UNIVERSIDADE CANDIDO MENDES – UCAM CAMPOS
PROGRAMA PÓS-GRADUAÇÂO EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL
CURSO DE MESTRADO EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL

ERNANI GASPAR MARTINS CORDEIRO DOS SANTOS

REPRESENTAÇÃO DE DESIGN RATIONALE NA ENGENHARIA DE REQUISITOS COM A ABORDAGEM KUABA

CAMPOS DOS GOYTACAZES
Março de 2010

UNIVERSIDADE CANDIDO MENDES – UCAM CAMPOS
PROGRAMA PÓS-GRADUAÇÂO EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL
CURSO DE MESTRADO EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL

ERNANI GASPAR MARTINS CORDEIRO DOS SANTOS

REPRESENTAÇÃO DE DESIGN RATIONALE NA ENGENHARIA DE REQUISITOS COM A ABORDAGEM KUABA

Dissertação apresentada ao Programa de Mestrado em Pesquisa Operacional e Inteligência Computacional da Universidade Candido Mendes – Campos dos Goytacazes/RJ, para obtenção do grau de MESTRE EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL.

Orientadora: Profª. Adriana Pereira de Medeiros, D.Sc.

CAMPOS DOS GOYTACAZES, RJ
Março de 2010

ERNANI GASPAR MARTINS CORDEIRO DOS SANTOS

REPRESENTAÇÃO DE DESIGN RATIONALE NA ENGENHARIA DE REQUISITOS COM A ABORDAGEM KUABA

Dissertação apresentada ao Programa de Mestrado em Pesquisa Operacional e Inteligência Computacional da Universidade Candido Mendes – Campos dos Goytacazes/RJ, para obtenção do grau de MESTRE EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL.

Aprovada em 26 de Março de 2010.

BANCA EXAMINADORA

_____

Profª. Geórgia Regina Rodrigues Gomes, D. Sc.
Universidade Candido Mendes - Campos

_____

Prof.ª Adriana Pereira de Medeiros, D. Sc. - Orientadora
Universidade Federal Fluminense (UFF/PURO) – Rio das Ostras

_____

Profª. Aline Pires Vieira de Vasconcelos, D. Sc.
Instituto Federal de Educação, Ciência e Tecnologia Fluminense - Campos

_____

Prof. Daniel Schwabe, Ph.D.
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) - RJ

CAMPOS DOS GOYTACAZES, RJ
2010

Este trabalho é dedicado à minha família: minha muito amada esposa Vera Lúcia, mães (mamãe e Dedé), irmã, sobrinhos, meus amados filhos, Michelle e Gaspar, e enteados, Carlos Eduardo, João e Renata, e ao meu falecido pai, com quem aprendi, sem que nada me dissesse, como se dar aos filhos, mesmo que eles não saibam, e de quem sinto muitas saudades.

AGRADECIMENTOS

À minha orientadora, Adriana Pereira de Medeiros, que me apoiou nesta jornada, por quem tenho muito respeito e estima.

Às minhas professoras e meus professores, que me surpreenderam pelo interesse e dedicação, coisas com as quais eu não estava acostumado na vida acadêmica.

À Robert Darimont pela cessão da licença do produto Objectiver usado nesta dissertação.

À Will Heaven, Anthony Finkelstein e Emmanuel Letier da UCL, *University College London*, por sua contribuição na obtenção da licença e da versão EMF do meta-modelo KAOS e das representações visuais do modelo em UML.

À Christophe Ponsard do CETIC , *Centre of Excellence in Information and Communication Technologies*, que disponibilizou os modelos e ofereceu ajuda no entendimento das construções mais "enigmáticas".

E também à meus colegas da turma de 2007, Alextian Liberato, Carlos Eduardo Dutra, Igor Franco, Leandro Foly, Leonardo Pio e Marcelo Schuster, com os quais tive oportunidade de viver a vida de estudante, mesmo que extemporânea, da forma mais pura, aproveitando todas as emoções que dela adviram.

“The first step toward the management of disease was replacement of demon theories and humors theories by the germ theory. That very step, the beginning of hope, in itself dashed all hopes of magical solutions. It told workers that progress would be made stepwise, at great effort, and that a persistent, unremitting care would have to be paid to a discipline of cleanliness. So it is with software engineering today.”

Frederick Brooks em *No Silver Bullet*, 1987.

RESUMO

REPRESENTAÇÃO DE DESIGN RATIONALE NA ENGENHARIA DE REQUISITOS COM A ABORDAGEM KUABA

Especificações de requisitos realizadas de maneira pobre ou incorreta têm sido reconhecidas como fonte de problemas no desenvolvimento de software, fazendo da Engenharia de Requisitos uma atividade crítica nesse processo, uma vez que todo o desenvolvimento baseia-se no conhecimento obtido nessa etapa. O registro de *design rationale* nessa atividade pode levar os engenheiros de software a refletir melhor sobre os requisitos e as formas de modelá-los, uma vez que eles precisam avaliar cuidadosamente seus argumentos e justificativas para as suas especificações. A abordagem Kuaba para *design rationale* integra seu modelo de representação, definido pela ontologia Kuaba, com a semântica fornecida pelo metamodelo de design usado para descrever o artefato sendo projetado. As opções de design consideradas são registradas no *design rationale* de maneira consistente com a semântica do metamodelo utilizado, o que possibilita o uso desse *design rationale* em outros projetos. Este trabalho investiga a representação de *design rationale* para modelos de requisitos utilizando a abordagem Kuaba e o metamodelo KAOS. Além disso, apresenta uma análise preliminar das possibilidades de representação de relacionamentos entre o *design rationale* gerado durante a Engenharia de Requisitos e os *designs rationales* de modelos criados em outras atividades do desenvolvimento, como os modelos de análise e de projeto. A representação de *design rationale* com Kuaba permite melhoria na qualidade dos modelos de requisitos, e também apóia a validação da capacidade da semântica dos metamodelos em representar os conceitos desses modelos de forma precisa. Além disso, contribui para o gerenciamento das mudanças em requisitos dando semântica ao rastreamento de artefatos e apoiando as análises de impacto das mudanças.

PALAVRAS-CHAVES: Design Rationale; Representação de Conhecimento; Ontologia; Engenharia de Requisitos; Meta-modelo; KAOS; UML; Engenharia de Software.

ABSTRACT

DESIGN RATIONALE REPRESENTATION IN REQUIREMENTS ENGINEERING USING KUABA APPROACH

Specifications of requirements made in a poor or incorrect manner have been recognized as a source of problems in software development, making the Requirements Engineering activity critical in this process, since all development is based on knowledge gained in this activity. The recording of design rationale in this activity can lead software engineers to better reasoning about the requirements and how to model them, since they must carefully evaluate their arguments and justifications for their specifications. The Kuaba approach to design rationale integrates its representation model, defined by Kuaba ontology, with the semantics provided by the design meta-model used to describe the artifact being designed. The design options considered are recorded in the design rationale in a manner consistent with the semantics of the meta-model used, which allows the use of this design rationale in other projects. This work investigates the design rationale representation for requirements models using the Kuaba approach and the KAOS meta-model. Moreover, it shows a preliminary analysis of the possibilities of representing relationships between the design rationale recorded during the Requirements Engineering activity and the designs rationales of models created in other development activities, like analysis and design models. The design rationale representation using Kuaba allows improvement in the quality of the requirements model, and also supports the validation of meta-models' semantics ability to represent the concepts of the model accurately. It contributes to the requirement changes management by giving semantics to requirements tracing and supporting analysis of the changes impact.

KEYWORDS: Design Rationale; Knowledge Representation; Ontology; Requirements Engineering; Meta-model; KAOS; UML; Software Engineering.

ÍNDICE DE FIGURAS

ABREVIATURAS E SIGLAS

CML	Conceptual Modeling Language
DRL	Decision Representation Language
gIBIS	Graphical IBIS
GRL	Goal Representation Language
IBIS	Issue Based Information System
ITU-T	International Telecommunication Union
KAOS	Knowledge Acquisition in Automated Specification Keep All Objectives Satisfied
KSE	Kuaba Software Engineering – Ferramenta de captura de design rationale na modelagem de artefatos usando UML
OMG	Object Management Group
OOHDM	Object-Oriented Hypermedia Design Method
PHI	Procedurarl Hierarchy of Issues
QOC	Questions, Options and Criteria
REMM	Requiremets Engineering Meta-model
RML	Requirements Modeling Language
SIREN	Simple Reuse of Software Requirements
UCM	Use Case Map
UML	Unified Modeling Language – Linguagem de modelagem proposta pelo OMG
URN	User Requirements Notation
W3C	World Wide Web Comsortium

SUMÁRIO

1 INTRODUÇÃO

Muitas abordagens para *Design Rationale* têm sido usadas na Engenharia de Software desde o final dos anos 80, tais como a abordagem DRL (*Decision Representation Language*) de Lee e Lai (1991) e a abordagem RATSpeak de Burge e Brown (2003), dentre outras que serão discutidas neste trabalho. Independentemente da abordagem, *rationale* é uma justificativa que apoia decisões, abrangendo a descoberta e a formalização de conhecimento tácito (DUTOIT, 2006). A atividade de *design* pode ser entendida como a busca pela descrição adequada e suficientemente detalhada para um artefato, e *design rationale* como sendo a descrição do raciocínio empregado para a determinação do *design* desse artefato (MACCLEAN, 1996). Essa descrição normalmente inclui as explicações sobre as alternativas de solução para os problemas de design encontrados na modelagem, as razões por trás das decisões tomadas a respeito da alternativa que melhor resolve o problema e quais alternativas foram descartadas. Embora existam muitas abordagens diferentes para *design rationale*, sua captura e representação continuam sendo um desafio na Engenharia de Software, especialmente durante a Engenharia de Requisitos, devido à volatilidade dos conceitos dos domínios abordados, obtidos usualmente em declarações em linguagem natural, e seu alto nível de abstração.

A Engenharia de Requisitos enfoca a descoberta, avaliação, análise da evolução e a documentação de objetivos, funcionalidades, qualidades e restrições obtidas para sistemas que usam software intensivamente (LAMSWEERDE, 2009). Durante essa atividade o domínio fornece aos engenheiros de software aqueles conceitos puramente relacionados com o mundo real, onde o software irá operar, trazendo valor para seus usuários. Esses conceitos representam abstrações que devem ser expressas de uma maneira que preserve sua semântica. Os metamodelos, a partir dos quais os modelos são obtidos, geralmente atendem a esse requisito, quando são suficientemente ricos para representar essas abstrações, mantendo seu

significado e criando a ponte para a produção de artefatos dirigidos para a implementação. Na Engenharia de Requisitos o metamodelo fornece semântica para a representação de abstrações sobre entidades e associações relativas às necessidades dos interessados (*stakeholders*) no software em desenvolvimento, sejam elas funcionais ou não funcionais, que levarão ao Modelo de Requisitos.

Geralmente, a semântica fornecida pelo metamodelo para descrever os artefatos não é aproveitada nas abordagens para representação de *design rationale* relatadas na literatura, como é o caso de IBIS (*Issue Based Information System*) (KUNZ; RITTEL, 1970), PHI (*Procedural Hierarchy of Issues*) (MCCALL, 1991), QOC (*Questions*, *Options* and *Criteria*) (MCCLEAN, 1991) e TEAM (LACAZE, 2005, 2006). Os conteúdos das representações de *design rationale* geradas a partir dessas abordagens são, em sua maioria, informais e incompletas. Isso impede o processamento computacional desse tipo de conhecimento e seu uso para apoiar o design de novos artefatos. Nessas abordagens o conhecimento aplicado no design não é representado de maneira padronizada, dado que não incorporam a semântica do artefato produzido. O *design rationale* registrado também não possui propriedades comuns, pois não é usada uma taxonomia comum em sua descrição, impossibilitando seu uso de maneira comparativa, uma vez que seus conteúdos, expressos pelos conceitos de cada taxonomia usada, não são formalmente equivalentes. Nesse caso, é apenas possível usar o *design rationale* registrado para cada modelo, isoladamente. Além disso, IBIS e QOC, por exemplo, não possuem em seus vocabulários elementos específicos para registrar as decisões tomadas ao longo da atividade de design e suas justificativas. Por isso, neste trabalho é usada a abordagem Kuaba[1] (MEDEIROS, 2006) para a representação de *design rationale*.

Kuaba é uma abordagem para *design rationale* que integra seu modelo de representação, definido pela ontologia Kuaba (MEDEIROS; SCHWABE; FEIJÓ, 2005a, 2005b), com a semântica fornecida pelo metamodelo de design usado para descrever o artefato sendo projetado. O uso dessa semântica em uma representação formal de *design rationale* permite que inferências e operações computáveis possam ser executadas para apoiar o uso do conhecimento registrado no design de novos artefatos (MEDEIROS, 2006). Além disso, permite a padronização da nomenclatura usada pelos engenheiros de software na modelagem de seus artefatos. As opções de design consideradas são registradas no *design rationale* de maneira consistente com a semântica do metamodelo utilizado, o que possibilita o uso desse *design rationale* em outros projetos.

[1] Kuaba significa “conhecimento” na língua Tupi-Guarani, um dos povos nativos do Brasil.

Na Engenharia de Requisitos, o registro de *design rationale* pode levar os engenheiros de software a refletir melhor sobre os requisitos e as formas de modelá-los, uma vez que eles precisam avaliar cuidadosamente seus argumentos e justificativas para as suas especificações. Desta forma, o registro de *design rationale* utilizando a abordagem Kuaba pode contribuir para a melhoria dos artefatos produzidos.

A abordagem Kuaba foi usada, até o momento, para a representação de *design rationale* de modelos conceituais (análise), usando no design o metamodelo UML (OMG, 2001), e de modelos de navegação de aplicações Web (projeto), usando o metamodelo OOHDM (*Object Oriented Hypermedia Design Method*) (SCHWABE e ROSSI, 1998). Requisitos são sabidamente voláteis, fruto de um processo de negociação inerentemente difícil, pois representam interesses usualmente distintos, declarados de maneira ambígua ou conflitante. Por isso, a representação de *design rationale* durante a modelagem de requisitos pode ser uma tarefa mais difícil que na modelagem conceitual ou no projeto de navegação de um aplicação, onde o nível de abstração e suscetibilidade a mudanças é geralmente mais baixo.

Especificações de requisitos realizadas de maneira pobre ou incorreta têm sido reconhecidas como fonte de problemas no desenvolvimento de software (LAMSWEERDE, 2009), fazendo da Engenharia de Requisitos uma atividade crítica nesse processo, uma vez que todo o desenvolvimento baseia-se no conhecimento obtido nessa etapa. Grande parte dos elementos dos modelos gerados durante o desenvolvimento de software são extraídos dos artefatos produzidos durante essa atividade, traduzidos na especificação de requisitos. Este trabalho investiga a representação de *design rationale* para modelos de requisitos utilizando a abordagem Kuaba e apresenta uma análise, preliminar, das possibilidades de representação de relacionamentos entre o *design rationale* gerado durante a Engenharia de Requisitos e os *design rationales* de modelos criados em outras atividades do desenvolvimento, como os modelos de análise e de projeto.

No Capítulo 2 desta dissertação são apresentadas e discutidas especificamente as abordagens de *design rationale* na Engenharia de Requisitos. Uma fundamentação teórica sobre a abordagem Kuaba para *Design Rationale* é apresentada no Capítulo 3. No Capítulo 4 são estudados e comparados os metamodelos geralmente usados na Engenharia de Requisitos e no Capítulo 5 é feito um estudo mais aprofundado do metamodelo KAOS, utilizado neste trabalho. No Capítulo 6 é desenvolvido o objeto da pesquisa, sendo apresentados e discutidos os exercícios de representação de *design rationale* para o domínio de uma biblioteca e para um exemplo do domínio de submissão de artigos para conferências científicas. No Capítulo 7

é apresentada uma análise preliminar das possibilidades de representação de relacionamentos entre os *design rationales* gerados na Engenharia de Requisitos e em outras atividades do processo de desenvolvimento de software. Conclusões sobre a pesquisa, suas contribuições e trabalhos futuros são abordados no Capítulo 8.

2 DESIGN RATIONALE NA ENGENHARIA DE REQUISITOS

A aplicação de *design rationale* na Engenharia de Software e na Engenharia de Requisitos tem sido uma área de pesquisa relativamente ativa nas últimas duas décadas (MORAN; CARROLL, 1996; DUTOIT, 2006). Entretanto, a complexidade e a abrangência do tema implicaram em uma evolução limitada no período, refletindo as dificuldades e o tamanho do espaço de pesquisa. Especificamente na Engenharia de Requisitos, a maioria dos trabalhos está mais focada nos processos de negociação e mudanças em requisitos do que propriamente nas técnicas usadas na especificação dos próprios requisitos. Ou seja, aparentemente, o registro dos elementos que levaram a certas decisões no processo de negociação e atividades iniciais do desenvolvimento do produto de software é o aspecto que tem guiado a atividade de pesquisa.

O design de um artefato de software é um processo de decisão onde os parâmetros das negociações podem ser relativos ao conhecimento e comportamento humanos, ou relativos a possibilidades oferecidas por recursos técnicos e tecnológicos. Os produtos finais dessa atividade, o artefato e sua especificação, representam apenas a solução final escolhida, que é somente parte do conhecimento aplicado pelos engenheiros de software no seu trabalho. Esta solução não mostra o raciocínio que culminou em uma escolha dentre alternativas, que fez os engenheiros de software optarem por uma, e descartarem outras. A solução final não representa o *design rationale* usado no design. *Design rationale* inclui todo o conhecimento usado no design incluindo as razões pelas quais as decisões foram tomadas (MEDEIROS, 2006).

2.1 ABORDAGENS ORIENTADAS PARA O PROCESSO DE NEGOCIAÇÃO

Boehm e Kitapci (2006) apresentam uma abordagem para a captura e gestão de *rationale* voltada para apoiar o processo onde os requisitos de software e de sistemas, assim como as soluções propostas, são negociadas. O modelo de processo *WinWin*, apoiado na Teoria W (BOHEM, 1996), o modelo de equilíbrio (Modelo Espiral *WinWin*) e a ferramenta colaborativa de negociação (*EasyWinWin*) fornecem condições para a captura dos artefatos resultantes.

A Teoria W é a base da abordagem *WinWin*. Seu princípio fundamental é que a condição necessária e suficiente para o sucesso de um empreendimento é tornar todos os interessados (*stakeholders*) considerados críticos para o sucesso "vencedores". Essa condição é considerada muito importante no processo de software, uma vez que envolve pessoas de forma intensiva. Embora tornar todos "vencedores" possa parecer inexequível, existem situações do tipo *win-win* (onde todos "vencem") que usualmente podem ser criadas através da atenção cuidadosa aos interesses e expectativas das pessoas. Os melhores resultados foram conseguidos no campo da negociação (BOHEM; KITAPCI, 2006).

Uma negociação de sucesso não pode ser conseguida a partir da discussão baseada em posições pré-estabelecidas, mas a partir de um processo de quatro passos, sugerido por Fisher e Ury (1981), cujo objetivo é criar situações onde todos vencem: *(1)* separar as pessoas dos problemas; *(2)* focar em interesses, não em posições; *(3)* criar opções para ganhos mútuos e *(4)* insistir no uso de critérios objetivos. A abordagem da Teoria W para o gerenciamento de software expande os quatro passos propostos estabelecendo um conjunto de pré-condições do tipo *win-win* e condições adicionais para a estruturação do processo de software e do produto de software resultante.

As atividades chaves do modelo de negociação WinWin são: *(a)* a indentificação dos interessados que são críticos para o sucesso; *(b)* a descoberta das condições primárias para "vencer" desses interessados críticos para o sucesso; *(c)* a negociação de pacotes de situações "vencer-vencer", tais como, requisitos, arquiteturas, planos e componentes críticos, mutuamente satisfatórios; *(d)* monitoramento e controle quantitativo do equilíbrio no processo de desenvolvimento.

O modelo possui quatro artefatos conceituais principais mostrados na Figura 2-1: *(1)* CONDIÇÃO "VENCER-VENCER", capturando os objetivos e restrições desejados pelos interessados; *(2)* ASSUNTO, capturando o conflito entre as condições e os riscos e incertezas a elas associados; *(3)* OPÇÃO, capturando uma decisão escolhida para resolver um assunto; e

(4) ACORDO, capturando o conjunto de condições acordadas que satisfazem as condições dos interessados e/ou capturando opções para resolver assuntos.

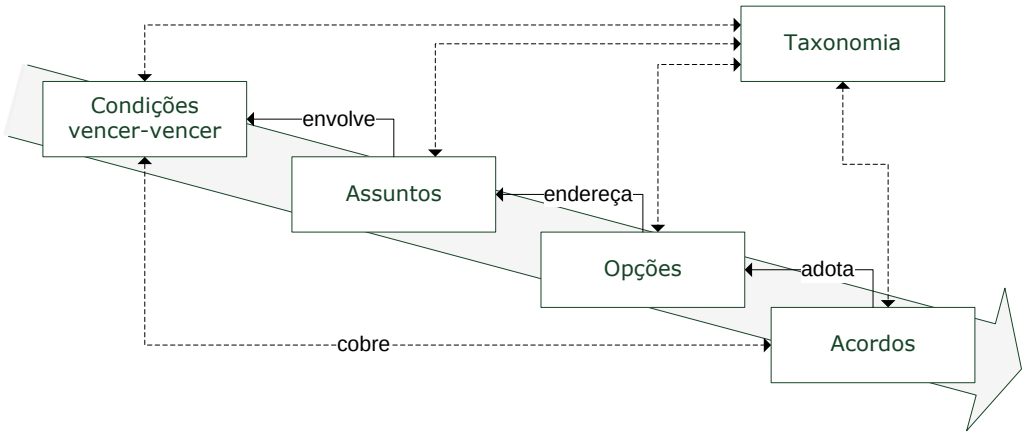

Figura 2-1. Modelo de negociação *WinWin* (BOHEM; KITAPCI, 2006).

Os interessados expressam seus objetivos através de condições "vencer-vencer". Havendo concordância de todos, significa que tal condição tornou-se um acordo, caso contrário, são identificadas tais condições conflitantes e assuntos são registrados para elas. Opções são criadas como possibilidades de benefícios mútuos e de troca. Essas opções são negociadas interativamente buscando tranformá-las em acordos quando há unanimidade. Neste processo os assuntos em aberto são considerados riscos para o projeto ou conflitos que precisam ser resolvidos.

Ao longo desse processo, obtem-se a taxonomia do domínio, como também ilustrado na Figura 2-1. Essa taxonomia é usada para organizar os artefatos e para a formação de um glossário com o vocabulário associado ao domínio. O modelo de negociação tem como objetivo a coordenação das atividades decisórias realizadas pelos interessados no processo de desenvolvimento do produto de software.

A metodologia e a ferramenta *EasyWinWin* para a negociação de requisitos complementam a abordagem oferecendo um processo que guia o desenvolvimento do trabalho e o uso de uma ferramenta de registro. A ferramenta oferece um ambiente concorrente e colaborativo para a realização da negociação. Isso apoia os interessados na obtenção de informação sobre os pontos de vistas dos demais, potencializando seu envolvimento e interação.

A abordagem *EasyWinWin* já foi usada em mais de cem projetos em vários domínios como bibliotecas digitais, comércio eletrônico, e tecnologias de colaboração (BOHEM; KITAPCI, 2006). Os autores defendem que a abordagem ajuda os interessados na priorização

dos requisitos e no registro de conhecimento sobre suas decisões, implicando em benefícios significativos em todas as fases do desenvolvimento, melhorando também a contextualização para análises de impacto em uma realidade em que alterações em requisitos são constantes e em número crescente. Além disso, referenciam qualidades de abrandamento do esforço na captura de *rationale* através do uso compartilhado da ferramenta *EasyWinWin* pelos interessados.

A descoberta de requisitos para grandes sistemas, com múltiplos interessados, significantes incertezas tecnológicas e prazos estendidos são tratados na abordagem híbrida *Wisdom* de Roobsky, Sommerville e Pidd (2006), extraída da Pesquisa Operacional e das técnicas de *Design Rationale* da Engenharia de Software. Os autores mostram um processo híbrido e uma ferramenta de apoio visando facilitar o consenso na definição de problemas. Nessa abordagem, a negociação ocorre através da combinação de grupos informais de estruturação de problemas, que usam os mapas cognitivos de Akerman, Eden e Cropper (1992) e Eden e Akerman (2001), e do formalismo incremental de requisitos através de mapas de diálogo de IBIS (KUNZ; RITTEL, 1970). O foco da abordagem é no que os autores chamam de "estágio inicial rumo à nascente (do rio)" (*early upstream stage*), onde as opções e compromissos estão apenas começando a emergir.

Um Mapa Cognitivo, como exemplificado na Figura 2-2, é um conjunto de nós representando conceitos expressados por pessoas envolvidas na discussão de um tema e relações direcionadas que associam esses conceitos. Essas associações possuem uma semântica abrangente podendo significar implicação, dedução, influência, ou seja, qualquer tipo de associação de um conceito para outro. O método não oferece taxonomia específica para os tipos de relacionamentos possíveis. A ilustração reflete parcialmente uma discussão de um grupo de pessoas sobre como um produto de software poderia ser melhor desenvolvido.

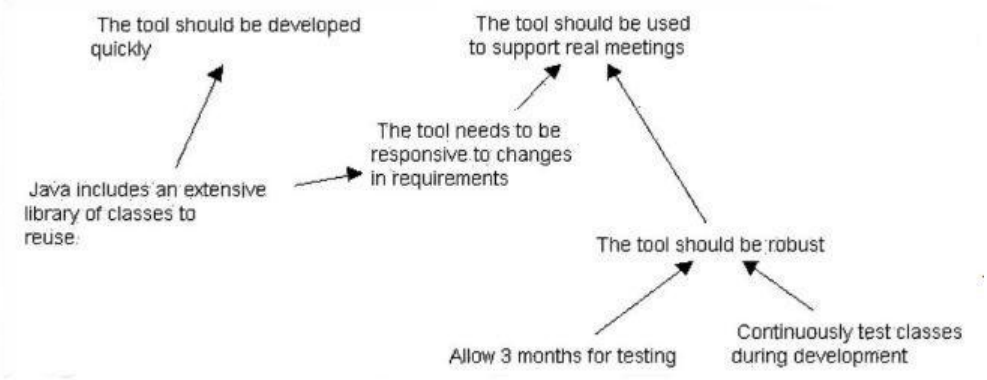

Figura 2-2. Exemplo de um Mapa Cognitivo (MACKENZIE, 2006).

Rooksby (2006) propõe o uso desse tipo de mapa com o objetivo de suprir uma estrutura inicial aos conceitos abordados no tratamento dos problemas, formando um conjunto estruturado de conceitos e associações sobre o qual há consenso. Essa etapa caracteriza a definição do espaço do problema. A partir daí, a proposta é usar um mecanismo eficaz para apoiar o processo de negociação partindo de conceitos específicos que foram definidos na etapa anterior. Nessa estapa são usados os Mapas de Diálogo da abordagem de *design rationale* IBIS.

A Figura 2-3 ilustra um exemplo de tal mapa cujo objetivo é registrar os possíveis "diálogos" que sucedem durante a negociação sobre os conceitos. IBIS é uma abordagem baseada em argumentação, tendo como elementos de sua linguagem perguntas (mostradas na ilustração como um ponto de interrogação), idéias (mostradas como lâmpadas) e argumentos representados por sinais de mais ou menos dependendo se são a favor ou contra as idéias, respectivamente. O mapa inicia com a formulação de uma pergunta que é respondida por idéias que têm argumentação contra ou favor. Nesse caso, o tema é um conceito ou assertiva do mapeamento da fase anterior, como o aumento da participação de uma empresa no mercado que no mapa de diálogo assume a forma da pergunta na sua raiz de Como aumentar nossa participação no mercado? (*How do we increase our market share?*).

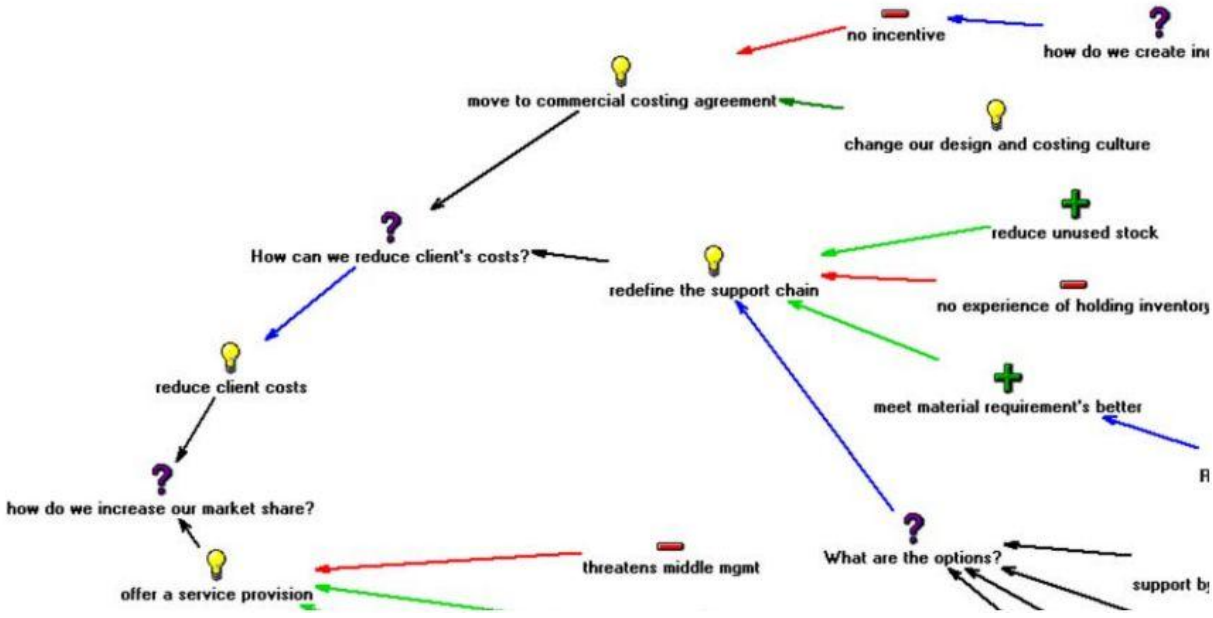

Figura 2-3. Exemplo de Mapa de Diálogo de IBIS (MACKENZIE, 2006).

A abordagem *Wisdom* (ROOKSBY, 2006) faz uso desses dois tipos de mapas, Mapa Cognitivo e Mapa de Diálogo, integrados em uma ferramenta de mesmo nome. *Wisdom* foi utilizada em dois tipos de projetos de grande porte e se mostrou, segundo seus criadores, capaz de estruturar e revelar o conhecimento e as razões por trás das decisões tomadas. Um ponto forte também ressaltado pelos autores é que essa abordagem, diferentemente daquela de Bohem e Kitapci (2006), *WinWin*, leva em consideração todos os aspectos relevantes do problema, sejam eles positivos ou negativos.

2.2 ABORDAGENS DE DESIGN RATIONALE PARA REQUISITOS

A proposta de gIBIS (CONKLIN; BEGEMAN, 1988) é fornecer uma ferramenta colaborativa com uma interface gráfica para a abordagem IBIS contribuindo para a diminuição do impacto da atividade de captura de *design rationale* no esforço de design. A pesquisa realizada por Conklin e Begeman (1988a) aborda o estudo de caso realizado durante um ano, com 32 pessoas usando a ferramenta sobre 33 grupos de assuntos, dentre os quais os de Análise Conceitual, Análise de Requisitos, Projeto de Software e Projeto de Interface Gráfica do Usuário, diretamente voltados para a Engenharia de Software e um particularmente para a Engenharia de Requisitos.

Nesse estudo foram feitas observações relevantes para a pesquisa conduzida nesta dissertação: *(a)* não há nó específico ou elemento de ligação para objetivos ou requisitos; *(b)* também não há apoio para o registro de decisões. Essas observações ressaltam que as proposições não podem representar idéias específicas da atividade de design de requisitos, assim como há limitação a respeito da representação da decisão do engenheiro de software sobre a solução escolhida.

Nguyen e Swatman, (2003, 2006) sugerem uma nova abordagem para *design rationale* suprindo as lacunas apontadas na sua análise das abordagens atuais em relação ao apoio à criatividade no processo de Engenharia de Requisitos. Essa abordagem envolve ciclos incrementais de construção seguidos de revisões conceituais guiadas pela introspecção a respeito do espaço do problema.

Os autores propõem a utilização de uma abordagem orientada para a exploração criativa tanto no levantamento como na análise e modelagem de requisitos, usando a abordagem IBIS complementada por uma abordagem que apoie a revisão e reestruturação conceitual do espaço do problema com base em QOC (*Questions, Options and Criteria*) (MACLEAN, 1991). A proposição do uso de abordagens de *design rationale* na Engenharia

de Requisitos vem da constatação em suas pesquisas que este pode oferecer aos engenheiros de software, notadamente de requisitos, e aos gerentes de projeto muitos benefícios potenciais relativos ao entendimento e monitoração do processo da Engenharia de Requisitos.

A abordagem contesta a teoria de vários autores como Kotonya e Sommerville (1997), Sommervile (2006) e Pressman (2009) de que a Engenharia de Requisitos é suavemente incremental, defendendo a idéia que os ciclos incrementais são interrompidos por reorganizações radicais no modelo de requisitos. Na experiência de sua pesquisa relatam que essas reorganizações ocorrem usualmente como consequência de revelações súbitas devido a introspecção (*insights*) e não devido ao esforço deliberado e sistemático.

Essa pesquisa é uma investigação em curso ainda em fase claramente teórica, uma vez que os autores não apresentam exemplos de sua aplicação nos trabalhos estudados para esta dissertação.

2.3 DISCUSSÃO

IBIS e QOC até agora foram a base para a representação de *design rationale* na Engenharia de Software. Como mostrado, Rooksby (2006) opta pela primeira, mesmo analisando QOC e DLR/SYBIL (LEE, 1991), entendendo que o enfoque IBIS, orientado para o apoio à discussão e à deliberação é mais adequado para seus propósitos.

IBIS e QOC são também bases para o trabalho de Nguyen e Swatman (2006) que propõem o aproveitamento dos pontos fortes de ambas em necessidades diferentes da atividade de Engenharia de Requisitos: uma abordagem orientada para a "exploração criativa" do levantamento de requisitos (IBIS), complementada pela caracterização conceitual do espaço do problema (QOC).

Este trabalho, embora tenha como espaço de aplicação a Engenharia de Requisitos, não tem similaridade com as abordagens apresentadas. Rooksby (2006) trata um momento no processo da Engenharia de Requisitos que se assemelha ao que Yu (1997a, 2001a) denomina fase de requisitos cedo (*early-phase requirements*). Essa parece ser a idéia de requisitos "rumo à nascente" (*upstream*), onde não há uma visão clara do probema, onde as opções e comprometimentos estão apenas emergindo. Nguyen e Swatman (2003, 2006) propõem uma abordagem que muda o paradígma da Engenharia de Requisitos levando em consideração momentos disruptivos no processo puramente incremental. Essa abordagem é apoiada pelo *design rationale* na elicitação, modelagem e especificação de requisitos. Entretanto, a pesquisa ainda não chegou a propor uma metodologia mais consolidada.

Esta dissertação, por outro lado, está focada na representação de *design rationale* em modelos de requisitos construídos com base na instanciação de um metamodelo, especificamente do modelo conceitual do *framework* KAOS. A preocupação não está voltada para o processo de negociação em si, ou para uma revisão mais radical do processo da Engenharia de Requisitos. O objetivo é investigar como as explicações sobre o design de requisitos (*design rationale*), ou seja, o conhecimento do engenheiro de software, manisfestado através do conhecimento do domínio, da semântica do metamodelo e sua experiência, podem contribuir para a prática da Engenharia de Requisitos.

3 A ABORDAGEM KUABA PARA REPRESENTAÇÃO DE DESIGN RATIONALE

Kuaba (MEDEIROS, 2006) é uma abordagem baseada em argumentação para representação de *design rationale*. Seu principal objetivo é permitir o processamento computacional de *design rationale* para apoiar o reuso de designs baseados em modelo, particularmente, designs de software. Design baseado em modelo é uma categoria de problemas de design que podem ser vistos como um processo de instanciação de um metamodelo. Esse metamodelo representa os modelos usados para descrever os artefatos produzidos. Um exemplo em design de software é o metamodelo UML (OMG, 2001, 2006a, 2006b) usado para descrever uma variedade de modelos da metodologia de desenvolvimento de sistemas orientados para objetos, dentre eles o modelo de classes. Neste trabalho, design refere-se à produção de um artefato de acordo com algum método, desde sua concepção até a sua realização como parte do software produzido.

