Universidade Federal de Pernambuco
Centro de Informática

Pós-Graduação em Ciência da Computação

OOM-NFR: transformando modelos i* em oo-method com base em requisitos não-funcionais

Por

Almir da Silva Moreira Buarque

Dissertação de Mestrado

Recife-PE
Fevereiro/2012

Universidade Federal de Pernambuco
Centro de Informática

Almir da Silva Moreira Buarque

OOM-NFR: transformando modelos i* em oo-method com base em requisitos não-funcionais

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof.º Dr. Jaelson Freire Brelaz de Castro
Co-Orientadora: Prof.ª Dr.ª Fernanda Maria Ribeiro de Alencar

Recife-PE
2012

Catalogação na fonte
Bibliotecária Jane Souto Maior, CRB4-571

Buarque, Almir da Silva Moreira
OOM-NFR: transformando modelos i* em oo-method com base em requisitos não-funcionais / Almir da Silva Moreira Buarque. - Recife: O Autor, 2012.
xiii, 193 p. : il., fig., tab.

Orientador: Jaelson Freire Brelaz de Castro.

Dissertação (mestrado) - Universidade Federal de Pernambuco. CIn, Ciência da Computação, 2012.

Inclui bibliografia e apêndice.

1. Ciência da Computação - Engenharia de software. 2. Engenharia de requisitos. I. Castro, Jaelson Freire Brelaz de (orientador). II. Título.

005.12 CDD (23. ed.) MEI2012 – 058

Dissertação de Mestrado apresentada por **Almir da Silva Moreira Buarque** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"OOM-NFR: transformando modelos i* em oo-method com base em requisitos não-funcionais"**, orientada pelo **Prof. Jaelson Freire Brelaz de Castro** e aprovada pela Banca Examinadora formada pelos professores:

Profa. Carla Taciana Lima Lourenco Silva Schuenemann
Centro de Informática / UFPE

Profa. Maria Lencastre Pinheiro de Menezes Cruz
Escola Politécnica / UPE

Prof. Gilberto Amado Cysneiros Filho
Departamento de Estatística e Informática / UFRPE

Visto e permitida a impressão.
Recife, 27 de fevereiro de 2012

Prof. Nelson Souto Rosa
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Dedico esta Dissertação de Mestrado aos meus pais Adauto e Joserene, à minha esposa Patrícia e a meu filho Vinícius

Agradecimentos

A Deus razão por ter me permitido a vida e existência na Terra.

A Deus por me ter dado a oportunidade de ter uma família e crescer no amor.

A Deus por ter me inserido numa sociedade onde pude me desenvolver e entender um pouco de seus problemas.

A Deus quem me colocou diante de pessoas talentosas com as quais pudesse compartilhar problemas, discuti-los e vislumbrar soluções.

Aos Meus Pais pela educação e amor recebidos.

À minha Amada Esposa Patricia pelo amor e incentivo constante.

Ao meu filhinho recém-nascido Vincícius.

Aos meus queridos e eternos avós, em especial, Vovó Júlia.

Ao meu Orientador Jaelson Castro pela oportunidade de iniciar este mestrado, motivar-me nas pesquisas acadêmicas e participar do conceituado grupo LER de estudos científicos.

À minha co-orientadora Fernanda Alencar por ter me apresentado inicialmente as principais questões referentes a este tema de pesquisa.

Aos amigos do Laboratório LER com os quais vivenciei vários momentos, compartilhando conhecimentos científicos, trabalhos e alegrias.

Ao SERPRO pelo apoio dado através do programa Uniserpro sem o qual não seria possível minha participação e frequência neste programa de pós-graduação.

À Helena Cristina do SERPRO pelo incentivo dado no ingresso e continuidade do curso.

A todos os Professores deste Centro de Informática pelos valiosos conhecimentos repassados; funcionários e técnicos no apoio administrativo e manutenção da infraestrutura de TI.

A Deus por tudo e por todas as bênçãos proporcionadas.

Disse Jesus aos serviçais: "Encham os potes com água".
E os encheram até a borda.
Então lhes disse: "Agora, levem um pouco ao encarregado da festa".
Eles assim fizeram, e o encarregado da festa provou a água que fora transformada em vinho, sem saber de onde este viera, embora o soubessem os serviçais que haviam tirado a água.

João 2:7,8,9

Resumo

Nos últimos anos, o paradigma de desenvolvimento de software dirigido a modelos tem se consolidado e ganho popularidade. Desta forma, um produto de software pode ser obtido através de transformações dos modelos de mais alto nível de abstração para os de mais baixo nível, sendo essa transformação guiada por metamodelos. Por exemplo, requisitos iniciais de um sistema de software podem ser transformados em modelos conceituais deste sistema. Observamos que, tanto a abordagem de modelagem de requisitos i* (iStar) como a linguagem de modelagem conceitual OO-Method (OO-M), têm ganho notoriedade no âmbito acadêmico e industrial. Nesta dissertação, estamos interessados em transformar modelos de requisitos iniciais, descritos em i*, em modelos conceituais especificados em OO-Method. Em particular, investigaremos como os requisitos não-funcionais (NFR) de produto, relacionados a softgoals internos a um ator sistema do i*, poderão ser analisados nas abordagens que transformam modelos de requisitos i* em modelos conceituais descritos em OO-Method. Erros e omissões dos requisitos não-funcionais têm impacto bastante negativo na qualidade do produto final: custos elevados de correção pós-implantação e inviabilização do projeto de software. Esses requisitos estão entre os mais caros e difíceis de corrigir. Nesta dissertação, é apresentada a proposta OOM-NFR, que analisa os softgoals internos a um ator sistema em modelos i* para verificar conflitos, prioridades e satisfação dos mesmos. Além disso, com base nos metamodelos i* de origem e OO-Method de destino, a proposta especifica um processo de transformação contendo regras que convertem uma instância de um modelo i* em outra instância do modelo OO-Method. Com o processo OOM-NFR, o modelo OO-M gerado reflete as prioridades dos softgoals especificados em i*. Para ilustrar e validar a proposta, foi concebido um exemplo de uma aplicação baseada num sistema real, utilizando-se a linguagem de transformação de modelos Query/View/Transformation (QVT) para automatizar e implementar essas transformações.

Palavras-chave: MDD, Modelo i*, OO-Method, Requisitos Não-Funcionais

Abstract

In recent years, the model-driven software development paradigm was consolidated gaining popularity. Thus, guided by metamodels, a software product can be obtained by transformations of the highest level of abstraction´s models to lower level ones. For example, the initial requirements of a software system may be transformed into conceptual models of this system. We observed that both the requirements modeling approach i* (iStar) and the language of OO-Method conceptual modeling has gained notoriety in academy and industry. In this dissertation we are interested in transforming models of initial requirements described in i* into conceptual models specified in OO-Method. In particular, we investigate how the non-functional requirements (NFR) of type product, related to internal softgoals of i* actor system, can be analyzed in the approaches that transform i* requirements models into conceptual models described in OO-Method. Errors and omissions of non-functional requirements have a very negative impact on the final product quality: high costs of correction after deployment and even software project unfeasibility. These requirements are among the most expensive and difficult to correct. In this dissertation, it is presented the OOM-NFR proposal that analyzes internal softgoals of an actor system in i* models to check for conflicts, priorities and satisfaction. Furthermore, based on i* source and OO-Method target metamodels, the proposal specifies a transformation process containing rules that convert an instance of an i* model in another instance of OO-Method model. With the process OOM-NFR, the OO-M model generated reflects the priorities of softgoals specified in i*. To illustrate and validate the aproach, an example of an application based on a real system was created using the model transformation language Query/ View/Transformation (QVT) to automate and implement these transformations.

Key words : MDD, i* Model , OO-Method, Non-Function Requirements, OOM-NFR.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Códigos QVT

Lista de Siglas e Abreviaturas

ATL – ATL Transformation Language
BPMN – Businnes Process Modeling Notation
CIM – Computing Independent Model
CNF – Conjunctive Normal Form
CWM - Common Warehouse Metamodel
DSML - Domain-Specific Modeling Language
GORE – Goal-Oriented Requirements Engineering
KAOS – Keep All Objectives Satisfied
MDD – Model-Driven Development
MDA – Model Driven Architecture
MOF – Meta Object Facility
NFR – Non-Functional Requirement
OCL – Object Constraint Language
OMG – Object Management Group
OO-M – OO-Method
PIM – Platform Independent Model
PSM – Platform Specific Model
QVT – Query/View/Transformation
RNF – Requisito Não-Funcional
RUP – Rational Unified Process
SD – Strategic Dependency
SR – Strategic Rationale
UML – Unified Modeling Language
XML – eXtensible Markup Language
XP – Extreme Programming
XMI –XML Metadata Interchange

CAPÍTULO 1

1. Introdução

Este capítulo apresenta uma visão geral da estrutura desta dissertação, elencando seu escopo, objetivos gerais e específicos, contexto científico em que está inserido seu tema e motivações para a realização deste trabalho de pesquisa.

Atualmente, o desenvolvimento de software dirigido a modelos (Model Driven Software Development), cujo acrônimo é MDD, vem se consolidando como processo de Engenharia de Software. A arquitetura dirigida a modelos (MDA–Model Driven Architecture) (MDA, 2003) definida pela OMG (Object Management Group) (OMG, 1997) especifica três níveis de modelos: Computation Independent Model (CIM) que é equivalente aos modelos de requisitos e negócios; Platform Independent Model (PIM) que é equivalente aos modelos de projetos/conceituais; e Platform Specific Model (PSM) que é voltado à implementação do software numa plataforma específica, considerando detalhes de linguagem de programação, camadas da arquitetura do software, middlewares, etc. Além disso, MDA também recomenda que as transformações entre esses níveis sejam guiadas através dos metamodelos das linguagens fonte e destino.

Ferramentas ou ambientes MDD atuais não contemplam transformações, partindo de modelos CIM (Modelos de requisitos e Modelos de negócios).

Um dos fatores que mais dificulta a inserção de modelos de requisitos em um processo MDD é a distância ou "gap" semântico entre os modelos CIM, que estão no espaço do problema, e os outros modelos PIM e PSM que estão no espaço da solução.

Dentre as diversas abordagens de engenharia de requisitos, a Engenharia de Requisitos Orientada a Metas (*Goal-Oriented Requirements Enginering ou* GORE) vem se destacando principalmente por considerar as intenções e razões dos stakeholders[*]. Ainda, entre as

[*] São pessoas ou organizações que serão afetadas pelo sistema e têm influência, direta ou indireta, sobre os requisitos do sistema – usuários finais, gerentes e outros envolvidos no processo organizacional influenciados pelo sistema, engenheiros (KOTONYA e SOMMERVILLE, 1998)

linguagens GORE, o Framework i* (YU, 1995) vem, nas últimas décadas, ganhando notoriedade no âmbito acadêmico e industrial, pois foca o contexto social onde o sistema está inserido, respondendo questões como: Quais requisitos o sistema deve atender? , Como os requisitos devem ser operacionalizados? Qual a razão da existência dos mesmos? O modelo i*, inclusive, através de softgoals, é capaz de modelar requisitos não-funcionais (RNFs) de um sistema. Entretanto, cabe-se ressaltar que os softgoals i* considerados nesta pesquisa estão relacionados a RNFs de produto (SOMMERVILLE, 1998) (ver seção 2.2) e esses softgoals estão sempre representados internamente ao ator sistema do modelo (ver seção 2.3). Além disso, i* é uma linguagem de modelagem bem consolidada (ALENCAR, 2009) e que conta com um bom suporte de ferramentas (GRAU, 2011).

O OO-Method (PASTOR et al., 2007), abreviado por OO-M, é considerado uma abordagem referência em MDD (ALENCAR, 2010a), pois faz de fato transformações de modelos, gerando uma aplicação completa final, partindo do seu modelo conceitual (PIM). Essa abordagem está num estágio bem maduro e vem sendo utilizada com sucesso no desenvolvimento de software industrial através da ferramenta Computer-Aided Software Engineering (CASE) OLIVANOVA (OLIVANOVA, 2005).

Por serem bem aceitas nas comunidades acadêmicas e industrial respectivamente, foram escolhidos os modelos i* e OO-Method para serem as linguagens fonte (CIM) e alvo (PIM) do processo de transformação proposto nesta pesquisa.

Esta dissertação propõe uma evolução do processo definido em (ALENCAR et al., 2009) para contemplar requisitos não-funcionais. No novo processo proposto, ao se inserir como entrada modelos de requisitos i* com softgoals, estes são analisados através de um pré-processamento (ver OOM-NFR capítulo 4), obtendo-se, ao final, modelos conceituais OO-Method que refletem os esses softgoals e suas prioridades. Nesse pré-processamento, é realizada uma análise de satisfação e resolução de conflitos desses requisitos não-funcionais, possibilitando, na ocorrência de conflitos, decidir quais deles serão efetivamente priorizados para, em seguida, serem transformados num modelo conceitual OO-Method.

É bom novamente frisar que existem inúmeros tipos de requisitos não-funcionais, tais como aqueles relacionados ao processo de desenvolvimento de software, ao produto e a outras restrições externas (SOMMERVILLE, 1998) (ver seção 2.2). Este trabalho foca os requisitos não-funcionais relacionados ao produto de software, tais como segurança, desempenho, acessibilidade, etc., passíveis de serem implementados na solução software do sistema que está sendo desenvolvido.

Alguns elementos do processo original definido em (ALENCAR et al., 2009) foram removidos, novas regras de transformação foram especificadas, bem como outras foram adicionadas ao processo transformativo. As regras especificadas podem ser implementadas através de linguagens de transformações de modelos tais como Query/View/Transformation (QVT), ATLAS Transformation Language (ATL) e EPSILON que são projetos do Eclipse (ECLIPSE, 2009) ou através de UML Profile (UML, 2000). Dentre essas linguagens optou-se por implementar as regras de transformação em QVT (QVT , 2009) por ser um padrão OMG bem difundido, com ambiente de desenvolvimento na plataforma livre Eclipse e por prover recursos suficientes para manipular metamodelos e transformações. Esta pesquisa envolve ainda a utilização de um sistema de informação bem conhecido (sistema de agência fotográfica) modelado em i*, visando exemplificar o processo proposto e as transformações. Além disso, um exemplo de aplicação do processo a um caso real (sistema de controle de acesso) será apresentado no capítulo 5, objetivando melhor analisar e validar o processo transformativo.

1.1 Motivação

Pesquisas confirmaram o crescente reconhecimento da Engenharia de Requisitos (ER) como uma das áreas de maior importância da Engenharia de Software (LAMSWEERDE, 2000). Isso está associado ao fato da Engenharia de Requisitos ser a base para todo o processo de Engenharia de Software. Erros, falhas e inconsistências nesta fase do processo têm grandes impactos na qualidade, viabilidade, custo e prazo do produto final. Em (LAMSWEERDE, 2000) é realizado um estudo dos processos da Engenharia de Requisitos, focando-se em modelos, por se considerar a modelagem a principal atividade (o denominador comum) da ER. Ainda em (LAMSWEERDE, 2000), faz-se uma retrospectiva e uma análise dos principais modelos que surgiram nas últimas décadas, enfatizando a relevância da modelagem organizacional baseada em metas (goals) e citando tendências na área da Engenharia de Requisitos. Neste contexto, onde modelagem representa uma atividade essencial da engenharia de software, o Desenvolvimento de Software Dirigido a Modelos (Model Driven Software Development), tem um importante papel.

Os principais argumentos para a utilização de um processo de desenvolvimento de software dirigido a modelos são: maior produtividade, portabilidade, interoperabilidade, menor custo, esforço, prazo, mais facilidade na evolução do software e maior qualidade do produto.

A principal idéia em MDD é a transformação de modelos de maiores níveis de abstração (domínio do problema) em modelos mais concretos (domínio da solução) até se obter, por fim, o código do sistema. O paradigma MDD preconiza que o desenvolvimento inicial e modificações futuras da aplicação sejam efetuados apenas no modelo mais abstrato. A automação de um processo de desenvolvimento de software MDD é facilitada ao se adotar o padrão (MDA) e ter disponibilidade de diversas tecnologias de transformação de modelos. Entretanto, é importante deixar claro que MDD não é sinônimo ou igual ao padrão (processo) MDA (ver seção 3.1 do capítulo 3). Segundo (KELLY, 2008) , processos MDD com base em MDA utilizam apenas o padrão UML, mas existem vários outros processos MDD que são baseados em Domain-Specific Modeling Languages (DSMLs). Esta pesquisa tem como escopo duas DSMLs (i* e OO-Method) que **não são** baseadas no padrão MDA (UML). Assim, o processo proposto nesta pesquisa é um processo MDD baseado em DSMLs.

Em processos MDD automatizados, o modelo abstrato (de maior nível de abstração) do sistema deve representar com precisão o código, ou seja, ele deve ser executável e ter uma equivalência funcional com todos os outros modelos mais concretos (PASTOR et al., 2007). Dessa forma, as modificações no modelo de mais alto nível de abstração são refletidas automaticamente nos modelos de mais baixo nível. Isso torna a atividade de modelar no nível mais abstrato, o centro de todo processo de desenvolvimento do software. E dispensa completamente, nos melhores ambientes MDD, a execução de atividades manuais nos modelos de mais baixo nível de abstração (projeto e implementação). Entretanto, a maioria desses processos não considera os modelos de requisitos, seja em UML (caso de uso), i*, modelos formais, etc. Somente a transformação em si, partindo de modelos de requisitos em i*, já é um grande desafio, pois ainda é uma questão aberta de pesquisa o relacionamento ou interoperabilidade entre modelos intencionais descritos em i* e modelos conceituais de nível projeto (caso do OO-M) que descrevem os aspectos estruturais, comportamentais, funcionais ou operacionais de um sistema.

Diversas abordagens já foram propostas, visando transformar um modelo de requisitos i* em OO-Method. Contudo, o sucesso foi limitado. Um dos fatores que mais dificulta a integração entre esses modelos é a distância ou "gap" semântico entre eles, devido a grande diferença dos níveis de abstração em que eles se situam. De fato, o i* está no nível de requisitos enquanto o modelo conceitual (ver seção 3.2) do OO-Method está num nível de projeto. Alguns processos para transformação de i* para OO-Method foram definidos em trabalhos (ver seção 3.3) como: "Conceptual schemas generation from organizational models

in a automatic software production process" (MARTINEZ, 2008), "From i* Requirements Models to Conceptual Models of a Model Driven Development Process" (ALENCAR et al., 2009) e em "Supporting Automatic Interoperability in Model-Driven Development Processes" (GIACHETTI, 2011). É importante destacar que em (ALENCAR et al., 2009) é definido um processo transformativo bem-sucedido no qual se baseia o trabalho de (GIACHETTI, 2011). Entretanto, esses processos abordam apenas os requisitos funcionais dos sistemas e ignoram ou não comtemplam os softgoals. Como se sabe, esses softgoals representam características que impactam diretamente na qualidade do software e não podem ser ignorados.

Assim, diante desses desafios, da relevância atual do tema, importância dos RNFs e da forte crença da comunidade acadêmica de que desenvolvimento dirigido a modelos (MDD) será, num breve futuro, uma das principais formas de Engenharia ou Desenvolvimento de Software, foram as principais razões que motivaram o desenvolvimento desta dissertação.

1.2 Definição do Problema

A transformação de modelos de requisitos, independente do modelo de origem escolhido, em modelos de menor nível de abstração (modelos análise ou projeto) é uma atividade complexa e com muitas questões de pesquisa ainda em aberto. A maioria dos processos, ambientes e ferramentas existentes começam o desenvolvimento do software a partir do modelo PIM, que é equivalente à fase de projeto, não contemplando praticamente modelos CIM (Requisitos ou Negócios). Um dos fatores que mais dificulta a inserção de modelos de requisitos em qualquer processo transformativo é a distância ou (gap) semântico entre esse modelos CIM e PIM, causando problemas de interoperabildade entre estes.

Como já descrito na seção anterior, os processos MDD existentes e que transformam modelos de requisitos modelos i* em modelos OO-M, não consideram os softgoals (requisitos não-funcionais de produto internos ao ator sistema) durante a transformação. Esse problema traz alguns impactos negativos como:

- Os Modelos gerados em OO-Method não refletem os RNFs (softgoals) especificados em i*;
- É impossível afirmar, dentre os modelos OO-M gerados, quais RNFs especificados em i* foram priorizados (satisfeitos) e estão sem conflitos;

- Os Softgoals, que são elementos significativos do i* são ignorados, perdendo-se um pouco da capacidade de raciocínio e de análise inerentes aos modelos i*.
- Impactos negativos na qualidade do produto de software, já que os RNFs estão associados a atributos de qualidade do sistema.

Como visto, um dos problemas com esses processos é que eles são insensíveis a esses requisitos não-funcionais e geram, como saída, um modelo OO-Method a partir do qual não se pode afirmar quais aspectos de qualidade (ex: segurança, desempenho, acessibilidade, etc.) estão sendo priorizados e sem conflitos.

É nesse contexto que está inserido este trabalho, propondo um processo transformativo que considera os requisitos não-funcionais descritos nos modelos i* e gerando um modelo conceitual (PIM) em OO-Method que reflita esses RNFs (ver OOM-NFR capítulo 4).

Sem a análise e tratamento dos RNFs, seria gerado um modelo OO-M com as classes contendo todos os serviços (ver seção 4.4) transformados a partir das tarefas i*, sem saber se esses serviços (transações) estariam se conflitando com os softgoals ou não os priorizando.

Além disso, o processo transformativo proposto é automatizado na linguagem de transformação de modelos QVT, tornando a atividade de gerar um modelo OO-M a partir do i* mais rápida e menos propensa a erros humanos.

Os objetivos que nortearam o desenvolvimento desta dissertação foram guiados pela necessidade de se considerar softgoals em i*, no processo de transformação MDD guiado por regras (diretrizes) e metamodelos (ver OOM-NFR capítulo 4). Os objetivos e a metodologia utilizada para alcançá-los são descritos nas próximas seções.

1.3 Objetivos

O principal objetivo deste trabalho é propor um processo MDD de transformação de i* para o modelo objeto (classe) do OO-Method (ver seção 3.2) com base em requisitos não-funcionais. O foco em requisitos não-funcionais se dá pelo fato deles serem imprescindíveis como atributos de qualidade e não poderem ser negligenciados num processo de desenvolvimento de software.

Para alcançar o objetivo principal, foram traçados alguns objetivos específicos:

- Abordar e analisar a engenharia de requisitos orientada a metas, focando na abordagem i* wiki (GRAU et al., 2008) e softgoals.

