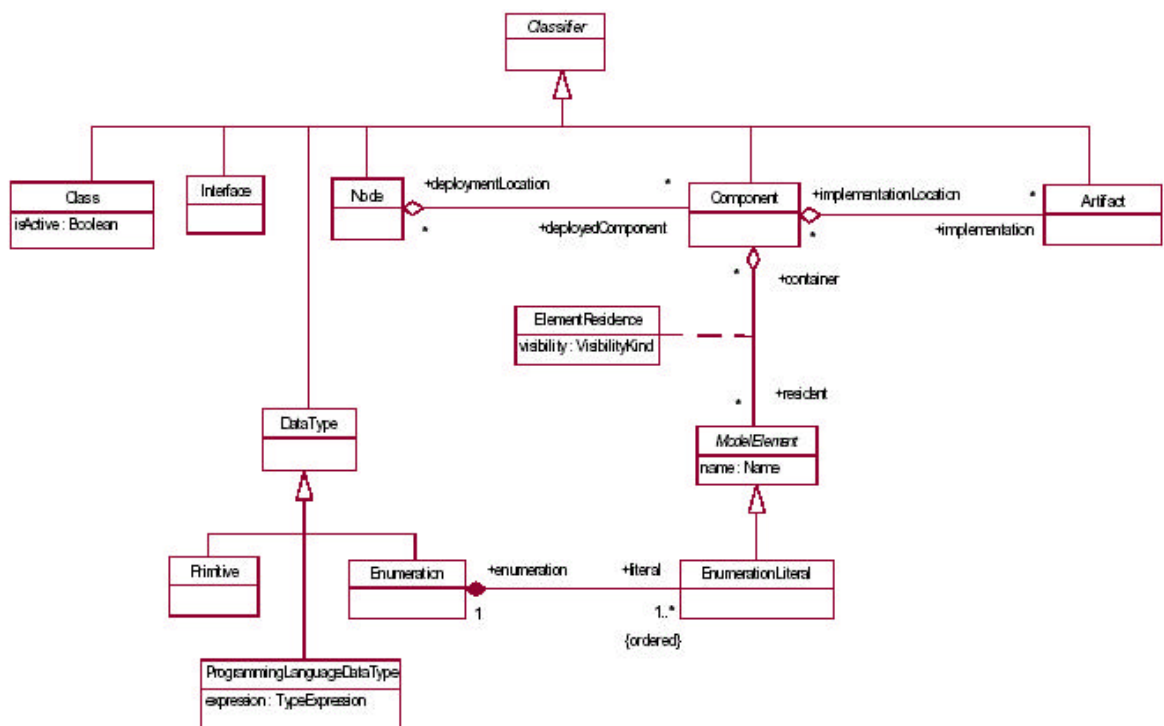


Figura 3.11 Meta-modelo dos classificadores de UML

Uma operação é um serviço que uma instância da classe pode ser solicitado para executar e ela possui um nome e uma lista de parâmetros. Uma operação é constituída de alguns elementos, como:

- Nome: o nome da operação que junto com a lista de tipos de parâmetros deve ser única.

- Tipo de Retorno: uma lista de tipos de valores retornados pela operação (classes, tipos de dados ou interfaces). Este elemento é opcional.
- Visibilidade: indica a visibilidade da operação. Ele pode assumir os valores *public*, *private*, *protected*, *package*. Outros valores podem ser adicionados para modelar certas linguagens de programação.
- Escopo: informa se a operação se aplica a objetos individuais ou a classe.

Uma interface é um especificador para operações externamente visíveis de uma classe, componente, ou outro classificador (incluindo subsistemas) sem a especificação de sua estrutura interna.

Os tipos de relacionamentos suportados por UML incluem associação, generalização, fluxos e dependências. Para diagramas de classes apenas associação, generalização e dependência são aplicáveis.

A seguir descreveremos em detalhe o relacionamento do tipo associação que será usado nos capítulos posteriores desta dissertação.

Uma associação é um relacionamento entre dois ou mais classificadores que (incluindo a possibilidade de uma associação de um classificador para ele mesmo) descreve a conexão entre suas instâncias. Cada instância de uma associação (uma ligação) é uma tupla (lista não ordenada) de referência para objetos. A estrutura de uma associação é composta de:

- Nome: uma associação tem um nome opcional que deve ser único entre associações e classes dentro de um mesmo pacote.
- Extremidade de associação (*association ends*): é a parte estrutural de uma associação que define a participação de uma classe na associação.

A extremidade da associação tem várias propriedades, tais como:

- Multiplicade: que indica o número possível de objetos que podem ser relacionados a um objeto.

- Ordenação: se a multiplicidade é maior do que um, então o conjunto de elementos relacionados pode ser ordenado ou não ordenado.
- Qualificador: uma lista de atributos usados como seletores para encontrar objetos relacionados por uma associação.
- Navegabilidade: uma seta que indica se a navegação é suportada em relação ao classificador ligado à extremidade da associação.
- Indicador de agregação: se anexada indica uma agregação (diamante vazio) ou composição (diamante preenchido).

Durante a definição dos requisitos finais (*late requirements*), muitas vezes se faz necessário descrever restrições aos modelos. No nosso trabalho estas restrições serão expressas em OCL, descrito na próxima seção.

3.6 OCL – OBJECT CONSTRAINT LANGUAGE

Em 1996, na mesma época que UML foi submetida para a força tarefa OA&D Domain da OMG em resposta a um pedido para propostas sobre Análise e Projeto Orientado a Objeto, a IBM e a ObjectTime Limited juntamente submeteram uma proposta que incluía a linguagem OCL (Object Constraint Language).

Após a escolha da proposta de UML pela OMG, a IBM e a ObjectTime se uniram aos parceiros do desenvolvimento do UML. A principal contribuição deles para UML 1.1 foi a OCL. A OCL foi desenvolvida por Jos Warmer como uma linguagem para modelagem de negócio dentro da IBM, derivada do método Syntropy de Steve Cook e John Daniels [COO94]. O Syntropy estendeu a OMT com linguagem textual baseada em Z [WOO96] para adicionar invariantes para diagramas de classe e anotar transições sobre diagramas de estado como pré-condições e pós-condições. OCL adotou uma sintaxe simples e não simbólica e restringiu-se a um pequeno conjunto dos conceitos centrais do Syntropy.

OCL é usada dentro de UML tanto para formalizar as semânticas da própria linguagem, como também para fornecer a facilidade para usuários de UML expressarem restrições precisas sobre a estrutura dos modelos.

Um diagrama de UML, como por exemplo, um diagrama de classe, tipicamente não é refinado o suficiente para fornecer todos os aspectos relevantes de uma especificação. Pode ser necessário descrever restrições adicionais sobre os objetos num modelo. Tais restrições são freqüentemente descritas em linguagem natural. A prática tem mostrado que isso sempre resulta em ambigüidades. Para escrever restrições sem ambigüidades, linguagens formais têm sido desenvolvidas. A desvantagem das linguagens formais tradicionais é que elas são úteis para pessoas com um grande conhecimento matemático, mas são difíceis de usar para a maioria dos analistas de negócio e de sistema. OCL foi desenvolvida para preencher esta lacuna.

OCL é uma linguagem declarativa, ou seja, as restrições não têm efeitos colaterais. Dessa forma o estado do sistema (ex.: o valor de um atributo de um objeto) não muda devido à avaliação de uma expressão. OCL permite especificar quais mudanças de estado irão ocorrer como resultado de uma operação específica através, por exemplo, de uma pós-condição. As vantagens dessa abordagem são:

- O modelador não precisa decidir como manipular a quebra de uma restrição. Primeiro, o modelador decide quais restrições devem ser válidas. Depois, o modelador ou alguém decide como resolver os problemas que ocorrem quando restrições são quebradas. Essa abordagem resulta numa clara separação entre especificação e implementação.
- As restrições devem ser estáveis dentro de seus domínios. Ou seja, elas não mudam muito ao longo do tempo. As ações que precisam ser tomadas quando as restrições não são atendidas mudam mais freqüentemente ou podem ser diferentes de uma aplicação para outra dentro de um mesmo domínio.
- A avaliação de uma expressão em OCL é instantânea. Isso significa que o estado dos objetos de um modelo não muda durante a avaliação.

OCL é uma linguagem tipada, no qual todos os operadores são de um tipo específico e cada operador pode ser aplicado sobre tipos de operandos específicos. Por exemplo, o operador + (adição) apenas pode ser aplicado sobre tipos de operando **Integer** (número inteiro) e **Real** (número de ponto flutuante). Os tipos em OCL podem ser básicos como **Integer** ou **Boolean**, coleções para representar associações, ou classes do modelo UML. Todas as expressões em OCL têm um tipo e avaliam para um valor quando aplicadas a um contexto específico.

Os usos mais comuns de OCL são em:

- **Invariantes para classes.** Um invariante é uma condição que deve ser válida durante todo o tempo pelos objetos daquela classe.
- **Invariantes para estereótipos.** Ele é usado quando da definição de um estereótipo para especificar uma condição que deve ser aplicável a todas as classes daquele estereótipo.
- **Pré e pós-condições para operações.** Uma pré-condição é uma restrição que especifica o que deve ser verdadeiro antes de uma operação de uma classe ser executada. Uma pós-condição é uma restrição que especifica o que deve ser verdadeiro depois que uma operação tem sido executada.
- **Regras navegacionais num modelo.** OCL pode ser usada para se especificar uma navegação. Começando de um objeto específico, pode-se navegar através de uma associação sobre o diagrama de classe para se referir a outros objetos e suas propriedades.
- **Regras de derivação.** Uma regra de derivação especifica como um valor específico é calculado de outros valores. Por exemplo, como um atributo é derivado de outros valores de outros atributos (ex.: a idade pode ser derivada da data de nascimento e da data atual).
- **Guardas.** Uma guarda é uma condição que especifica quando se deve executar uma atividade específica (ou processo) ou, quando várias alternativas existem,

qual das alternativas deve ser executada. Também podem ser usadas guardas em diagramas de estado para mostrar sobre que condições uma transição de estado acontece, ou num diagrama de sequência ou colaboração para especificar quando uma mensagem específica dever ser enviada. Uma guarda é uma expressão OCL que, quando avaliada verdadeira, indica que a mensagem, a transição de estado, ou a atividade daquela guarda deve ser executada ou enviada.

Na próxima seção mostraremos como a OCL pode ser usada para descrever regras de negócio.

3.7 REGRAS DE NEGÓCIO COM OCL

Um modelo de negócio contém regras de negócio que definem restrições, condições e políticas que mostram como os processos de negócios são executados. Regras de negócio são definidas tanto para satisfazer requisitos externos sobre o sistema (regulamentações e leis, ou restrições impostas) ou internamente para a garantia da satisfação de objetivos do negócio.

As regras de negócio podem ser especificadas formalmente em uma linguagem particular (que pode ser executável) ou informalmente em forma de texto. Regras complementam os diagramas, juntos eles contêm todas as informações como o negócio deve ser executado. Nesta seção vamos ilustrar o uso de OCL para especificar regras de negócio.

A linguagem OCL consiste de expressões que são relacionadas a um contexto específico (como uma classe, um atributo ou uma operação de uma classe). O contexto decide quais conjuntos de objetos a regra restringe (ex: tipicamente todos os objetos de uma classe específica ou todos os objetos que tomam parte num relacionamento de associação) ou define o lugar no modelo de onde a navegação através de OCL começa. Se a restrição é mostrada num diagrama com o estereótipo correto e uma linha pontilhada conectando ele a seu elemento contextual não há necessidade de explicitar a declaração do contexto. O contexto é especificado através da palavra reservada *context*

seguida pelo nome da parte do contexto (ex: nome da classe que a expressão deve ser aplicada). Por exemplo, na Figura 3.12 temos que:

```
context Pedido
```

indica que a expressão deve ser aplicada sobre a classe Pedido.

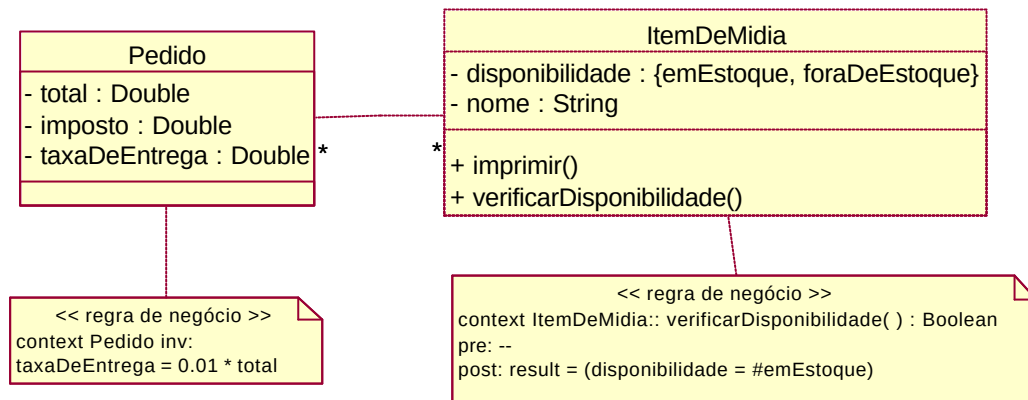


Figura 3.12 Regras de Negócio com OCL

Na Figura 3.12 temos a aplicação de um invariante sobre o atributo `taxaDeEntrega` que especifica indiretamente como o atributo deve ser calculado. Conforme mostrado a seguir:

```
context Pedido inv:
taxaDeEntrega = 0.01 * total
```

Quando uma propriedade, como um atributo, é usada numa expressão, fica implícito que a propriedade pertence à classe contexto. É possível mostrar esse relacionamento explicitamente usando a palavra reservada *self*. A palavra reservada *self* é uma referência a uma instância do tipo do contexto corrente. No exemplo abaixo, o contexto é a classe `Pedido` e a palavra reservada *self* refere-se ao objeto `Pedido` sobre o qual a expressão é aplicada.

```
context Pedido inv:
self.taxaDeEntrega = 0.01 * self.total
```

Hífenos duplos (--) indicam o início de um comentário em OCL. O comentário termina no fim da linha.

```
context Pedido inv:  
  taxaDeEntrega = 0.01 * total -- cálculo da taxa de entrega
```

Uma expressão OCL também pode ser aplicada a uma operação em vez de uma classe ou atributo. Restrições ou regras sobre operações são especificados através de pré e pós-condições. Por exemplo, a regra especificada na nota anexada ao ItemDeMidia na Figura 3.12 tem uma pós-condição especificada.

```
context ItemDeMidia:: verificarDisponibilidade( ) : Boolean  
pre: --  
post: result = (disponibilidade = #emEstoque)
```

O contexto dessa expressão é a operação `verificarDisponibilidade` da classe `Item de Mídia`. Nesse exemplo, temos que a pós-condição da operação `verificarDisponibilidade` mostra como calcular o valor de retorno desta operação. Quando uma pós-condição é especificada, a palavra reservada *result* é usada na expressão para descrever o valor de retorno. Essa pós-condição especifica que a operação irá retornar *true* se o item de mídia em questão tiver o atributo `disponibilidade` definido com o valor `emEstoque`. O atributo `disponibilidade` é um tipo enumerado com os valores possíveis `emEstoque` e `foraEstoque`. Em OCL, constantes de tipos enumerados devem ser precedidos pelo símbolo (#).

OCL tem um conjunto de operadores pré-definidos que podem ser consultados em: [OMG01, WAR99].

Como discutido anteriormente, usaremos na fase inicial dos requisitos a notação *i** e para a fase final dos requisitos a notação UML. Na próxima seção mostraremos como a partir de uma descrição inicial em *i** obter descrições em UML.

3.8 DIRETRIZES DE MAPEAMENTO DE *I** PARA UML

Nesta seção são apresentadas as diretrizes de mapeamento da técnica *i** para um modelo conceitual. Um modelo conceitual é uma representação de conceitos em um domínio do

problema [MAR95, FOW96, LAR01]. Em UML, um modelo conceitual é exibido como um conjunto de diagramas de estrutura estática. O modelo conceitual enfatiza os conceitos do domínio, e não entidades de software.

O Unified Process [JAC99] também sugere que os desenvolvedores-chaves (o arquiteto, em particular e alguns dos analistas sêniores) precisam de um bom entendimento do contexto no qual o sistema faz parte. Ele afirma que há pelo menos duas abordagens para expressar o contexto de um sistema numa forma que seja útil para desenvolvedores de software: modelagem do domínio e a modelagem de negócio. Um modelo do domínio descreve os conceitos importantes do contexto como objetos do domínio, e a ligações entre esses objetos.

A seguir, apresentaremos uma série de diretrizes para o mapeamento dos elementos do modelo organizacional em *i** num modelo conceitual representado através de um diagrama de classes. A definição das diretrizes está baseado no trabalho de Alencar [ALE99a] e o objetivo desse mapeamento é diminuir a lacuna existente entre a modelagem organizacional e a modelagem de sistemas de software [CAS00a, CAS00b, CAS01a, ALE00]. Devemos salientar que o modelo produzido oriundo do mapeamento é feito em nível conceitual, onde os aspectos de implementação ainda não são considerados.

No capítulo 6, é apresentado um estudo de caso que utiliza as diretrizes mostradas a seguir.

Diretriz D1: Relacionada com o mapeamento dos atores do modelo de *i**.

- Os *atores*, representados nos modelos de *i**, podem ser mapeados para *classes* em UML.

Por exemplo, o ator Medi@ representado na Figura 3.10 seria mapeado como uma classe com o nome de Medi@ (ver Figura 3.13).

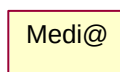


Figura 3.13 Classe Medi@

Diretriz D2: Relacionada com o mapeamento de tarefas.

- As tarefas, representadas nos modelos de i^* , podem ser mapeadas para operações de classes em UML.

Diretriz D2.1: Relacionada com as tarefas definidas no modelo de Dependência Estratégica (SD).

- Uma tarefa representada por uma ligação de dependência entre dois atores é mapeada para uma operação com visibilidade *public* sobre a classe que representa o ator responsável pela execução da tarefa (*dependee*).

A visibilidade da operação deve ser pública, para indicar que a classe que representa o ator dependente (*dependor*) precisa que a outra classe (que representa o *dependee*) realize a operação que representa a tarefa. Para expressar esse relacionamento de dependência entre os dois atores é criado um relacionamento estrutural (associação) entre as duas classes (caso ele já não exista) e os papéis de cada ator são representados na associação.

Por exemplo, a Figura 3.4 representa uma dependência de tarefa, onde o ator Medi@ é responsável por realizar a tarefa Fazer Pedido que o ator Cliente precisa. Através do mapeamento, a tarefa é representada como uma operação (ver Figura 3.14) sobre a classe que representa o ator Medi@ e é criada uma associação entre as classes Cliente e Medi@ que representam o *dependor* e o *dependee*, respectivamente.

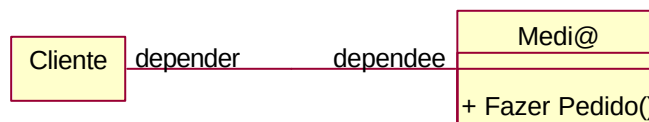


Figura 3.14 Mapeamento de uma Tarefa

Diretriz D2.2: Relacionada com o mapeamento das tarefas (sub-tarefas) definidas no modelo de Razão Estratégica (SR).

- Uma tarefa interna a um ator representada através do Modelo de Razão Estratégica (SR) pode ser mapeada para uma operação em UML com visibilidade *private* sobre a classe que representa o ator.

Quando construímos o Modelo de Razão Estratégica (SR) observamos o comportamento interno de cada ator. Como as tarefas (sub-tarefas) representadas através do SR são realizadas pelo o ator que se está detalhando o comportamento, elas são mapeadas como uma operação com a visibilidade privada sobre a classe que representa o Ator.

Por exemplo, a Figura 3.10 mostra as sub-tarefas Produzir Estatísticas e Gerenciar Loja da Internet sobre o ator Medi@ que são representadas como operações privadas sobre a classe Medi@, conforme ilustrado na Figura 3.15.

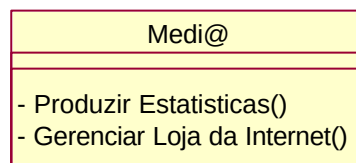


Figura 3.15 Mapeamento de uma sub-Tarefa

Diretriz D3: Relacionada com o mapeamento de Recursos.

Diretriz D3.1: Relacionada com os recursos definidos no modelo de Dependência Estratégica (SD).

- Um recurso representado por uma ligação de dependência entre dois atores é mapeado para uma classe.

Como um recurso representa uma entidade física ou uma informação em *i**, ele é mapeado como uma classe em UML e são criadas associações entre a classe que representa o recurso e classes que representa o ator (*dependor*) que necessita do recurso e entre a classe que representa o recurso e a classe que representa o ator (*dependee*) responsável por disponibilizar o recurso.

Por exemplo, a Figura 3.5 mostra uma dependência de recurso, onde o ator Fornecedor de Midia é responsável por disponibilizar o recurso Midia para o ator Media Shop.

Através do mapeamento, o recurso é representado como uma classe (ver Figura 3.16) e são criados uma associação entre a classe Midia e a classe Media Shop e entre a classe Mídia e a classe Fornecedor Mídia.

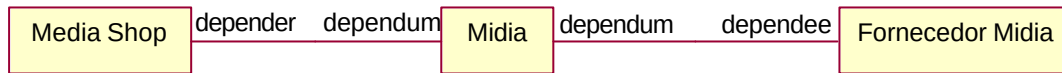


Figura 3.16 Mapeando um Recurso

Diretriz D3.2: Relacionada com o mapeamento dos recursos (sub-recursos) definidos no modelo de Razão Estratégica (SR).

- Um recurso interno a um ator, representado através do Modelo de Razão Estratégica (SR), pode ser mapeado para um atributo em UML com visibilidade *private* sobre a classe que representa o ator.

Como a informação ou a entidade física que o recurso representa só é utilizada internamente pelo ator, ele é mapeado como um atributo de visibilidade privada.

Diretriz D4: Relacionado com o mapeamento de objetivos.

- Os objetivos definidos na modelagem organizacional em *i** podem ser mapeados para atributos em UML.

Diretriz D4.1: Relacionado com os objetivos definidos no modelo de Dependência Estratégica (SD)

- Um objetivo representado por uma ligação de dependência entre dois atores é mapeado para um atributo com visibilidade *public* sobre a classe que representa o ator que é responsável por satisfazer o objetivo (*dependee*).

Os objetivos são bem definidos, portanto sempre é possível estabelecer se eles foram satisfeitos. Assim, o atributo possui o tipo *boolean* para indicar se o objetivo foi alcançado (*true*) ou não (*false*).

A ligação de dependência entre os dois atores pode indicar um relacionamento estrutural entre eles que pode ser representado em UML através de uma associação.

Por exemplo, a Figura 3.1 representa uma dependência de objetivo, onde o ator Cliente depende que o ator Media Shop atenda o seu objetivo de Comprar Itens de Mídia. Através do mapeamento o objetivo é representado como um atributo (ver Figura 3.17) sobre a classe que representa o ator Media Shop e é criada uma associação entre as classes Cliente e Media Shop que representam o *dependor* e o *dependee*, respectivamente.

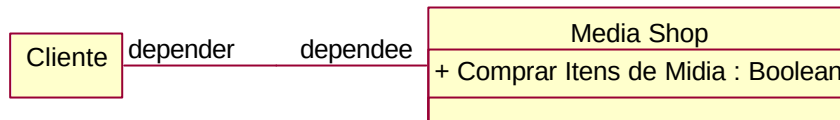


Figura 3.17 Mapeando um Objetivo

Diretriz D4.2: Relacionado com o mapeamento de objetivos (sub-objetivos) definidos no modelo de Razão Estratégica (SR).

- Um objetivo interno a um ator, representado através do Modelo de Razão Estratégica (SR), pode ser mapeado para um atributo em UML com visibilidade *private* sobre a classe que representa o ator.

O atributo possui o tipo *boolean* para indicar se objetivo foi alcançado (*true*) ou não (*false*). E a visibilidade é privada para indicar que o objetivo é interno ao ator.

Por exemplo, a Figura 3.10 mostra os sub-objetivos Tratar Pesquisa de Item e Tratar Pedidos da Internet sobre o ator Medi@ que são representadas como atributos privados do tipo Boolean sobre a classe Medi@, conforme ilustrado na Figura 3.18.

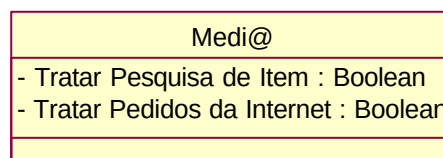


Figura 3.18 Mapeando de sub-bjetivos

Diretriz D5: Relacionada com o mapeamento de objetivos soft.

- Os objetivos soft definidos na modelagem organizacional em *i** podem ser mapeados em atributos em UML.

Os objetivos soft definidos na modelagem organizacional em I^* podem ser mapeados para atributos em UML.

Diretriz D5.1: Relacionado com os objetivos soft definidos no modelo de Dependência Estratégica (SD).

- Um objetivo representado por uma ligação de dependência entre dois atores é mapeado para um atributo com visibilidade *public* sobre a classe que representa o ator que é responsável por satisfazer o objetivo soft (*dependee*).

Os objetivos soft não são bem definidos, eles podem ser “satisfeitos” para algum grau. Portanto, o tipo enumerado é mais adequado para sua representação em UML, cujos valores representam diferentes graus de realização do objetivo soft. A ligação de dependência entre os dois atores pode indicar um relacionamento estrutural entre eles que pode ser representado em UML através de uma associação.

Por exemplo, a Figura 3.3 representa uma dependência de objetivo soft, onde o ator Cliente depende que o ator Bank Cty atenda o seu objetivo soft de Segurança. Através do mapeamento o objetivo soft é representado como um atributo do tipo enumerado ({satisfeito, parcialmente, não satisfeito}) sobre a classe que representa o ator Bank Cty e é criada uma associação entre as classes Cliente e Bank Cty que representam o *depender* e o *dependee*, respectivamente (ver Figura 3.19).

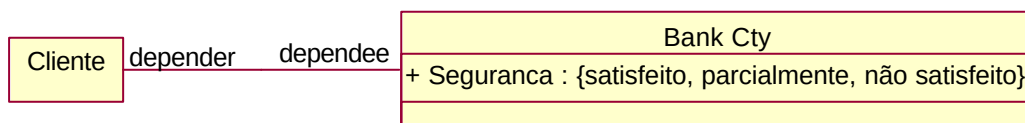


Figura 3.19 Mapeando um Objetivo Soft

Diretriz D5.2: Relacionado com o mapeamento de objetivos soft (sub-objetivos soft) definidos no modelo de Razão Estratégica (SR).

- Um objetivo soft interno a um ator representado através do Modelo de Razão Estratégica (SR) pode ser mapeado para um atributo em UML com visibilidade *private* sobre a classe que representa o ator.

O atributo possui o tipo enumerado para indicar o grau de realização do objetivo soft. A visibilidade é privada para indicar que o objetivo soft é interno ao ator.

Por exemplo, a Figura 3.10 mostra o sub-objetivo soft Atrair Novos Clientes sobre o ator Medi@ que é representado como um atributo privado do tipo enumerado ({satisfeito, parcialmente, não satisfeito}) sobre a classe Medi@, conforme ilustrado na Figura 3.20.

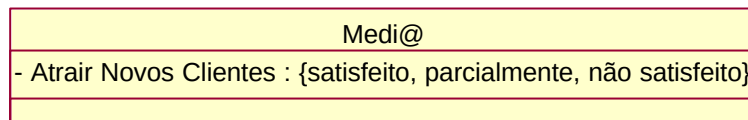


Figura 3.20 Mapeando um sub-objetivo Soft

Diretriz D6: Relacionada com o mapeamento de relacionamentos de decomposição de tarefa nos modelos em i^* .

- A decomposição de tarefa é representada através de pré e pós-condições (expressas em OCL) da operação correspondente em UML. A pré-condição é a conjunção (conector AND OCL) das pré-condições das sub-tarefas. A pós-condição é a conjunção (conector AND OCL) de todas: (i) pós-condições das sub-tarefas, (ii) atributos Boolean dos sub-recursos; (iii) atributos Boolean dos sub-objetivos; (iv) atributos enumerados do objetivo soft.

Assim, se a operação Op relativa à tarefa decomposta em uma sub-tarefa T_1 , um sub-objetivo Ob_1 , um sub-recurso Re_1 e um objetivo-soft $ObSoft_1$, então, segundo a diretriz, tem-se em OCL:

Op
pre: (pre-condOp₁)
post: (pos-condOp₁) AND (Atr_{Re1}) AND (Atr_{obj1}) AND (Atr_{objSoft1} >= 'valor')

Considere o exemplo da Figura 3.10, onde a tarefa Gerenciar Loja da Internet é decomposta nos objetivos Tratar Pesquisa de Item e Tratar Pedidos da Internet, na tarefa Produzir Estatística e no objetivo soft Atrair Novos Clientes. Vamos usar pre-Subtarefa e pos-Subtarefa para indicar pré e pós-condições genéricas da sub-tarefa.

Context Medi@:Gerenciar Loja da Internet : Boolean
pre: pre-Produzir Estatística
post: result = Tratar Pesquisa de Item AND Tratar Pedidos da Internet AND post -Produzir Estatística AND
Atrair Novos Clientes = 'Satisfeito'

As pré e pós-condições serão refinadas durante a modelagem dos requisitos finais. Mas poderíamos sugerir que uma pré-condição de Produzir Estatística seria ter registro de vendas e uma pós-condição seria ter o relatório dos 10 itens mais vendidos gerados.

Diretriz D7: Relacionada com o mapeamento de relacionamentos de meio-fim nos modelos em i^* .

- Análise de meio-fim é representada através de disjunções de todos os possíveis meios de alcançar o fim.

Diretriz D7.1: Relacionada com ligações do tipo meio-fim, onde o meio é uma tarefa e o fim é uma tarefa.

- Se o fim é uma tarefa e os meios são tarefas então a disjunção das pós-condições das tarefas meio implica a pós-condição da tarefa fim.

Com esta restrição, procuramos mostrar que existem n alternativas (os meios) para se executar uma tarefa.

Assim, se a operação Op representa a tarefa fim e pos_1 , pos_2 e pos_n representam as pós-condições das tarefas meios, temos que:

Context Class:Op

pre: pre_1 AND pre_2 AND pre_n

post: pos_1 OR pos_2 OR pos_n

Diretriz D7.2: Relacionadas com ligações do tipo meio-fim, onde o meio é uma tarefa e o fim é um objetivo, um objetivo-soft ou um recurso.

- O valor do fim é igual a disjunção das pós-condições das tarefas meio

No exemplo da figura 3.9, temos uma ligação de meio-fim do ator Media Shop, onde o objetivo fim Tratar Pedidos dos Clientes pode ser realizado através das tarefas meios Pedidos Por Telefone ou Pedidos Pela Internet ou Pedidos Na Loja. Vamos usar $pos_PedidosPorTelefon$ e $pos_PedidosPelaInternet$ e $pos_PedidosNaLoja$ para representar as pós condições genéricas das tarefas: Pedidos Por Telefone, Pedidos Pela Internet, Pedidos Na Loja, respectivamente. Assim a ligação meio-fim seria representado em OCL através da regra a seguir:

Context Media Shop inv:

Tratar Pedidos dos Clientes = pos-PedidosPorTelefone OR pos-PedidosPelaInternet OR
pos-PedidosNaLoja

Diretriz D7.3: Relacionadas com a expressão dos relacionamentos meio-fim dos modelos i^* , quando o meio e fim são, simultaneamente, objetivos ou objetivos-soft.

- Se o fim é um objetivo (soft) e os meios são objetivos (soft) então a disjunção dos valores dos meios implica o valor fim.