A abordagem Kuaba difere das outras abordagens para *design rationale* propostas na literatura pelo fato de aproveitar a semântica fornecida pelos metamodelos de *design* para instanciar os elementos do seu modelo de representação, descrito na ontologia Kuaba (MEDEIROS; SCHWABE; FEIJÓ, 2005a, 2005b). O uso dessa semântica permite fornecer um suporte mais automatizado para o registro de *design rationale*, uma vez que grande parte de sua estrutura pode ser obtida através da análise do metamodelo utilizado. O uso da semântica do artefato também define um padrão para a representação de *design rationale* usando a ontologia Kuaba, permitindo que o *design rationale* gerado possa ser processado e combinado de forma automática para apoiar seu uso no design de novos artefatos (MEDEIROS, 2006).

A ontologia Kuaba é um modelo de representação de conhecimento para o domínio de *design rationale*. Seu vocabulário permite atribuir semântica ao conteúdo do *design rationale*

registrado e definir um conjunto de regras que possibilite a realização de inferências e operações computáveis para apoiar o seu uso. Kuaba estende a estrutura de argumentação da abordagem IBIS (*Issue Based Information System*) (KUNZ; RITTEL, 1970) para *design rationale*, que consiste em registrar "questões" que ocorrem durante o design, as "posições" que endereçam essas questões e os argumentos "contra" as posições, ou "a favor" delas. A extensão enriquece a estrutura de argumentação através da formalização da representação de decisões feitas durante o *design* e suas justificativas (MEDEIROS; SCHWABE; FEIJÓ, 2005a, 2005b). Essa extensão consiste também da integração da estrutura de argumentação com as descrições dos artefatos produzidos, através de propriedades históricas e descritivas desses artefatos. A Figura 3-1 mostra os elementos do vocabulário definido pela ontologia Kuaba usando a notação UML a fim de auxiliar a visualização. Nessa ilustração pode-se observar que os elementos de raciocínio, assim como as decisões, quem as tomou, qual o método de design usado, e outras propriedades são representados na ontologia Kuaba (MEDEIROS; SCHWABE; FEIJÓ, 2005a).

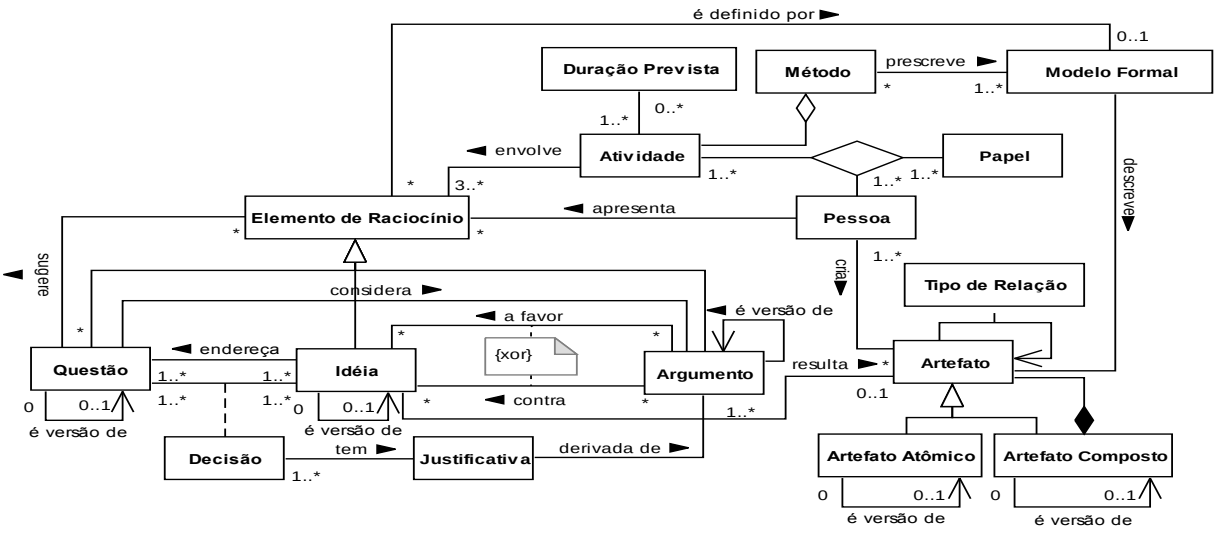

Figura 3-1. Elementos do vocabulário da ontologia Kuaba.

O design de um artefato de software envolve uma série de elementos de raciocínio. Estes elementos são usados pelo projetista na formulação de uma solução final para o problema de design que ele tem em mãos. De forma similar à notação IBIS, os elementos de raciocínio incluem as questões relacionadas ao design do artefato, as idéias de solução para essas questões e os argumentos contra ou a favor das idéias apresentadas. Cada elemento possui um conjunto de propriedades e relações que formam a estrutura do *rationale*

desenvolvido durante a atividade de *design*. A Figura 3-2 detalha a Figura 3-1 ilustrando as propriedades dos elementos de raciocínio que podem estar envolvidos numa atividade de *design* e as relações entre eles.

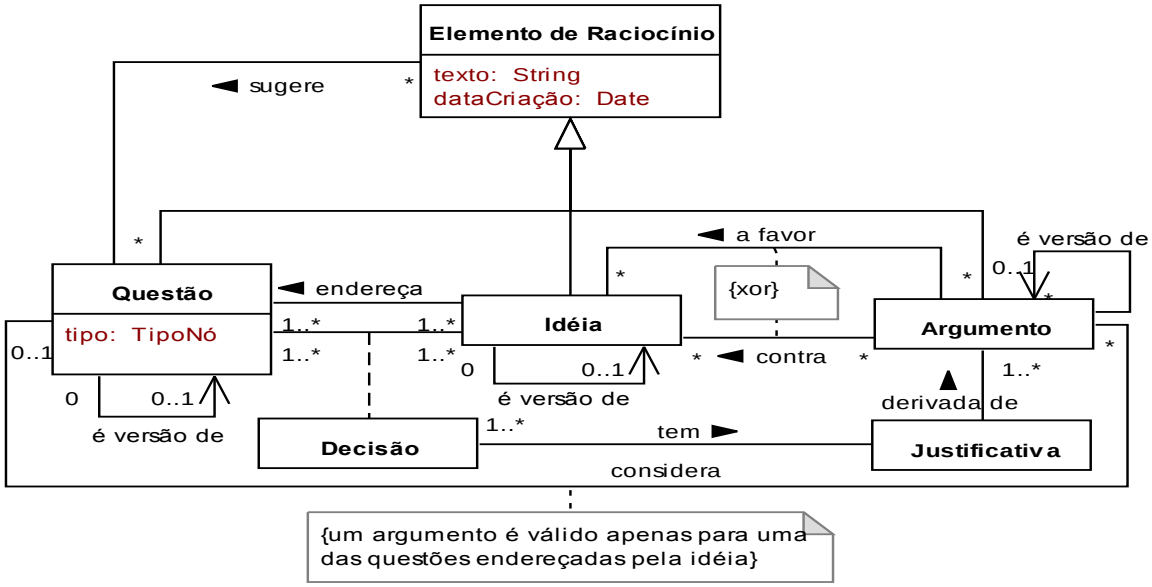

Figura 3-2. Elementos de raciocínio da ontologia Kuaba.

Uma questão representa um problema de *design* que precisa ser resolvido. Além do texto que indica qual problema deve ser tratado e a data de criação da questão, o elemento Questão possui também um atributo tipo, através do qual podemos definir se as idéias propostas como possíveis soluções para o problema devem ser consideradas mutuamente exclusivas ou não. Esta informação permite a definição de regras que podem ser usadas para sugerir decisões sobre a aceitação ou rejeição das idéias de solução apresentadas. TipoNó é uma enumeração que contém os possíveis valores para o atributo tipo: AND, OR ou XOR. O valor "XOR" indica que todas as idéias que respondem a questão são mutuamente exclusivas, ou seja, apenas uma destas idéias poderá ser aceita como solução para a questão. O valor "AND" indica que o projetista deve aceitar todas as idéias que respondem a questão ou rejeitar todas elas. Finalmente, o valor "OR" indica que várias idéias podem ser aceitas como solução para a questão.

Uma idéia representa uma possível solução, ou parte de uma solução de *design*, para o problema apresentado no elemento Questão. Um argumento representa uma razão contra ou a favor da adoção de uma idéia como uma solução para a questão que está sendo tratada. É através do elemento Argumento que os projetistas podem registrar o conhecimento que eles estão empregando no design do artefato.

Como pode ser observado na Figura 3-2, uma idéia responde a uma ou mais questões e deve possuir pelo menos um argumento associado a ela. Uma instância de Argumento só pode participar de uma das relações possíveis com uma dada instância do elemento Idéia, ou seja, contra, ou exclusivamente a favor. Isto é representado no modelo pela restrição {xor}. Caso um argumento seja válido apenas para uma das questões respondidas pela idéia à qual ele se refere, este argumento deve estar associado também à questão respondida pela idéia através da relação considera.

Durante o processo de *design* novas questões podem ser sugeridas a partir de um elemento de raciocínio (questão, idéia ou argumento). Estas questões indicam que outros problemas de *design* precisam ser resolvidos para que o projetista possa chegar a uma decisão sobre uma solução para a questão inicial. A sugestão de novas questões é representada no modelo pelas relações sugere. Estas relações também podem ser usadas para representar a decomposição de uma questão em questões mais simples, para facilitar a exploração de idéias sobre as possíveis soluções para o *design* do artefato a ser construído.

Outro tipo de relação ilustrada na Figura 3-2 é a relação é versão de, definida para os elementos Questão, Idéia e Argumento. Esta relação permite representar que um elemento de raciocínio foi aproveitado da representação do *rationale* de outro *design*. Esta representação pode ser de uma versão anterior do *design*, que está sendo usada para evoluir o artefato, ou de um *design* diferente que está sendo reusado em uma nova situação.

Durante a realização de uma atividade de *design*, uma pessoa ou um grupo pode decidir, com base nos argumentos apresentados, se uma idéia deve ou não ser aceita como uma solução para o *design* de um artefato. Na ontologia Kuaba, a aceitação ou a rejeição de uma idéia como uma solução para uma questão de design é registrada pelo elemento Decisão, como ilustrado na Figura 3-3. Diferente de outras notações para *design rationale*, nessa ontologia a aceitação ou a rejeição de uma idéia é representada como uma propriedade da relação entre os elementos Questão e Idéia. Considera-se que a aceitação ou rejeição de uma idéia não é uma propriedade intrínseca do elemento Idéia, mas deve ser definida com relação à determinada Questão, uma vez que a mesma idéia pode endereçar mais de uma questão, e pode ser aceita para uma, e ser rejeitada para outra.

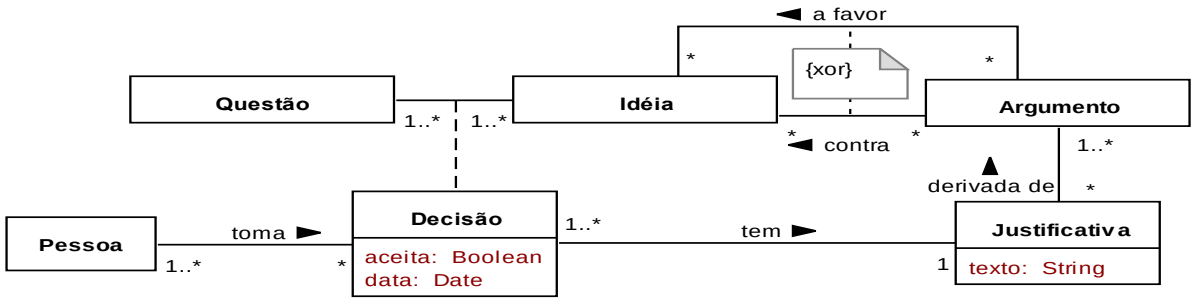

Figura 3-3. Detalhes do elemento Decisão e suas associações na ontologia Kuaba

A Figura 3-3 mostra também a propriedade aceita do elemento Decisão que registra a aceitação ou a rejeição de uma idéia, e a propriedade data que registra quando a decisão foi tomada pelo projetista. A partir dos valores atribuídos a esta propriedade será possível obter um histórico sobre a seqüência de decisões tomadas pelo projetista durante o *design*. Uma decisão deve possuir uma justificativa para a aceitação ou rejeição de uma idéia proposta como solução para uma determinada questão. Esta justificativa é sempre derivada de um ou mais argumentos apresentados durante a atividade de *design*.

Conhecer as razões que levaram à aceitação de alternativas, ou à sua rejeição, é importante. As alternativas aceitas trazem soluções válidas para determinado contexto de design. Embora as alternativas rejeitadas devam ser normalmente evitadas, por vezes, elas são uma má proposição em um contexto, podendo ser boas para outros propósitos em um contexto diferente. As soluções dependem do conjunto de requisitos, e nesse tipo de ambigüidade, notadamente de requisitos não funcionais. Embora a importância do registro de *design rationale* para a melhoria do processo de desenvolvimento de sistemas de software seja dada como um fato na literatura (DUTOIT, 2006), sua representação tem sido um desafio. A ontologia Kuaba foi proposta a fim de contribuir para a solução desse problema, sendo que neste trabalho a investigação é para a Engenharia de Requisitos.

A versão atual da ontologia Kuaba está descrita nas linguagens de representação de ontologias F-logic (KIFER; LAUSEN, 1989) e OWL (*Web Ontology Language*) (W3C, 2004). Na Listagem 3-1 são mostrados os elementos de raciocínio do vocabulário da ontologia Kuaba aqui abordados, descritos em F-Logic.

```
//================================================================
//  SCHEMA DATA: CLASSES & SIGNATURES
//================================================================

atomic_artifact::artifact.
composite_artifact::artifact.
question::reasoning_element.
idea::reasoning_element.
argument::reasoning_element.
reasoning_element[hasText*=>string,
                  hasCreationDate*=>string,
                  isInvolved*=>activity,
                  suggests*=>>question,
                  isPresentedBy*=>person,
                  isDefinedBy*=>formal_model].
question[hasType=>string,
         isAddressedBy=>>idea,
         isSuggestedBy=>>reasoning_element,
         isVersionOf=>question,
         hasDecision=>>decision].
idea[address=>>question,
     hasArgument=>>argument,
     results=>artifact,
     isVersionOf=>idea,
     isConcludedBy=>>decision].
argument[inFavorOf=>>idea,
         objectsTo=>>idea,
         considers=>question,
         isVersionOf=>argument].
decision[isAccepted=>boolean,
         hasDate=>string,
         isMadeBy=>>person,
         hasJustification=>justification,
         concludes=>idea].
justification[hasText=>string,
           isDerivedOf=>>argument].
```

Listagem 3-1. Código em F-Logic dos elementos de raciocínio da ontologia Kuaba (MEDEIROS, 2006)

Algumas representações de *design rationale* dessa dissertação apresentam uma legenda, mostrada na Figura 3-4, ilustrando a representação gráfica dos elementos de raciocínio da ontologia Kuaba, ilustrados na Figura 3-2. Essa representação é feita a partir de uma extensão da linguagem UML. As representações dos elementos de raciocínio Questão, Idéia e Argumento estendem o conceito de Classe da UML, com elementos gráficos modificados, assim como os relacionamentos entre eles. Os conceitos endereça e Decisão (aceitação ou rejeição), são extensões de associação direcionada, e a sugestão de questões, e os relacionamentos de argumentação contra ou a favor estendem associações de dependência.

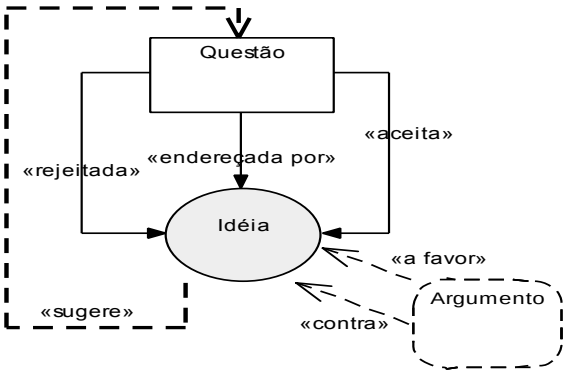

Figura 3-4. Representação visual dos elementos de raciocínio da ontologia Kuaba.

A representação de *design rationale* para um artefato normalmente começa com uma questão genérica que estabelece o problema de design a ser resolvido. Esta questão genérica pode dar origem a novas questões que representam novos problemas de design relacionados ao problema principal. Para cada questão apresentada, os projetistas podem sugerir idéias, formulando possíveis soluções para o problema descrito na questão. Argumentos são apresentados contra ou a favor das idéias apresentadas e decisões são tomadas com base nesses argumentos. Quando a abordagem Kuaba é usada, os elementos da ontologia devem ser instanciados com base no metamodelo utilizado na atividade específica de design do sistema.

As instâncias de Argumento, Decisão e Justificativa são usualmente dadas pelo projetista, podendo em casos especiais ser dadas pelo metamodelo. Esse é o caso quando o metamodelo possui regras que já determinam a solução, dada pelas condições de navegação descritas em sua rede semântica. As instâncias de Idéia, em um primeiro momento, são criadas pelo projetista a partir dos conceitos obtidos do domínio, que são chamados de idéias de domínio. O metamodelo prescrito pelo método de design utilizado determina as demais idéias. Essas últimas são chamadas de idéias de design.

A Figura 3-5 mostra um exemplo de representação de *design rationale* utilizando a abordagem Kuaba para um modelo de domínio de um catálogo de CD, representado como um diagrama de classes da UML.

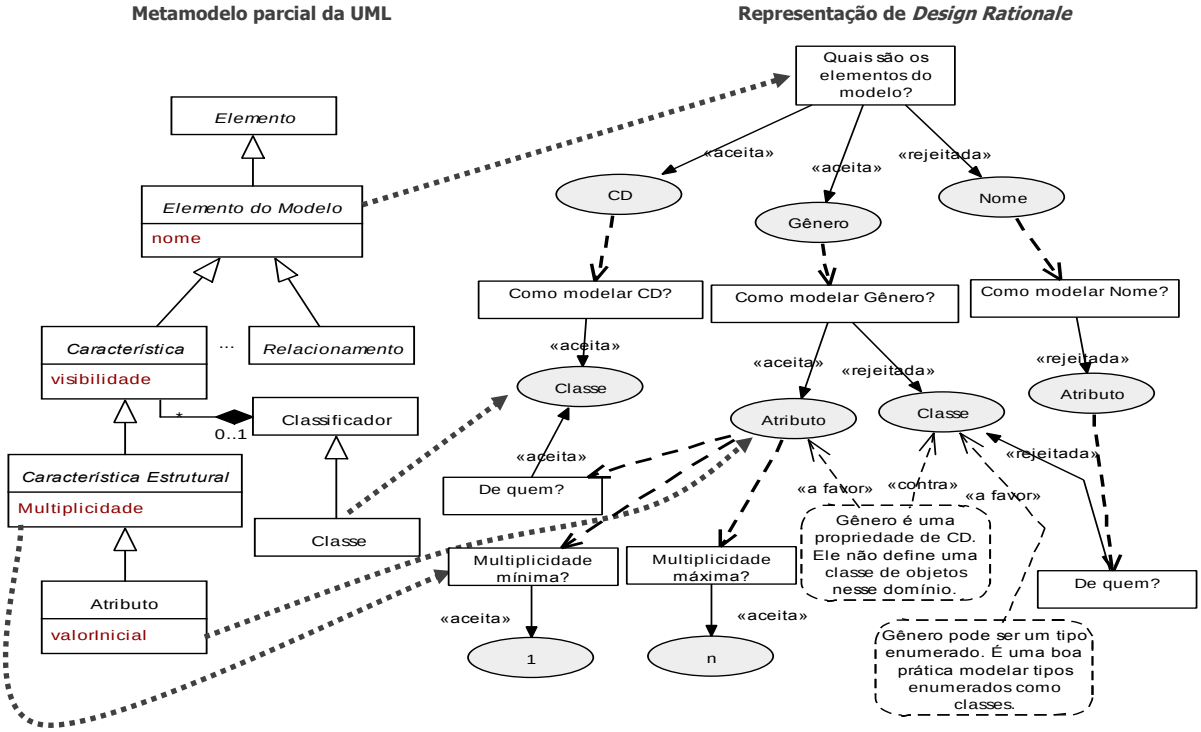

Figura 3-5. Exemplo de representação de *design rationale* usando a abordagem Kuaba e o metamodelo da UML para diagramas de classes.

O exemplo mostra como a semântica do metamodelo da UML é utilizada na representação de *design rationale* para instanciar os elementos de raciocínio da ontologia Kuaba. De acordo com esse metamodelo, o primeiro problema a ser resolvido no design de um diagrama de classes é a identificação dos elementos do modelo (metaclasse Elemento do Modelo). Aplicando o vocabulário da ontologia Kuaba, o resultado é a criação do elemento Questão (representado como um retângulo) com a instância Quais são os elementos do modelo?. Essa questão inicial é respondida pelas idéias CD, Gênero e Nome, representadas como elipses. Esses valores são determinados pelo conhecimento do projetista sobre o domínio. Essas idéias de domínio, por sua vez, sugerem as questões de design Como modelar CD?, Como modelar Gênero? e Como modelar Nome?. As possíveis idéias de design que respondem essas questões também são determinadas pelo metamodelo da UML. Por questões de simplicidade, apenas as idéias de design Classe e Atributo foram consideradas. Da mesma forma, as questões Multiplicidade Mínima? e Multiplicidade Máxima? associadas à idéia Atributo também são instanciadas de acordo com o metamodelo da UML.

Dessa forma, é possível visualizar como a semântica do metamodelo é usada na abordagem Kuaba para gerar, automaticamente, parte da representação de *design rationale* (questões e idéias de design). Assim, os projetistas têm apenas o trabalho de registrar os

argumentos (retângulos pontilhados na Figura 3-5) contra e a favor de cada idéia de solução e as razões para as decisões tomadas. As decisões do projetista são ilustradas com os rótulos dos estereótipos estendidos <<aceita>> ou <<rejeitada>> nas setas entre questões e idéias.

Vale lembrar que as idéias induzidas pelo metamodelo que aparecem diversas vezes nos registros de *design rationale* não são cópias, ou seja, instâncias iguais. O mesmo nome é usado apenas para facilidade, mas internamente são instâncias com identificadores diferentes e assim poderiam ter sido nomeadas.

4 ABORDAGENS PARA ENGENHARIA DE REQUISITOS BASEADAS EM METAMODELOS

Sommerville (2006) define requisitos como as descrições dos serviços e das restrições de um sistema, sendo o processo de identificá-los, defini-los, analisá-los, documentá-los e validá-los, chamado de Engenharia de Requisitos. Zave (1997) acrescenta que esse processo preocupa-se com os objetivos do mundo real e com a perspectiva temporal da relação entre os requisitos e a correta especificação do comportamento do software. Nuseibeh e Easterbrook (2000) complementam com a observação que fatores relacionados ao ambiente onde o software vai operar também devem ser levados em conta na Engenharia de Requisitos.

Durante a atividade de Engenharia de Requisitos o domínio traz aos projetistas aqueles conceitos puramente relacionados com o mundo real, referenciados por Zave (1997), no ambiente onde o software irá operar, ao qual Nuseibeh e Easterbrook (2000) se referem, trazendo valor para seus usuários. Requisitos devem ser descobertos a partir das necessidades dos interessados (*stakeholders*) no desenvolvimento do software, sejam elas funcionais ou não funcionais. Essas necessidades são expressas pelas visões específicas de cada pessoa envolvida, sendo usualmente conflitantes, ou restritas por aspectos econômicos, técnicos ou tecnológicos, tornando complexos os problemas abordados.

Entender a natureza desses problemas pode ser muito difícil, especialmente para novos sistemas ou processos de negócios, dificultando o estabelecimento do que exatamente o sistema deve fazer (SOMMERVILLE, 2006). Foi observado que atrasos nas entregas previstas, gastos excessivos, e insatisfação de usuários e clientes devido a um produto difícil de usar e caro para manter, são problemas que têm como fatores subjacentes comuns as dificuldades com os requisitos (KOTONYA; SOMMERVILLE, 1997). Como o produto da análise de requisitos é usado nas fases subsequenes do desenvolvimento, problemas com requisitos podem colocar em risco todo o projeto.

A Engenharia de Requisitos é tão antiga quanto a Engenharia de Software. No final dos anos 60, devido à iminente crise na produção de software provocada pela informalidade, pela ausência de padrões e métodos, e por outros fatores, teve origem a área da Engenharia de Software, cuja missão seria a definição de métodos e processos para apoiar o desenvolvimento do produto de software. Nessa ocasião, viu-se que as práticas de análise e definição de requisitos eram fontes de potencial alavancagem para a disciplina da Engenharia de Software, dado que requisitos são potencialmente importantes para a melhoria da qualidade do produto final (GREENSPAN, 1994). Essa preocupação começou a produzir efeitos e gerar uma boa quantidade de dados empíricos por volta da metade da década de 70. Deste então, o termo Engenharia de Requisitos foi reconhecido como uma subárea da Engenharia de Software. A formalização de requisitos na época significava organizar, descrever, rotular e documentar os requisitos, listando-os como um conjunto de necessidades expressas de forma textual e livre, eminentemente descritiva. Lamsweerde (2000) destaca que uma grande mudança começa no trabalho seminal de Ross e Schoman (1979), que além de uma explicação abrangente sobre o escopo da Engenharia de Requisitos, sugere as bases de uma ontologia para descrever esse domínio, que busca a formalização para a especificação de requisitos. Essas iniciativas darão origem aos métodos baseados em metamodelos.

A meta modelagem é a análise, construção e desenvolvimento de regras, restrições, modelos e teorias aplicáveis e úteis para modelar uma classe específica de modelos, cujo resultado é um metamodelo (RUMBAUGH; JACOBSON; BOOCH, 1999). Esse metamodelo representa abstrações cuja instanciação representa um modelo, ou as representações das instâncias de seus elementos formam modelos. Na Engenharia de Requisitos, o metamodelo é a representação das abstrações de conceitos relativos às necessidades das pessoas envolvidas no projeto de desenvolvimento, que devem ser expressas de uma maneira que preserve sua semântica. Os metamodelos geralmente atendem a esse requisito quando são suficientemente ricos para representar essas abstrações, mantendo seu significado e criando a ponte para a criação de artefatos dirigidos para implementação.

Em meados da década de 80, Greenspan (1984) apresenta uma linguagem baseada em representação de conhecimento para requisitos, RML (*Requirements Modeling Language*), sendo a precursora das linguagens de programação orientadas para objetos. Desde então, surgiram linhas de evolução tais como a orientação para objetivos (*goal-oriented*), a orientação para aspectos (*aspect-oriented*) e a orientação para casos de uso. Além destas abordagens, há metamodelos, ou linguagens, voltados para domínios específicos e abordagens híbridas.

Este trabalho de pesquisa estudou os seguintes metamodelos usados para o design de modelos de requisitos: RML, KAOS – *Knowledge Acquisition in AutOmated Systems*, (DUBISY, 1991, DARDENNE, 1993; LAMSWEERDE, 1995, 2009; DARIMONT, 1997), *i** (YU; 1997, 2001a, 2001,b, 2001c), Tropos (GIUNCHIGLIA; MYLOPOULOS; PERINI, 2002; GIORGINI, 2003; BRESCIANE, 2004; FUXMAN, 2004), URN, *User requirements Notation* (ITU-T, 2008) e REMM – *Requirements Engineering MetaModel*, (CHICOTE; MOROS; TOVAL, 2007). Esses metamodelos foram estudados com o objetivo de obter um metamodelo capaz de apoiar o design dos modelos de requisitos com semântica suficiente para investigar a representação de *design rationale* para esses modelos usando a abordagem Kuaba.

Neste capítulo foi feita uma classificação dos metamodelos que mostra a sua evolução histórica ou temática. Por exemplo, a Seção 4.1 trata aspectos históricos estudando abordagens precursoras no uso de metamodelos para representar requisitos. No entanto, as demais seções abordam os metamodelos sob o aspecto de sua orientação para representar requisitos. Assim, a Seção 4.2 apresenta abordagens orientadas para objetivos e a Seção 4.3 abordagens descritivas, onde é estudado o metamodelo REMM. Neste capítulo também são discutidas as razões pela escolha do metamodelo do *framework* KAOS para os testes de modelagem e representação de *design rationale* desta dissertação.

4.1 ABORDAGENS PRECURSORAS

A RML foi anunciada pela primeira vez em Greenspan, Mylopoulos e Borgida (1982), sendo refinada e consolidada em Greenspan (1984a, 1984b). Essa *linguagem* de modelagem de requisitos é centrada em objetos, abordando modelos como objetos de vários tipos, grupados em classes que são instâncias de metaclasses. O metamodelo prevê três níveis: meta, modelo, e instâncias. A ontologia proposta para a modelagem de requisitos sugere que no mundo real existem três tipos de coisas sobre as quais se pode falar a respeito: Entidade, Atividade e Assertiva. Uma semântica formal é dada para RML através da definição de um mapeamento de suas descrições para assertivas ou axiomas em lógica de primeira ordem. Esta semântica inclui todos os axiomas e predicados associados com as classes específicas definidas pelo modelador.

Stanley (1986) apresenta a CML (*Conceptual Modeling Language*) que aprimora RML dotando a linguagen de mecanismos para a criação de outros conceitos além dos três originalmente oferecidos (atividade, entidade e assertiva), permitindo que os engenheiros de

requisitos pudessem definir extensões específicas para suas classes de aplicações. Telos (MYLOPOULOS, 1990) continuou o processo de aprimoramento de RML através do tratamento homogêneo dado aos elementos do metamodelo. Telos encerra a linha evolucionária dos metamodelos orientados para objetos, quando a pesquisa já indicava que uma mudança de paradigma se consolidava com a tendência da orientação para objetivos.

4.2 ABORDAGENS ORIENTADAS PARA OBJETIVOS

As abordagens orientadas para objetivos surgiram no final da década de 80 e levaram a projetos como KAOS (DUBISY, 1991). Alguns anos mais tarde, Yu e Mylopoulos (1994) apresentam outro *framework* chamado de *i** (pronunciado "aistar") que dá origem a um grande projeto multinacional e inter-universidades chamado Tropos cujo objetivo é desenvolver uma abordagem para a análise, o projeto e a construção de sistemas, e não apenas a Engenharia de Requisitos.

Além do projeto Tropos, a evolução do *framework i** juntamente com a notação para modelagem de cenários UCM (*Use Case Maps*) de Buhr e Casselman (1995) dá origem à URN, *User Requirements Notation*, aprovada em novembro de 2008 como padrão internacional, sendo recomendação da ITU-T (*International Telecommunication Union*) da Suiça. A URN consiste de uma linguagem orientada para objetivos, a GRL (*Goal Representation Language*), baseada em *i** e UCM. Os "casos de uso" dos mapas são conceitos diferentes daqueles da UML, apesar de também abordarem a descrição de cenários e interações.

Embora *i** tenha como seu elemento central o conceito de agente e seus relacionamentos estratégicos, ele é um *framework* para a representação de modelos de requisitos orientados para objetivos, pois os relacionamentos estratégicos entre agentes implicam na satisfação de objetivos, sendo assim consistente com a idéia central desse paradígma.

4.2.1 *Framework* i*, Tropos e URN

As técnicas usuais de levantamento de requisitos, como no Processo Unificado (JACOBSON; BOOCH; RUMBAUGH, 1999), no qual os requisitos são representados através da descrição de casos de uso e como atores interagem com eles, procuram obter o conhecimento sobre "o que" o sistema deve fazer, em oposição ao "como" ele deve fazer. A

inspiração de *i** vem do fato de que o entendimento do contexto organizacional, seus *rationales* e seus "porquês", que levam aos requisitos de um sistema, podem ser tão importantes para o sucesso do sistema quanto o "o que". A modelagem de requisitos também pode apoiar as fases bem iniciais do levantamento, embora as técnicas mais usuais sejam as que usam a modelagem nas fases finais do levantamento (YU, 1997a). Além disso, uma abordagem de modelagem de requisitos centrada em agentes oferece novas formas de caraterização e análise das interações e relacionamentos entre entidades do sistema e do ambiente. Baseado nas obervações sobre necessidades de mudanças na modelagem de requisitos, Yu (2001b) relaciona seis propriedades que são desejáveis para o meta-conceito de um agente: intencionalidade, autonomia, sociabilidade, identidade e fronteiras eventualmente identificáveis, reflectibilidade estratégica e auto-interesse racional. Não serão abordados aqui detalhes sobre estas propriedades na medida em que, apesar de terem valor para menção, fogem do escopo do trabalho.

Erick Yu (YU; MYLOPOULOS, 1994) foi o primeiro a classificar a análise de requisitos em fases, como inicial e final, usando-se uma tradução não literal para *early* e *late*, respectivamente. O *framework* foi consolidado em seu trabalho de doutorado em 2001 (YU, 2001a). Este *framework* aborda o modelo de requisitos como a composição de dois sub-modelos: o modelo de Dependências Estratégicas (SD, do inglês *Stratategic Dependencies*) e o modelo de *Rationale* Estratégico (SR, *Strategic Rationale*).

O primeiro sub-modelo descreve a rede de dependências entre atores a fim de satisfazer um objetivo. Somente através da colaboração estratégica e intencional entre atores, os objetivos poderão, ou não, ser satisfeitos. O modelo de *Rationale* Estratégico descreve explicitamente as razões por trás das estruturas de processos definidas. O processo é descrito através de objetivos, tarefas, recursos, objetivos fracos (*softgoal*) e sub-objetivos, relacionados através de associações entre "meios-e-fins" e decomposição de tarefas. Os elementos são incluídos nesse modelo em função de sua importância para a satisfação de um objetivo.

A Figura 4-1 ilustra os quatro níveis do metamodelo de *i** (Yu, 2001a). O diagrama mostra os elementos para a representação de requisitos na linguagem desde o nível mais abstrato (meta-meta) até o nível das instâncias dos elementos do modelo, chamados de *tokens*. Na ilustração, o nível do metamodelo (o segundo de cima para baixo) mostra três setores. À esquerda é mostrado o conjunto de conceitos relativos a agentes e seus papéis. Na área central são ilustradas as dependências entre os próprios agentes e destes com os recursos, sendo que os recursos são mostrados no setor direito do diagrama.

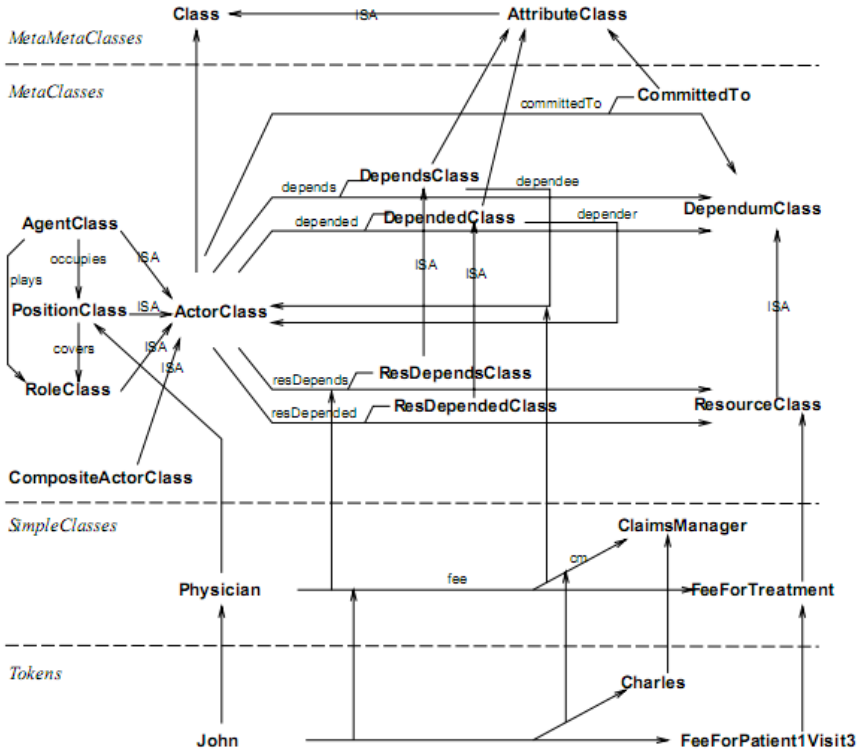

Figura 4-1. Os quatro níveis de representação *i (YU, 2001a).**

A Figura 4-2 mostra um modelo de Dependências Estratégicas do domínio de tratamento de saúde. Nela, são parcialmente ilustradas as dependências entre pacientes, médicos, laboratórios e empresas de seguro-saúde. Os pacientes dependem dos médicos para tratamento, enquanto os médicos dependem dos pacientes para que tomem a medicação prescrita, e também dos laboratórios para realizar os exames necessários para o diagnóstico do paciente. Os médicos e os laboratórios dependem do pagamento das tarifas de tratamento e exames feitos pelas companhias de seguro-saúde. Essas empresas dependem de seus funcionários que gerenciam o faturamento para que possam realizar os pagamentos necessários e ao mesmo tempo os funcionários precisam que suas companhias informem os dados do paciente. Os pacientes devem pagar suas mensalidades para que continuem a ter a cobertura do serviço através do qual dependem da companhia de seguro saúde para terem direito aos tratamentos, quando necessários.