- Descrever o desenvolvimento orientado a modelos (MDD), conforme padrão MDA e também a partir das DSMLs (i* e OO-Method), bem como analisar os processos existentes que realizam transformações da abordagem i* para o modelo objeto do OO-Method.
- Definir o processo OOM-NFR que transforma i* para o modelo de classe OO-Method, considerando e tratando softgoals relacionados a requisitos não-funcionais de produto.
- Especificar e apresentar um exemplo de aplicação para o processo OOM-NFR.
- Apresentar processo OOM-NFR, automatizando-o na linguagem de transformação de modelos QVT.

1.4 Metodologia

Dentre as diversas opções metodológicas utilizadas e atividades realizadas na construção do raciocínio lógico e estrutura deste projeto de pesquisa, cabe-se mencionar:

- Pesquisa de referências bibliográficas correlatas ao tema: Nesta fase, visando adquirir conhecimento e aprofundar o estudo, foi realizada uma extensa revisão bibliográfica sobre Requisitos, Processos de desenvolvimento orientado a Modelos, Qualidade de Software, Modelagem de Processos, e Tecnologias de Transformação de Modelos utilizando Metamodelagem.
- Análise do os modelos i* e OO-Method: Foram analisados os modelos existentes de i* Original (YU, 1995), i* wiki (GRAU et al., 2008), KAOS (LAMSWEERDE, 2000) e o Framework NFR (CHUNG et al., 1999). Por ser um padrão adotado pela comunidade, o i* wiki, é o utilizado nesta dissertação como modelo de partida no processo transformativo. Como modelo de destino, considera-se o modelo OO-Method (PASTOR at al., 2007) que é referência em MDD e implementado na ferramenta OLIVANOVA (OLIVANOVA, 2010).
- Análise comparativa dos processos de transformação de i* para OO-Method: Vários outros processos de transformação de i* para OO-Method foram analisados. Dentre os estudados, destacam-se três mais significativos e que estão expostos na seção 3.4.1. Nesta fase, foi identificada a necessidade de se considerar softgoals (requisitos não-funcionais de produto) no processo de transformação.

- Planejamento da proposta OOM-NFR: Para contemplar RNFs no processo de transformação foi idealizada uma proposta tendo por base a transformação inicial proposta em (ALENCAR et al., 2009). A nova proposta foi denominada OOM-NFR, originada pela fusão da sigla do OO-Method com a sigla NFR (Non-Functional Requirement). O processo proposto exige que os modelos iniciais de entrada contenham requisitos não-funcionais (softgoals). Uma análise inicial dos RNFs deve ser realizada antes da geração do modelo OO-Method.
- Automatização da proposta OOM-NFR: Para automatizar as atividades do processo transformativo proposto (processo núcleo), algumas tecnologias de transformação com base em metamodelos foram analisadas: ATL, QVT, EPSILON, UML Profile. Dentre essas linguagens, optou-se em automatizar (implementar) o processo núcleo (ver seção 4.1) do OOM-NFR em QVT, em função de ser um padrão OMG bem difundido, possuir suporte da plataforma de desenvolvimento livre Eclipse e prover recursos suficientes para manipular metamodelos e transformações.
- Validação da proposta: Para validar e exemplificar o processo transformativo proposto foi selecionado e especificado, em i* wiki, um um sistema de controle de acesso a sistemas de informação desenvolvido pelo Serpro - Serviço Federal de Processamento de Dados, onde RNFs como segurança, desempenho e usabilidade são prioritários.
- Realização de análise conclusiva: Por fim, para melhor avaliar a proposta e retirar lições aprendidas, contribuições e limitações, foi realizada uma análise conclusiva da aplicação do processo OOM-NFR ao exemplo de aplicação apresentado.

1.5 Organização do Trabalho

Além deste Capítulo introdutório, que contextualiza e descreve como a pesquisa foi conduzida, este trabalho está estruturado em mais 5 capítulos, conforme descrito a seguir:

- Capítulo 2. Engenharia de Requisitos Orientada a Metas. É dada uma visão geral da abordagem Goal-Oriented Requirements Engineering (GORE) i* Wiki.

- Capítulo 3. Desenvolvimento Orientado a Modelos. Faz-se uma revisão de literatura sobre a arquitetura MDA e a abordagem MDD OO-Method.

- Capítulo 4. Transformando i* para OO-Method com NFR. Neste capítulo é proposto o processo OOM-NFR.
- Capítulo 5. Exemplo de Aplicação. Através da aplicação em um sistema de controle de acesso modelado em i*, é validado e exemplificado o processo OOM-NFR.
- Capítulo 6. Conclusões e Trabalhos Futuros. Neste capítulo finaliza-se o trabalho, elencando as principais contribuições, limitações, lições aprendidas e trabalhos futuros.

CAPÍTULO 2

2. Engenharia de Requisitos Orientada a Metas

Neste capítulo, apresenta-se uma visão geral da Engenharia de Requisitos, particularmente, a Engenharia de Requisitos Orientada a Metas (GORE). Serão abordados tópicos sobre modelos i*, Requisitos não-funcionais (NFRs), Tratamento de Conflitos e metamodelo i*.

2.1 Engenharia de Requisitos

Nesta seção será visto como é o processo da Engenharia de Requisitos, apresentando suas atividades básicas de elicitação, análise e negociação, documentação e validação de requisitos. Essas atividades são complementadas durante todo ciclo de vida do software com a atividade de gerenciamento de requisitos que visa principalmente manter atualizada a documentação desses requisitos, analisar impactos causados pelas sucessivas mudanças, etc. Geralmente os requisitos podem ser classificados em requisitos funcionais e requisitos não-funcionais. Como o trabalho presente propõe um projeto de transformação MDD que considera os requisitos não-funcionais de um sistema, foi reservada a seção 2.2 para detalhar a terminologia, natureza, importância, classificação e abordagens que tratam esses requisitos.

Antes de se abordar os tópicos mais específicos sobre o tema proposto neste trabalho, é importante se ter uma visão geral sobre o contexto da Engenharia de Requisitos como subprocesso da Engenharia de Software. Tomando-se, como referência base, o ciclo de vida walterfall, ilustrado na figura 1, percebe-se que a Engenharia de Requisitos é o processo inicial.

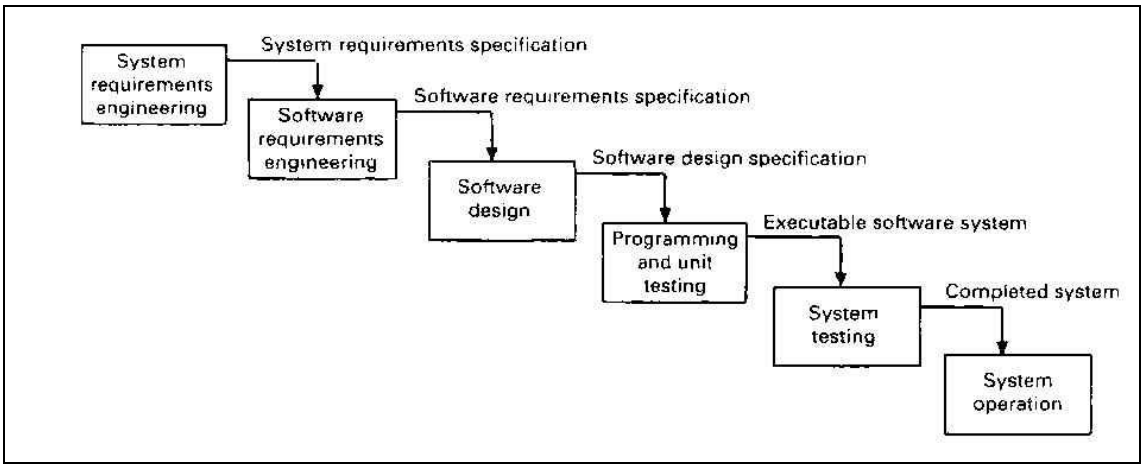


Figura 1. Modelo Cascata (KONTONYA, 1998)

O Modelo Processo de Engenharia de Requisitos de Alto Nível proposto por (KONTONYA, 1998) desdobra essa primeira fase do processo geral da engenharia de software em outros processos, conforme mostrado na figura 2, modificada da figura original, para destacar exemplos de modelos de sistemas elaborados no processo de documentação (especificação) de requisitos.

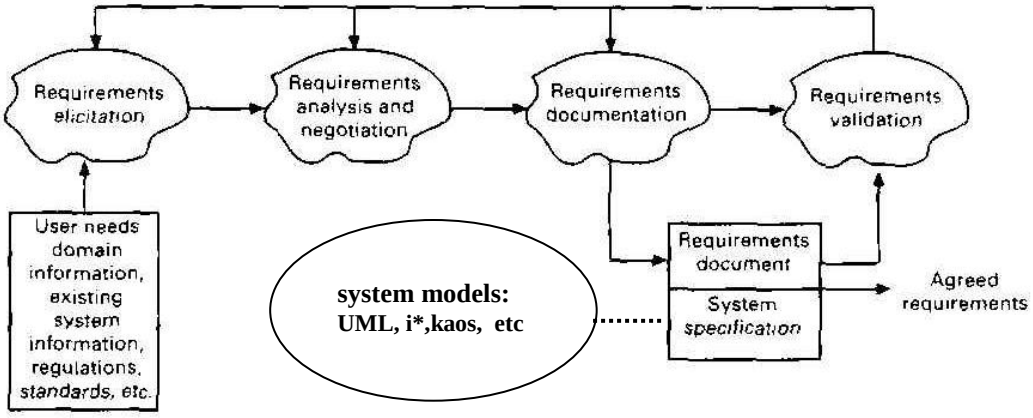


Figura 2. Processo de Requisitos de Alto Nível (KONTONYA, 1998)

De acordo com a figura 2, as atividades desse processo são:

- **Elicitação**: os requisitos do sistema são descobertos através:
    - da consulta aos stakeholders;
    - a partir de documentos;
    - do conhecimento do domínio de aplicação;
    - dos estudos de mercado;
    - das informações de sistemas existentes;
    - das necessidades das partes envolvidas;

- dos padrões organizacionais;
- da regulamentação;

- **Análise e Negociação**: Requisitos são analisados e os conflitos são resolvidos através de negociação.
- **Documentação dos Requisitos**: um documento de requisitos é especificado.

  Esta atividade é fundamental para o entendimento deste trabalho, pois, nesse subprocesso, são elaborados os modelos de mais alto nível de abstração dos sistemas (softwares) a serem desenvolvidos. Assim, na figura 2, inseriu-se uma elipse central, associada à especificação de sistemas, visando salientar esses modelos (system models), instanciados para UML (nível requisitos), i* ou KAOS (LAMSWEERDE, 2000), mas que poderia ser instanciado para qualquer outro modelo existente no nível de requisitos, como: análise estruturada, análise de requisitos orientado a aspectos, BPMN, modelos formais (Z,VDM, SCR), Problem Frame (JACKSON, 1995), View points (NUSEIBEH, 1994), etc. Na seção 2.3 deste capítulo, será detalhado o modelo de requisitos i* Wiki, uma importante abordagem de Engenharia de Requisitos Orientada a Metas e que será utilizada como origem para o processo de transformação proposto nesta dissertação.
- **Validação**: É checada a consistência e a completude do documento de requisitos ou modelo(s) de requisitos.

  Esta atividade é também muito importante no contexto deste estudo, pois quando se tem um modelo de requisitos bem definido e preciso semanticamente, fica mais fácil automatizar esse processo de validação em um ambiente MDD.

Paralelamente a essas atividades, durante todo ciclo de vida do software, ocorre a atividade de gerenciamento de requisitos, pois o software evolui e seu modelo inicial de requisitos varia com o decorrer do tempo. Além disso, o gerenciamento de requisitos se preocupa com a rastreabilidade dos mesmos para se obter informações diversas, que vão desde a origem de um determinado requisito até todos outros elementos com os quais ele está relacionado (tanto no mesmo nível quanto nos diferentes níveis de abstração).

Também essa atividade de gerenciamento de requisitos é bastante facilitada quando se tem modelos de requisitos bem definidos e regras precisas de mapeamento com outros modelos de mais baixo nível de abstração (“design” e implementação) como orienta a tecnologia MDD.

Ainda, segundo (KONTONYA, 1998), não há um modelo (processo) de requisitos ideal, que seja adequado para todos os fins e tipos de software. Cada abordagem ou modelo adotado tem seus pontos fortes para deterninado tipo de aplicação ou aspecto específico que se queira modelar. Um dos pontos fortes do processo proposto neste trabalho é analisar softgoals em i* wiki e gerar, automaticamente, de diferentes modelos em OO-Method que sejam reflexos desses softgoals e suas prioridades (ver seção 6.1 contribuições).

2.2 Requisitos Não Funcionais-RNF

Esta seção fará uma explanação sobre requisitos não-funcionais, abordando sua terminologia, importância, classificação e abordagens de tratamento.

O termo requisito não-funcional (RNF) foi utilizado inicialmente na literatura por (ROMA, 1985), referindo-se às restrições a que um sistema de software deve atender. Requisitos não-funcionais são comumente referenciados como atributos de qualidade (BOEHM, 1978), requisitos extra-funcionais (KELLER, 1990) e requisitos não-comportamentais (DAVIS, 1993). Atualmente o termo requisito não-funcional é o mais difundido principalmente quando se quer diferenciar dos requisitos funcionais. Portanto, requisito não-funcional será o termo adotado neste trabalho. Entretanto, o conceito de requisito não-funcional é mais amplo do que o conceito de softgoal do i* wiki como será visto mais adiante nesta seção e na próxima seção 2.3.

A complexidade de um sistema de software é determinada parcialmente por suas funcionalidades (requisitos funcionais) que representam o que realmente o sistema faz e também parcialmente pelos requisitos não-funcionais que representam os requisitos globais do desenvolvimento (CHUNG et al., 1999). Estes requisitos não-funcionais desempenham um papel crítico durante o desenvolvimento de um sistema, servindo de critérios de seleção para se escolher entre uma míriade de alternativas de projetos e implementações (CHUNG et al., 1999). Sabe-se que erros e omissões desses requisitos não-funcionais têm impacto bastante negativo na qualidade do produto final, acarretando custos elevados na correção dos mesmos, após o sistema estar implantado, ou mesmo, chegando a inviabilizar a utilização do software

concebido. Estudos apontam estes requisitos como estando entre os mais caros e difíceis de corrigir (DAVIS, 1999), (CYSNEIROS, 1999).

Além disso, o apelo de gerentes, engenheiros e usuários por software que seja melhor, barato, rápido, seguro e amigável, ilustra essa necessidade crescente de se considerar e lidar compreensivamente com esses requisitos não-funcionais durante o processo de desenvolvimento do software (CHUNG et al., 1999).

Mas quais são, na prática, estes requisitos não-funcionais? Como são classificados? O que realmente os diferenciam dos requisitos funcionais? Quais as abordagens utilizadas durante um processo de desenvolvimento de software que focam a especificação (documentação) destes requisitos?

As respostas a estas indagações são várias e não há um consenso geral entre elas. Requisitos funcionais descrevem as funções que o sistema deve fazer, ou seja, suas funcionalidades específicas que, ao final do processo de desenvolvimento do software, estarão implementadas numa linguagem de programação específica, sendo disponibilizada para os usuários finais. Por isso, os requisitos funcionais têm um efeito localizado e específico. Alguns exemplos de requisitos funcionais de um suposto sistema de controle de bibliotecas são mostrados a seguir:

- O sistema deve manter (cadastrar, excluir, alterar) os livros;
- O sistema deve gerenciar empréstimos, reservas e devoluções de livros;
- O sistema deve prover consultas por título, autor, assunto de livros.

Requisitos não-funcionais geralmente fixam restrições sobre como os requisitos funcionais serão desenvolvidos e estão relacionados com os aspectos de qualidade. Como exemplo, para o sistema de controle de bibliotecas anterior, poder-se-ia definir os alguns requisitos não-funcionais tais como:

- O sistema deve ser fácil de usar para quaisquer usuários, inclusive deficientes visuais e auditivos, atendendo a padrões W3C de acessibilidade;
- O sistema deve ser seguro, utilizando login via certificado digital e ou dados biométricos;
- O sistema deve ser rápido e ter um bom desempenho;
- O sistema deve atender ao padrão internacional ISBN;
- O sistema deve ser desenvolvido utilizando um processo MDD automatizado, de preferência, um que seja livre “Framework Deimoiselle”;
- O sistema deve estar disponível no padrão WEB;

- A base de dados do sistema deve ser armazenada em Oracle 11g;
- O sistema não deve ultrapassar o tamanho de 1000 pontos de função (PF);
- O sistema não deve ter um esforço de desenvolvimento maior que 1000 Homens-dia (HD);
- O sistema não deve ultrapassar o custo de 1 milhão de reais;

Dependendo da imaginação e exigência de clientes, gerentes, desenvolvedores e usuários, além desses, vários outros requisitos não-funcionais poderiam ser especificados. Fazendo uma rápida análise desses requisitos, observa-se que:

- Alguns deles estão vagamente especificados;
- Outros estão bem especificados, inclusive com métricas bem definidas;
- Alguns podem ser viáveis outros podem ser inviáveis tecnologicamente ou financeiramente;
- Alguns estão intrisicamente relacionados (interdependência);
- Alguns são conflitantes (qualidade x custo, segurança e desempenho);
- Alguns se confundem inclusive com requisitos funcionais;
- A maioria deles (Logs, Usabilidade, Segurança, etc) permeia e está intrisicamente disperso por vários requisitos funcionais;

Assim, não existe uma definição formal unânime ou uma lista completa de requisitos não-funcionais. O padrão IEEE-Std 830 1998 classifica os RNFs, propondo uma lista de 13 requisitos não-funcionais como ilustra a figura 3 a seguir:

Requisitos Específicos
Requisitos funcionais
Requisitos de performance
Requisitos de Interface
Requisitos Operacionais
Requisitos de Recursos
Requisitos de Verificação
Requisitos de Aceitação
Requisitos de Documentação
Requisitos de Segurança (security)
Requisitos de Portabilidade
Requisitos de Qualidade
Requisitos de Confiabilidade
Requisitos de de Manutenibilidade
Requisitos de *Safety*

Figura 3. IEEE-Std 830 1998 RNFs

Em (CHUNG et al., 1999) são definidos catálogos que organizam os RNFs em hierarquias. Segurança, Desempenho e Precisão "Accuracy" são alguns dos RNFs definidos nesses catálogos.

Outra proposta interessante no sentido de classificação dos RNFs é encontrada em (SOMMERVILLE, 1998), onde os RNFs podem ser categorizados em: requisitos de processo, requisitos do produto e externos, conforme ilustrado na figura 4 a seguir.

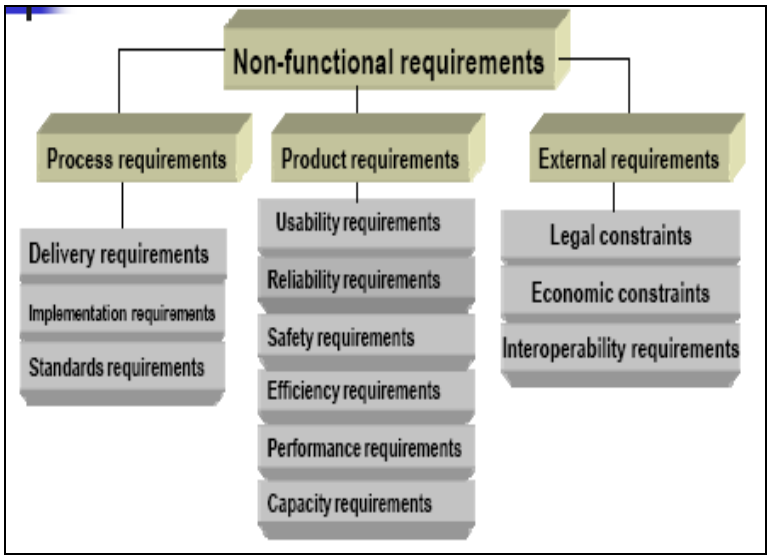


Figura 4. Classificação dos Requisisto Não-Funcionais (SOMMERVILLE, 1998)

Os RNFs de Produto especificam as características desejadas a que um produto deve ter e atender:

- O sistema deve ter disponibilidade maior que 99%;
- O sistema deve processar, no mínimo, 100 transações por segundo;
- O sistema deve fornecer um "help online" para facilitar seu uso.

Os RNFs de Processo especificam restrições relacionadas ao processo de desenvolvimento do sistema:

- O sistema deve ser desenvolvido conforme normas ISO;
- O sistema deve ser desenvolvido utilizando processo ágil XP;
- O sistema deve ser desenvolvido utilizando processo RUP;
- O sistema deve utilizar processso MDD automatizado e implementado na plataforma destino Microsoft .NET.

Já os RNFs Externos são derivados do ambiente externo em que o sistema está inserido e representam restrições legais, econômicas e de interoperabilidade as quais o sistema deva atender:

- Custo do sistema não deve ultrapassar 500.000 mil reais;
- Sistema deve vir com contrato de manutenção periódico;
- Sistema deve atender às legislações Trabalhistas, Tributárias e Ambientais;
- Sistema deve interoperar com outros sistemas através do padrão Web Services;
- Sistema deve ter interfaces externas com os Sistemas: Financeiro Nacional, Tributário Municipal, Fazendário Estadual, Justiça Federal, SPC e Justiça Eleitoral.

Esta pesquisa propõe transformações MDD focando os RNFs de produto segundo esta classificação, uma vez que estes são mais factíveis de terem uma solução software e mais facilmente implementáveis (operacionalização). Sendo assim, o processo proposto OOM-NFR (ver capítulo 4) focará apenas softgoals do modelo i* internos ao ator sistema, uma vez que esses softgoals podem ser considerados de produto e passíveis de operacionalização.

Entretanto, o processo OOM-NFR pode também contemplar e ser estendido para analisar RNFs externos, desde que estes sejam modelados internamente ao ator sistema em i* wiki e sejam operacionalizáveis. Por exemplo, caso o ator sistema de software modelado, tenha um softgoal que é ter um usuário validado em outro ator sistema externo (Sistema Eleitoral, Sistema Tributário, Sistema Judiciário, etc) e essa validação é operacionalizada via tarefas em i*. Nesse caso, esse softgoal, que está relacionado a um RNF do tipo externo, vai também poder ser analisado pelo OOM-NFR. Já RNFs externos tais como: "Sistema deve vir com contrato de manutenção periódico", "Custo do sistema não deve ultrapassar 500.000 mil reais" e outros mais, são praticamente infactíveis de serem operacionalizáveis via tarefas do i*, sendo assim, não podem ser contemplados pelo processo proposto nesta pesquisa (OOM-NFR). Então, doravante, sempre que aparecer, no texto dessa dissertação, o termo requisito não-funcional ou RN, este deve ser entendido que se trata de um requisito não-funcional de produto e que está associado a um softgoal i* interno a um ator sistema de software passível de operacionalização. Do mesmo modo, quando for utilizado o termo softgoal, entende-se que seja um softgoal interno ao ator sistema, relacionado a um RNF de produto e operacionalizável.