Assim, se o objetivo fim for representado pelo atributo $\text{Objetivo}_{\text{fim}}$ e os objetivos meios pelos atributos Objetivo_1 , Objetivo_2 e Objetivo_n , temos:

Context Class inv:

$\text{Objetivo}_{\text{fim}} = \text{Objetivo}_1 \text{ OR } \text{Objetivo}_2 \text{ OR } \text{Objetivo}_n$

Diferentes mapeamentos podem ser sugeridos, conforme apresentado na Seção 5.2. As diretrizes apresentadas nesta seção se concentram em alguns aspectos do i^* . Por exemplo, o refinamento de atores (agente, papel, posição), ligações de contribuição e o grau da dependência (comprometimento) não são tratados.

3.9 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentadas as técnicas de modelagem organizacional e análise orientada a objeto. Foram descritas as técnicas i^* e UML e uma série de diretrizes para o mapeamento dos modelos em i^* para UML. Naturalmente nem todos conceitos capturados na fase de requisitos iniciais corresponderão a modelos de sistema de software. Os modelos não têm um relacionamento um para um. Vários elementos do modelo organizacional não são parte do modelo de software, já que nem todas tarefas organizacionais requerem um sistema de software. Várias tarefas contêm atividades que são executadas manualmente fora do sistema de software e assim não se torna parte do modelo de sistema de software. Por outro lado, vários elementos no modelo de software compreendem detalhes técnicos de soluções de software e construtores que não são parte do modelo organizacional.

Contudo, para um efetivo uso das técnicas *i** e UML serão necessárias ferramentas de apoio. Para modelagem organizacional existe a ferramenta OME (Organization Modeling Environment) [OME02]. Para modelagem em UML existem várias ferramentas comerciais, sendo a Rational Rose [RAT02] uma das líderes de mercado [KIR01]. Porém, não basta ter ferramentas isoladas, é preciso que os modelos de *i** sejam mapeados para os diagramas de classes da UML. Portanto, será necessário averiguar os mecanismos que permitam mapear as descrições iniciais do *i** construídas através do OME para descrições finais no Rational Rose.

No próximo capítulo apresentaremos as ferramentas OME e Rational Rose. Descreveremos as suas características e os mecanismos utilizados para o mapeamento dos requisitos iniciais e finais entre as ferramentas.

Capítulo 4

Ferramentas OME e Rational Rose

Este capítulo apresenta uma breve introdução sobre integração de ferramentas. Em seguida são descritas as características das ferramentas OME e Rational Rose. Em particular, o repositório de dados do OME em Telos e os mecanismos de extensibilidade do Rational Rose (REI) são discutidos.

4.1 INTRODUÇÃO

As ferramentas CASE (Computer Aided Software Engineering) auxiliam nas atividades associadas ao processo da Engenharia de Software. Entre os benefícios que as ferramentas CASE podem produzir estão a redução do custo e do tempo de desenvolvimento de software. Elas são boas para automatizar tarefas repetitivas, manter os artefatos de projetos estruturados e gerenciar grandes quantidades de informação. Outro benefício do uso de ferramentas CASE é que elas podem fornecer novas formas de visualização da informação levando aos Engenheiros de Software a tomar melhores decisões e produzir software de maior qualidade.

Segundo Sommerville, as ferramentas CASE podem ser classificadas, para que sejam avaliadas e comparadas, nas seguintes perspectivas [SOM00]:

- Perspectiva Funcional – ferramentas são classificadas de acordo com suas funções específicas;
- Perspectiva de Processo – ferramentas são classificadas de acordo com as atividades de processo que suportam;
- Perspectiva de Integração – ferramentas são classificadas de acordo com a sua organização em unidades integradas, como ferramentas, bancadas e ambientes. As ferramentas suportam tarefas individuais do processo. Bancadas suportam fases ou atividades do processo. Os ambientes suportam todo ou pelo menos parte substancial do processo de software;

Apesar dos benefícios do uso de ferramentas CASE individuais, elas são mais úteis quando trabalham juntas e de uma forma integrada. Os benefícios da integração de ferramentas CASE são [PRE01, SOM00]:

- Transferência suave da informação (modelos, programas, documentos, dados) de uma ferramenta para outra;

- Aumento do controle do projeto que é alcançado através de um melhor planejamento, monitoramento e comunicação;
- Comunicação melhorada entre as pessoas que estão trabalhando sobre grandes projetos de software;
- Ferramentas especializadas podem ser combinadas para fornecer amplo suporte para as atividades do processo de desenvolvimento de software;
- Se as interfaces do usuário do sistema são integradas, o tempo de aprendizado e as taxas de erro são provavelmente reduzidos.

Os níveis de integração de ambientes ou bancadas de ferramentas CASE podem ser classificados, como [SOM00]:

- Integração de Plataforma – Ferramentas rodam sobre a mesma plataforma de hardware/sistema operacional;
- Integração de Dados – Ferramentas operam usando um mesmo modelo de dados compartilhados;
- Integrações por Apresentação – Ferramentas oferecem uma interface de usuário comum;
- Integrações por Controle – Ferramentas podem ativar e controlar a operação de outras ferramentas;
- Integração por Processo – O uso da ferramenta é guiado através de um modelo de processo explícito.

Contudo, a maioria das ferramentas CASE em uso atualmente não são construídas usando todos esses níveis de integração. De fato, a maioria das ferramentas permanece como soluções pontuais. Ou seja, uma ferramenta é usada para auxiliar uma atividade particular da engenharia de software, mas não se comunica diretamente com outras ferramentas e não é parte de um ambiente integrado.

Uma maneira de melhorar a integração de ferramentas individuais é através de mecanismos de troca de dados. Ou seja, as ferramentas devem produzir saídas num formato padrão que permitam que outras ferramentas leiam nesse formato. Dessa forma, a construção de programas que transformam as saídas de uma ferramenta em entradas de outra ferramenta fazem a integração das ferramentas. Outro mecanismo de integração usado por ferramentas pontuais é a disponibilidade de interfaces para acesso de dados e serviços. Esse último mecanismo é mais eficiente, devido a transparência dos formatos de armazenamento dos dados, além de permitir que ferramentas externas usem os serviços fornecidos pela ferramenta.

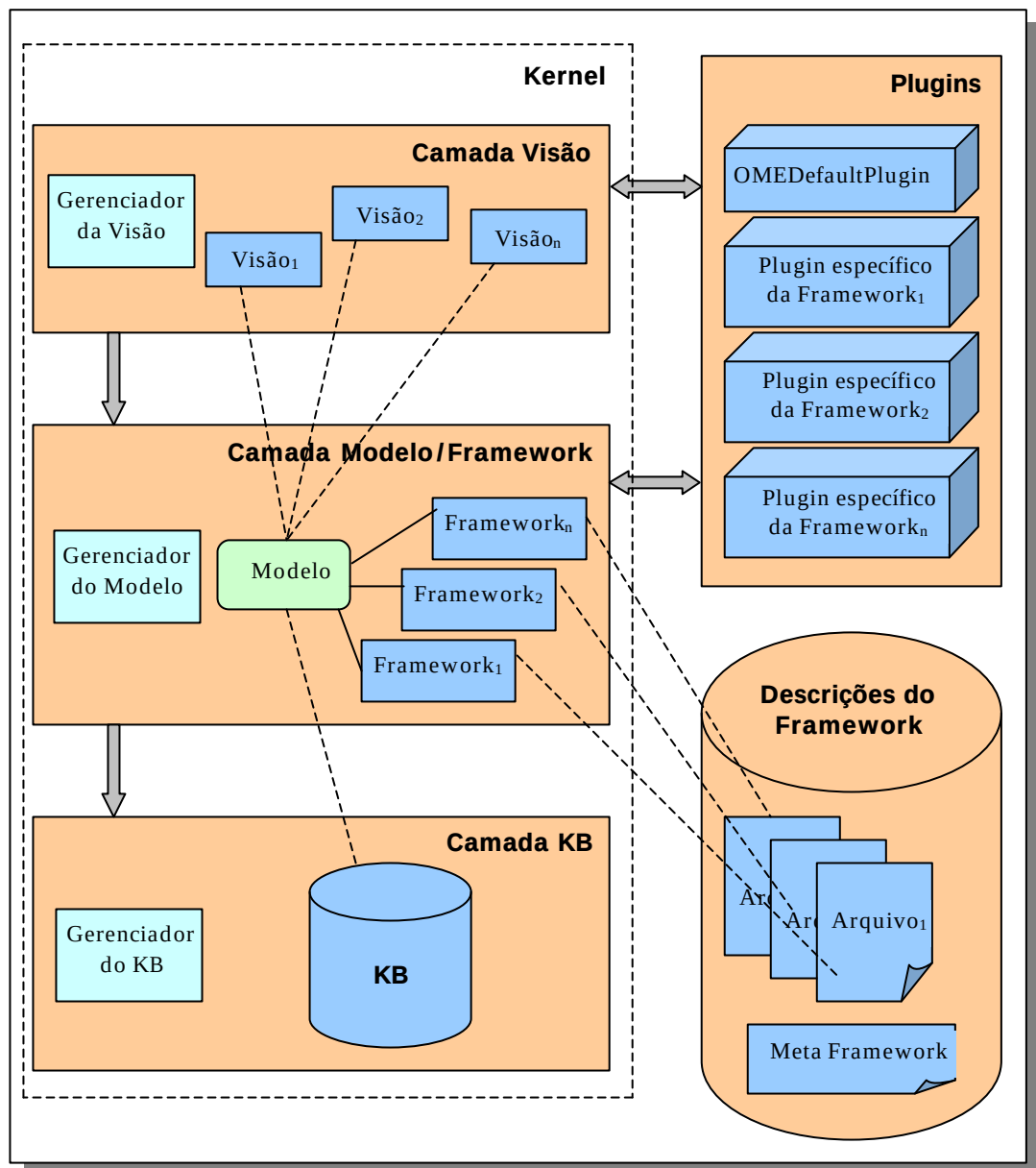
Nas próximas seções descrevemos a ferramenta OME que suporta a modelagem dos requisitos iniciais (*early requirements*) ou organizacionais e a ferramenta Rational Rose que poderá ser usada para modelar os requisitos finais (*late requirements*).

4.2 ORGANIZATION MODELING ENVIRONMENT (OME)

O OME é uma ferramenta para análise e modelagem orientada a agente e/ou orientada a objetivo. Ela está sendo desenvolvido na Universidade de Toronto pelo grupo de pesquisa do Knowledge Management Lab [KML02]. Atualmente a ferramenta suporta modelagem em *i** [YU95a], NFR [CHU00] e GRL [GRL02]. Contudo a ferramenta foi projetada com uma arquitetura que permite estendê-la e adaptá-la facilmente para suportar novas *frameworks* de modelagem. Nesta seção vamos mostrar a arquitetura da ferramenta e em particular como são representados os modelos em *i** no OME. O OME é composto basicamente de três componentes: o *kernel*, os *plugins* e um grupo de arquivos que descrevem a configuração dos *frameworks* conforme representado na Figura 4.1. O *Kernel* do OME tem uma arquitetura em camadas, compreendida de três níveis: a camada Visão, a camada Modelo/*Framework* e a camada Base de Conhecimento (*Knowledge Base* - KB). Os *plugins* são classes que implementam as funcionalidades específicas dos *frameworks* que são acoplados ao *kernel* do OME.

No núcleo do *Kernel* está a camada KB. Ela armazena os objetos de um modelo, seus relacionamentos, e seus atributos. Os detalhes dos *frameworks* para o modelo são

também mantidos nesta camada. Esta camada é responsável pela checagem semântica (para garantir que o modelo que está sendo desenvolvido adere ao *framework* armazenado no KB). Há um componente gerenciador do KB que é responsável por criar uma nova base de dados (KB) para cada modelo criado ou carregado. O KB é um repositório Telos [KOU89, MYL90], onde os modelos e seus *frameworks* são mantidos, como uma série de proposições Telos. A camada KB também fornece uma interface que



permite adicionar, modificar e remover essas proposições.

Figura 4.1 Diagrama de Arquitetura do OME

A próxima camada do Kernel é a *Modelo/Framework*. O papel dessa camada é abstrair o KB do resto da ferramenta. De forma a fornecer uma interface mais simples para a informação armazenada no KB, como também de isolar o resto da ferramenta de uma potencial re-implementação do KB. Quando o usuário da ferramenta deseja criar ou abrir um modelo, os detalhes desse pedido são passados para o Gerenciador de Modelo que instrui o Gerenciador do KB para criar um novo KB. O Gerenciador de Modelo inicia a construção dos componentes do módulo de Visão. Ele também determina quais *plugins* devem ser carregados e fornece uma interface para que esses *plugins* possam ser acessados pelo módulo de Visão. O Gerenciador de Modelo também é responsável por salvar os modelos, o que envolve instruir o KB para colocar o seu conteúdo para um arquivo. A camada *Modelo/Framework* tem acesso a camada KB, mas a camada KB é totalmente inconsciente da camada *Modelo/Framework*.

A camada Visão é responsável pela apresentação das representações dos modelos para os usuários, e por dar acesso ao usuário para manipular o modelo e/ou suas representações através de uma interface gráfica fornecida por cada Visão. Cada modelo pode ter vários tipos de visões e para cada modelo há um Gerenciador de Visão que controla todas as visões. A camada Visão também é responsável por fornecer para os usuários o acesso às funcionalidades fornecidas pelos *plugins*, como também é o meio pelo quais os *plugins* recebem as entradas dos usuários.

O componente Plugins é um conjunto de módulos que estende a funcionalidade do OME. Um *plugin* é simplesmente uma coleção de métodos, cada um dos quais pode analisar e/ou alterar a visão do modelo, ou o próprio modelo associado. Plugins são carregados dinamicamente, dependendo do modelo e dos *framework(s)* em uso. Essa arquitetura permite que funcionalidades específicas de uma *framework* sejam implementadas num *plugin*. Apenas, os *plugins* que são associados com os *frameworks* usados pelo modelo corrente serão carregados.

Os *frameworks* disponíveis para um modelo particular são determinados pelos vários arquivos de Descrições de Framework. Esses arquivos especificam os tipos de objetos que podem ocorrer dentro de um modelo, e as restrições sobre os relacionamentos

permitidos entre eles. Atualmente, informações descrevendo a representação dos objetos são também contidas dentro das descrições desses *frameworks*. Quando da criação de um novo modelo, o usuário é apresentado com uma opção de escolher quais *frameworks* ele deseja usar com o novo modelo. Existe o arquivo “ome.tel” que fornece a raiz da hierarquia sobre o qual todos os outros arquivos de descrição de framework devem ser baseados. O arquivo “istar.tel” é o arquivo de descrição da *framework i** (ver Apêndice D).

4.2.1 Telos

Nesta seção, nós introduzimos a linguagem de representação de conhecimento Telos [KOU89, MYL90] que é utilizada pelo OME para representar os seus modelos. Telos tem sua origem na linguagem RML (Requirements Modelling Language) [GRE84] e numa versão mais recente, chamada de CML (Conceptual Modelling Language) [STA86, KOU88]. A principal diferença é que a última adota uma estrutura de dados mais sofisticada para representação de conhecimento e suporta características novas para representação de conhecimento temporal e definição de meta conceitos. Telos, por sua vez é uma versão mais simples do CML, tanto de uma perspectiva da definição da linguagem como também da implementação.

Toda a informação em Telos é representada através de proposições. Formalmente, proposições são quádruplos com os componentes *from*, *label*, *to* e *when* denotando respectivamente a fonte, o rótulo, o destino e o intervalo de tempo da proposição, ou seja, o ciclo de vida do relacionamento que está sendo representado pela proposição. Esses componentes por si mesmos são proposições. Proposições podem ser de dois tipos disjuntos: indivíduos (entidades, nós, objetos, etc.) e atributos (relacionamentos, ligações, etc.).

Em Telos, as proposições podem ser classificadas em quatro tipos: Token, SimpleClass, MetaClass e MetametaClass. Token contém indivíduos que não podem ter instâncias (as folhas da hierarquia de instanciação). SimpleClass contém indivíduos que podem ter apenas Tokens como instâncias (as classes no primeiro nível da hierarquia de

instanciação). MetaClass contém indivíduos que podem ter apenas SimpleClass como instâncias (classes no segundo nível da hierarquia da instanciação).

As proposições em Telos apresentam a cláusula IN que dá uma lista de classes no qual o objeto definido pode torna-se uma instância, a cláusula ISA que define herança enquanto a cláusula WITH introduz atributos.

A representação completa da *framework i** em Telos para ferramenta OME pode ser verificada através do arquivo “istar.tel” (Apêndice D) contido na ferramenta. Contudo para efeito do mapeamento das representações em *i** da ferramenta OME, apenas é necessário explicarmos como os *tokens* que representam os elementos do *i** são formatados. Nesta seção vamos apresentar um exemplo de cada tipo *token* utilizado pelo mapeamento da integração com o Rational Rose. Os exemplos foram retirados do arquivo armazenado pela ferramenta OME quando da sua utilização para modelagem do estudo de caso que será apresentado no Capítulo 6. Os *tokens* utilizados representam os elementos do *i**. Eles podem ser de dois tipos: elementos (ator, recurso, tarefa, objetivo e objetivo soft) e ligações (dependência, meio-fim, decomposição de tarefa).

Por exemplo, para o ator Medi@ do nosso estudo de caso, mostrado na Figura 4.2, temos a seguinte representação em Telos gerado pela ferramenta OME:

```
Token Element_32
  IN IStarActorElement
  WITH
    attribute, name
      : "Medi@"
    attribute, children
      : Element_166
    attribute, children
      : Element_165
    attribute, children
      : Element_164
    attribute, children
      : Element_163
    attribute, children
      : Element_162
    attribute, children
      : Element_161
    attribute, children
      : Element_160
    attribute, children
      : Element_80
    attribute, children
      : Element_81
    attribute, children
      : Element_82
    attribute, children
      : Element_83
    attribute, children
```

```

        : Element_90
    attribute, children
        : Element_91
    attribute, children
        : Element_92
    attribute, children
        : Element_93
    attribute, children
        : Element_94
    attribute, children
        : Element_95
    attribute, children
        : Element_141
END

```

onde:

- **Token** é uma palavra reservada da linguagem Telos que representa a instanciação de uma classe simples.
- **Element_32** é um identificador gerado automaticamente pela ferramenta quando da criação do novo elemento (o ator Medi@).
- **IN** é palavra reservada da linguagem Telos que indica um relacionamento de instanciação.
- **IstarActorElement** é a classe simples pelo qual o token **Element_32** é instância (ver Apêndice D). Essa propriedade é que indica que o elemento é um Ator.
- **WITH** palavra reservada que indica o início da descrição dos atributos.
- **attribute, name** indica o atributo que representa o nome dado pelo usuário ao ator (no exemplo, o atributo recebeu o valor **Media Shop**).
- **attribute, children** indica o atributo que representa um elemento interno (uma sub-tarefa, um sub-recurso, um sub-objetivo, ou um sub-objetivo soft) ao ator. No caso do ator Medi@, temos os seguintes elementos internos: **Element_166** (tarefa Formulário Padrão), **Element_165** (tarefa Formulário Seguro), **Element_164** (tarefa Por Telefone), **Element_163** (tarefa Por Fax), **Element_162** (objetivo Tratar pela Internet), **Element_161** (objetivo Comunicação Clássica), **Element_160** (objetivo Obter Identificação), **Element_80** (tarefa Gerenciar Loja da Internet), **Element_81** (tarefa Produzir Estatística), **Element_82** (objetivo Tratar Pedidos da Internet), **Element_83** (objetivo Tratar Pesquisa de Item), **Element_90** (objetivo soft Atrair Novos Clientes), **Element_91** (tarefa Carrinho de Compra), **Element_92** (objetivo Selecionar Item), **Element_93** (tarefa Adicionar Item), **Element_94** (tarefa Fazer Pre Pedido de Item Nao Disponível), **Element_95** (tarefa Confirmar Compra), **Element_141** (tarefa Pegar Item Selecionado).
- **attribute, links** representa ligações (*links*) com o toke. No caso do ator essas ligações poderiam ser depências de tarefa, objetivo, recurso, ou objetivo soft. O exemplo da Figura 4.2 não apresenta ligações.

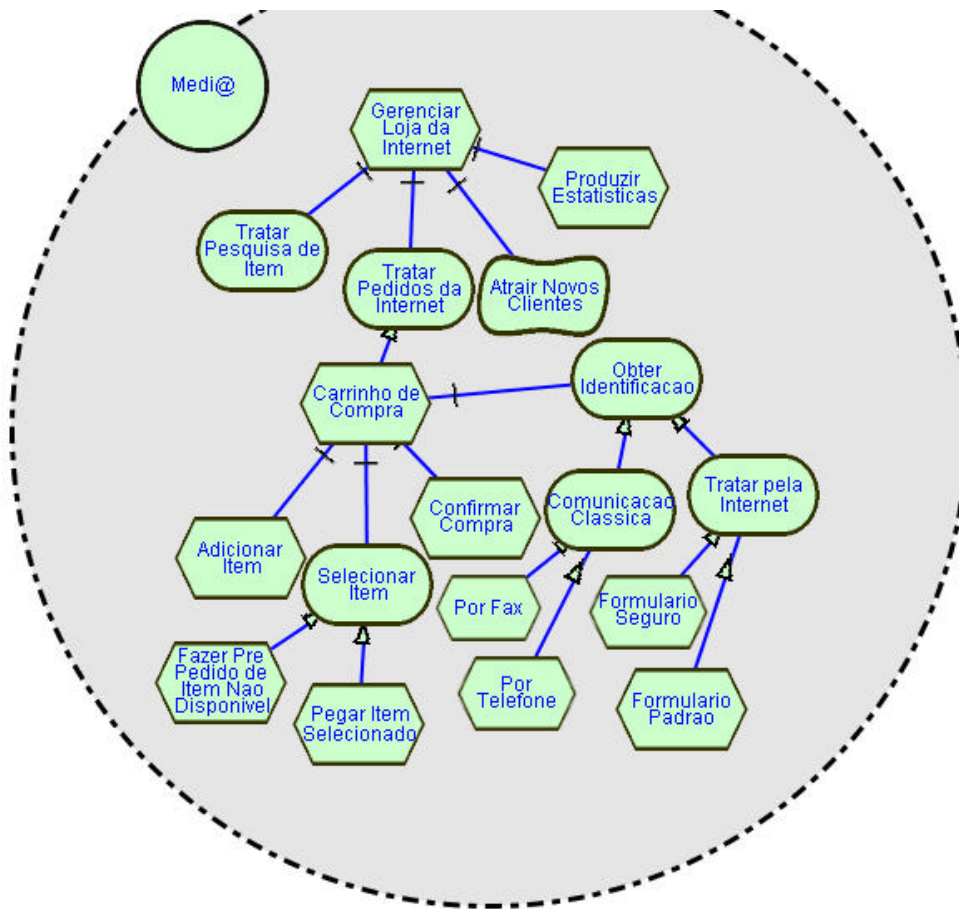


Figura 4.2 Ator Medi@

Por exemplo, para o objetivo Comprar Itens de Mídia que faz parte da ligação de dependência entre o ator Cliente e o sub-objetivo Tratar Pedidos dos Clientes que pertence ao ator Media Shop do nosso estudo de caso (ver Figura 4.3), temos a seguinte representação:

```
Token Element_1
  IN IStarGoalElement
  WITH
    attribute, name
      : "Comprar Itens de Midia"
    attribute, links
      : Link_4
    attribute, links
      : Link_54
  END
```

onde:

- Token é uma palavra reservada que representa a instanciação de uma classe simples.
- Element_1 é um identificador gerado automaticamente pela ferramenta quando da criação do novo elemento (o objetivo Comprar Itens de Mídia).
- IN palavra reservada que indica um relacionamento de instanciação.

- `IstarGoalElement` é a classe simples pelo qual o token `Element_1` é instância (ver Apêndice D). Essa propriedade é que indica que o elemento é um Objetivo.
- `WITH` palavra reservada que indica o início da descrição dos atributos.
- `attribute, name` indica o atributo que representa o nome dado pelo usuário ao ator (no exemplo, o atributo recebeu o valor `Comprar Itens de Midia`).
- `attribute, parent` atributo que indica que o objetivo é um elemento interno a um ator (ele é sub-objetivo). No nosso exemplo, o objetivo não pertence a nenhum elemento, por isso essa propriedade não é definida.
- `attribute, links` representa tokens do tipo `link` que estão associados ao token `Element_1`. No exemplo, as ligações (*links*) `Link_4` e `Link_54` representam ligações de dependência de objetivo. O `Link_4` representa uma ligação de dependência entre o ator `Cliente` e o objetivo `Comprar Itens de Mídia` e o `Link_54` representa uma dependência entre o objetivo `Comprar Itens de Mídia` e o sub-objetivo `Tratar Pedidos dos Clientes` que pertence ao ator `Media Shop` (ver Figura 4.3).

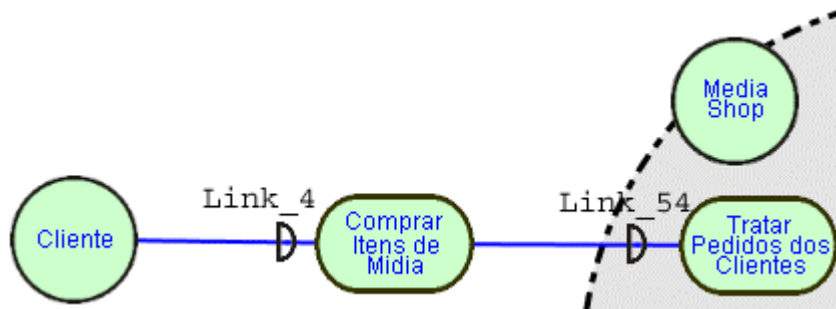


Figura 4.3 Objetivo Comprar Itens de Mídia

Para a tarefa `Fazer Pedido` que faz parte da ligação de dependência entre o ator `Cliente` e a sub-tarefa `Carrinho de Compra` que pertence ao ator `Medi@` (ver Figura 4.4) do nosso estudo de caso temos a seguinte representação:

```
Token Element_37
  IN IstarTaskElement
  WITH
    attribute, name
      : "Fazer Pedido"
    attribute, links
      : Link_67
    attribute, links
      : Link_154
  END
```

onde:

- `Token` palavra reservada que representa a instanciamento de uma classe simples.
- `Element_37` é um identificador gerado automaticamente pela ferramenta quando da criação do novo elemento (a tarefa `Fazer Pedido`).
- `IN` palavra reservada que indica um relacionamento de instanciamento.
- `IstarTaskElement` é a classe simples pelo qual o token `Element_37` é instância (ver Apêndice D). Essa propriedade é que indica que o elemento é uma Tarefa.

- WITH palavra reservada que indica o início da descrição dos atributos.
- attribute, name indica o atributo que representa o nome dado pelo usuário a tarefa (no exemplo, o atributo recebeu o valor Fazer Pedido).
- attribute, parent atributo que indica que a tarefa é um elemento interno a um ator (no exemplo, este atributo não ocorreu).
- attribute, links representa um relacionamento ligado ao token (no exemplo, a ligação Link_67 e Link_154). No exemplo, as ligações (links) Link_67 e Link_154 representam ligações de dependência de tarefa. O Link_67 representa uma ligação de dependência entre o ator Cliente e a tarefa Fazer Pedido e o Link_154 representa uma dependência entre a tarefa Fazer Pedido e a sub-tarefa Carrinho de Compra do ator Medi@ (ver Figura 4.4).

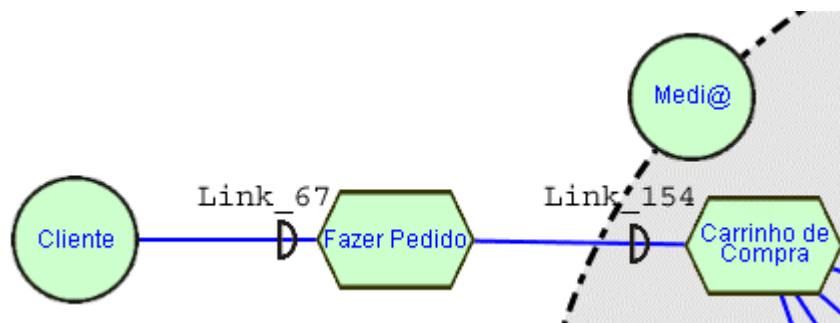


Figura 4.4 Tarefa Fazer Pedido

Para o objetivo soft Segurança que faz parte da ligação de dependência entre o ator Cliente e o ator Bank Cty do nosso estudo de caso, temos a seguinte representação:

```
Token Element_38
  IN IStarSoftGoalElement
  WITH
    attribute, links
      : Link_70
    attribute, links
      : Link_69
    attribute, name
      : "Seguranca"
END
```

onde:

- Token palavra reservada que representa a instanciação de uma classe simples.
- Element_38 é um identificador gerado automaticamente pela ferramenta quando da criação do novo elemento (o objetivo soft Segurança).
- IN palavra reservada que indica um relacionamento de instanciação.
- IStarSoftGoalElement é a classe simples pelo qual o token Element_38 é instância (ver Apêndice D). Essa propriedade é que indica que o elemento é um Objetivo Soft.
- WITH palavra reservada que indica o início da descrição dos atributos.
- attribute, name indica o atributo que representa o nome dado pelo usuário a tarefa (no exemplo, o atributo recebeu o valor Seguranca).

- `attribute, parent` atributo que indica que a tarefa é um elemento interno a um ator (no exemplo, este atributo não ocorreu).
- `attribute, links` representa um relacionamento ligado ao token (no exemplo, a ligação `Link_70` e `Link_69`). No exemplo, as ligações (*links*) `Link_69` e `Link_70` representam ligações de dependência de objetivo soft. O `Link_69` representa uma ligação de dependência entre o ator `Cliente` e o objetivo soft `Seguranca` e o `Link_70` representa uma dependência entre o objetivo soft `Seguranca` e o ator `Bank Cpy` (ver Figura 4.5).

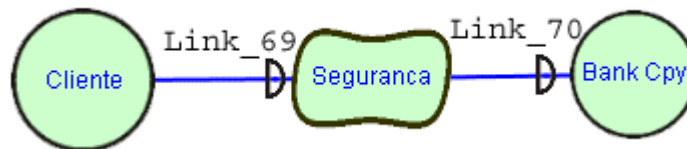


Figura 4.5 Objetivo Soft Seguranca

Para o recurso Item de Mídia que faz parte da ligação de dependência de recurso entre o ator Media Shop e o ator Fornecedor de Midia do nosso estudo de caso, temos a seguinte representação:

```
Token Element_7
  IN IStarResourceElement
  WITH
    attribute, links
      : Link_47
    attribute, links
      : Link_9
    attribute, name
      : "Item de Midia"
END
```

onde:

- `Token` palavra reservada que representa a instanciação de uma classe simples.
- `Element_7` é um identificador gerado automaticamente pela ferramenta quando da criação do novo elemento (recurso Item de Midia).
- `IN` palavra reservada que indica um relacionamento de instanciação.
- `IStarResourceElement` é a classe simples pelo qual o token `Element_7` é instância (ver Apêndice D). Essa propriedade é que indica que o elemento é um Recurso.
- `WITH` palavra reservada que indica o início da descrição dos atributos.
- `attribute, name` indica o atributo que representa o nome dado pelo usuário a tarefa (no exemplo, o atributo recebeu o valor `Item de Midia`).
- `attribute, parent` atributo que indica que a tarefa é um elemento interno a um ator (no exemplo, este atributo não ocorreu).
- `attribute, links` representa um relacionamento ligado ao token (no exemplo, a ligação `Link_47` e `Link_9`). No exemplo, as ligações (*links*) `Link_47` e `Link_9` representam ligações de dependência de recurso. O `Link_47` representa uma ligação de dependência entre a sub-tarefa Aumentar Estoque que pertence ao Media Shop e o recurso Item de Mídia. O `Link_9` representa uma dependência de recurso entre o recurso Item de Mídia e o ator Fornecedor Midia (ver Figura 4.6).