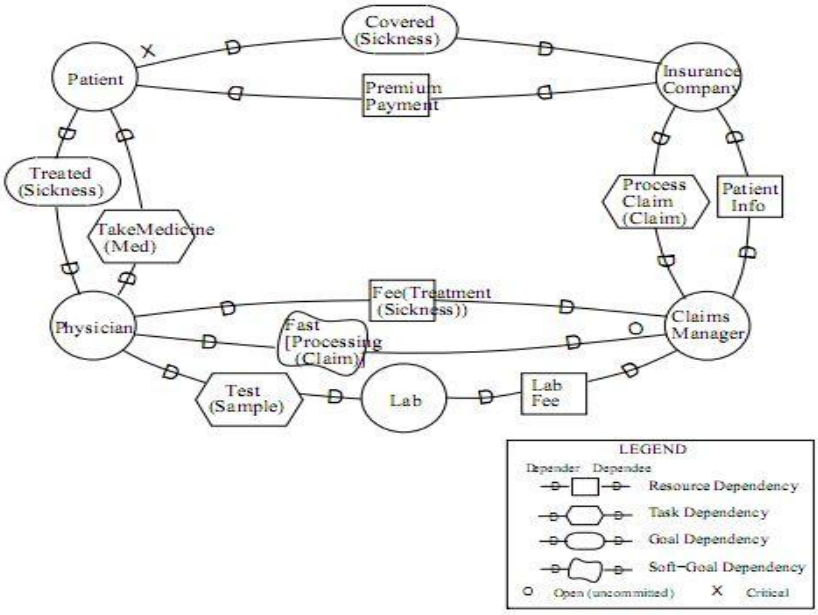

Figura 4-2. Diagrama de um modelo SD de *i para o domínio do tratamento de saúde (YU, 2001a).**

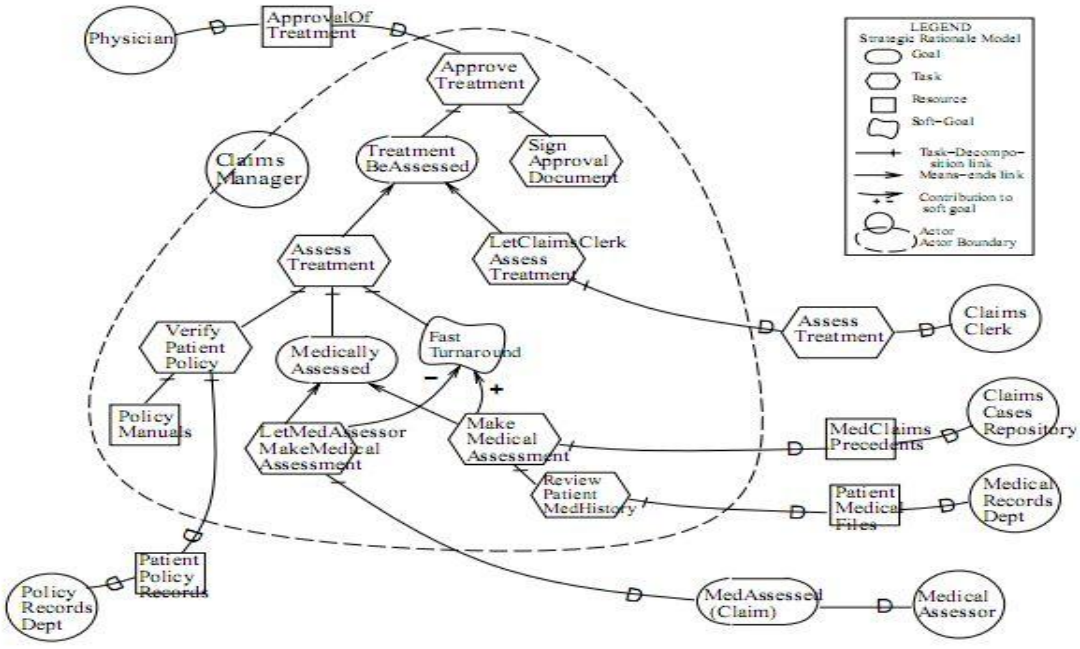

Figura 4-3. Modelo de *Rationale* Estratégico para para o ator que gerencia a cobrança (YU, 2001a).

A representação gráfica do modelo de *Rationale* Estratégico que captura os *rationales* envolvidos no estabelecimento das faturas pela empresa de seguro-saúde é ilustrada na Figura 4-3. O médico deve submeter um programa de tratamento à companhia de seguro para aprovação prévia, caso contrário o tratamento não será pago. A empresa verifica se aquele

tratamento é coberto pela apólice do paciente e verifica, através de aconselhamento médico próprio, se o tipo de tratamento proposto é compatível.

Tropos é uma metodologia de desenvolvimento de sistemas de software que utiliza o *framework i**. Baseia-se em duas características-chaves: *(a)* as noções de agentes, metas, planos e vários outros conceitos em nível do conhecimento e *(b)* o reconhecimento de um papel de extrema importância para a especificação e análise de requisitos (GIUNCHIGLIA; MYLOPOULOS; PERINI, 2002). Em Tropos o processo da Engenharia de Requisitos é dividido em duas fases: de requisitos "cedo" (*early requirements*) e "tarde" (*late requirements*). Durante a primeira fase os engenheiros de requisitos identificam os envolvidos no domínio e os modela como atores sociais que dependem uns dos outros para a satisfação de objetivos e para que planos possam ser executados e recursos possam ser supridos, como em *i**. Através da definição clara dessas dependências é possível determinar o "porque", além do "como" e "o que", sobre a funcionalidade e verificar se a implementação final tem conformidade com a real necessidade. Na análise "tarde" o modelo é estendido para abordar outras dependências entre atores do domínio. A atividade de análise de requisitos na metodologia Tropos define os requisitos funcionais e não funcionais do sistema em desenvolvimento (BRESCIANE, 2002).

De maneira muito parecida com *i**, os modelos conceituais e diagramas são desenvolvidos com instâncias de conceitos intencionais e sociais tais como: ator, meta, dependência, plano, recurso, capacidade e crença. Um ator modela uma entidade que possui metas e intencionalidade; representa um agente físico, como uma determinada pessoa, ou um agente do sistema tal como um papel ou posição (cargo). Um papel é uma representação abstrata do comportamento de um ator em determinado contexto. Uma posição trata-se de um conjunto de papéis que na maioria dos casos é desempenhado por um agente. Os interesses estratégicos dos atores são representados por um objetivo. Objetivos podem ser obrigatórios, ou facultativos, denotando a necessidade de sua satisfação. Uma dependência entre atores indica que um deles depende do outro para satisfazer um objetivo, executar algum plano ou suprir um recurso. Um plano representa uma maneira de satisfazer um objetivo, sendo o recurso a entidade física ou de informação que um ator necessita e outro pode suprir. Capacidade é a habilidade que um ator possui para definir, escolher e executar um plano a fim de satisfazer uma meta, dado um ambiente operacional particular. O conhecimento de cada ator sobre o mundo é representado por crenças.

A Figura 4-4 mostra um diagrama de atores em Tropos, incluindo o ator PAT (Província Autônoma de Trento). A Figura 4-5 apresenta o diagrama de objetivos, chamado

no *framework* Tropos de meios-e-fins, para o ator PAT, no sistema de *eCommerce* dessa província.

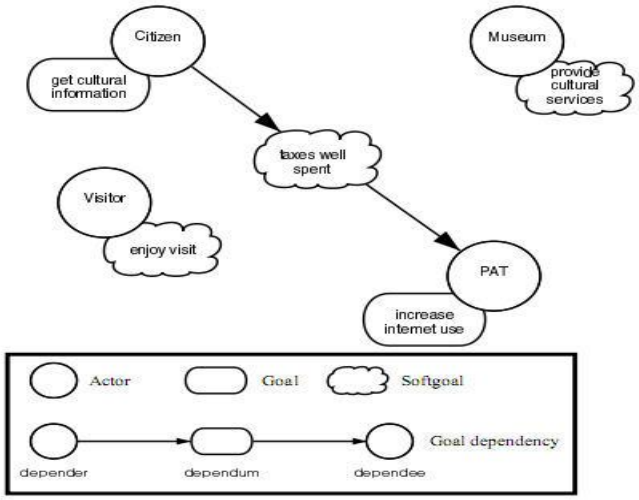

Figura 4-4. Diagrama de atores em Tropos/*i (BRESCIANI, 2002).**

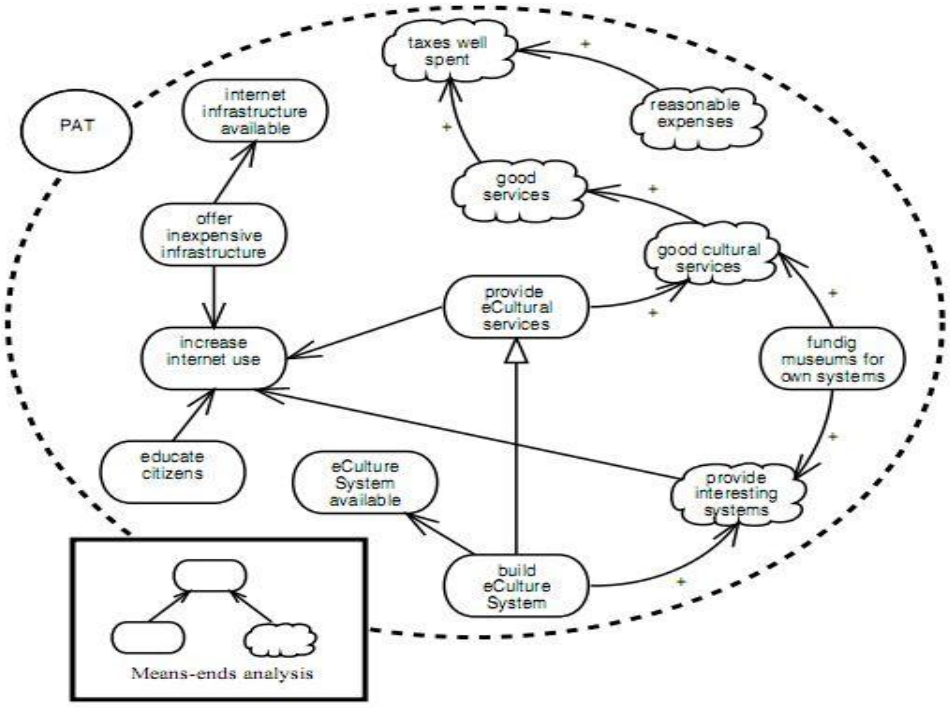

Figura 4-5. Modelos de objetivos (meios-e-fins) em Tropos/*i (BRESCIANI, 2002).**

A linguagem de modelagem é elemento central da metodologia Tropos. A Tabela 4-1 mostra os quatro níveis do metamodelo. O nível meta-meta provê a base para extensões do metamodelo, contendo as primitivas da linguagem. Componentes para modelar entidades e conceitos no nível do conhecimento são fornecidos pelo nível do metamodelo. No nível de

domínio são realizadas as representações das entidades e conceitos de um domínio de aplicação específico, sendo que o nível de instância contem as realizações dos conceitos de dado domínio.

Tabela 4-1. Níveis de representação Tropos/*i (GIUNCHIGLIA; MYLOPOULOS; PERINI, 2002).**

Level	Description	Examples
meta metamodel	Basic language structural elements	Attribute, Entity
metamodel	Knowledge level notions	Actor, Goal, Dependency
domain	Application domain entities	PAT, Citizen, Museum
instance	Domain model instances	Mary: instance of Citizen

O metamodelo que descreve os conceitos do diagrama de meios-e-fins é mostrado no diagrama UML da Figura 4-6. A diferença entre objetivo obrigatório e facultativo é expressa por uma associação de especialização do conceito de Objetivo propriamente dito. Objetivos podem ser analisados, sob o ponto de vista de um ator, de duas maneiras: a análise de meios-e-fins (*means-ends*), que é uma associação ternária envolvendo Ator, um Objetivo (o fim) e um Plano, recurso ou outro Objetivo (o meio).

Este tipo de análise é mostrado na Figura 4-5. Objetivos de prover educação para os cidadãos e prover serviços do sistema eCultural são meios para atingir o objetivo aumentar o uso da internet. Além disso, a redução AND-OR tenta identificar objetivos que podem contribuir positiva ou negativamente no sentido de satisfazer a um objetivo.

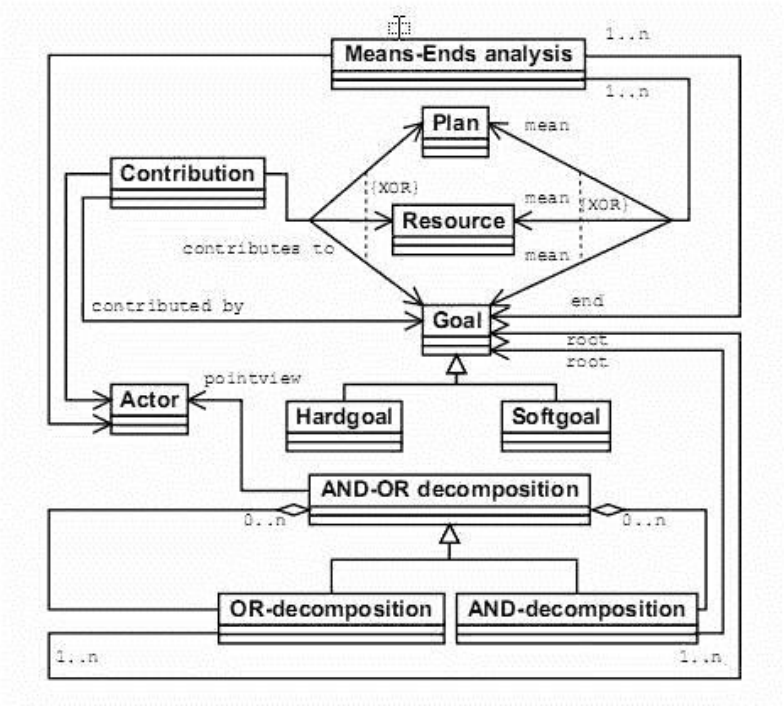

Figura 4-6. Conceitos do modelo de meios-e-fins em Tropos/*i, mostrados através de um diagrama de classes UML (GIUNCHIGLIA; MYLOPOULOS; PERINI, 2002).**

4.2.2 KAOS

O modelo conceitual do *framework* KAOS apóia a construção de modelos de requisitos. Esse metamodelo possui semântica que permite a captura tanto de requisitos funcionais, como não funcionais para qualquer tipo de sistema (DARDENNE, 1991, 1993; LAMSWEERDE, 1995). Os elementos do metamodelo fornecem abstrações para modelar necessidades relacionadas com os objetivos do sistema em análise; restrições aplicadas a eles, sejam elas obstruções ou conflitos; agentes, sejam eles seres humanos, dispositivos, ou componentes de software, que através de seu comportamento propiciam a satisfação dos objetivos; eventos que produzem ações sobre entidades, e outras, como será mostrado nesta dissertação. Além disso, oferece um conjunto de tipos de associação: subclasses, associações de refinamento de objetivos e de obstáculos, associações de responsabilidade e de comissionamento entre agentes e objetivos, dentre outros (DARDENNE, 1993; LAMSWEERDE, 1995; DARIMONT, 1997). A Figura 4-7 dá uma visão completa do metamodelo KAOS usando a sua própria representação visual.

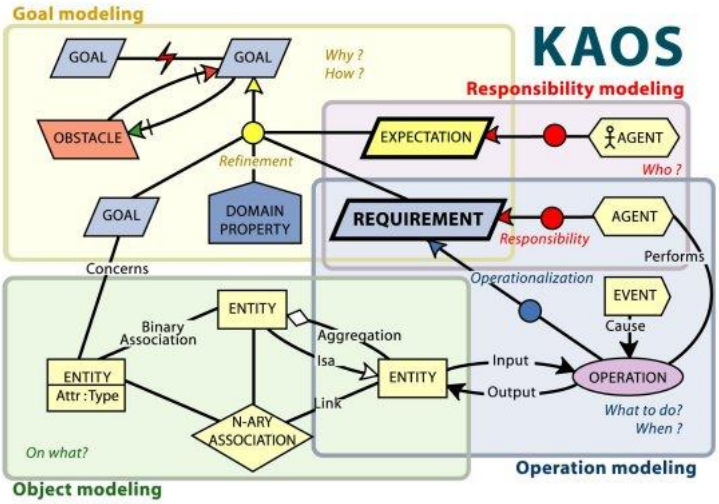

Figura 4-7. Meta-modelo KAOS mostrando conceitos dos quatro sub-modelos, suas associações em sua própria representação visual (RESPECT-IT, 2007).

O *framework* KAOS é apresentado em detalhe no Capítulo 5 desta dissertação pelo fato do seu metamodelo ter sido escolhido para a realização dos testes de representação de *design rationale* desta pesquisa. Uma discussão sobre as razões que levaram a esta escolha são apresentadas na Seção 4.4.

4.3 REMM

O metamodelo REMM é uma proposta de integração de requisitos ao enfoque MDE, *Model-Driven Engineering*. Ele é necessário uma vez que uma representação descritiva baseada em documentos não atende a este fim. Os elementos do metamodelo para a representação de requisitos são altamente dependentes do contexto em que ele é usado: planejamento de liberações, gerenciamento de requisitos, etc (DAHLSTEDT, 2003; CHICOTE; MOROS; TOVAL, 2007).

O metamodelo REMM é orientado para o reuso de requisitos, mais especificamente baseado na abordagem SIREN (*SImple REuse of software requiremeNts)* para a Engenharia de Requisitos de Toval (2002). Os meta-conceitos utilizados em REMM são, em sua maioria, obtidos daqueles descritos em SIREN. Chicote, Moros e Toval (2007) acreditam que, apesar desta relação, REMM tem aplicabilidade na Engenharia de Requisitos em contextos mais gerais.

O metamodelo apóia o registro de requisitos, envolvidos (*stakeholders*), casos de teste e outros conceitos ilustrados no diagrama de classes UML apresentado na Figura 4-8. No metamodelo REMM todos os requisitos são armazenados em um Catálogo (Catalog), que tem

o objetivo de propiciar o reuso. O catálogo é descrito pelos atributos: name, purpose e type. Há possibilidade de haver dois tipos de catálogos: DOMAIN e PROFILE. Um catálogo pode conter três tipos de requisitos: *a)* de sistema, que representam as necessidades do ambiente do sistema como um todo; *b)* de software, que representam como o requisito de sistema será satisfeito pelo software, por exemplo, a necessidade do sistema de filtrar dados para exibição é satisfeito por uma caixa de diálogo, um pop-up, ou outra solução de software; e *c)* restrições (Constraint no metamodelo), representando o grau de liberdade que se tem no provimento da solução.

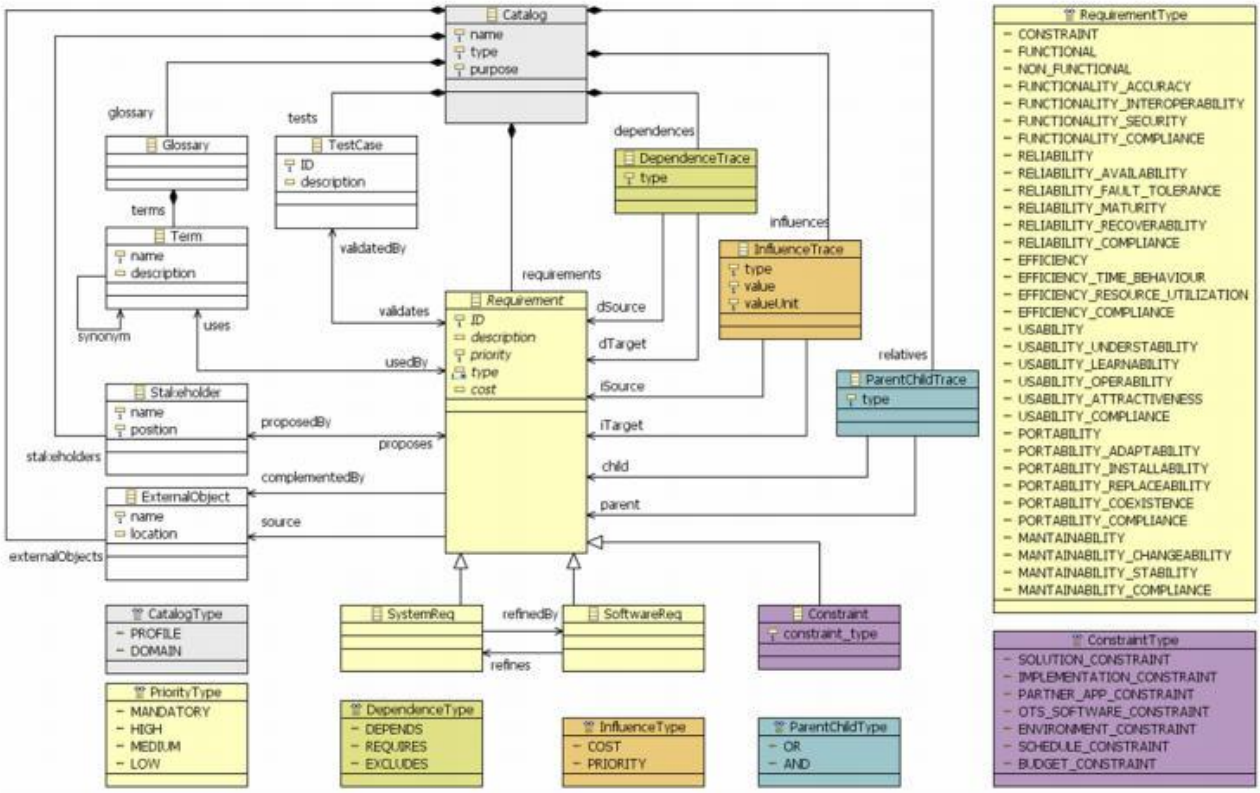

Figura 4-8. Representação em UML do metamodelo REMM (CHICOTE; MOROS; TOVAL, 2007).

Os requisitos são caracterizados por um identificador único (ID), uma descrição textual (description), um tipo (type), cujos valores são definidos através de uma enumeração, um custo (cost), e outra enumeração referente à prioridade (priority) que pode assumir os valores: MANDATORY, HIGH, MEDIUM, LOW. Uma restrição, ou seja, um requisito do tipo Constraint, deve obedecer à regra de prioridade = MANDATORY, que não é mostrada no metamodelo. O tipo das restrições é definido pelo atributo constraint type, obtido do conjunto enumerado de

valores ConstraintType. A enumeração RequirementType, tipo de requisito, obedece a norma ISO/IEC 9126 (ISO/IEC, 1991).

REMM possui uma implementação em REMM-Studio, que é um ambiente gráfico para a modelagem, registro e análise de requisitos. A interface gráfica da ferramenta de Catálogo (*CatalogTool*) é mostrada na Figura 4-9 (CHICOTE; MOROS; TOVAL, 2007).

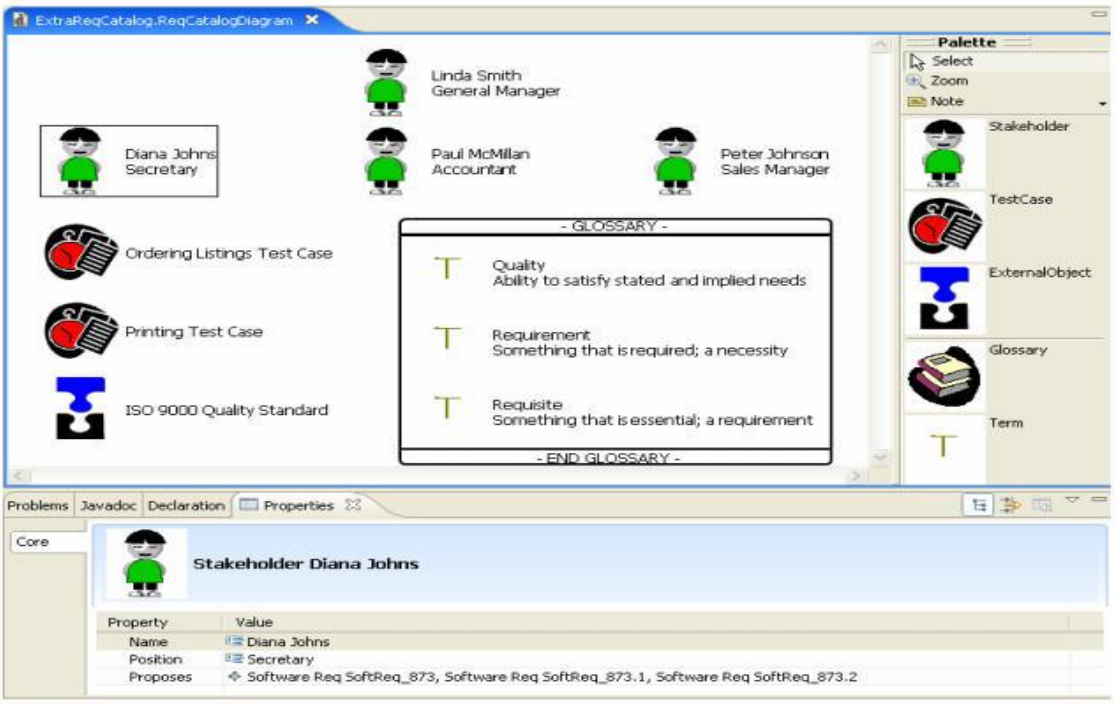

Figura 4-9. Exemplo de uma das telas da ferramenta REMM-Studio de Chicote, Moros e Toval (2007).

A ferramenta de catálogo permite a representação de interessados (*stakeholders*), casos de testes, objetos externos e termos do dicionário de dados. Por exemplo, em um caso de uso onde se quer especificar requisitos proposto por Diana Johns, é necessário carregar os arquivos contendo os requisitos definidos, selecioná-la (como mostra a imagem representando Diana na interface gráfica) e associar a ela, escolhendo cada requisito desejado e usando o atributo proposes. Esse dado é automaticamente atualizado nos modelos de requisitos correspondentes, mantendo a consistência entre eles, isto é, uma instância da classe requisito terá como valor do campo proposedBy, Diana Johns. Na ilustração do metamodelo mostrada na Figura 4-8, este campo é representado pelo papel do requisito no relacionamento entre as classes Requirement e Stakeholder.

4.4 DISCUSSÃO

A história da metamodelagem na Engenharia de Requisitos mostra um processo de enriquecimento semântico dos meta-conceitos. Esse processo reflete a busca pela representação adequada dos problemas complexos desta atividade que possa guiar o desenvolvimento, já que a compreensão dos requisitos orienta todas as atividades subsequentes do desenvolvimento.

A notação GRL (IUT-T, 2008) é evolução de *i** como padrão recomendado pela IUT-T. Na recomendação o metamodelo é especificado com clareza, apesar da sua complexidade. Embora sua adoção por um grande número de engenheiros de requisitos e o suporte mais difundido em ferramentas ainda deva esperar alguns anos, é uma linguagem de representação que tem capacidade de representar abstrações complexas na Engenharia de Requisitos. A URN como um todo, é um contra-ponto importante em relação ao *framework* KAOS mais largamente adotado na Europa.

O metamodelo REMM, embora seja um metamodelo descritivo, é importante para a organização, armazenamento, e gerenciamento dos requisitos, além de capturar algumas associações de dependência entre eles com semântica relevante. Entretanto, a quantidade de tipos de requisitos (enumeração `RequirementType`) dificulta sua representação através de uma ferramenta visual e principalmente seu entendimento na prática usual de modelagem pelo engenheiro de requisitos. As meta-classes do metamodelo parecem ter por objetivo a obtenção de instâncias de estruturas de dados para a formação de um repositório para reuso não computacional, como preconiza a metodologia SIREN (TOVAL, 2002).

Os metamodelos orientados para objetivos mostram, de maneira geral, uma semântica, cujas abstrações podem representar os conceitos complexos da Engenharia de Requisitos. Embora todos sejam interessantes para esta pesquisa, o objetivo principal era obter um metamodelo rico e estável, que tivesse sido suficientemente testado em sistemas reais. Portanto, o metamodelo do *framework* KAOS foi escolhido para os testes de modelagem e representação dessa dissertação.

5 KAOS

O método KAOS (*Knowledge Acquisition in AutOmated Specification*) surgiu no cenário da Engenharia de Requisitos com uma mudança de paradígma ocorrida no final da década de 80, início dos 90, que foi chamada de orientação para objetivos (*goal-orientation*). Na Universidade de Louvain, na Bélgica, foi criado um conjunto de projetos abordando esse tema central, dentre os quais, os dois mais conhecidos são KAOS, direcionado para a descoberta de requisitos e ICARUS, direcionado para a formalização de requisitos (DUBISY; LAMSWEERDE; DARDENNE, 1991; AL-SUBAIE, 2007). O acrônimo KAOS teve seu significado recentemente revisado por Lamsweerde (2009) como *Keep All Objectives Satisfied*.

KAOS oferece uma linguagem e uma ferramenta de modelagem gráfica para a modelagem de requisitos de sistemas através de múltiplas visões, uma linguagem formal para a especificação dos modelos, um método sistemático para sua elaboração e técnicas para refinamento de objetivos, gerenciamento de conflitos e obstruções.

5.1 O FRAMEWORK KAOS

O *framework* KAOS consiste de um modelo conceitual, uma linguagem formal de especificação de requisitos, um método para a elaboração da especificação de requisitos e um conjunto de heurísticas para apoiar o engenheiro de requisitos no seu uso. Essa dissertação usa apenas o modelo conceitual, o metamodelo do *framework* KAOS.

KAOS captura requisitos em três níveis. O nível meta é aquele do conhecimento independente do domínio que define o modelo de abstração dos conceitos que orientam a identificação dos requisitos e seus relacionamentos. Os conceitos principais desse nível são: Objeto, Objetivo, Agente, Operação, Relacionamento, e Entidade. Esses conceitos são

mostrados na Figura 5-1a, com os nomes originais Object, Objective, Agent, Operation, Relationship e Entity[2]. O nível de domínio, exemplificado como um diagrama misto de responsabilidades mostrando também o refinamento de um objetivo na Figura 5-1b, define um grafo de conceitos exclusivamente do domínio do sistema. O terceiro nível trata de instâncias específicas desses conceitos. Esse nível está fora do escopo deste trabalho que está voltado para abstrações que representam conceitos até o nível de domínio.

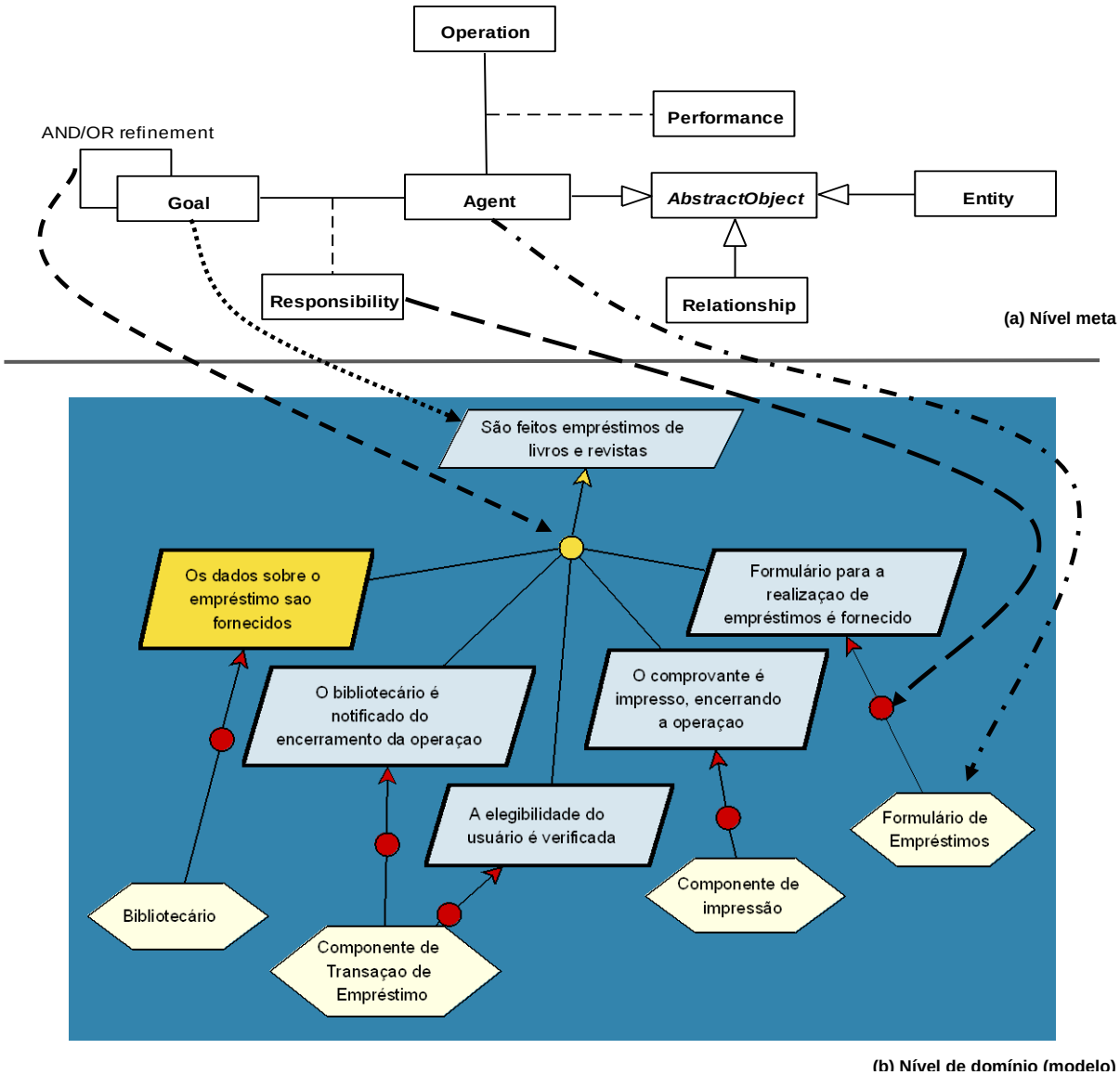

Figura 5-1. Níveis de representação do framework KAOS: (a) Metamodelo KAOS (Nível meta) (RESPECT-IT, 2007); (b) Exemplo de um diagrama de responsabilidade.

O metamodelo KAOS, cuja instância guia a obtenção de modelos de requisitos, permite representar tanto requisitos funcionais, quanto não funcionais, para uma quantidade

[2] O metamodelo KAOS foi originalmente criado em inglês e foi mantido nesta língua. A tradução sempre será feita a fim de não deixar dúvidas sobre que conceito é abordado.

significativa de domínios, como mostram os trabalhos de Dardenne (1991, 1993), Lamsweerde, Darimont e Massonet (1995), Lamsweerde (2000, 2009), Darimont, (1997) e Letier (2001), dentre outros.

O *framework* KAOS usa quatro modelos: Modelo de Objetivos, Modelo de Objetos, Modelo de Responsabilidade (Agentes) e o Modelo de Operações mostrados na Figura 5-2, que ilustra o metamodelo completo na sua representação visual. Os modelos se relacionam através de associações entre conceitos de sub-modelos diferentes permitidas pelo metamodelo. Por exemplo, um agente, uma instância de Agent, pode ser responsável por instâncias de Requirement, requisitos, ou exclusivamente de uma expectativa, instância de Expectation, dependendo do tipo de agente.

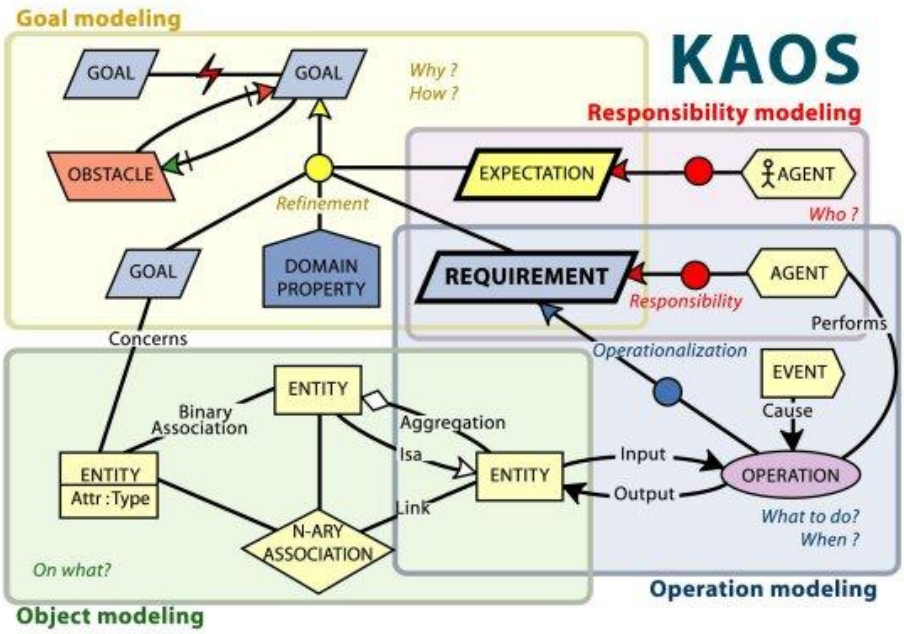

Figura 5-2. Metamodelo KAOS mostrando conceitos dos quatro sub-modelos, suas associações em sua própria representação visual (RESPECT-IT, 2007).