Um trabalho bastante atual e interessante sobre RNFs é apresentado em (MAIRIZA et al., 2010), onde, pela primeira vez, se faz um estudo bastante extenso sobre esses requisitos,

tratando-os em três dimensões: terminologia, tipos e mapeamento desses RNFs mais freqüentes em relação aos tipos de sistemas e de domínios de aplicação mais comuns.

A figura 5 mostra os termos e entendimentos mais comuns utilizados no meio acadêmico e indústria quando se refere a requisitos não-funcionais.

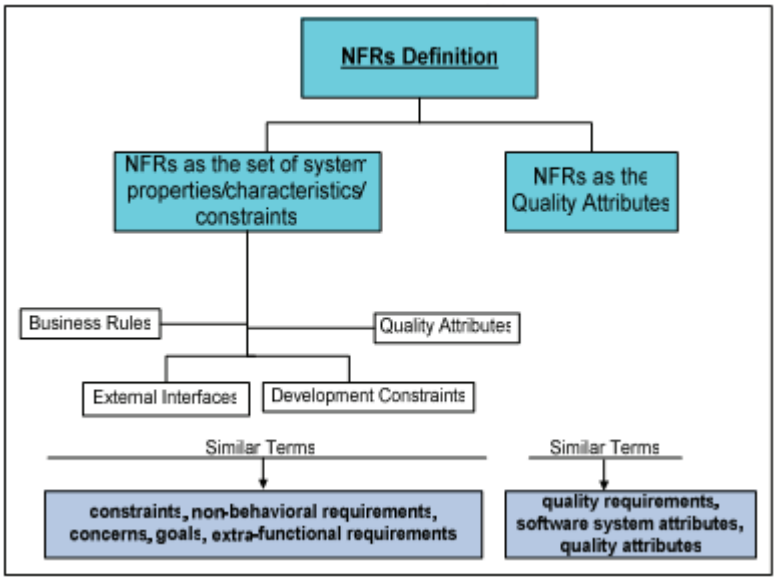


Figura 5. Definição e Terminologia NFR (MAIRIZA et al., 2010)

Ainda em (MAIRIZA, et al., 2010) faz-se um estudo estatístico sobre o estado de definição (conceitual) de 114 RNFs (Figura 6). Observa-se que muitos dos requisitos não funcionais estão sem definição e atributos (métricas).

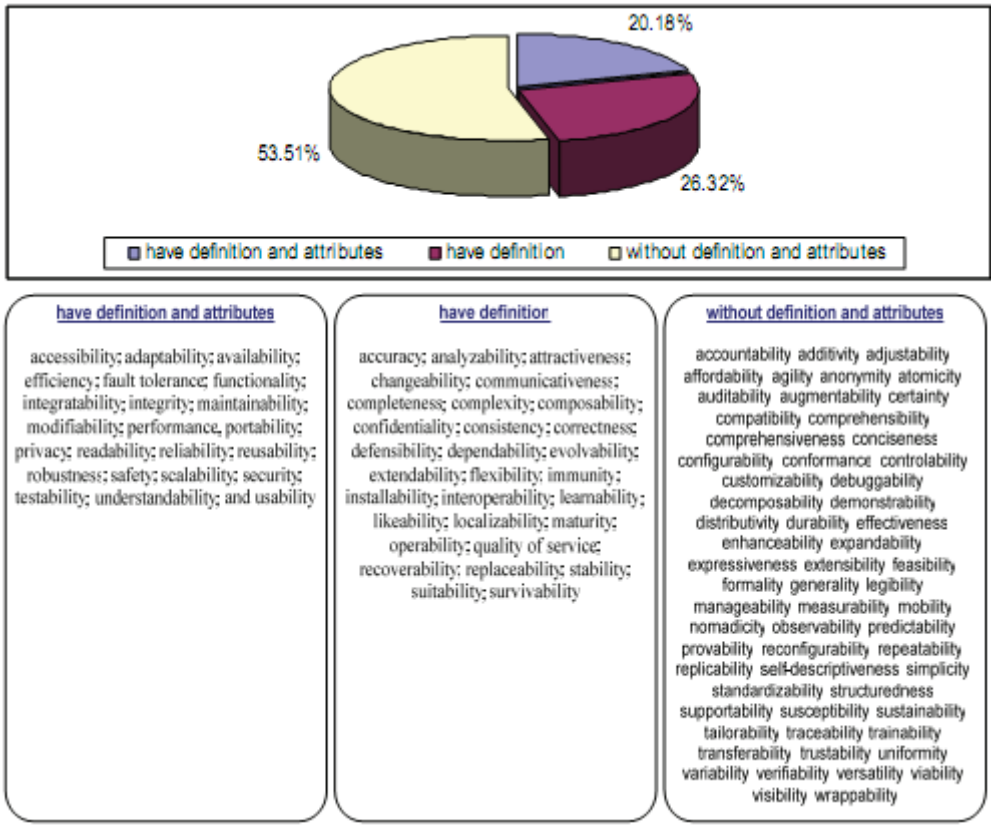
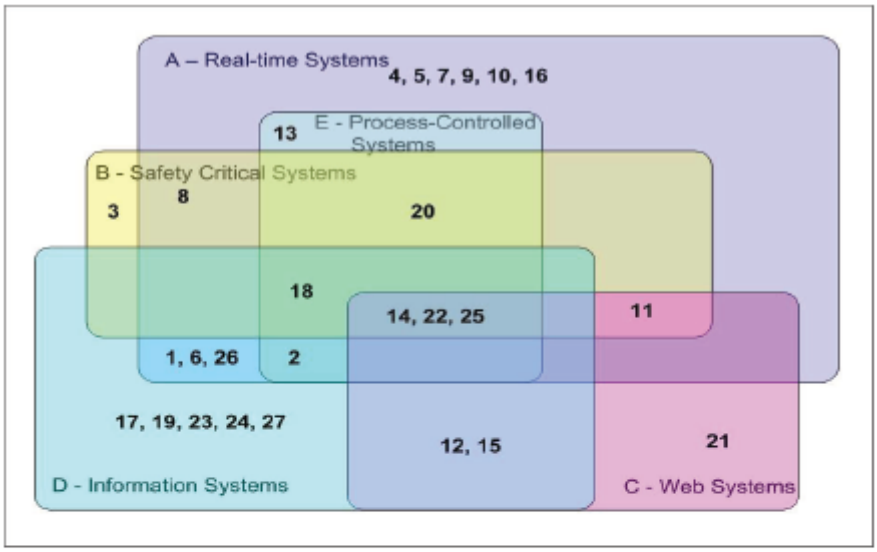


Figura 6. Definição do NFRs e Atributos (MAIRIZA, et al., 2010)

Nesse trabalho é revelado quais são os três mais importantes requisitos não- funcionais por tipo de sistema (14-desempenho, 22-segurança e 25-usabilidade), conforme ilustrado no centro da figura 7.



Legend:

1 Accuracy	10 Installability	19 Reusability
2 Availability	11 Integrity	20 Safety
3 Communicativeness	12 Interoperability	21 Scalability
4 Compatibility	13 Maintainability	22 Security
5 Completeness	14 Performance	23 Standardizability
6 Confidentiality	15 Privacy	24 Traceability
7 Conformance	16 Portability	25 Usability
8 Dependability	17 Provability	26 Verifiability
9 Extensibility	18 Reliability	27 Viability

Figura 7. Tipos de Sistemas e NFRs mais Relevantes (MAIRIZA, et al., 2010)

Por fim, a pesquisa enumera os 5 (cinco) mais importantes (do inglês " top 5") RNFs quando estes são mapeados com os domínios de aplicação, segundo figura 8.

NFRs	Definition	Attributes
Performance	requirements that specify the capability of software product to provide appropriate performance relative to the amount of resources needed to perform full functionality under stated conditions	response time, space, capacity, latency, throughput, computation, execution speed, transit delay, workload, resource utilization, memory usage, accuracy, efficiency compliance, modes, delay, miss rates, data loss, concurrent transaction processing
Reliability	requirements that specify the capability of software product to operates without failure and maintains a specified level of performance when used under specified normal conditions during a given time period	completeness, accuracy, consistency, availability, integrity, correctness, maturity, fault tolerance, recoverability, reliability, compliance, failure rate/critical failure
Usability	requirements that specify the end-user-interactions with the system and the effort required to learn, operate, prepare input, and interpret the output of the system	learnability, understandability, operability, attractiveness, usability compliance, ease of use, human engineering, user friendliness, memorability, efficiency, user productivity, usefulness, likeability, user reaction time
Security	requirements that concern about preventing unauthorized access to the system, programs, and data	confidentiality, integrity, availability, access control, authentication
Maintainability	requirements that describe the capability of the software product to be modified that may include correcting a defect or make an improvement or change in the software	testability, understandability, modifiability, analyzability, changeability, stability, maintainability compliance

Figura 8. Os 5 RNFs mais comuns, suas definições e atributos (MAIRIZA, et al., 2010)

Assim, esta proposta de dissertação considerará, como RNFs (softgoals) prioritários no processo de transformação MDD, os requisitos de segurança, desempenho e usabilidade, pois esses são considerados os mais importantes, segundo (MAIRIZA, et al., 2010).

Como visto, há vários problemas enfrentados para especificar ou expressar os RNFs, por exemplo:

- Restrições são desconhecidas no estágio de requisitos, só sendo especificadas detalhadamente na fase de soluções do projeto do sistema;
- RNFs, normalmente, estão interrelacionados com outros RNFs e com os requisitos funcionais (RFs);
- Regras que determinem claramente quando um NFR estará completamente atendido inexistem;
- RNFs são conflitantes entre si;

Além disso, existem poucos métodos ou abordagens que tratam requisitos não-funcionais de forma efetiva. A maioria dos métodos de Engenharia de Requisitos baseia-se na análise funcional ou orientada a objetos voltada aos requisitos funcionais. A principal razão para isso é porque, geralmente, os RNFs são diversos, dependentes de domínio, além de difíceis de serem especificados conforme exemplos anteriores citados.

Para tratar RNFs, existem na literatura dois tipos de abordagens:

- Abordagens orientadas a processos que integram o esforço de descrever e atender RNFs durante o processo de desenvolvimento;
- Abordagens orientadas a produto que avaliam o grau a que o produto final atende a determinados RNFs.

Na abordagem orientada a produtos o sistema é avaliado pelo grau que ele atende a determindados RNFs, através de um processo de controle de qualidade capaz de fazer tais verificações. Para isso, esse tipo de abordagem propõe-se o uso de métricas que aferem o nível de qualidade do sistema. Existem várias propostas na literatura sobre essa sistemática, inclusive com aplicabilidade das normas ISO sobre controle de qualidade relacionado ao produto. A maior parte dos trabalhos que abordam RNFs faz o controle orientado ao produto, ou seja, se situa no campo da quantificação do grau de conformidade de um software para com os RNFs a que ele deve satisfazer (KELLER, 1990), (BOEHM, 1978).

As abordagens orientada a produtos utilizam métricas (atributos) e seguem, em geral, os seguintes passos:

- Determinação do conjunto de atributos desejáveis (RNFs);
- Determinação da importância relativa de cada atributo;
- Avaliação da conformidade do sistema em relação aos atributos;
- Cálculo da pontuação obtida em cada atributo e a qualidade geral do sistema.

A figura 9 ilustra alguns exemplos de métricas e do processo de avaliação.

Atributo	Métrica
velocidade	Transações/seg, tempo de resposta
Facilidade de uso	Linguagem usada, interface gráfica, help online
tamanho	Kbytes, LOCs, Pontos de função, medidas de complexidade

Avaliação da qualidade geral do sistema

Atributo	Peso relativo	Escore de conformidade	Escore final
Velocidade	.3	6	1.8
Facilidade de uso	.6	5	3.0
tamanho	.1	7	0.7
Qualidade geral			5.5/10

Figura 9. Métricas de Qualidade

Nesta figura, velocidade, facilidade de uso e tamanho são os atributos de qualidade (RNFs) do sistema. A velocidade (desempenho) do sistema tem como métrica transações/seg e tempo de resposta. Esse tipo de métrica é bastante objetiva e relativamente fácil de se medir através de testes simples. Já a métrica facilidade de uso tem três métricas (linguaguem usada, interface gráfica e help online) e, poderia ter mais outras métricas para melhor avaliar se a facilidade de uso do sistema foi ou não alcançada. Assim, esse tipo de métrica não é tão simples de ser mensurada quantitativamente e objetivamente.

Ainda na figura 9, na avaliação da qualidade geral do sistema, a facilidade de uso teve o maior peso e *score* final obtido; já o RNF tamanho teve o menor peso, mas o maior grau de conformidade (aderência) com a especificação. Para se obter o *score* final de cada atributo (RNF), multiplica-se o peso relativo pelo *score* de conformidade.

Assim, com esse tipo de abordagem nem sempre é fácil especificar objetivamente e quantitativamente os atributos e métricas RNFs. Abordagens orientadas a produto são normalmente indicadas quando os requisitos estão bem definidos e podem ser especificados em termos de funcionalidades e fatores qualitativos (métricas) mensuráveis.

Outra forma de tratar os RNFs exigidos pelos stakeholderes são as abordagens orientadas a processos, geralmente, recomendadas no estágio inicial da análise de requisitos. Ao invés de avaliar a qualidade do produto final, a ênfase dessa abordagem é orientar o processo de desenvolvimento do sistema em relação aos NFRs que ele precisa atender.

Nesse contexto, o Framework NFR (CHUNG, 1999) é abordagem GORE mais completa orientada a processo. Além de ser qualitativa e capaz de representar sistematicamente requisitos não-funcionais. O Framework NFR pode dar suporte às atividades de elicitação, análise e negociação (tomada de decisões), documentação (especificação, modelagem) e validação realizadas ao longo do processo de Engenharia de Requisitos. Além disso, ele também é bastante utilizado para tomada de decisão de projetos arquiteturais, tendo grande impacto nas arquiteturas dos sistemas. Os principais elementos do Framework NFR são:

- Softgoals - são unidades básicas para representar requisitos não-funcionais;
- Interdependências – estabelecem relacionamentos entre softgoals;
- Métodos – oferecem técnicas de operacionalização;
- Correlações – oferecem catálogos para inferir possíveis interações.

Neste trabalho não se utiliza o Framework NFR, detalhes podem ser encontrados em Chung (1999). Um dos motivos para a não utilização desse framework é a ausência de uma definição formal de "conflito", o que dá margens a outras interpretações e conceitos.

Na seção 2.4 será vista a abordagem de Análise NFR utilizada nesta pesquisa e a definição formal de conflito.

2.3 A Abordagem i* Wiki

A Engenharia de Requisitos Orientada a Objetivos é constituída por um conjunto de processos, técnicas, frameworks e abordagens de requisitos focados em modelagem organizacional de um sistema. Essa modelagem organizacional de um sistema é centrada nas necessidades dos *stakeholders* e nos seus relacionamentos, pois esses atores dependem uns dos outros para alcançarem seus objetivos (Goals). A modelagem organizacional captura o comportamento do sistema, respondendo a perguntas como "o quê" este sistema deve fazer e também "como fazer". Entretanto, o grande diferencial de uma modelagem organizacional é ser capaz de capturar as razões e motivações (o por quê). Esse adicional, que as demais abordagens de Engenharia de Requisitos geralmente não possuem, faz com que uma abordagem GORE seja capaz também de responder a questões tais: Por que os atores executam os processos? Por que os atores possuem, processam ou geram informações dentro de contexto organizacional de um sistema de informação? A figura 10 ilustra esse tipo de modelagem, ou seja, modelagem organizacional é a soma de modelagem comportamental ("o

quê" o sistema deve fazer e "como" deve fazer) com as razões e motivações do que está sendo modelado, focando-se o contexto organizacional onde o sistema de informação está inserido.

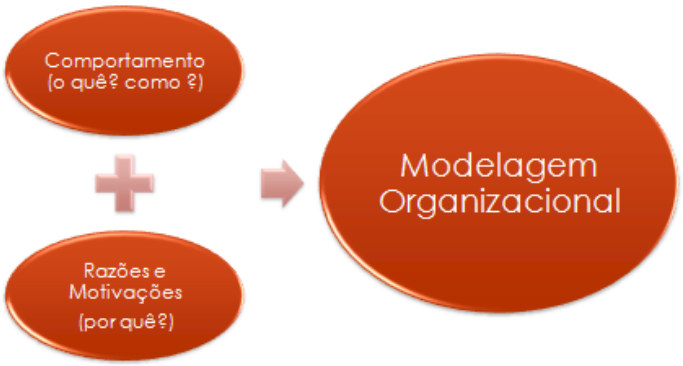


Figura 10. Contexto da modelagem organizacional (CASTRO, 2008)

Dentre as abordagens GORE existentes, a abordagem i* (i-estrela, que significa *Intencionalmente Distribuído*) é amplamente utilizada por diversos grupos de pesquisa espalhados pelo mundo: Canadá, Itália, Espanha, Brasil, entre outros. A origem dessa abordagem remonta do trabalho pioneiro do *framework* i* (YU, 1995). Este framework é utilizado para fazer a modelagem organizacional de um sistema, especificando suas visões comportamentais estratégicas e intencionais (razões).

A estrutura conceitual do i* deve ser utilizada para obter uma compreensão mais apurada sobre os relacionamentos da organização, de acordo com os diversos atores do sistema e compreender as razões envolvidas nos processos de decisões.

A seguir, algumas potencialidades do i* (YU, 1995):

- A modelagem do ambiente organizacional em termos de relacionamentos intencionais provê uma modelagem de ambiente mais rica em comparação aos *frameworks* de requisitos existentes, que produzem modelos em termos de entidades e atividades;
- A modelagem explícita das razões (do inglês, *Rationales*) proporcionam uma compreensão mais profunda sobre o porquê de um determinado sistema ser implantado em uma organização, bem como a maneira que ele será incorporado;
- O suporte à análise dos sistemas propostos e configurações organizacionais em relação aos interesses estratégicos dos atores. Interdependências entre atores são analisadas em termos de oportunidades e vulnerabilidades. Atores são analisados pela aplicabilidade, viabilidade e credibilidade.

O i* Wiki (GRAU et al., 2008) é uma versão simplificada do *framework* i* original (YU, 1995). Esta versão é voltada especialmente aos novos usuários como guia de estudo, mas também serve como guia para os usuários mais experientes. Nessa versão, são abordados os conceitos fundamentais do i*, tais como os modelos SD e SR, bem como seus elementos e ligações.

As idéias principais do framework i* original estão também no i* wiki. Em geral, a notação gráfica e semântica de seus elementos são semelhantes, variando apenas algumas restrições sobre os relacionamentos do modelo. Em (PAES, 2011) há um estudo comparativo minucioso sobre essas abordagens. Como os modelos i* wiki serão o foco do processo MDD de transformação deste trabalho, a abordagem aqui descrita será o i* wiki. Sendo assim, doravante, quanto se mencionar i*, subtende-se i* Wiki (GRAU et al., 2008). Quando necessário falar do framework i* original (YU, 1995), será feita a devida referência.

2.3.1Elementos e características do i* Wiki

A estrutura básica do i* wiki é ilustrada na figura 11. Nesta figura, vê-se que o i* wiki é formado basicamente por dois tipos de modelos responsáveis por representar diferentes níveis de abstração sobre um ambiente organizacional: o modelo de Dependência Estratégica ou SD (do inglês, *Strategic Dependency*) e o modelo de Razão Estratégica ou SR (do inglês, *Strategic Rationale)*. Esses modelos são constituídos, basicamente, de Atores, Elementos intencionais e Dependências/Relacionamentos.

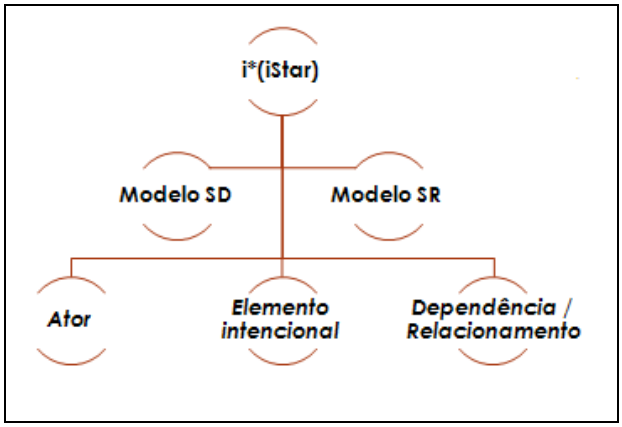


Figura 11. Estrutura básica do i* wiki (CASTRO, 2008)

O ator é considerado uma entidade ativa que realiza ações estratégicas e intencionais para atingir seus objetivos. Atores dependem uns dos outros (possuem dependências) para

atingirem objetivos, realizar tarefas e disponibilizar recursos. Assim, os atores conseguem atingir objetivos difíceis de alcançar sozinhos.

Atores, conforme mostrados na figura 12 (GRAU et al., 2008), são representados por um círculo nos diagramas, podendo ser agentes (humanos ou não), papéis (funções) ou posições (locais de trabalho/cargos).

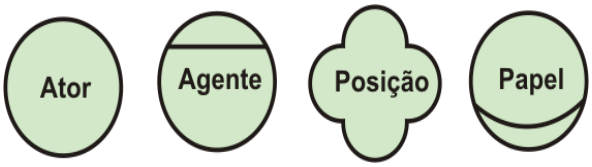


Figura 12. Tipos de Atores

A seguir são detalhados e definidos esses tipos de atores:

- Agente é um ator que possui manifestações físicas concretas. Tanto pode referir-se a humanos quanto a agentes de software ou hardware. Um agente pode executar um determinado papel ou ocupar uma determinada posição que cobre um papel.
- Papel representa a caracterização abstrata do comportamento de um ator dentro de determinados contextos sociais ou domínio de informação. Apresenta as funções que podem ser exercidas por um agente dentro da organização.
- Posição representa uma entidade intermediária entre um agente e um papel. É o conjunto de papéis tipicamente ocupados por um agente, ou seja, representa uma posição dentro da organização onde o agente pode desempenhar várias funções (papéis).

O ator intencional não somente executa atividades e produz entidades, mas tem motivações, intenções e raciocínio por trás de suas ações. Os aspectos intencionais de um ator podem ser caracterizados por metas, crenças, habilidades e compromissos. Um ator é estratégico quando ele não está meramente focalizado em encontrar suas metas imediatas, mas está interessado nas implicações de seus relacionamentos com outros atores.