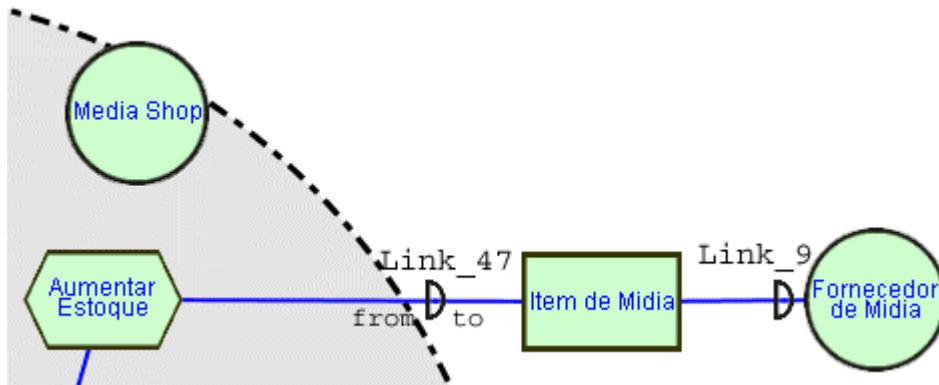


Figura 4.6 Recurso Item de Midia

Uma ligação de dependência como o Link_47 apresentado na Figura 4.6 é representado, como:

```
Token Link_47
  IN IStarDependencyLink
  WITH
    attribute, name
      : ""
    attribute, from
      from : Element_30
    attribute, to
      to : Element_7
  END
onde:
```

- Token palavra reservada que representa a instanciação de uma classe simples.
- Link_47 é um identificador gerado automaticamente pela ferramenta quando da criação do novo elemento.
- IN palavra reservada que indica um relacionamento de instanciação.
- IStarDependencyLink é a classe simples pelo qual o token Link_47 é instância (ver Apêndice D). Essa propriedade é que indica que o token é uma ligação de dependência.
- WITH palavra reservada que indica o início da descrição dos atributos.
- attribute, name indica o atributo que representa o nome dado pelo usuário a ligação de dependência (no exemplo, o usuário não deu nenhum valor) .
- attribute, to atributo que indica o destino da ligação. No exemplo, ele assume o valor Element_7 que representa o recurso Item de Midia.
- attribute, from atributo que indica a origem da ligação. No exemplo, ele assume o valor Element_30 que representa a tarefa Aumentar Estoque que pertence ao Media Shop.

Na Figura 4.2, temos a decomposição de tarefa Gerenciar Loja da Internet (Element_80) na tarefa Tratar Pesquisa de Item (Element_83). Essa ligação (*link*) é representada através do token Link_95, mostrado a seguir:

```
Token Link_95
  IN IStarDecompositionLink
  WITH
    attribute, to
      to : Element_80
    attribute, from
      from : Element_83
    attribute, name
      : ""
  END
```

onde:

- Token palavra reservada que representa a instanciação de uma classe simples.
- Link_95 é um identificador gerado automaticamente pela ferramenta quando da criação do novo elemento.
- IN palavra reservada que indica um relacionamento de instanciação.
- IStarDecompositionLink é a classe simples pelo qual o token Link_95 é instância (ver Apêndice D). Essa propriedade é que indica que o token representa uma ligação de decomposição.
- WITH palavra reservada que indica o início da descrição dos atributos.
- attribute, name indica o atributo que representa o nome dado pelo usuário a ligação de decomposição (no exemplo, o usuário não deu nenhum valor) .
- attribute, to atributo que indica que a tarefa que está sendo decomposta. No nosso exemplo a tarefa Gerenciar Loja da Internet (Element_80)
- attribute, from indica a decomposição da tarefa. No nosso exemplo a tarefa Tratar Pesquisa de Item (Element_83).

Na Figura 4.2, temos a ligação meio-fim (Link_59), onde o meio para atingir o objetivo Tratar Pedidos da Internet (Element_82) é através da tarefa Carrinho de Compra (Element_91) que é representado a seguir:

```
Token Link_159
  IN IStarMeansEndsLink
  WITH
    attribute, name
      : ""
    attribute, from
      from : Element_91
    attribute, to
      to : Element_82
  END
```

onde:

- Token palavra reservada que representa a instanciação de uma classe simples.

- `Link_` é um identificador gerado automaticamente pela ferramenta quando da criação do novo elemento.
- `IN` palavra reservada que indica um relacionamento de instanciação.
- `IStarMeansEndLink` é a classe simples pelo qual o token `Link_159` é instância (ver Apêndice D). Essa propriedade é que indica que o token representa uma ligação de meio-fim.
- `WITH` palavra reservada que indica o início da descrição dos atributos.
- `attribute, name` indica o atributo que representa o nome dado pelo usuário a ligação de meio-fim. (no exemplo, o usuário não deu nenhum valor) .
- `attribute, to` atributo que indica o fim. No exemplo, o objetivo Tratar Pedidos da Internet (`Element_82`).
- `attribute, from` atributo que representa o meio. No exemplo, o meio é a tarefa Carrinho de Compra.

4.3 RATIONAL ROSE

O Rational Rose é uma ferramenta CASE desenvolvida pela Rational Corporation [RAT02] que suporta a análise, modelagem e o projeto de sistemas de software orientados a objeto através das notações de Booch [BOO93], Object Modeling Technique (OMT) [RUM91] e Unified Modeling Language (UML) [OMG01]. O IDC (International Data Corporation) através do seu relatório intitulado "Analysis, Modeling and Design Tools Market Forecast, 2001-2005" [KIR01] publicou informações sobre o mercado de vendas de ferramentas de Análise, Modelagem e Projeto e discutiu a tendência desse segmento até 2005. Segundo o relatório o Rational Rose foi líder de vendas no ano 2000 com 28.7% do mercado e também preve que a indústria de ferramentas de análise, modelagem e projeto de software terá um crescimento positivo nos próximos anos.

O Rational Rose possui as seguintes características:

- Disponibilidade em várias plataformas: Windows e UNIX/Linux.
- Suporte multi-usuário.
- Suporte completo a UML: Suporta os nove diagramas do UML (Diagramas de Caso de Uso, Diagramas de Seqüência, Diagramas de Colaboração, Diagramas de Atividade, Diagramas de Estado, Diagramas de Classe, Diagramas de Objeto,

Diagramas de Componente e Diagramas de Implantação), além de permitir o usuário adicionar os seus próprios estereótipos.

- Modelos de Dados: Suporte o *Data Modeling Profile* [USI01] que permite a criação de modelos de dados baseados em UML. A ferramenta permite a engenharia direta de modelos de objetos para modelos de dados e de modelos de dados para um banco de dados ou um para um script DDL (Data Definition Language) de um banco de dados. Ela também tem possibilita a engenharia reversa de um banco de dados ou de um script DDL para um modelo de dados e em seguida para um modelo de classes.
- Rose Model Integrator: é uma ferramenta que possibilita comparar e integrar (*merging*) os modelos do Rose. Por exemplo, dois desenvolvedores podem precisar modificar um modelo compartilhado ao mesmo tempo. Eles podem cada um copiar o modelo, modificá-lo separadamente, e em seguida usarem o Model Integrator para combinar as mudanças de volta para uma única cópia compartilhada do modelo. Ou eles podem usar Model Integrator para comparar os modelos e identificar as diferenças entre eles.
- Framework Wizard: fornece uma biblioteca de templates que podem ser usadas como modelos quando da criação de um projeto. Através da escolha da *framework* apropriada, o projeto é automaticamente inicializado com uma arquitetura predefinida e conjunto de elementos que podem ser reusados.
- Web Publisher: permite que você crie uma versão web (HTML) do modelo do Rose que pode ser visualizada através de navegador como o Netscape Navigator ou Microsoft Internet Explorer.
- Integração de Ferramentas: possui integração com diversas ferramentas, como Microsoft Project, Rational RequisitePro, Rational ClearCase, Visual Basic, Jbuldier, etc.
- XMI: possui uma ferramenta para importar e exportar modelos do Rose para e do formato de intercâmbio de dados XML (eXtensible Markup Language)

[BRA00], usando o XMI (XML Metadata Interchange) 1.0 DTD [OMG00] padrão da Object Management Group (OMG) [OMG02] para promover a troca de dados entre ferramentas de modelagem.

- Rose Extensibility Interface: uma interface que a permite a personalização dos menus do Rose e a criação de scripts para automatizar funções do Rose e acessar elementos do Rose. O Rose Extensibility Interface será descrito em mais detalhe na próxima seção.

4.3.1 Rational Rose Extensibility (REI)

O Rational Rose Extensibility [REI01] fornece várias maneiras para estender e personalizar suas capacidades de forma a atender as necessidades de desenvolvimento de software específicas. Ele permite:

- Personalizar os menus do Rose;
- Automatizar funções manuais do Rose através de scripts (por exemplo, criação diagramas, criação de classes, geração de documentação);
- Executar funções do Rational Rose dentro de outras aplicações usando o objeto **Rational Rose Automation (RoseApp)**.
- Acessar classes, propriedades e métodos do Rational Rose dentro de seu ambiente de desenvolvimento de software (ex.: Visual Basic, Visual C++) incluindo o **Rational Rose Extensibility Type Library** no seu ambiente.
- Ativar o **Rational Rose add-ins** usando o **Add-In Manager**.

O Rational Rose é desenvolvido baseado em componentes. Os seus componentes são definidos através do Rose Extensibility Interface Model que é essencialmente o meta modelo do Rose que expõe os seus pacotes, classes, propriedades e métodos que definem e controlam a aplicação do Rose e todas as suas funções. A Figura 4.7 mostra o núcleo dos componentes do Rational Rose, os componentes da interface de

estensibilidade do Rational Rose, e os relacionamentos entre eles. Esses componentes são:

- **Rational Rose Application** – objetos do Rational Rose Extensibility que fazem a interface com as funcionalidades da aplicação do Rational Rose.
- **Diagrams** – objetos do Rational Rose Extensibility que fazem a interface com os diagramas e visões do Rational Rose.
- **Model Elements** – objetos do Rational Rose Extensibility que fazem interface com os elementos do modelo do Rational Rose.
- **Rational Rose Extensibility Interface** – um conjunto comum de interfaces usadas pelo Rational Rose Script e Rational Rose Automation para acessar o Rational Rose.
- **Rational Rose Script** – um conjunto de objetos do Rational Rose Script que permite o Rational Rose Scripts automatizar funções do Rational Rose.
- **Rational Rose Automation** – um conjunto de objetos do Rational Rose para funcionar como um controlador ou servidor de automação OLE.

Os componentes: Rational Rose Script e o Rational Rose Automation são responsáveis pela comunicação externa do Rose.

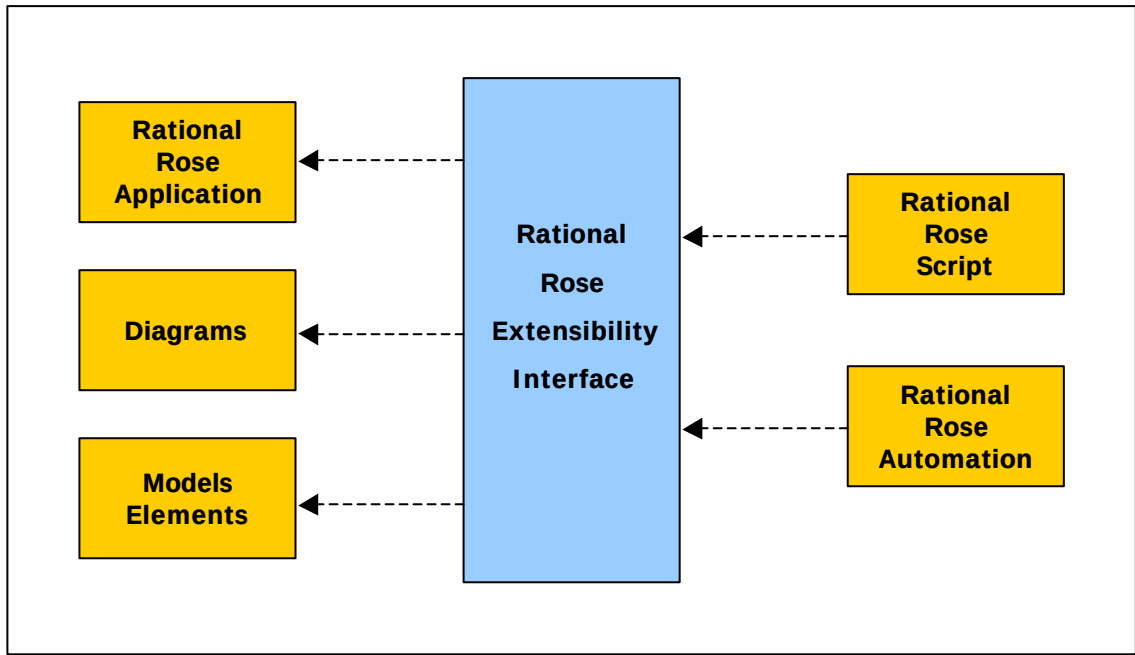


Figura 4.7 Diagrama de Componentes do Rose

4.3.1.1 Personalizando os menus do Rose

O Rational Rose permite que os seus menus principais e os menus de atalho sejam personalizados. Contudo, o Rose não permite que novos menus sejam adicionados a barra de menu. Ele permite apenas que os menus sejam estendidos através da adição:

- Sub menus;
- Separadores de menus;
- Itens de menus: que podem executar scripts do Rose, comandos do sistema e programas externos.

O seguinte procedimento resume os passos gerais para personalizar menus do Rose:

1. Abra o arquivo Rose.mnu com qualquer editor de texto (arquivo que reside no diretório onde o Rose está instalado).
2. Adicione submenus, itens de menu e separadores de menu.

3. Se o item do menu executa um *script*, adicione ou edite o caminho virtual para os *scripts*, a menos que ele já esteja definido.

4. Salve o arquivo Rose.mnu.

O apêndice A detalha como deve ser configurado o arquivo Rose.mnu para a utilização da ferramenta GOOD descrita no próximo capítulo.

O Rational Rose também permite que seus menus de atalho sejam personalizados. As ferramentas (*add-ins*) podem tirar vantagem dessa característica da seguinte maneira:

- Fornecendo um acesso mais rápido as funcionalidades das ferramentas (*add-ins*);
- Controlando quando o item de menu deve aparecer (ex.: quando clicado de um diagrama, classe, atributo, caso de uso, etc.).

4.3.1.2 Rose Script

Os *scripts* do Rational Rose são escritos através da linguagem Rose Script que é uma versão estendida da linguagem Summit BasicScript. As extensões do Rose permitem a automatização de funções específicas do Rose e em alguns casos, executar funções que não estão disponíveis através da interface do usuário do Rose.

Através do Rose Scripting se pode comunicar com o modelo do REI (Rose Extensibility Interface). O modelo do REI é essencialmente um meta-modelo do modelo do Rose, expondo os pacotes, classes, propriedade e métodos que definem e controlam a aplicação do Rose e todas a suas funções. A Figura 4.8 mostra os pacotes que compreende o modelo do REI, descritos a seguir:

- Application Classes – Para criar um novo modelo, selecionar um modelo existente como um modelo corrente e determinar as características do Rational Rose que estão sendo controlado pelo *script*.
- Model Classes – Fornece propriedades e métodos que permitem usar os objetos de um modelo, como adicionar objetos (classes, categorias, diagramas, etc.),

recuperar objetos de um modelo e remover objetos de um modelo. As classes Model expostas através do REI são: ControllableUnit, DefaultModelProperties, DependencyRelation, DeploymentUnit, Element, ExternalDocument, Model, ObjectFlow, Package, Property, Relation, RoseItem e StringCollection.

- View Classes – Permite você definir as representações visíveis dos elementos lógicos do modelo. Por exemplo, um ícone de uma classe sobre um diagrama de classe. O ícone representa a classe real e ao mesmo tempo ele pode ser manipulado como um objeto gráfico (*view object*). As classes View expostas através do REI são: Diagram, RoseItemView e NoteView.
- Logical Classes – Permite você definir e manipular as classes e categories num modelo. Essas classes incluem: Category, Class, ClassDiagram e ClassView.
- Physical Classes – Elas definem a realização física das classes e objetos num projeto lógico de um sistema. Essas classes definem os módulos (componentes) e subsistemas que contém as unidades de código que implementam o projeto. As classes Physical expostas através da interface do REI são: Module, ModuleDiagram, Subsystem, ComponentView e SubsystemView.
- Scenario Classes – Permite você definir instâncias do caso de uso (*use case*) num modelo. Essa capacidade é fornecida no REI através das seguintes classes: ObjectInstance, Message, ScenarioDiagram, Link, InstanceView, Persistence, Frequency, Synchronization.
- Deployment Classes – As classes Deployment definem a implementação do projeto lógico do modelo. As classes Deployment exposta no REI são: Process, Processor, Device e DeploymentDiagram.
- Use Case Classes – O Rose Extensibility Interface expõe a classe UseCase e a classe UseCaseDiagram que permite a criação e edição de casos de uso (*use case*) de cenários e diagramas de classe num modelo.

- **State Classes** – As classes State expõe as propriedades e métodos que permitem a criação e a manipulação das definições de estado. Isso inclui a definição de estados, máquinas de estados e gráficos de estados (*statecharts*), transições e os eventos que as disparam, como também ações resultantes de tais eventos. As classes de estado também expõem as propriedades e métodos que permitem você criar e manipular diagramas de atividade. As classes State são: Action, Activity, Decision, Event, State, StateMachine, SwimLane, SyncItem, Transition.
- **Base Classes** – Conjunto de classes que disponibilizam funções básicas, como coleções de strings (StringCollection) e propriedades do modelo, como nome, tipo e ferramentas associadas.
- **Collections** – Para a maioria dos elementos do Rose, há uma coleção correspondente. Por exemplo, para cada classe há uma classe coleção (*collection*).
- **Fundamental Types** – Fornece os tipos básicos, tais como Boolean, Date, Integer, String, Currency e Long.
- **Rich Types** – Fornece os tipos complexos do REI como Arrays, Object (usado para declarar variáveis que referenciam objetos dentro de uma aplicação usando OLE), Variant (tipo usado para declarar variáveis que podem assumir tipos

variados).

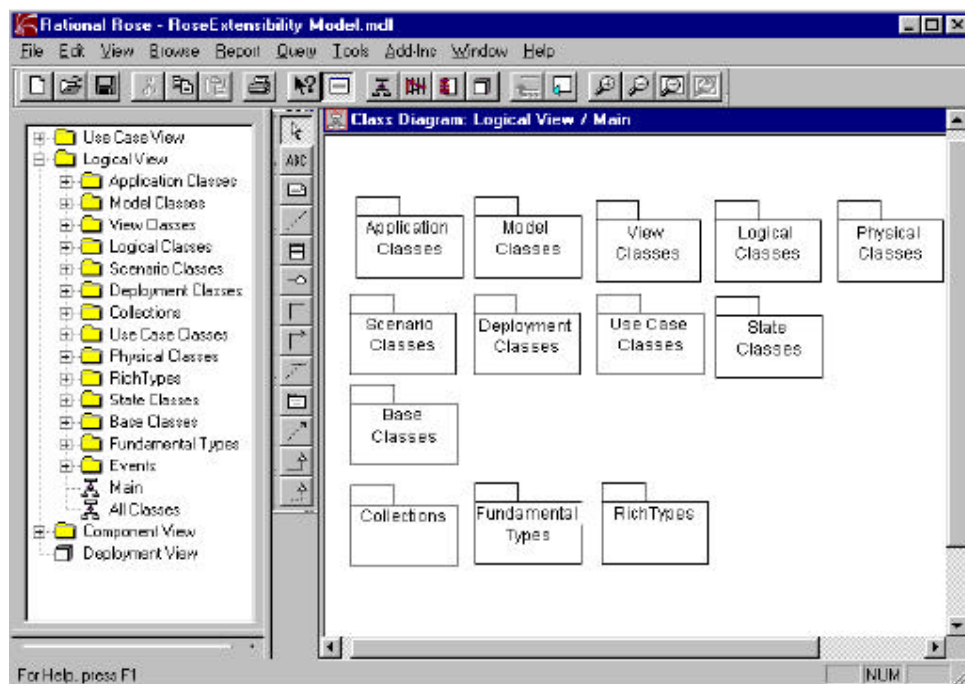


Figura 4.8 – Modelo do Rose Extensibility Interface

Os detalhes sobre as classes contidas em cada pacote, propriedades e métodos de cada classe podem ser encontradas em [REI01] e no *Help online* da ferramenta Rational Rose.

4.4 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentadas as ferramentas OME e Rational Rose que são utilizadas na nossa proposta para modelar os requisitos iniciais e finais, respectivamente. O repositório de dados em Telos onde são armazenados os requisitos iniciais em *i** da ferramenta OME e os mecanismos de extensibilidade do Rational Rose que são utilizados pelo protótipo da ferramenta GOOD foram discutidos.

No próximo capítulo apresentaremos o protótipo da ferramenta GOOD que suporta o mapeamento das descrições de modelos organizacionais em *i** (construídos através da ferramenta OME) para diagramas de classes em UML (construídos através da ferramenta Rational Rose).

Capítulo 5

A Ferramenta GOOD

Este capítulo apresenta a ferramenta GOOD (GOAL INTO OBJECT ORIENTED DEVELOPMENT) que suporta o mapeamento das descrições organizacionais em i, construídas através da ferramenta OME (Organization Modeling Environment), em descrições de diagramas de classe da notação UML construídos através do uso da ferramenta Rational Rose.*

5.1 INTRODUÇÃO

O GOOD (Goals into Object Oriented Development) é o protótipo de uma ferramenta que faz o mapeamento das descrições dos requisitos organizacionais modelados em *i**, através do OME [OME02], em diagramas de classe de UML, construídos, através da ferramenta Rational Rose [RAT02]. O mapeamento será feito de acordo com as diretrizes apresentadas na Seção 3.8. A Figura 5.1 ilustra os componentes utilizados para o mapeamento dos requisitos iniciais em *i** para elementos em UML, que são: o repositório de dados da ferramenta OME, a ferramenta GOOD e o Rational Rose. A ferramenta GOOD é responsável pelo mapeamento dos arquivos armazenados no repositório da ferramenta OME que representam modelos organizacionais em *i** para um modelo conceitual do sistema na ferramenta Rational Rose.

A ferramenta GOOD pode ser vista como uma extensão do Rose. Ela foi construída utilizando a linguagem Rose Scripting, descrita na seção 4.3.2.1. que permite o acesso a funções do Rose através da API definida no REI (Rational Rose Extensibility Interface). Depois de instalada, conforme descrito no Apêndice A, ela pode ser invocada através do menu principal do Rose **Tool > GOOD**.



Figura 5.1 - Diagrama de Componentes da Integração das ferramentas

Nas próximas seções, nós descreveremos em detalhe a estrutura e o funcionamento da ferramenta. Na Seção 5.2 explicaremos o funcionamento do componente responsável pela configuração do mapeamento. Na Seção 5.3 é abordado o componente principal da ferramenta GOOD que é responsável pela execução do mapeamento.

5.2 CONFIGURANDO O ARQUIVO DE MAPEAMENTO

A ferramenta GOOD foi construída de maneira a permitir que as regras de mapeamento possam ser customizadas para se adequar ao domínio para o qual ela está sendo utilizada. Para isso foi construído um componente que permite que o usuário da ferramenta defina como ele deseja que cada elemento representado no modelo organizacional em *i** seja mapeado num elemento de UML.

O componente responsável pela configuração do mapeamento (ver Apêndice B) pode ser acessado através do menu do Rose **Tool > GOOD > Configurar Mapeamento**. A Figura 5.2 mostra a tela que é exibida quando o item do menu é selecionado.

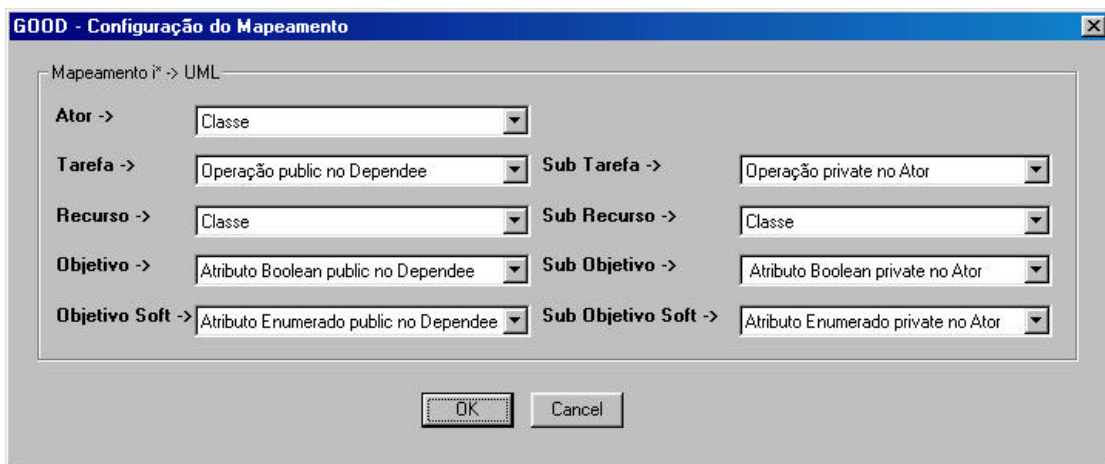


Figura 5.2 Tela de Configuração do Mapeamento

As caixas de lista (*listbox*) apresentam as opções de mapeamento. Os valores que são exibidos na tela mostrada na Figura 5.2 são lidos do arquivo **mapSetting.cfg**. Até que o usuário modifique as opções padrão de mapeamento, elas refletem as regras de mapeamento apresentadas na seção 3.7. Outros valores possíveis para o mapeamento podem ser definidos através das caixas de lista. Os valores possíveis escolhidos são baseados em heurísticas. Contudo, o *script* criado pode ser facilmente modificado para atender a novas regras de mapeamento.

O funcionamento do componente: **Configurar Mapeamento** consiste basicamente na escolha das opções de mapeamento, apresentadas na Figura 5.2, pelo usuário e sua

confirmação através do botão **OK**. A partir daí, as regras do mapeamento definidas são armazenadas no arquivo de configuração **mapSetting.cfg**. Esse arquivo é utilizado durante a inicialização do componente **Executar Mapeamento** que é responsável pelo mapeamento do modelo organizacional em *i** para um modelo conceitual em UML. O componente **Executar Mapeamento** será descrito em detalhe na próxima seção.

Mostrareremos a seguir as opções disponíveis nas caixas de lista apresentadas na Figura 5.2:

Um ator em *i** pode ser mapeado numa classe (ver diretriz D1, Seção 3.8) ou num ator em UML. Em particular, no Rose um ator UML é representado como um estereótipo de uma classe (<<Actor>>). A Figura 5.3 ilustra os possíveis mapeamentos utilizados pela ferramenta GOOD.

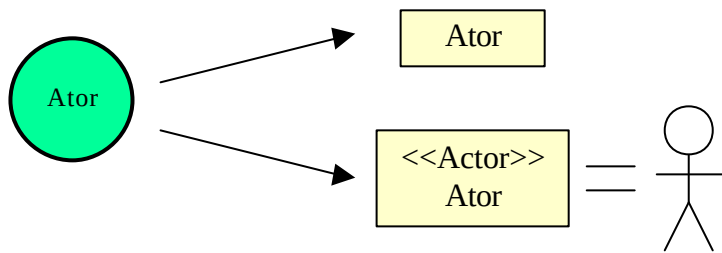


Figura 5.3 Representação Gráfica do Mapeamento de Ator *i**

Uma tarefa em *i** pode ser mapeada como uma operação (ver Diretriz D2.1, Seção 3.8) sobre a classe que é responsável por realizar aquela tarefa em *i**, ou seja, a classe que representa o *dependee*. Esta operação é definida com a propriedade de visibilidade como *public* para indicar que esta tarefa é visível para o ator em *i** (o *dependor*) que está esperando que a tarefa seja realizada. É dada também a opção do usuário mapear a tarefa em *i** como um estereótipo de uma classe (<<Task>>) ou como um caso de uso. Quando a tarefa é mapeada para uma operação, é criada uma associação (caso ela já não exista) entre a classe que representa o *dependor* e a classe que representa o *dependee*. Quando a tarefa é mapeada para um estereótipo de uma classe são criadas associações tanto entre o elemento que representa o *dependor* em UML e a classe estereotipada como entre o *dependee* e a classe estereotipada. Quando a tarefa é mapeada para um caso de uso é criada uma associação entre o caso de uso e o ator em UML que

representa o *dependee*. A Figura 5.4 mostra os mapeamentos utilizados pela ferramenta GOOD para uma tarefa.

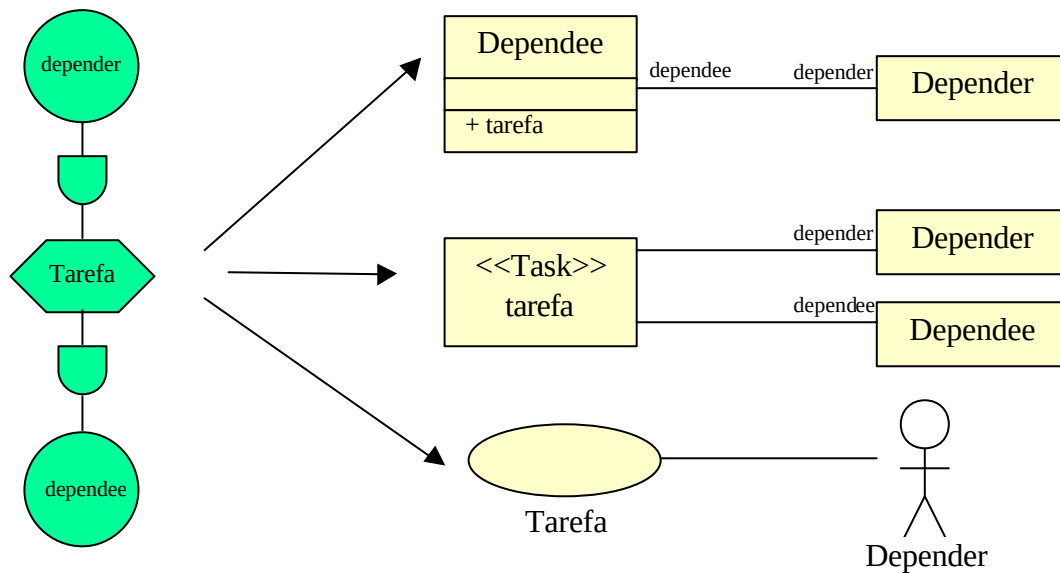


Figura 5.4 Representação Gráfica do Mapeamento de uma Tarefa

Um recurso em *i** pode ser mapeado para uma classe (ver diretriz D3.1, Seção 3.8), ou para um estereótipo de uma classe (<<Resource>>), ou para um atributo público do tipo booleano (para indicar a disponibilidade do recurso) na classe que representa o *dependee*. Quando o recurso é mapeado para uma classe ou para um estereótipo de uma classe são criadas associações entre a classe que representa o recurso e as classes que representam o *dependee* e o *dependee*. Quando o recurso é mapeado para um atributo na classe que representa o *dependee*, é criada uma associação (caso ele não já exista) entre esta classe que representa o *dependee* e a classe que representa o *dependee*. A Figura 5.5 mostra os mapeamentos possíveis usados pela ferramenta GOOD para um recurso em *i**.