No Modelo de Objetivos estes são refinados, segundo estratégias definidas, desde o objetivo raiz, a Missão do Sistema, até que conceitos indivisíveis sejam identificados, guiados pelo metamodelo que define a estratégia de descoberta. Refinamentos podem ser do tipo AND ou OR. Objetivos com refinamentos tipo AND são satisfeitos quando todos os objetivos que o refinam são satisfeitos. O Modelo de Objetos é um modelo conceitual gerado a partir do metamodelo UML. O Modelo de Operações descreve o comportamento dos agentes a fim de realizar suas responsabilidades na satisfação de requisitos ou de expectativas. Eventos podem disparar ou interromper operações onde objetos (instâncias de Entity) participam seja como

entrada ou saída (Input ou Output). No Modelo de operações também são mostradas as operacionalizações (operationalization) daqueles requisitos que descrevem propriedades dinâmicas do sistema. Devido ao fato de não serem reponsabilidade de agentes de software, expectativas não podem ser operacionalizadas (Figura 5-2).

Os sub-modelos respondem perguntas sobre o sistema. O modelo de Objetivos responde COMO o sistema deve funcionar e as razões subjacentes de sua funcionalidade, isto é, PORQUE. O Modelo de Responsabilidade responde à pergunta QUEM. O Modelo de Operações responde O QUE FAZER e QUANDO, e finalmente o Modelo de Objetos: SOBRE O QUE (RESPECT-IT, 2007; LAMSWEERDE, 2009).

O Modelo de Requisitos produzido no *framework* KAOS é um conjunto de sub-modelos que são construídos usualmente na chamada fase de análise em um modelo de ciclo de vida usual. No Processo Unificado (JACOBSON; BOOCH; RUMBAUGH, 1999) o Modelo de Requisitos é representado principalmente por Casos de Uso. A UML apóia além dessa, outras visões, através de diagramas comportamentais, que podem ser usados para enriquecer o Modelo de Requisitos, como os diagramas de sequência e colaboração de objetos, atividades e estado. Pode-se também lançar mão da modelagem estrutural usando diagramas de classes para representar o modelo conceitual das entidades do sistema a fim de aprimorar ainda mais o modelo de requisitos. Entretanto, o Modelo de Objetivos não tem similar, ou representação em UML, a menos que se desenvolva uma extensão.

O Modelo de Objetivos é importante para a adoção do paradigma, se comparado com outros métodos, na medida em que permite a captura de necessidades em todo tipo de informação disponível, de forma abrangente, de qualquer nível ou natureza (DARDENNE, 1991, 1993; LAMSWEERDE, 2000, 2009). Estas necessidades são submetidas a um processo definido de refinamento até que objetivos atômicos sejam encontrados, podendo ser verificados em relação à critérios de qualidade e validados pelos interessados em relação à sua fidedignidade às necessidades declaradas.

Depois da publicação da UML 1.4 pela OMG (2001), baseada em MOF (*Meta Object Facility*) (OMG, 2006c), a linguagem tornou-se definitivamente um template para a criação de outros metamodelos através de mecanismos de extensão. Os mecanismos leves, através de *profiles* têm capacidades apenas aditivas. Já os mecanismos ditos pesados, adicionam explicitamente meta-classes e outros meta-artefatos, sendo que esta capacidade é dependente do uso de ferramentas e repositórios que suportem MOF. Embora a UML nativa não possa representar um Modelo de Objetivos KAOS, ela pode ser estendida para tal. Este trabalho usou, a fim de consolidar o entendimento do metamodelo e instanciar a ontologia Kuaba de

forma correta, o metamodelo e uma extensão UML de KAOS (HEAVEN; FINKELSTEIN, 2007).

5.2 O MODELO CONCEITUAL KAOS

O *framework* KAOS sugere estratégias e templates a fim de orientar os engenheiros de software na atividade de elicitação de requisitos (RESPECT-IT 2007). Os sub-modelos são projetados segundo uma ordem que parte do Modelo de Objetivos, seguido do Modelo de Objetos, depois Responsabilidade, Operações e finalmente seus relacionamentos. Não há obrigatoriedade imposta pelo metamodelo para o uso dessa ordem. Esta é apenas uma orientação que pode tornar mais fácil o trabalho do engenheiro de software. Pode-se também particionar os modelos em conjuntos de abstrações coesas a fim de facilitar a representação visual e abrandar a complexidade.

5.2.1 Modelo de Objetivos

Ao modelar objetivos, o modelo pode ser iniciado tendo como raiz o objetivo referente à missão do sistema que, mesmo não sendo explícito nos levantamentos, pode ser obtido com facilidade. Entretanto, a descoberta de seus refinamentos diretos pode não ser tão fácil. Muitas vezes não se consegue obter, através das reuniões de levantamento de requisitos e estudos sobre documentos relacionados, refinamentos de maneira tão direta. Esse fato não é crítico, uma vez que KAOS permite que o processo de refinamento comece até mesmo sem o objetivo raiz, sendo que o modelo completo será obtido durante a modelagem.

O processo de descoberta de objetivos é de decomposição e síntese, não importando a ordem. Esse processo é mostrado na Figura 5-3, que ilustra o exemplo de um modelo que tem como ponto de partida uma raiz, a missão do sistema, e consegue obter refinamentos a partir dela. A ilustração indica que o caminho da decomposição é no sentido dos refinamentos para um objetivo respondendo a pergunta de COMO a síntese pode ser obtida, ou seja, os objetivos A Biblioteca empresta livros e revistas [...], A Biblioteca cuida das compras de títulos novos [...], e O bibliotecário é um empregado da Biblioteca que interage com os usuários [...] explicam como satisfazer o objetivo É um sistema de apoio operacional para uma Biblioteca. O sentido da síntese é inverso, do objetivo refinado, respondendo à pergunta PORQUE são necessários esses refinamentos, ou seja, aqueles refinamentos citados explicam porque aquele objetivo é satisfeito.

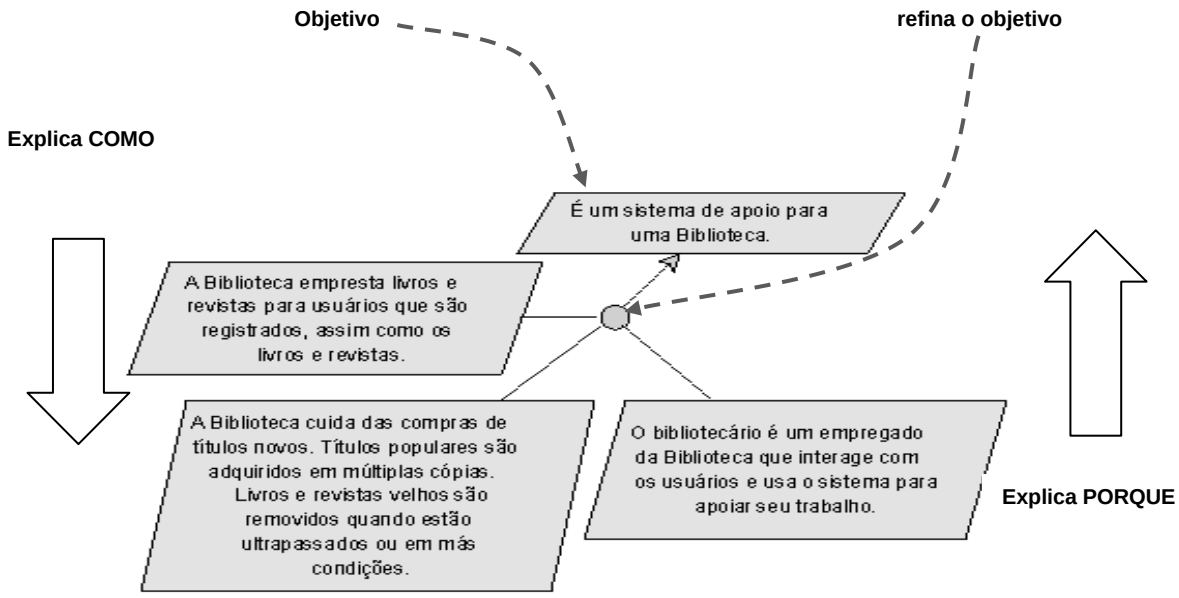

Figura 5-3. Diagrama de objetivos.

A Figura 5-4 mostra parcialmente o metamodelo KAOS apresentando os conceitos usados para o *design* de modelos de objetivos. Os conceitos fundamentais nesse metamodelo são Objetivo (Goal), Objetivo fraco (Sofgoal)[3], Obstáculo (Obstacle), Requisito (Requirement) e Expectativa (Expectation).

Um objetivo é uma declaração prescritiva de intenção que o sistema deve satisfazer através da cooperação entre agentes (LAMSWEERDE, 2009). Objetivos são refinados em associações do tipo AND e OR. Associações do tipo AND obrigam que para a satisfação do objetivo refinado, todos os refinamentos sejam satisfeitos, o que não precisa acontecer se forem do tipo OR.

A representação no metamodelo para essas associações não é clara e não é feita através de um meta-conceito. No caso das associações tipo AND, deve-se reparar no relacionamento entre GRefinement e Objective. Esse elemento Objective pode ser representado por "qualquer" tipo de objetivo, pois dele todos são herdeiros diretos ou indiretos. O tipo OR é representado pelo relacionamento entre GRefinement e Goal, e só é permitido para objetivos intermediários na cadeia de refinamento, ou objetivos fracos (Softgoal).

[3] Softgoal não quer dizer exatamente Objetivo fraco. O termo foi usado para manter a mesma idéia do neologismo de Softgoal em inglês em relação a Goal. Softgoal em KAOS quer dizer que este tipo de objetivo é uma declaração prescritiva de preferências em relação a uma determinada situação, e não uma declaração prescritiva de comportamento.

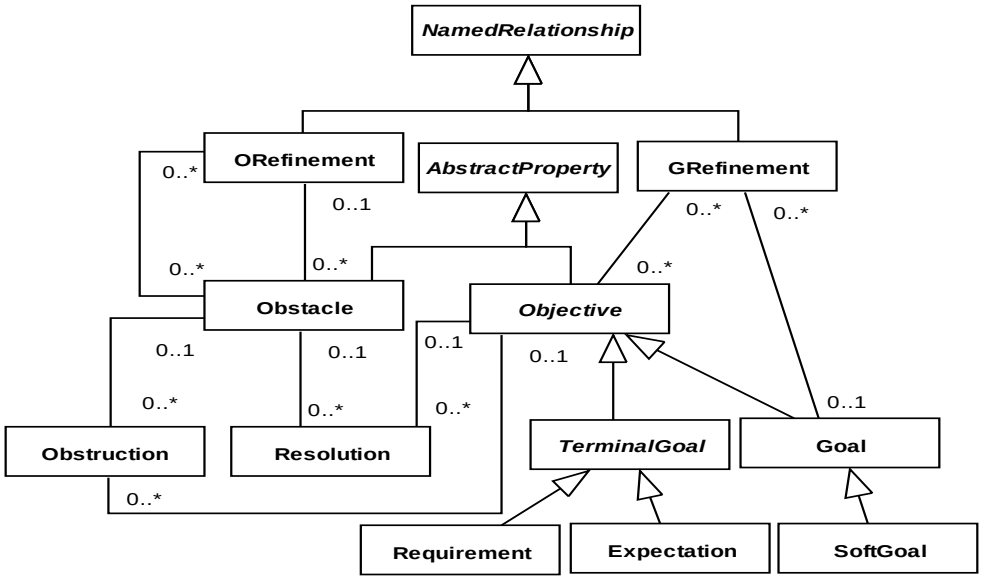

Figura 5-4. Conceitos do metamodelo KAOS relativos ao modelo de objetivos.
Fonte: Objectiver 2.1 (RESPECT-IT, 2007).

O processo de refinamento termina quando requisitos e expectativas são encontrados. A diferença semântica ente eles é que um requisito tem satisfação mandatória. Por outro lado, uma expectativa tem satisfação facultativa e isso deve ser levado em conta na análise. Requisitos e expectativas têm um e apenas um agente como responsável.

Objetivos podem ser conflitantes ou obstruídos. O diagrama mostrado na Figura 5-4 apenas ilustra a situação de obstrução de objetivos. A versão do metamodelo usada neste trabalho (RESPECT-IT, 2007) não inclui a representação de conflitos. Por essa razão, essa dissertação a exemplificou, mostrando apenas uma análise de obstáculo, que será discutida no capítulo 6. A fim de ilustrar a representação de conflitos, é mostrada na Figura 5-5 uma versão do metamodelo apresentada em Lamsweerde (2009) que trata esta lacuna da versão anterior. Nela, os conflitos são tratados sem a existência de um obstáculo. Nesse caso apenas objetivos participam da relação de divergência (Divergence). Conflitos também podem ser abrandados, solucionados ou contornados.

Independentemente da representação das abstrações do metamodelo, a semântica dos conceitos obstáculo e conflito é similar. Entretanto, dois objetivos devem tomar parte em uma relação de conflito. No caso de obstáculos apenas um objetivo toma parte. Um obstáculo causa uma obstrução (Obstruction) a um objetivo, que pode ser resolvida (Resolution) por um abrandamento, uma solução definitiva ou definição de uma maneira, paleativa, de lidar com o

problema. Essas medidas podem ser obtidas através da criação de novos objetivos ou requisitos que tragam algum tipo de resolução para a obstrução.

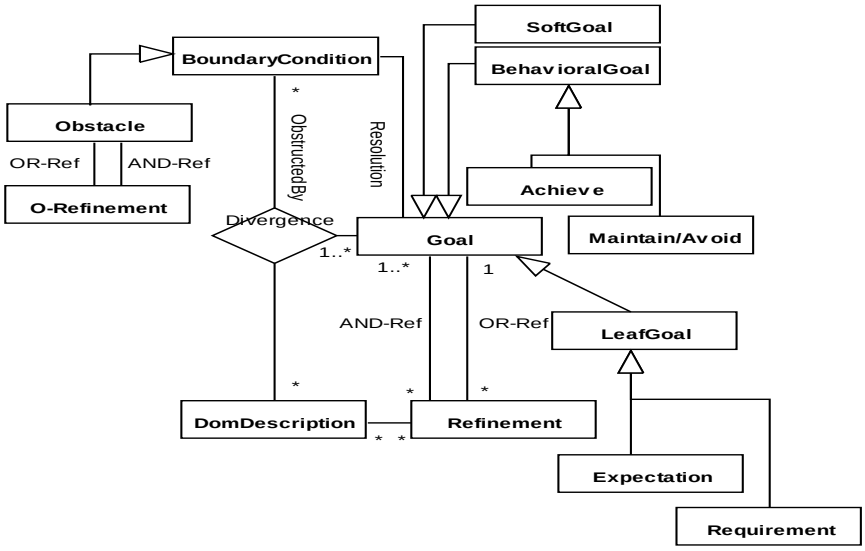

Figura 5-5. Metamodelo KAOS para os conceitos associados a objetivos, revisado por Lamsweerde(2009).

5.2.2 Modelo de Responsabilidades

O Modelo de Responsabilidades descreve as responsabilidades de cada agente do sistema em relação à satisfação de requisitos ou expectativas, e o comissionamento (Assignment) de agentes em relação a objetivos no curso da atividade de modelagem, como mostra o metamodelo parcial da Figura 5-6. Para cada requisito e expectativa é feita a atribuição de responsabilidade. Os diagramas podem ser construídos para cada agente, como uma estrela, ou podem ser feitos como mostra o exemplo da Figura 5-7. Este sub-modelo é importante, por definição, no *framework* KAOS, e na orientação para objetivos, que enfatiza que objetivos devem ser satisfeitos através de agentes (DARDENNE, 1993; LAMSWEERDE, 2000; LETIER, 2001).

O metamodelo da Figura 5-6 prescreve que agentes podem assumir um papel frente a objetivos através de relações de comissionamento e de responsabilidade. Estas relações usualmente dependem do momento no qual a análise de requisitos se encontra. Não é escopo deste trabalho ater-se aos aspectos metodológicos. Porém, vale salientar que o comissionamento é feito em um momento mais cedo, quando ainda não foi atingido o

refinamento final para os objetivos, porém tem-se indicação pelas fontes de informação, que determinado agente contribui para a realização de um objetivo. A atribuição de responsabilidade é feita mais tarde, quando os objetivos já estão refinados até a definição de requisitos e expectativas, onde se estabelece um critério de diagrama em estrela, como citado no parágrafo anterior.

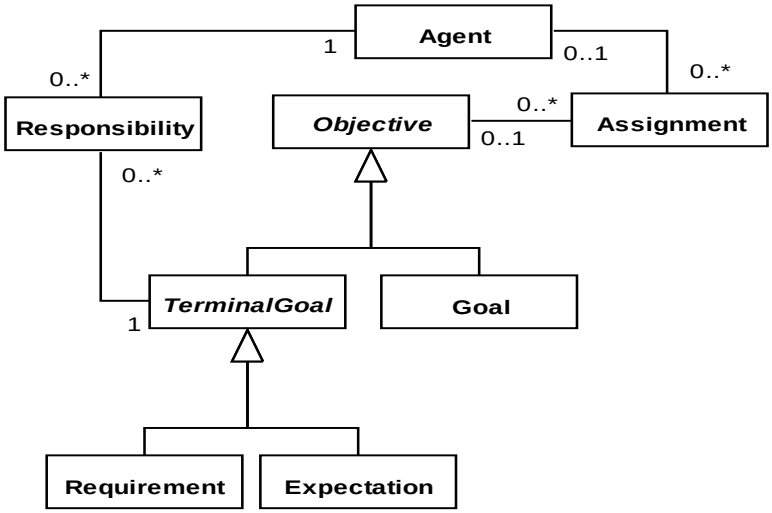

Figura 5-6. Metamodelo parcial de KAOS para os conceitos relativos a agentes.
Fonte: Objectiver 2.1 (RESPECT-IT, 2007).

Ilustrando um exemplo do cenário de empréstimo de livros e revistas, o diagrama da Figura 5-7 mostra agentes de software e de ambiente (Bibliotecário) e suas responsabilidades. Note que um agente humano, ou seja, um agente de ambiente, como é conceituado em KAOS, tem uma relação de responsabilidade com a expectativa Os dados sobre o empréstimo são fornecidos (o paralelogramo mais escuro). Os demais agentes são de software e têm responsabilidade sobre requisitos (os paralelogramos em cinza mais claro com contorno marcante, diferenciando-os do objetivo que, tem uma linha mais fina).

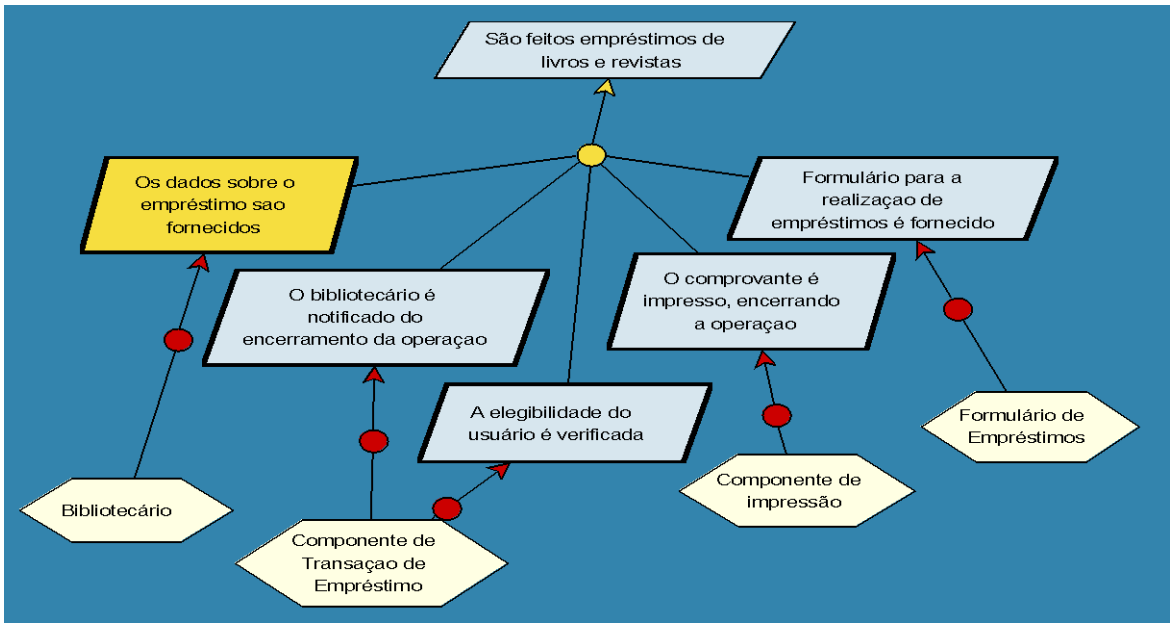

Figura 5-7. Diagrama de Responsabilidade.

5.2.3 Modelo de Operações

O modelo de operações descreve todos os comportamentos que agentes precisam exibir para satisfazer os requisitos pelos quais são responsáveis, e que foram definidos no processo de refinamento do modelo de objetivos. Esses comportamentos são expressos em termos de operações.

O metamodelo apresentando os conceitos que devem ser instanciados no modelo de operações é mostrado na Figura 5-8. Os conceitos são as operações (Operation) propriamente ditas, eventos (Event) e objetos (Object). A execução de uma operação por um agente é representada pelo conceito Performance. Operações podem ser iniciadas ou interrompidas por eventos (Cause), ou podem ainda criar um novo evento (Output) para iniciar outra operação. Objetos são entradas e saídas de operações (Input e Output).

O modelo de operações é um integrador de elementos dos demais sub-modelos do modelo conceitual KAOS através do conceito de operacionalização (Operationalization), que quer dizer “satisfação do requisito”. Uma operacionalização envolve os elementos operação, objeto, agente e requisito. As instâncias desse conceito no modelo de requisitos, ou seja, no espaço do problema, definem restrições arquitetônicas aplicáveis ao modelo de projeto, criando uma ponte entre o modelo de requisitos e o de implementação, definindo possibilidades para esses modelos no espaço da solução.

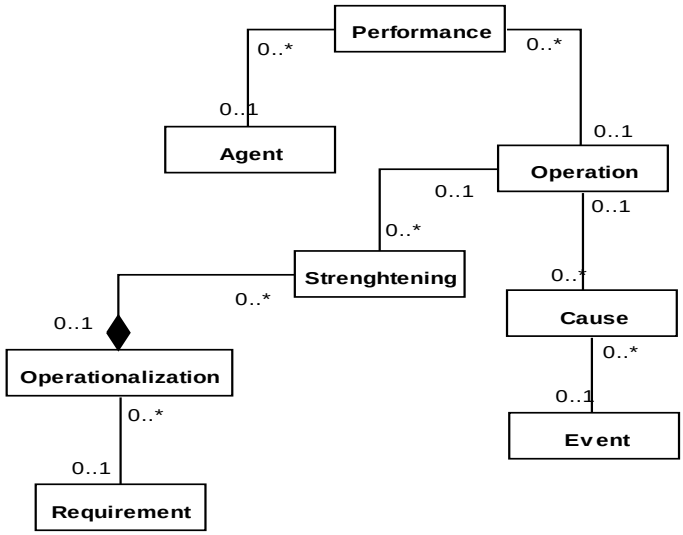

Figura 5-8. Metamodelo parcial de KAOS para os conceitos relativos a operações.
Fonte: Objectiver 2.1 (RESPECT-IT, 2007).

A Figura 5-9 ilustra um cenário parcial que trata dos empréstimos de livros e revistas, como mostrados na Seção 5.2.2, na visão da atribuição de responsabilidades aos agentes. Aqui é apresentado um diagrama misto para que as operacionalizações fiquem claras, assim como as relações entre operações (elipses), objetos (retângulos) e eventos (mostrados como uma seta grossa).

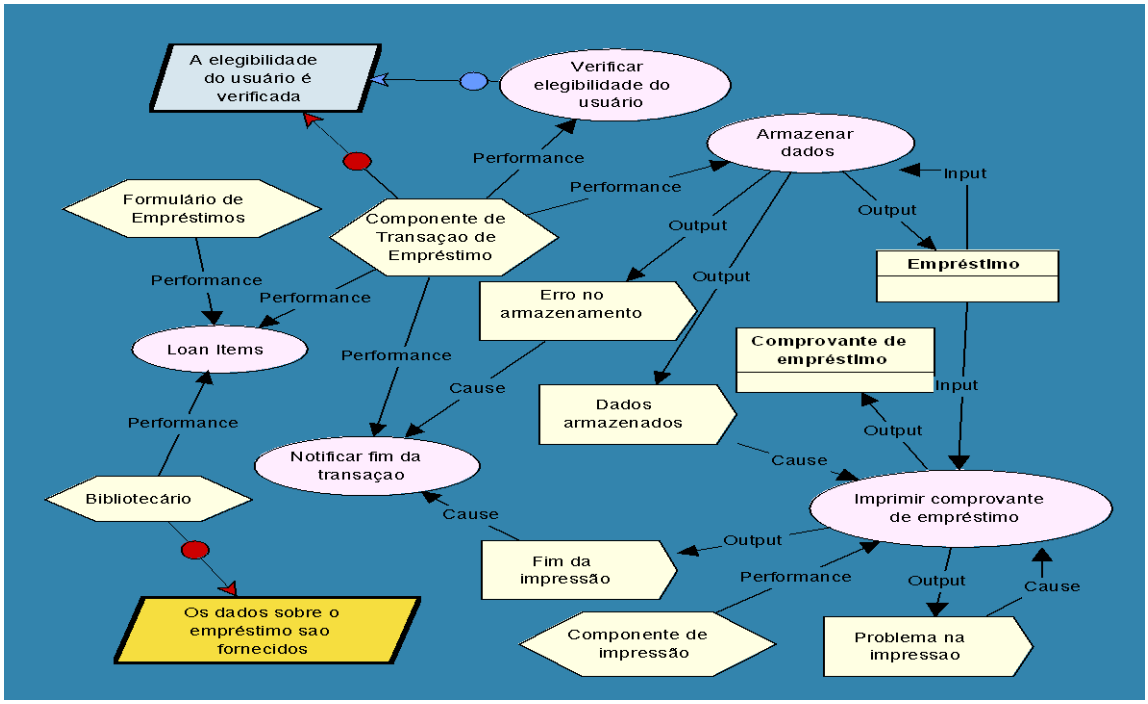

Figura 5-9. Modelo parcial de operações.

Os hexágonos significam agentes na representação visual de KAOS. Na ilustração da Figura 5-9 existem agentes de software e de ambiente. O Componente de Transação de Empréstimos e o Componente de impressão são agentes de software e o Bibliotecário um agente de ambiente. Agentes executam (relação Performance) operações, como o componente de transação notifica o fim da transação, quando recebe um evento de finalização da impressão do comprovante de empréstimo. Eventos são representados por setas grossas e podem ser saídas (Output) de uma operação levando outra a ser executada (Cause). O diagrama mostra que o engenheiro de software define o problema com intenções claras sobre aspectos arquitetônicos, segurança do processo, e outras que guiam a definição do design de projeto. Um exemplo pode ser observado na ilustração em que a operação de armazenamento de dados, quando concluída, dispara (relação Output) um evento que acionará (relação Cause) a operação de impressão do comprovante, ou seja, esse comprovante só pode ser liberado após os dados estarem armazenados. Isso garante a integridade dos dados e do processo, pois quando o bibliotecário entrega o livro ou revista ao usuário com o comprovante, os dados sobre aquela operação já estão armazenados e o processo completado. A situação alternativa mostrada na ilustração é da ocorrência de erro na armazenagem, que encerra a transação sem armazenamento e notificando o fim da transação, também garantindo a integridade do processo, dado que o comprovante não pode ser impresso e o processo não será completado no sistema, nem fisicamente com o comprovante e o livro ou a revista entregues ao usuário. Fica claro, que esse modelo precisa ser refinado para tratar essas exceções, mas para a finalidade tratada aqui essa situação é suficiente. As relações entre operações e requisitos que têm um círculo cinza mais claro no meio da linha são as operacionalizações e a com o círculo escuro é de responsabilidade. A ilustração mostra que a operação que verifica a elegibilidade do usuário operacionaliza o requisito referente à necessidade dessa validação, através da participação do agente de transação, mostrada pela seta que liga a operação ao requisito, no topo da Figura 5-9. Poderiam também estar ilustrados objetos como o próprio usuário a fim de participar da operacionalização. Neste caso, entende-se pelo modelo que isso não é necessário.

6 REPRESENTAÇÃO DE DESIGN RATIONALE USANDO A ABORDAGEM KUABA E O METAMODELO KAOS

O desenvolvimento do produto de software é guiado pelas necessidades de um grupo de pessoas que buscam através de seu uso obter valor para elas seja nos negócios, na vida pessoal ou qualquer outro contexto. Essas necessidades são usualmente uma lista de necessidades de diversas naturezas, declaradas por essas pessoas diferentes umas das outras, em diferentes situações. Além disso, essas necessidades ainda podem ser conflitantes. A definição do problema deve ser obtida a partir desse conjunto desconexo de conhecimentos e intenções. Os engenheiros de software, juntamente com esse grupo de pessoas, têm a missão de coletar essa informação e definir o problema a ser resolvido, assim como formular sua solução.

A representação de *design rationale* durante a modelagem de requisitos cria a possibilidade de usar o conhecimento do grupo em tarefas da Engenharia de Requisitos. O objetivo desse trabalho é investigar a representação de *design rationale* usando a abordagem Kuaba para modelos de requisitos, e estudar as possibilidades de usar o conhecimento registrado para apoiar a prática da Engenharia de Requisitos. Para tanto, em primeiro lugar, a abordagem deve ser adequada para representar o conhecimento aplicado pelos engenheiros de software ao modelar requisitos. Esta etapa da pesquisa dedica-se a essa investigação.

Neste capítulo são apresentados e discutidos modelos da Engenharia de Requisitos e suas representações de *design rationale* usando a abordagem Kuaba. Esta é uma abordagem para representação de *design rationale* que trabalha sobre o domínio de design baseado em modelo, ou seja, onde modelos descrevem os artefatos desse domínio. Neste trabalho, os modelos de requisitos são obtidos através do processo de instanciação do metamodelo do *framework* KAOS, agregando, assim, sua semântica aos artefatos produzidos. Os exemplos apresentados têm como objetivo simular tarefas reais de design dos engenheiros de software

no seu trabalho de especificação. Aqui é mostrado como a instância da ontologia Kuaba é construída a partir do modelo conceitual de KAOS, gerando representações de *design rationale* para todos os modelos preconizados pelo *framework*. O uso de um metamodelo adequado para descrever artefatos do modelo de requisitos, capaz de preservar a semântica de conceitos complexos, voláteis e de alto nível de abstração, é importante para a obtenção de bons resultados na investigação.

6.1 METODOLOGIA

O planejamento desta investigação envolve duas atividades: (a) simular a tarefa real de um engenheiro de software ao modelar requisitos e (b) criar um conjunto de testes de modelagem de requisitos e de representação de *design rationale* usando os conceitos mais importantes do modelo conceitual do framework KAOS, abrangendo os seus quatro submodelos. Embora não exaustivos, os testes projetados são suficientes para atingir o objetivo deste trabalho e simular a situação desejada. Este planejamento permite verificar a possibilidade de representação de *design rationale* utilizando a abordagem Kuaba e os conceitos do meta-modelo KAOS. Permite também investigar como as representações de *design rationale* obtidas podem apoiar a atividade da Engenharia de Requisitos.

Os testes de modelagem dividem-se em três categorias. A primeira usa o método usual de representar *design rationale* durante a tarefa de modelagem. Neste caso, o engenheiro de software escolhe a melhor solução de design para o artefato sendo produzido, a representa no modelo e, em seguida, faz a representação de *design rationale* para esta solução. A segunda categoria é da representação de *design rationale* depois de terminada a tarefa de modelagem. O engenheiro de software trabalha nessa representação como se estivesse documentando o raciocínio utilizado nas escolhas de design para os artefatos produzidos. A terceira categoria é a de não modelar, representando o *design rationale* diretamente a partir das declarações sobre o sistema desejado. Esta situação supõe um conhecimento profundo do meta-modelo utilizado, uma vez que deve-se instanciar a ontologia Kuaba a partir dos conceitos do meta-modelo, sem representá-los em um modelo.

As seções seguintes apresentam os resultados desses testes, detalhando os modelos produzidos e suas respectivas representações de *design rationale*. Na Seção 6.2 são apresentados os resultados dos testes de modelagem de requisitos realizados para o domínio de uma biblioteca, e na Seção 6.3 os resultados para o domínio de submissão e revisão de artigos para uma conferência. Os modelos de requisitos foram produzidos usando a

ferramenta comercial Objectiver 2.1, cedida pela Respect-IT da Bélgica. A representação gráfica dos elementos do vocabulário da ontologia Kuaba e as representações de *design rationale* foram produzidas usando uma extensão da UML na ferramenta comercial Enterprise Architect 7.5 da empresa australiana Sparx Systems. A reprodução do metamodelo KAOS e dos elementos da ontologia Kuaba usaram a notação UML, também nesta última ferramenta.

6.2 DESIGN RATIONALE PARA O DOMÍNIO DE BIBLIOTECA

O primeiro conjunto de representações de *design rationale* apresentado neste capítulo é para o domínio de uma Biblioteca. A Tabela 6-1 apresenta uma lista de declarações feitas sobre o sistema de apoio operacional para uma Biblioteca que se tem intenção de desenvolver (ERIKSSON; PENKER, 1998). Nem todos os itens declarados serão abordados neste trabalho. Pode-se observar que o primeiro é a missão do sistema e o segundo aquele que representa o processo principal que será apoiado pelo sistema. O item de número 6 prescreve preferências em relação à atualização dos dados (“facilmente”) e o penúltimo de interoperabilidade. Esses últimos descrevem características não funcionais, embora possa parecer que no item 6 seja mostrada uma característica funcional de criar, alterar e apagar informações sobre as entidades do sistema. No entanto, esta funcionalidade já é abordada no item 2, uma vez que neste item é dito que “os usuários são registrados, assim como livros e revistas”, predominando no item a expressão “facilmente”.

Tabela 6-1. Declarações sobre o Sistema de Operação da Biblioteca

1	É um sistema de apoio para uma Biblioteca.

2	A Biblioteca empresta livros e revistas para usuários que são registrados, assim como os livros e revistas.
3	A Biblioteca cuida das compras de títulos novos. Títulos populares são adquiridos em múltiplas cópias. Livros e revistas velhos são removidos quando estão ultrapassados ou em más condições.
4	O bibliotecário é um empregado da Biblioteca que interage com os usuários e usa o sistema para apoiar seu trabalho.
5	Um usuário pode reservar um livro ou revista que não está disponível na Biblioteca. Quando o livro emprestado for devolvido ou algum outro exemplar for comprado o usuário deve ser notificado. A reserva é cancelada quando o usuário faz o empréstimo do livro ou revista, ou através de um procedimento de solicitação de cancelamento.
6	O bibliotecário pode facilmente criar, alterar e apagar informações sobre títulos, usuários, empréstimos e reservas no sistema.
7	Pode operar em todos os ambientes computacionais mais conhecidos e tem interface gráfica moderna.
8	O sistema pode ser facilmente estendido com novas funcionalidades.

O sistema de apoio para a Biblioteca deve ser um produto usado apenas pelo bibliotecário na sua estação de trabalho. Os usuários interagem com ele indo à recepção da Biblioteca, onde podem fazer empréstimos de volumes de livros ou revistas. Podem ainda obter informações sobre a disponibilidade do item de interesse, ou fazer reservas, no local ou pelo telefone. As compras de novos títulos e assinaturas, assim como o descarte daqueles sem condições de uso ou ultrapassados, também podem ser realizados pelo bibliotecário apoiado pelo sistema.

6.2.1 Representação de *design rationale* para o Modelo de Objetivos

A representação de *design rationale* é modelada a partir da instância da ontologia Kuaba para o modelo conceitual do *framework* KAOS. O metamodelo parcial, ilustrado na Figura 6-1, mostra apenas os conceitos do modelo de objetivos do *framework* KAOS.