Elemento intencional pode ser de quatro tipos: Meta, Meta soft , Recurso e Tarefa.Os elementos intencionais são representados graficamente conforme ilustrado na Figura 13(GRAU et al., 2008).

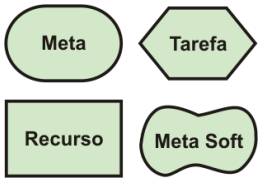


Figura 13. Elementos Intencionais

Meta é o objetivo a ser alcançado, geralmente pela realização de alguma Tarefa. Recursos representam elementos informacionais ou físicos, geralmente intercambiados entre atores ou utilizados pelas tarefas. Meta Soft é uma meta que pode ou não ser satisfeita, ou seja, seu grau ou nível de satisfação é subjetivo.

A tabela 1 mostra os sinônimos geralmente utilizados para esses elementos intencionais. Ao longo da dissertação todos estes termos e seus sinônimos são usados. Entretanto, cabe-se esclarecer que, o termo requisito não-funcional é mais abrangente do que o conceito de softgoal do i*. No escopo desta proposta de pesquisa, considera-se que os softgoals analisados e presentes no ator sistema de software são RNFs de produtos (SOMMERVILLE, 1998). Sendo assim, serão considerados apenas esses softgoals de produto dentro do ator sistema, porque no modelo i* como um todo, podem aparecer softgoals do tipo "tornar clientes felizes, "dar conselhos relevantes", dentre outros, que não fazem parte da classificação definida em (SOMMERVILLE, 1998).

Tabela 1. Sinônimos mais comuns dos elementos intencionais i*

Elemento Intencional	**Sinônimos mais utilizados**		
Meta	Goal	Objetivo	
Meta soft	SoftGoal	Objetivo soft	Requisito não funcional (RNF)
Tarefa	Task		
Recurso	Resource		

Além desses construtores básicos, o i* para alcançar seu objetivo maior de modelagem organizacional dispõe de diversas formas de representar dependências e relacionamentos entre essas primitivas básicas.

A figura 14 mostra, em geral, como estão relacionados os elementos atores e seus tipos específicos. (GRAU et al., 2008).

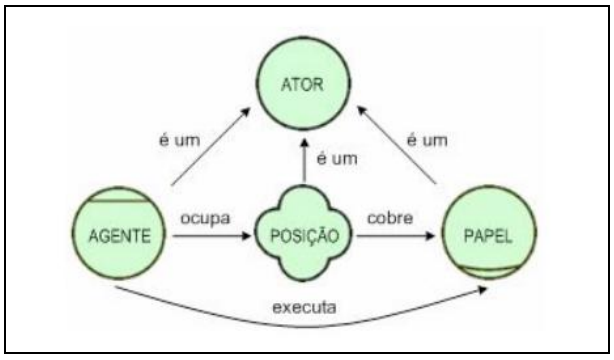


Figura 14. Relacionamento entre os tipos de Atores

Para melhor detalhar e descrever esses relacionamentos, a figura 15 os exibe separadamente, lembrando-se de que "ISA" significa "é um', "Ocuppies" significa "ocupa", "Covers" significa "cobre" e "Plays" significa "executa"; e adicionando ainda os relacionamentos "Is part of" e "INS".

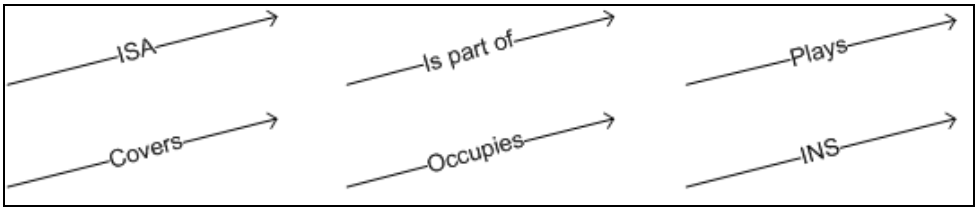


Figura 15. Associações entre Atores

- Is part of: nessa associação cada ator, papel, posição e agente pode ter subpartes. Em Is-part-of há dependências intencionais entre o todo e sua parte. Por exemplo, a dependência do todo sobre suas partes para manter a unidade na organização. Esse relacionamento pode ocorrer entre atores; entre atores e seus tipos agente, posição e papel; entre agente e agente, posição e posição, papel e papel;
- ISA: essa associação representa uma generalização, com um ator sendo um caso especializado de outro ator. Ambas, ISA e Is-part-of, podem ser aplicadas entre quaisquer duas instâncias do mesmo tipo de ator.
- Plays: a associação plays é usada entre um agente e um papel, com um agente executando um papel. A identidade do agente que executa um papel não deverá ter efeito algum nas responsabilidades do papel ao qual está associado, e similarmente, aspectos de um agente deverão permanecer inalterados mesmo associados a um papel que este desempenha.
- Covers: a associação covers é usada para descrever uma relação entre uma posição e os papéis que esta cobre.

- Occupies: esta associação é usada para mostrar que um agente ocupa uma posição na organização, ou seja, o ator executa todos os papéis que são cobertos pela posição que ele ocupa.
- INS: esta associação é usada para representar uma instância específica de uma entidade mais geral. Esse relacionamento só acontece de um agente para outro agente

2.3.1.1 O Modelo de Dependências Estratégica (SD)

O modelo SD representa os requisitos iniciais do sistema e foca no conceito de dependência. Uma dependência descreve uma relação intencional e estratégica entre os atores da organização. A representação básica de uma dependência estratégica SD é da forma ATOR1 -> DEPENDUM -> ATOR2, onde os atores 1 (de origem) e o ator 2 (destino) dependem um do outro por meio de uma dependência intencional chamada Dependum que pode ser qualquer um dos quatro(4) elementos intencionais: meta (objetivo), meta soft (objetivosoft), recurso ou tarefa.

No modelo SD são capturadas as motivações, razões, intenções e os desejos dos atores que fazem parte da organização e apresentadas a sua rede de relacionamentos. Dado um modelo SD, podemos perguntar que novos relacionamentos entre os atores são possíveis, identificar a viabilidade ou não das dependências, ou ainda relacionar os desejos de um ator com as habilidades do ator do qual depende, para poder explorar as oportunidades disponíveis a esses atores.

Mais precisamente, o dependum é um acordo entre dois atores: o depender e o dependee (Figura 16). O depender é o ator que depende de um outro ator (dependee) para que o acordo (dependum) possa ser realizado.

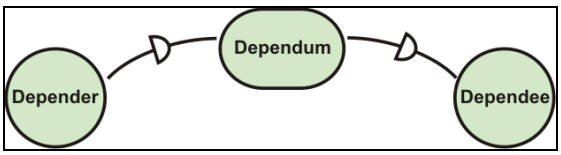


Figura 16. Dependência entre atores

O Dependum (acordo ou contrato entre atores) pode ser qualquer um dos quatro(4) elementos intencionais:meta (objetivo/goal), meta-soft (softgoal), recurso ou tarefa. Quando, em um relacionamento entre atores, o *dependee* disponibilizar um recurso, executar uma tarefa de sua responsabilidade, atender a um objetivo do ator dependente, ou realizar alguma

tarefa para alcançar um *softgoal*, o acordo será satisfeito ou finalizado. Na figura 17 (GRAU et al., 2008), tem-se esses quatro tipos de ligações de dependência.

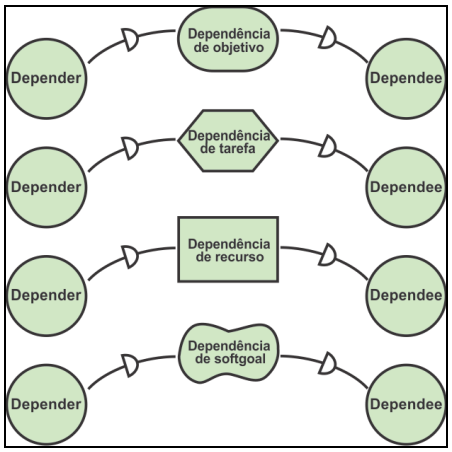


Figura 17. Tipos de relacionamentos de dependência entre atores no i*

Estes relacionamentos entre os atores possuem diferentes graus de dependência ou vulnerabilidades. Para melhor vizualizar e explicar estes graus de vulnerabilidades, a figura 18 ilustra, para uma dada associação Depender->Dependum->Dependee, os três níveis de dependências: aberta, compromissada e crítica.

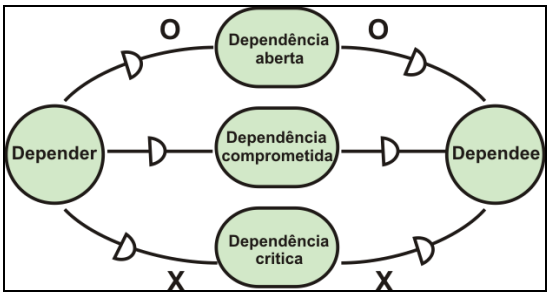


Figura 18. Graus de dependência ou vulnerabilidade em i*

- **Aberta (*open*):** em uma possível falha na satisfação da dependência entre os atores, o ator depender não será afetado. A dependência aberta é graficamente representada por "O".
- **Compromissada (*commited*):** em uma possível falha na satisfação da dependência entre os atores, as intenções do ator *depender* serão afetadas, porém sem conseqüências graves. A dependência compromissada graficamente não possui sinal relacionado.
- **Crítica (*critical*):** na falha da relação de dependência, as intenções do ator depender são afetadas gravemente, sendo necessário gerenciar a viabilidade de toda a rede de relacionamentos e dependências. Graficamente é representada por "X".

2.3.1.2 O Modelo de Razões Estratégicas (SR)

Uma vez que os **atores relevantes** e seus **objetivos** foram identificados, o modelo SD pode ser dado como encerrado e inicia-se a especificação do modelo SR.

O modelo SD é considerado como um **modelo em alto nível** do modelo SR ou, em outras palavras, o modelo SR é um refinamento do modelo inicial SD.

O modelo SR modela as relações de intenção **dentro da fronteira dos atores**. O ator e sua fronteira é representado conforme figura 19.

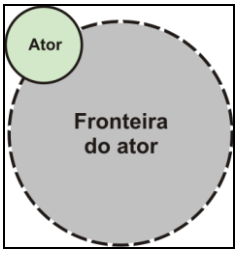


Figura 19. Ator e sua fronteira

Enquanto o modelo SD trata apenas dos relacionamentos externos entre os atores, o modelo SR é utilizado para descrever os relacionamentos internos: interesses, preocupações e motivações dos atores participantes de um processo. Ele possibilita a avaliação das possíveis alternativas de definição do processo, investigando mais detalhadamente as razões existentes, ou intencionalidades, por trás das dependências entre os atores. As intencionalidades de cada ator são identificadas e representadas dentro da fronteira de cada ator.

Assim como o SD, o modelo SR também é composto por ligações de dependência. Além de utilizar os quatro tipos de dependências apresentados no modelo SD - recurso, tarefa, objetivo e objetivo-soft, o modelo SR dispõe de outros três tipos de relacionamentos:

- Relações de **meio-fim**;
- Relações de **decomposição de tarefas**;
- Relações de **contribuição**.

As relações de meio-fim indicam um relacionamento entre um nó fim e um meio para atingi-lo. Este tipo de relacionamento sugere alternativas existentes para se alcançar um determinado fim, onde os meios são expressos em forma de tarefas e os fins expressos em forma de objetivos. A representação deste tipo de relacionamento pode ser visualizada na figura 20 (GRAU et al., 2008).

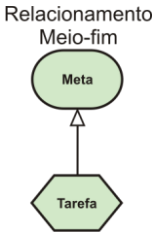


Figura 20. Relacionamento Meio-Fim

Os relacionamentos de decomposição de tarefas ligam um nó de tarefa a seus nós componentes que podem ser outras tarefas, objetivos, recursos ou objetivos-soft. Uma tarefa decomposta só pode ser considerada completa quando todos os seus nós componentes forem realizados ou satisfeitos. A representação deste tipo de relação pode ser visualizada na figura 21.

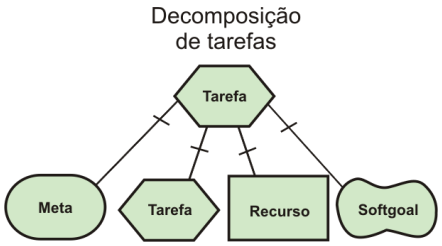


Figura 21. Decomposição de tarefas

Por fim, as relações de contribuição modelam a forma que um elemento (objetivo, tarefa, recurso ou *softgoal*) contribui para a satisfação de um determinado *softgoal*.

A figura 22 mostra os tipos possíveis de relacionamentos de contribuição.

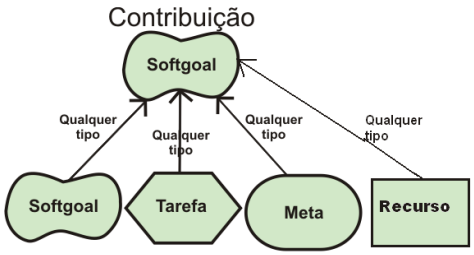


Figura 22. Relacionamento de Contribuição

Como se vê na figura 22 acima os relacionamentos de contribuição ainda possuem atributos que indicam o tipo que esse link entre o elemento intencional e o softgoal pode assumir. Na figura 22, o atributo "Qualquer tipo" pode ser um dos seguintes tipos de contribuição:

- ***Make*:** é uma contribuição positiva forte o suficiente para satisfazer um *softgoal*;
- ***Some* +:** é uma contribuição positiva, entretanto possui um nível de satisfação desconhecido;

- ***Help*:** é uma contribuição parcialmente positiva, pois ela não consegue satisfazer um *softgoal* sozinha;
- ***Unknown*:** é uma contribuição cuja influência é desconhecida;
- ***Some -*:** é uma contribuição negativa, entretanto possui um nível de satisfação desconhecido;

- ***Hurt*:** é uma contribuição parcialmente negativa, pois ela não consegue comprometer um *softgoal* sozinha;
- ***Or*:** é uma contribuição cujo destino é satisfeito se algum dos elementos origem for satisfeito;
- ***And*:** é uma contribuição cujo destino é satisfeito se todos os elementos origem forem satisfeitos;
- ***Break*:** é uma contribuição negativa forte o suficiente para comprometer um *softgoal*.

  A figura 23 abaixo resume graficamente esses tipos:

Figura 23. Tipos de relacionamentos de contribuições para *softgoals*

Esses tipos de contribuição são bastante utilizados na análise de requisitos não funcionais a fim de tomar decisões e analisar conflitos.

2.3.2 Metamodelo do i* Wiki

Esta seção descreve o metamodelo i* wiki utilizado no processo de transformação MDD proposto nesta dissertação. A figura 24 mostra a representação gráfica desse metamodelo ECORE (EMF, 2009). Estes metamodelo é o mesmo da ferramenta IStarTool (ISTARTOOL, 2009) que edita graficamente modelos i* wiki e é composto pelos seguintes construtores ou primitivas:

- **Model (Modelo)** – representa o "palco" onde os elementos de modelagem **Model** é um repositório de atores, elementos intencionais e relacionamentos.
- **Element** – é a metaclasse que representa os elementos intencionais. Possui o atributo type que referencia a Enumeração **ElementType** constituída pelos itens: GOAL, SOFTGOAL,TASK e RESOURCE.

- **Actor** – esta metaclasse representa os atores do modelo que podem ser do tipo Enumeração **ActorType**. Como no modelo SR o Ator pode conter outros elementos i*. Essa metaclasse tem as seguintes agregrações: **Element, MeansEnd, DecompositionTask, ContributionLink.**
- **MeansEnd –** representa o construtor Meio-Fim que possui associado um elemento fonte e destino do tipo **Element**, onde o fonte é um elemento do tipo TASK e o destino um GOAL.
- **DecompositionTask –** representa o construtor Decomposição de Tarefa. Possui um elemento fonte e destino do tipo **Element**, onde o elemento que está sendo decomposto só pode ser do tipo TASK.
- **ContributionLink –** representa o construtor Link de Contribuição para elementos destinos SOFGOAL e pode ser do tipo especificado na Enumeração **ContributionLinkType.**
- **Link** – é uma metaclasse abstrata que tem a função de permitir o relacionamento de dois elementos representado pela metaclasse concreta **DependencyLink** que é o link de dependência entre esses elementos. **DependencyLink** pode ser do tipo Enumeração **DependencyLinkType** (COMMITED,OPEN,CRITICAL). O Modelo (**Model)** é um agregado desses links de dependências. Cada Link do Modelo tem um elemento fonte e destino do tipo **NodeObject,** uma metaclasse também abstrata que pode representar tanto um elemento intencional **Element** quanto um ator **Actor.**
- **ActorAssociation** – é o construtor que representa uma associação entre atores, tendo uma origem e alvo do tipo **Actor.** Essa associação pode ser de um dos tipos definidos em **ActorAssociationType.**

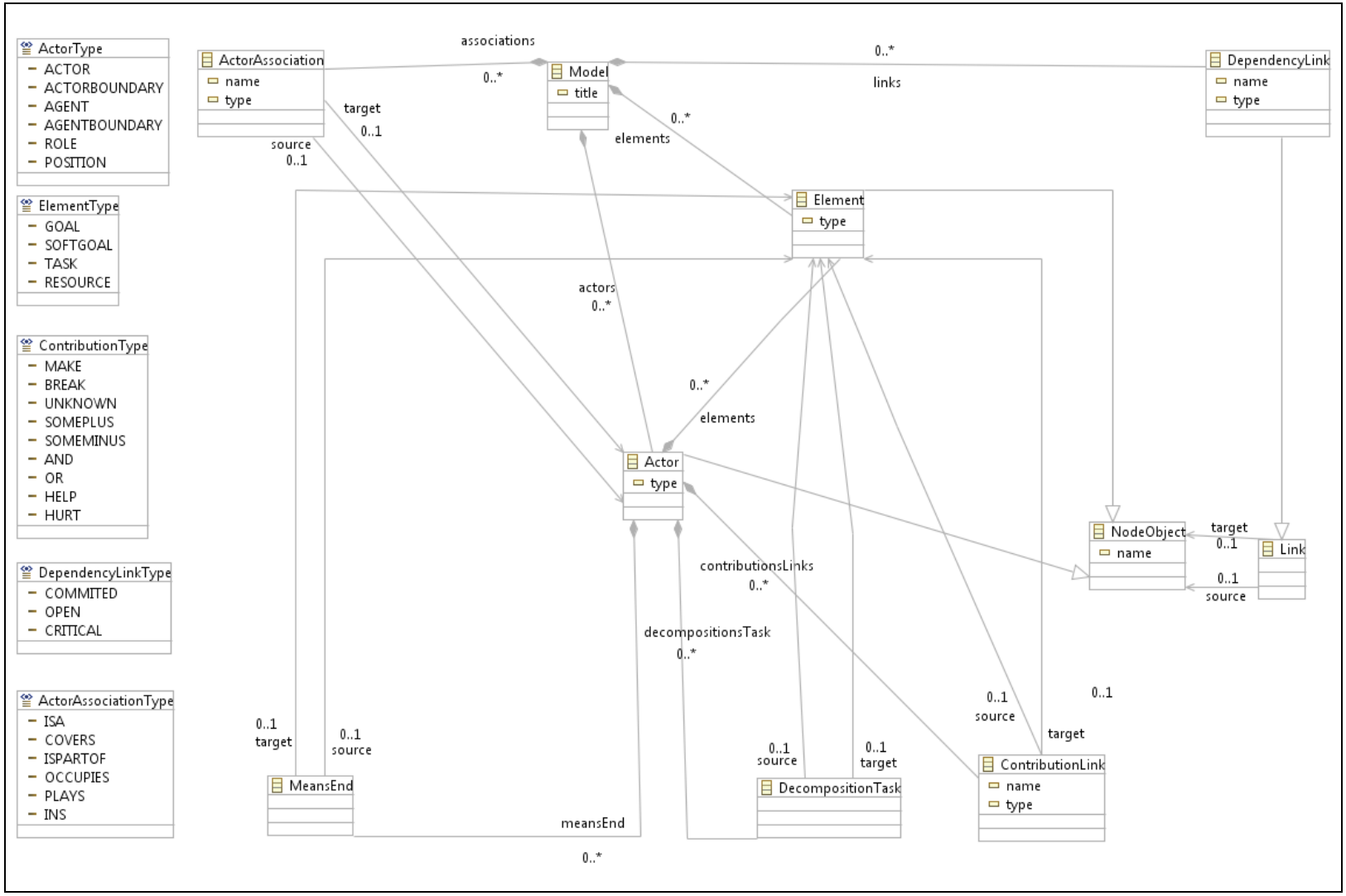


Figura 24. Metamodelo i* wiki (ISTARTOOL, 2009)

Uma das limitações desse metamodelo ilustrado na figura 24 é o uso de "enumerations" para representar os tipos de elementos intencionais. Com "enumerations" fica inviável a criação de atributos extras, principalmente para o elemento intencional "recurso" que precisa de atributos adicionais para fazer a distinção entre um recurso físico e um recurso informativo necessário ao processo transformativo OOM-NFR proposto. O ideal seria criar metaclasses distintas para cada elemento intencional do i*, para então, criar atributos extras dessas metaclasses que facilitassem o processo transformativo. Outras limitações desse metamodelo são mencionadas na seção 6.2 do capítulo 6 deste trabalho.

2.4 Análise NFR e Tratamento de Conflitos

Esta seção faz uma breve sinopse sobre a metodologia utilizada nesta pesquisa para analisar NFR, tratar conflitos e realizar a tomada de decisões.

Na literatura há diversas formas de tratamento conflitos, tais como o Processo Hieráquico Análitico, do inglês, AHP "Analytic Hierarchy Process" (SAAT, 2008); a Lógica Difusa "Fuzzy Logic" (KOSKO, 1997), entre outros.

A lógica difusa define elementos tais como: conjuntos difusos, operadores, qualificadores, predicados e regras difusas capazes de fazer inferências, e tomar decisões ou raciocionar em termos de dados ou axiomas imprecisos (KOSKO, 1997). Sistemas baseados em lógica difusa têm a habilidade de codificar o conhecimento de forma próxima à usada pelos especialistas. Atualmente, existe uma vasta quantidade de áreas de aplicação de sistemas que usam lógica fuzzy.

O Processo Hierárquico Analítico-AHP (SAAT, 2008) é um método múltiplo critério de tomada de decisão já validado e com inúmeras aplicações na vida real. Com o AHP é possível tomar decisões complexas, tendo-se como entrada as soluções alternativas (opções) e suas prioridades. Essas alternativas e prioridades são expressas em forma de matrizes através das quais é realizado um processamento matemático e estatístico que resulta num *ranking* das melhores opções ou soluções encontradas, levando-se em consideração as prioridades definidas em termos de fatores ou pesos.