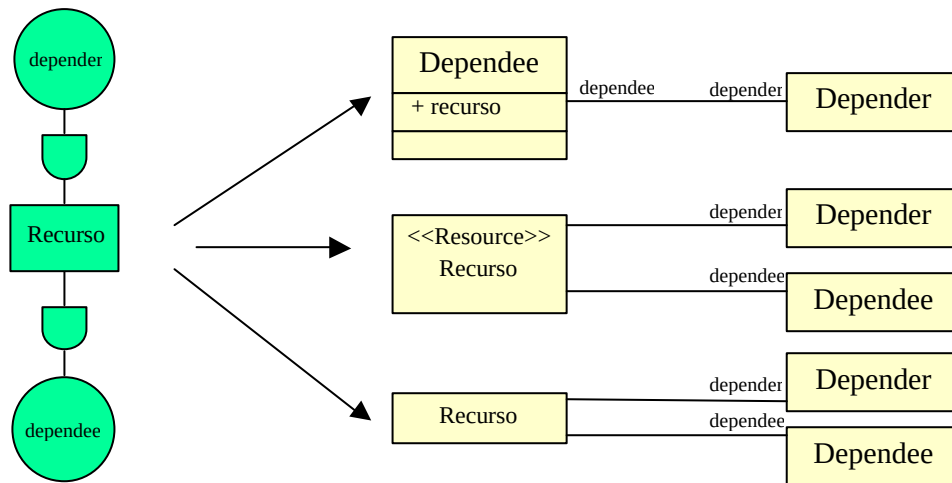


Figura 5.5 Representação Gráfica do Mapeamento de um Recurso

Um objetivo pode ser mapeado para uma classe, ou para um estereótipo de uma classe (<<Goal>>), ou para um atributo público do tipo booleano (para indicar a satisfação do objetivo) na classe que representa o *dependee* (ver diretriz D4.1, Seção 3.8). Quando o objetivo é mapeado para uma classe ou para um estereótipo de uma classe são criadas associações entre a classe que representa o objetivo e as classes que representam o *dependee* e o *depender*. Quando o objetivo é mapeado para um atributo na classe que representa o *dependee*, é criada uma associação (caso ele não já exista) entre esta classe e a classe que representa o *depender*. A Figura 5.6 mostra os mapeamentos usados pela ferramenta GOOD para um objetivo em *i**.

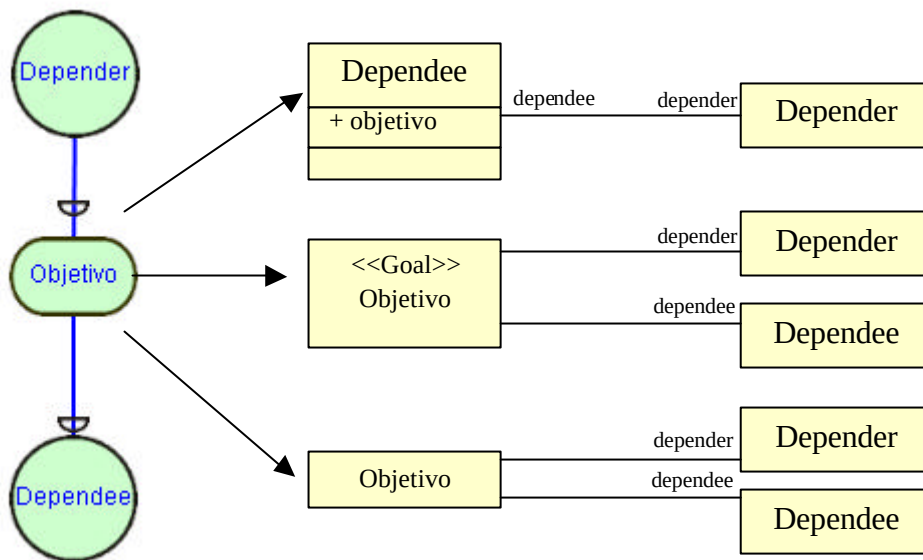


Figura 5.6 Representação Gráfica do Mapeamento de um Objetivo

Um objetivo soft pode ser mapeado para uma classe, ou para um estereótipo de uma classe (<<SoftGoal>>), ou para um atributo público do tipo enumerado (para indicar o grau de satisfação do atributo) na classe que representa o *dependee* (ver diretriz D5.1, Seção 3.8). Quando o objetivo é mapeado para uma classe ou para um estereótipo de uma classe são criadas associações entre a classe que representa o objetivo soft e as classes que representam o *dependee* e o *dependor*. Quando o objetivo soft é mapeado para um atributo na classe que representa o *dependee*, é criada uma associação (caso ele não já exista) entre esta classe e a classe que representa o *dependor*. A Figura 5.7 mostra os mapeamentos usados pela ferramenta GOOD para um objetivo em *i**.

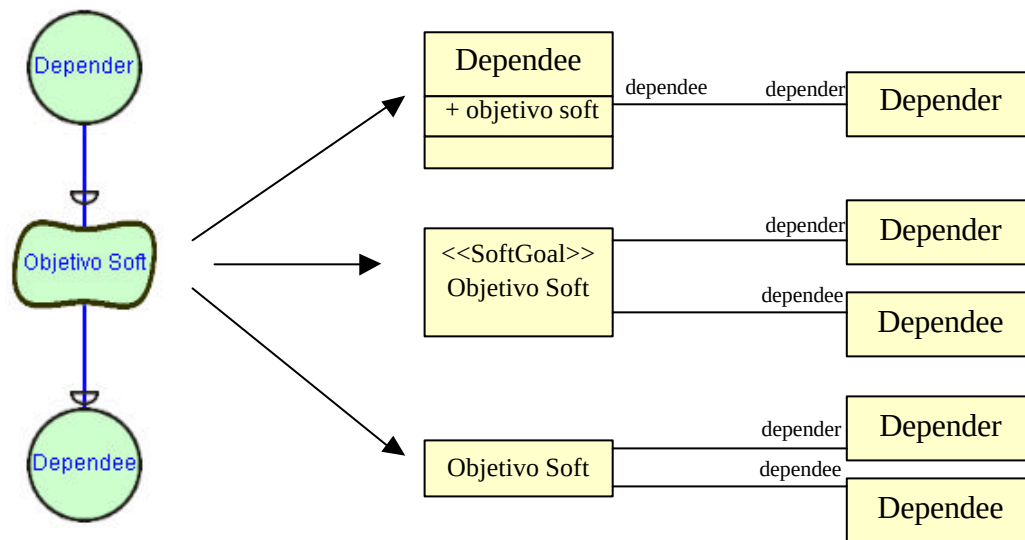


Figura 5.7 Representação Gráfica do Mapeamento de um Objetivo

Nós descrevemos como os atores, tarefas, recursos, objetivos e objetivos soft, representados através do modelo de Dependência Estratégica (SD) podem ser mapeados para elementos do UML através da ferramenta GOOD. A seguir apresentaremos como as tarefas, recursos, objetivos e objetivos soft representados no modelo de Razão Estratégica (SR) podem ser mapeados através da ferramenta GOOD. Para diferenciar os elementos do modelo de Dependência Estratégica dos elementos do modelo de Razão Estratégica iremos chamá-los de sub-tarefas, sub-recursos, sub-objetivos e sub-objetivos soft.

Na ferramenta GOOD o usuário pode escolher mapear sub-tarefas para uma operação privada na classe que corresponde ao ator que a sub-tarefa pertence (ver diretriz 2.2, Seção 3.8) ou como uma classe estereotipada (`<<Task>>`) como ilustrado na Figura 5.8.

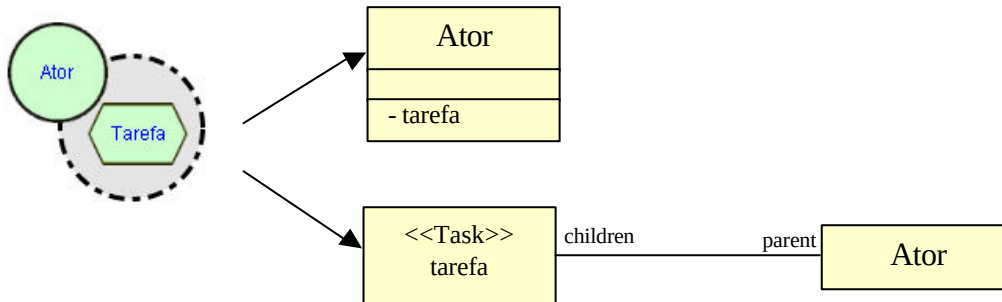


Figura 5.8 Representação Gráfica do Mapeamento de uma sub-Tarefa

Caso existam relacionamentos de dependência que estejam associados à sub-tarefa, no mapeamento é criada uma associação entre a classe que representa o Ator que possui a sub-tarefa e o elemento sobre o qual existe a dependência (*dependum*). A Figura 5.9 mostra um exemplo de uma ligação de dependência de objetivo associada com uma sub-tarefa.

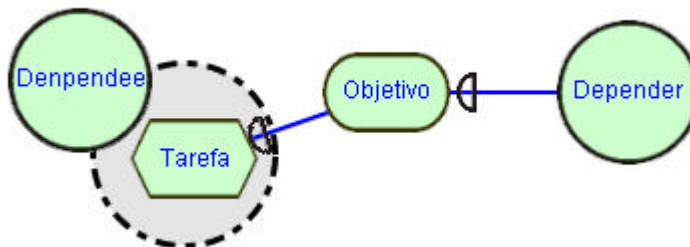


Figura 5.9 Exemplo de um modelo *i** para uma sub-Tarefa

Quando o *dependum* é representado como um elemento do *dependee* (atributo para objetivo, objetivo soft e recurso e operação para tarefa) a associação é criada com a classe que representa o *dependee* (ator que depende do *dependum*), caso ela já não exista. Por exemplo, a Figura 5.9 seria mapeada conforme a Figura 5.10.

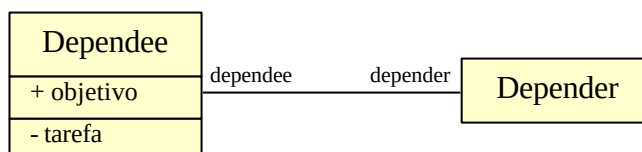


Figura 5.10 Mapeamento de uma sub-Tarefa quando um Objetivo é mapeado como atributo

Quando o *dependum* é representado como uma classe (ou estereótipo de uma classe) é criada uma associação entre a classe que representa o *dependum* e a classe que possui a sub-tarefa. Por exemplo, a Figura 5.9 seria mapeada conforme a Figura 5.11.

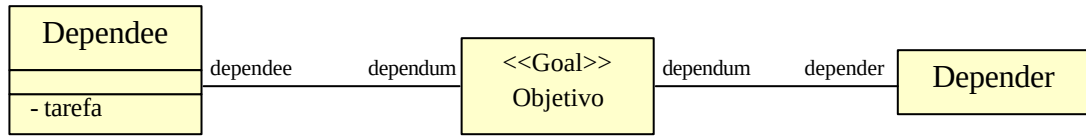


Figura 5.11 Mapeando uma sub-Tarefa quando o Objetivo é um estereótipo de classe

Um sub-recurso pode ser mapeado para um atributo booleano privado (ver diretriz 3.2, Seção 3.8) sobre a classe que corresponde ao ator que o sub-recurso pertence ou como uma classe estereotipada como mostrado na Figura 5.12.

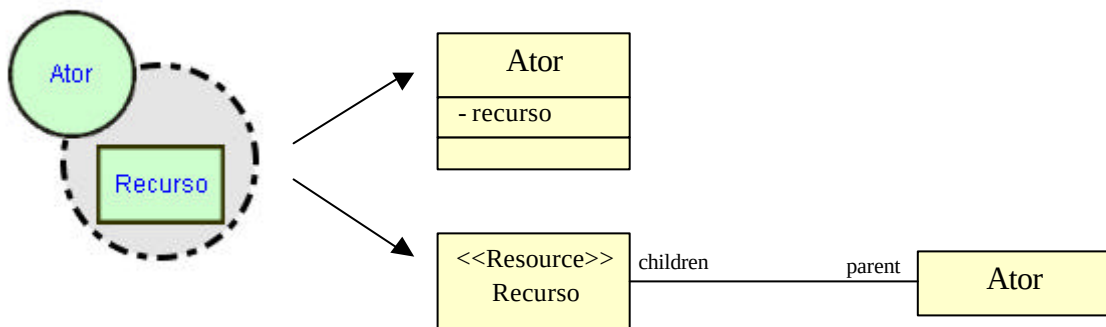


Figura 5.12 Representação Gráfica do Mapeamento de um sub-Recurso

Caso existam relacionamentos de dependência que estejam associados ao sub-recurso, no mapeamento é criada uma associação entre classe que representa o Ator que possui o sub-recurso e o elemento sobre o qual existe a dependência (*dependum*). A Figura 5.13 mostra um exemplo de uma ligação de dependência de objetivo associada com um sub-recurso.

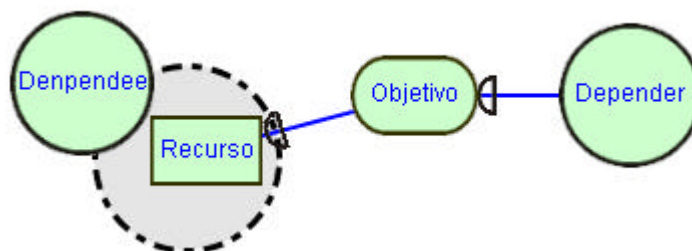


Figura 5.13 Exemplo de um modelo i* para um sub-Recurso

Quando o *dependum* é representado como um elemento do *dependee* (atributo para objetivo, objetivo soft e recurso e operação para tarefa) a associação é criada com a classe que representa o *dependee* (ator que depende do *dependum*), caso ela já não exista. Por exemplo, o modelo em *i** apresentado na Figura 5.13 seria mapeado conforme a Figura 5.14.

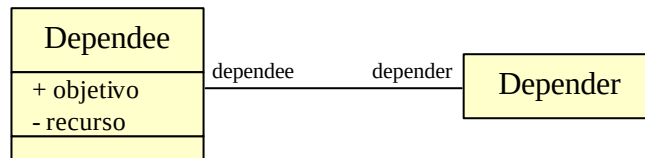


Figura 5.14 Mapeamento de um sub-Recurso quando um Objetivo é mapeado como atributo

Quando o *dependum* é representado como uma classe (ou estereótipo de uma classe) é criada uma associação entre a classe que representa o *dependum* e a classe que possui o sub-recurso. Por exemplo, a Figura 5.13 seria mapeada conforme a Figura 5.15.

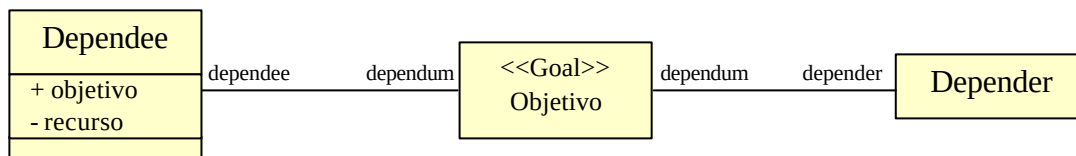


Figura 5.15 Mapeamento de um sub-Recurso quando o Objetivo é um estereótipo de classe

Um sub-objetivo pode ser mapeado para um atributo booleano privado (ver diretriz D4.2, Seção 3.8) sobre a classe que corresponde ao ator que a sub-tarefa pertence ou como uma classe estereotipada como mostrado na figura 5.16.

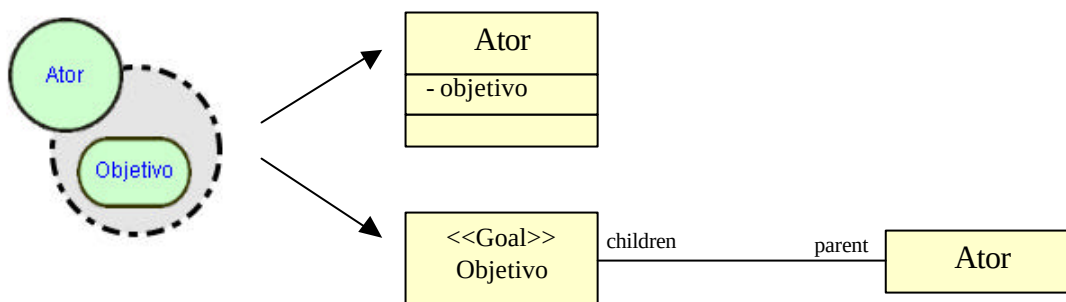


Figura 5.16 Representação Gráfica do Mapeamento de um sub-Recurso

Caso existam relacionamentos de dependência que estejam associados ao sub-objetivo, no mapeamento é criada uma associação entre classe que representa o Ator que possui o sub-objetivo e o elemento sobre o qual existe a dependência (*dependum*). A Figura 5.17 mostra um exemplo de uma ligação de dependência de tarefa associada com um sub-objetivo.

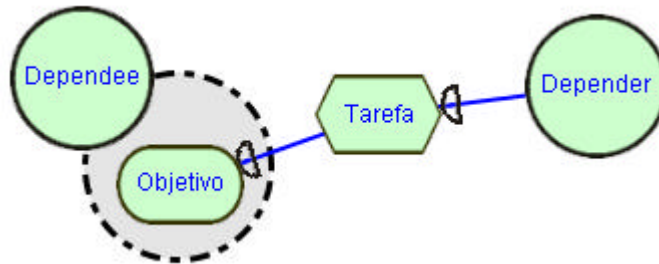


Figura 5.17 Exemplo de um modelo *i** para uma sub-Objetivo

Quando o *dependum* é representado como um elemento do *dependee* (atributo para objetivo, objetivo soft e recurso e operação para tarefa) a associação é criada com a classe que representa o *dependee* (ator que depende do *dependum*), caso ela já não exista. Por exemplo, o modelo em *i** apresentado na Figura 5.17 seria mapeado conforme a Figura 5.18.

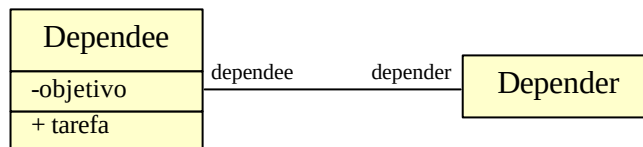


Figura 5.18 Mapeamento de um sub-Objetivo quando uma tarefa é mapeada como operação

Quando o *dependum* é representado como uma classe (ou estereótipo de uma classe) é criada uma associação entre a classe que representa o *dependum* e a classe que possui o sub-objetivo. Por exemplo, a Figura 5.17 seria mapeada conforme a Figura 5.19.

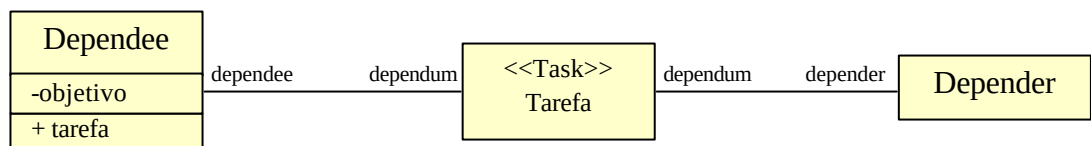


Figura 5.19 Mapeamento de um sub-Objetivo para Tarefa mapeada como estereótipo de classe

Um sub-objetivo soft pode ser mapeado para um atributo enumerado privado (ver diretriz 5.2, Seção 3.8) sobre a classe que corresponde ao ator que o sub-objetivo soft pertence ou como uma classe estereotipada como mostrado na Figura 5.20.

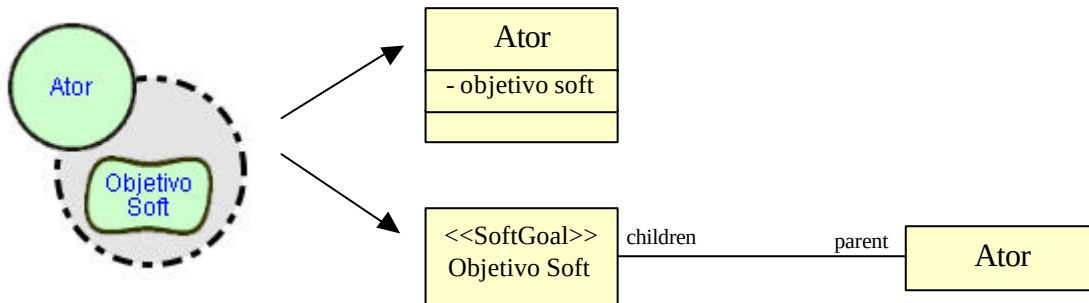


Figura 5.20 Representação Gráfica do Mapeamento de um sub-Objetivo soft

Caso existam relacionamentos de dependência que estejam associados ao sub-objetivo soft, no mapeamento é criada uma associação entre classe que representa o Ator que possui o sub-objetivo soft e o elemento sobre o qual existe a dependência (*dependum*). A Figura 5.21 mostra um exemplo de uma ligação de dependência de tarefa associada com um sub-objetivo soft.

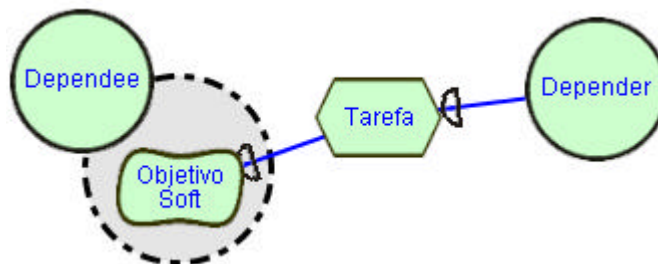


Figura 5.21 Mapeamento de um sub-Objetivo quando uma tarefa é mapeada como operação

Quando o *dependum* é representado como um elemento do *dependee* (atributo para objetivo, objetivo soft e recurso e operação para tarefa) a associação é criada com a classe que representa o *dependee* (ator que depende do *dependum*), caso ela já não exista. Por exemplo, o modelo em *i** apresentado na Figura 5.21 seria mapeado conforme a Figura 5.22.

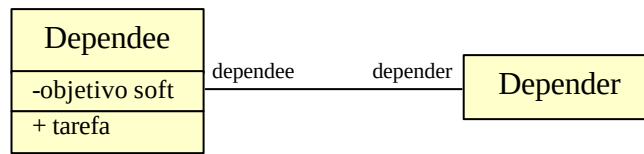


Figura 5.22 Mapeamento de um sub-Objetivo quando uma tarefa é mapeada como operação

Quando o *dependum* é representado como uma classe (ou estereótipo de uma classe) é criada uma associação entre a classe que representa o *dependum* e a classe que possui o sub-objetivo. Por exemplo, a Figura 5.21 seria mapeada conforme a Figura 5.23.

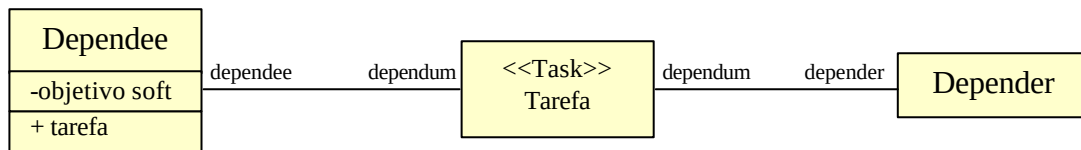


Figura 5.23 Mapeamento de um sub-Objetivo para Tarefa mapeada como estereótipo de classe

Além das opções de mapeamento apresentadas nesta seção, o usuário pode também escolher não mapear um determinado tipo de elemento do *i** para UML, exceto para o tipo ator que é uma entidade central do mapeamento. Quando um tipo de elemento não é mapeado para UML, os relacionamentos que ocorrem com elementos desse tipo não são representados também.

A ferramenta pode ser estendida para permitir que quando o usuário escolha modificar as opções de mapeamento padrão apresentadas na Seção 3.9, seja dada a opção a ele para armazenar a configuração escolhida de forma que ela possa ser reutilizada para um mesmo contexto.

5.3 EXECUTANDO O MAPEAMENTO

O componente responsável pela realização do mapeamento (ver Apêndice C) pode ser acessado através do menu do Rose **Tool > GOOD > Executar Mapeamento**. O diagrama de atividades representado na Figura 5.24 mostra o fluxo de atividades do script que implementa o componente Executar Mapeamento.

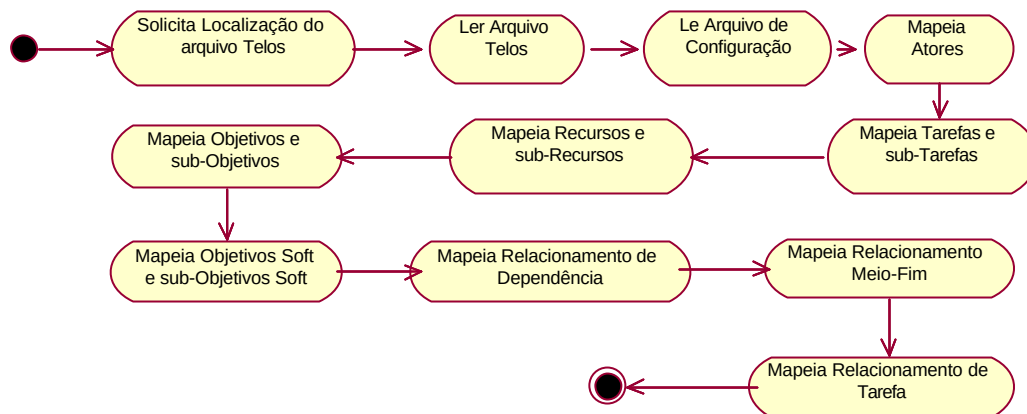


Figura 5.24 Diagrama de Atividades do Componente Executar Mapeamento

A primeira atividade do script é exibir a tela apresentada na Figura 5.25 que solicita que o usuário informe a localização do arquivo Telos que contém o modelo organizacional em *i** desenvolvido através da ferramenta OME.

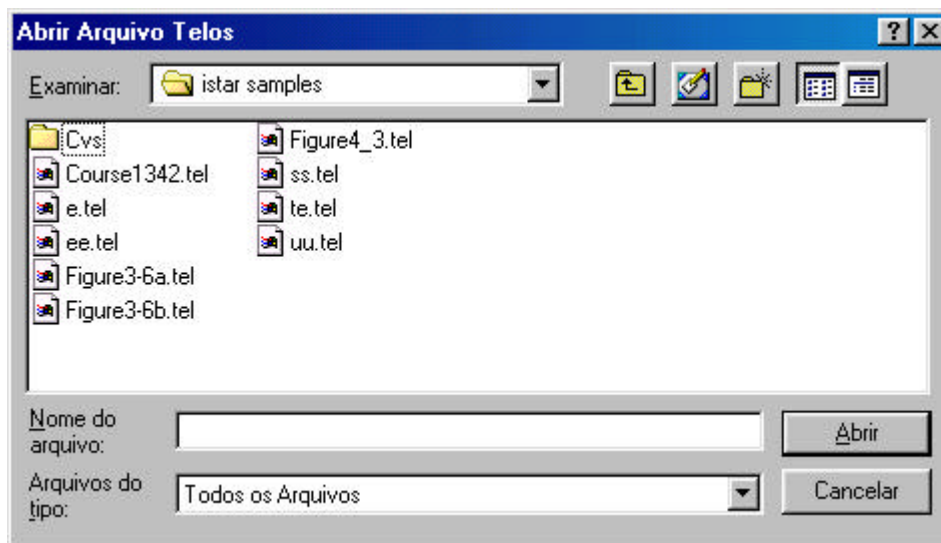


Figura 5.25 – Tela de Localização do Arquivo Telos

A próxima atividade executada pela ferramenta é ler o arquivo Telos e extrair os elementos organizacionais (atores, tarefas, recursos, objetivos e objetivo soft) e os seus relacionamentos (dependência, meio-fim, decomposição de tarefa) definidos e armazená-los no programa.

Num próximo passo o script ler o arquivo de configuração **mapSetting.cfg** que contém as regras de mapeamento explicadas na seção anterior. A partir dessas regras o programa efetua o mapeamento de cada tipo de elemento e depois dos tipos de relacionamentos, e os representa graficamente no Rational Rose.

5.4 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentada a ferramenta GOOD, como também as regras de mapeamento que estão disponíveis atualmente na ferramenta.

No próximo capítulo apresentaremos um estudo de caso que utilizará a ferramenta como mecanismo de suporte para integrar os modelos organizacionais em *i** num modelo conceitual em UML.

Capítulo 6

Estudo de Caso

Este capítulo apresenta o estudo de caso de uma loja de vendas de itens de mídia. Inicialmente, apresentamos a modelagem organizacional da loja e posteriormente mostramos o resultado da utilização da ferramenta GOOD para mapear um modelo organizacional em i para um modelo de classes em UML anotado com restrições em OCL.*

6.1 OBJETIVOS

O principal objetivo desse capítulo é mostrar a utilização da ferramenta GOOD para apoiar o processo de modelagem de requisitos. Para isso foi escolhido o exemplo de um sistema de gerenciamento de comércio eletrônico descrito em [CAS01b]. A primeira fase do desenvolvimento do sistema foi dedicada ao entendimento do problema (Seção 6.2), através do estudo do seu ambiente organizacional cuja saída produziu um modelo organizacional (Seção 6.3) que inclui os atores relevantes e seus respectivos objetivos.

A partir do modelo organizacional desenvolvido foi utilizada a ferramenta GOOD para gerar um modelo conceitual em UML da loja de mídia. Na Seção 6.4 descreveremos em detalhes o processo de mapeamento e por fim faremos as considerações finais na Seção 6.5.

6.2 ESPECIFICAÇÃO DO ESTUDO DE CASO

A *Media Shop* é uma loja que vende e entrega vários tipos diferentes de itens de mídia, tais como: livros, jornais, revistas, cds de áudio, fitas VHS, DVDs e outros. Os clientes (remotos ou locais) da *Media Shop* podem usar um catálogo atualizado regularmente que descreve os itens de mídia disponíveis para especificar o seu pedido. A *Media Shop* é abastecida com os últimos lançamentos, pelos produtores de mídia, e itens já cadastrados, pelos fornecedores de mídia. Para aumentar as vendas, a *Media Shop* decidiu implementar um serviço de vendas pela Internet. Com a nova configuração o cliente pode pedir itens da *Media Shop* pessoalmente, por telefone, ou através da Internet. Este novo sistema foi chamado de *Medi@* e está disponível na web através das facilidades fornecidas pela *Telecom Cpy*. Ele também usa os serviços financeiros fornecidos pelo *Bank Cpy*, que é especializado em transações financeiras on-line. O objetivo básico do novo sistema é permitir que um cliente on-line examine itens no catálogo da Internet através do sistema *Medi@* e possa também fazer pedidos de compra através da Internet.

O *Medi@* está disponível para qualquer cliente potencial que possua acesso à Internet e um navegador web. Não há restrições de registro ou procedimentos de identificação para a navegação no catálogo ou consulta de um item do banco de dados. Mesmo sem nenhuma compra efetivada, um visitante anônimo é considerado um cliente on-line do *Medi@*.

Clientes potenciais podem pesquisar a loja on-line tanto através do catálogo on-line ou consultando um item do banco de dados. O catálogo agrupa itens da mídia do mesmo tipo em (sub)hierarquias e gêneros (ex.: cds de áudio classificados em pop, rock, jazz, opera, musica clássica, trilha sonora, ...) de forma que o cliente possa consultar apenas as (sub)categorias que interessa a ele.

Um mecanismo de pesquisa on-line permite aos clientes com itens particulares em mente pesquisar por título, autor/artista/diretor e campos de descrição através de palavras chave ou de um texto completo. Se o item não estiver disponível no catálogo, o cliente tem a opção de solicitar a *Media Shop* para fazer um pedido do item desejado ao *Fornecedor de Mídia*.