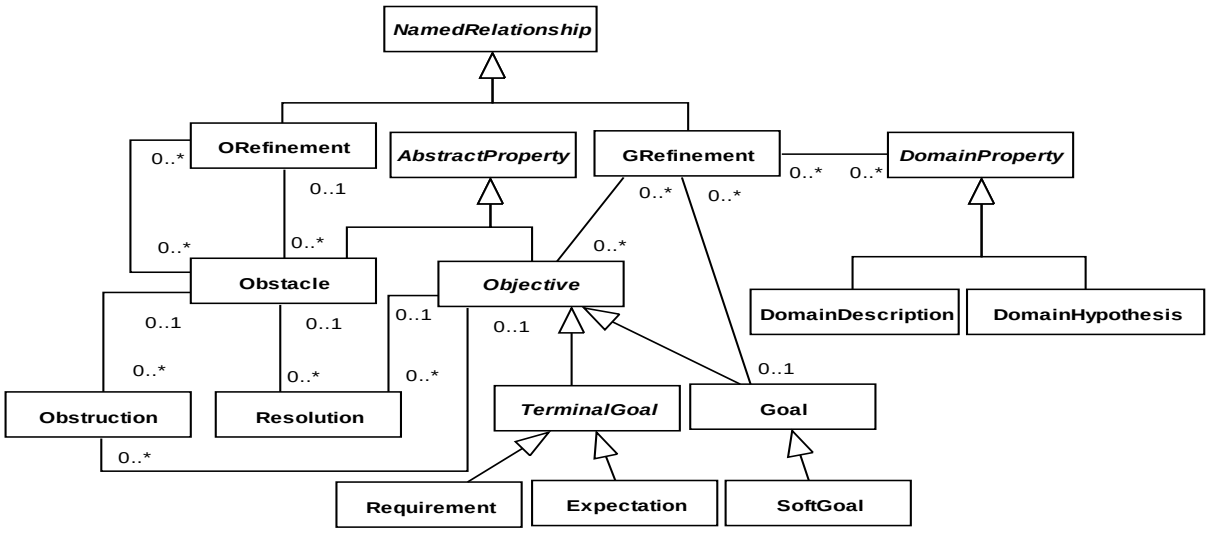

Figura 6-1. Conceitos do metamodelo KAOS para modelos de objetivos.

Para instanciar os elementos de raciocínio da ontologia Kuaba incorporando a semântica do metamodelo usa-se os elementos concretos do metamodelo para instanciar as idéias, e os conceitos que referem-se a relacionamentos dão origem a questões. Neste caso os conceitos Goal, GRefinement, Obstacle, ORefinement, SoftGoal, DomainDescription, DomainHypothesis, Requirement e Expectation dão origem na instância Kuaba às idéias Objetivo, Refinamento de Objetivo, Obstáculo, Refinamento de Obstáculo, Objetivo Fraco, Descrição de Domínio, Hipótese de Domínio, Requisito e Expectativa, sendo que Resolution e Obstruction dão origem às questões Resolução? e Obstrução? incorporando a semântica dos elementos do metamodelo que são resultado de relacionamentos. Isso significa que essas idéias e questões serão usadas para representação de *design rationale* do modelo de objetivos.

A Figura 6-2 ilustra graficamente o *design rationale* registrado durante o design do diagrama parcial do modelo de objetivos do domínio da Biblioteca mostrado na Figura 6-4. Os conceitos de domínio exibidos tanto na representação de *design rationale*, como no modelo parcial de objetivos, são diretamente retiradas dos itens 1, 2 e 6 da lista de declarações sobre o sistema da Biblioteca da Tabela 6-1. A Figura 6-3 pode ser usada como guia para a representação gráfica dos elementos de raciocínio da ontologia Kuaba, nos diagramas de representação de *design rationale*, como o da Figura 6-2.

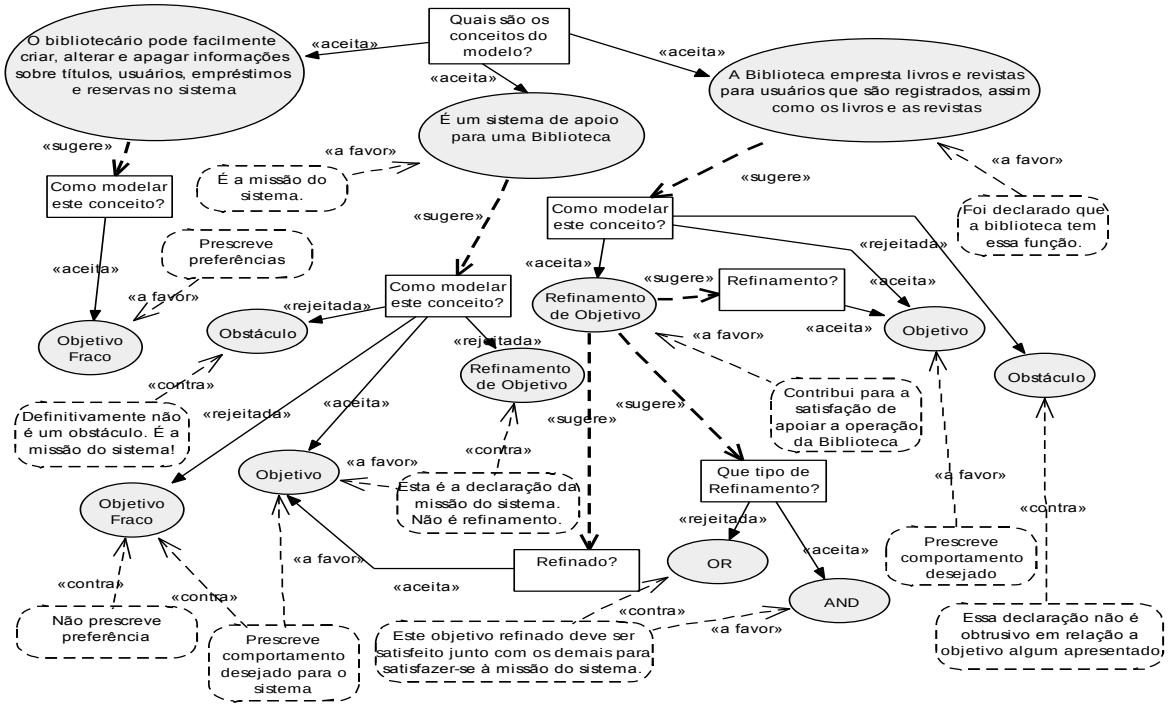

Figura 6-2. Representação de *design rationale* para um diagrama do modelo de objetivos.

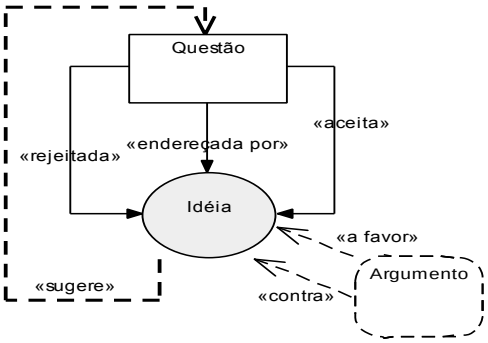

Figura 6-3. Representação gráfica dos elementos de raciocínio da ontologia Kuaba.

A representação de *design rationale* é sempre iniciada pela questão (representada como um retângulo no topo da Figura 6-2) que indaga sobre que conceitos do domínio são conhecidos. As idéias, representadas como elipses, que endereçam essa questão são os itens É um sistema de apoio para uma Biblioteca, A Biblioteca empresta livros e revistas para usuários registrados, assim como os livros e revistas, e O bibliotecário pode facilmente criar, alterar e apagar informações sobre títulos, usuários, empréstimos e reservas no sistema.

Essas idéias de domínio sugerem questões de como modelar os conceitos. Quando ainda não se tem uma solução para o design a associação entre a questão e a idéia é adornada com o rótulo <<endereçada por>>. Feita a argumentação e tendo o engenheiro de software

decidido por aceitar uma solução e rejeitar outras, este rótulo é mudado para <<aceita>> ou <<rejeitada>>. As idéias de design são obtidas da semântica e das possibilidades de navegação descritas pelo metamodelo KAOS que guia o design (Figura 6-1).

A rigor, quando a questão Como modelar este conceito? é feita, todas as opções de design disponíveis no metamodelo são consideradas na representação de *design rationale* ilustrada na Figura 6-2 para responder à questão para a idéia de domínio É um sistema de apoio a uma Biblioteca. No entanto, para uma melhor visualização são ilustradas apenas algumas dessas idéias: Obstáculo, Objetivo Fraco, Objetivo e Refinamento de Objetivo. A idéia de obstáculo é contra argumentada, uma vez que a missão do sistema não deve significar uma obstrução à satisfação de um objetivo. De forma similar, a característica de ser prescrição de um comportamento desejado, descarta um objetivo fraco, pois não se trata de uma prescrição de preferências. A missão do sistema não é certamente um refinamento de outro objetivo, tomando-se assim as decisões de aceitar a idéia Objetivo e rejeitar as demais.

Durante a elaboração da representação de *design rationale* para o modelo de objetivos, são registrados argumentos (retângulos com cantos arredondados e contornados por uma linha pontilhada na Figura 6-2) contra e a favor de cada idéia de solução; as decisões tomadas pelo engenheiro de software são ilustradas por associações direcionadas com os rótulos aceita e rejeitada nas setas entre questões e idéias.

No caso da idéia de domínio A Biblioteca empresta livros e [...], tem-se uma situação um pouco diferente, apesar das argumentações em relação às ideías de obstrução e objetivo fraco terem o mesmo significado. Nessa situação há um objetivo, que é argumentado a favor, assim como a idéia de refinamento de objetivo. O metamodelo prescreve que um refinemento de objetivo é também um objetivo. Navegando pelo metamodelo (Figura 6-1), percebe-se que a associação de refinamento possui duas terminações que devem ser expressas como questões na instância da ontologia Kuaba. Essas questões são Qual é o refinamento? e Refina qual objetivo?. Nesse caso a idéia A Biblioteca empresa livros e [...] é aceita como objetivo e como refinamento de É um sistema de apoio a uma Biblioteca, sendo as demais idéias rejeitadas. As justificativas para as decisões tomadas sempre derivam dos argumentos. Justificativas não são representadas graficamente nos exemplos de *design rationale*, mas são registradas através da representação nas linguagens formais F-Logic ou OWL.

Uma idéia de refinamento de objetivo sugere questões sobre os objetivos que são as terminações da associação de refinamento (Refina qual objetivo?, Qual é o refinamento?). Além dessas questões, ainda é sugerida a questão sobre o tipo de refinamento (AND/OR), a fim de definir a regra de satisfação do objetivo. Na Figura 6-2, a questão sobre a terminação do refinamento é respondida pelo próprio objetivo e a questão sobre qual é o objetivo refinado, nesse caso, é respondida pela missão do sistema. A argumentação dada para o tipo de refinamento leva em conta que esse objetivo, assim como outros que refinem a missão do sistema, deve ser satisfeito para que essa missão seja satisfeita. Sendo assim, as decisões tomadas são de aceitar o refinamento do tipo AND e de rejeitar o do tipo OR. As decisões ilustradas na representação de *design rationale* da Figura 6-2 resultam no diagrama parcial de objetivos mostrado na Figura 6-4.

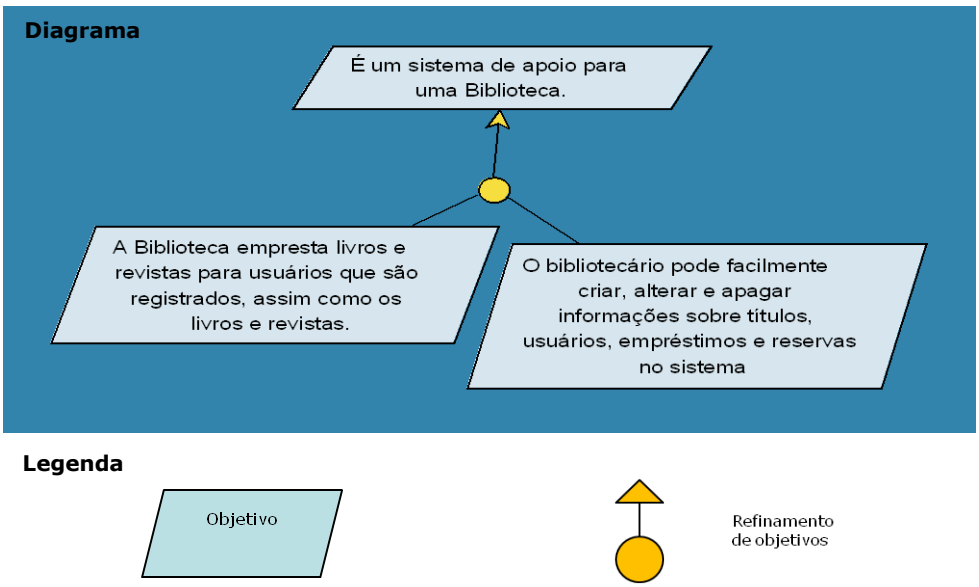

Figura 6-4. Diagrama de objetivos mostrando a raiz do grafo do sistema da Biblioteca.

A Figura 6-6 mostra a representação de *design rationale* para a modelagem do item 2 da Tabela 6-1 e seu refinamento, que resulta no diagrama parcial de objetivos ilustrado na Figura 6-5. Como mencionado anteriormente na explicação sobre a representação de *design rationale* ilustrada na Figura 6-2, quando uma idéia de refinamento endereça a questão de como modelar determinado conceito, ela sugere questões sobre as suas terminações (Refina qual objetivo? e Qual é o refinamento?) e sobre qual o tipo de refinamento, se do tipo AND, ou do tipo OR. De acordo com o metamodelo ilustrado na Figura 6-1, há a possibilidade de que esse refinamento seja terminal, indicando o fim do processo de refinamento, e as idéias de

requisitos ou expectativas devem endereçar a questão Qual é o refinamento?. Isso não implica que uma questão na representação de *design rationale*, tal como Terminal?, deva ser sugerida pela idéia de refinamento, uma vez que a associação é de generalização/especialização ("*is a*") de classes concretas, indicando que os conceitos de objetivo do tipo terminal, requisito e expectativa, são instanciados como idéias endereçando diretamente as questões sobre as terminações do refinamento. Isto é ilustrado na Figura 6-6 para as idéias Requisito e Expectativa endereçando a questão Qual é o refinamento?.

A diferença semântica entre os conceitos Requisito e Expectativa é significativa. Requisitos são responsabilidades de agentes de software havendo garantia de sua satisfação. Em contrapartida, expectativas são responsabilidades de agentes do ambiente, onde seres humanos são classificados no *framework* KAOS, significando que é esperado do agente que ele realize sua responsabilidade, embora isto não seja garantido.

A verificação da elegibilidade do usuário, que é mostrada bem abaixo da questão raiz na Figura 6-6, deve ser garantida. Portanto a idéia de requisito para modelar esse conceito é aceita. Isso não acontece com o conceito "os dados sobre o empréstimo são fornecidos", uma vez que estes podem ser registrados pelo usuário ou não. Isso também está refletido nos argumentos mostrados nesse exemplo. Requisitos e expectativas são mostrados na representação de *design rationale* da Figura 6-6 no canto inferior direito e pode ser notado que dessas idéias não há mais refinamentos. Esses conceitos são representados na Figura 6-5 por paralelogramos com contorno de linhas mais grossas, sendo os requisitos exibidos na cor azul clara e as expectativas em cor mostarda.

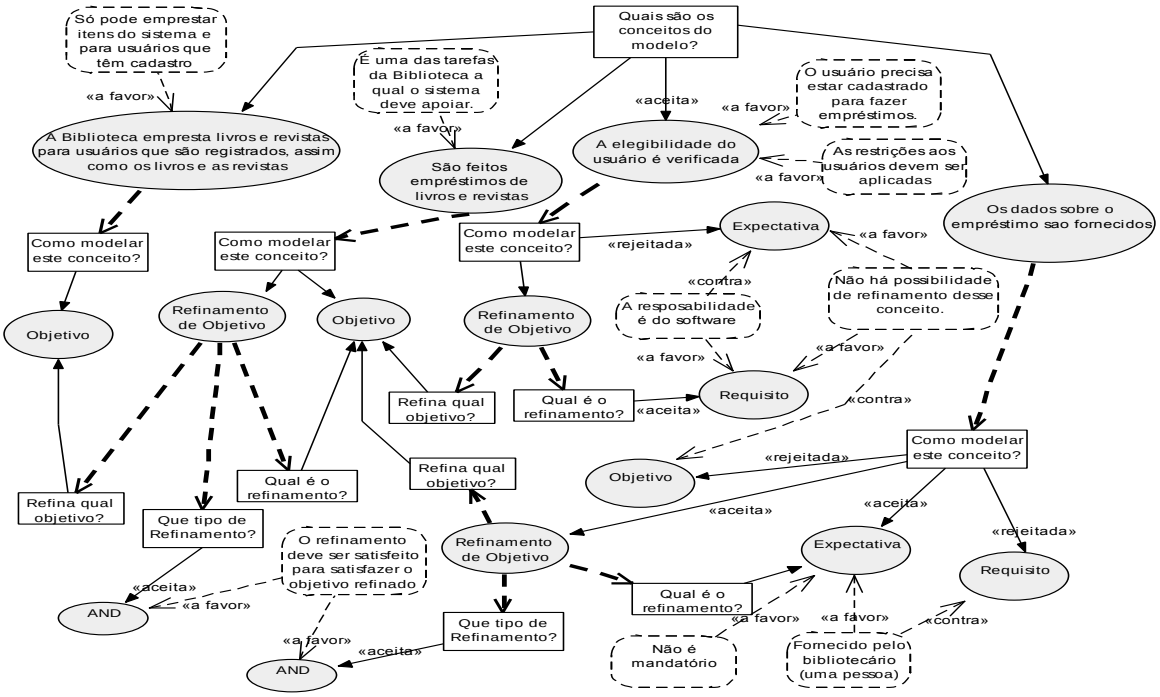

Figura 6-6. Representação de *deisgn rationale* para um refinamento que atinge o limite.

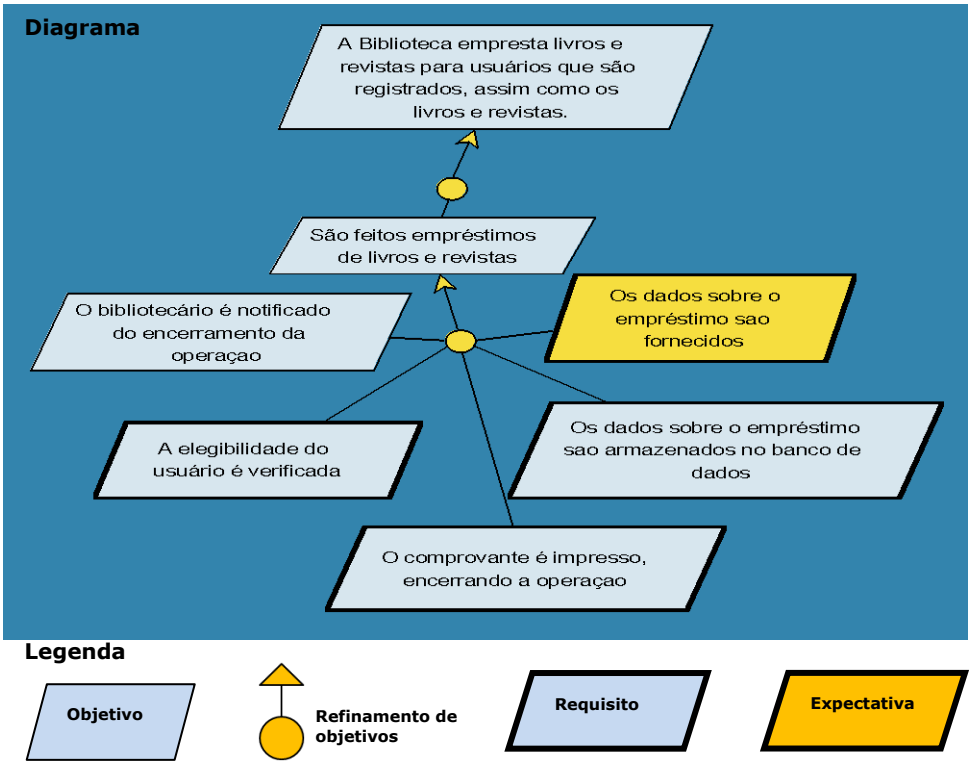

Figura 6-5. Refinamento de objetivos para o sistema da Biblioteca.

A análise de obstáculos é uma tarefa importante na atividade de modelagem de requisitos no *framework* KAOS, na qual é possível prever situações que obstruam ou impeçam a consecução de objetivos. A representação de *design rationale* da Figura 6-7 apresenta o raciocínio que analisa a situação mostrada no diagrama da Figura 6-8, em que a notificação de conclusão da transação pode ser obstruída. O exemplo mostra uma obstrução devido a falta de inteligibilidade da notificação, que é refinada indicando que este problema pode ocorrer devido ao tamanho da caixa de diálogo da notificação. Novos objetivos ou requisitos podem surgir a fim de prevenir a obstrução ou pelo menos abrandá-la. Nesse caso a idéia de domínio que define um tamanho mínimo para a caixa de diálogo é um requisito que previne o acontecimento.

Uma nova situação pode também ser observada na representação de *design rationale* da Figura 6-7, em relação à questão de como modelar o conceito, sugerida pela idéia A caixa de diálogo é muito pequena, que tem como idéias de solução aceitas Refinamento de Obstáculo e Obstáculo. A idéia de obstáculo sugere uma revisão nos conceitos do domínio (mostrada pela seta com adorno <<sugere>> que "volta" para a questão raiz) que traz uma nova idéia (que não existia no modelo anteriormente) que será resolução para esse obstáculo. Como pode ser observado na ilustração, essa nova idéia de domínio propõe um tamanho definido para a caixa de diálogo e tem argumento a favor, tanto para ela quanto para a idéia de requisito que endereça a questão de como modelá-la. Vê-se também que a idéia Requisito ainda sugere a questão Resolução de? que é endereçada pela idéia do obstáculo anterior. Essa proposta de solução é aceita.

A análise de obstáculos é parte da modelagem de objetivos. A estratégia usada é a de testar cada objetivo a fim de verificar situações onde eles podem não ser satisfeitos.

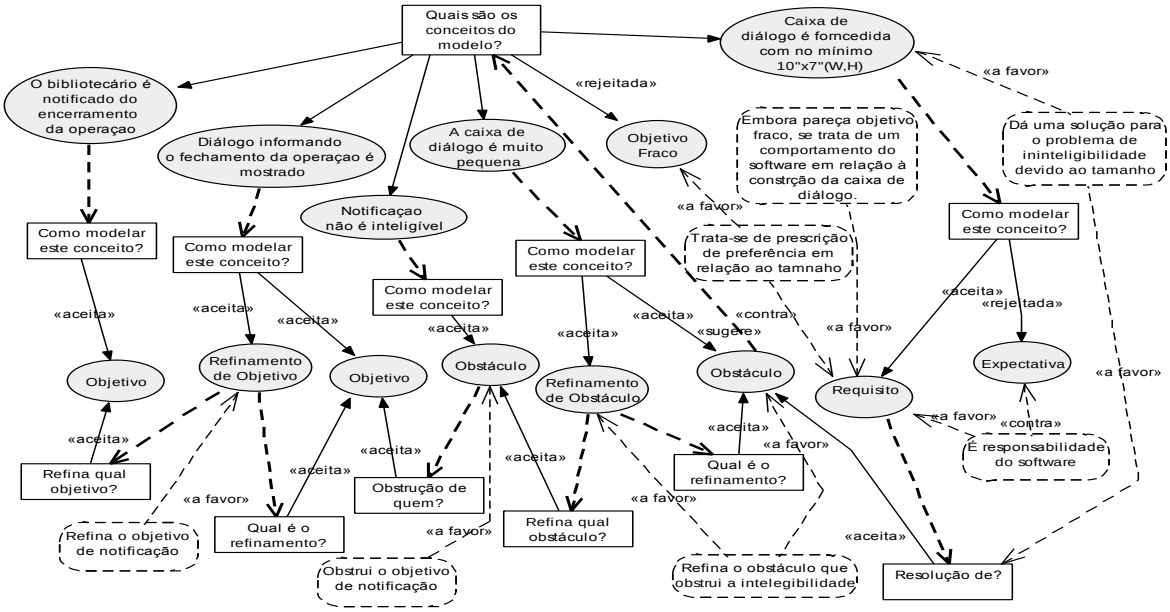

Figura 6-7. Representação de design rationale para o diagrama de obstáculos da Biblioteca.

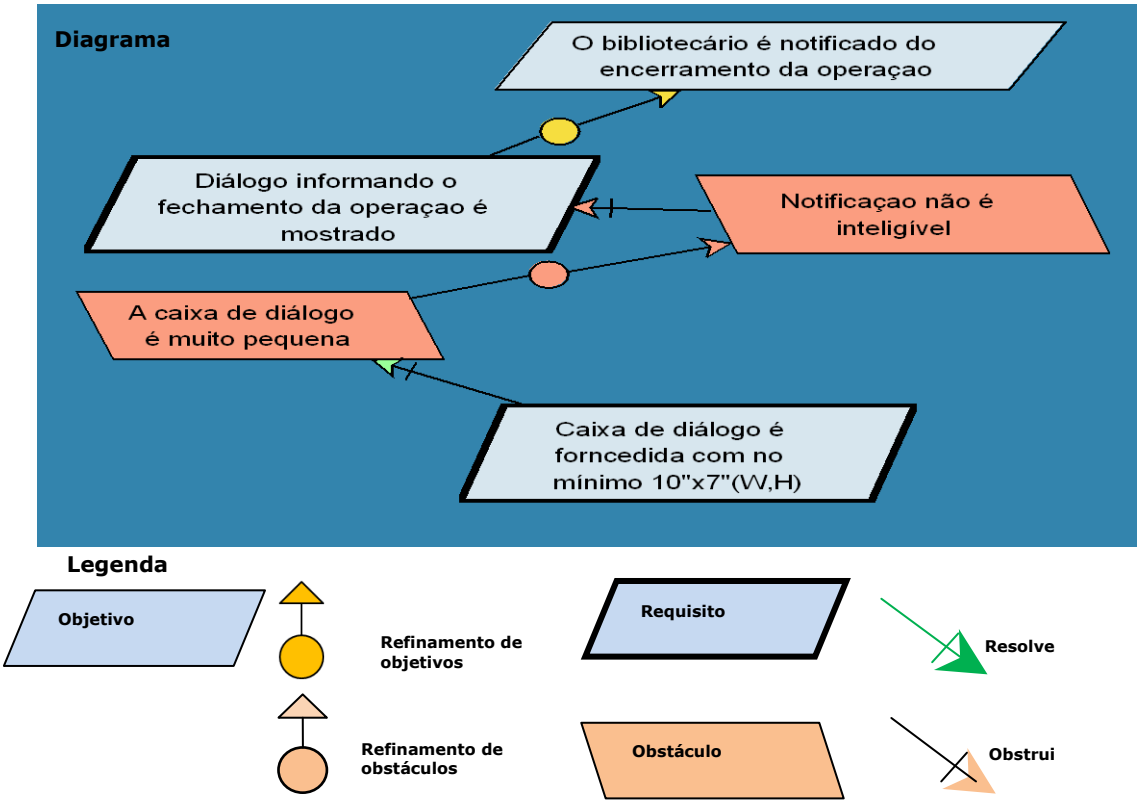

Figura 6-8. Análise de obstáculos para o sistema da Biblioteca.

6.2.2 Representação de *Design Rationale* para o Modelo de Agentes

O diagrama de classes UML ilustrado na Figura 6-9 mostra o metamodelo parcial de KAOS para os conceitos do modelo de agentes, também conhecido no *framework* KAOS como modelo de responsabilidades. Esse diagrama apresenta os conceitos de agentes (Agent), objetivos (neste caso traduzido de Objective e não de Goal) e Requirement (Requisito). O metamodelo KAOS, na versão estudada (RESPECT-IT, 2007), faz distinção sobre as regras de separação das responsabilidades dos agentes em relação a requisitos e objetivos de acordo com a metodologia KAOS. Isso é mostrado nos meta-relacionamentos Responsibility e Assignment. A associação de responsabilidade é permitida apenas entre agentes e requisitos (instâncias de Requirement), sendo que Assignment relaciona qualquer tipo de agente com um objetivo (conceito Objective) que ainda não tenha sido completamente refinado. Na representação de *design rationale* para o metamodelo KAOS da Figura 6-10 as meta-associações Responsibility e Assignment são representadas pelas questões É Responsabilidade de? e É atribuição de?, respectivamente.

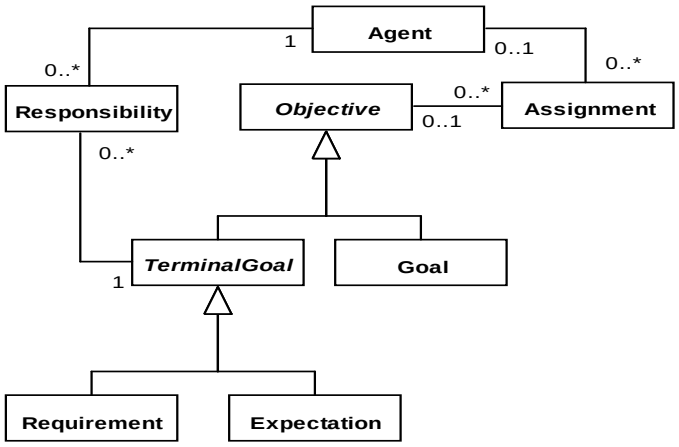

Figura 6-9. Conceitos do metamodelo KAOS para modelos de agentes ou responsabilidades.

A Figura 6-10 mostra a representação de *design rationale* de uma visão parcial do modelo de agentes no domínio da Biblioteca, ilustrado na Figura 6-11. Nessa representação, as idéias de domínio Bibliotecário e Componente de Transação de Empréstimo têm a idéia de design Agente aceita como solução de modelagem. Componente de Transação de Empréstimo é um agente de software e Bibliotecário um agente de ambiente, conceito no qual as pessoas são classificadas no *framework* KAOS. O objetivo São feitos empréstimos de livros e revistas é atribuição do bibliotecário, uma vez que essa é a sua função e que este objetivo não está

totalmente refinado, sendo vedada pelo metamodelo a possibilidade de uma associação de responsabilidade. Por outro lado, o componente de transação de empréstimos é responsável pelo requisito que prescreve que a elegibilidade do usuário deve ser verificada para que ele possa fazer empréstimos. Requisitos têm satisfação obrigatória no *farmework* KAOS.

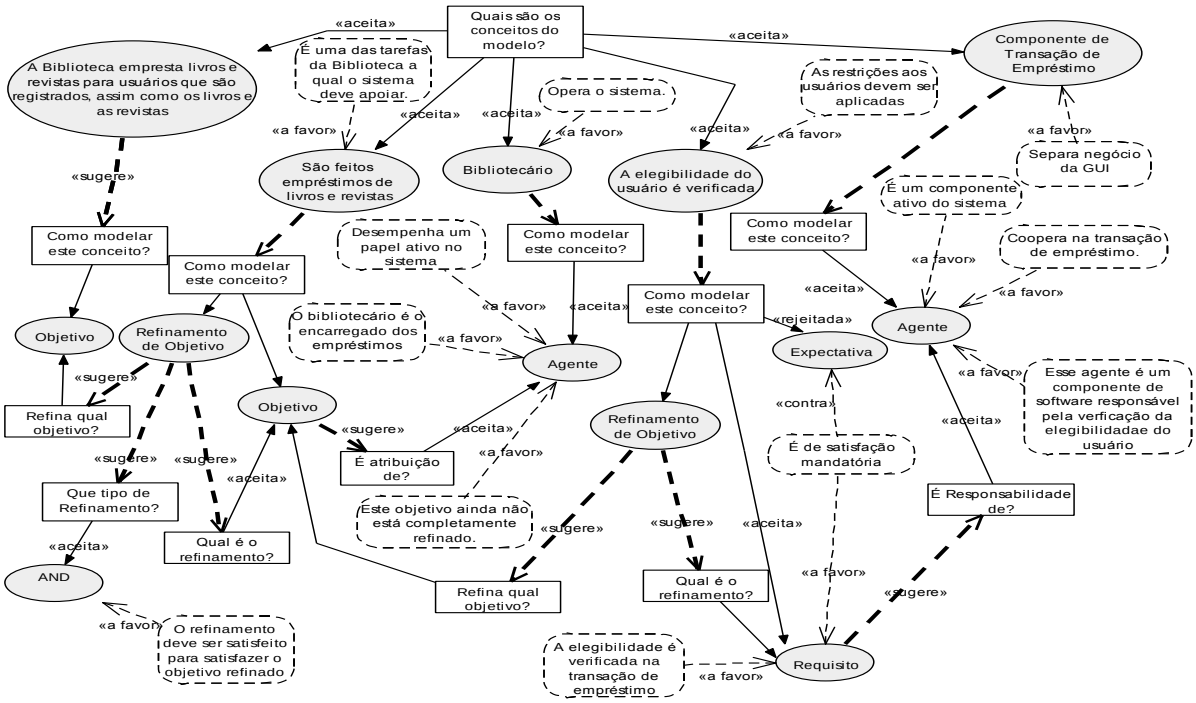

Figura 6-10. Representação de design rationale para o diagrama parcial do modelo de agentes da Biblioteca.

O metamodelo KAOS, na versão usada nesta dissertação, não tem elementos para representar a distinção feita no *framework* sobre agentes de software e de ambiente. Os agentes de software responsabilizam-se por requisitos, uma vez que sua satisfação é mandatória. Já agentes de ambiente são responsáveis por expectativas, pelo fato de que sua satisfação é facultativa. Essas regras são aplicadas através de argumentos na representação de *design rationale* da Figura 6-10.

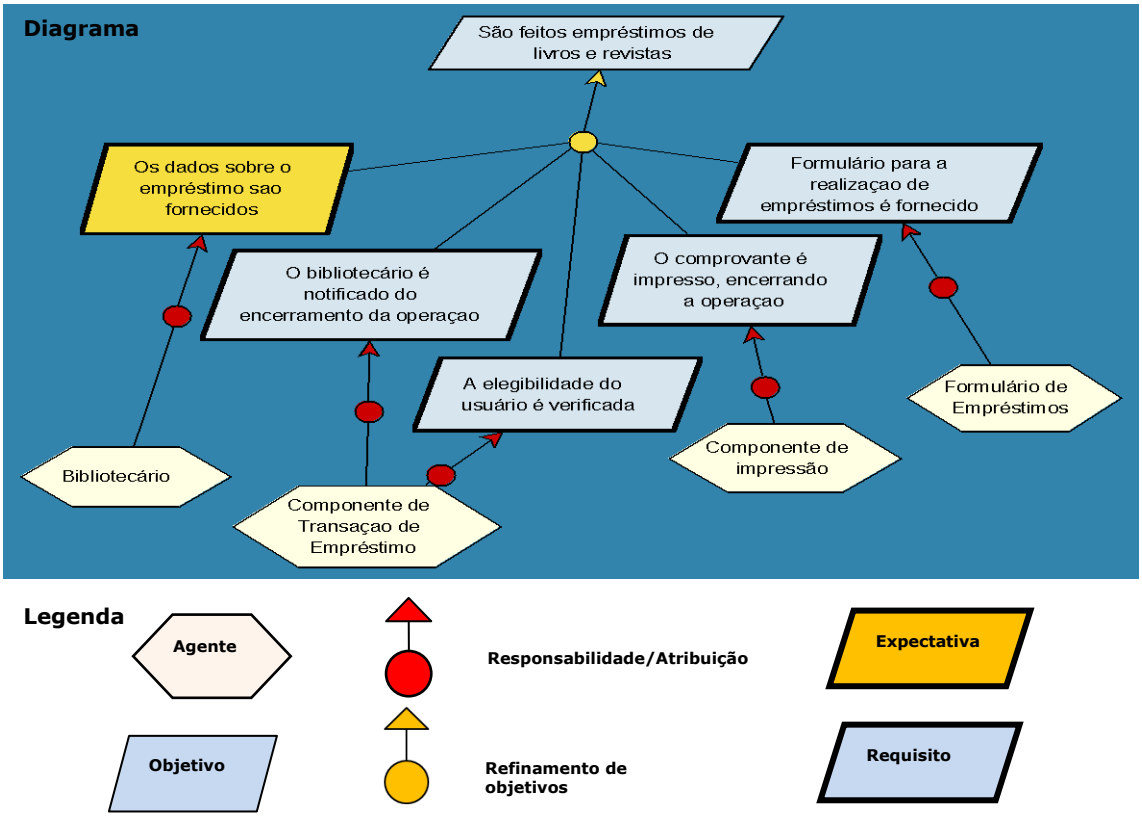

Figura 6-11. Diagrama parcial do modelo de agentes da Biblioteca.