Em (HORKOFF et al. 2010) é definida uma abordagem intitulada "Finding Solutions in Goal Models: An Interactive Backward Reasoning Approach" , ou abordagem de raciocínio inverso interativo, onde se realiza uma análise NFR *top-down*. Essa abordagem verifica se determinado alvo, SOFGOAL OU GOAL, pode ser satisfeito totalmente ou parcialmente ou

negado ou se há conflitos a serem resolvidos e quais operacionalizações (tasks) devem ser escolhidas para que essa resolução ocorra. Esse método é o oposto a abordagem "forward" (ver seção 2.4.1) e tem como principal vantagem a capacidade de responder a questões como "o que" , "como" e "o por quê" das decisões tomadas, pois seu processo de resolução parte das raízes (elementos alvos) do modelo para as folhas, ou seja, é "top-down".

Além disso, essa abordagem possui as seguintes características:

- Utiliza um SAT-Solver (resolvedor de fórmulas lógicas).
- É interativa, ou seja, exige interação e julgamento humano.
- É iterativa: o SAT-Solver é chamado várias vezes até encontrar ou não uma atribuição que satisfaça a uma expressão lógica na forma normal conjuntiva - CNF passada como argumento, ou seja, a cada iteração, o algoritmo é chamado com uma nova fórmula CNF passada pelo usuário e através de vários refinamentos (iterações) vai fazendo a resolução. Uma fórmula lógica (booleana) está na forma normal conjuntiva quando ela é composta de cláusulas conectadas pelo conector lógico de conjunção "^".
- Já foi validada e implementada na ferramenta de modelagem i* OpenOME (OPENOME, 2010).

Nesta dissertação escolheu-se a abordagem "Interactive and Iterative Forward/Backward Analysis" (HORKOFF et al. 2010), em detrimento das abordagens lógica fuzzy e AHP, pelas seguintes razões:

- É interativa e baseada na lógica ou semântica dos modelos i*;
- Não exige que sejam especificadas antecipadamente e estaticamente alternativas e prioridades na forma matemática tal como fazem as abordagens AHP e lógica fuzzy;
- Define **formalmente** o que é um conflito. Já as abordagens AHP e lógica fuzzy carecem de uma definição formal de conflito.

Esses critérios são importantes para o desenvolvimento deste trabalho de pesquisa que versa sobre transformação de modelos focado em tratamento, resolução e priorização de conflitos de requisitos não-funcionais especificados em i* wiki. A resolução de conflitos e tomada de decisões em modelos i* têm uma natureza interativa nas situações reais e seguem uma lógica que depende de regras de propagação específicas desses modelos, por essa razão é que a abordagem (HORKOFF et al. 2010) se mostra a mais adequada a ser utilizada no processo OOM-NFR proposto.

Enquanto isso, os métodos baseados em lógica fuzzy e AHP, não são adequados à natureza de um processo de decisão para modelos i*, ou seja, não há interesse de se fixar previamente elementos (alternativas) e prioridades (fatores/pesos) e esperar que, ao final de um processamento em lote (“batch”, não interativo), saia automaticamente como resultado, as melhores decisões ou resoluções de conflitos sem haja alguma intervenção humana.

Os métodos AHP e lógica fuzzy poderiam ser utilizados, mas tornariam a atividade de tratamento de conflitos e tomada de decisões bastante complexa, custosa e ineficiente considerando a natureza já apresentada dos modelos i*.

Por ser a abordagem escolhida, na próxima seção, tem-se uma breve explanação sobre a Análise Backward/Forward interativa e iterativa.

2.4.1 Análise Backward/Forward

Inicialmente, na figura 25 são ilustradas as regras básicas de propagação:

Originating Label		**Contribution Link Type**						
Label	Name	Make	Break	Help	Hurt	Some+	Some-	Unknown
✓	Satisficed	✓	✗	✓.	✗.	✓.	✗.	?.
✓.	Partially Satisficed	✓.	✗.	✓.	✗.	✓.	✗.	?.
↯	Conflict	↯	↯	↯	↯	↯	↯	?.
?.	Unknown	?.	?.	?.	?.	?.	?.	?.
✗.	Partially Denied	✗.	✓.	✗.	✓.	✗.	✓.	?.
✗	Denied	✗	✓.	✗.	✓.	✗.	✓.	?.

Figura 25. Regras de Propagação (HORKOFF et al. 2010)

A partir dessas regras (Figura 25) intuitivamente lógicas, pode-se chegar a algumas inferências, por exemplo:

- Se é selecionado ou satisfeito um elemento de origem que tem como *link* de contribuição “Break” para algum destino, como resultado, estará também se negando ou quebrando ”Break” este elemento destino (primeira linha, segunda coluna da figura 25);
- Se não é selecionado ou satisfeito um elemento de origem que tem como link de contribuição um “Make” para algum elemento destino, o resultado da propagação é a negação ou “Break” para esse elemento destino (última linha, primeira coluna da figura 25);
- Se não é selecionado ou satisfeito um elemento de origem que tem como link de contribuição um “Break” para algum elemento destino, o resultado da propagação é parcialmente satisfeito “Help” para esse elemento destino (última linha, segunda coluna da figura 25).

Para facilitar a aplicação dessas regras, na figura 26 mostra-se um modelo bastante simplificado em i* e explica-se como é realizado o processo de análise "Forward" NFR.

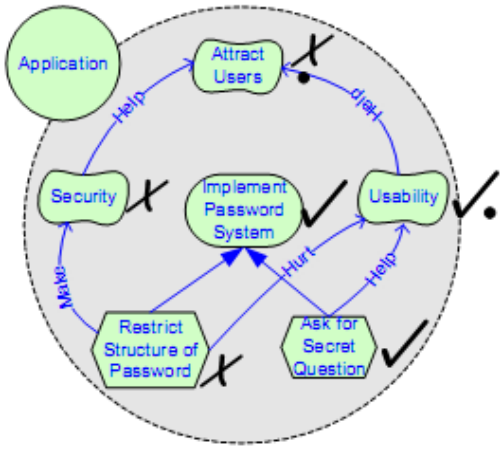


Figura 26. Exemplo do processo de Análise Forward NFR (HORKOFF et al. 2010)

O processo da Figura 26 é "forward" (para frente), também conhecido como "bottom-up", ou seja, parte-se dos elementos folhas para o(s) elemento(s) raiz(es). Nesse caso, poder-se-ia escolher três configurações possíveis para as tarefas "Restrict Structure of Password" e "Ask for Secret Question" como formas de verificar se essas configurações atendem ou satisfazem o requisito NFR prioritário de "Attract Users". No exemplo da figura 26 acima, o analista modelador escolheu em não selecionar (negar ou não satisfazer) "Restrict Structure of Password" e selecionar "Ask for Secret Question".

Com essa seleção feita manualmente, o processo a satisfaz o elemento intencional goal "Implement Password System" pois o Means-End representa logicamente, diante, várias alternativas, o conectivo "OU" que só não seria satisfeito se nenhuma das duas tarefas folhas fossem escolhidas.

O processo também propaga automaticamente para "Security" a negação, pois de acordo com as regras de propagação está se quebrando uma tarefa que, de acordo com o modelo satisfaria completamente "Make" o requisito NFR "Security".

Já sobre o NFR "Usability", o processo requer a interação humana, pois recebe como entrada dois resultados "parcialmente satisfeito" ou "Help" vindos através dos links de contribuições "Restrict Structure of Password" e "Ask for Secret Question". Nesse momento o analista decide, de acordo com seu bom senso, intuição e prioridades o que realmente deve ser colocado como label sobre "Usability". Nesse caso, como vieram dois resultados "parcialmente satisfeito" decidiu-se que a "Usabilidade" ficaria também como "parcialmente satisfeito".

Por fim, o processo interage novamente com o analista para receber um julgamento humano sobre "Attract Users" que recebe dois resultados: parcialmente satisfeito, propagado a partir de "Usability" e parcialmente negado vindo de "Security".

Nessa situação, o analista decide que somente com usabilidade parcial e nenhuma segurança o requisito não funcional "Attract Users" é parcialmente negado.

Entretanto ele também poderia decidir que essa situação fosse marcada como um conflito ou parcialmente satisfeita.

Nesse modelo da figura 26, o melhor caso para satisfazer "Attract Users" seria selecionando ambas tarefas "Restrict Structure of Password" e "Ask for Secret Question". Neste caso, ainda haveria interação humana e o resultado final para o nó NFR raiz "Attract Users" seria um parcialmente satisfeito.

Caso fosse optado pela abordagem "backward" (para trás) primeiramente seria selecionada a raiz "Attract Users" para ser satisfeita e depois seria acionado o processamento interativo, iterativo e para trás. Esse processo tentaria escolher automaticamente, através de regras de propagação e axiomas "backward" a melhor configuração dos elementos folhas para satisfazer essa raiz. Quando não fosse possível inferir automaticamente, requisitaria o julgamento humano para tomar as devidas decisões ou resolução de conflitos. Essa abordagem é mais eficiente e completa que a abordagem "forward, no sentido de ter capacidade de responder as questões "Como" e "Por quê" das decisões escolhidas durante o processo de resolução (HORKOFF et al. 2010). A seguir algumas das características dessas abordagens:

Análise Forward:

- Responde a questão:
    - Se forem escolhidas um determinado conjunto de alternativas folhas do modelo i*, o que será ou poderá ser satisfeito?
- É tediosa para análise de modelos i* grandes com várias alternativas, pois há necessidade de realizar a análise n vezes (exaustivamente) até se obter um conjunto ideal que satisfaça o que se deseja.

Análise Backward:

- Responde as questões:
    - É possivel satisfazer determinados elementos raízes do modelo i* ?
    - Se sim, como?
    - Se não, por quê?

- A análise é realizada uma única vez, partindo de um conjunto de elementos raízes do modelo i*.

Mais exemplos sobre análise forward e backward serão vistos nos capítulos 4 e 5 deste trabalho durante a fase de pré-processamento da abordagem proposta.

2.4.2 Implementação e Formalização do Processo

O processo apresentado na seção anterior já é integrado com a ferramenta de modelagem OpenOME (OPENOME, 2010). Essa abordagem escolhida funciona apenas como uma caixa-preta (ferramenta de apoio) para o processo proposto. Desse modo, houve necessidade de utilizar também essa ferramenta, pois a ferramenta IStarTool (ISTARTOOL, 2009) não implementa o processo de resolução (HORKOFF et al. 2010). Nos capítulos 4 e 5 são dados vários exemplos de tratamento e resolução de requisitos não-funcionais, utilizando-se a ferramenta OpenOME como apoio.

Há necessidade de que se passe formalmente como argumento para o algoritmo de resolução, ou seja, utilizando-se os elementos de lógica de primeira ordem, uma expressão lógica na forma nornal conjuntiva (CNF) que expresse o alvo a ser analisado e o modelo i* para um resolvedor de fórmulas que, na abordagem, utiliza o Zchaff SAT solver (MAHAJAN et al., 2004).

O processo proposto OOM-NFR utiliza na análise NFR a abordagem "Interactive and Iterative Forward/Backward Analysis" (HORKOFF et al. 2010) descrita na seção 2.4.1. Como essa abordagem, implementada na ferramenta OpenOME, define formalmente conflito, as regras de propagação e axiomas do modelo i*, faz-se necessário apresentar brevemente algumas dessas definições que, de fato, formalizam a abordagem i*:

Na figura 27, formaliza-se o i* como uma tupla formada por três conjuntos: elementos intencionais, relacionamentos e conjunto de atores.

Definition: agent-goal model. An i* model is a tuple $\mathcal{M} =< \mathcal{I}, \mathcal{R}, \mathcal{A} >$, where $\mathcal{I}$ is a set of intentions, $\mathcal{R}$ is a set of relations between intentions, and $\mathcal{A}$ is set of actors.

Figura 27. Formalização do Modelo i*(HORKOFF et al. 2010)

Na figura 28, define-se o conjunto dos elementos intencionais do i*, onde uma intenção (I) é mapeada em um dos elementos intencionais (IntentionType) do i*, ou seja, uma intenção é mapeada em softgoal, goal, task ou resource.

Definition: element type. Each intention maps to one type in the $IntentionType$ set, $\mathcal{I} \mapsto IntentionType$, where $IntentionType = \{Softgoal, Goal, Task, Resource\}$.

Figura 28. Formalização do Modelo i*(HORKOFF et al. 2010)

Na figura 29, têm-se os predicados que podem ser aplicados a cada elemento intencional.

Definition: analysis predicates. We express agent-goal model analysis labels using the following set of predicates, $\mathcal{V}$, over $i \in I$: $v(i) \in \mathcal{V} \mapsto AnalysisPredicates = \{S(i), PS(i), C(i), U(i), PD(i), D(i)\}$ where S(i)/PS(i) represents full/partial satisfaction, C(i) represents conflict, U(i) represents unknown, and D(i)/PD(i) represents full/partial denial.

Figura 29. Formalização do Modelo i*(HORKOFF et al. 2010)

Basicamente esta figura 29 define formalmente os predicados ou níveis de satisfação para um determinado elemento intencional do i*, onde: S(i) ou PS(i) representa totalmente ou parcialmente satisfeito; C(i) representa um conflito; U(i) representa nível de satisfação desconhecido e D(i) ou PD(i) representa totalmente ou parcialmente negado.

Na figura 30 é expressa a propagação qualitativa e interativa na forma normal conjuntiva (CNF). É uma fórmula desse tipo que é passada para o SAT solver para resolução. O alvo (Target) é expresso em forma de predicado de acordo com o nível de satisfação desejado, por exemplo, o alvo Segurança, pode ser formalizado através da expressão PS(Segurança) que representa a segurança sendo parcialmente satisfeita.

To express the problem of assigning evaluation labels to an agent-goal model in terms of a CNF SAT formula, we follow the formalization in [4], adopting their classification of the components of the formula as follows:

- The target values for the procedure, $\phi Target$
- Axioms describing forward propagation, $\phi Forward$
- Axioms describing backward propagation, $\phi Backward$
- Axioms describing invariant properties of evaluation labels, $\phi Invariant$
- Any additional constraints on propagation, $\phi Constraints$

The SAT formula is constructed as follows:

$$\phi = \phi Target \wedge \phi Forward \wedge \phi Backward \wedge \phi Invariant \wedge \phi Constraints \ . \quad (2)$$

Figura 30. Formalização do Modelo i*(HORKOFF et al. 2010)

Na figura 31, define-se formalmente um conflito.

Definition: (conflict label vs. conflict.) A conflict label is the $\lightning$ label originating when a user has selected conflict in human judgment. A conflict between labels for an intention $i \in I$ is when a predicate from more than one of the following four sets is true: $\{S(i), PS(i)\}$, $\{U(i)\}$, $\{C(i)\}$, $\{PD(i), D(i)\}$.

Figura 31. Formalização da definição de Conflito num Modelo i*(HORKOFF et al. 2010)

Traduzindo-se em português, tem-se a mesma definição a seguir:

Definição: (rótulo de conflito versus conflito) Um rótulo de conflito é o rótulo "Z" originado quando o usuário seleciona conflito no julgamente humano. Um conflito entre rótulos para uma intenção i que pertence a I é quando um predicado a partir de mais de um dos seguintes quatro conjuntos é verdadeiro: {S(i),PS(i)}, {U(i)},{C(i)},{PD(i),D(i)}.

Em outras palavras, a definição de conflito, segundo as regras de propagação que o algoritmo implementa, ocorre quando para um elemento intencional alvo há avaliações (ver figura 25 de propagação das regras) que pertencem a mais de um dos conjuntos definidos (Figura 31).

Por exemplo, se dois resultados de avaliações pertencerem a apenas um desses conjuntos não há conflito, mas se pertencerem a mais de um desses conjuntos então o conflito ocorre e o algoritmo marca o elemento intencional onde o conflito ocorre com o seu rótulo de identificação "Z**"**.

Na figura 32, tem-se a formalização dos axiomas forward e backward de propagação para os diversos tipos de relacionamentos do Modelo i*.

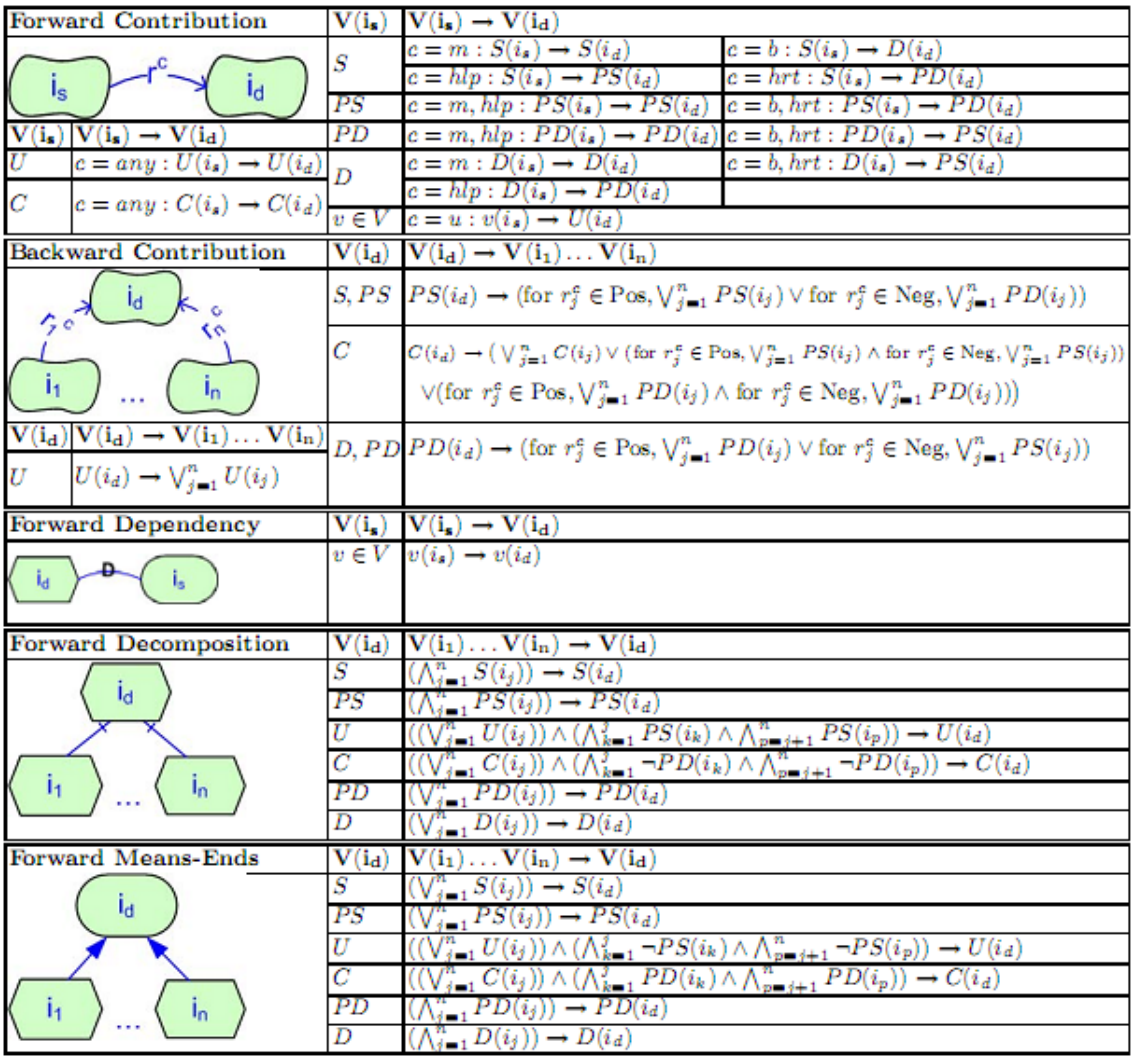


Figura 32. Formalização dos axiomas Forward e Backward i*(HORKOFF et al. 2010

Mais detalhes podem ser encontrados em (HORKOFF et al. 2010). A implementação dessa abordagem é uma caixa preta (ferramenta de suporte) para o processo proposto nesta pesquisa. Foi utilizado por ser mais adequado à natureza do problema de análise NFR e o tratamento de conflitos em modelos i*.

2.5 Considerações Finais

Neste capítulo foi visto como é o processo da Engenharia de Requisitos, apresentando suas atividades básicas de elicitação, negociação, documentação e validação de requisitos.

Como o trabalho presente propõe um projeto de transformação MDD que considera os requisitos não-funcionais de um sistema, foi reservada a subseção 2.2 para detalhar mais a terminologia, natureza, importância, classificação e abordagens de tratamento dos RNFs.

Na seção 2.3 foi elencada a principal abordagem de engenharia de requisitos orientada a objetivos (GORE) que subsidia este trabalho de pesquisa: i* Wiki.

O i* Wiki será utilizado como modelo de entrada do processo MDD proposto nesta dissertação. O i* Wiki foi escolhido por ser um modelo que está com um bom nível de maturidade e aceitação na comunidade acadêmica, inclusive com aplicações na área industrial e comercial. Além disso, o i* Wiki é um modelo onde se é possível especificar o que, o como e o porquê (raciocínio) sobre um sistema a ser desenvolvido. Possibilita modelar os requisitos não-funcionais (softgoals) e tem relacionamentos de análise de contribuição, característica essa que será explorada no processo transformativo proposto. Também foi visto que os softgoals utilizados no escopo desta pesquisa são os internos ao ator sistema e relacionados a requisitos não-funcionais de produto.

Na seção 2.4, foi apresentado como é realizada a análise NFR e tratamento conflitos nesta pesquisa, além de ser definido o conceito de conflito.

Num processo de desenvolvimento de software padrão faz-se necessário ainda gerar modelos conceituais no nível de projeto, que correspondam ao modelo de requisitos inicialmente especificado. Em geral, esse processo de transformação de modelos pode ser realizado por um processo manual ou automatizado. No próximo capítulo, será visto como a arquitetura MDA endereça essas questões e como desenvolvimento de software é realizado através do paradigma *Model-Driven Development* – MDD.

CAPÍTULO 3

3. Desenvolvimento Orientado a Modelos

Este capítulo objetiva descrever uma visão geral dos padrões OMG, arquitetura MDA da OMG e seus conceitos básicos. Serão também apresentados a abordagem MDD OO-Method e alguns trabalhos de pesquisa de grande importância na área de transformação de modelos i* para OO-Method.