Detalhes sobre os itens da mídia incluem título, categoria da mídia (ex.: livro) e gênero (ex.: ficção científica), autor/artista/diretor, descrição, editora/gravadora, data, custo e algumas figuras (quando disponíveis).

6.3 REQUISITOS INICIAIS EM I*

Os dois principais atores para a aplicação são o cliente e a loja nomeados respectivamente de *Cliente* e *Media Shop*. O cliente tem um objetivo relevante que é *Comprar Itens de Mídia*, enquanto a loja tem o objetivo de *Manter os Clientes Satisfeitos*, e *Aumentar as Vendas*. Desde que os dois últimos objetivos não são bem definidos, eles são representados como objetivos soft conforme mostrado na Figura 6.1.

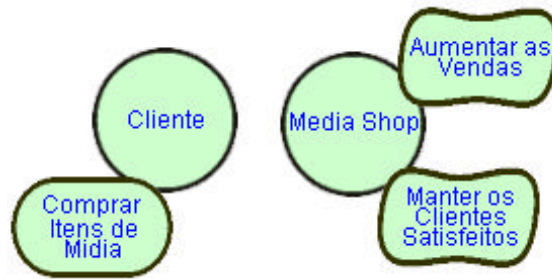


Figura 6.1 Principais Atores e Objetivos

Numa análise mais detalhada do ambiente organizacional podemos identificar os atores: *Fornecedor de Mídia* e *Produtor de Mídia*. A Figura 6.2 ilustra os relacionamentos inicialmente identificados entre os quatros atores. O *Cliente* depende da loja *Media Shop* para ter seu objetivo *Comprar Itens de Mídia* alcançado. Reciprocamente, a loja *Media Shop* depende do *Cliente* para *Aumentar as Vendas* e *Manter os Clientes Satisfeitos*. Como a meta de *Manter os Clientes Satisfeitos* não pode ser definida precisamente, ela é representada como um objetivo soft. O *Cliente* também depende da *Media Shop* para *Consultar Catálogo* (dependência de tarefa). Além disso, a *Media Shop* depende do *Fornecedor de Mídia* para prover itens de mídia de forma contínua (dependência de objetivo: *Fornecimento Contínuo*) e para adquirir um *Item de Mídia* (dependência de recurso). Deseja-se que os itens são esperados que sejam de boa qualidade porque, caso contrário, a dependência *Continuar no Negocio* não será cumprida. Finalmente, *Fornecedor de Mídia* espera que o *Produtor de Mídia* o abasteça com *Pacotes de Qualidade*.

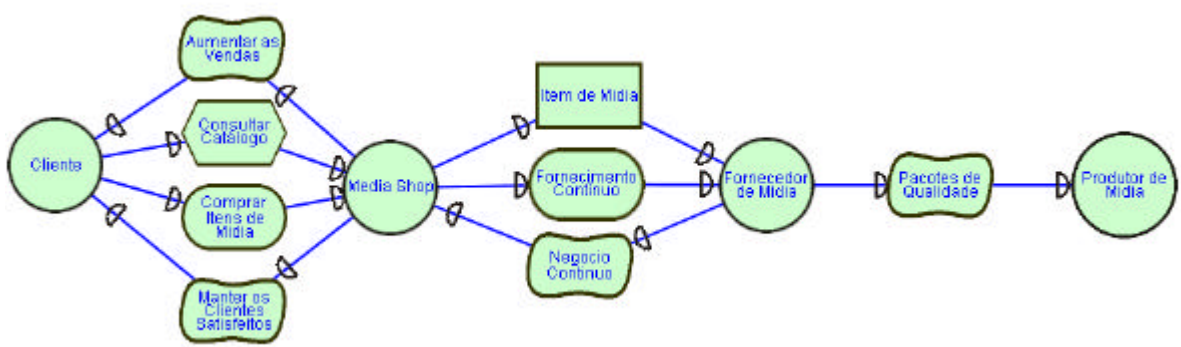


Figura 6.2 Modelo de Dependência Estratégica da loja *Media Shop*

Uma vez que os atores relevantes e seus objetivos foram identificados, o modelo de Razão Estratégica determina através da análise meio-fim como esses objetivos (incluindo objetivos soft) podem realmente ser cumpridos através da contribuição de outros atores. A Figura 6.3 focaliza sobre um das dependências de objetivo soft identificada para *Loja de Mídia*, que foi denominada de *Aumentar Vendas*. Para alcançar esse objetivo soft, a análise feita postula que o objetivo *Gerenciar Loja* pode ser cumprido por meio da tarefa *Dirigir Loja*. As tarefas normalmente são ordenadas em sequências de passos voltados para realizar algum objetivo (soft). Ou seja, as tarefas podem ser decompostas em objetivos (soft) e/ou subtarefas, que coletivamente completam o cumprimento da tarefa. Na Figura 6.3, a tarefa *Dirigir Loja* é decomposta nos objetivos *Controlar Faturamento* e *Tratar Pedidos do Cliente*, na tarefa *Gerenciar Estoque*, e no objetivo soft *Melhorar Serviço* que juntos realizam a tarefa *Dirigir Loja*.

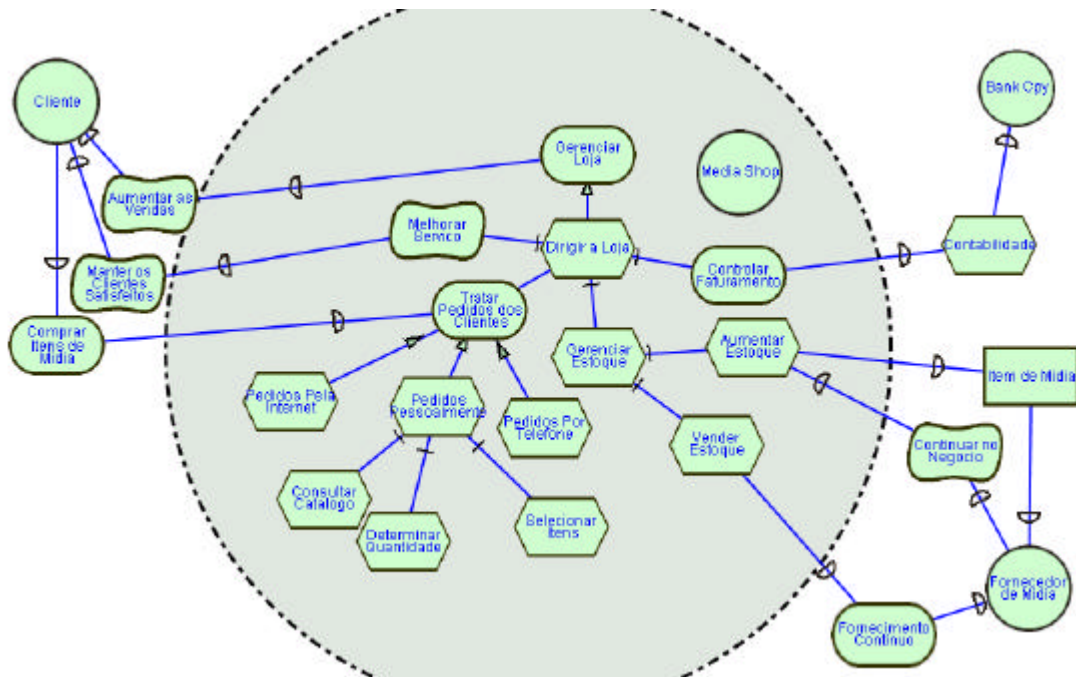


Figura 6.3 Modelo de Razão Estratégica para Media Shop

Os sub-objetivos e as sub-tarefas podem ser especificados mais precisamente através de refinamentos. Por exemplo, o objetivo *Tratar Pedidos dos Clientes* é atendido tanto pelas tarefas *Pedidos Por Telefone*, *Pedidos Pessoalmente* ou *Pedidos Através da Internet*, enquanto a tarefa *Gerenciar Estoque* deve ser coletivamente realizada pelas tarefas *Vender Estoque* e *Aumentar Estoque*. Essa decomposição nos permite eventualmente identificar atores que podem realizar um objetivo, executar uma tarefa,

ou entregar algum recurso necessário para *Loja de Mídia*. Tais dependências na Figura 6.3 são, entre outras, as dependências de objetivo e de recurso sobre o *Fornecedor de Mídia* para o fornecimento, de uma forma contínua de itens de mídia para aumentar o catálogo e vender produtos e as dependências de objetivo soft sobre o *Cliente* para Aumentar Vendas (através da operação da loja) e *Manter os Clientes Satisfeitos* (através da melhoria do serviço), e a dependência de tarefa *Contabilidade* sobre o banco *Bank Cpy* para manter o rastreamento das transações financeiras.

Num próximo passo da análise da organização podemos incluir o ator *Medi@* que representa o sistema de informação que vai implementar o sistema de vendas pela Internet. Por exemplo, na Figura 6.4 o ator *Medi@* é adicionado ao Modelo de Dependência Estratégica.

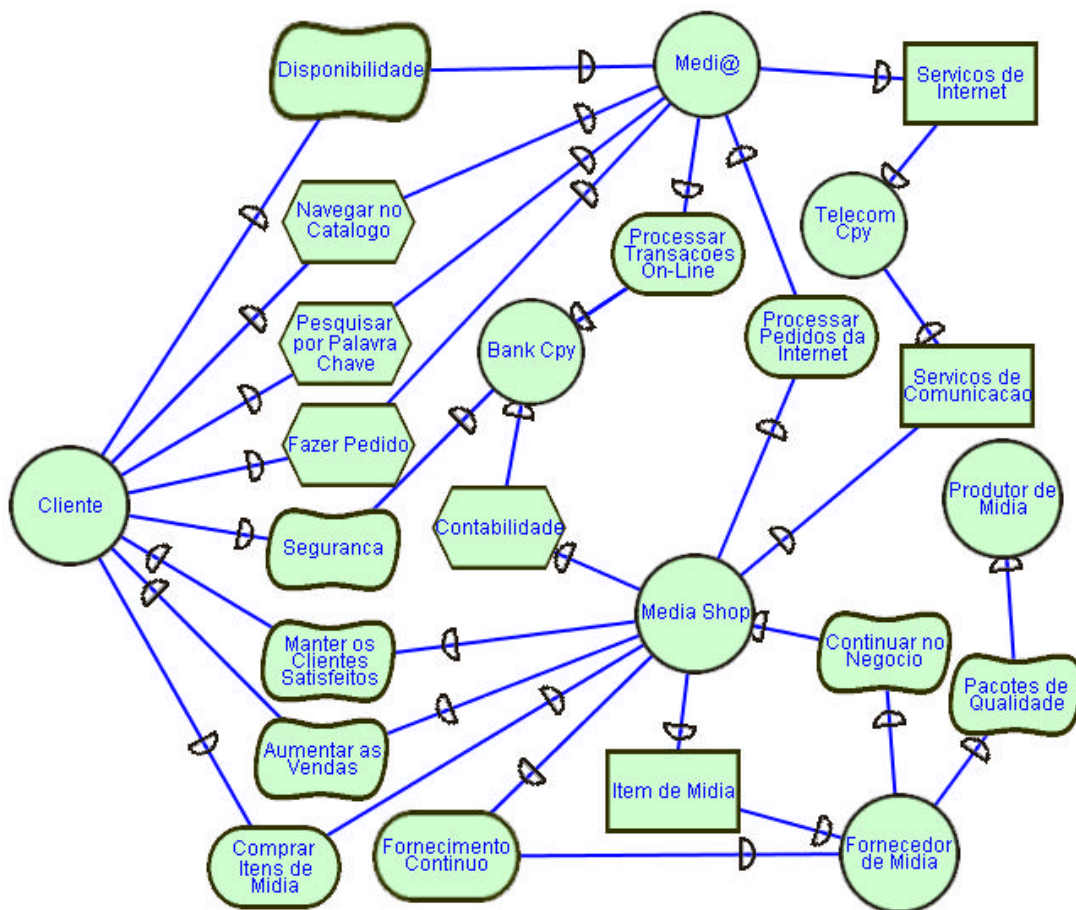


Figura 6.4 Modelo de Dependência Estratégica da Media Shop com o ator Medi@

A loja *Media Shop* depende do sistema *Medi@* para processar *Pedidos Pela Internet*. O *Cliente*, por sua vez, depende do *Medi@* para *Fazer Pedidos* através da Internet, pesquisar o banco de dados por palavras chaves (*Pesquisar Por Palavras Chaves*), ou simplesmente *Navegar no Catálogo* on-line. Com respeito a qualidade do serviço, o *Cliente* requer que os serviços de transação sejam seguros e fáceis de usar. Finalmente, o *Medi@* confia nos serviços de internet fornecidos pela empresa *Telecom Cpy* e em transações financeiras on-line seguras controladas pelo banco *Bank Cpy*.

Apesar do Modelo de Dependência Estratégica fornecer dicas sobre porque os processos são estruturados de uma certa maneira, ele não suporta suficientemente ferramentas para sugerir, explorar e avaliar soluções alternativas. À medida que processo de análise prossegue, são descobertas responsabilidades adicionais para o *Medi@*, conforme representado na Figura 6.5.

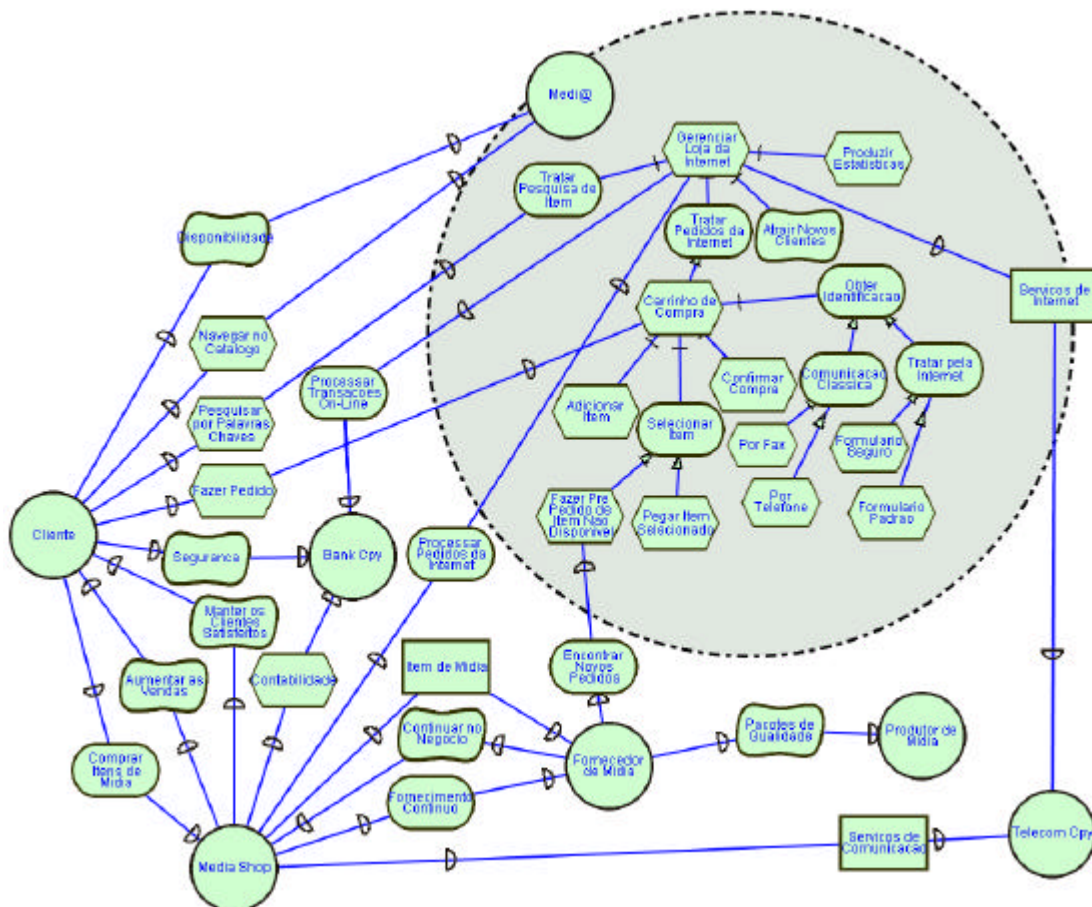


Figura 6.5 Modelo de Razão Estratégica para o ator Medi@

A figura 6.5 mostra a sub tarefa *Gerenciar Loja da Internet* que é inicialmente refinada para objetivos *Tratar Pedidos da Internet* e *Tratar Pesquisa de Item*, objetivos *Atrair Novos Cliente* e na tarefas *Produzir Estatística*. Para gerenciar pedidos da Internet, o objetivo *Tratar Pedidos da Internet* é alcançado através da tarefa *Carrinho de Compras* que é decomposto em sub-tarefas *Selecionar Item*, *Adicionar Item* e *Confirmar Compra*. Essas são as atividades do processo principal que são necessárias para projetar a operacionalização de uma compra on-line [CON00]. O objetivo *Obter Identificação* é alcançado tanto através do sub-objetivo tratar *Comunicação Clássica* através de pedidos por telefone e fax ou *Tratrar* através de formulário seguros ou padrões. Para permitir o pedido de novos itens não listado no catálogo, *Selecionar Item* é refinado em duas sub-tarefas alternativas, uma dedicada a selecionar itens catalogados e a outra para fazer o pré-pedido de produtos indisponíveis.

O estudo de caso não seguiu nenhum padrão de estilo para a nomeação dos elementos do *i** (ator, tarefa, recurso, objetivo e objetivo soft). Contudo a critério do usuário poderia ser adotado um padrão, por exemplo, usar substantivos para atores e recursos e utilizar verbos no infinitivo para nomear tarefas. Além disso não foram acentuados os nomes, pois a ferramenta OME não suporta acentuação.

6.4 MAPEAMENTO PARA UML

Nesta seção apresentaremos o resultado do mapeamento do modelo organizacional da loja Medi@ utilizando a ferramenta GOOD. A ferramenta foi configurada para utilizar as regras de mapeamento definidas na Seção 3.8 da dissertação. Num primeiro mostraremos passo a passo a aplicação das regras de mapeamento para o estudo de caso da loja Medi@. No final exibiremos através da Figura 6.4 o resultado do mapeamento gerado pela ferramenta GOOD no Rational Rose.

Pela diretriz D1, temos que os atores em *i**: *Cliente*, *Media Shop*, *Fornecedor de Mídia*, *Produtor de Mídia*, *Bank Cpy*, *Telecom Cpy* e *Medi@*, são mapeados em classes com seus respectivos nomes, como ilustrado na Figura 6.6.

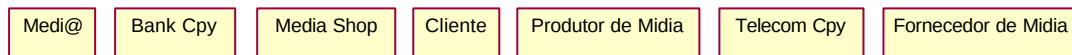


Figura 6.6 Atores da loja de música mapeados em classes

Pela diretriz D2.1, temos que as dependências de tarefas: *Navegar no Catalogo*, *Pesquisar por Palavras Chaves*, *Fazer Pedido*, *Contabilidade*, conforme ilustrado na Figura 6.5, são mapeadas nas operações sobre as classes, *Medi@* e *Bank Cpy* que representam os respectivos *dependee* (Figura 6.7). São criadas também associações entre as classes que representa o *dependee* e o *dependor*.

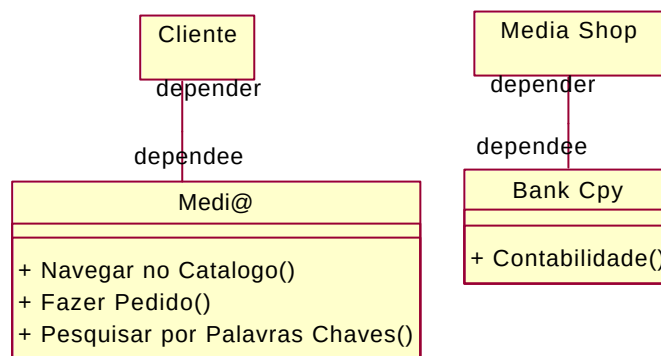


Figura 6.7 Mapeamento das Dependências de Tarefas

Pela diretriz D2.2, temos que as sub-tarefas do ator Media Shop (ver Figura 6.3): *Vender Estoque*, *Aumentar Estoque*, *Gerenciar Estoque*, *Selecionar Itens*, *Determinar Quantidade*, *Consultar Catalogo*, *Pedidos Pessoalmente*, *Pedidos Por Telefone*, *Pedidos Pela Internet* e *Dirigir Loja*, são mapeadas para operações privadas sobre a classe que representa o ator *Media Shop* (ver Figura 6.8). As sub-tarefas do ator *Medi@* (ver Figura 6.5): *Formulário Padrão*, *Formulário Seguro*, *Por Telefone*, *Por Fax*, *Pegar Item Selecionado*, *Confirmar Compra*, *Fazer Pré Pedido de Item Nao Disponível*, *Adicionar Item*, *Carrinho de Compra*, *Produzir Estatísticas* e *Gerenciar Loja da Internet* são representadas como operações privadas sobre a classe que representa o ator *Medi@* (ver Figura 6.8).

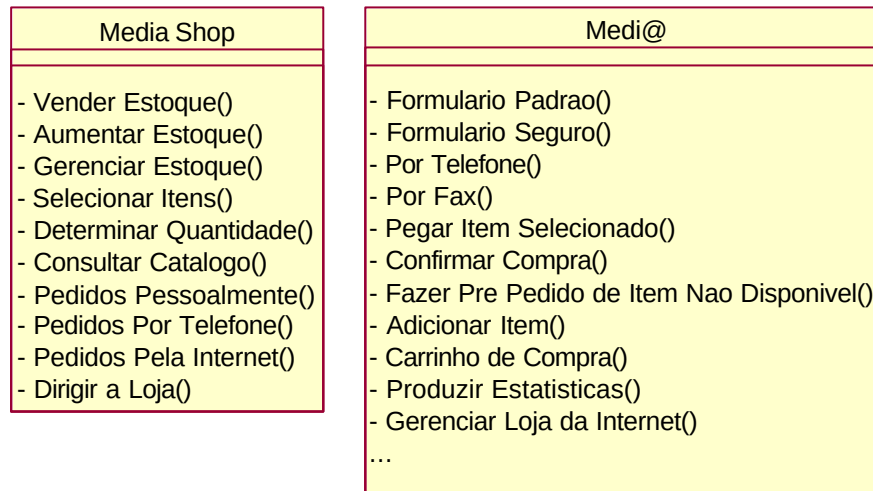


Figura 6.8 Mapeamento das sub Tarefas

Pela diretriz D3.1, temos que os recursos mostrados na Figura 6.5: *Item de Midia*, *Serviços de Comunicação*, *Serviços de Internet* são mapeados para classes em UML com seus respectivos nomes. São criadas também associações entre as classes que representa o recurso e a classe que representa o ator que depende do recurso (*dependor*) e entre a classe que representa o recurso e o ator responsável por fornecer o recurso (*dependee*), conforme mostrados na Figura 6.9.

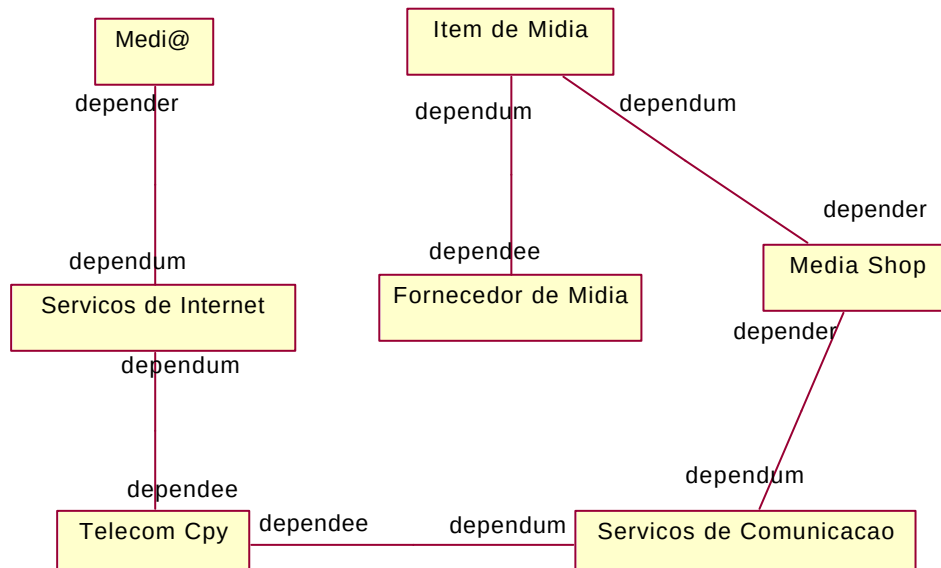


Figura 6.9 Mapeamento dos Recursos

Pela diretriz D4.1, temos que os objetivos: *Processar Pedidos da Internet*, *Encontrar Novos Pedidos*, *Fornecimento Continuo*, *Processar Transacoes On-Line*, *Comprar Itens de Mídia* (ver Figura 6.5) são mapeadas para atributos sobre as classes que são responsáveis por satisfazer o objetivo (*dependee*). No nosso estudo de caso são as classes: *Media Shop*, *Fornecedor Midia*, *Medi@*. São criadas também associações entre as classes que representam o *dependee* e o *dependor* (ver Figura 6.10).

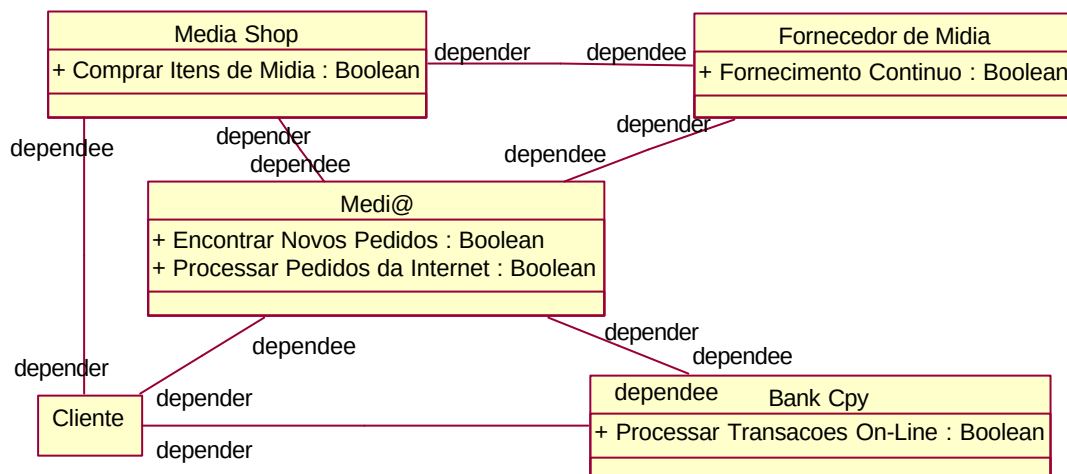


Figura 6.10 Mapeamento dos Objetivos

Pela diretriz D4.2, temos que os sub objetivos do ator *Media Shop* (ver Figura 6.3): *Gerenciar Loja*, *Controlar Faturamento*, *Tratar Pedidos dos Clientes* (ver Figura 6.11) são mapeadas para atributos privados do tipo Boolean sobre a classe Media Shop. Os sub-objetivos do ator *Medi@* (ver Figura 6.5): *Tratar Pesquisa de Item*, *Tratar Pedidos da Internet*, *Obter Identificação*, *Selecionar Item*, *Comunicação Clássica*, *Tratar pela Internet* são representadas como atributos privados do tipo Boolean sobre a classe *Medi@* (ver Figura 6.11).

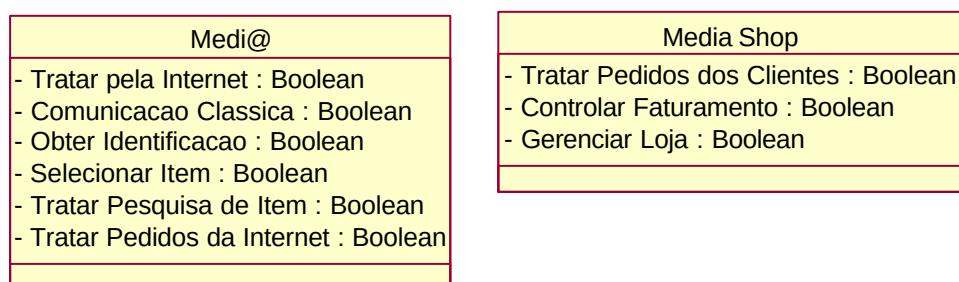


Figura 6.11 Mapeamento dos sub Objetivos

Pela diretriz D5.1, temos que os objetivos soft: *Disponibilidade*, *Segurança*, *Aumentar Vendas*, *Continuar no Negocio*, *Pacotes de Qualidade* (ver Figura 6.5) são mapeadas para atributos do tipo enumerado sobre as classes que são responsáveis por satisfazer o objetivo (*dependee*). No nosso estudo de caso são as classes: *Cliente*, *Bank Cpy*, *Medi@*, *Media Shop*, *Produtor de Mídia*. São criadas também associações entre as classes que representam o *dependee* e o *depender*.

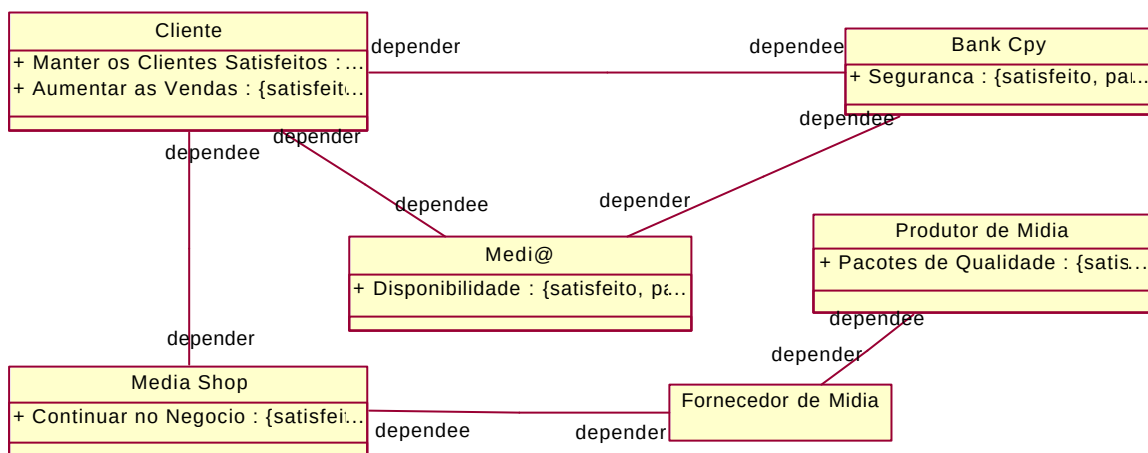


Figura 6.12 Mapeamento dos Objetivos Soft

Pela diretriz D5.2, temos o sub objetivo soft do ator *Media Shop* (ver Figura 6.3): *Melhorar Serviço* é mapeado para um atributo privado do tipo enumerado na classe *Media Shop* (ver Figura 6.13). O sub-objetivo soft do ator *Medi@* (ver Figura 6.5): *Atrair Novos Clientes* é representado como atributo privado do tipo Boolean sobre a classe *Medi@* (ver Figura 6.13).

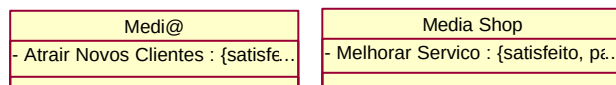


Figura 6.13 Mapeamento de sub Objetivos Soft

Na Figura 6.14 é apresentado o resultado do mapeamento como um tdo que foi realizado pela ferramenta GOOD sobre o modelo organizacional em *i** da loja *Media Shop*.