Apesar de não ser a forma usual de apresentação de um modelo de responsabilidade, esse último diagrama foi construído assim a fim de possibilitar que responsabilidades possam ser atribuídas ao refinamento de objetivos obtido no diagrama da Figura 6-5. Há apenas uma diferença entre os requisitos e expectativas desse diagrama e os da Figura 6-11. O requisito Os dados do empréstimo são armazenados no banco de dados, foi substituído pelo requisito Formulário para a realização de empréstimos é fornecido. Essa mudança foi realizada a fim de mostrar que um componente da interface gráfica do usuário é considerado um agente e o fato do formulário ser fornecido é um requisito do sistema. Isso significa uma mudança relevante, embora sutil, em relação a outras abordagens para modelagem de requisitos, como por exemplo, na descrição de requisitos através de casos de uso, onde não há semântica para representar essa situação. No modelo de classes em UML é possível adornar uma classe com o estereótipo <<boundary>> indicando que ela tem caraterísticas de fronteira do sistema. No entanto, esta não seria uma representação de um requisito.

6.2.3 Representação de *Design Rationale* para o Modelo de Operações

Operações definem o comportamento dos agentes no sentido de satisfazer aos requisitos definidos pelo processo de refinamento dos objetivos. O Modelo de Operações é como um integrador dos demais submodelos do modelo conceitual KAOS.

O metamodelo, apresentando os conceitos relacionados com operações e agentes, é mostrado na Figura 6-12. A semântica do meta-conceito de associação Performance define que operações são executadas por agentes de qualquer tipo. Estas operações têm como entradas e saídas instâncias Event (Evento) e Entity (Objeto). Operações podem ser iniciadas por eventos (Cause) ou produzi-los para iniciar outras operações. O meta-conceito Streghtening é parte de uma composição no metamodelo, sendo tratado como inserido na meta-classe de associação Operationalization (Operacionalização). Essa meta-classe, sendo uma associação é representada pela questão Operacionaliza?, que é sugerida por todas as idéias de operação e tem como terminação aceita sempre um requisito. Apenas requisitos são operacionalizados.

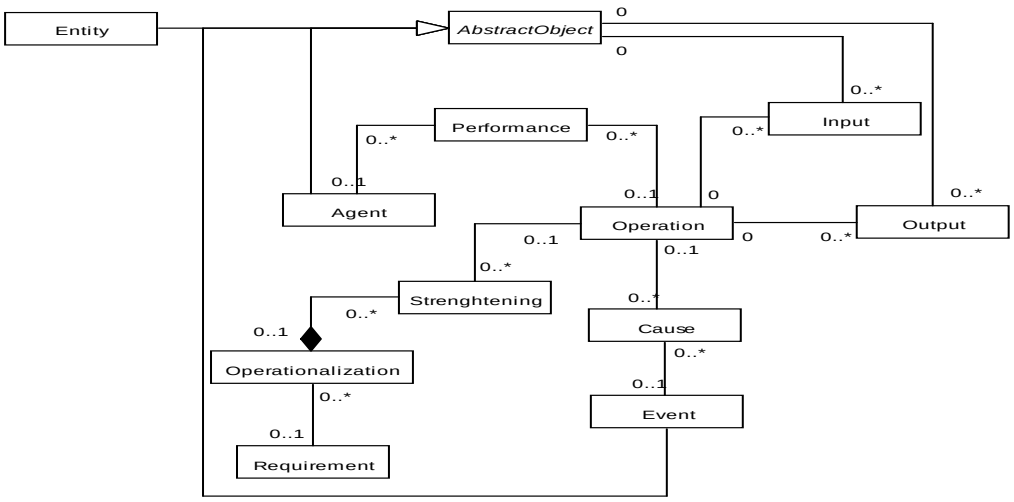

Figura 6-12. Conceitos do metamodelo KAOS para modelos de operações.

A representação de *design rationale* para o diagrama do modelo parcial de operações é mostrado na Figura 6-13. Ela refere-se ao cenário parcial de empréstimo de livros e revistas pela Biblioteca da Figura 6-14 que mostra o requisito (A elegibilidade do usuário é verificada) e a expectativa (Os dados sobre empréstimos são fornecidos). Esse cenário é também mostrado através da visão de agentes e suas responsabilidades, na Figura 6-11.

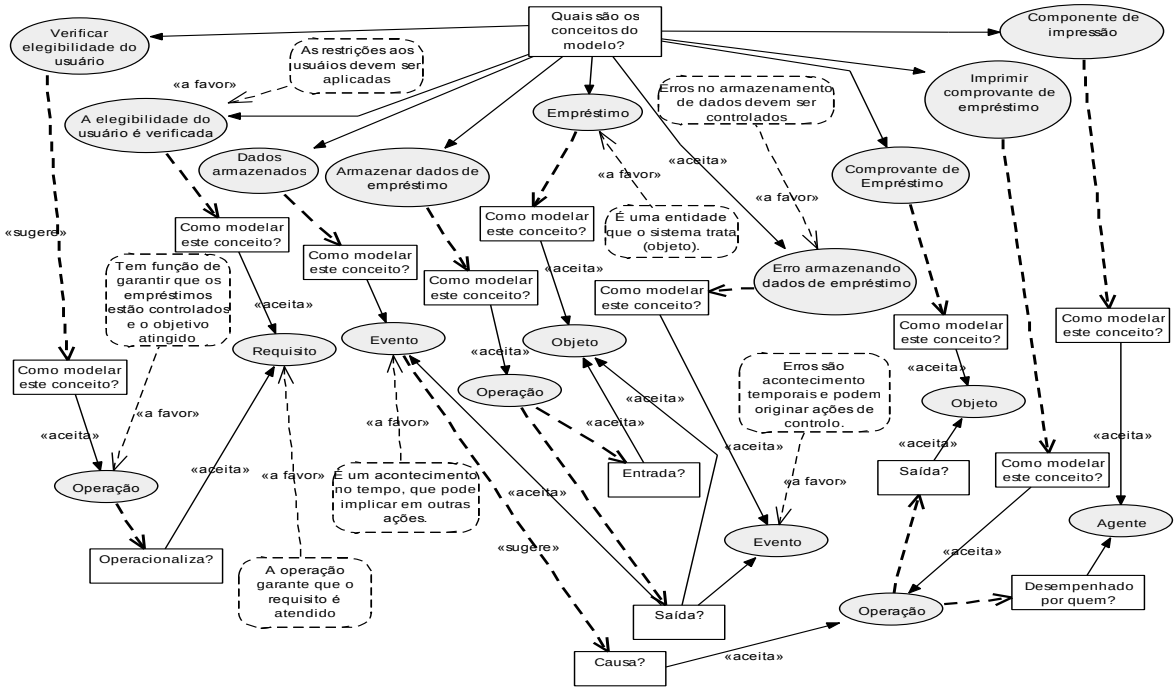

Figura 6-13. Representação de design rationale para o diagrama parcial de operações do sistema da Biblioteca.

Agentes executam operações (associação Performace) que tornam possível satisfazer requisitos associados a propriedades dinâmicas do sistema, conforme a semântica do metamodelo. O conceito que traduz esse significado é a associação chamada de Operationalization (Figura 6-12). Isso é mostrado na representação de *design rationale* ilustrada na parte inferior direita da Figura 6-13, pela instância de questão Desempenhado por quem?, sugerida pela idéia Operação. Na ilustração ainda é mostrado como objetos podem ser criados nas operações, como é o caso de Comprovante de Empréstimo, ou têm seu estado alterado por ela, como é o caso de Empréstimo, além de uma série de eventos que ocorrem na transação de empréstimos de livros e revistas na Biblioteca.

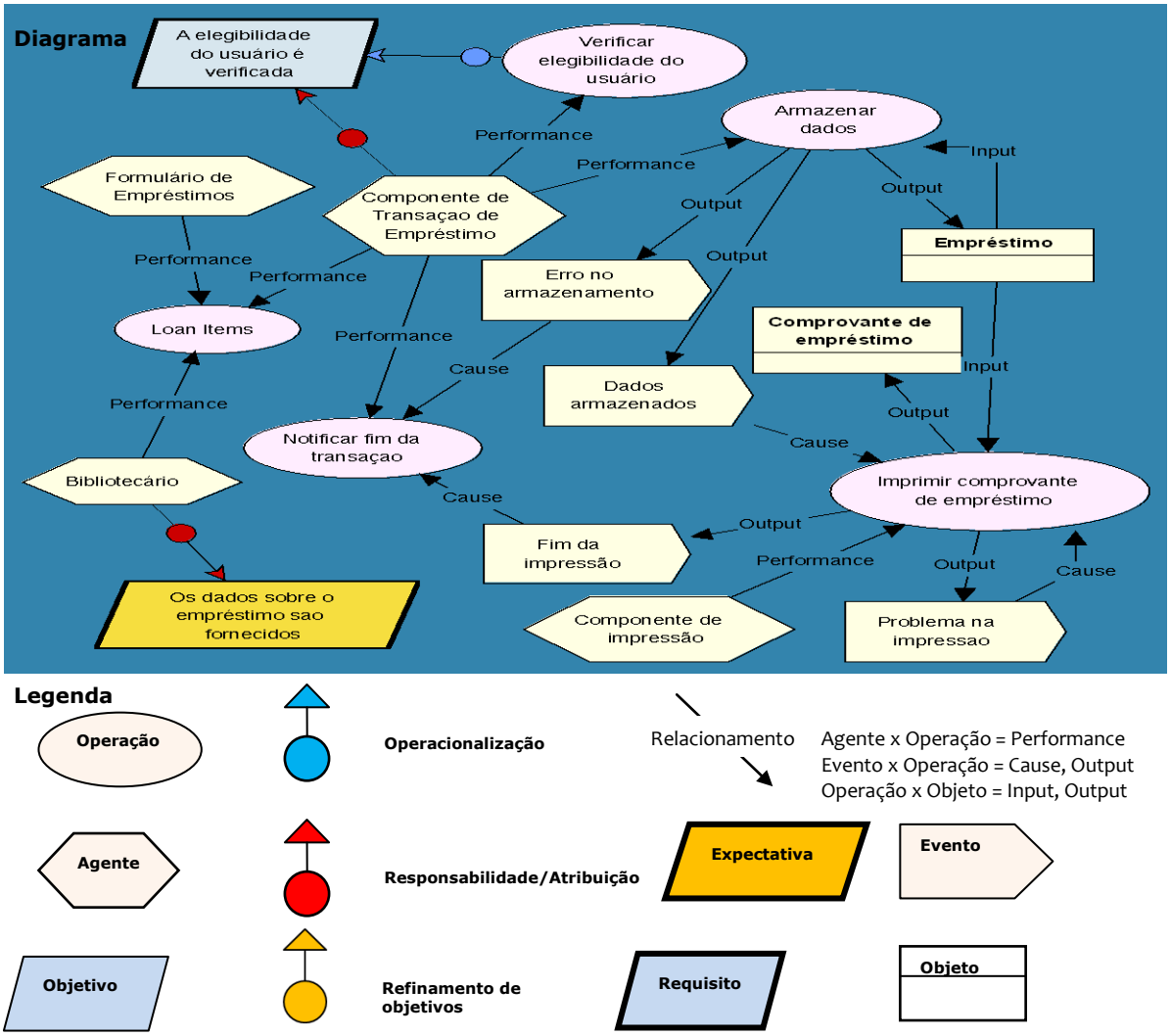

Figura 6-14. Diagrama do modelo parcial de operações da Biblioteca para o cenário de emprestimo de livros e revistas.

6.3 DESIGN RATIONALE PARA O DOMÍNIO DE SUBMISSÃO E REVISÃO DE ARTIGOS EM CONFERÊNCIAS

A Tabela 6-2 apresenta uma declaração parcial de necessidades extraídas de uma documentação referente à especificação dos requisitos do sistema da BYU (Brigham Young University), diretamente do sítio da instituição (LIDDLE, 2009). Esta declaração não está publicada em documentos científicos. No entanto, o sistema é citado no trabalho de Di Mauro (2005).

Essa lista é de natureza diferente da apresentada na seção anterior para o sistema de apoio operacional para uma Biblioteca, pelo fato de não demonstrar as características mais didáticas daquele primeiro exemplo. Nela, as necessidades funcionais e não funcionais podem estar misturadas em uma única declaração como, por exemplo, em vários itens que citam o

fato do trabalho ser feito via Web. Da mesma maneira que no domínio da Biblioteca, foram escolhidos itens significativos da lista para esse trabalho. Nesse caso, os itens um e dois.

Tabela 6-2. Declaração parcial de necessidades para o Sistema de Submissão e Revisão de Trabalhos Científicos para Eventos da Brigham Young University (BYU)

1	Submissão via Web de resumos e artigos pelos autores. A submissão é um processo de 2 passos: (1) um resumo deve ser submetido e o autor deve escolher uma palavra-chave para o artigo, (2) então os autores devem submeter o artigo. Eles podem modificar a submissão até que o Presidente do Comitê do Programa (CP) desabilite esta funcionalidade depois que a data limite para a submissão de artigos foi atingida. Depois que artigos já foram aceitos, o Presidente do CP pode reabilitar a submissão e autores podem usar o mesmo mecanismo para substituir o artigo original pela versão final.
2	Submissão de revisões via Web pelos membros do CP. Depois de submeter uma revisão, um membro do CP pode consultar e alterar a revisão.
3	Os membros do CP podem fazer "download" dos artigos via Web.
4	Reuniões do CP via Web. O Presidente do CP pode habilitar membros do CP para ler revisões de outros membros na hora apropriada.
5	Rastreamento do progresso das revisões. O Presidente do CP pode consultar o status de revisões e observar discrepâncias entre avaliações. Os artigos podem ser ordenados por revisor, ID do artigo, classificação geral, ou discrepâncias das classificações.
6	Podem ser estabelecidas múltiplas categorias para os membros do comitê, se for desejado.
7	Pode-se estabelecer tópicos relacionados com a Conferência. Se for o caso, será permitido aos autores selecionar tópicos que se relacionem com seu artigo, e membros do CP podem selecionar os tópicos nos quais estejam interessados. Isso possibilita a atribuição de artigos para revisão baseada no interesse.
8	Permitir que membros do CP naveguem pelos resumos (ordenados pela melhor adaptação do melhor tópico de pesquisa ao interesse do membro do CP.
9	Comissionamento de artigos para revisão via Web.
10	Notificar autores que seu artigo foi recebido e está em revisão.
11	Notificar o comitê do programa de atribuição de revisão.
12	Notificar o comitê do programa de discrepâncias entre sua revisão e as de outros.
13	Lembrar ao comitê do programa de revisões que ainda precisam ser submetidas.

6.3.1 Representação de *Design Rationale* para o Modelo de Objetivos

O domínio da Biblioteca fornece uma lista de requisitos de conteúdo didático que permite mais facilidade na identificação dos objetivos do sistema. Esse novo exemplo é utilizado para realçar a aplicação da abordagem Kuaba usando o mesmo metamodelo do *framework* KAOS para modelar o domínio da submissão de artigos. É verificado, já no item 1, que uma necessidade pode ser descrita de maneira extensa, envolvendo muitos conceitos, mostrando como é difícil trabalhar com uma descrição em um texto livre. Isso pode ser

observado mais claramente quando os conceitos são separados na representação de *design rationale* da Figura 6-15. Para ilustração é fornecido o diagrama do modelo parcial de objetivos para o item 1, na Figura 6-16. Esse diagrama mostra o refinamento do objetivo Artigos e resumos submetidos via Web, que é um desmembramento da declaração do item 1. Ilustra também o refinamento do objetivo Resumos são submetidos, que é um dos refinamentos obtidos no processo. Esse diagrama não ilustra a missão do sistema como raiz do grafo. Foi mostrado no Capítulo 5 que isso não é necessário.

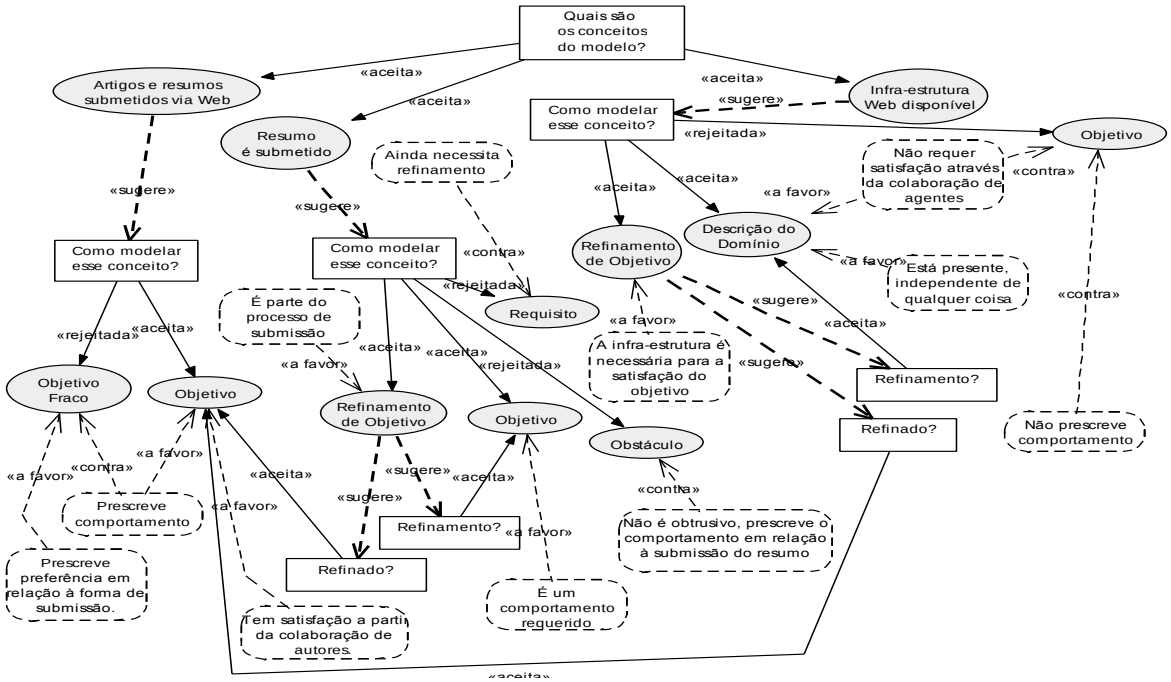

Figura 6-15. Representação de design rationale do modelo parcial de objetivos do domínio de submissão de artigos.

A descrição em linguagem natural do item 1, As submissões são feitas via Web pelos autores [...] é decomposta em vários conceitos, dentre eles Artigos e resumos submetidos via Web, que é mostrado no topo do diagrama de objetivos da Figura 6-16. A representação de *design rationale* da Figura 6-15 mostra que a questão de como modelar a idéia de domínio Artigos e resumos submetidos via Web poderia ter como solução um Objetivo Fraco (Softgoal), uma vez que a submissão via Web pode ser interpretada como uma prescrição de preferências. No entanto, os argumentos mostram que esse conceito está relacionado com comportamento e deve ser modelado como objetivo.

Esse objetivo, prescrevendo que a submissão deve ser via Web, introduz também o conceito de DomainInvariant, mostrado na Figura 6-1 do metamodelo parcial de KAOS, para a modelagem de objetivos. Esse conceito é novo em relação aos exemplos apresentados até agora, defindo uma condição de invariabilidade no domínio. Para que as submissões sejam

feitas via Web, essa infra-estrutura deve estar presente, indepedente do software em análise, como mostram os argumentos na representação de *design rationale* da Figura 6-15.

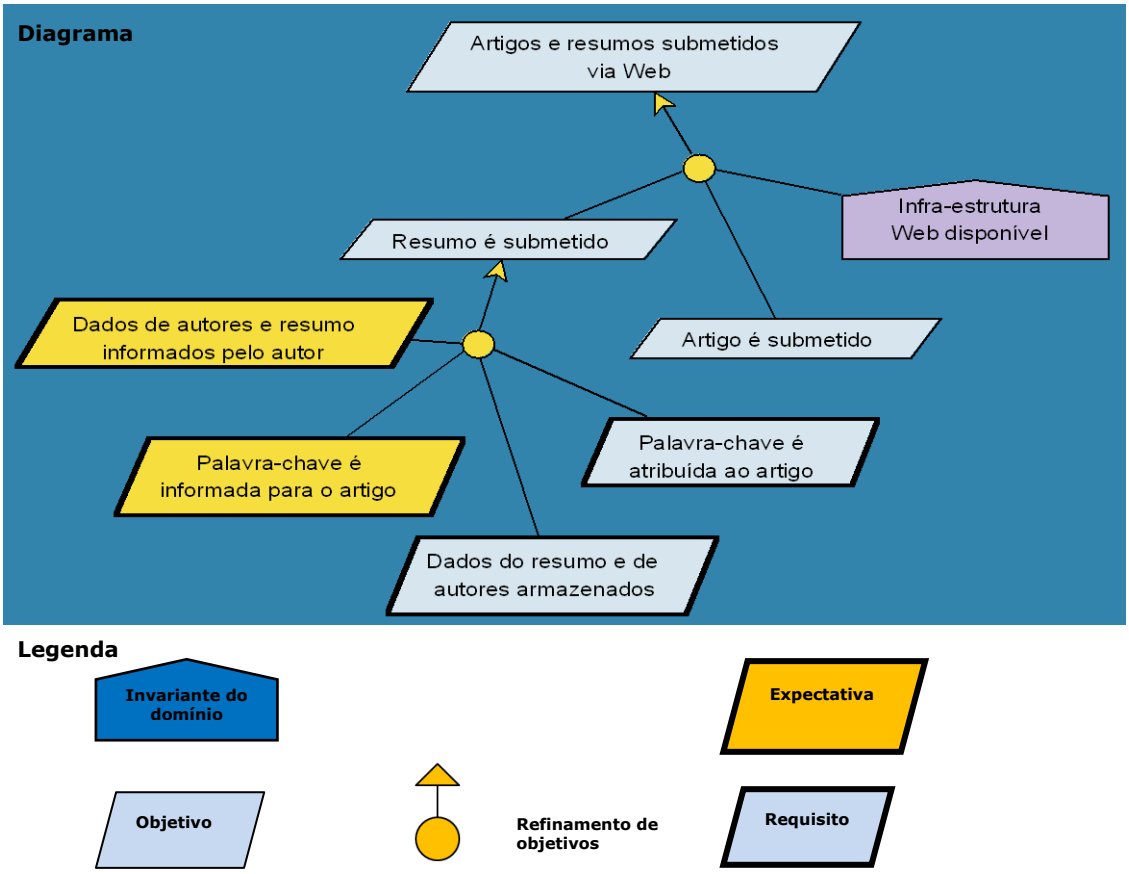

Figura 6-16. Diagrama do modelo parcial de objetivos para o domínio de submissão de artigos.

6.4 DISCUSSÃO SOBRE OS MODELOS E REPRESENTAÇÕES DE DESIGN RATIONALE

Lee (1997), assim como Burge e Brown (2002), ressalta que *design rationale* pode ser usado de inúmeras maneiras em benefício da qualidade do software em desenvolvimento. Esses autores apontam que os benefícios podem ser obtidos tanto na verificação da satisfação dos requisitos e da intenção do projetista, como na avaliação de designs e escolhas buscando inconsistências. No que diz respeito à manutenção de software, *design rationale* é útil na identificação de fontes de problemas no design com o propósito de indicar onde as modificações são necessárias e também de assegurar que as alternativas de solução rejeitadas não serão inadvertidamente usadas no design modificado. A documentação também é beneficiada, uma vez que, além das decisões finais, informações justificando as decisões

tomadas são registradas. D*esign rationale* também traz benefícios para a qualidade do software a ser desenvolvido ao fornecer apoio para as discussões e para verificações do impacto de mudanças e de consistência. Além disso, pode fornecer ajuda no abrandamento de conflitos ao observar as violações de restrições entre os múltiplos envolvidos no projeto (*stakeholders*), sejam eles os projetistas, especialistas de processo de negócios, ou qualquer outro envolvido. Raciocínios representados explicitamente podem fornecer um vocabulário comum e manter a memória dos projetos, facilitando as negociações e o consenso (LEE, 1997; BURGE; BROWN, 2002; FIGUEIREDO, 2006).

Os testes de modelagem e de representação de *design rationale* feitos nesta dissertação revelaram aspectos que corroboram algumas das observações anteriores. Embora a hipótese da pesquisa de que *design rationale* pode contribuir para as tarefas da Engenharia de Requisitos seja específica para esta atividade, não se atendo ao processo de desenvolvimento de software como um todo, foi observado que há indicações de que pode haver melhoria de qualidade nos modelos de requisitos e que há possibilidade de rastreamento de mudanças, favorecendo a análise de impacto através das representações de *design rationale*. Essas observações são abordadas a seguir.

6.4.1 Apoio à Melhoria da Qualidade

A Figura 6-17 mostra a evolução da representação de *design rationale* do modelo de objetivos refinados do sistema da Biblioteca apresentado nos exemplos da Figura 6-6 e da Figura 6-5. Essa evolução foi feita a partir da observação da argumentação registrada no *design rationale* na qual foi identificado um defeito, ou a falta de completude e de correção no modelo, de acordo com a prática recomendada pela IEEE para especificações de requisitos de software (IEEE, 1998, p. 4-5).

Esse guia preconiza que uma especificação de requisitos é correta se, e somente se, toda necessidade declarada seja um requisito que o produto de software deve atender, embora não haja feramentas para garantir a correção de especificações. Também afirma que uma especificação de requisitos é completa se, e somente se algumas condições são satisfeitas, dentre elas a que declara que todas as necessidades importantes relativas à funcionalidade, restrições de design, atributos e interfaces externas devem ser endereçados e tratados na especificação (IEEE, 1998). Esta seção mostra que representação de *design rationale* do modelo de requisitos pode contribuir de maneira relevante para melhorias nesses aspectos da qualidade da espcificação apoiando na identificação dos defeitos.

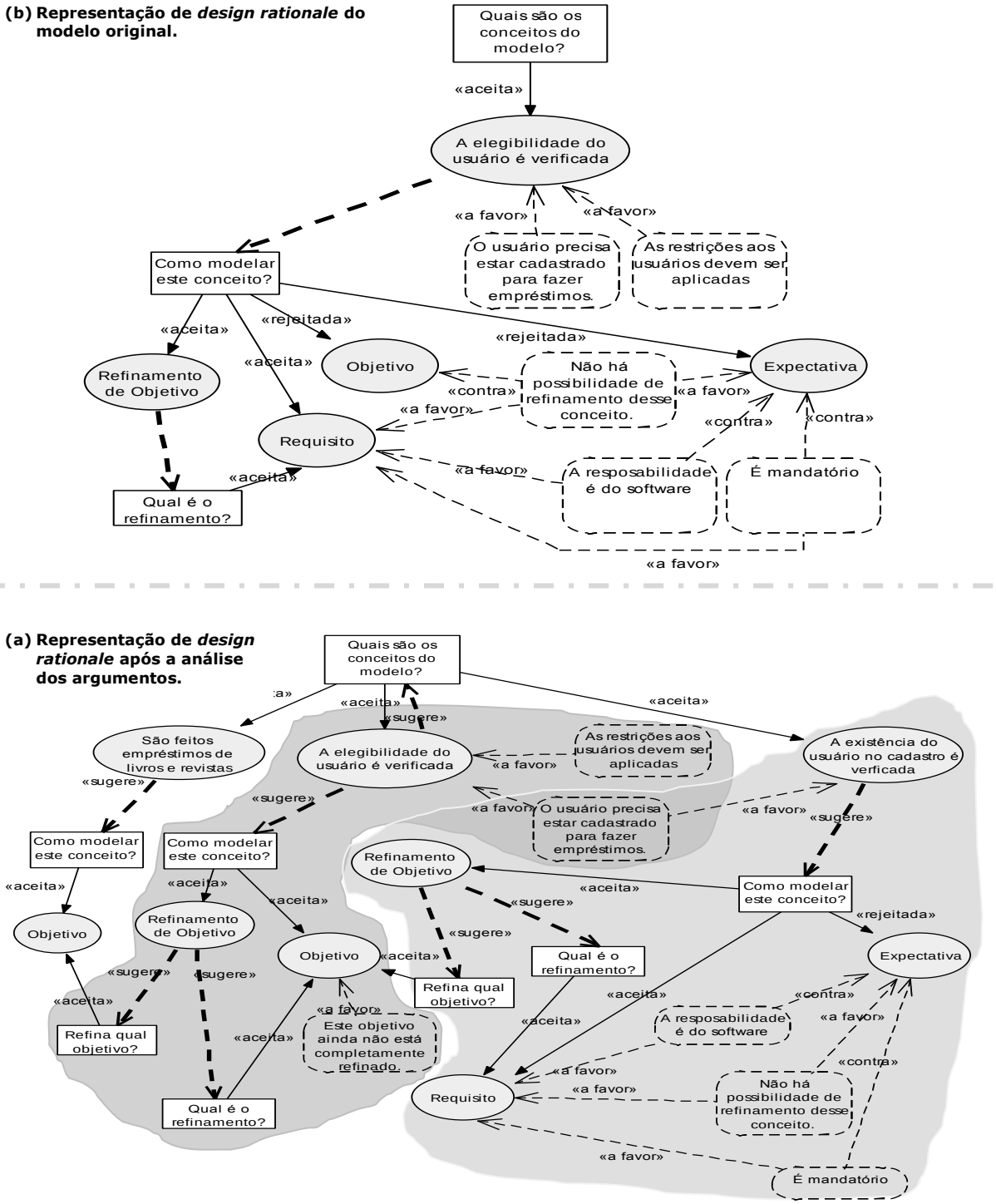

Figura 6-17. Exemplo da melhoria da qualidade: (a) Original; (b) Revisão baseada nos argumentos e consequente alteração na representação de *design rationale*.

A Figura 6-17-a ilustra a representação de *design rationale* do modelo parcial original, que tem a idéia de Requisito aceita como solução para a questão de como modelar a idéia de domínio A elegibilidade do usuário é verificada. A aceitação se dá em função da assunção de que o argumento Não há possibilidade de refinamento deste conceito é aplicável neste caso.

Entretando, uma análise mais cuidadosa dos argumentos que apóiam a aceitação dessa idéia de domínio, levando também em consideração as declarações sobre o sistema da biblioteca da Tabela 6-1, mostra que essa idéia de domínio foi escolhida pelo engenheiro de software considerando a declaração que os usuários precisam estar cadastrados no sistema para fazer empréstimos de livros e revistas. A idéia escolhida como solução é mais genérica do que a declarada na Tabela 6-1, podendo indicar uma intenção do engenheiro de software de deixar a possibilidade de outras regras de elegibilidade serem introduzidas com pequeno impacto, o que será confirmado quando for introduzida a nova regra de elegibilidade, que será abordada na Seção 6.4.2. No entanto, a declaração feita é explícita e não foi inserida no modelo de requisitos, sendo a idéia mais geral aceita como requisito. O correto seria existir apenas a idéia de domínio O usuário precisa estar cadastrado para fazer empréstimos, e essa idéia ser aceita como requisito (idéia de design). Outra solução seria manter a idéia de domínio A elegibilidade do usuário é verificada, mais geral, mas não aceitar a idéia de requisito como solução de design. Desta forma surgiriam as idéias de design de refinamento de objetivo e objetivo para essa idéia de domínio. A declaração específica sobre o usuário estar cadastrado teria como idéia de design aceita a idéia requisito, refinando a idéia de design objetivo aceita para a ideía de domínio original. Essa última solução é ilustrada na Figura 6-17-b.

A Figura 6-17-b mostra a revisão feita na representação de *design rationale*.A área de sombra mais escura ressalta a revisão da solução original e a área mais clara ilustra a representação de *design rationale* para a nova idéia resgatada das declarações da Tabela 6-1. Essa representação de *design rationale*define a evolução do modelo parcial de objetivos mostrado na Figura 6-18. Nesse diagrama a declaração A elegibilidade do usuário é verificada passa a ser modelada como objetivo (com as linhas do paralelogramo mais fracas) e o requisito A existência do usuário no cadastro é verificada é adicionado ao modelo, como refinamento do objetivo anterior.

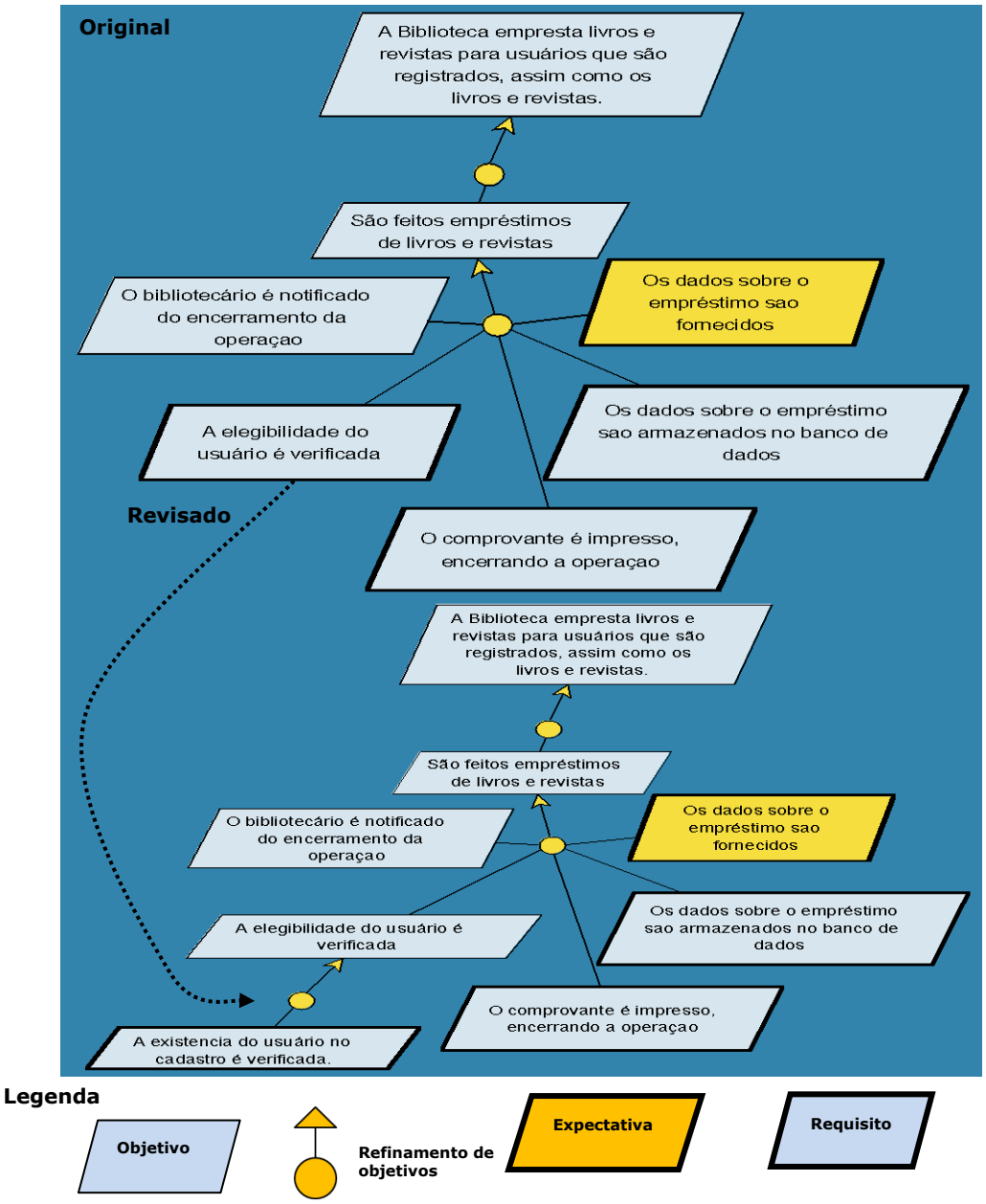

Figura 6-18. Revisão no modelo provocada pela representação de *design rationale*.

6.4.2 Apoio ao Gerenciamento da Evolução de Requisitos e Análise de Impacto

O processo da Engenharia de Requisitos usualmente envolve o conhecimento do domínio, a descoberta de requisitos (descoberta é usada como a tradução de *elicitation*), sua avaliação, especificação e documentação. Abrange também o controle de qualidade do modelo de requisitos e o gerenciamento da evolução desses requisitos, ou suas mudanças ao longo do ciclo de vida de desenvolvimento do sistema. Embora esses aspectos sejam abordados com ênfase variada nos trabalhos de vários autores, como Pohl (1996), Kotonya e

Sommerville (1997), Sommerville (2006), Pressman (2009) e Lamsweerde (2009), há certa unanimidade em relação ao papel crucial da Engenharia de Requisitos no desenvolvimento de software. É também aceito que requisitos são voláteis e o impacto de suas mudanças pode ser muito grande, dependendo da fase em que o projeto se encontra.