Dentro do contexto de que modelar é uma atividade essencial da engenharia de software, o MDD representa atualmente um papel central no processo de engenharia de software. Convém lembrar que essa idéia não é nova. Desde a década de 1970, que os métodos formais difundiram o desenvolvimento de software a partir de modelos formais matemáticos que eram transformados até se gerar o código fonte final do sistema. A partir de um desenvolvimento formal é possível elevar a qualidade do software com técnicas formais de validação e verificação (SAALTINK, 1997).

Com o amadurecimento das linguagens de modelagem de software e a complexidade da conjuntura atual da indústria de software, cada vez mais, essa idéia tem se consolidado através de abordagens que adotam MDD como um padrão de desenvolvimento. Em 2001, ao especificar a MDA, o grupo OMG, na verdade, definiu uma nova instância (padrão) de processo de desenvolvimento de software dirigido a modelos que já havia sido pensado e criado há algumas décadas anteriores pela área de especificação e desenvolvimento formal de software.

Entretanto, é importante deixar claro que MDD não é sinônimo de MDA. Segundo (KELLY, 2008) , existem os processos MDD com base em MDA que utilizam apenas o padrão UML e também existem vários outros processos MDD que são baseados em Domain-Specific Modeling Languages (DSMLs). Esta pesquisa tem como escopo duas DSMLs (i* e

OO-Method) que **não são** baseadas no padrão MDA (UML). Assim, o processo proposto nesta pesquisa é um processo MDD baseado em DSMLs.

Os principais argumentos em se utilizar um processo de desenvolvimento de software dirigido a modelos são os seguintes: maior produtividade, portabilidade, interoperabilidade, menor custo, mais facilidade na evolução do software, enfim, maior qualidade do produto.

Neste capítulo, será apresentada uma visão geral dos padrões OMG, arquitetura MDA e seus conceitos básicos, com o objetivo de explorar a teoria básica sobre o processo de desenvolvimento de software orientado a modelos. Será dado um foco maior sobre a transformação de modelos a partir de metamodelos, pois, segundo a arquitetura MDA essa técnica facilita a automação do processo ao se utilizar tecnologias de transformações modelo-modelo tais como ATL, QVT (ECLIPSE, 2009).

3.1 Arquitetura Dirigida a Modelos

Para um melhor entendimento da arquitetura dirigida a modelos, o padrão MDA (MDA, 2003) do grupo OMG (OMG, 1997) define os seguintes conceitos:

- **Modelo**

  Um modelo de um sistema é a sua representação (especificação) funcional, estrutural e comportamental. Uma especificação é dita como formal quando é baseada numa linguagem que tem uma sintaxe e semântica bem definida e possivelmente tem também regras de análise, inferências ou prova de seus elementos. Essa sintaxe pode ser gráfica (visual) ou textual. A semântica pode ser mais ou menos formal.

- **Dirigido a Modelos**

  MDA é uma abordagem de desenvolvimento de sistema que usa o poder dos modelos. É dirigida a modelos porque provê meios de usar modelos para direcionar o curso de entendimento, projeto, construção, distribuição, operação, manutenção e modificação.

- **Arquitetura**

Arquitetura de um sistema é a especificação de suas partes e conectores, além das regras de interação dessas partes usando os conectores.

- **Ponto de vista (Viewpoint)**

  Um ponto de vista de um sistema é uma técnica de abstração, usando um conjunto selecionado de conceitos arquiteturais e regras de estruturação que visa focar ou representar um aspecto (característica) dentro desse sistema. O termo abstração está sendo usado para significar o processo de suprimir (esconder) um detalhe selecionado para estabelecer um modelo simplificado.

- **Plataforma**

  Uma plataforma é um conjunto de subsistemas e tecnologias que provê um conjunto coerente de funcionalidade através de interfaces e padrões de uso especificados, que qualquer aplicação (sistema) suportada por essa plataforma pode usar, sem ter que saber detalhes de como essa funcionalidade provida pela plataforma é implementada.

- **Pontos de Vistas (Modelos) MDA**

  - **Modelo Independente de Computação (CIM)**

    É uma visão do sistema a partir de um ponto de vista (viewpoint) independente de computação. O CIM não mostra detalhes da estrutura dos sistemas, sendo usualmente chamado de modelo de domínio ou modelo de négocio e usa, em sua especificação, um vocabulário familiar aos usuários do domínio (problema) em questão. Os usuários do CIM geralmente não têm conhecimento sobre modelos ou artefatos usados para realizar as funcionalidades definidas através dos requisitos. Esse modelo foca no ambiente do sistema e nos seus requisitos, deixando os detalhes da estrutura e processamento (computação) do sistema escondidos aos usuários (stakeholders) ou, mesmo, esses detalhes são indeterminados.

    Dessa forma o CIM tem um importante papel de fazer a ponte (reduzir a lacuna “gap”) entre aqueles que são especialistas no domínio do problema e seus requisitos, e aqueles que são especialistas em projeto (arquitetura) e construção dos artefatos que juntos vão satisfazer aos requisitos do domínio, elicitados pelos usuários.

    Fazendo-se uma analogia com o modelo de processo de Engenharia de Requisitos de alto nível definido na seção anterior 2.1, o CIM seria

obtido no processo de documentação e especificação dos requisitos, ou seja, ao se especificar um modelo de requisitos para o sistema.

- **Modelo Independente de Plataforma (PIM)**

O PIM foca na operação do sistema (modelo computacional), mas escondendo os detalhes necessários para implantar esse modelo numa plataforma específica. O PIM é único para o sistema e não muda quando se varia de uma plataforma para outra. Esse ponto de vista independente de plataforma pode ser especificado usando-se uma linguagem de modelagem de propósito geral (UML) ou uma linguagem específica (OO-Method) como será visto no capítulo 3. O PIM é um modelo conceitual do sistema.

- **Modelo Específico de Plataforma (PSM)**

Este modelo é uma visão do sistema que agrega características e elementos constituintes de uma plataforma específica, contendo informações da tecnologia utilizada na aplicação como a linguagem de programação, os componentes de middleware, a arquitetura de hardware e de software. Para que isso seja possível é necessário o suporte de ferramentas que façam o mapeamendo adequado de uma especificação abstrata (PIM) para uma determinada plataforma. O PSM, por sua vez, passa por processo(s) de refinamento(s) para obtenção do nível de especificação desejado. A obtenção desse nível torna possível a transformação do mesmo no código (implementação) da aplicação. O modelo PSM é o responsável por lidar com toda heterogeneidade e complexidade dos diversos tipos de plataformas existentes.

- **Transformações (Mapeamentos)**

A força motriz do padrão MDA é a transformação de modelos, que pode ser realizada entre modelos de um mesmo ponto de vista ou entre pontos de vistas diferentes, tanto num sentido direto quanto inverso (reverso). Em qualquer caso, sempre um modelo é usado como parâmetro de entrada para ser transformado em outro modelo.

Na figura 33 mostra-se o ciclo (sentido) mais natural de um processo de desenvolvimento de software que segue o padrão MDA, partindo do CIM (modelo de requisitos), passando-se pelos modelos PIM e PSM, até o nível mais baixo de código (implementação)

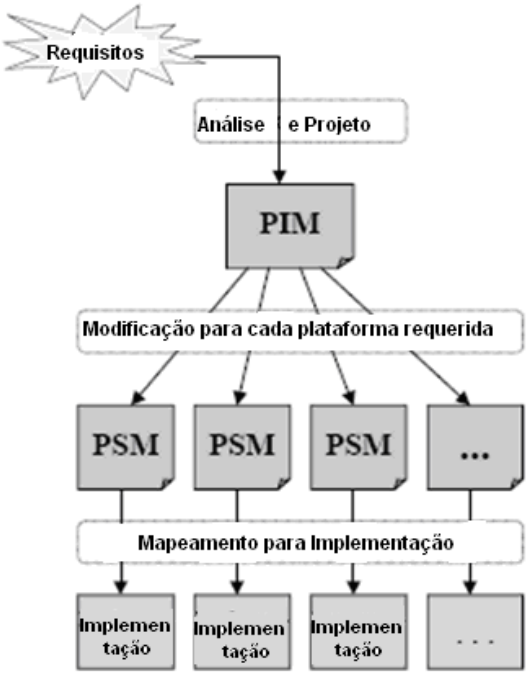


Figura 33. Transformações em MDA (HITACHI 2002)

Entretanto, é possível também as seguintes transformações (mapeamentos):

- PSM => PIM (Engenharia Reversa)
- PIM => PIM, PSM => PSM (Modelos de mesmo nível
- Implementação => PSM (Engenharia Reversa)
- PIM => Implementação

Como também se observa na figura 34, MDA, normalmente, propõe transformação consecutiva de modelos de níveis de abstrações mais altos para níveis de abstrações mais baixos até se obter o produto de software final.

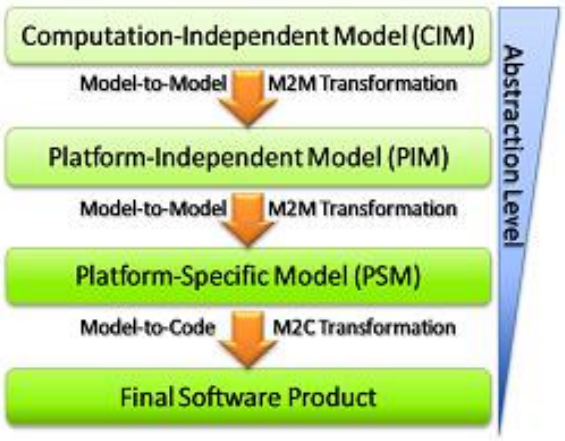


Figura 34. Níveis de Abstração dos Modelos MDA (HITACHI 2002)

MDA baseia-se no uso de UML para realizar as atividades de modelagem relativas aos modelos CIM, PIM e PSM. Entretanto, outras abordagens de modelagem que não são

baseadas em UML podem também ser utilizadas. Segundo (KELLY, 2008), existem processos MDD que são baseados em Domain-Specific Modeling Languages (DSMLs). Esta pesquisa tem como escopo duas DSMLs (i* e OO-Method) que **não são** baseadas no padrão MDA (UML). Assim, o processo proposto nesta pesquisa é um processo MDD baseado em DSMLs.

Além disso, abordagens distintas de modelagem podem ser combinadas dentro de cada nível de abstração visando obter uma representação conceitual precisa a partir de diferentes perspectivas, por exemplo, perspectivas dinâmica, estrutural e de apresentação.

3.1.1 Padrões OMG e arquitetura MDA

O surgimento da arquitetura MDA em 2001 foi resultado da necessidade cada vez mais emergente de realizar manutenções em aplicações, integrá-las com outros sistemas, mudar suas infraestruturas, alterar seus requisitos e lidar com a frequente evolução e criação de novas tecnologias. Além disso, processos MDD têm como objetivo proporcionar os seguintes benefícios: produtividade, portabilidade, interoperabilidade, o que ainda representa um desafio atual. Para atingir esses objetivos e separar os níveis de abstrações MDA (MDA, 2003) foi definida pela OMG em três camadas conforme figura 35, tendo, na primeira camada de especificação (núcleo da arquitetura), padrões que ditam um conjunto de regras para estruturação da especificação expressa nos modelos e que não abordam características de plataformas específicas.

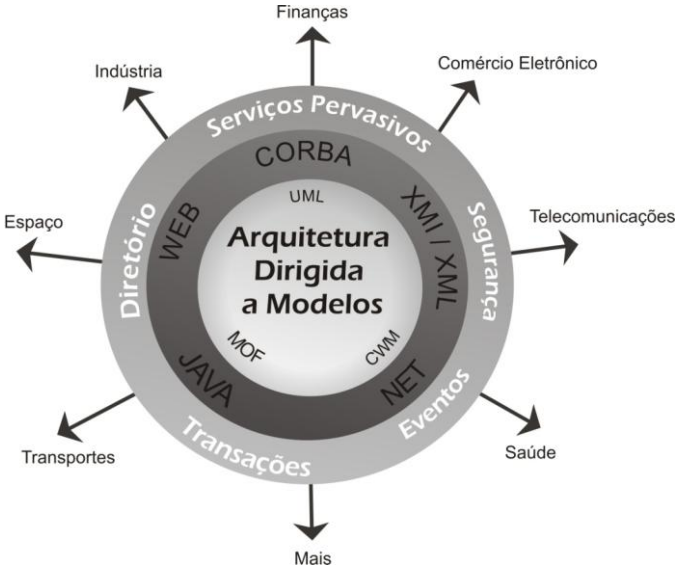


Figura 35. Padrões MDA

Esta camada representa o mundo (modelo) PIM. Os padrões também constituem definições propostas pela OMG. São eles:

- Unified Modeling Language (UML): padrão que define uma linguagem de modelagem geral orientada a objetos para especificação, construção e documentação de artefatos de sistemas complexos de software;
- Common Warehouse Metamodel (CWM): padrão para armazenamento de dados que permite fácil manipulação dos mesmos entre ferramentas e plataformas de armazenamento em ambientes heterogêneos distribuídos;
- Meta Object Facility (MOF): padrão que define uma linguagem abstrata para definição de linguagens de modelagem (metamodelos). Ela é utilizada para descrever modelos da UML, CWM e do próprio MOF, além de definir o formato de intercâmbio para modelos, base do padrão XMI (XML Metadata Interchange);

Para se ter uma linguagem de modelagem bem definida, é preciso determinar inicialmente a estrutura da linguagem. Essa estrutura é definida através de outra linguagem em mais alto nível, como por exemplo, MOF (*Meta Object Facility*) ou EMF Core (Ecore) (EMF, 2009). Estas linguagens, também conhecidas como meta-metamodelo, possuem uma gama de elementos bem definidos que são utilizados na concepção estrutural de linguagens de modelagem. Um meta-metamodelo utiliza os conceitos de classe, atributo, operação, parâmetro de relacionamento, e restrições herdados da linguagem UML especificamente dos diagramas de classe.

Para descrever a linguagem propriamente dita, é utilizado o mecanismo chamado de metamodelagem. Metamodelagem é o ato de utilizar a estrutura oferecida pelos meta-metamodelos para a definição de metamodelos. Metamodelo por sua vez define todos os elementos de modelagem e possíveis relacionamentos, ou seja, a sintaxe das linguagens de modelagem. Assim, um modelo deve estar em conformidade com o metamodelo, e um metamodelo de estar em conformidade com um meta-metamodelo A figura 36 a seguir mostra os quatro níveis de abstração de modelagem segundo a OMG. O nível mais alto M3 é o MOF, no nível M2 estaria o metamodelo de UML (sintaxe abstrata), no nível M1 estariam os modelos usando a notação UML (sintaxe concreta) e, no nível, mais baixo, M0, as instâncias desses modelos.

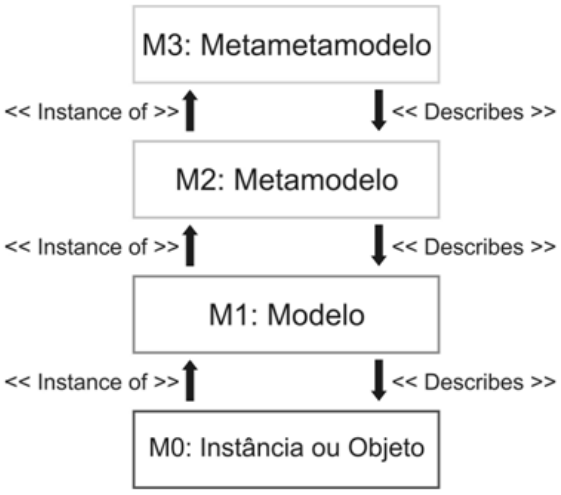


Figura 36. Níveis de abstração de modelos propostos pela OMG

Através de um metamodelo também é possível definir as restrições impostas em uma linguagem de modelagem. Restrições podem ser entendidas como um conjunto de regras que expressam como elementos de modelagem de uma determinada linguagem podem se relacionar, ou seja, é através da representação das restrições que a estruturação bem definida de uma linguagem é concebida em um metamodelo. Essas restrições podem ser expressas em OCL (Object Contraints Language).

O processo proposto nesta pesquisa é um processo MDD baseado em duas DSMLs (i* e OO-Method) que **não são** baseadas no padrão MDA. Assim, os dois metamodelos dessas DSMLs serão utilizados no processo de transformação: o i* wiki, apresentado na subseção 2.3.2 do capítulo 2 e o OO-Method, apresentado na seção 3.3.2 deste capítulo.

Na segunda camada (sentido dentro para fora) da figura 35 sobre Padrões MDA, encontram-se os modelos PSM que possuem características próprias a determinadas tecnologias e plataformas. Entre elas, algumas seguem padronização da OMG, elevando a resolução dos problemas de integração através da definição de especificações voltadas para interoperabilidade, que sejam abertas e independentes de fornecedores ou fabricantes específicos. São elas:

- XML Metadata Interchange (XMI): padrão para o intercâmbio de modelos através do mapeamento da linguagem definida pelo padrão MOF para o padrão XML do World Wide Web Consortium (W3C);
- Common Object Request Broker Architecture (CORBA): arquitetura que estabelece e simplifica a troca de dados entre sistemas distribuídos.

Na camada PSM da figura 35, pode-se ter também outros padrões como JAVA EJB, Microsoft, NET, etc.

Por fim, na camada mais externa da figura 35, são exibidos os serviços que a maioria dos domínios de aplicações necessita, para então serem apresentados os múltiplos domínios que fazem uso desses serviços. Esses serviços podem ser de segurança, persistência, controle de transações, tratamentos de eventos, etc.

3.1.2 Metamodelagem

Existe uma quantidade enorme de ferramentas para suportar transformação de modelos. Transformações podem utilizar diferentes técnicas que vão desde uma transformação manual, semi-automática e automatizada. Por exemplo, transformações de PIM para PSM podem ser realizadas através de uso de UML Profiles (extensões UML), uso de padrões (patterns), marcas (markings), metamodelos e transformações automáticas via algoritmo (MDA, 2003). Os elementos centrais dessas transformações são os mapeamentos dos modelos. Segundo a arquitetura MDA, um mapeamento é um conjunto de regras e técnicas utilizadas para modificar, refinar ou transformar um modelo e se obter um outro modelo. Esse mapeamento, usando-se metamodelos (modelos que descrevem e especificam os modelos originais), facilita a automação. Na figura 37 descreve-se o diagrama geral de um processo de desenvolvimento baseado na arquitetura MDA (MDA, 2003). Observa-se que os modelos PIM, PSM a serem transformados e técnicas de mapeamento são baseados e descritos através de seus respectivos metamodelos que, por sua vez, são expressos utilizando as tecnologias núcleo: UML, MOF ou CWM.

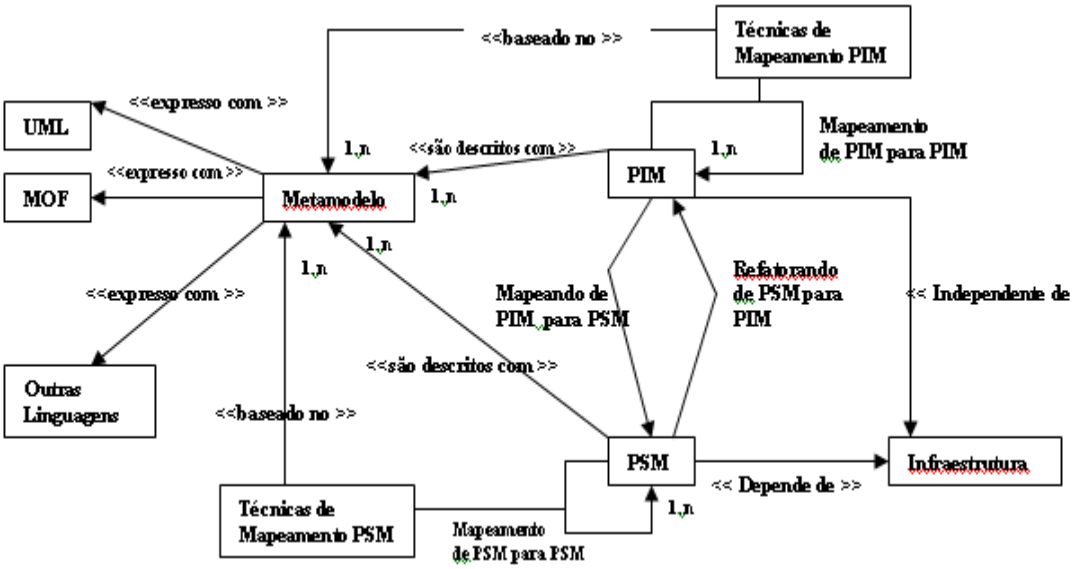


Figura 37. Transformações através de Metamodelos

Nesta figura, observa- se ainda que:

- Existe a possibilidade de mapeamento reverso (PSM para PIM) e de mapeamento no mesmo nível de abstração (PIM para PIM, PSM para PSM);
- O modelo PIM não depende de infraestrutura tecnológica ou plataformas específicas. Já o modelo PSM depende diretamente dessas infraestruturas;
- Só com a especificação dos modelos através de seus metamodelos, não há transformação, havendo a necessidade se definir o mapeamento entre esses modelos e a técnica utilizada para realizar a transformação;
- Não é contemplado nesse processo MDA, o próprio Modelo Independente de Computação (CIM) o que, sob a ótica da Engenharia de Requisitos é como se deixasse uma grande lacuna "gap" a ser preenchida ou, como se o próprio modelo PIM (nível de projeto em UML) se unificasse com o modelo CIM (UML), passando a idéia de ser um único modelo capaz de representar de modo completo e consistente todo um sistema.

Para ilustrar o esquema geral de transformações através de metamodelos, considere a figura 38 onde um modelo 1 é transformado num modelo 2, usando como entrada do processo o metamodelo A do modelo 1 e produzindo o modelo 2 expresso através do metamodelo B. Também, para realizar essa transformação, é necessário ter regras de mapeamento entre esses metamodelos. Conforme mencionado em (KELLY, 2008), este processo MDD de transformação pode ser baseado em UML (caso de MDA) ou baseado em Domain-Specific Modeling Languages. O processo MDD proposto OOM-NFR (ver capítulo 4) é baseado nas DSMLs i* e OO-Method.