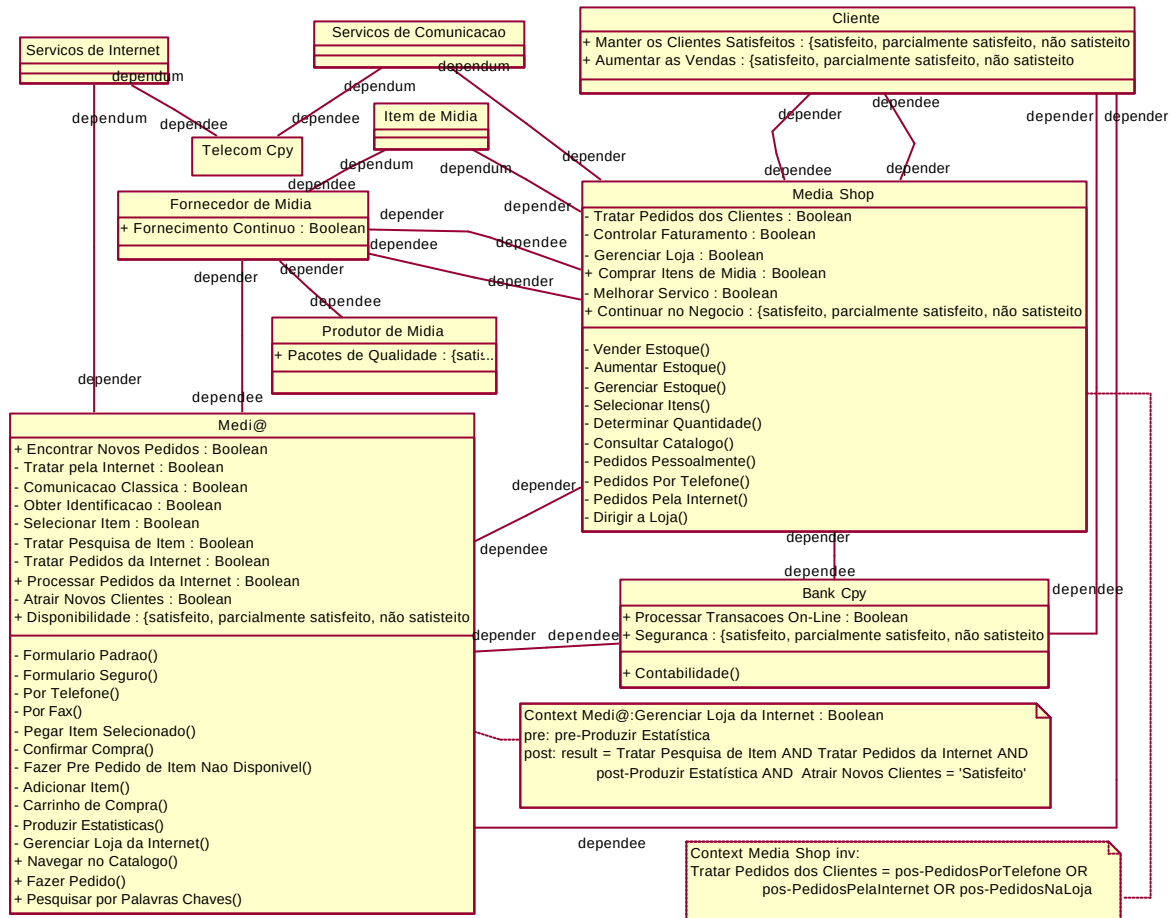


Figura 6.14 Mapeamento do modelo organizacional da loja Media Shop

A Figura 6.14 representa um modelo inicial do domínio do sistema Medi@ que corresponde apenas a fase inicial da modelagem de negócio. Esse modelo deve ser usado quando do desenvolvimento dos modelos de caso de uso (use case) e dos modelos de análise (diagramas de classes e diagramas de interação).

As diretrizes 6 e 7 apresentadas na Seção 3.8 não são suportadas pela ferramenta, mas para efeito de ilustração elas foram incluídas no diagrama mostrado na Figura 6.14.

A diretriz 6 está relacionada com o mapeamento do relacionamento de decomposição de tarefa em *i**. Na Figura 6.5, temos a decomposição da tarefa *Gerenciar Loja da Internet* nos objetivos *Tratar Pesquisa de Item* e *Tratar Pedidos da Internet*, na tarefa *Produzir*

Atrair Novos Clientes = 'Satisfeito'

pos -PedidosNaLoja

Capítulo 7

Conclusões e Trabalhos Futuros

Este capítulo mostra as principais contribuições do trabalho para a área de Engenharia de Requisitos, apresenta alguns trabalhos relacionados e descreve trabalhos futuros que poderão ser realizados a partir dessa dissertação.

7.1 CONTRIBUIÇÕES DO TRABALHO

Neste trabalho iniciamos apresentando os principais conceitos da Engenharia de Requisitos, a motivação para seu estudo, como também as etapas do processo da Engenharia de Requisitos. Nós descrevemos as formas como os requisitos podem ser classificados e a necessidade de se distinguir os requisitos iniciais (*early requirements*) dos requisitos finais (*late requirements*). As atividades da fase da captura de requisitos iniciais são tipicamente informais e tratam de questões organizacionais e não funcionais. A ênfase durante esta fase está no entendimento das motivações e das razões que estão associadas aos requisitos do sistema. As atividades da captura de requisitos finais, por outro lado, focam na completude, consistência e verificação dos requisitos.

Os requisitos iniciais equivalem à definição do contexto e uma pesquisa entre as alternativas dentro daquele espaço. Os requisitos finais equivalem a refinar, eliminar ambigüidades e completar a descrição da alternativa escolhida. A análise estruturada e a análise orientada a objeto são boas para tratar os requisitos finais. Contudo, a análise orientada a metas é mais apropriada para análise dos requisitos iniciais porque ela foca na definição e exploração do espaço de alternativas.

Desta forma, nós argumentamos que UML sozinha não é adequada para tratar dos requisitos iniciais (*early requirements*) e finais (*late requirements*). Em vez disso, nós advogamos o uso de duas técnicas complementares de modelagem, o *i** e o UML anotado com restrições em OCL.

Para modelar e entender as questões da aplicação e do domínio do negócio (a empresa), um desenvolvedor pode usar o *framework i** que enfoca sobre os relacionamentos sociais entre atores e as suas intenções. Para a captura dos requisitos finais, nós propomos o uso do UML anotado com restrições em OCL.

Acreditamos que cada uma das linguagens tem seus próprios méritos para suportar a captura de requisitos. Porém, como técnicas diferentes estão sendo usadas, então uma questão fundamental é o desenvolvimento de uma abordagem que permita a partir de descrições organizacionais em *i** se obter descrições em UML. Para isso nós propomos

uma série de diretrizes de mapeamento dos requisitos iniciais descritos em *i** para descrições em UML que podem ser anotadas com OCL.

Contudo, para viabilizar a utilização da proposta de mapeamento apresentada, surge a necessidade do uso de uma ferramenta que suporte o processo de mapeamento. Inicialmente foram selecionadas as ferramentas que foram alvo da integração, o OME (Organization Modeling Environment) para descrição dos requisitos iniciais em *i** e o Rational Rose para descrição dos requisitos finais em UML. A partir da escolha das ferramentas foi feito um estudo das características de cada ferramenta, como apresentado no Capítulo 4.

Como resultado dessa dissertação tivemos o desenvolvimento do protótipo da ferramenta GOOD (Goal Into Object Oriented Development) que suporta o mapeamento dos elementos do modelo organizacional em *i** armazenados num repositório de dados em Telos pela ferramenta OME para elementos em UML através da ferramenta Rational Rose. A ferramenta GOOD consiste de dois componentes: Configurar Mapeamento e Executar Mapeamento. O componente Configurar Mapeamento permite que sejam definidas as regras de mapeamento que são aplicadas pela ferramenta. O objetivo desse componente é possibilitar que outras regras de mapeamento sejam aplicadas, além das regras sugeridas no Capítulo 3, de acordo com a necessidade do usuário da ferramenta. O componente Executar Mapeamento captura os elementos do *i** no repositório Telos e através da interface REI (Rose Extensibility Interface) do Rational Rose cria as descrições em UML baseada nas regras selecionadas.

O objetivo do desenvolvimento dessa ferramenta foi contribuir para diminuir o *gap* existente entre a modelagem organizacional e a modelagem orientada a objeto. É esperado a partir dessa integração, que o desenvolvimento de sistemas de informação sejam baseados nos negócios que eles se propõem a suportar. Desta forma, teremos associados às especificações dos requisitos funcionais as razões que os levaram a eles existirem. Portanto, estaremos atacando uma das maiores causas de falha no desenvolvimento de sistemas que é falta de satisfação das necessidades dos usuários finais e da organização que está pagando pelo sistema.

A aplicação da proposta de mapeamento é apresentada através de um exemplo de um estudo de caso de uma loja de venda de itens de mídia, onde é mostrado passo a passo o funcionamento da ferramenta GOOD e o resultado do mapeamento.

A abordagem da modelagem de negócio, apresentada nesta dissertação, permite que os requisitos do sistema sejam derivados a partir das descrições iniciais em i^* . Acreditamos que os sistemas desenvolvidos a partir desses requisitos possam suportar adequadamente as atividades da organização, contribuindo para a construção de sistemas de maior qualidade que atendem a necessidades tanto dos usuários finais, como da organização que está pagando pelo sistema.

7.2 TRABALHOS RELACIONADOS

Existem dois trabalhos que se propoem integrar a técnica i^* que descreve os aspectos organizacionais com a modelagem de sistemas: o trabalho de Santander [SAN00, SAN01] e o projeto Tropos [CAS00c, CAS01b, CAS01c, MYL00].

O trabalho de Santander propõe algumas diretrizes para auxiliar engenheiros de requisitos no desenvolvimento dos casos de uso a partir de modelos organizacionais representados pela técnica i^* . A metodologia Tropos é uma abordagem de desenvolvimento de software orientado a agentes que é centrada a requisitos. Ela propõe uma metodologia de desenvolvimento de software e uma *framework* de desenvolvimento que é fundamentada em conceitos usados para modelagem de requisitos iniciais. Ela adota a *framework* de modelagem i^* e a usa como base para modelar os requisitos iniciais e finais, o projeto de arquitetura, o projeto detalhado e o desenvolvimento em agentes.

7.3 TRABALHOS FUTUROS

Nesta seção, levantamos algumas das possibilidades de trabalhos futuros que podem se baser nas idéias apresentados nesta dissertação.

- **Suporte a XMI.** O XMI (XML Metadata Interchange) [OMG00] é um padrão proposto pela OMG que tem se tornado um padrão de fato para troca de dados entre ferramentas CASE. O objetivo principal do XMI é criar um mecanismo fácil de troca de metadados entre ferramentas de modelagem (baseadas no padrão UML) e repositórios de metadados (baseado no padrão MOF) em ambientes heterogêneos e distribuídos. O XMI integra três padrões: o XML (eXtensible Markup Language) [BRA00], o UML (Unified Modeling Language) [OMG01] e o MOF (Meta Object Facility) [MOF00] que é um padrão de repositório de metadados distribuídos. Poderia ser estudada a possibilidade de se criar uma ferramenta que permitisse o mapeamento dos modelos do *i** para o formato XMI, assim a ferramenta ficaria independente do Rational Rose e ela poderia ser utilizada por outras ferramentas de modelagem de sistema que suportassem o padrão XMI.
- **Suporte a outras Técnicas de modelagem de requisitos iniciais.** A ferramenta poderia ser estendida para permitir que outras linguagens de modelagem de requisitos iniciais pudessem ser suportadas, já que não existe ainda um padrão para modelagem de requisitos iniciais.
- **Suporte a OCL.** Desenvolver uma ferramenta que possa suportar as diretrizes 6 e 7 apresentadas na Seção 3.8 que são aplicadas com o uso do OCL.
- **Aplicação em Outros Estudos de Casos.** Aplicar a *framework* integrada em outros estudos de casos para validar a técnica e possivelmente aperfeiçoá-la.
- **Permitir a Seleção dos Elementos Mapeados.** Como nem todos os conceitos capturados na fase de requisitos iniciais irão corresponder aos modelos de sistema de software, a ferramenta poderia ser estendida para permitir que o usuário selecionasse quais elementos ele deseja que participe do mapeamento. Por exemplo, o usuário poderia escolher quais atores organizacionais ele deseja que participe do processo de mapeamento.

Apêndice A

INSTALANDO A FERRAMENTA GOOD

Neste apêndice é descrito o que é necessário para configurar o Rational Rose para executar a ferramenta GOOD.

O primeiro passo é criar o diretório Good em c:\Arquivos de Programas\ e em seguida copiar os scripts GOOD_Configure.ebx e GOOD_Map.ebx para este diretório.

Em seguida você deve abrir o arquivo Rose.mnu que fica na raiz do diretório de instalação do Rose (o diretório padrão é c:\Arquivos de Programas\Rational\Rose\) e adicionar depois do item de Menu Tools as linhas:

```
Menu "GOOD"
{
    option "Configurar Mapeamento"
    {
        RoseScript $GOOD_PATH\GOOD_Configure.ebx
    }
    option "Executar Mapeamento"
    {
        RoseScript $GOOD_PATH\GOOD_Map.ebx
    }
}
```

Um exemplo do arquivo Rose.mnu com item de menu GOOD adicionado é mostrado a seguir:

```
Menu Help
{
    option "Contacting &Technical Support"
    {
        RoseScript $SCRIPT_PATH\webgorationalsupport.ebx
    }
    Menu "Rational on the &Web"
    {
        Option "&Rational Home Page"
        {
            RoseScript $SCRIPT_PATH\webgorationalhome.ebx
        }
        Option "Rose &Home Page"
        {
            RoseScript $SCRIPT_PATH\webgorational.ebx
        }
        Option "&Technical Support"
        {
            RoseScript $SCRIPT_PATH\webgorationalsupport.ebx
        }
    }
}
Menu Report
{
    option "Show &Participants in UC..."
}
```

```

    {
        enable %selected_items:empty:false
        RoseScript $SCRIPT_PATH\participants.ebx
    }
}
Menu Tools
{
    Separator
    option "Class Wizard..."
    {
        enable %model:writeable
        RoseScript $SCRIPT_PATH\cwizard.ebx
    }
    Separator
    Menu "GOOD"
    {
        option "Configurar Mapeamento"
        {
            RoseScript $GOOD_PATH\GOOD_Configure.ebx
        }
        option "Executar Mapeamento"
        {
            RoseScript $GOOD_PATH\GOOD_Map.ebx
        }
    }
}

```

Outra opção é incluir o caminho absoluto em vez de usar a variável de ambiente \$GOOD_PATH. Definir um caminho virtual, em vez do caminho absoluto, simplifica o processo de edição do arquivo de menu (Rose.mnu).

Por fim, você tem que criar a variável de ambiente \$GOOD_PATH, você terá que adicioná-la da seguinte forma:

1. Inicie o Rational Rose.
2. Clique no menu **File > Edit Path Map** para mostrar a caixa de diálogo **Virtual Path Map**. (veja Figura A1)
3. Insira na entrada **Symbol** o texto: **GOOD_PATH**, na entrada **Actual Path** incluía a localização do diretório good (c:\Arquivos de programas\Good).
4. Em seguida clique no botão **Add**
5. Feche a janela clicando no botão **Close**.

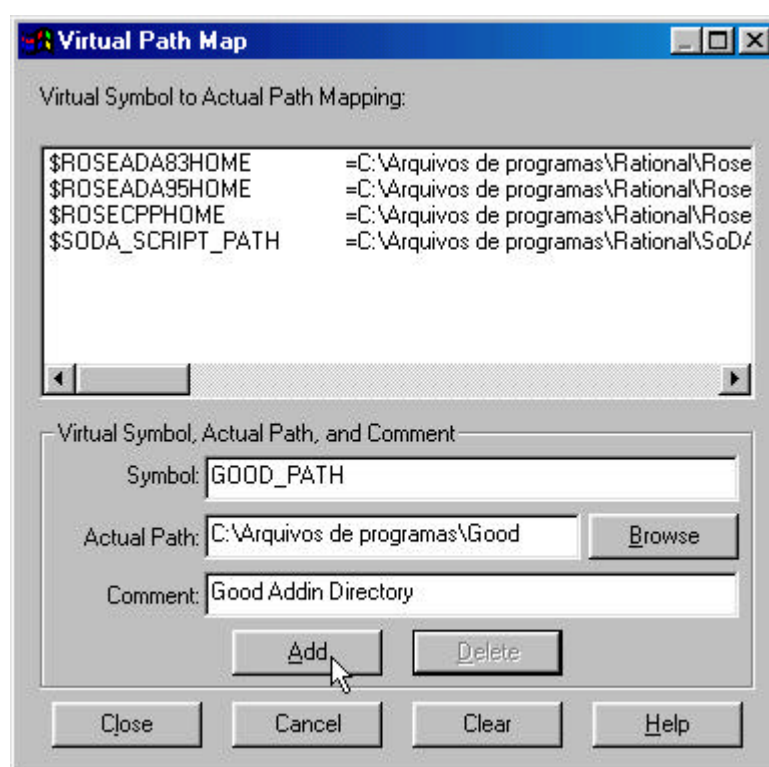


Figura A1. Defindo Caminho Virtual

Apêndice B

CÓDIGO DO *SCRIPT* DO COMPONENTE CONFIGURAR MAPEAMENTO

```
Sub Main()

    Dim DropListBoxActor$(1)
    DropListBoxActor$(0) = "Classe"
    DropListBoxActor$(1) = "<<Ator>>"

    Dim DropListBoxGoal$(3)
    DropListBoxGoal$(0) = ""
    DropListBoxGoal$(1) = "Classe"
    DropListBoxGoal$(2) = "<<Goal>>"
    DropListBoxGoal$(3) = "Atributo Boolean public no Dependee"

    Dim DropListBoxSoftGoal$(3)
    DropListBoxSoftGoal$(0) = ""
    DropListBoxSoftGoal$(1) = "Classe"
    DropListBoxSoftGoal$(2) = "<<SoftGoal>>"
    DropListBoxSoftGoal$(3) = "Atributo Enumerado public no Dependee"

    Dim DropListBoxTask$(3)
    DropListBoxTask$(0) = ""
    DropListBoxTask$(1) = "<<Task>>"
    DropListBoxTask$(2) = "Caso de Uso"
    DropListBoxTask$(3) = "Operação public no Dependee"

    Dim DropListBoxResource$(3)
    DropListBoxResource$(0) = ""
    DropListBoxResource$(1) = "Classe"
    DropListBoxResource$(2) = "<<Resource>>"
    DropListBoxResource$(3) = "Atributo public no Dependee"

    Dim DropListBoxSubGoal$(2)
    DropListBoxSubGoal$(0) = ""
    DropListBoxSubGoal$(1) = "<<Goal>>"
    DropListBoxSubGoal$(2) = "Atributo Boolean private no Ator"

    Dim DropListBoxSubSoftGoal$(2)
    DropListBoxSubSoftGoal$(0) = ""
    DropListBoxSubSoftGoal$(1) = "<<SoftGoal>>"
```

```

DropListBoxSubSoftGoal(2) = "Atributo Enumerado private no Ator"

Dim DropListBoxSubTask$(2)
DropListBoxSubTask(0) = "<<Task>>"
DropListBoxSubTask(1) = "Operação private no Ator"

Dim DropListBoxSubResource$(2)
DropListBoxSubResource$(0) = ""
DropListBoxSubResource$(1) = "<<Resource>>"
DropListBoxSubResource$(2) = "Atributo Boolean Private no Ator"

Begin Dialog ConfigureDialog 21,28,475,170,"GOOD - Configuração do Mapeamento"

    OKButton 178,140,40,14
    CancelButton 225,140,40,14

    Text 20,25,68,8,"Ator ->",<<TextActor>>,ebBold
    DropListBox 85,25,135,40,DropListBoxActor$,DropListBoxActor

    Text 20,45,68,8,"Tarefa ->",<<TextTask>>,ebBold
    DropListBox 85,45,135,40,DropListBoxTask$,DropListBoxTask

    Text 20,65,68,8,"Recurso ->",<<TextResource>>,ebBold
    DropListBox 85,65,135,40,DropListBoxResource$,DropListBoxResource

    Text 20,85,68,8,"Objetivo ->",<<TextGoal>>,ebBold
    DropListBox 85,85,135,40,DropListBoxGoal$,DropListBoxGoal

    Text 20,105,68,8,"Objetivo Soft ->",<<TextSoftGoal>>,ebBold
    DropListBox 85,105,135,40,DropListBoxSoftGoal$,DropListBoxSoftGoal

    Text 230,45,77,8,"Sub Tarefa ->",<<TextSubTask>>,ebBold
    DropListBox 315,45,135,40,DropListBoxSubTask$,DropListBoxSubTask

    Text 230,65,77,8,"Sub Recurso ->",<<TextSubResource>>,ebBold
    DropListBox 315,65,135,40,DropListBoxSubResource$,DropListBoxSubResource

    Text 230,85,77,8,"Sub Objetivo ->",<<TextSubGoal>>,ebBold
    DropListBox 315,85,135,40,DropListBoxSubGoal$,DropListBoxSubGoal

    Text 230,105,77,8,"Sub Objetivo Soft ->",<<TextSubObjetivoSoft>>,ebBold
    DropListBox 315,105,135,40,DropListBoxSubSoftGoal$,DropListBoxSubSoftGoal

    GroupBox 12,8,450,120,"Mapeamento i* -> UML",GroupBox1

End Dialog

```

```
Dim D_Configure As ConfigureDialog
Dim linha As String
Dim var1 As String
Dim var2 As String
Dim var3 As String

filename$ = "c:\Arquivos de Programas\Good\mapSetting.cfg"
Open filename$ For Input As #1
While Not EOF(1)
    Line Input #1, linha$
    var1 = Token(linha, 1)
    var2 = Token(linha, 2)
    var3 = Token(linha, 3)
    If (var1 = "Map") Then
        If (var2 = "Actor") Then
            D_Configure.DropListBoxActor = CInt(var3)
        End If
        If (var2 = "Goal") Then
            varI = CInt(var3)
            D_Configure.DropListBoxGoal = CInt(var3)
        End If
        If (var2 = "SoftGoal") Then
            varI = CInt(var3)
            D_Configure.DropListBoxSoftGoal = CInt(var3)
        End If
        If (var2 = "Task") Then
            varI = CInt(var3)
            D_Configure.DropListBoxTask = CInt(var3)
        End If
        If (var2 = "Resource") Then
            varI = CInt(var3)
            D_Configure.DropListBoxResource = CInt(var3)
        End If
        If (var2 = "SubGoal") Then
            varI = CInt(var3)
            D_Configure.DropListBoxSubGoal = CInt(var3)
        End If
        If (var2 = "SubSoftGoal") Then
            varI = CInt(var3)
            D_Configure.DropListBoxSubSoftGoal = CInt(var3)
        End If
        If (var2 = "SubTask") Then
            varI = CInt(var3)
            D_Configure.DropListBoxSubTask = CInt(var3)
        End If
        If (var2 = "SubResource") Then
            varI = CInt(var3)
            D_Configure.DropListBoxSubResource = CInt(var3)
        End If
    End If
End While
```

```

        End If
    Wend
    Close #1

    button% = Dialog (D_Configure,1)

    MapActor = D_Configure.DropListBoxActor
    MapGoal = D_Configure.DropListBoxGoal
    MapSoftGoal = D_Configure.DropListBoxSoftGoal
    MapBoxTask = D_Configure.DropListBoxTask
    MapResource = D_Configure.DropListBoxResource
    MapSubGoal = D_Configure.DropListBoxSubGoal
    MapSubSoftGoal = D_Configure.DropListBoxSubSoftGoal
    MapSubTask = D_Configure.DropListBoxSubTask
    MapSubResource = D_Configure.DropListBoxSubResource

    If button% = -1 Then SaveFile MapActor, MapGoal, MapSoftGoal, MapBoxTask,
MapResource, MapSubGoal, MapSubSoftGoal, MapSubTask, MapSubResource

End Sub

Sub SaveFile (MapActor, MapGoal, MapSoftGoal, MapBoxTask, MapResource, MapSubGoal,
MapSubSoftGoal, MapSubTask, MapSubResource)
    Open "c:\Arquivos de Programas\Good\mapSetting.cfg" For Output Access Write Lock
Read Write As #1

    Print #1, "Map Actor ";MapActor
    Print #1, "Map Goal ";MapGoal
    Print #1, "Map SoftGoal ";MapSoftGoal
    Print #1, "Map Task ";MapBoxTask
    Print #1, "Map Resource ";MapResource
    Print #1, "Map SubGoal ";MapSubGoal
    Print #1, "Map SubSoftGoal ";MapSubSoftGoal
    Print #1, "Map SubTask ";MapSubTask
    Print #1, "Map SubResource ";MapSubResource
    Close
End Sub

Function Token (lin As String, pos As Integer) As String
    Dim posToken As Integer
    Dim char As String
    Dim length As Integer
    Dim i As Integer
    Dim beginToken As Boolean
    Dim cod As Integer
    Dim tok As String
    i=1
    posToken = 1
    beginToken = False

```

```

length = Len(lin)
While (posToken <= pos And i <= length)
    char = Mid(lin,i,1)
    cod = Asc(char)
    If ((cod = 9) Or (cod = 0) Or (cod = 26) Or (cod = 13) Or (cod = 10) Or
(cod = 32)) Then
        If (beginToken = True)Then
            beginToken = False
            posToken = posToken +1
        End If
    Else
        If (beginToken = False) Then
            beginToken = True
        End If
        If (posToken = pos) Then
            tok = tok & char
        End If
    End If
    i = i+1
Wend
Token = tok
End Function

```

Apêndice C

CÓDIGO DO *SCRIPT* DO COMPONENTE EXECUTAR MAPEAMENTO

```
'-----
'OROSE.ebs
'
'   This script makes the mapping of organizational models in i * for models
'       in UML.
'
'   Created by Gilberto Cysneiros Filho - Centro de Informatica - UFPE
'
'                                   Recife - Brazil - nLength1
'-----

Const nLength = 200 ' Define the length array

' Types definition of i* elements
Type ActorElement
    cod As String
    name_ As String
    children(nLength) As String
    links(nLength) As String
End Type

Type GoalElement
    cod As String
    parent As String
    name_ As String
    links(nLength) As String
End Type

Type TaskElement
    cod As String
    parent As String
    name_ As String
    links(nLength) As String
End Type

Type SoftGoalElement
    cod As String
    parent As String
```

```

    name_ As String
    links(nLength) As String
End Type

```

```

Type ResourceElement
    cod As String
    parent As String
    name_ As String
    links(nLength) As String
End Type

```

```

Type DependencyLink
    cod As String
    name_ As String
    to_ As String
    from As String
End Type

```

```

Type MeansEndsLink
    cod As String
    name_ As String
    to_ As String
    from As String
End Type

```

```

Type DecompositionLink
    cod As String
    name_ As String
    to_ As String
    from As String
End Type

```

```

Dim actors(nLength) As ActorElement
Dim iactor As Integer
Dim goals(nLength) As GoalElement
Dim igoal As Integer
Dim tasks(nLength) As TaskElement
Dim itask As Integer
Dim softgoals(nLength) As GoalElement
Dim isoftgoal As Integer
Dim resources(nLength) As ResourceElement
Dim iresource As Integer
Dim dependencies(nLength) As DependencyLink
Dim idependency As Integer
Dim meansEnds(nLength) As MeansEndsLink
Dim imeansEnd As Integer
Dim decompositions(nLength) As DecompositionLink
Dim idecomposition As Integer

```

```

Dim thePackageUseCase As Category
Dim thePackageClass As Category
Dim classDependum As Class
Dim classDepender As Class
Dim classDependee As Class
Dim theDiagramClass As ClassDiagram
Dim theDiagramUseCase As ClassDiagram

' Function: Main
' Description:      - opens a dialog box for the user to define the location of the file
telos.
'                  - call the function Mapping
Sub Main
    Dim path As String
    path = OpenFileDialog$("Abrir Arquivo Telos", "Arquivos Telos: *.tel;Todos os Arquivos:
*.*")
    If path <> "" Then
        Mapping path
    Else
        MsgBox "Nenhum Arquivo Foi Escolhido"
    End If
End Sub

Sub Mapping (path As String)

    Dim heranca As String
    Dim actor As ActorElement
    Dim goal As GoalElement
    Dim task As TaskElement
    Dim softgoal As GoalElement
    Dim resource As ResourceElement
    Dim dependency As DependencyLink
    Dim meansEnd As MeansEndsLink
    Dim decomposition As DecompositionLink

    Dim i As Integer
    Dim j As Integer
    Dim k As Integer
    Dim l As Integer
    Dim ichildren As Integer
    Dim ilinks As Integer
    Dim lin As String
    Dim cod As String
    Dim typeAttribute As String

```

```

iactor = -1
igoal = -1
itask = -1
isoftgoal = -1
iresource = -1
idependency = -1
imeansEnd = -1

Open path For Input As #1
While Not EOF(1)
    Line Input #1, lin
    cod = IsTokenOME (lin)
    If cod <> "" Then
        Line Input #1, lin
        heranca = Token(lin,2)
        If (UCase(heranca) = "ISTARACTORELEMENT") Then
            ichildren = 0
            ilinks = 0
            Erase actor.links
            Erase actor.children
            actor.cod = cod
            Line Input #1, lin
            Line Input #1, lin
            While (Token(lin,1) <> "END")
                typeAttribute = Token(lin,2)
                Line Input #1, lin
                If (typeAttribute = "name") Then
                    actor.name_ = ReadName(lin)
                ElseIf (typeAttribute = "children") Then
                    actor.children(ichildren)= Token(lin,2)
                    ichildren = ichildren+1
                ElseIf (typeAttribute = "links") Then
                    actor.links(ilinks) = Token(lin,2)
                    ilinks = ilinks +1
                End If
                Line Input #1, lin
            Wend
            iactor = iactor +1
            actors(iactor) = actor
        ElseIf (UCase(heranca) = "ISTARGOALELEMENT") Then
            ilinks = 0
            goal.parent = ""
            Erase goal.links
            goal.cod = cod
            Line Input #1, lin
            Line Input #1, lin
            While (Token(lin,1) <> "END")
                typeAttribute = Token(lin,2)
                Line Input #1, lin

```

```

        If (typeAttribute = "name") Then
            goal.name_ = ReadName(lin)
        ElseIf (typeAttribute = "parent") Then
            goal.parent = Token(lin,2)
        ElseIf (typeAttribute = "links") Then
            goal.links(ilinks) = Token(lin,2)
            ilinks = ilinks +1
        End If
        Line Input #1, lin
    Wend
    igoal = igoal + 1
    goals(igoal) = goal
ElseIf (UCase(heranca) = "ISTARTASKELEMENT") Then
    ilinks = 0
    task.parent=""
    Erase task.links
    task.cod = cod
    Line Input #1, lin
    Line Input #1, lin
    While (Token(lin,1) <>"END")
        typeAttribute = Token(lin,2)
        Line Input #1, lin
        If (typeAttribute = "name") Then
            task.name_ = ReadName(lin)
        ElseIf (typeAttribute = "parent") Then
            task.parent = Token(lin,2)
        ElseIf (typeAttribute = "links") Then
            task.links(ilinks) = Token(lin,2)
            ilinks = ilinks +1
        End If
        Line Input #1, lin
    Wend
    itask = itask + 1
    tasks(itask) = task

ElseIf (UCase(heranca) = "ISTARSOFTGOALELEMENT") Then
    ilinks = 0
    softgoal.parent = ""
    Erase softgoal.links
    softgoal.cod = cod
    Line Input #1, lin
    Line Input #1, lin
    While (Token(lin,1) <>"END")
        typeAttribute = Token(lin,2)
        Line Input #1, lin
        If (typeAttribute = "name") Then
            softgoal.name_ = ReadName(lin)
        ElseIf (typeAttribute = "parent") Then
            softgoal.parent = Token(lin,2)