Na seção anterior foi exemplificada uma possível evolução no modelo de objetivos do sistema da Biblioteca provocada pela análise dos argumentos registrados na representação de *design rationale*. Essa evolução é devido à necessidade de ajuste no modelo buscando a melhoria de qualidade quanto à sua correção e competude com respeito às declarações sobre o sistema. Nesta seção, além do ajuste, é introduzida uma nova declaração de necessidade sobre esse sistema. Essa declaração refere-se ao fato de que os envolvidos no desenvolvimento desejam que o sistema não permita que usuários em débito de devolução possam fazer novos empréstimos. Essas evoluções são apresentadas nas figuras Figura 6-19, 6-20 e 6-21. Elas ilustram as representações de *design rationale* para modelos de objetivos e operações, assim como a evolução do modelo de objetivos.

A representação de *design rationale* para a evolução do modelo de objetivos é mostrada na Figura 6-19. A Figura 6-19-a mostra a representação de *design rationale* do modelo de objetivos refinados original mostrada na Figura 6-6, com o argumento que contempla a declaração de que o usuário deve estar cadastrado para realizar empréstimos. A análise desse argumento leva a uma revisão na representação e consequente evolução no modelo de objetivos como mostrado na Figura 6-18. Esta é a primeira evolução, ou mudança, efetuada, que através da representação de *design rationale* pode ter seu rastreamento e implementação facilitados, como será exemplificado e discutido nesta seção.

Na Seção 6.2.3 foi discutido o exemplo de representação de *design rationale* para o modelo de operações. Foi mostrado que esse modelo funciona como um integrador dos demais submodelos do modelo conceitual do framework KAOS e influencia fortemente a arquitetura do software em desenvolvimento. No que diz respeito aos elementos da representação de *design rationale* e seus relacionamentos pode-se verificar que a idéia de design Requisito, antes aceita para a questão de como modelar a idéia de domínio A elegibilidade do usuário é verificada, é alterada para a idéia Objetivo, uma vez que essa última passa a ser refinada pela nova idéia de domínio introduzida pela análise dos argumentos (área sombreada no centro da Figura 6-19-b). A semântica do metamodelo obriga que a argumentação seja revisada a fim de que as novas decisões passem a ser consistentes.

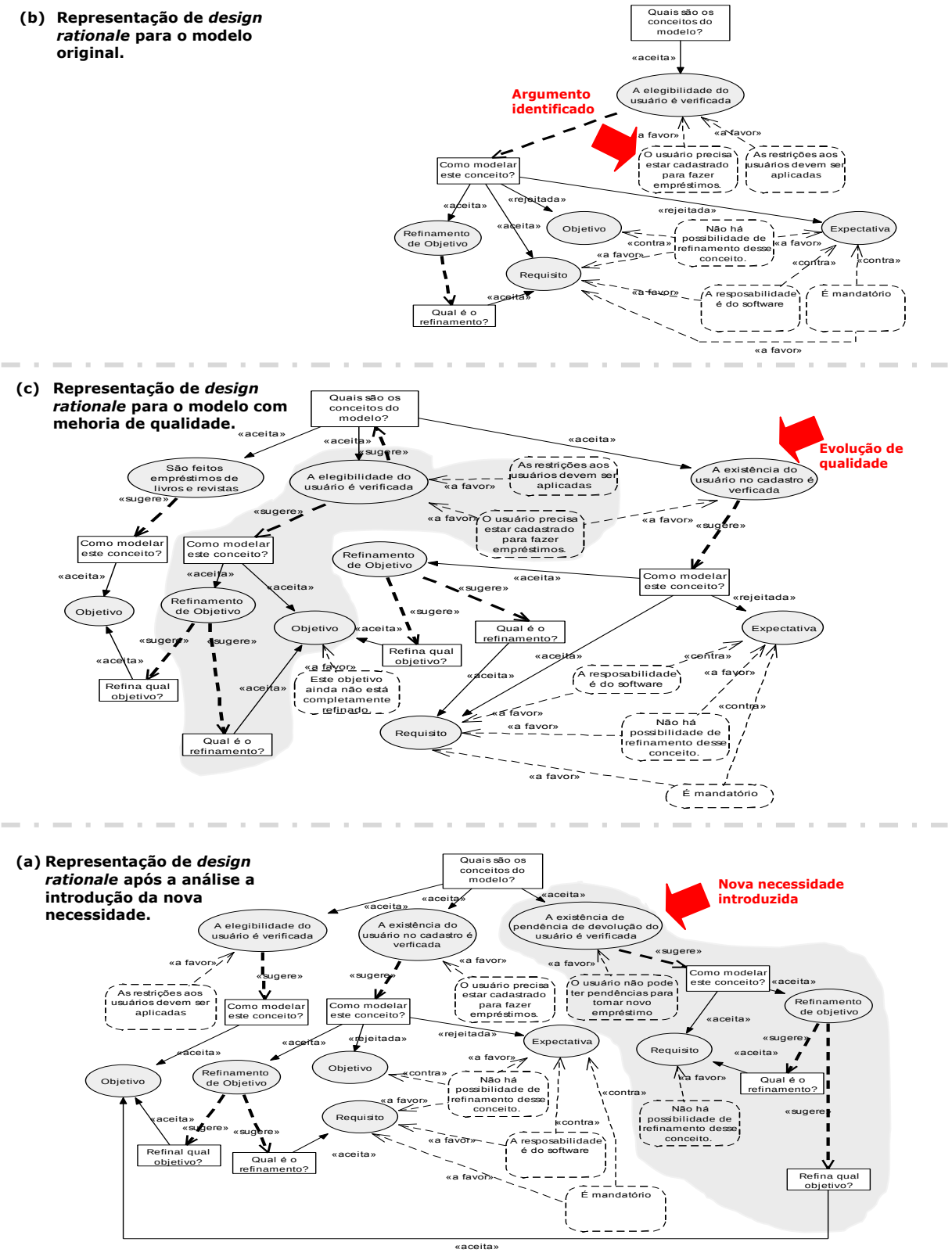

Figura 6-19. Exemplo do rastreamento: (a) Original; (b) Revisão após introdução da melhoria de qualidade (c) Revisão após a introdução de uma nova necessidade e consequente alteração na representação de *design rationale*.

A Figura 6-19-c ilustra a representação de *design rationale* para o modelo de objetivos refinados com a revisão das necessidades do sistema da Biblioteca (área sombreada à diretita

do diagrama). Essa revisão adiciona uma nova necessidade sobre as regras de elegibilidade para empréstimos por parte dos usuários. A regra A existência de pendência de devolução do usuário é verificada é uma nova idéia de domínio adicionada à representação de *design rationale* existente. A idéia de design Requisito é aceita como solução de design para a questão de como modelá-la. Isso acontece a partir da análise dos argumentos e da observação que trata-se de mais um refinamento, do tipo AND, da idéia A elegibilidade do usuário é verificada. O impacto no modelo de objetivos é mostrado na Figura 6-20.

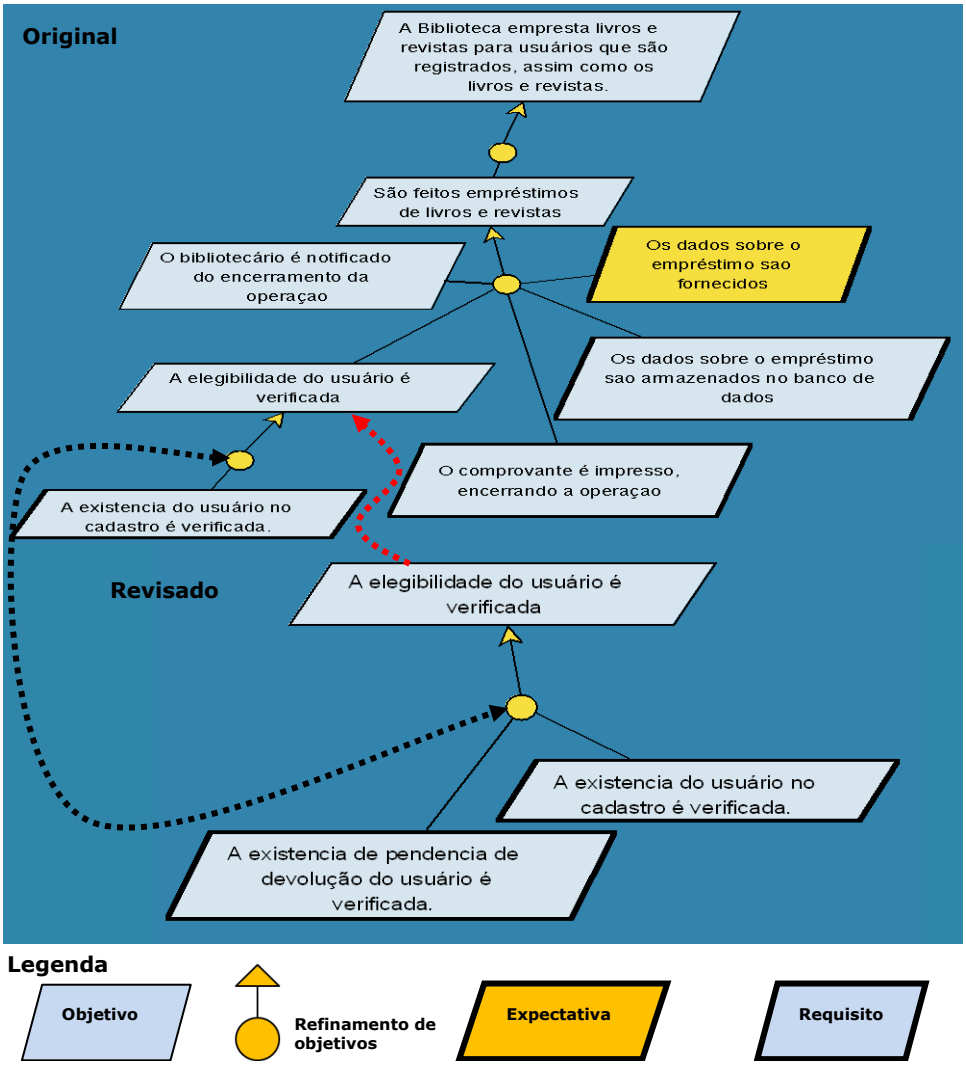

Figura 6-20. Revisão no modelo provocada pela representação de design rationale.

A Figura 6-21 ilustra outro aspecto importante no apoio à gestão da evolução de requisitos e à análise de impacto dessa evolução. Essa ilustração mostra como a melhoria de qualidade e a introdução de uma nova necessidade alteram a representação de *design*

rationale para o modelo de operações, apresentada de forma parcial a fim de facilitar a visualização dessas alterações.

As alterações na representação de *design rationale* do modelo de objetivos impactam diretamente a representação do modelo de operações, através do rastreamento da idéia de domínio A elegibilidade do usuário é verfificada. No design modificado esta idéia não é mais "operacionalizável", uma vez que é um objetivo, e a semântica do metamodelo KAOS não permite este relacionamento. Esta idéia de domínio foi substituída por duas novas idéias: A existência do usuário no cadastro é verificada e A existência de pendências de devolução do usuário é verificada. Essas idéas têm como solução de design aceita as idéias de Requisito. A partir da análise da representação de *design rationale* original, o engenheiro de software opta por criar duas novas idéias de domínio que têm solução no modelo pela idéia Operação. Essa opção é feita pelo fato do engenheiro de software respeitar o princípio que uma operação deve possuir apenas uma responsabilidade (Single Resposibility Principle) (MARTIN, 2008). Estas alterações são mostradas na Figura 6-21-b. A área sombreada dá lugar à novos elementos da instância da ontologia Kuaba consistentes com os conceitos de domínio e com a semântica do metamodelo KAOS.

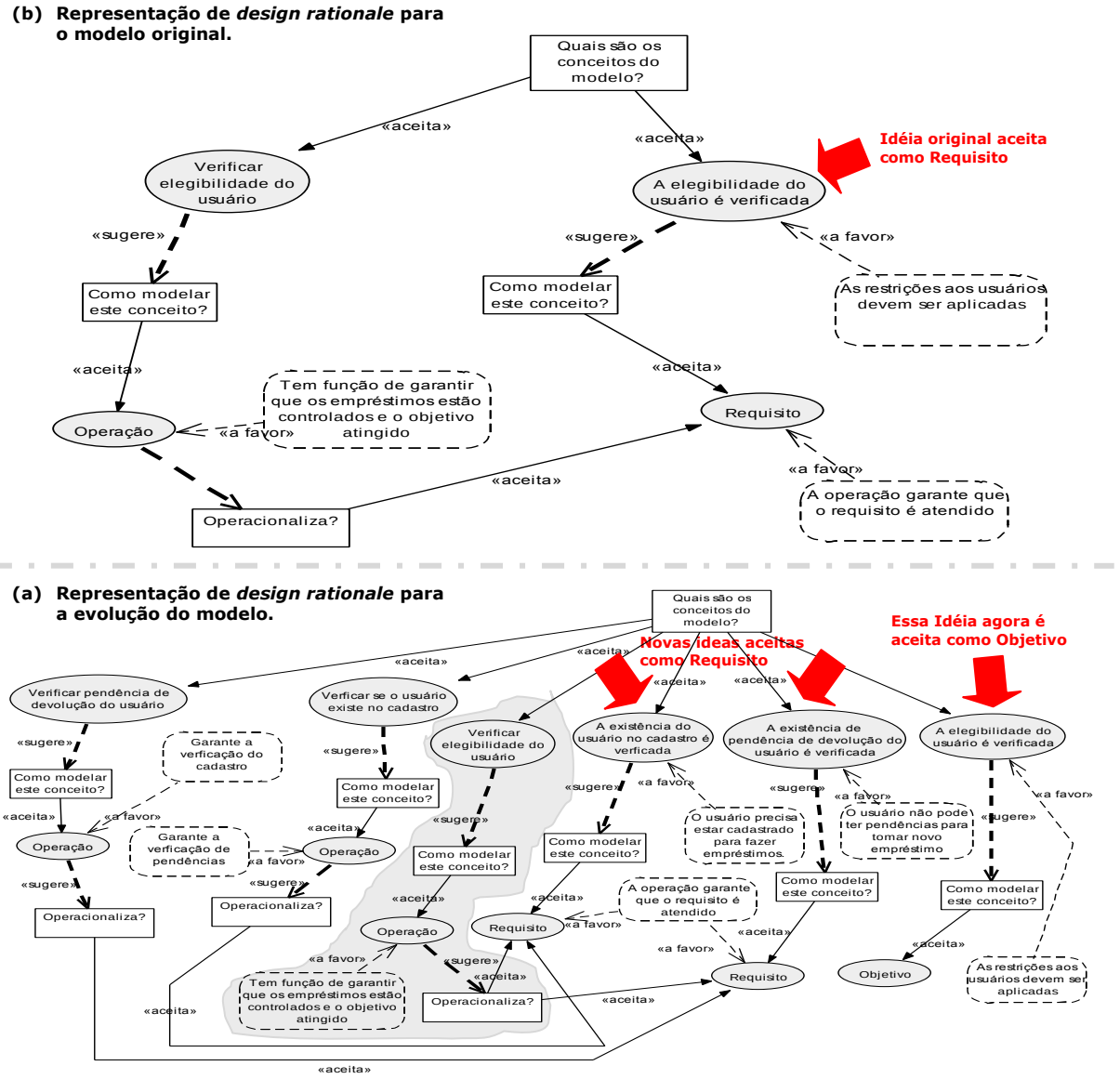

Figura 6-21. Representação de design rationale para a evolução do modelo de operações: (a) Original; (b) Aplicadas as mudanças de qualidade e nova necessidade.

7 RELACIONAMENTOS ENTRE DESIGN RATIONALES

Os modelos construídos e as representações de *design rationale* geradas durante a etapa de testes realizada nesta pesquisa foram fontes importantes de informação sobre potencialidades da utilização da abordagem Kuaba para representação de *design rationale* na Engenharia de Requisitos. Embora tenha sido possível representar, com base no vocabulário da ontologia Kuaba, a totalidade dos conceitos explícitos no metamodelo KAOS, incorporando completamente sua semântica, não foi possível representar o raciocínio empregado pelos engenheiros de software durante a análise das possíveis relações entre esses modelos.

Essa análise ocorre quando se está modelando certo grupo de artefatos em determinada atividade do desenvolvimento, e sabe-se que em outro modelo de outra atividade haverá um artefato cujas razões que levam ao design escolhido estão associadas com este grupo de artefatos sendo modelado, como por exemplo, quando se está modelando uma classe de projeto que está relacionada com uma classe modelada na atividade de modelagem conceitual. Isto se dá em função da experiência e *insight* do engenheiro de software, assim como do conhecimento que ele tem do processo e do método de design que está utilizando. Embora as decisões sejam tomadas a respeito do design dos artefatos (ou modelos) produzidos na mesma atividade de design, ou em atividades de design diferentes, a relação entre eles ou entre essas decisões, é definida pelo engenheiro de software.

O mesmo também pode ocorrer no trabalho colaborativo, em que outro membro da equipe é responsável por produzir determinado modelo em outra atividade de design. Entretanto, os relacionamentos entre os modelos, ainda assim, serão definidos pelo engenheiro, que poderá usar os recursos do metamodelo usado no design, se estiverem disponíveis, para criar essas associações e apoiar com mais detalhe semântico suas decisões. Assim, é necessário representar também os possíveis relacionamentos entre os *design*

rationales registrados para os diferentes modelos construídos durante o desenvolvimento, a fim de capturar esse conhecimento que os engenheiros de software possuem sobre as relações entre os modelos.

Existem situações nas quais o engenheiro de software não poderá sequer associar os elementos dos modelos, como é o caso em que o metamodelo usado no design não possui semântica para apoiar a associação entre artefatos. Pode também ser o caso de serem usados metamodelos diferentes nas diferentes atividades de design. Para a situação onde o mesmo metamodelo é usado e este fornece semântica para apoiar o relacionamento entre artefatos de modelos diferentes, não se trata de relacionamento entre *design rationale*, mas simplesmente da representação de *design rationale* para o relacionamento entre elementos do modelo.

O metamodelo do *framework* KAOS possui quatro submodelos, como ilustrado na Figura 5-2, no Capítulo 5. Esse metamodelo fornece a semântica adequada para apoiar as relações entre conceitos dos submodelos. No exemplo da Figura 7-1 é ilustrada essa relação entre os modelos usando um refinamento do modelo de objetivos (paralelogramos) e suas relações (setas direcionadas de ponta cheia e círculos escuros sobre a linha) com agentes (representados por hexágonos) no modelo de responsabilidade.

A semântica das relações entre agentes e objetivos, requisitos ou expectativas está representada no *design rationale* do modelo parcial de agentes da Figura 7-1, ilustrado na Figura 7-2. O agente Bibliotecário, que no modelo parcial tem uma relação de responsabilidade com uma expectativa (Os dados de empréstimo são fornecidos), pode ter também uma relação de atribuição com o objetivo São feitos empréstimos de livros e revistas, identificada em uma etapa anterior do design onde outros refinamentos ainda não haviam sido modelados. Isso foi mostrado na representação de *design rationale* com o objetivo de enriquecer a ilustração e mostrar a capacidade de representação da abordagem Kuaba.

Após o refinamento do objetivo São feitos empréstimos de livros e revistas, o requisito A elegibilidade do usuário é verificada também é obtido e a responsabilidade por satisfazê-lo é do agente de software Componente de Transação de Empréstimo. Neste ponto pode-se notar que características arquitetônicas do software em desenvolvimento já estão emergindo e são registradas no *design rationale*. Este é um aspecto importante do metamodelo estudado.

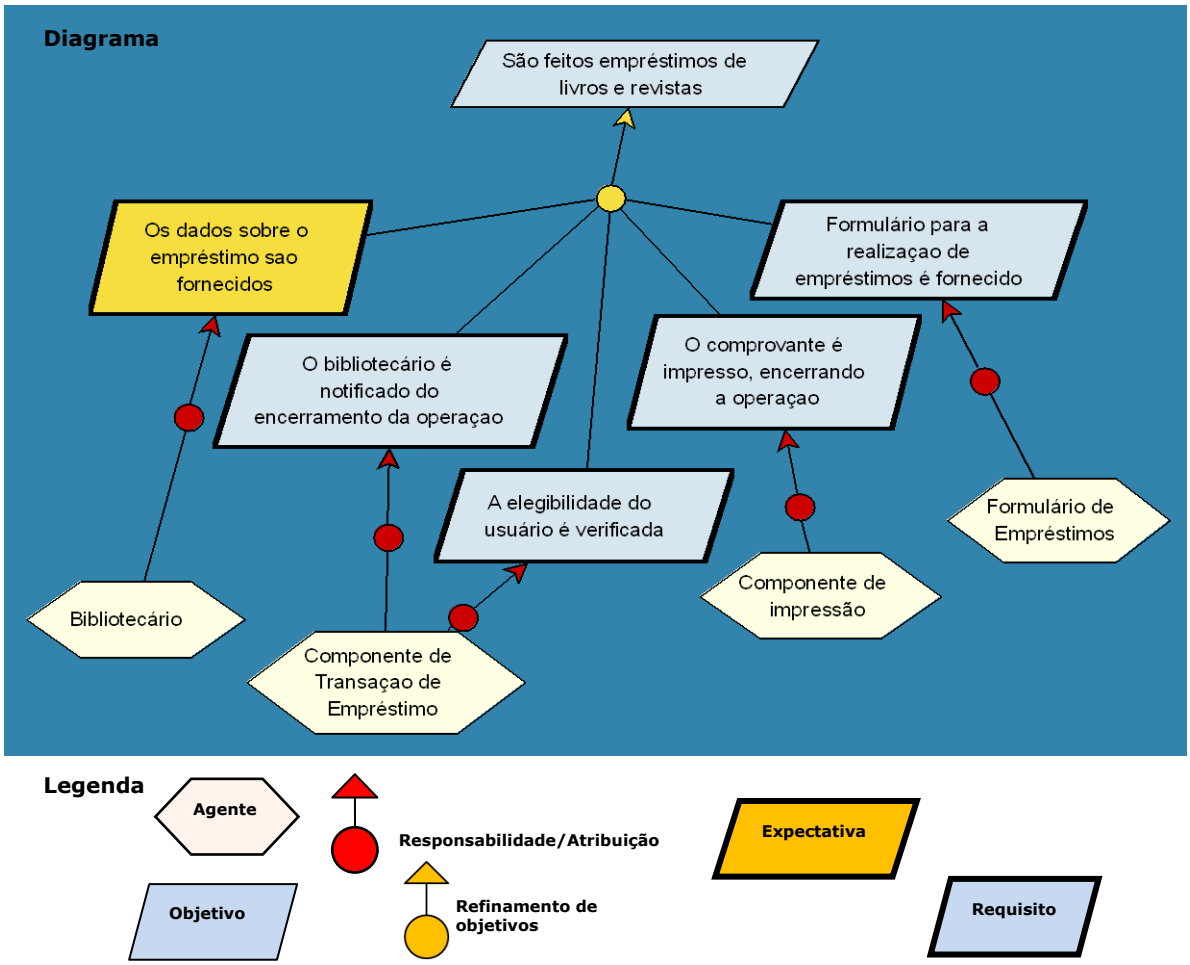

Figura 7-1. Diagrama do modelo parcial de agentes.

Na Figura 7-2, pode ser observado que as relações existentes entre elementos dos modelos de objetivos e de resposabilidades são registradas no *design rationale*, pois a semântica do relacionamento entre os elementos desses dois modelos, fornecida pelo metamodelo KAOS, é incorporada à sua representação (instância da ontologia Kuaba). Outro metamodelo que fornece semântica para representar as associações entre modelos diferentes e, consequentemente, representar através da abordagem Kuaba o *design rationale* desses relacionamentos é o metamodelo da UML (OMG, 2001, 2006a, 2006b). Nesse caso, o elemento do metamodelo que possibilita a representação é a Dependência.

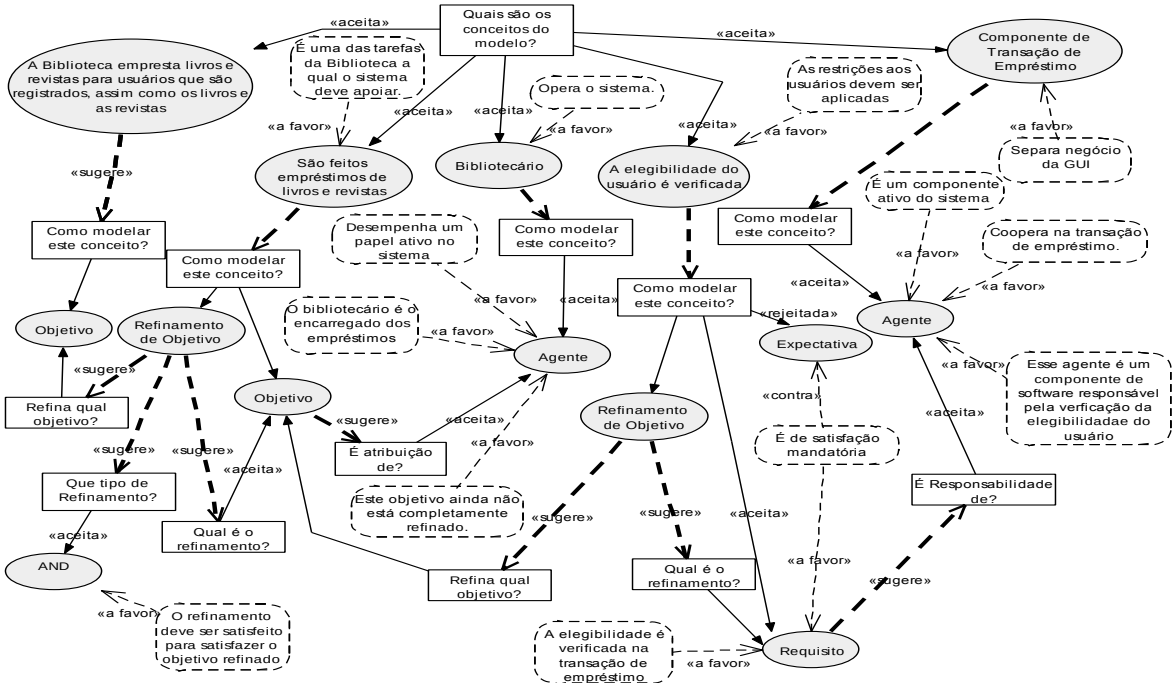

Figura 7-2. Representação de design rationale ilustrando as relações entre elementos dos modelos de objetivos e de responsabilidades.

Na Figura 7-3 é ilustrado o design de dois modelos e suas representações de *design rationale* para um cenário de registro de pedidos. Nessa situação, elementos de dois modelos produzidos em atividades diferentes são relacionados através de uma associação de Dependência, fornecida na semântica do metamodelo da UML.

No metamodelo da UML a associação de dependência pode ser decomposta em alguns outros classificadores buscando dar mais significado a esse tipo de associação. Esse é o caso da representação da Realização, ilustrada por uma linha tracejada com uma seta de ponta fechada. Essa representação significa que os componentes modelados realizam a funcionalidade descrita no caso de uso na arquitetura de implementação do software exemplificado na Figura 7-3.

A representação de relacionamentos entre *designs rationales* pode contribuir para o rastreamento dos modelos produzidos a partir do mesmo metamodelo, pois oferece uma semântica comum para a representação. Isso não acontece quando os modelos são instanciados de metamodelos diferentes ou a ferramenta de design não apóia tal mecanismo.

No caso do uso de metamodelos diferentes para a representação dos artefatos produzidos em diferentes atividades de design, a semântica das possíveis relações entre os elementos dos modelos não está definida, pois não há equivalência explícita entre os conceitos definidos nesses metamodelos. Portanto, a ontologia Kuaba deve suprir elementos que possam apoiar a representação dos relacionamentos entre os *designs rationales*

registrados para os diferentes modelos produzidos a partir desses metamodelos. É importante ressaltar que isso também é válido para um mesmo metamodelo que não ofereça semântica para relacionar elementos de modelos diferentes. No caso em que a semântica é suprida pelo metamodelo, trata-se, como visto anteriormente, de representar *design rationale* de relacionamentos entre elementos cuja semântica é fornecida pelo próprio metamodelo.

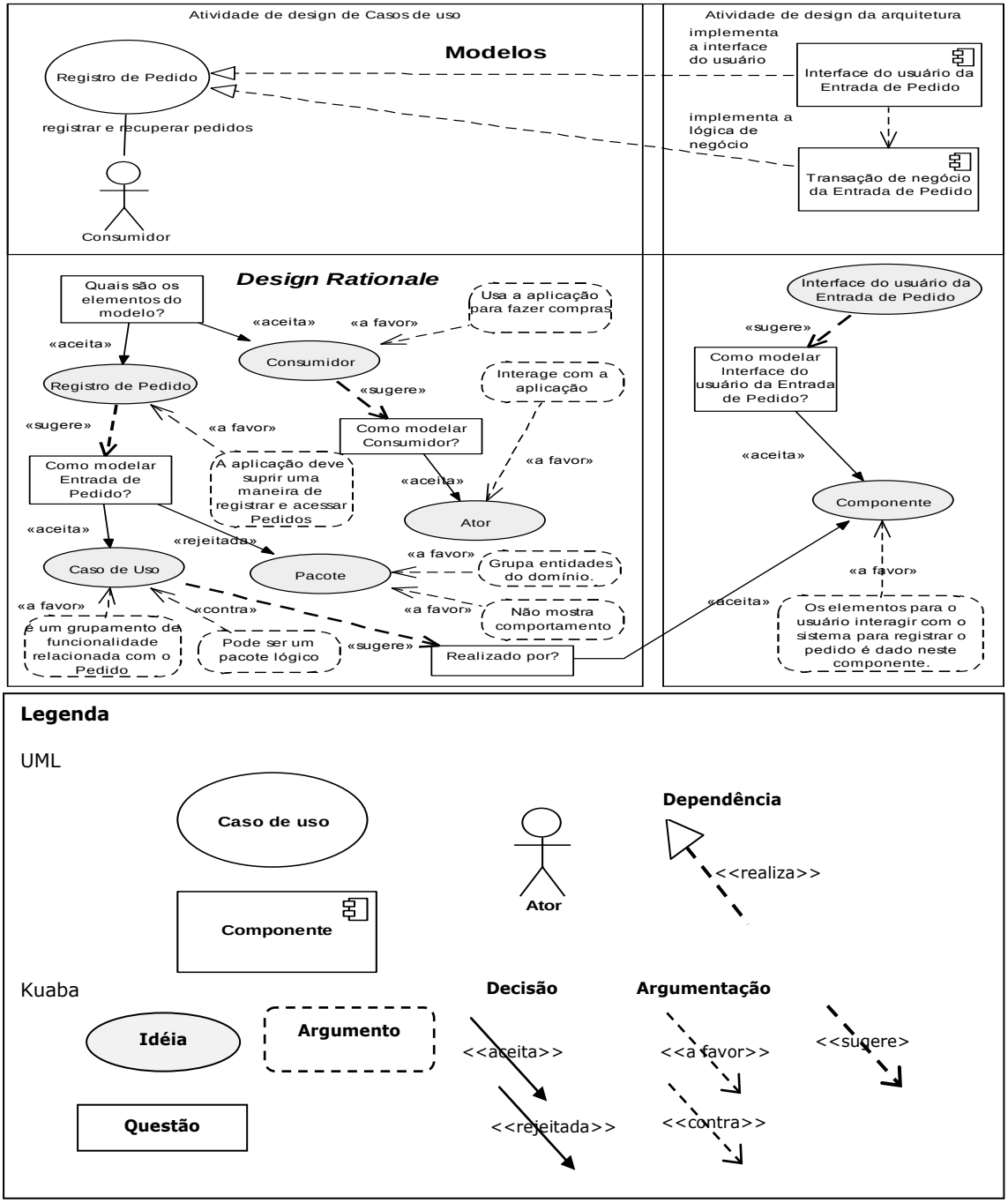

Figura 7-3. Exemplo de design rationale para a modelagem de uma relação de realização utilizando o metamodelo da UML.

Este tipo de representação é importante por tratar-se de um conhecimento dos engenheiros de software diferente daquele para o qual a ontologia Kuaba demonstrou habilidade em representar - conhecimento do domínio e do design apoiado na semântica do metamodelo. Esse conhecimento está associado à habilidade de saber ou intuir relacionamentos entre raciocínios, que são decisões a respeito de elementos de modelos que ele sabe que se relacionam, seja na mesma etapa do design, ou não, ou em modelos diferentes. A ontologia Kuaba na sua versão atual não oferece apoio a este tipo de representação.

O relacionamento entre *design rationales* é uma investigação importante, tanto na Engenharia de Requisitos, como na própria Engenharia de Software como um todo. Embora analisado neste trabalho de pesquisa, o estudo sobre as alterações necessárias no vocabulário da ontologia Kuaba para apoiar sua representação é deixado como trabalho futuro.

O registro de relacionamentos entre *design rationales* também pode contribuir para o rastreamento de artefatos do modelo de requisitos e outros modelos. Essa contribuição é maior quando há semântica na ontologia para representar os relacionamentos entre *design rationales*, nos casos em que os modelos foram instanciados de metamodelos diferentes ou o metamodelo usado no design não apoie esta semântica.

O rastreamento de requisitos é definido por Gotel e Finkelstein (1994) como a habilidade de descrever a vida do requisito nos dois sentidos, para frente e para trás, isto é, desde suas origens, através de sua análise e sua especificação, até sua subsequente implantação e uso. E ainda, através de períodos de refinamentos em qualquer dessas atividades. Seguindo nesta mesma linha de pensamento, Jarke (1998) classifica os rastreamentos como: *(a)* dos requisitos para frente, em que a responsabilidade pela satisfação de requisitos é definida para componentes do sistema de tal forma que possa ser feita a avaliação de impacto de mudanças nesses requisitos; *(b)* para trás, para os requisitos, uma vez que a adequação do sistema aos requisitos deve ser verificada para evitar designs que não estejam relacionados com algum requisito; *(c)* para frente para requisitos, pois mudanças nas necessidades dos envolvidos, e também premissas técnicas, podem implicar em reavalições radicais na importância de certos requisitos; e *(d)* para trás de requisitos, uma vez que a cadeia de contribuições ou benefícios, subjacentes aos requisitos, são cruciais na sua validação, especialmente em ambientes onde há grande influência política.

Essa classificação é combinada com a classificação do aspecto ao qual se relaciona o rastreamento em Ramesh e Jarke (2001), que define os três focos do rastreamento como sendo no que os autores chamam de “fonte”, nos “envolvidos” e nos “objetos” ou “artefatos”. O primeiro aspecto tem relação com a gestão de configuração e versão de documentos

abordados em Rose (1991) e Conradi e Westfechtel (1998), que enfatizam o papel da rastreabilidade na gestão de documentos; o segundo trata os envolvidos com a gestão da rastreabilidade, como em Yu e Mylopoulos (1994), usando as dependências entre esses agentes como ponto de partida para guiar e documentar os processos da Engenharia de Requisitos; e o terceiro, mais comum, trabalha com os tipos de artefatos de design e os tipos de ligações entre eles, como aquelas definidas no parágrafo anterior. Esse último possibilita que a gestão de mudanças seja exercida em um nível bem mais refinado de detalhe, ao passo que trabalha sobre os próprios elementos que compõem o sistema durante todas as atividades do desenvolvimento.

O registro dos relacionamentos entre *design rationales* pode dar apoio direto às duas taxonomias, de sentido e do foco do rastreamento, em relação ao versionamento de modelos e associações entre artefatos de design. O rastreamento dirigido para o aspecto dos "envolvidos" pode ser tratado indiretamente por elementos no vocabulário da ontologia Kuaba como Pessoa e Papel.

8 CONCLUSÕES

Este trabalho de pesquisa estudou as possibilidades de representação de *design rationale* usando a abordagem Kuaba e como ela pode contribuir para as práticas da Engenharia de Requisitos. Como observado no Capítulo 2, apesar dos estudos sobre a representação de *design rationale* na Engenharia de Software não serem recentes, este é um tema muito complexo e abrangente. As pesquisas relatadas na literatura são em pequena quantidade e ainda com o objetivo formativo no que diz respeito a esta área de conhecimento. A abordagem Kuaba para representação de *design rationale* ainda é mais recente, sendo abordada pela primeira vez em Medeiros, Schwabe e Feijó (2005a, 2005b).

Estudar as possibilidades de representação de *design rationale* para o modelo de requisitos utilizando a abordagem Kuaba pressupõe que esse modelo seja resultado do processo de instanciação de um metamodelo. Embora esse fato seja significativo, ele, apenas, não é suficiente para explorar a potencialidade de tal abordagem, que incorpora a própria semântica do metamodelo na instância de seu modelo de representação. É necessário que a semântica do metamodelo usado no design seja capaz de dar significado aos conceitos do domínio estudado e que seja possível usar essa semântica para instanciar a ontologia Kuaba. Isso se torna ainda mais importante quando se trata do modelo de requisitos, onde os conceitos são definidos com alto grau de abstração e usualmente informados aos engenheiros de software em linguagem natural, seja em sessões de levantamento de requisitos, seja através de textos que os descrevem.