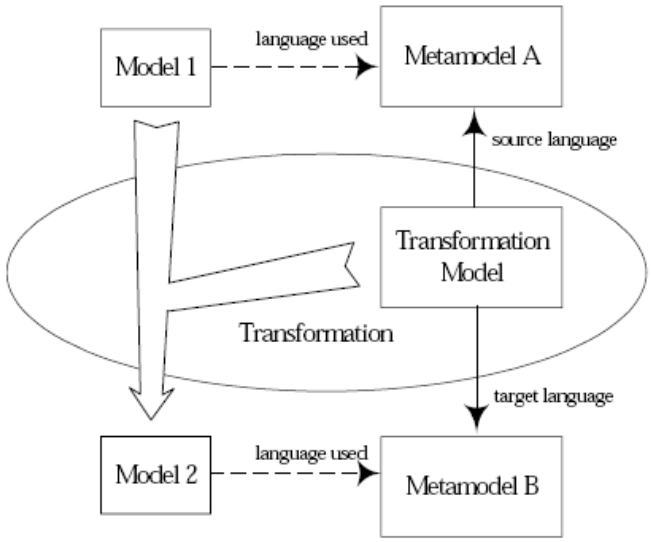


Figura 38. Transformações de mapeamentos por metamodelos

3.2 Abordagem OO-Method

Nesta seção serão vistas as características da abordagem de desenvolvimento de software MDD chamada OO-Method, ou abreviadamente OO-M. Os elementos e processo dessa abordagem são detalhados nas próximas seções.

3.2.1 Elementos e características do Modelo

O OO-Method (PASTOR et al., 2007) inova com o conceito de compilador de modelos que, de fato, é uma máquina virtual de transformação de modelos. Além disso, nessa abordagem o modelo de alto nível, chamado modelo conceitual, tem todos seus elementos (primitivas) descritos numa notação visual (gráfica) e são especificados numa linguagem formal orientada a objetos, chamada OASIS (PASTOR et al., 1992). Essas características fazem com que o OO-Method tenha precisão sintática e semântica suficientes para prover um ambiente capaz, inclusive, de fazer validação de modelos e, consequentemente, gerar um produto de software final de qualidade.

A primeira versão do OO-Method foi introduzida em 1992, através da tese de doutorado de Oscar Pastor (PASTOR, 1992). Deste então, o método incorporou vários componentes até chegar à versão apresentada neste trabalho. Segundo o seu criador, o método cobre todas as fases do processo de desenvolvimento de software, desde as fases iniciais de obtenção de requisitos e representação, passando pelo desenvolvimento do esquema conceitual OO correspondente, até a geração do produto final de software numa plataforma específica. O centro do desenvolvimento do software dirigido a modelos do OO-Method é o Esquema Conceitual, conhecido também como Modelo Conceitual, fundamentado a partir da seguinte afirmação do Prof. Antoni Olivé (PASTOR et al., 2007):

“Para desenvolver um sistema de informação é necessário e suficiente definir seu esquema conceitual”

Esta idéia também aparece em trabalhos e propostas de outros pesquisadores de prestígio. Toni Morgan (MORGAN, 2002) defende a idéia de usar Programação não Extrema (Extreme Non-Programming) em oposição à metodologia ágil XP, argumentando que a principal atividade no desenvolvimento de software é a modelagem, e não a programação,

pois a modelagem está no espaço do problema, enquanto programar está no espaço da solução. O objetivo final é tornar verdadeira a sentença "O Modelo é o Código" (MORGAN, 2002), em vez de "O Código é o Modelo". Tudo isso é possível de se obter quando se tem um Modelo Conceitual Executável que representa, de modo completo e consistente, todos os aspectos estáticos, dinâmicos e de interação (interface usuário) de um sistema, tal como é o modelo conceitual do OO-Method, passível de transformação através de um compilador de Esquema Conceitual.

3.2.1.1 O Processo Básico de Transformação.

OO-Method estabelece uma distinção clara entre o espaço do problema, onde está definido o esquema conceitual e o espaço da solução, onde é obtido o produto de software representado pelo esquema conceitual. Na figura 39, a primeira atividade da abordagem OO-Method é obter o esquema conceitual a partir dos requisitos, não importando o processo de engenharia de requisitos utilizado. De forma que esses requisitos sejam insumos para se criar (projetar) o esquema conceitual. Especificados os quatro modelos (ver seção 3.2.1.3) que compõem o esquema conceitual (modelo objeto, funcional, dinâmico e de apresentação), é gerado um repositório para conter todos os elementos (primitivas) especificados nos modelos desse esquema conceitual, utilizando-se a linguagem formal orientada objetos OASIS, conforme ilustrado na figura 39 a seguir.

Regras de mapeamentos das primitivas desse esquema conceitual para um modelo de aplicação específico de cada plataforma são definidas e, por fim, é realizada uma transformação automática desse modelo de aplicação para o código da aplicação correspondente à plataforma específica escolhida. No OO-Method, o modelo de aplicação é equivalente ao modelo específico de plataforma (PSM), conforme tabela 2, apresentada na próxima seção 3.2.1.2.

O processo garante que há uma equivalência funcional entre toda primitiva definida no esquema conceitual com sua(s) respectiva(s) primitiva(s) no modelo de aplicação. A implementação da abordagem OO-Method é suportada através da ferramenta OlivaNova da Care Technologies (OLIVANOVA, 2009). Utilizando essa ferramenta, como se observa na figura 39, o modelo conceitual do OO-Method é transformado para um modelo de aplicação (modelo específico de plataforma) constituído por uma arquitetura de três camadas: lógica da aplicação, persistência e interface com usuário. Nessas camadas, podem ser utilizados modelos e tecnologias específicos tais como: CORBA, ORACLE, XML, entre outros.

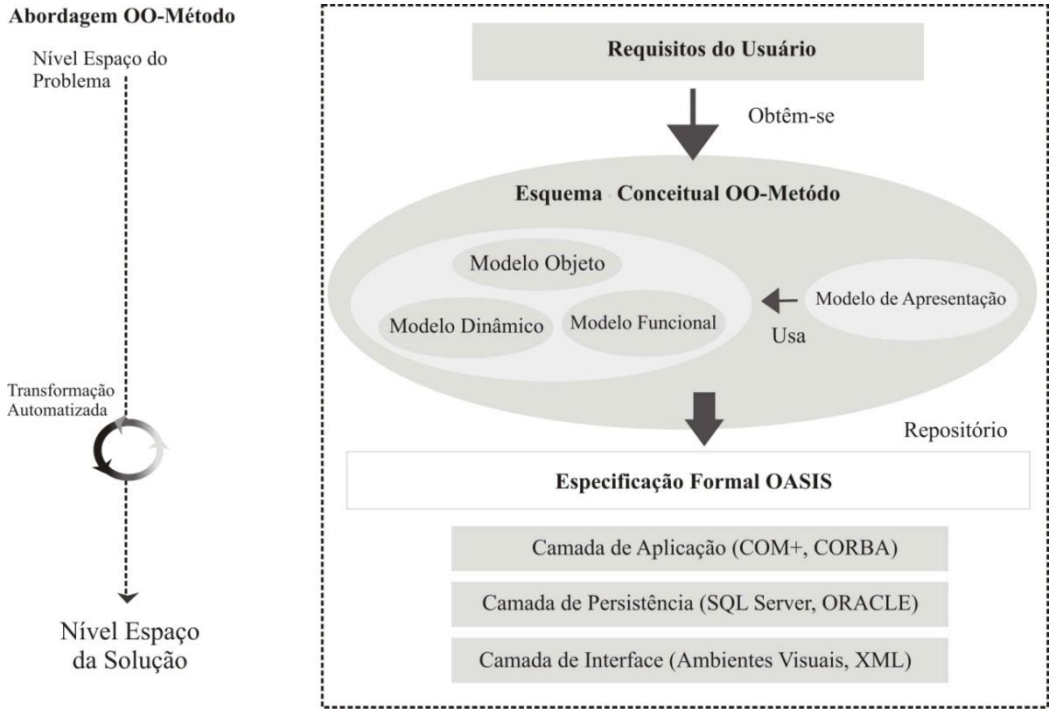


Figura 39. Abordagem OO-Method (PASTOR et al., 2007)

3.2.1.2 Comparação com MDA.

Os modelos do OO-Method, seus mapeamentos e transformações podem ser comparados com o padrão MDA baseado em UML, como ilustrado através da tabela 2. Entretanto, cabe-se destacar que esses modelos do OO-M são Domain-Specific Modeling Languages (DSMLs), segundo (KELLY, 2008), devido o fato deles não serem baseados em UML.

Tabela 2. Comparação do OO-Method com MDA (PASTOR et al., 2007)

MDA	**OO-Method**
Modelo Independente de Plataforma (PIM)	Modelo Conceitual
Modelo Específico de Plataforma (PSM)	Modelo de Aplicação
Modelo de Implementação (IM)	Código de Aplicação
Transformação PIM para PSM	Mapeamentos
Transformação PSM para IM	Transformações

Algumas propriedades do OO-Method estão ausentes no padrão MDA, tais como:

- O processo de compilação de modelos que, de fato, compila um modelo fonte (entrada) em outro modelo destino (saída), a partir de regras de mapeamento precisas, as quais são implementadas através de uma máquina virtual de compilação de modelos, onde se realiza uma verdadeira transformação desses modelos.
- Em termos de PIM, OO-Method provê uma solução semanticamente precisa, porque está especificado usando uma linguagem formal OASIS, que é computacionalmente completa. Em outras palavras, os modelos do esquema conceitual do OO-Method são também computacionalmente completos e contêm toda a informação que é necessária para gerar automaticamente código-fonte.

3.2.1.3 O Modelo Conceitual.

O modelo ou esquema conceitual é composto por quatro visões ou modelos: modelo objeto, modelo dinâmico, modelo funcional e modelo de apresentação.

O Modelo Conceitual do OO-Method utiliza uma notação visual (gráfica) similar a UML, formada por elementos necessários e suficientes para representar um sistema de informação. Além disso, a grande diferença, quando comparado com UML, é que o modelo conceitual do OO-Method tem uma semântica e sintaxe precisas e baseadas na linguagem formal OASIS. Isso propicia a validação automática do modelo conceitual, a fim de não deixar passar falhas (erros) para os modelos posteriores de mais baixo nível.

A seguir, serão apresentadas, de modo geral, as principais características desses modelos. Mais detalhes podem ser encontrados em (PASTOR et al., 2007).

- **Modelo Objeto**

Este modelo especifica as propriedades estáticas do sistema, definido pelo diagrama de configuração de Classe que é composto por:

- Classes
  - Atributos
  - Precondições e Serviços
  - Restrições de Integridade
- Relacionamento entre as Classes
  - Associação, Agregação e Composição
  - Herança
- Agentes

O OO-Method representa precisamente e formalmente (PASTOR et al., 2007), o que a linguagem UML não o faz, os relacionamentos dos seguintes tipos: associação, agregação e composição. A associação considera aspectos de cardinalidade, papéis e também de temporalidade. A temporalidade de uma associação se define como estática ou dinâmica. É estática quando os objetos associados não mudam durante o tempo ou ciclo de vida. É dinâmica quando as instâncias associadas mudam com o tempo durante o ciclo de vida dos objetos que participam do relacionamento.

A agregação é um tipo de associação mais restrita onde uma das classes é o todo e a outra representa a parte do todo (objetos agregados ao todo), significando que uma das classes é constituída ou formada por um conjunto de outros objetos agregados.

A composição é um tipo mais forte de agregação onde a parte e o todo estão vitalmente associados, ou seja, para um objeto-todo existir os demais objetos-parte também devem existir.

Além da precisão que o OO-Method trata os conceitos de associação, agregação e composição, ele também considera quais são os impactos que eventos de inserção, deleção e mudança de objetos têm sobre esses relacionamentos entre as classes.

O OO-Method suporta também os conceitos de Herança (generalização e especialização) e herança múltipla; e para gerenciar a complexidade do modelo, pode-se representar um subsistema através de uma notação visual igual à de um pacote (*package*) em UML.

O processo proposto OOM-NFR (ver capítulo 4) tem como escopo apenas este **modelo objeto** do modelo conceitual do OO-Method. Foi especificado um metamodelo simplificado ECORE deste **modelo objeto** (ver seção 3.2.2). Esse metamodelo foi simplificado para contemplar apenas os elementos passíveis de serem gerados a partir do modelo i* wiki e especificados nas diretrizes de transformações (ver seção 4.3 do capítulo 4).

Modelo Dinâmico

Este modelo representa o comportamento do sistema, especificando suas propriedades dinâmicas através de dois diagramas:

o Diagrama de Transição de Estado: especifica o ciclo de vida válido dos objetos de uma classe e seus serviços disponíveis em cada estado.

o Diagrama de Interação de Objeto: especifica as interações válidas entre os objetos através das transações, operações e gatilhos.

Um gatilho é uma condição sobre um estado do objeto que, tornando-se verdadeira, faz com que este objeto dispare eventos ou transações sobre si mesmo (*self*) ou sobre outros objetos (*object*) do sistema. A sintaxe de gatilhos na linguagem formal OASIS é:

<destination>::<condition>:<service>

Onde, **<destination> := self | object | class | for al**l

E "**self**" significa para a si mesmo, "**object**" para uma instância de outra classe, "**class**" para todas as instâncias da classe e "**for all**' para um subconjunto de objetos de uma classe.

- **Modelo Funcional**

Este modelo especifica o relacionamento entre os modelos objeto (que é estático) e dinâmico visto no tópico anterior. O modelo funcional é responsável pela:

-Definição da semântica relacionada às transições de estado

-Descrição de como a execução dos eventos muda o valor dos atributos das classes

O modelo funcional trata também eventos de criação e destruição de objetos, além de transações e operações que afetam os estados (valores dos atributos) dos objetos. O modelo funcional do OO-Method, combinado com sua linguagem formal provê de modo completo e preciso (PASTOR et al., 2007) uma solução para especificar os aspectos funcionais de um sistema via modelo conceitual.

O recurso de Semântica de Ações da UML 2.0 (UML, 2000), criado para suprir essa necessidade de modelagem de aspectos funcionais, não possui uma semântica definida de forma clara e precisa. O modelo funcional do OO-Method e sua linguagem formal OASIS proveem uma solução adequada, permitindo:

- Acesso a dados de acordo com o Modelo Objeto
- Definição de lógica sequencial
- Manipulação de classes e objetos
- Manipulação de relacionamentos
- Uso de operadores lógicos, aritméticos e relacionais

- **Modelo de Apresentação**

Este modelo especifica os requisitos de Interface de Usuário, modelando uma interface abstrata que é independente de plataforma ou dispositivo. O modelo de apresentação em nível de análise (conceitual) é considerado uma inovação do OO-Method em relação a outras abordagens que, na maioria das vezes, descrevem interface-usuário apenas em nível de implementação.

Enfim, ao se definir os quatro modelos básicos (objeto, dinâmico, funcional e apresentação) do esquema conceitual do OO-Method, tem-se toda infraestrutura necessária e suficiente para representar um sistema de informação no contexto do espaço do problema.

Para ser aderente ao padrão de desenvolvimento MDD, OO-Method precisa de alguma forma de transformar o modelo conceitual, que contém todas as propriedades estáticas, dinâmicas e de interação com usuário (apresentação), em um modelo de implementação (código). Essa transformação é automatizada através do Compilador de Modelos que pode ser visto como uma máquina virtual de programação. OO-Method inova com essa idéia de compilador de modelos, tornando-o diferente de outras abordagens que apenas focam na geração de código a partir de modelos utilizando técnicas diversas como integração, sincronização de código, refatoração etc.

A implementação do OO-Method é suportada através da ferramenta OlivaNova da Care Technologies (OLIVANOVA, 2009). O principal objetivo da OlivaNova é separar o que deve ser feito (espaço do problema), de como deve ser feito (espaço da solução). Ela é composta por duas principais ferramentas: o modelador e a máquina de transformação. No Capítulo 4 e 5 há os exemplos de modelos OO-Method gerados após processo de transformação.

3.2.2 O Metamodelo do OO-Method

O metamodelo OO-Method que será utilizado no processo MDD de transformação OOM-NFR é apresentado na figura 40 a seguir. Este é um metamodelo ECORE desenvolvido com o Eclipse GMF e metamodela apenas o **modelo objeto** (ver seção 3.2.1.3) do modelo conceitual do OO-Method. Os elementos desse metamodelo, necessários ao processo transformativo OOM-NFR, foram obtidos e adaptados a partir de (GIACHETTI, 2010). Entretanto, esse metamodelo, no padrão ECORE, é uma contribuição do autor desta pesquisa.

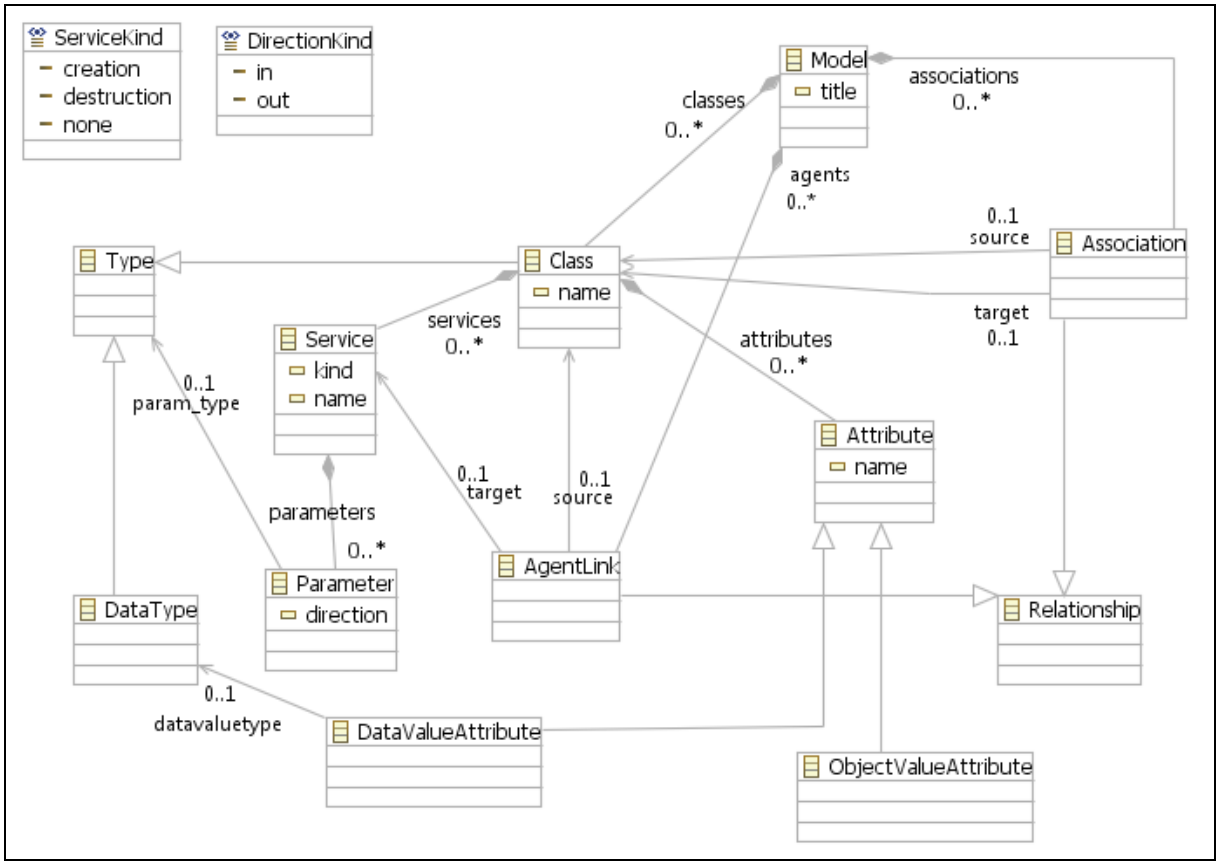


Figura 40. Metamodelo ECORE do OO-Method. Adaptado de (GIACHETTI, 2010).

Este metamodelo representa apenas o modelo objeto ou de classes do modelo conceitual do OO-Method. Os construtores ou primitivas deste metamodelo são:

- **Model (Modelo)** – representa o "palco" onde os elementos de modelagem **Model** é um repositório de classes, associações e agent links.

- **Class** – é a metaclasse que representa as classes. Possui o atributo "name" para denominar o nome da classe. Cada classe é composta por serviços do tipo **Service** e atributos do tipo **Attribute**.

- **Service** – representa os métodos da classe que, em OO-Method, são chamados de serviços. Esses serviços possuem parâmetros do tipo **Parameter** que podem ser de entrada ou saída (atributo **direction)**, conforme seu tipo especificado na Enumeração **DirectionKind.** Além disso, para se compatibilizar com UML, esse parâmetro está associado a um tipo **Type.** Já o atributo **kind** representa o tipo de serviço especificado pela Enumeração **ServiceKind** que indica se um serviço é um evento atômico de criação ou destruição de objetos ou simplemente uma transação.
- **Attribute** – esta metaclasse representa os atributos das classes. Para manter compatibilidade com UML, esse atributo pode ser do tipo dado simples (tipos simples) ou um tipo Objeto.
- **Association** – esta metaclasse representa uma associação entre classes e possui origem e destino do tipo **Class. Association** é uma herança da metaclasse abstrata **Relationship.**
- **AgentLink –** representa o relacionamento agent link que tem como origem uma classe cliente do tipo **Class** e, como destino, um serviço do tipo **Service** de outra classe (servidora). Este é um tipo de relacionamento especial do OO-Method onde uma determinada classe (cliente) define que tipo de visibilidade tem sobre um serviço de outra classe (servidora).

3.3 Transformações de i* para OO-Method

Conforme apresentado na introdução, alguns processos para transformação de i* em OO-Method já foram definidos em trabalhos anteriores. Esta seção mencionará três dos mais significativos pesquisados e tomados como referência e apoio no desenvolvimento desta dissertação. Entre os trabalhos pesquisados estão: (MARTINEZ, 2008), (ALENCAR et al., 2009) e (GIACHETTI, 2011). Cabe-se ressaltar que esses trabalhos têm como escopo apenas a geração do **modelo objeto (**ver seção 3.2.1.3) do esquema conceitual do OO-Method. Nas seções seguintes, será apresentada uma breve visão desses três trabalhos.

3.3.1 Esquema Conceitual a partir de Modelos Organizacionais

Uma das contribuições da abordagem (MARTINEZ, 2008) é estender o modelo i* SR Requisitos Final (SR late) com novos elementos chamados "Concerned Objects" que são associados aos elementos intencionais Recursos, Tarefas ou Metas (Goals) do i*.

Um "concerned object" representa de fato um recurso que está sendo utilizado em um processo organizacional, ou uma entidade abstrata de informação que será usada no sistema de software a ser desenvolvido "to-be". Um concerned object é representado no diagrama i* por um círculo e seus atributos por círculos menores associados ao concerned object.

Na figura 41 é ilustrado um exemplo do "concerned object" **Customer** que estende o recurso "Customer data". Além disso, como processo inicia com o modelo SR final do i*, o modelo de entrada deve sempre ter o ator sistema (SSA " software system actor").