```

```
        ElseIf (typeAttribute = "links") Then
            softgoal.links(ilinks) = Token(lin,2)
            ilinks = ilinks +1
        End If
        Line Input #1, lin
    Wend
    isoftgoal = isoftgoal + 1
    softgoals(isoftgoal) = softgoal

ElseIf (UCase(heranca) = "ISTARRESOURCEELEMENT") Then
    ilinks = 0
    resource.parent = ""
    Erase resource.links
    resource.cod = cod
    Line Input #1, lin
    Line Input #1, lin
    While (Token(lin,1) <>"END")
        typeAttribute = Token(lin,2)
        Line Input #1, lin
        If (typeAttribute = "name") Then
            resource.name_ = ReadName(lin)
        ElseIf (typeAttribute = "parent") Then
            resource.parent = Token(lin,2)
        ElseIf (typeAttribute = "links") Then
            resource.links(ilinks) = Token(lin,2)
            ilinks = ilinks +1
        End If
        Line Input #1, lin
    Wend
    iresource = iresource + 1
    resources(iresource) = resource

ElseIf (UCase(heranca) = "ISTARDEPENDENCYLINK") Then
    dependency.cod = cod
    dependency.name_ = ""
    Line Input #1, lin
    Line Input #1, lin
    While (Token(lin,1) <>"END")
        typeAttribute = Token(lin,2)
        Line Input #1, lin
        If (typeAttribute = "name") Then
            dependency.name_ = ReadName(lin)
        ElseIf (typeAttribute = "to") Then
            dependency.to_ = Token(lin,3)
        ElseIf (typeAttribute = "from") Then
            dependency.from = Token(lin,3)
        End If
        Line Input #1, lin
    Wend
```

```

idependency = idependency + 1
dependencies(idependency) = dependency

ElseIf (UCase(heranca) = "ISTARMEANSENDSLINK") Then
    meansEnd.cod = cod
    meansEnd.name_ = ""
    Line Input #1, lin
    Line Input #1, lin
    While (Token(lin,1) <>"END")
        typeAttribute = Token(lin,2)
        Line Input #1, lin
        If (typeAttribute = "name") Then
            meansEnd.name_ = ReadName(lin)
        ElseIf (typeAttribute = "to") Then
            meansEnd.to_ = Token(lin,3)
        ElseIf (typeAttribute = "from") Then
            meansEnd.from = Token(lin,3)
        End If
        Line Input #1, lin
    Wend
    imeansEnd = imeansEnd + 1
    meansEnds(imeansEnd) = meansEnd

ElseIf (UCase(heranca) = "ISTARDECOMPOSITIONLINK") Then
    decomposition.cod = cod
    decomposition.name_ = ""
    Line Input #1, lin
    Line Input #1, lin
    While (Token(lin,1) <>"END")
        typeAttribute = Token(lin,2)
        Line Input #1, lin
        If (typeAttribute = "name") Then
            decomposition.name_ = ReadName(lin)
        ElseIf (typeAttribute = "to") Then
            decomposition.to_ = Token(lin,3)
        ElseIf (typeAttribute = "from") Then
            decomposition.from = Token(lin,3)
        End If
        Line Input #1, lin
    Wend
    idecomposition = idecomposition + 1
    decompositions(idecomposition) = decomposition

Else
    While (Token(lin,1) <>"END")
        Line Input #1, lin
    Wend
End If
End If

```

Wend

Close #1

'Lendo Regra de Mapeamento

'Tabela de Equivalência de Ator

' 0 = "Classe"

' 1 = "<<Ator>>"

'Tabela de Equivalência de Objetivo

'0 = ""

'1 = "Classe"

'2 = "<<Goal>>"

'3 = "Atributo Boolean public no Dependee"

'Tabela de Equivalência de Objetivo Soft

'0 = ""

'1 = "Classe"

'2 = "<<SoftGoal>>"

'3 = "Atributo Enumerado public no Dependee"

'Tabela de Equivalência de Tarefa

'0 = ""

'1 = "<<Task>>"

'2 = "Caso de Uso"

'3 = "Operação public no Dependee"

'Tabela de Equivalência de Recurso

'0 = ""

'1 = "Classe"

'2 = "<<Resource>>"

'3 = "Atributo public no Dependee"

'Tabela de Equivalência de SubObjetivo

'0 = ""

'1 = "<<Goal>>"

'2 = " Atributo Boolean private no Ator"

'Tabela de Equivalência de SubObjetivo Soft

'0 = ""

'1 = "<<SoftGoal>>"

'2 = "Atributo Enumerado private no Ator"

'Tabela de Equivalência de SubTarefa

'0 = ""

'1 = "<<Task>>"

'2 = "Operação private no Ator"

'Tabela de Equivalência de SubRecurso

'0 = ""

```
'1 = "<<Resource>>"
'2 = "Atributo Boolean Private no Ator"
filename$ = "c:\Arquivos de Programas\Good\mapSetting.cfg"
Open filename$ For Input As #2
While Not EOF(2)
    Line Input #2,linha$
    var1 = Token(linha, 1)
    var2 = Token(linha, 2)
    var3 = Token(linha, 3)
    If (var1 = "Map") Then
        If (var2 = "Actor") Then
            rActor = CInt(var3)
        End If
        If (var2 = "Goal") Then
            rGoal = CInt(var3)
        End If
        If (var2 = "SoftGoal") Then
            rSoftGoal = CInt(var3)
        End If
        If (var2 = "Task") Then
            rTask = CInt(var3)
        End If
        If (var2 = "Resource") Then
            rResource = CInt(var3)
        End If
        If (var2 = "SubGoal") Then
            rSubGoal = CInt(var3)
        End If
        If (var2 = "SubSoftGoal") Then
            rSubSoftGoal = CInt(var3)
        End If
        If (var2 = "SubTask") Then
            rSubTask = CInt(var3)
        End If
        If (var2 = "SubResource") Then
            rSubResource = CInt(var3)
        End If
    End If
Wend
Close #2

Dim model As Model
Dim theRole As Role
Dim theCategories As CategoryCollection
Dim thePackage As Category
Dim theAssociation As Association
Dim classes As ClassCollection
Dim useCases As UseCaseCollection
Dim theClass As Class
```

```
Dim theClass2 As Class
Dim theUseCase As UseCase
Dim added As Boolean
Dim dependeeName As String
Dim operation As Operation
Dim attribute As Attribute

'Criando Pacotes Visao Organizacional
Set model = RoseApp.CurrentModel
Set theCategories = model.GetAllCategories ( )
Set thePackage = theCategories.GetFirst ("Use Case View")
Set thePackageUseCase = thePackage.AddCategory("Visao Organizacional")
Set theDiagramUseCase = thePackageUseCase.AddClassDiagram ("Diagrama Organizacional")

Set thePackage = theCategories.GetFirst ("Logical View")
Set thePackageClass = thePackage.AddCategory("Visao Organizacional")
Set theDiagramClass = thePackageClass.AddClassDiagram ("Diagrama Organizacional")

'Mapeando Ator
For i = 0 To iactor
    actor = actors(i)
    If (rActor = 0) Then
        Set theClass = thePackageClass.AddClass(actor.name_)
        added = theDiagramClass.AddClass (theClass)
    ElseIf (rActor = 1) Then
        Set theClass = thePackageUseCase.AddClass (actor.name_)
        theClass.Stereotype = "Actor"
        added = theDiagramUseCase.AddClass (theClass)
    End If
Next

'Mapeando Entidades
'Mapeando Tarefa e SubTarefa

For i = 0 To itask
    task = tasks(i)
    If (task.parent = "") Then 'Tarefa
        If (rTask = 0) Then 'Nao mapear
            'Nao faca nada
        ElseIf (rTask = 1) Then 'mapear para <<Task>>
            Set theClass = thePackageClass.AddClass (task.name_)
            added = theDiagramClass.AddClass (theClass)
            theClass.Stereotype = "Task"
        ElseIf (rTask = 2) Then 'mapear para Use Case
            Set theUseCase = thePackageUseCase.AddUseCase (task.name_)
            added = theDiagramUseCase.AddUseCase (theUseCase)

            ElseIf (rTask = 3) Then 'mapear para Operacao public no Dependee
```

```

dependeeName = findDependee(task.cod)
If (rActor = 0) Then
    Set classes = thePackageClass.GetAllClasses()
Else
    Set classes = thePackageUseCase.GetAllClasses()
End If
If dependeeName <> "" Then
    Set theClass = classes.GetFirst(dependeeName)
    Set operation = theClass.AddOperation (task.name_, "")
    operation.ExportControl.Name = "PublicAccess"
End If
End If
Else 'SubTarefa
    If (rSubTask = 0) Then 'Nao mapear
        'Nao faca nada
    ElseIf (rSubTask = 1) Then 'mapear para <<Task>>
        Set theClass = thePackageClass.AddClass (task.name_)
        added = theDiagramClass.AddClass (theClass)
        theClass.Stereotype = "Task"
        If (rActor = 0) Then
            Set classes = thePackageClass.GetAllClasses()
        Else
            Set classes = thePackageUseCase.GetAllClasses()
        End If
        auxString$ = findActor(task.parent)
        Set theClass2 = classes.GetFirst(auxString$)
        Set theAssociation = theClass.AddAssociation ("parent",
theClass2.name)

        added = theDiagramClass.AddAssociation (theAssociation)
        Set theRole = theAssociation.GetOtherRole (theClass2)
        theRole.name = "children"
        theRole.Navigable = True
        theRole.ExportControl.Value = 3
        Set theRole = theAssociation.GetOtherRole (theClass)
        theRole.ExportControl.Value = 3
    ElseIf (rSubTask = 2) Then 'mapear para Operacao private no Ator
        If (rActor = 0) Then
            Set classes = thePackageClass.GetAllClasses()
        Else
            Set classes = thePackageUseCase.GetAllClasses()
        End If
        If dependeeName <> "" Then
            Set theClass = classes.GetFirst(theFindActor(task.parent))
            Set operation = theClass.AddOperation (task.name_, "")
            operation.ExportControl.Name = "PrivateAccess"
        End If
    End If
End If
Next

```

```

'Mapeando Recurso e SubRecurso

For i = 0 To iresource
    resource = resources(i)
    If (resource.parent = "") Then 'Recurso
        If (rResource = 0) Then      'Nao mapear
            'Nao faca nada
        ElseIf (rResource = 1) Then 'mapear para Classe
            Set theClass = thePackageClass.AddClass (resource.name_)
            added = theDiagramClass.AddClass (theClass)
        ElseIf (rResource = 2) Then 'mapear para <<Resource>>
            Set theClass = thePackageClass.AddClass (resource.name_)
            theClass.Stereotype = "Resource"
            added = theDiagramClass.AddClass (theClass)
        ElseIf (rResource = 3) Then 'mapear para Atributo public no Dependee
            dependeeName = findDependee(resource.cod)
            If (rActor = 0) Then
                Set classes = thePackageClass.GetAllClasses()
            Else
                Set classes = thePackageUseCase.GetAllClasses()
            End If
            If dependeeName <> "" Then
                Set theClass = classes.GetFirst(dependeeName)
                Set attribute = theClass.AddAttribute (resource.name_,
"Boolean","")
                attribute.ExportControl.Name = "PublicAccess"
            End If
        End If
    Else 'SubRecurso
        If (rSubResource = 0) Then      'Nao mapear
            'Nao faca nada
        ElseIf (rSubResource = 1) Then 'mapear para <<Resource>>
            Set theClass = thePackageClass.AddClass (resource.name_)
            added = theDiagramClass.AddClass (theClass)
            theClass.Stereotype = "Resource"
            If (rActor = 0) Then
                Set classes = thePackageClass.GetAllClasses()
            Else
                Set classes = thePackageUseCase.GetAllClasses()
            End If
            auxString$ = findActor(resource.parent)
            Set theClass2 = classes.GetFirst(auxString$)
            Set theAssociation = theClass.AddAssociation ("parent",
theClass2.name)

            added = theDiagramClass.AddAssociation (theAssociation)
            Set theRole = theAssociation.GetOtherRole (theClass2)
            theRole.name = "children"
            theRole.Navigable = True

```

```

        theRole.ExportControl.Value = 3
        Set theRole = theAssociation.GetOtherRole (theClass)
        theRole.ExportControl.Value = 3
    ElseIf (rSubResource = 2) Then 'mapear para Atributo private no Ator
        If (rActor = 0) Then
            Set classes = thePackageClass.GetAllClasses()
        Else
            Set classes = thePackageUseCase.GetAllClasses()
        End If
        If dependeeName <> "" Then
            Set                                     theClass                                     =
classes.GetFirst(theFindActor(resource.parent))
            Set attribute = theClass.AddAttribute (resource.name_,
"Boolean", "")
            attribute.ExportControl.Name = "PrivateAccess"
        End If
    End If
End If
Next

'Mapeando Objetivo e SubObjetivo

For i = 0 To igoal
    goal = goals(i)
    If (goal.parent = "") Then 'Objetivo
        If (rGoal = 0) Then 'Nao mapear
            'Nao faca nada
        ElseIf (rGoal = 1) Then 'mapear para classe
            Set theClass = thePackageClass.AddClass (goal.name_)
            added = theDiagramClass.AddClass (theClass)
        ElseIf (rGoal = 2) Then 'mapear para <<Resource>>
            Set theClass = thePackageClass.AddClass (goal.name_)
            theClass.Stereotype = "Goal"
            added = theDiagramClass.AddClass (theClass)
        ElseIf (rGoal = 3) Then 'mapear para Atributo public no Dependee
            dependeeName = findDependee(goal.cod)
            If (rActor = 0) Then
                Set classes = thePackageClass.GetAllClasses()
            Else
                Set classes = thePackageUseCase.GetAllClasses()
            End If
            If dependeeName <> "" Then
                Set theClass = classes.GetFirst(dependeeName)
                Set attribute = theClass.AddAttribute (goal.name_,
"Boolean", "")
                attribute.ExportControl.Name = "PublicAccess"
            End If
        End If
    Else 'SubObjetivo

```

```

If (rSubGoal = 0) Then 'Nao mapear
    'Nao faca nada
ElseIf (rSubGoal = 1) Then 'mapear para <<Goal>>
    Set theClass = thePackageClass.AddClass (goal.name_)
    added = theDiagramClass.AddClass (theClass)
    theClass.Stereotype = "Goal"
    If (rActor = 0) Then
        Set classes = thePackageClass.GetAllClasses()
    Else
        Set classes = thePackageUseCase.GetAllClasses()
    End If
    auxString$ = findActor(goal.parent)
    Set theClass2 = classes.GetFirst(auxString$)
    Set theAssociation = theClass.AddAssociation ("parent",
theClass2.name)

    added = theDiagramClass.AddAssociation (theAssociation)
    Set theRole = theAssociation.GetOtherRole (theClass2)
    theRole.name = "children"
    theRole.Navigable = True
    theRole.ExportControl.Value = 3
    Set theRole = theAssociation.GetOtherRole (theClass)
    theRole.ExportControl.Value = 3
ElseIf (rSubGoal = 2) Then 'mapear para Atributo private no Ator
    If (rActor = 0) Then
        Set classes = thePackageClass.GetAllClasses()
    Else
        Set classes = thePackageUseCase.GetAllClasses()
    End If
    If dependeeName <> "" Then
        Set theClass = classes.GetFirst(theFindActor(goal.parent))
        Set attribute = theClass.AddAttribute (goal.name_,
"Boolean", "")
        attribute.ExportControl.Name = "PrivateAccess"
    End If
    End If
End If
Next

'Mapeando Objetivo Soft e SubObjetivo Soft

For i = 0 To isoftgoal
    softgoal = softgoals(i)
    If (softgoal.parent = "") Then 'Objetivo
        If (rSoftGoal = 0) Then 'Nao mapear
            'Nao faca nada
        ElseIf (rSoftGoal = 1) Then 'mapear para classe
            Set theClass = thePackageClass.AddClass (softgoal.name_)
            added = theDiagramClass.AddClass (theClass)
        ElseIf (rSoftGoal = 2) Then 'mapear para <<Resource>>

```

```

        Set theClass = thePackageClass.AddClass (softgoal.name_)
        theClass.Stereotype = "SoftGoal"
        added = theDiagramClass.AddClass (theClass)
    ElseIf (rSoftGoal = 3) Then 'mapear para Atributo public no Dependee
        dependeeName = findDependee(softgoal.cod)
        If (rActor = 0) Then
            Set classes = thePackageClass.GetAllClasses()
        Else
            Set classes = thePackageUseCase.GetAllClasses()
        End If
        If dependeeName <> "" Then
            Set theClass = classes.GetFirst(dependeeName)
            Set attribute = theClass.AddAttribute (softgoal.name_,
"{satisfeito, parcialmente satisfeito, não satisfeito","")
            attribute.ExportControl.Name = "PublicAccess"
        End If
    End If
Else 'SubObjetivo Soft
    If (rSubSoftGoal = 0) Then      'Nao mapear
        'Nao faca nada
    ElseIf (rSubSoftGoal = 1) Then 'mapear para <<SoftGoal>>
        Set theClass = thePackageClass.AddClass (softgoal.name_)
        added = theDiagramClass.AddClass (theClass)
        theClass.Stereotype = "SoftGoal"
        If (rActor = 0) Then
            Set classes = thePackageClass.GetAllClasses()
        Else
            Set classes = thePackageUseCase.GetAllClasses()
        End If
        auxString$ = findActor(softgoal.parent)
        Set theClass2 = classes.GetFirst(auxString$)
        Set theAssociation = theClass.AddAssociation ("parent",
theClass2.name)

        added = theDiagramClass.AddAssociation (theAssociation)
        Set theRole = theAssociation.GetOtherRole (theClass2)
        theRole.name = "children"
        theRole.Navigable = True
        theRole.ExportControl.Value = 3
        Set theRole = theAssociation.GetOtherRole (theClass)
        theRole.ExportControl.Value = 3
    ElseIf (rSubSoftGoal = 2) Then 'mapear para Atributo private no Ator
        If (rActor = 0) Then
            Set classes = thePackageClass.GetAllClasses()
        Else
            Set classes = thePackageUseCase.GetAllClasses()
        End If
        If dependeeName <> "" Then
            Set
                                theClass
                                =
classes.GetFirst(theFindActor(softgoal.parent))

```

```

        Set attribute = theClass.AddAttribute (softgoal.name_,
"Boolean","")
        attribute.ExportControl.Name = "PrivateAccess"
    End If
End If
End If
Next

'Mapeando Relacionamentos
'Mapeando Relacionamentos de Dependencia

For i = 0 To idependency
    dependency = dependencies(i)
    typeTo$ = findTypeElement(dependency.to_)
    typeFrom$ = findTypeElement(dependency.from)
    If (typeFrom$ = "Actor") Then
        'Ele e
um dependor
        dependerName$ = findActor(dependency.from)
        dependerCod$ = dependency.from
        dependumName$ = findElement(dependency.to_)
        dependumCod$ = dependency.to_
        For j = 0 To idependency
            dependency = dependencies(j)
            If (dependumCod$ = dependency.from) Then
                If (findTypeElement(dependency.to_) = "Actor") Then
                    dependeeName$ = findElement(dependency.to_)
                    If (rActor=1) Then
                        Set classes = thePackageUseCase.GetAllClasses()
                    Else
                        Set classes =
thePackageClass.GetAllClasses()
                End If
                Set classDepender = classes.GetFirst(dependerName$)
                Set classDependee = classes.GetFirst(dependeeName$)
                If (findTypeElement(dependency.from) = "Goal") Then
                    If (rGoal=0) Then
                        'Nao mapeia o relacionamento
                    ElseIf ((rGoal=1) Or (rGoal=2)) Then
                        Set classes =
thePackageClass.GetAllClasses()
                        Set classDependum =
classes.GetFirst(dependumName$)
                        createAssociationDDD
                    ElseIf (rGoal=3) Then
                        createAssociationDD
                    End If
                ElseIf (findTypeElement(dependency.from) = "SoftGoal")
Then
                    If (rSoftGoal=0) Then

```

```

        'Nao mapeia o relacionamento
    ElseIf ((rSoftGoal=1) Or (rSoftGoal=2)) Then
        Set          classes          =
thePackageClass.GetAllClasses()

        Set          classDependum    =
classes.GetFirst(dependumName$)

        createAssociationDDD
    ElseIf (rSoftGoal=3) Then
        createAssociationDD
    End If

    ElseIf (findTypeElement(dependency.from) = "Task") Then
        If (rTask=0) Then
            'Nao mapeia o relacionamento
        ElseIf (rTask=1) Then
            Set          classes          =
thePackageClass.GetAllClasses()

            Set          classDependum    =
classes.GetFirst(dependumName$)

            createAssociationDDD
        ElseIf (rTask=2) Then
            Set          useCases         =
thePackageUseCase.GetAllUseCases()

            Set theUseCase                =
useCases.GetFirst(dependumName$)

            createAssociationUseCase theUseCase
        ElseIf (rTask=3) Then
            createAssociationDD
        End If

    ElseIf (findTypeElement(dependency.from) =
"Resource")Then
        If (rResource=0) Then
            'Nao mapeia o relacionamento
        ElseIf (rResource=1) Or (rResource=2) Then
            Set          classes          =
thePackageClass.GetAllClasses()

            Set          classDependum    =
classes.GetFirst(dependumName$)

            createAssociationDDD
        ElseIf (rResource=3) Then
            createAssociationDD
        End If

    End If
Else
    If (findTypeElement(dependency.from) = "SubGoal") Then

```

```

ElseIf (findTypeElement(dependency.from) =
"SubSoftGoal") Then
ElseIf (findTypeElement(dependency.from) = "SubTask")
Then
ElseIf (findTypeElement(dependency.from) =
"SubResource")Then
End If
End If
End If
Next
End If
Next

End Sub

Sub createAssociationUseCase (theUseCase As UseCase)
Dim theAssociation As Association
Dim theRole As Role
Dim added As Boolean
Set theAssociation = theUseCase.AddAssociation ("", classDepender.name)
added = theDiagramUseCase.AddAssociation (theAssociation)
End Sub

'Cria associacao entre as classes Depender-Dependum-Dependee
Sub createAssociationDDD ()

Dim theAssociation As Association
Dim theRole As Role
Dim added As Boolean
Set theAssociation = classDependum.AddAssociation ("dependor", classDepender.name)
added = theDiagramClass.AddAssociation (theAssociation)
Set theRole = theAssociation.GetOtherRole (classDepender)
theRole.name = "dependum"
theRole.Navigable = True
theRole.ExportControl.Value = 3
Set theRole = theAssociation.GetOtherRole (classDependum)
theRole.ExportControl.Value = 3

Set theAssociation = classDependum.AddAssociation ("dependee", classDependee.name)
added = theDiagramClass.AddAssociation (theAssociation)
Set theRole = theAssociation.GetOtherRole (classDependee)
theRole.name = "dependum"
theRole.Navigable = True
theRole.ExportControl.Value = 3
Set theRole = theAssociation.GetOtherRole (classDependum)
theRole.ExportControl.Value = 3

End Sub

```

```

Sub createAssociationDD ()
    Dim theAssociation As Association
    Dim theRole As Role
    Dim added As Boolean
    Dim theAssociateRoles As RoleCollection
    Dim lengthRole As Integer
    Set theAssociateRoles = classDepender.GetAssociateRoles()
    lengthRole = theAssociateRoles.Count
    For i=1 To lengthRole
        Set theRole = theAssociateRoles.GetAt(i)
        If (theRole.GetClassName() = classDependee.Name) And (theRole.Name = "dependee")
Then
            i = lengthRole + 1
            var$ = "Encontrou"
        End If
    Next
    If (var$ <> "Encontrou") Then
        Set theAssociation = classDepender.AddAssociation ("dependee",
classDependee.name)
        added = theDiagramClass.AddAssociation (theAssociation)
        Set theRole = theAssociation.GetOtherRole (classDependee)
        theRole.name = "dependee"
        theRole.Navigable = True
        theRole.ExportControl.Value = 3
        Set theRole = theAssociation.GetOtherRole (classDepender)
        theRole.ExportControl.Value = 3
    End If
End Sub

Function ReadName (lin As String) As String
    Dim vector(150) As String
    Dim var As String
    Dim length As Integer
    Dim posBegin As Integer
    Dim posEnd As Integer
    posBegin = -1
    length = Len(lin)
    For i = 1 To length
        If (Asc(Mid(lin,i,1)) = 34) Then
            If posBegin = -1 Then
                posBegin = i
            Else
                posEnd = i
            End If
        End If
        vector(i) = Mid(lin,i,1)
    Next
    For i = posBegin+1 To posEnd-1

```

```

        var = var & vector(i)
    Next
    ReadName = var
End Function

```

'Funcao que verifica se a linha representa o início de um Token
 'Entrada: uma string representando uma linha
 'Saida: uma string: - uma string vazia quando a linha nao representa um Token
 ' - string com o valor do codigo gerado pelo OME

```
Function IsTokenOME (lin As String) As String
```

```

    Dim amount As Integer
    Dim palavra As String
    amount = TokenCount(lin)
    palavra = Token(lin,1)
    If (amount >= 1) Then
        palavra = Token(lin,1)
        If (amount = 2) Then
            If (UCase(palavra) = "TOKEN") Then
                IsTokenOME = Token(lin,2)
            End If
        End If
    End If
End If
End Function

```

```
Function Token (lin As String, pos As Integer) As String
```

```

    Dim posToken As Integer
    Dim char As String
    Dim length As Integer
    Dim i As Integer
    Dim beginToken As Boolean
    Dim cod As Integer
    Dim tok As String
    i=1
    posToken = 1
    beginToken = False
    length = Len(lin)
    While (posToken <= pos And i <= length)
        char = Mid(lin,i,1)
        cod = Asc(char)
        If ((cod = 9) Or (cod = 0) Or (cod = 26) Or (cod = 13) Or (cod = 10) Or (cod =
32)) Then
            If (beginToken = True)Then
                beginToken = False
                posToken = posToken +1
            End If
        Else

```

```

        If (beginToken = False) Then
            beginToken = True
        End If
        If (posToken = pos) Then
            tok = tok & char
        End If
    End If
    i = i+1
Wend
Token = tok
End Function

Function TokenCount (lin As String) As Integer
    Dim count As Integer
    Dim beginToken As Boolean
    Dim char As String
    Dim i As Integer
    beginToken = False
    count = 0
    i=1
    length = Len(lin)
    While (i <= length)
        char = Mid(lin,i,1)
        cod = Asc(char)
        If ((cod = 9) Or (cod = 0) Or (cod = 26) Or (cod = 13) Or (cod = 10) Or
(cod = 32)) Then
            If (beginToken = True)Then
                beginToken = False
            End If
        Else
            If (beginToken = False) Then
                beginToken = True
                count = count +1
            End If
        End If
        i = i+1
    Wend
    TokenCount = count
End Function

Function findDependee (codSearch As String) As String
    For i=0 To idependency
        If dependencies(i).from = codSearch Then
            dependee$= findActor(dependencies(i).to_)
            If (dependee$ = "") Then
                dependee$ = findGoalParent(dependencies(i).to_)
                If (dependee$ = "") Then

```

```

        dependee$ = findSoftGoalParent(dependencies(i).to_)
        If dependee$ = "" Then
            dependee$ = findResourceParent(dependencies(i).to_)
            If dependee$ = "" Then
                dependee$ = findTaskParent(dependencies(i).to_)
            End If
        End If
    End If
End If
End If
Next
findDependee = dependee$
End Function

```

'Retorna o nome do Ator a partir de seu código ou uma string caso ele não encontre

```

Function findActor (codSearch As String) As String
    findActor = ""
    For i=0 To iactor
        If actors(i).cod = codSearch Then
            findActor = actors(i).name_
            i = iactor+1
        End If
    Next
End Function

```

```

Function findGoal (codSearch As String) As String
    findGoal = ""
    For i=0 To igoal
        If goals(i).cod = codSearch Then
            findGoal = goals(i).name_
            i = igoal+1
        End If
    Next
End Function

```

```

Function findSoftGoal (codSearch As String) As String
    findSoftGoal = ""
    For i=0 To isoftgoal
        If softgoals(i).cod = codSearch Then
            findSoftGoal = softgoals(i).name_
            i = isoftgoal+1
        End If
    Next
End Function

```

```

Function findTask (codSearch As String) As String
    findTask = ""
    For i=0 To itask

```

```

        If tasks(i).cod = codSearch Then
            findTask = tasks(i).name_
            i = itask+1
        End If
    Next
End Function

Function findResource (codSearch As String) As String
    findResource = ""
    For i=0 To iresource
        If resources(i).cod = codSearch Then
            findResource = resources(i).name_
            i = iresource+1
        End If
    Next
End Function

'Retorna o nome do Elemento ou uma String vazia "" caso ele nao seja encontrado
Function findElement (codSearch As String) As String
    typeElement$ = findTypeElement(codSearch)
    If (typeElement = "") Then
        findElement = ""
    ElseIf (typeElement = "Goal") Then
        findElement = findGoal (codSearch)
    ElseIf (typeElement = "SoftGoal") Then
        findElement = findSoftGoal (codSearch)
    ElseIf (typeElement = "Task") Then
        findElement = findTask (codSearch)
    ElseIf (typeElement = "Resource") Then
        findElement = findResource (codSearch)
    ElseIf (typeElement = "SubGoal") Then
        findElement = findSubGoal (codSearch)
    ElseIf (typeElement = "SubSoftGoal") Then
        findElement = findSubSoftGoal (codSearch)
    ElseIf (typeElement = "SubTask") Then
        findElement = findSubTask (codSearch)
    ElseIf (typeElement = "SubResource") Then
        findElement = findSubResource (codSearch)
    ElseIf (typeElement = "Actor") Then
        findElement = findActor (codSearch)
    End If
End Function

'Retorna o nome do Pai caso seja um objetivo ou uma string vazia
Function findGoalParent (codSearch As String) As String
    findGoalParent = ""
    For i=0 To igoal

```

```

        If goals(i).cod = codSearch Then
            auxString$ = goals(i).parent
            findGoalParent = findActor(auxString$)
            i = igoal+1
        End If
    Next
End Function

'Retorna o nome do Pai caso seja um sub-objetivo ou uma string vazia
Function findTaskParent (codSearch As String) As String
    findTaskParent = ""
    For i=0 To itask
        If tasks(i).cod = codSearch Then
            auxString$ = tasks(i).parent
            findTaskParent = findActor(auxString$)
            i = itask+1
        End If
    Next
End Function

'Retorna o nome do Pai caso seja um sub-objetivo soft ou uma string vazia
Function findSoftGoalParent (codSearch As String) As String
    findSoftGoalParent = ""
    For i=0 To isoftgoal
        If softgoals(i).cod = codSearch Then
            auxString$ = softgoals(i).parent
            findSoftGoalParent = findActor(auxString$)
            i = isoftgoal+1
        End If
    Next
End Function

'Retorna o nome do Pai caso seja um sub-recurso ou uma string vazia
Function findResourceParent (codSearch As String) As String
    findResourceParent = ""
    For i=0 To iresource
        If resources(i).cod = codSearch Then
            auxString$ = resources(i).parent
            findResourceParent = findActor(auxString$)
            i = iresource+1
        End If
    Next
End Function

'Retorna o tipo do Elemento ou vazio caso ele nao seja encontrado
Function findTypeElement (codSearch As String) As String
    findTypeElement = ""
    If (findActor(codSearch) <> "") Then