No Capítulo 6 foram descritos e discutidos exemplos de representação de *design rationale* para modelos de requisitos usando a abordagem Kuaba. Os exemplos foram escolhidos a fim de permitir que os meta-conceitos tivessem sua semântica usada integralmente na instância da ontologia. Além disso, foi escolhido um metamodelo capaz de representar a complexidade do domínio de requisitos e consequentemente dar significado aos

conceitos dos domínios da Biblioteca e da Submissão de artigos para conferências científicas, estudados nesta pesquisa. Um metamodelo descritivo não serviria ao objetivo deste trabalho como, por exemplo, o modelo de casos de uso utilizado na UML, que descreve interações entre atores e sistema de forma textual. A única semântica possível de ser obtida, neste caso, é da estruturação dos atores e dos casos de uso, que dá pouco significado ao modelo, onde os textos são a grande fonte de requisitos (BOOCH; RUMBAUGH; JACOBSON, 1999; OMG, 2001, 2006a, 2006b, AMBLER; 2004).

Neste capítulo são apresentadas as observações feitas durante os exercícios de representação de *design rationale* para modelos de requisitos, que se traduzem nas principais contribuições deste trabalho de pesquisa. Essas observações e contribuições são classificadas em função do tipo de descoberta relevante e apresentadas na Seção 8.1. A Seção 8.2 trata dos trabalhos futuros decorrentes desta pesquisa.

8.1 CONTRIBUIÇÕES DA PESQUISA

A investigação sobre a representação de *design rationale* na Engenharia de Requisitos, usando a abordagem Kuaba é uma pesquisa abrangente e extensa, uma vez que existem aspectos dessa investigação que ainda não foram abordados em pesquisas anteriores. Para maior clareza na exposição e discussão dos achados e contribuições deste trabalho de pesquisa, esta seção foi dividida em subseções temáticas. Embora alguns temas possam estar relacionados, os assuntos abordados em cada tema foram separados a fim de possibilitar um melhor entendimento.

8.1.1 Representação de *Design Rationale* para Modelos de Requisitos com Kuaba

No Capítulo 6 foi mostrado como instanciar a ontologia Kuaba incorporando a semântica do metamodelo do *framework* KAOS, demonstrando que a abordagem Kuaba também pode ser utilizada para representar *design rationale* de modelos de requisitos. Um padrão é naturalmente definido para gerar representações de *design rationale* com base na ontologia Kuaba, uma vez que o formalismo fornecido pelo metamodelo prescrito pelo método de design é usado na instanciação dos elementos da ontologia. Isso quer dizer que o conhecimento dos engenheiros de software sobre o domínio, sobre a semântica dos conceitos instanciados através do metamodelo e a experiência aplicada por esses engenheiros podem ser representados usando essa abordagem.

Um aspecto geral que foi observado é que a associação da semântica do metamodelo ao *design rationale* registrado durante a especificação de requisitos de software indica uma melhoria da qualidade dos modelos criados, na medida em que as possibilidades de navegação do metamodelo são levadas em conta quando a ontologia Kuaba é instanciada. Isto evita incorreções sintáticas e previne incorreções semânticas nas especificações dos requisitos (SANTOS; MEDEIROS, 2009).

A avaliação dos conjuntos de testes projetados para as três categorias revelou que a representação de *design rationale* é mais completa e cuidadosa quando se trabalha em paralelo. Além disso, há indicações que o modelo tem mais qualidade. Isso acontece porque nenhum argumento é esquecido e mesmo aqueles argumentos que são fruto do que é aparentemente óbvio são registrados quando o *design rationale* é registrado durante a modelagem. Cada novo elemento acrescentado ao modelo leva ao raciocínio sobre argumentos e decisões para a representação de *design rationale*, que por sua vez pode implicar em uma revisão no próprio modelo, criando-se assim ciclos de aprimoramento durante o processo de modelagem e representação de *design rationale*.

No caso do registro de *design rationale* ser feito após o modelo ter sido finalizado, notou-se que as representações ficam mais pobres e os modelos com mais defeitos, pois não há condições de recorrer a todas as lembranças sobre os raciocínios realizados e as alternativas de solução vislumbradas na ocasião da modelagem, mesmo que o registro seja realizado quase que imediatamente após a conclusão do modelo.

Outro aspecto observado é que quando o engenheiro de software possui conhecimento suficiente da semântica do metamodelo que ele está utilizando no design é possível fazer a representação de *design rationale* sem construir o modelo previamente. Essa prática levou a representações mais completas, ou seja, com argumentação mais cuidadosa, melhores escolhas e com maior riqueza de detalhes. Neste caso, o modelo poderia ser computacionalmente derivado da representação de *design rationale*, uma vez que esta é obtida a partir da semântica do metamodelo de design.

Em relação ao detalhamento dos registros de *design rationale*, ou seja, o volume de argumentos registrados para a tomada de decisão sobre uma solução de design, também foram feitas algumas observações. Geralmente, na tarefa de design de software, seja com qual metamodelo se esteja trabalhando, nem todos os conceitos levam a raciocínios mais cuidadosos a seu respeito como, por exemplo, modelar o tipo de uma propriedade de uma classe pode ser uma solução bastante simples, se o atributo é o nome de uma pessoa. Neste caso, é claro que o tipo cadeia de caracteres é uma solução muito simples. No entanto, no caso

do endereço, já é exigido um raciocínio mais cuidadoso, uma vez que se pode prescindir da adição de uma entidade mais elaborada no modelo para descrever o tipo desse atributo. Nem todas as decisões tomadas nas representações de *design rationale* necessitam de muito detalhamento, e isto não as torna mais pobres.

8.1.2 Apoio à Validação de Metamodelos

O modelo conceitual do *framework* KAOS, na versão estudada (RESPECT-IT, 2007), não possui elemento sintático explícito para diferenciar as representações de refinamentos de objetivos do tipo AND e do tipo OR. Outro caso acontece com o metamodelo da UML, em todas as versões, no que diz respeito à definição de sintaxe específica para apoiar as regras de operações abstratas e concretas de classes no modelo, uma vez que não existe na semântica do metamodelo uma maneira de garantir que classes concretas não tenham operações abstratas. Também não há semântica para garantir que se uma classe concreta é herdeira de uma classe abstrata, todas as operações abstratas devem ser concretas e possuir especificações em métodos (suas devidas implementações).

Essas constatações foram feitas em função de que uma instância da ontologia Kuaba (*design rationale*), integrando a semântica do metamodelo, necessita de conceitos precisos para sua representação. Isso indica uma qualidade da abordagem Kuaba observada durante o trabalho de pesquisa não prevista e que não era objetivo de investigação específica, que seja apoiar a validação da capacidade de representação de certos conceitos através do metamodelo.

8.1.3 Apoio à Representaçao de Relacionamentos entre *Design Rationale*

Uma das formas de manifestação do conhecimento e experiência dos engenheiros de software, individualmente ou de maneira colaborativa, no desenvolvimento de software, é a associação entre raciocínios, ou entre as soluções de design dadas para os artefatos de seus modelos. Por exemplo, elementos de um modelo de operações em KAOS (um modelo de requisitos), pode estar relacionado com alternativas de soluções sobre a arquitetura dos componentes do software (modelo de arquitetura), ou sobre o design de projeto de determinadas classes de implementação (modelo de implementação).

O registro desse conhecimento é importante para o próprio engenheiro de software, uma vez que permite que algumas alternativas de solução para outros designs já tenham sido registradas e explicadas. É importante também para outros engenheiros de software que

podem fazer uso dessas associações entre raciocínios para apoiar novos designs, além de contribuir também como ferramenta de rastreabilidade, neste caso, da rastreabilidade de raciocínio e não daquela usual apoiada por várias ferramentas da Engenharia de Software, que tratam do rastreamento entre artefatos dos modelos.

Esta pesquisa contribui para a discussão do tema de relacionamentos entre *design rationales* por ter identificado que quando se tem modelos diferentes e o metamodelo utilizado não possui semântica para representar as associações entre elementos desses modelos, mesmo que o engenheiro de software saiba que há uma relação entre os artefatos de ambos os modelos, é necessário que a ontologia Kuaba supra essa lacuna semântica. Além disso, quando o design dos modelos produzidos envolve o uso de metamodelos diferentes, a semântica dos metamodelos não pode ser utilizada, uma vez que não são equivalentes, novamente necessitando de semântica na ontologia para essa representação. Foi observado na pesquisa que a ontologia Kuaba não possui elementos para apoiar a representação dessas associações, que são os relacionamentos de *design rationale*. O caso em que há apoio da semântica do metamodelo às associações entre elementos de modelos diferentes, como em KAOS, não faz parte do espaço do problema, pois a representação de *design rationale* para associações entre elementos de modelos diferentes já está resolvida. Uma discussão mais detalhada sobre o tema foi apresentada no Capítulo 7.

8.1.4 Uso de *Design Rationale* na Evolução de Requisitos

A maioria das ferramentas atuais de modelagem de software (incluindo a modelagem de requisitos) fornece algum tipo de controle de versão, próprio da ferramenta, ou integrado a ela através de um mecanismo de acoplamento qualquer. O controle de versão pode ser feito sobre o modelo, arquivo de modelo, ou sobre o artefato sendo projetado. Além disso, as ferramentas costumam oferecer mecanismos de rastreamento de artefatos nos dois sentidos, tanto os produzidos em atividades mais cedo, para outros produzidos posteriormente, quanto o inverso (NUSEIBEH e EASTERBROOK, 2000). Essas ferramentas apoiam a avaliação do impacto de mudanças em especificações em todo o ciclo de vida do desenvolvimento de software. Embora esses mecanismos sejam muito úteis, o gerenciamento da evolução de artefatos, notadamente em requisitos é complexo e exige muito esforço, mesmo apoiado por ferramentas. Wiegers (2003) apresenta um *checklist* que dá uma dimensão das dificuldades inerentes ao processo de análise de impacto.

O rastreamento de artefatos geralmente leva em conta uma dependência sem significado entre artefatos limitando a eficácia da análise do impacto no projeto, uma vez que identificada a origem da mudança, ou seja, que requisito será alterado, indica apenas a quantidade de artefatos que poderão ser alterados, produzidos em cada atividade do desenvolvimento. Nos mecanismos de controle de versões o apontamento é uma descrição textual, em linguagem natural, das modificações realizadas. A representação de *design rationale* com Kuaba leva em conta as decisões tomadas no design dos artefatos, com base em argumentos que traduzem o raciocínio dos engenheiros de software sobre o domínio do problema e sobre o uso dos conceitos do metamodelo, para expressar as abstrações desse domínio. Esses argumentos podem apoiar a análise de impacto através da possibilidade do uso de elementos semânticos nessa análise. Além disso, como as representações de *design rationale* possuem semântica comum, elas podem ser comparadas e medidas, assim como o impacto das mudanças em requisitos através de mecanismos de rastreamento da sua própria rede semântica.

A representação de *design rationale* com Kuaba não é apenas um texto escrito em linguagem natural, descrevendo a mudança. Essa representação oferece uma maneira superior de abordagem da análise de mudanças, pois a análise é feita sobre conhecimento registrado a partir de um modelo formal que usa a semântica do metamodelo usado no design dos artefatos, sendo os argumentos que apóiam as decisões tomadas no design, registrados através de uma linguagem computável. Embora uma linguagem formal seja importante, a própria análise das representações de *design rationale* já oferece mecanismo superior para análise de impacto.

8.2 TRABALHOS FUTUROS

Esta dissertação deixa alguns trabalhos futuros a fim de suprir aspectos complementares, resumidos a seguir.

8.2.1 Estudo de Caso

Os testes de simulação de tarefas do dia-a-dia dos engenheiros de software na modelagem foram realizados por apenas um engenheiro, e não obteve dados quantitativos com base em um método de medição pré-definido. Um importante trabalho futuro é a concepção de um estudo de caso envolvendo um grupo de engenheiros de software definindo

os procedimentos e os critérios de medição para o trabalho desses engenheiros. Estes resultados podem contribuir para a verificação das observações qualitativas obtidas neste trabalho de pesquisa.

8.2.2 Especificação Formal e Reuso de *Design Rationale* para Requisitos

Este trabalho de pesquisa investigou a representação de *design rationale* para modelos de requisitos usando a abordagem Kuaba e o metamodelo KAOS. As representações de *design rationale* foram geradas apenas em uma versão gráfica construída com o apoio da ferramenta de modelagem *Enterprise Architect*, usando uma extensão da linguagem UML, com foi explicado no Capítulo 6. Um trabalho futuro importante é a especificação das representações geradas em uma linguagem formal, F-Logic ou OWL, e a aplicação das operações implementadas por Medeiros (2006) para avaliar as possibilidades de reuso dos modelos de requisitos a partir do *design rationale* registrado. Esse trabalho poderia ratificar algumas observações feitas nesta pesquisa sobre o uso de *design rationale* no apoio à evolução de requisitos.

8.2.3 Alteração da ferramenta KSE para apoiar a captura de *Design Rationale* para Modelos de Requisitos

A ferramenta KSE (*Kuaba Software Engineering*, pronunciada como CASE - *Computer Aided Software Engineering* - Queisse, em inglês) desenvolvida por Nunes e Medeiros (2009) tem como objetivo apoiar os engenheiros de software no registro de *design rationale* durante seu trabalho de design. KSE permite que durante o trabalho de modelagem as instâncias dos elementos de raciocínio da ontologia Kuaba sejam automaticamente obtidas em função do elemento do metamodelo de design que se está usando no artefato, sendo os argumentos capturados imediatamente. Nesta ferramenta a ontologia Kuaba é utilizada para apoiar o registro de *design rationale* de modelos baseados no metamodelo da UML. Utilizar a ontologia apoiando o registro de *design rationale* para modelos de requistos que usam o metamodelo KAOS na ferramenta é um trabalho futuro importante. Este estudo pode trazer possibilidades de novos testes de modelagem e representação, criando perspectivas de novas descobertas sobre a representação de *design rationale* usando a abordagem Kuaba na Engenharia de Requisitos.

8.2.4 Alterações na Ontologia Kuaba para Apoiar Relacionamentos entre *Design Rationales*

A continuação dos estudos para que a ontologia da abordagem Kuaba possa apoiar a representação de relacionamentos entre *design rationales* é um trabalho futuro muito importante decorrente desta pesquisa, uma vez que, além de complementar as possibilidades de registro do conhecimento aplicado ao design de software, possibilita que testes mais cuidadosos e elaborados possam ser realizados para estudar sua eficácia na análise de impacto de mudanças em requisitos baseadas em rastreamento de raciocínio.

8.2.5 Estudar as Possibilidades de Inserção da Abordagem Kuaba no Processo da Engenharia de Software

Esta pesquisa realizou testes e estudou a representação de *design rationale* utilizando a abordagem Kuaba para o modelo de requisitos orientado para objetivos. A pesquisa foi dirigida para viabilizar a representação de *design rationale* para os modelos do *framework* KAOS e investigar sua contribuição para a prática da Engenharia de Requisitos. No entanto, é importante analisar implicações dessa aplicação sob o foco metodológico, buscando convergir sobre aspectos de ordenação das tarefas de uma maneira geral onde a representação de *design rationale* seja inserida no processo de trabalho, ou seja, analisar o uso da representação de *design rationale* sistematicamente como parte de uma metodologia de desenvolvimento de software.

8.2.6 Abordagem Holística para *Design Rationale* com Kuaba na Engenharia de Requisitos

Os trabalhos estudados na literatura sobre a aplicação de *design rationale* na Engenharia de Requisitos conduzem a uma reflexão de que podem existir três fases na Engenharia de Requisitos: a fase inicial, a fase de descoberta e a fase de especificação. A primeira delas onde se sabe pouco sobre os requisitos e não há muito compromisso entre os participantes pelo esforço de desenvolvimento, levando a uma melhor abordagem, como no trabalho de Mackenzie (2006) e Rooksby, Sommerville e Pidd (2006) que parecem referir-se aos requisitos iniciais (cedo) de Yu (1997a, 1997b, 2001a). A abordagem de Nguyen e

Swatman (2003, 2006) parece ajustar-se ao processo de descoberta, assim como a abordagem Kuaba, que também contribui para o processo de descoberta, embora esta última esteja mais relacionada com a modelagem, ou seja, a especificação de sistemas e software.

A abordagem Kuaba trabalha com o domínio de design baseado em modelos, ou seja, pressupõe a existência de um metamodelo que guia o design. Embora o metamodelo de *i** possa ter uma semântica interessante para a representação dos requistos inicias (*early requirements*) de Yu, ele não oferece semântica para representar o processo de negociação de requisitos como abordados em Mackenzie (2004), Rooksby, Sommervile e Pidd (2006) e Bhoem e Kiptaci (2006). Os trabalhos estudados apresentam a obtenção de *design rationale* de maneira informal, não havendo um metamodelo que permita a equivalência entre os conceitos e possua semântica para a representação das negociações. Entretanto, a literatura estudada para esta pesquisa indica referências para estudos sobre metamodelos que apoiam a representação de processos de negociação. É proposto no trabalho de Robinson e Volkov (1998) um metamodelo similar aqueles estudados, orientados para objetivos, de onde evoluiram KAOS e *i**.

O estudo da representação de *design rationale* usando Kuaba, e a semântica de um metamodelo que apóie a representação do processo de negociação, como o proposto por Robinson e Volkov, é um trabalho futuro de investigação interessante. Ainda não há uma abordagem mais abrangente para este tema. Essa pode ser uma linha interessante de investigação que busca integrar os esforços da pesquisa realizada em um único corpo de conhecimento científico, que venha trazer benefícios aos praticantes da Engenharia de Requisitos.

REFERÊNCIAS BIBLIOGRÁFICAS

ACKERMAN, F.; EDEN, C.; CROPPER, S. Getting Started with Cognitive Mapping. In: YOUNG OR CONFERENCE, 7, 1992, University of Warwick. **Proceedings...** Coventry, UK, 1992. p. 65 - 82.

AL-SUBAIE, H. S. M. **Evaluating the Effectiveness of a Goal-Oriented Requirements Engineering Method.** 2007. 1 v. Thesis (PhD) - King´s College, London - UK, 2007.

AMBLER, S. W. **The Object Primer : Agile Model-Driven Development with UML 2.0**. Cambridge, UK: Cambridge University Press, 2004.

BOHEM, B. Anchoring the Software Process. **IEEE Software**. v. 3, n. 1, 1996. p. 73-82. Los Alamitos, CA, USA: IEEE Computer Society Press. DOI: 10.1109/52.526834

BOHEM, B.; KITAPCI, H. The WinWin Approach: Requirements Negotiation Tool for Rationale Capture and Use. In: DUTOIT, A. H. et al. **Rationale Management in Software Engineering.** Secausus, New Jersey - USA: Springer-Verlag New York Inc., p. 173-208, 2006.

BOOCH G.; RUMBAUGH J.; JACOBSON I. **The Unified Modeling Language – User Guide**. Reading, MA, USA: Addison Wesley, 1999.

BRESCIANI, P.; GIORGINI, P.; GIUNCHIGLIA, F.; MYLOPOULOS J.; PERINI, A. Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agents Systems. DIT-02-0015. **Technical Report**, 2004, p. 203-236. Department of Information and Communication Technology, University of Trento, Italy.

BURGE, J.; BROWN, D. C. Rationale Support for Maintenance of Large Scale Systems. **Workshop on Evolution of Large-Scale Industrial Software Applications (ELISA)**, ICSM '03, Amsterdam, NL, 2003.

BURGE, J. E.; BROWN, D. C. Discovering a Research Agenda for Using Design Rationale in Software Maintenance. **Computer Science Technical Report**, Worcester Polytechnic University, WPI-CS-TR-02-03, 2002.

CHICOTE, C. V.; MOROS, B.; TOVAL A. REMM-Studio: an integrated model-driven environment for requirements specification, validation and formatting. In: **Journal of Object Technology**, Special Issue TOOLS EUROPE 2007-6, pages 437-454, October 2007.

CONKLIN, J.; BEGEMAN, M. L. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. **ACM Transactions Information Systems**. 1988a. v. 6, n. 4, October 1988, p. 303-331. DOI= http://doi.acm.org/10.1145/58566.59297

CONRADI, R.; WESTFECHTEL, B. Version models for software configuration management. **ACM Computing Surveys**, v. 30, n. 2 1998, p. 232-282.

DAHLSTEDT, A. G. Requirements interdependencies - moulding the state of research into a research agenda. In: International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ), 9, 2003, **Proceedings ...** p. 71-80, 2003, held in conjunction with CaiSE.

DARDENNE, A., FICKAS, S., LAMSWEERDE, A. Goal-directed concept acquisition in requirements elicitation. In: International Workshop on Software Specification and Design. 6. Como, Italy, October 25 - 26, 1991. **Proceedings ...** p.14-21. International Workshop on Software Specifications & Design. IEEE Computer Society Press, Los Alamitos, CA, 1991.

DARDENNE, A.; LAMSWEERDE, A.; FICKAS, S. Goal-directed requirements acquisition. In: SINTZOFF, C. et al. 1993. **Selected Papers of the Sixth international Workshop on Software Specification and Design**. Amsterdam, The Netherlands: Elsevier Science Publishers B. V., 1993, p.3-50.

DARIMONT, R.; DELOR, E.; MASSONET, P.; LAMSWEERDE, A. GRAIL/KAOS: an environment for goal-driven requirements engineering. In: International Conference on Software Engineering, 19. Boston, Massachusetts, United States, May 17 - 23, 1997, ICSE '97. **Proceedings ...** p. 612-613. New York, NY:ACM, 1997.

DI MAURO, N.; BASILE, T. M.; FERILLI, S. GRAPE: an expert review assignment component for scientific conference management systems. M. Ali and F. Esposito, Eds. Lecture In: ALI, M.; ESPOSITO, F. **Lecture Notes in Computer Science**. London, UK: Springer-Verlag, 2005, p.789-798. DOI= http://dx.doi.org/10.1007/11504894_109

DUBISY, F.; LAMSWEERDE A.; DARDENNE, A. The KAOS Project: Knowledge acquisition in automated specification of software. In: AAAI. **Proceedings ...** Spring Symposium Series, California, United Sates of America, 1991.

DUTOIT, A. H.; MACALL, R.; MISTRIK I.; PAECH, B. **Rationale Management in Software Engineering**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

DUTOIT, A. H.; MACALL, R.; MISTRIK I.; PAECH, B. Rationale Management in Software Engineering: Concepts and Techniques. **Rationale Management in Software Engineering**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006a.

EDEN, C. Using Cognite Mapping for Strategic Options Development and Analysis (SODA). In: ROSENHEAD **Rational Analysis for a Problematic World**. Chichester, UK: Wiley, 1990.

EDEN, C.; AKERMAN, F. SODA - The Principles. In: ROSENHEAD, J.; MINGERS, J. **Rational Analysis for a Problematic World Revisited**. 2001. Chichester, UK: Wiley, 2001.

ERIKSSON, H.; PENKER, M. **UML Toolkit**. London, UK: John Wiley & Sons, Inc., 1998.

FIGUEIREDO, S.; ROCHA, A. C. R.; SANTOS, G.; MONTONI, M. Uma Abordagem de Apoio à Solução Técnica em Ambientes de Desenvolvimento de Software Orientados à Organização. In: Simpósio Brasileiro de Qualidade de Software. **Anais …** 2006. Vila Velha, ES-Brasil. 2006.

FISHER R.; URY W. **Getting To Yes: Negotiating Agreement Without Giving In**. Boston, USA: Houghton Mifflin, 1981.

FUXMAN A.; LIU L.; PISTORE, M.; ROVERI, M.; MYLOPOULOS, J. Specifying and analyzing early requirements in Tropos: Some experimental results. **Requirements Engineering**, v. 9, n. 2, 2004, p. 132–150.

GIORGINI, P.; KOLP M.; MYLOPOULOS J.; PISTORE, M. The Tropos Methodology: an overview. In: Methodologies and Software Engineering for Agent Systems, **Proceeding ...** Norwell, MA, USA: Kluwer, 2003.

GIUNCHIGLIA, F.; MYLOPOULOS J.; PERINI, A. The Tropos software development methodology: processes, models and diagrams. In: AAMAS '02: International joint conference on Autonomous agents and multi-agent systems, 2002. **Proceedings ...** p. 35-36, New York, NY, USA: ACM, 2002.

GOTEL, Orlena; FINKELSTEIN, Anthony. An analysis of the requirements traceability problem. In: International Conference on Requirements Engineering, 1994. **Proceedings…** 1994, p. 94-101. IEEE Computer Society Press.

GREENSPAN, S. J., MYLOPOULOS, J., BORGIDA, A. Capturing more world knowledge in the requirements specification. In: International Conference on Software Engineering, Tokyo, Japan, September 13-16, 6, 1982. International Conference on Software Engineering, 1982, p.225-234. Los Alamitos, CA, USA . IEEE Computer Society Press.

GREENSPAN, S. J. **Requirements Modeling: A Knowledge Representation Approach to Software Requirements Definition**. Ph. D. Thesis - University of Toronto (Canada), 1984a.

GREENSPAN, S. J., MYLOPOULOS, J., BORGIDA, A. On Formal Requirements Modeling Languages: RML Revisited. International Conference on Software Engineering, 1994b, **Proceedings ...** p.135-147.

HEAVEN, W.; FINKELSTEIN A. A UML Profile to support requirements engineering with KAOS. **IEEE Proceedings Software**, 2007, p.541–560.

ISO/IEC, **ISO/IEC 9126: Information technology - Software Product Evaluation - Quality characteristics and guidelines for their use**, 1991.

IUT-T, International Union for telecommunication. **Z.151 Recommendation**, 2008.

IEEE - Institute of Electrical & Electronics Engineers. **IEEE Recommended Practice for Software Requirements Specifications**. Institute of Electrical & Electronics Engineers, Inc. 1998.

JACOBSON I.; BOOCH G.; RUMBAUGH J. **The Unified Development Process**. Reading, MA, USA: Addison Wesley, 1999.

JARKE, M. Requirements Tracing. **Communications of the ACM** v. 41, n. 12, 1998. p. 32-36. DOI= http://doi.acm.org/10.1145/290133.290145, 1998.

KIFER, M.; LAUSEN, G. F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance and Scheme. **ACM SIGMOD**, 1989, p.134-146.

KOTONYA, G., SOMMERVILLE, I. **Requirements Engineering**. New York, USA: John Wiley & Sons, Inc. 1997.

KUNZ W.; RITTEL H. W. J. Issues as Elements of Information Systems. **Institute of Urban and Regional Development Working Paper 131**, University of California, Berkeley, CA, http://www-iurd.ced.berkeley.edu/pub/WP-131.pdf, 1970.

LACAZE, X. **Conception rationalise pour les systèmes interactifs - Une notation semi formelle et un environnement d'édition pour une modélisation des alternatives de conception**. Thésis (Ph.D.), Université Toulouse I, Toulouse, France, 2005.

LACAZE, X.; PALANQUE, P.; BARBONNI, E.; BASTIDE, R.; NAVARRE, D. From DREAM to Reality: Specificities of Interactive Systems Development With Respect to Rationale Management, In: DUTOIT, A. H. Et al. **Rationale Management in Software Engineering**, Secaucus, New Jersey, USA: Springer-Verlag New York Inc., 2006, p. 155-172.

LAMSWEERDE A.; DARIMONT, R.; MASSONET, P. Goal-directed elaboration of requirements for a meeting scheduler: problems and lessons learnt. IEEE International Symposium on Requirements Engineering, RE '95, 1995. **Proceedings ...** p.194–203.

LAMSWEERDE A. Requirements engineering in the year 00: a research perspective. ICSE '00: International Conference on Software Engineering, 22nd, 2000, **Proceedings ...** p.5–19, New York, NY, USA: ACM, 2000.

_____. **Requirements Engineering**. London, UK: John Wiley & Sons Inc. 2009.

LEE, J.; LAI, K. What's in design rationale?, **Human-Computer Interaction**. v. 6, n. 3, 1991, p. 251-280.

LEE, J. Design rationale systems: understanding the issues, **IEEE Expert**. v. 12, n. 13, 1997, p. 78-85.

LETIER, E. **Reasoning about Agents in Goal-Oriented Requirements Engineering**. Thesis(PhD), Université Catholique de Louvain, Louvain, Belgium, May 2001.

LIDDLE, Stephen W. **The BYU Paper Review System**. Available at: <http://blondie.cs.byu.edu/PaperReview/>. Last access: Março 2009.

MCCALL, R. PHI: a conceptual foundation for design hypermedia, **Design Studies**. v. 12, n.1, 1991, p. 30-41.

MACKENZIE, A.; PIDD, M.; ROOKSBY, J.; SOMMERVILLE, I.; WARREN, I.; WESTCOMBE, M. Wisdom, decision support and paradigms of decision making. **European Journal of Operational Research** n. 170, p. 156–171, 2006.

MACLEAN, A.; YOUNG, R. M.; BELLOTTI, V. M.; MORAN, T. P. Questions, options, and criteria: elements of design space analysis, **Human-Computer Interaction**. v. 6, n. 3, 1991, p. 201-250.

MACLEAN, A.; YOUNG, R. M.; BELLOTTI, V. M.; MORAN, T. P. Questions, Options and Criteria, In: MORAN, T.; CAROLL, J. M. **Design Rationale, Concepts, Techniques and Use**. Mahwah, New Jersey USA: Lawrence Erlbaum Associates, 1996, p. 53-106.

MARTIN, R. **Clean Code: A Handbook of Agile Software Craftsmanship**. Prentice Hall, 2008.

MEDEIROS, A.; SCWABE, D.; FEIJÓ, B. Kuaba ontology: Design rationale representation and reuse in model-based designs. International Conference on Conceptual Modeling, 24, ER2005, **Proceedings ...** Klagenfurt, Austria: Springer-Verlag 2005a, p.241–255.

_____. A design rationale representation of model-based designs in software engineering. Conference on Advanced Information Systems Forum, 17th, CAiSE05, **Proceedings ...** 2005b, p.163–168. Porto, Portugal: FEUP ,2005 v1.

MEDEIROS, A. **Kuaba: Uma Abordagem para Representação de Design Rationale para o Reuso de Designs baseados em Modelo**. Tese (PhD), 1 v, Pontificia Universidade Catlica Rio de Janeiro PUC-RIO, Rio de Janeiro, Brasil, 2006.

MEDEIROS, A.; SCWABE, D. Kuaba Approach: Integrating Formal Semantics and Design Rationale Representation to Support Reuse. **Artificial Intelligence for Engineering Design, Analysis and Manufacturing**, v. 22, 2008, p. 399-419. Cambridge University Press, 2008.

MORAN, T.; CAROLL, J. M. **Design Rationale, Concepts, Techniques and Use**. Mahwah, New Jersey USA: Lawrence Erlbaum Associates, 1996

MYLOPOULOS, J., BORGIDA, A., JARKE, M., KOUBARAKIS, M. Telos: representing knowledge about information systems. **ACM Transactions on Information Systems** v. 8, n. 4, October 1990, p.325-362. DOI= http://doi.acm.org/10.1145/102675.102676

NGUYEN, L.; SWATMAN, P. A. Managing the requirements engineering process. **Requirements Engineering**, v. 8, 2003, p. 55-68. DOI: 10.1007/s00766-002-0136-y.

NGUYEN, L.; SWATMAN, P. A. Promoting and Supporting Requirements Engineering Creativity, In: DUTOIT, A. H. et al. **Rationale Management in Software Engineering.** Secausus, New Jersey - USA: Springer-Verlag New York Inc., 2006. p. 209-230.

NUNES, T. R.; MEDEIROS, A. P. KSE: Ferramenta de Apoio à Captura e Representação semi-automática de Design Rationale em Engenharia de Software. In: **XXIII Simpósio Brasileiro de Engenharia de Software (SBES)** - Sessão de Ferramentas, 2009, Fortaleza. XVI Sessão de Ferramentas - SBES 2009, 2009.

NUSEIBEH, B., EASTERBROOK, S. Requirements Engineering: A Roadmap. In: Conference on the Future of Software Engineering, ICSE, 2000, **Proceedings ...** p.35-46. New York, NY, US: ACM Press. DOI: 10.1145/336512.336523, 2000.

OMG **Unified Modeling Language Specification version 1.4**. Available at http://www.uml.org, 2001.

OMG **Unified Modeling Language 2.0: Infrastructure**. Available at <http://www.uml.org>, 2006a.

OMG **Unified Modeling Language 2.0: Superstructure**. Available at <http://www.uml.org>, 2006b.

OMG **Meta Object Facility (MOF) Core Specification**. Available at http://www.mof.org, 2006c.

POHL, K. **Process-Centered Requirements Engineering**. UK: Wiley, 1996.

PRESSMAN, R. S. **Software Engineering : a Practitioner's Approach**. 7.ed. NY, USA: McGraw-Hill, Inc, 2009.

RAMESH, B.; JARKE, M. Toward Reference Models for Requirements Traceability. **IEEE Transactions Software Engineering**, v. 27, n. 1, January 2001, p. 58-93. DOI= http://dx.doi.org/10.1109/32.895989.

RESPECT-IT. **KAOS Tutorial version 1.0**. Available at <http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf>, 2007.

ROBINSON, W. N.; VOLKOV, V. Supporting the negotiation life cycle**. Communications of the ACM**, v. 41, n. 5, May, 1998, p. 95-102. DOI = http://doi.acm.org/10.1145/274946.274962

ROOKSBY, J.; SOMMERVILLE, I.; PIDD, M. A Hybrid Approach to Upstream Requirements: IBIS and Cognitive Mapping. In: DUTOIT, A. H. et al. **Rationale Management in Software Engineering.** Secausus, New Jersey - USA: Springer-Verlag New York Inc., 2006. p. 137-154.

ROSE, T.; JARKE, T.; GOCEK, M.; MALTZMAN, C. G.; NISSEN, H. W. A decision-based configuration process environment. **IEEE Software Engineering Journal**, v. 6, n. 5, 1991, p. 332-346.

ROSS, D. T., SCHOMAN Jr., K. E. Structured analysis for requirements definition. In: **Classics in software engineering**, 1979, p.363-386, Yourdon Press, Upper Saddle River, NJ, USA. ISBN: 0-917072-14-6.

RUMBAUGH J.; JACOBSON I.; BOOCH G. **The Unified Modeling Language – User Guide**. Reading, MA, USA: Addison Wesley, 1999.

SANTOS, E. G.; MEDEIROS, A. P. Associando Semântica ao Rationale na Engenharia de Requisitos. In: Conferência IADIS Ibero-Americana WWW/Internet (CIAWI) 2009, Ácala de Henares, Madrid, Espanha. **Proceedings …** 2009, p. 295-298. ISBN: 978-9728924-90-4, IADIS, 2009.

SCWABE, D.; ROSSI G. An object-oriented approach to Web-based application design. Theory and Practice of Object Systems (TAPOS), 1998. **Proceddings ...** p. 207-225.

SOMMERVILE, I. **Software Engineering**. 8.ed. NY, USA: Addison-Wesley, 2006.

STANLEY, M. **CML: A Knowledge Representation Language with Applications to Requirements Modeling**. Thesis (M.Sc.), Department of Computer Science, University of Toronto, Canada, 1986.

TOVAL, A. Requirements reuse for improving information systems security: A practitioner's approach. **Requirements Engineering**, v. 6, 2002, p. 205-219. Springer, London, UK, January 2002.

W3C. **Web Ontology Language Reference**. February 2004.

WIEGERS, Karl. **Software Requirements: Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle**. 2.ed. Redmond, WA, USA: Microsoft Press, 2003.

YU, Eric. Towards modeling and reasoning support for early-phase requirements engineering. In: IEEE International Symposium on Requirements Engineering, 3rd, RE'97, **Proceedings...** 1997a, p. 226-235. Washington, DC, USA IEEE Computer Society.

_____. Why Agent-Oriented Requirement Engineering. In: DUBOIS, E. et al. International Workshop on Requirements Engineering: Foundations for Software Quality, June 16-17, 1997b, Barcelona, Catalonia. **Proceedings…** Presses Universitaires de Namur, Namur, Begium, 1997.

_____. Modeling Strategic Relationships for Process Engineering. 2001a. 1 v. Thesis(PhD), Université Catholique de Louvain, Louvain, Belgium.

_____. Agent orientation as modeling paradigm. **Wirtschaftsinformatik**, v. 43, 2001b. p. 123–132. Vieweg, Wiesbaden, Germany.

YU, Eric, MYLOPOULOS, John. Understanding 'why' in software process modeling. In: International Conference on Software Engineering, 16, 1994, **Proceedings ...** p.159-168. Sorrento, Italy.

ZAVE, Pamela; JACKSON, Michael. Four dark corners of requirements engineering. **ACM Transactions on Software Engineering and Methodology (TOSEM)**, New York, NY, USA, v. 6, n. 1, p.1-30, 1997. DOI= doi.acm.org/10.1145/237432.237434.