```

```

        findTypeElement = "Actor"
    ElseIf (findResource(codSearch) <> "") Then
        If findResourceParent(codSearch) = "" Then
            findTypeElement = "Resource"
        Else
            findTypeElement = "SubResource"
        End If
    ElseIf (findTask(codSearch) <> "") Then
        If findTaskParent(codSearch) = "" Then
            findTypeElement = "Task"
        Else
            findTypeElement = "SubTask"
        End If
    ElseIf (findGoal(codSearch) <> "") Then
        If findGoalParent(codSearch) = "" Then
            findTypeElement = "Goal"
        Else
            findTypeElement = "SubGoal"
        End If
    ElseIf (findSoftGoal(codSearch) <> "") Then
        If findSoftGoalParent(codSearch) = "" Then
            findTypeElement = "SoftGoal"
        Else
            findTypeElement = "SubSoftGoal"
        End If
    End If
End Function

```

Apêndice D

ARQUIVO TELOS DO FRAMEWOK I*

```
##
#   Istar modelling framework.
##

MetaClass IStarElementLabelClass
    IN OMEAttributeMetaClass
    ISA OMEAttributeClass
END

MetaClass IStarLinkLabelClass
    IN OMEAttributeMetaClass
    ISA OMEAttributeClass
END

MetaClass IStarClass
    ISA OMEObjectClass
END

# we want to be able to create 'links'
MetaClass IStarLinkClass
    ISA IStarClass, OMELinkClass
END

# elements is another class of istar object.
MetaClass IStarElementClass
    ISA IStarClass, OMEElementClass
END

# actors are a kind of element
MetaClass IStarActorElementClass
    ISA IStarElementClass
END

# intentional elements are another heading that we want to list under.
MetaClass IStarIntentionalElementClass
    ISA IStarElementClass
END
```

```

###
# Elements:
# We want to be fairly unrestrictive.
# Towards that end we have been flatter in inheritance,
# than we could have been. In particular we do not distinguish
# between those intentional elements that are within actors, and those
# that are not. Presumably, those that are not do not have an actor.
###

SimpleClass IStarElement
    IN IStarClass
    ISA OMEElement
    WITH
        attribute
            name : String
END

SimpleClass IStarActorElement
    IN IStarClass, IStarActorElementClass, OMEInstantiableClass
    ISA IStarElement, OMEGrowableElement
    WITH
        attribute
            children : IStarIntentionalElement
        name
            : "Actor"
        imagename
            : "Actor.gif"
        imagesize
            width : 80;
            height : 80
END

SimpleClass IStarIntentionalElement
    IN IStarClass, OMEElementClass, OMEVisibleClass
    ISA IStarElement
    WITH
        attribute
            parent : IStarActorElement
        imagesize
            width : 80;
            height : 50
END

# the 8 individual element types:

# actors:

# 3 actors here, actually, since the first is IStarActorElement :)

```

```
SimpleClass IStarRoleElement
    IN IStarActorElementClass, OMEInstantiableClass
    ISA IStarActorElement
    WITH
        name
            : "Role"
        imagename
            : "Role.gif"
        imagesize
            width : 80;
            height : 80
```

END

```
SimpleClass IStarPositionElement
    IN IStarActorElementClass, OMEInstantiableClass
    ISA IStarActorElement
    WITH
        name
            : "Position"
        imagename
            : "Position.gif"
        imagesize
            width : 80;
            height : 80
```

END

```
SimpleClass IStarAgentElement
    IN IStarActorElementClass, OMEInstantiableClass
    ISA IStarActorElement
    WITH
        name
            : "Agent"
        imagename
            : "Agent.gif"
        imagesize
            width : 80;
            height : 80
```

END

Intentional Elements:

```
SimpleClass IStarGoalElement
    IN IStarIntentionalElementClass, OMEInstantiableClass
    ISA IStarIntentionalElement
    WITH
        name
```

```

        : "Goal"
    imagename
        : "Goal.gif"
    imagesize
        width : 80;
        height : 50
    OMEAttributes
        label : IStarElementLabelClass
END

SimpleClass IStarTaskElement
    IN IStarIntentionalElementClass, OMEInstantiableClass
    ISA IStarIntentionalElement
    WITH
        name
            : "Task"
        imagename
            : "Task.gif"
        imagesize
            width : 80;
            height : 50
        OMEAttributes
            label : IStarElementLabelClass
END

SimpleClass IStarSoftGoalElement
    IN IStarIntentionalElementClass, OMEInstantiableClass
    ISA IStarIntentionalElement
    WITH
        name
            : "Softgoal"
        imagename
            : "Softgoal.gif"
        imagesize
            width : 80;
            height : 50
        OMEAttributes
            label : IStarElementLabelClass
END

SimpleClass IStarResourceElement
    IN IStarIntentionalElementClass, OMEInstantiableClass
    ISA IStarIntentionalElement
    WITH
        name
            : "Resource"
        imagename
            : "Resource.gif"

```

```

        imagesize
            width : 80;
            height : 50
        OMEAttributes
            label : IStarElementLabelClass
    END

```

```

###
# Links:
###

```

```

SimpleClass IStarLink
    IN IStarLinkClass, OMEVisibleClass
    ISA OMELink
    WITH
        attribute
            name : String
        imagesize
            height : 20;
            width : 20
    END

```

```

# the various types of links:
#
# Right now the ends of the links are tossed into the 'links' category,
# rather than into the special purpose fields that each of the link classes
# have. Thus checking cannot really take place. I have not thought of a
# scaleable way to allow users to add new types, and also to allow them to

```

```

SimpleClass IStarDependencyLink
    IN IStarLinkClass, OMEInstantiableClass
    ISA IStarLink
    WITH
        name
            : "Dependency link"
        imagename
            : "Dependency.gif"
        imagesize
            height : 20;
            width : 20
        OMEAttributes
            label : IStarLinkLabelClass
    END

```

```

# Decomposition link.

```

```
# Within IStar, only Tasks may be decomposed:

SimpleClass IStarDecompositionLink
    IN IStarLinkClass, OMEInstantiableClass
    ISA IStarLink
    WITH
        attribute
            to : IStarTaskElement;
            from : IStarIntentionalElement
        name
            : "Decomposition link"
        imagename
            : "Decomposition.gif"
        imagesize
            height : 20;
            width : 20
        OMEAttributes
            label : IStarLinkLabelClass
    END

# contributions:
# From intentional elements to softgoals. I believe all
# intentional elements, including other softgoals, may contribute
# to a softgoal.
SimpleClass IStarContributionLink
    IN IStarLinkClass, OMEInstantiableClass
    ISA IStarLink
    WITH
        attribute
            to : IStarSoftGoalElement;
            from : OMEElement
        name
            : "Contribution link"
        imagename
            : "Contribution.gif"
        imagesize
            height : 20;
            width : 20
        OMEAttributes
            label : IStarLinkLabelClass
    END

# Means ends:
# Means ends links, from means to ends, ie: from intentional elements to
# goals.
#
SimpleClass IStarMeansEndsLink
    IN IStarLinkClass, OMEInstantiableClass
```

```

    ISA IStarLink
WITH
    attribute
        to : IStarGoalElement;
        from : IStarIntentionalElement
    name
        : "Means-ends link"
    imagename
        : "Means-Ends.gif"
    imagesize
        height : 20;
        width : 20
    OMEAttributes
        label : IStarLinkLabelClass
END

# isa link:
#   hmm. This one seems to be only be used between actors, and other actors.
#   it might be interesting to have other types of object be classified using
#   isa and ins, but for now we'll just restrict them to actors. jo's rationale:
#   the isa and ins relationships will tend to leave the bounds of an actors's
#   guts alot, and this will clutter the graph. If one wants to extend it, then
#   just make them between IStarElements.

SimpleClass IStarISALink
    IN IStarLinkClass, OMEInstantiableClass
    ISA IStarLink
WITH
    attribute
        to : IStarActorElement;
        from : IStarActorElement
    name
        : "ISA"
    imagename
        : "Arrow.gif"
    imagesize
        height : 20;
        width : 20
    defaultname
        : "ISA"
END

# INS link:
#   Here we are more unrestrive, specifically we do not enforce anything about
#   the actual classes instantiated, or their specialization. We have instead
#   treated these characteristics as mere links. Additional symantic checking
#   was thought to be too restrictive for most users of the tool. In addition
#   to implement specialization and instantiation as telos instantiation and telos

```

```
# specialiazation would have significantly complicated things. It is
# certainly doable, however, as the IStar framework only really uses one
# level of instantiation, so ActorClass would have been a metaclass, Doctor,
# a Class, and Dr. Fred, a token. Specialization would have been no big deal
# except in the case of multiple inheritance. Anyway, this is what we are
# using now:
```

```
SimpleClass IStarINSLink
    IN IStarLinkClass, OMEInstantiableClass
    ISA IStarLink
    WITH
        attribute
            to : IStarActorElement;
            from : IStarActorElement
        name
            : "INS"
        imagename
            : "Arrow.gif"
        imagesize
            height : 20;
            width : 20
        defaultname
            : "INS"
END
```

```
# actor refinement links:
# nothing surprising here.
```

```
SimpleClass IStarPlaysLink
    IN IStarLinkClass, OMEInstantiableClass
    ISA IStarLink
    WITH
        attribute
            to : IStarRoleElement;
            from : IStarAgentElement
        name
            : "Plays"
        imagename
            : "Arrow.gif"
        imagesize
            height : 20;
            width : 20
        defaultname
            : "PLAYS"
END
```

```
SimpleClass IStarOccupiesLink
```

```

    IN IStarLinkClass, OMEInstantiableClass
    ISA IStarLink
    WITH
        attribute
            to : IStarPositionElement;
            from : IStarAgentElement
        name
            : "Occupies"
        imagename
            : "Arrow.gif"
        imagesize
            height : 20;
            width : 20
        defaultname
            : "OCCUPIES"
END

```

```

SimpleClass IStarCoversLink
    IN IStarLinkClass, OMEInstantiableClass
    ISA IStarLink
    WITH
        attribute
            to : IStarRoleElement;
            from : IStarPositionElement
        name
            : "Covers"
        imagename
            : "Arrow.gif"
        imagesize
            height : 20;
            width : 20
        defaultname
            : "COVERS"
END

```

```

SimpleClass IStarDeniedElementLabel
    IN OMEVisibleClass, IStarElementLabelClass
    WITH
        name
            : "Denied"
        imagename
            : "Graphic1.gif"
        imagesize
            width : 50;
            height : 50
        imageposition
            dx : 60;

```

```

        dy : 0
END

SimpleClass IStarWeaklyDeniedElementLabel
    IN OMEVisibleClass, IStarElementLabelClass
    WITH
        name
            : "Weakly Denied"
        imagename
            : "WeaklyDenied.gif"
        imagesize
            width : 50;
            height : 50
        imageposition
            dx : 60;
            dy : 0
END

SimpleClass IStarUndecidedElementLabel
    IN OMEVisibleClass, IStarElementLabelClass
    WITH
        name
            : "Undecided"
        imagename
            : "Undecided.gif"
        imagesize
            width : 50;
            height : 50
        imageposition
            dx : 60;
            dy : 0
END

SimpleClass IStarWeaklySatisficedElementLabel
    IN OMEVisibleClass, IStarElementLabelClass
    WITH
        name
            : "Weakly Satisficed"
        imagename
            : "WeaklySatisficed.gif"
        imagesize
            width : 50;
            height : 50
        imageposition
            dx : 60;
            dy : 0
END

```

```
SimpleClass IStarSatisficedElementLabel
  IN OMEVisibleClass, IStarElementLabelClass
  WITH
    name
      : "Satisficed"
    imagename
      : "Check.gif"
    imagesize
      width : 50;
      height : 50
    imageposition
      dx : 60;
      dy : -0
END
```

```
SimpleClass IStarConflictElementLabel
  IN OMEVisibleClass, IStarElementLabelClass
  WITH
    name
      : "Conflict"
    imagename
      : "Conflict.gif"
    imagesize
      width : 50;
      height : 50
    imageposition
      dx : 55;
      dy : 0
END
```

```
SimpleClass IStarSatisfiedLinkLabel
  IN OMEVisibleClass, IStarLinkLabelClass
  WITH
    name
      : "Satisficed"
    imagename
      : "Check.gif"
    imagesize
      width : 20;
      height : 20
    imageposition
      dx : 20;
      dy : 10
END
```

Referências Bibliográficas

- [AAL99] Aalst, W., Desel, J., Oberweis, A.: *Business Process Management. Models, Techniques, and Empirical Studies*, Lecture Notes in Computer Science, vol. 1806, Springer-Verlag, 1999.
- [ALE98a] Alencar, F., Castro, J., *Guidelines from Organizational Requirements to Formal Specification*, Workshop Iberoamericana de Engenharia de Requisitos e Ambientes de Software, IDEAS'98, 1998.
- [ALE98b] Alencar, F. and Castro, J., *Formalização de Requisitos Organizacionais*. I Workshop em Engenharia de Requisitos – WER'98, 1998.
- [ALE99a] Alencar, F., *Mapeando a Modelagem Organizacional em Especificações Precisas*. Tese de Doutorado, Centro de Informática, Universidade Federal de Pernambuco, 1999.
- [ALE99b] Alencar, F. e Castro, J.: *Integrating Early and Late-Phase Requirements: A Factory Case Study*. XIII Simpósio Brasileiro de Engenharia de Software – SBES'99, 1999.
- [ALE00] Alencar, F., Castro, J., Cysneiros, G., Mylopoulos, J.: *From Early Requirements Modeled by i* Technique to Later Requirements Modeled in Precise UML*. In Proceedings of the III Workshop de Engenharia de Requisitos, Rio de Janeiro, Julho, 2000, pp 92-108.
- [BOE89] Boehm, B. W, *Software Risk Management*, IEEE Computer Society Press, Washington, 1989.
- [BOM97] Boman, M., Bubenko, J., Johannesson, P. and Wangler, B.: *Conceptual Modeling*, Prentice Hall Series in Computer Science. 1997.
- [BOO93] Booch, G., *Object-Oriented Analysis and Design with Applications*, 2nd edition. Redwood City, CA: The Benjamin/Cummings Publishing Company, 1993.
- [BOO99] Booch, G. Rumbaugh, J. Jacobson, I., *The Unified Modeling Language User Guide*, Addison Wesley, 1999.
- [BRA00] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., *Extensible Markup Language (XML) 1.0*, 2nd Ed., W3C, 2000.
- [BRO87] Brooks, F., *No Silver Bullet: Essence and Accidents of Software Engineering*, IEEE Computer, Vol. 20 No. 4, April 1987, pp. 10-19.
- [BUB83] Bubenko, J., *On Concepts and Strategies for Requirements and Information Analysis*. In Information Modelling. Chartwell-Bratt, 1983.
- [BUB95] Bubenko, J., *Challenges in Requirements Engineering*, Second IEEE International Symposium on Requirements Engineering, IEEE Computer Society Press. York, England, 1995.
- [CAS95] Castro, J., *Introdução a Engenharia de Requisitos*, XV Congresso da Sociedade Brasileira de Computação, XIV Jornada de Atualização em Informática - JAI 95. Canela, RS, 1995.
- [CAS00a] Castro, J., Alencar, F., Cysneiros, G., *Closing the Gap between Organizational Requirements and Object Oriented Modeling*, XIV Simpósio Brasileiro de Engenharia de Software, 2000.
- [CAS00b] Castro, J., Alencar, F., Cysneiros, G., *Closing the GAP between organizational Requirements and object oriented modeling*, Journal of the Brazilian Computer Society, v.7, n.1, 2000.

- [CAS00c] Castro, J., Kolp, M. and Mylopoulos, J.: *Developing Agent-Oriented Information Systems for the Enterprise*, Proceedings of the Second International Conference on Enterprise Information Systems - ICEIS00, Stafford, UK, July, 2000.
- [CAS01a] Castro, J., Alencar, F., Cysneiros, G., *Integrating Organizational Requirements and Object Oriented Modeling*. In: Fifth International Symposium on Requirements Engineering - RE'01, Toronto. 2001.
- [CAS01b] Castro, J., kolp, M. and Mylopoulos, J. *Towards Requirements-Driven Information Systems Engineering*, Information Systems, Elsevier, Amsterdam, The Netherlands, 2001.
- [CAS01c] Castro, J., Kolp, M. and Mylopoulos, J. *A Requirements-Driven Development Methodology*. In: CAISE'01 The 13Th Conference on Advanced Information Systems Engineering, 2001, Interlaken, Suíça. Proceedings of the 13th Conference on Advanced Information Systems Engineering, Springer Lecture Notes in Computer Science - LNCS 2068, 2001.
- [CHU00] Chung, L. Nixon, B. Yu, E. and Mylopoulos, J., *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publisher, 2000.
- [COA90] Coad, P., Yourdon, E., *Object-Oriented Analysis*, Prentice Hall, 2nd edition, 1990,
- [COH90] Cohen, P., Levesque, H., *Intention is choice with commitment*, Artificial Intelligence, 1990.
- [COL93] Coleman D., *Object-oriented development: The Fusion Method*, Prentice-Hall, 1993.
- [CON00] Conallen, J. *Building Web Applications with UML*. Addison-Wesley, 2000.
- [COO94] Cook, S., Daniels, J.. *Designing Object Systems: Object-Oriented Modelling with Syntropy*, Prentice-Hall, 1994.
- [DAM97] Damian, A., Hong, D. and Li, H., *Joint Application Development and Participatory Design*. Disponível em www.cpsc.ucalgary.ca/~pan/seng/613/report.html
- [DAR93] Dardenne, A, van Lamsweerde, Fickas, S., *Goal Directed Requirements Acquisition*, Science of Computer Programming, vol. 20, pp. 3-50, Apr., 1993.
- [DAV92] Davis, A. *Operational Prototyping: A New Development Approach*, Software, 1992.
- [DAV93] Davis, A., *Software Requirements: Objects, Functions, and States*, PTR Prentice Hall, Englewood Cliffs, New Jersey, USA, 1993.
- [DEM78] DeMarco T., *Structured Analysis and System Specification*, Yourdon Press, New York, USA, 1978.
- [DOB89] Dobson, J., McDermid, J., *Security Models and Enterprise Models*. In C. E. Landwehr (Ed.), Database Security: Status and Propects II, Elsevier. 1989.
- [DOB92] Dobson, J. E.: *Information and Denial of Service*. In C. E. Landwehr and S. Jajodia (Eds.), Database Security: Status and Propects V. Elsevier. 1992.
- [DOB94a] Dobson, J. E and Strems, R.: *Organizational Requirements Definition for Information Technology*, IEEE International Conference on Requirements Engineering, IEEE Computer Society Press, April 1994.
- [DOB94b] Dobson, J. E, Blyth, A.J. C., Chudge, J. and Strems, R., *The ORDIT approach to organisational requirements*. In M. Jirotko and J. Goguen (Eds.), Requirements Engineering, Academic Press Ltd, London, 1994.
- [EAS97] Eason, K., Harker, S., Olphert, W., *Working with Users to Generate Organisational Requirements: The ORDIT Methodology*. ICL Systems Journal.Vol. 11. Iss.2. Jan. 1997.
- [ERI00] Eriksson, H. e Penker, M., *Business Modeling with UML: business patterns at work*, John Wiley & Sons, 2000.

- [ESI96] European Software Institute, *European User Survey Analysis*, Report USV_EUR 2.1, ESPITI Project, January 1996.
- [FOW96] Fowler, M., *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 1996.
- [GRE84] Greenspan, S.J., *Requirements Modelling: A Knowledge Representation Approach to Software Requirements Definition*, Ph. D. Thesis, Dept. of Comp. Sci., Univ. of Toronto, 1984.
- [GRL02] Goal-oriented Requirement Language, <http://www.cs.toronto.edu/km/GRL/>
- [HAU88] Hauser J. and Clausing, D.: *The House of Quality*, The Harvard Business Review, vol 3. 1988.
- [HOF93] Hofman, H., *Requirements Engineering: A Survey of Methods and Tools*, Institut für Informatik der Universität Zürich, Nr. 93.05, 1993.
- [IEE84] IEEE Std. 830, *IEEE Guide to Software Requirements Specification*, The Institute of Electrical and Electronics Engineers, New York, EUA, 1984.
- [IEE97] IEEE, *IEEE Software Engineering Standards Collection*, Computer Society Press, 1997.
- [JAC92] Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G., *Object-Oriented Software Engineering—A Use Case-Driven Approach*, Addison-Wesley, 1992.
- [JAC95] Jackson, M., *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices*, 1ed. Addison-Wesley, Massachusetts, USA, 1995.
- [JAC99] Jacobson, I., Booch, G., Rumbaugh, J., *The Unified Software Development Process*, Addison-Wesley, 1999.
- [KAR98] Karlsson, K. *A Systematic Approach for Prioritizing Software Requirements*, Department of Computer and Information Science, Linköping University, Suécia, 1998.
- [KIR01] Kirzner, R., *Analysis, Modeling, and Design Tools Market Forecast and Analysis, 2001–2005*, IDC #24809, 2001.
- [KIR94] Kirikova, M., Bubenko, J., *Software Requirements Acquisition through Enterprise Modelling*, Stockholm University. Sweden. 1994.
- [KML02] The Knowledge Management Lab, <http://www.cs.toronto.edu/km/>
- [KOT97] Kotonya, G., Sommerville, I.: *Requirements Engineering: Processes and Techniques*, John Wiley & Sons Ltd, 1997.
- [KOU88] Koubarakis, M., *An Implementation of CML*, MSc. Thesis, Department of Computer Science, University of Toronto, 1988.
- [KOU89] Koubarakis, M., Mylopoulos, J., Stanley, M., Jarke, M., *Telos: Feature and Formalization*, KRR-TR-89-4, Department of Computer Science, University of Toronto, 1989.
- [KRU99] Kruchten, P.: *The Rational Unified Process*. Addison-Wesley, EUA, 1999.
- [LAM95] Lamsweerde, A. van, Darimont, R. and Massonet, P.: *Goal-Directed Elaboration of Requirements for a Meeting Scheduler Problems and Lessons Learnt*. Proc. 2nd IEEE Symposium on Requirements Engineering. pp. 194-203. York. March, 1995.
- [LAM98] van Lamsweerde, A., Darimont, R. and Letier, E.: *Managing Conflicts in Goal-Driven Requirements Engineering*, IEEE Transaction on Software Engineering, Special Issue on Inconsistency Management in Software Development, November, 1998.
- [LAM00a] Lamsweerde, A. van, *Requirements Engineering in the Year 00: A Research Perspective*, 22nd Proceedings of International Conference on Software Engineering, Limerick, Ireland. Jun. 2000.

- [LAM00b] Lamsweerde, A. van, Letier, E., *Handling Obstacles in Goal-Oriented Requirements Engineering* IEEE Transactions on Software Engineering, *Special Issue on Exception Handling*, Vol. 26, September 2000.
- [LAR01] Larman, C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design, and the Unified Process*. Prentice Hall. 2nd edition
- [LEF99] Leffingwell, D., Widrig D., *Managing software requirements: a unified approach*. Addison Wesley Longman, 1999.
- [LEI87] Leite, J., *A Survey on Requirements Analysis*, Technical Report, Department of Information and Computer Science, California, 1987.
- [LEI97] Leite, J., Rossi, G., Maiorana, V., Balaguer, F., Kaplan, G., Hadad, G., Oliveros, A.: *Enhancing a Requirements Baseline with Scenarios*, Third IEEE International Symposium on Requirements Engineering - IEEE Computer Society Press. Los Alamitos, Ca, USA. 1997. pp. 44 -53.
- [LEI00] Leite, J. *Engenharia de Requisitos*, 1ª ed. Brasil. (Notas de Aula), 1999.
- [LOU95] Loucopoulos, P. e Karakostas, V., *System Requirements Engineering*, McGraw-Hill, 1995.
- [MAC96] Macaulay, L., *Requirements Engineering*, Springer, London, 1996
- [MAI98] Maiden, N. *CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements*. Automated Software Engineering, 1998.
- [MAM99] Mamani, M., Leite, J., *Elicit@99 um Protótipo de Ferramenta para a Elicitacao de Requisitos*. Workshop de Engenharia de Requisitos II – WER99, Buenos Aires, 1999.
- [MAR95] Martin, J, Odell, J., *Object-Oriented Methods: A Foundation*, Prentice-Hall, 1995.
- [MOF00] Meta Object Facility (MOF) Specification, Version 1.3, March, 2000.
- [MYL90] Mylopoulos, J., Borgida, A., Jarke, M., Jarke, M., *Telos: Representing Knowledge About Information Systems*, ACM Transactions on Information Systems, October, 1990.
- [MYL92] Mylopoulos, J., Chung, L. and Nixon, B., *Representing and Using Non-Functional Requirements: A Process Oriented Approach*, IEEE Transaction on Software Engineering, vol. SE-18, nº . 6, pp. 483-497. June, 1992.
- [MYL95] Mylopoulos, J., *Capturing Intentions in Requirements Engineering: A modeling Perspective*. Invited Talk at the IFIP W2.9 meeting, Bramshill, UK, March, 1995.
- [MYL00] Mylopoulos, J., Castro, J., *Tropos: A Framework for Requirements-Driven Software Development*, Brinkkemper, J. and Solvberh, A. (eds), Information Systems Engineering: State of Art and Research Themes, Lectures Notes in Computer Science, Springer-Verlag, 2000.
- [NUS00] Nuseibeh, B. e Easterbrook, S., *Requirements Engineering: A Roadmap*, Proceedings of the 22nd International Conference on Software Engineering. Limerick, Ireland. Jun. 2000.
- [OMG00] *OMG XML Metadata Interchange (XMI) Specification*, Version 1.1, November 2000.
- [OMG01] *OMG Unified Modeling Language Specification*, Version 1.4, Setembro, 2001.
- [OMG02] Object Management Group, <http://www.omg.org>
- [OME02] <http://www.cs.toronto.edu/km/ome/>
- [ORD93] ORDIT Consortium: “ORDIT Process Manual”. HUSAT Research Institute, Loughborough University, Loughborough, UK. 1993.
- [PRE01] Pressman R. *Software Engineering: A Practioner's Approach*, 5th Edition, McGraw-Hill, 2001.

- [PRE02] <http://www.mhhe.com/engcs/compsci/pressman/information/olc/BPE.html>
- [QUA99] Quatrani, T. e Booch, G., Visual Modeling with Rational Rose 2000 and UML The Addison-Wesley Object Technology Series) 2nd edição, 1999.
- [RAT02] Rational Corporation, <http://www.rational.com>
- [REI01] Rose Extensibility User's Guide. Version: 2001A.04.00, Rational Corp, 2001.
- [ROS77] Ross, D. T. e Shoman, K.E., *Structured Analysis for Requirement Definition*, IEEE Trans. Soft. Eng., Vol. SE-3, No. 1, Jan. 1977.
- [RYA93] Ryan, M. *Defaults in Specifications*. Proceedings of IEEE International Symposium on Requirements Engineering. IEEE Computer Society Press, EUA, 1993.
- [RUM91] Rumbaugh, J., Blaha M., Premerlani, W., Eddy, F., Lorensen, W., *Object-Oriented Modeling and Design*, Englewood Cliffs, NJ: Prentice Hall, 1991.
- [RUM99] Rumbaugh, J., Jacobson, I. and Booch, G.: *Unified Modeling Language Reference Guide*. Rational Software Corporation. Addison-Wesley Object Technology Series. Jan., 1999.
- [SAA90] Saaty, T. *The Analytic Hierarchy Process*, McGraw-Hill, 1990.
- [SAN00] Santander, V., Castro, J., *Desenvolvendo Use Cases a Partir de Modelagem Organizacional*, III Workshop de Engenharia de Requisitos - WER, Brasil, 2000.
- [SAN01] Santander, V., Castro, J., *Developing Use Cases from Organizational Modeling*, IV Workshop de Engenharia de Requisitos - WER, Argentina, 2001.
- [SCH98] Schneider, G., Winters, J. *Applying Use Cases: a practical guide*. Addison-Wesley. 1998.
- [SHL88] Shlaer, S., Mellor, J., *Object-Oriented Systems Analysis: Modeling the World in Data*. Yourdon Press Computing Series, 1998.
- [SHA96] Shaw, M. e Gaines, B. *Requirements Acquisition*, Software Engineering Journal, vol 11, 1995.
- [SHA99] Sharp, H. Finkelstein, A. e Galal, G. *Stakeholder Identification in the Requirements Engineering Process*. Workshop on Requirements Engineering Process. Florença, Itália. 1999.
- [SOM00] Sommerville, I. *Software Engineering*. Sixth Edition. Addison-Wesley. 2000.
- [STA86] Stanley, M., CML: A Knowledge Representation Language with Applications to Requirements Modelling, MSc. Thesis, Department of Computer Science, University of Toronto, 1986.
- [STA95] The Standish Group, "Software Chaos", <http://www.standishgroup.com/chaos.html>.
- [TEC02] Technology Builders. <http://www.tbi.com>
- [TEL02] Telelogic Corp. <http://www.telelogic.com>
- [TOR99] Toranzo, M., Castro, J. *A Comprehensive Traceability Model to Support the Design of Interactive Systems*, International Workshop on Interactive System Development and Object Models - WISDOM 99. Lisboa, Portugal, 1999.
- [USI01] *Using Data Modeler*, Version: 2001A.04.00, Rational Corp, 2001.
- [VIL99] Viller, S. and Sommerville, I. *Social Analysis in the Requirements Engineering Process: from ethnography to method*. 4th International Symposium on Requirements Engineering, Ireland, 1999.
- [WAR99] Warmer, J., Kleppe, A., *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley Object Technology Series. March, 1999.
- [WEI96] Weiringa, J., *Requirements Engineering – Frameworks for Understanding*. New York, John Wiley e Sons Ltd. 1996.

- [WIE98] Wieringa, R., Dubois, E., Huyts, S., *Integrating Semi-Formal and Formal Requirements*, In Proceedings of the 7th Int. Conference on Advanced Info. Systems Engineering – CaiSE’97, Barcelona, Spain, 1997.
- [WIE99] Wiegers, K. *Software Requirements*. Microsoft Press. 1999.
- [WIR90] Wirfs-Brock R., Wilkerson, B., Wiener, L., *Designing Object-Oriented Software*. Prentice-Hall, 1990.
- [WOO96] Woodcock, J., Davies, Jim, *Using Z: Specification, Refinement, and Proof* Prentice-Hall, 1996.
- [YU93] Yu, E., *An organization modelling framework for information systems requirements engineering*. In Proc. Of the 3rd Workshop on Information Technologies and Systems WITS’93, Orlando FL, December 4-5, 1993.
- [YU94a] Yu, E., Mylopoulos, J.: *Understanding ‘Why’ in Software Process Modelling, Analysis and Design*, Sixteenth International Conference on Software Engineering. Sorrento, 1994.
- [YU94b] Yu, E., Mylopoulos, J., *From E-R to “A-R” – modelling strategic actor relationships for business process reengineering*. In Proc. Of the 13th International Conference on the Entity-Relationship Approach – ER’94, Manchester (UK), December 13-16, 1994.
- [YU95a] Yu, E., *Modelling Strategic Relationships for Process Reengineering*, Phd Thesis, University of Toronto, 1995.
- [YU95b] Yu, E., Bois, P., Mylopoulos, J., *From Organization Models to System Requirements – A Cooperative Information Systems*, CoopIS-95, Vienna (Austria), 1995.
- [YU97a] Yu, E.: *Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering*. Proceedings of IEEE International Symposium on Requirements Engineering - RE97, pp.226-235, Jan. 1997.
- [YU97b] Yu, Eric and Mylopoulos, J., *Modelling Organizacional Issues for Enterprise Integration*., ICEIMT’97 - International Conference on Enterprise Integration and Modelling Technology. Turin, Italy. October, 1997.
- [ZAV97] Zave, P., *Classification of Research Efforts in Requirements Engineering*, ACM Computer Surveys, vol 29, n° 4. 1997.