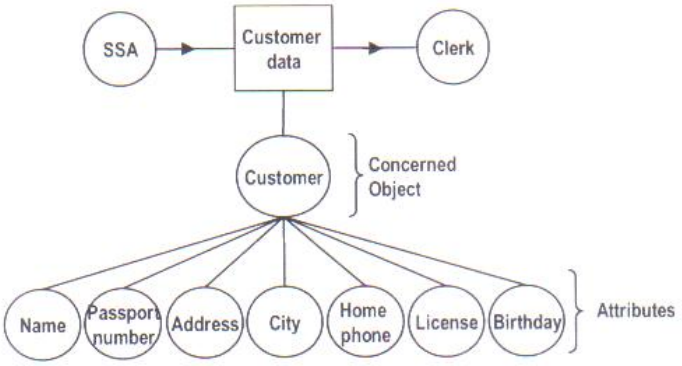


Figura 41. Notação para Concerned Object (MARTINEZ, 2008)

Após a extensão do modelo i* para concerns objects é gerada também uma tabela contendo os cenários desses concerns objects. Cada cenário descreve como esses concerns objects estão associados com outros elementos do novo modelo i* estendido. O processo define regras e algoritmos para identificação dos concerned objects e criação dos cenários.

A próxima atividade do processo é a otimização dessa tabela de cenários, visando classificação e modularização dos concerned objects.

A fase final do processo é a definição das regras de transformação que toma com base o modelo e cenários de concerned objects para identificação e geração do modelo OO-Method. São especificadas regras para geração de classes, serviços, atributos, relacionamentos de associação, agregação e herança. Inclusive nesses relacionamentos o processo identifica também cardinalidade e restrições (pré e pós-condições). Para mais detalhes dessas regras e abordagem, consultar (MARTINEZ, 2008).

O forte desse processo é que ele introduz o novo elemento ao modelo i* que ajuda principalmente no detalhamento dos recursos, o que é fundamental para construção das classes e serviços do modelo conceitual do OO-M.

Como desvantagem essa abordagem não utiliza metamodelagem, nem técnicas de transformações utilizando metamodelos, o que torna o processo transformativo muito custoso do ponto de vista de processamento computacional. Além disso, não abrange requisitos não-funcionais.

3.3.2 De Requisitos em i* para Modelos Conceituais

Trabalhos pioneiros visando integrar modelos requisitos GORE a modelos conceituais PIM podem ser encontrados em pesquisas de (ALENCAR, 1999), (CASTRO at al., 2000), (CASTRO at al., 2001) e (ALENCAR et al., 2003). Esses trabalhos contemplavam, de modo geral, a transformação de i* em Orientação Objeto (UML diagrama Classes), mapeando softgoals i* em atributos enumerados das classes geradas em UML, mas não focavam na análise prévia desses softgoals, no sentido de satisfação, priorização e resolução de conflitos, visando gerar um modelo de mais baixo nível de abstração que fosse reflexo dessa análise de softgoals.

Entretanto, o principal trabalho, no qual se apóia o desenvolvimento dessa dissertação, foi apresentado em (ALENCAR et al., 2009) que propõe um novo processo transformativo MDD de requisitos em i* para o modelo objeto conceitual OO-Method.

Esse processo, conforme ilustrado na figura 42, prevê um conjunto de atividades que vão desde a seleção de elementos i* a serem transformados, passando por aplicação de regras de transformação para geração de classes, atributos, serviços e relacionamentos; até gerar finalmente um modelo OO-Method. É de se frisar que algumas atividades de processo são automatizáveis, outras semi-automatizáveis e outras manuais.

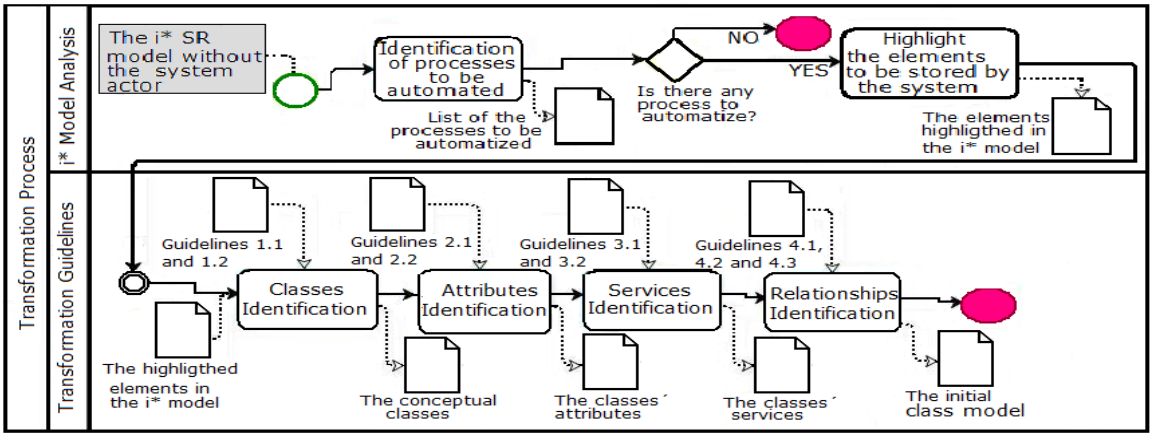


Figura 42. Processo de Transformação (ALENCAR et al., 2009)

A partir de algumas limitações observadas nesse processo, um trabalho posterior (ALENCAR et al., 2010a) propôs algumas soluções, tais como:

- Impossibilidade de inferir se o elemento intencional recurso corresponde a uma entidade informacional ou física. Esse problema poderia ser resolvido com definição de um metamodelo extendido de i*, onde pudesse ser modelada essa informação.
- Impossibilidade de rastreabilidade, pois um ou mais tipos de elementos do modelo i* estava sendo transformado num mesmo tipo de elemento Class do OO-Method. Esse problema poderia ser resolvido, utilizando-se um modelo intermediário entre i* e OO-Method.
- Algumas informações relevantes do modelo i* poderiam ser perdidas no processo transformativo, ou seja, após o modelo de classe gerado, seria impossível identificar quais elementos OO-Method estavam relacionados com alguns elementos do i* tais como: depender, dependee, dependum, tarefas envolvidas nas decomposições de tarefa, serviços que correspondessem às tarefas dos means-end, etc. Esse problema poderia também ser resolvido, conforme descrito no trabalho, usando modelos intermediários ou auxiliares.
- Não era possível diretamente especificar quais elementos do modelo i* deveriam ser automatizados. Esse problema poderia ser resolvido com definição de um metamodelo extendido de i* onde pudesse ser modelada essa informação.

Uma proposta de utilização de metamodelos intermediários foi consubstanciada em (ALENCAR et al., 2010b). A partir daí, essa abordagem de utilização de metamodelos intermediários passou a ser explorada em trabalhos de pesquisas.

Além dessas limitações já mencionadas, essas abordagens também não contemplam análise dos softgoals para transformação do modelo i* em OO-Method. Essa limitação ocasiona o problema de se gerar um modelo OO-M que não é capaz de refletir as prioridades dos softgoals, nem capaz de dizer se esses softgoals estão satisfeitos e sem conflitos.

3.3.3 Metamodelos Intermediários

Em (PASTOR et al., 2010) é apresentado um processo transformativo que utiliza metamodelos intermediários, também chamados de metamodelos de integração. Depois, em (GIACHETTI, 2010), esse mesmo processo é utilizado numa abordagem para demonstrar a interoperabilidade entre os modelos i* e OO-Method, realizando-se um experimento de transformação.

A idéia fundamental de se utilizar metamodelos intermediários ou de integração, é reduzir o "gap" semântico entre os metamodelos de origem e destino, facilitando assim as transformações. Segundo (GIACHETTI, 2010), para viabilizar MDD é necessário formas eficientes e sistemáticas de estender o metamodelo de entrada com novos elementos a fim de facilitar o processo transformativo. Para atingir esse objetivo em (PASTOR et al., 2010), é definido um processo básico de geração automática de extensões de metamodelos, utilizando-se modelos de integração modelos intermediários, como ilustrado na figura 43.

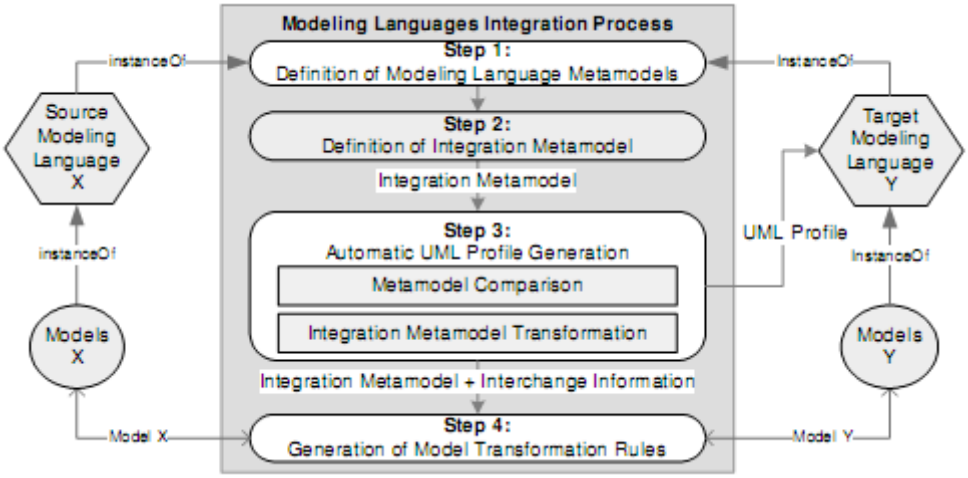


Figura 43. Processo automático de extensão e geração de modelos a partir de modelo de integração

Nesse processo especifica quatro passos onde são definidos metamodelos dos modelos de origem e destino, um modelo intermediário ou de integração, geração automática de UML Profile, regras de transformação para o modelo alvo a ser gerado.

O Processo Genérico Automatizável para ligar i* ao modelo conceitual do OO-Method é definido a seguir:

- **Passo 1:** Definição das Diretrizes (Guidelines) de Transformação, partindo dos metamodelos de i* e OO-Method: Nesta etapa definem-se os metamodelos de entrada (inicial) e saída (final), ou seja os metamodelos i* e OO-M respectivamente. O processo utiliza, conforme trabalho (ALENCAR, 2009), diretrizes necessárias para identificação das Classes, Atributos, Serviços e Relacionamentos gerados em OO-M a partir do modelo i*.
- **Passo 2:** Definição do Modelo de Requisitos (OO-M Req. Model): Definem-se as metaclasses que são relevantes e necessárias para implementação e processamento das Diretrizes de Transformação-Guidelines especificadas no Passo 1. Esse metamodelo intermediário é obtido a partir do metamodelo de entrada i*, adicionando informações extras/adicionais e é o modelo mais próximo do modelo de conceitual inicial (classes) do OO-Method, passível de automatização das diretrizes.
- **Passo 3:** Validação do Modelo de Requisitos: Nesta etapa, o modelo OO-M Req. Model é validado, principalmente as validações das informações extras (adicionais), tais como se um recurso é físico ou informacional, etc. Geralmente essa validação é realizada usando a linguagem OCL. A principal razão dessa validação é não propagar erros, nem realizar transformações incorretas para o modelo final (alvo).
- **Passo 4:** Aplicação da Abordagem de Integração: Neste passo é definido o Metamodelo de Integração que é o modelo i* estendido (enriquecido com informações extras), ou seja, é um intermediário entre i* original de entrada e o Modelo de Requisitos definido no passo 2. O modelo de integração é construído através do processo básico de geração automática de extensão de metamodelos, usando UML Profile. Esse processo é o mesmo ilustrado na figura 59 anterior. Esse passo final é o passo principal e mais complexo do processo genérico de transformação. Em (GIACHETTI et al., 2008) e (GIACHETTI et al., 2009) é definido um processo para geração dese Modelo de Integração.

Apesar de suas vantagens, esta abordagem torna-se mais complexa que as outras anteriormente citadas, pois, com a especificação de metamodelos intermediários, há necessidade de se definir mais tabelas de mapeamento (regras de transformações) entre os metamodelos, o que aumenta a probabilidade de ocorrência de erros e perda de informação durante o processo transformativo.

Além disso, a sistemática para geração de Modelo de Integração proposta em (GIACHETTI et al., 2009) é um ponto crítico dessa abordagem, pois tem aspectos subjetivos que dependem do modelador. Essa abordagem também não contempla os requisitos não-funcionais.

Sendo assim, devido a esses fatores, apesar desta abordagem (metamodelos intermediários) ser considerada uma evolução da abordagem de (ALENCAR et al., 2009) no que se refere à redução de "gap semântico" (aspecto que ainda precisa ser mais validado), é que a abordagem de (GIACHETTI, 2010) não foi utilizada nesta pesquisa.

Por fim, como o processo MDD proposto OOM-NFR é baseado em duas DSMLs (i* wiki e OO-Method), que estão em níveis de abstração bastante distintos e têm também diferentes propósitos, o autor desta pesquisa acredita que, independentemente de se utilizar modelos intermediários, o "gap semântico" entre esses modelos é uma **constante**, pois haverá sempre um conjunto de elementos específicos dessas DSMLs impossíveis de interoperar, ou seja, incapazes de serem utilizados ou gerados (derivados) nas transformações (ver capítulo 6, seção 6.2-limitações).

3.4 Considerações Finais

Este capítulo apresentou uma visão geral da arquitetura dirigida a modelos- MDA que é um padrão do OMG. Como o foco da MDA é a transformação de modelos, foram vistos os termos CIM, PIM, PSM e metamodelos. Alguns padrões, como o MOF foram ilustrados e também foi descrito o processo geral de transformação, utilizando metamodelos. Foi feita a distinção entre processo MDD que são baseados em MDA (UML) e processos MDD que são baseados em DSMLs. Na seção 3.3 foi mostrada a abordagem OO-Method: seus elementos e características. Foi definido o metamodelo do modelo objeto da DSML OO-M que será utilizado no capítulo 4 como metamodelo para o processo de transformação proposto.

Na seção 3.4, foram elencados três trabalhos de pesquisa de grande importância na área de transformação de modelos i* para OO-M. Os principais aspectos desses processos foram detalhados.

Entretanto, para se obter um software de qualidade, faz-se necessário também considerar os seus requisitos não-funcionais e os processos vistos na seção 3.4 são insensíveis a essa categoria de requisitos. Simplesmente esses processos ignoram os softgoals, ou seja, os modelos OO-M gerados não são capazes de refletir os requisitos não-funcionais de produto especificados no nível de requisitos em i* e nem há um tratamento de conflitos e prioridades para os RNFs.

No próximo capítulo, será visto como isso pode ser realizado através da proposta de um processo de transformação de modelos (MDD) que contempla softgoals.

CAPÍTULO 4

4. O Processo OOM-NFR: Transformando i* para OO-Method

Este capítulo tem como objetivo descrever o processo de transformação proposto. O capítulo está dividido nas seguintes seções: visão geral do processo OOM-NFR; descrição do pré-processamento, onde é realizada a análise e tratamento dos softgoals (requisitos não-funcionais de produto internos ao ator sistema); exemplos de análises NFR; visão do processo núcleo através da descrição detalhada das regras de transformação; resultados e considerações finais sobre a proposta apresentada.

4.1 O Processo

Nesta seção é apresentado o processo **OOM-NFR,** de transformação de i* para o modelo objeto (ver seção 3.2.1.3) do OO-Method, com foco em softgoals. Cabe-se ressaltar que os softgoals i*, considerados nesta pesquisa, estão relacionados a RNFs de produto (SOMMERVILLE, 1998) (ver seção 2.2) e são internos ao ator sistema do modelo i* (ver seção 2.3). A figura 44 representa este processo, na notação Business Process Model and Notation (BPMN, 2009), desenhado com a ferramenta Bizagi (BIZAGI, 2010).

O processo OOM-NFR pode ser melhor compreendido como sendo composto basicamente por dois subprocessos chamados de **Pré-Processamento** e **Processo Núcleo.**

No Pré-Processamento são realizadas todas as atividades necessárias à análise de softgoals, tratamento de prioridades e conflitos destes requisitos. Esses softgoals são relevantes, pois representam atributos que têm impacto na qualidade do produto do software final, sendo assim, o pré-processamento foca na análise, resolução e tratamento destes.

Já o Processo Núcleo realiza todas as atividades de transformação dos elementos i* para os construtores do **modelo objeto** do OO-Method. É importante salientar que o modelo objeto é apenas um dos modelos do esquema conceitual do OO-Method (ver seção 3.2.1.3) e que o processo OOM-NFR tem como escopo apenas esse modelo objeto**.**

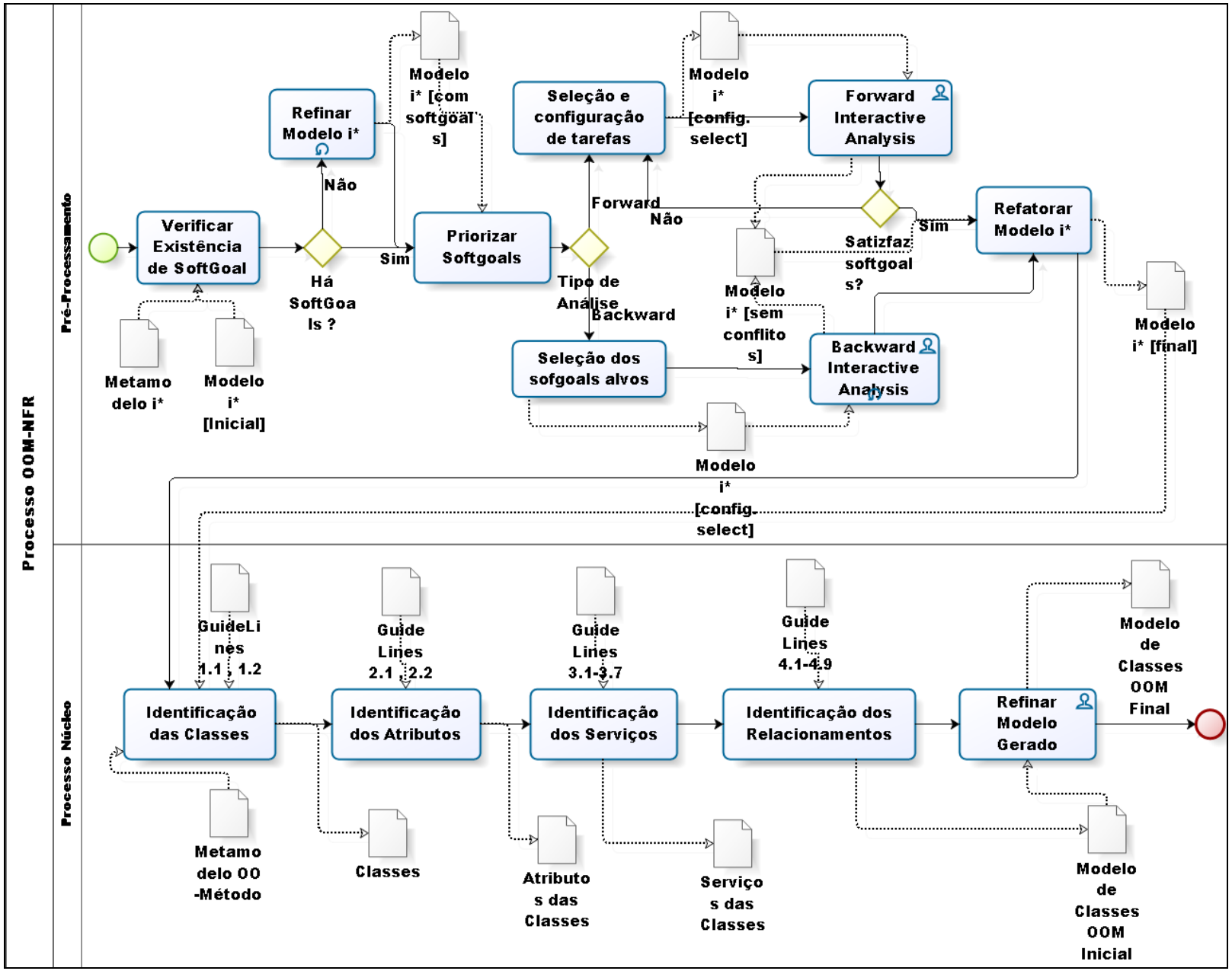


Figura 44. Visão geral do processo OOM-NFR

Quando comparado com o processo original proposto em (ALENCAR et al., 2009), o processo OOM-NFR apresenta algumas diferenças fundamentais, tais como:

- O OOM-NFR trata requisitos não-funcionais através de um pré-processamento;
- No OOM-NFR não há necessidade do analista escolher ou definir o(s) processo(s) de negócios a serem automatizados;
- No OOM-NFR não há necessidade de destacar os elementos do(s) processo(s) de negócio(s) selecionado(s) a serem transformados.
- No OOM-NFR foram adicionadas três novas diretrizes: **4.7, 4.8 e 4.9** que especificam a identificação dos relacionamentos.

Após esta breve apresentação do processo OOM-NFR, serão detalhados nas próximas seções 4.2 e 4.3, esses dois processos: Pré-Processamento e Processo Núcleo respectivamente.

4.2 O Pré-Processamento

Para melhor visualizar graficamente e descrever os subprocessos **Pré-Processamento** e **Processo Núcleo** do **OOM-NFR**, foram realizados dois recortes da figura 44 anterior que ilustra o processo OOM-NFR. A figura 45 ilustra o primeiro recorte (Pré-Processamento) a ser detalhado nesta seção, enquanto o segundo recorte (Processo Núcleo) será apresentado e detalhado na seção 4.3 deste capítulo.

Segundo a BPMN, as atividades ou tarefas são elementos básicos que constituem os processos, sendo que as tarefas são as atividades atômicas. Assim, para evitar confusão e ambiguidade com o termo tarefa do elemento do i*, será apenas utilizado o termo atividade para descrever as etapas do processo.

O **Pré-Processamento** é composto pelas seguintes atividades básicas:

- **Verificar Existência de SoftGoal**

  A partir do modelo SR do i* wiki (GRAU et al., 2008) gerado com a ferramenta IstarTool e representado pelo objeto de dados **Modelo i*[Inicial]** (figura 45), essa atividade verifica se o modelo i*, expresso em XML Metadata Interchange (XMI) e passado para o processo transformativo OOM-NFR, tem elementos intencionais i* do tipo softgoal. Um **objeto de dados**, representado em BPMN com um ícone de uma folha de papel branca dobrada na borda superior direita, indica a entrada ou saída de dados, relacionada às atividades do processo. Essa verificação deve ser realizada, pois se o modelo i*

de entrada não possuir softgoals, ou seja, elementos que representam os requisitos não-funcionais de produto internos ao ator sistema, a atividade de **Refinar Modelo i*** deve ser realizada para que se possa dar prosseguimento ao pré-processamento do OOM-NFR.

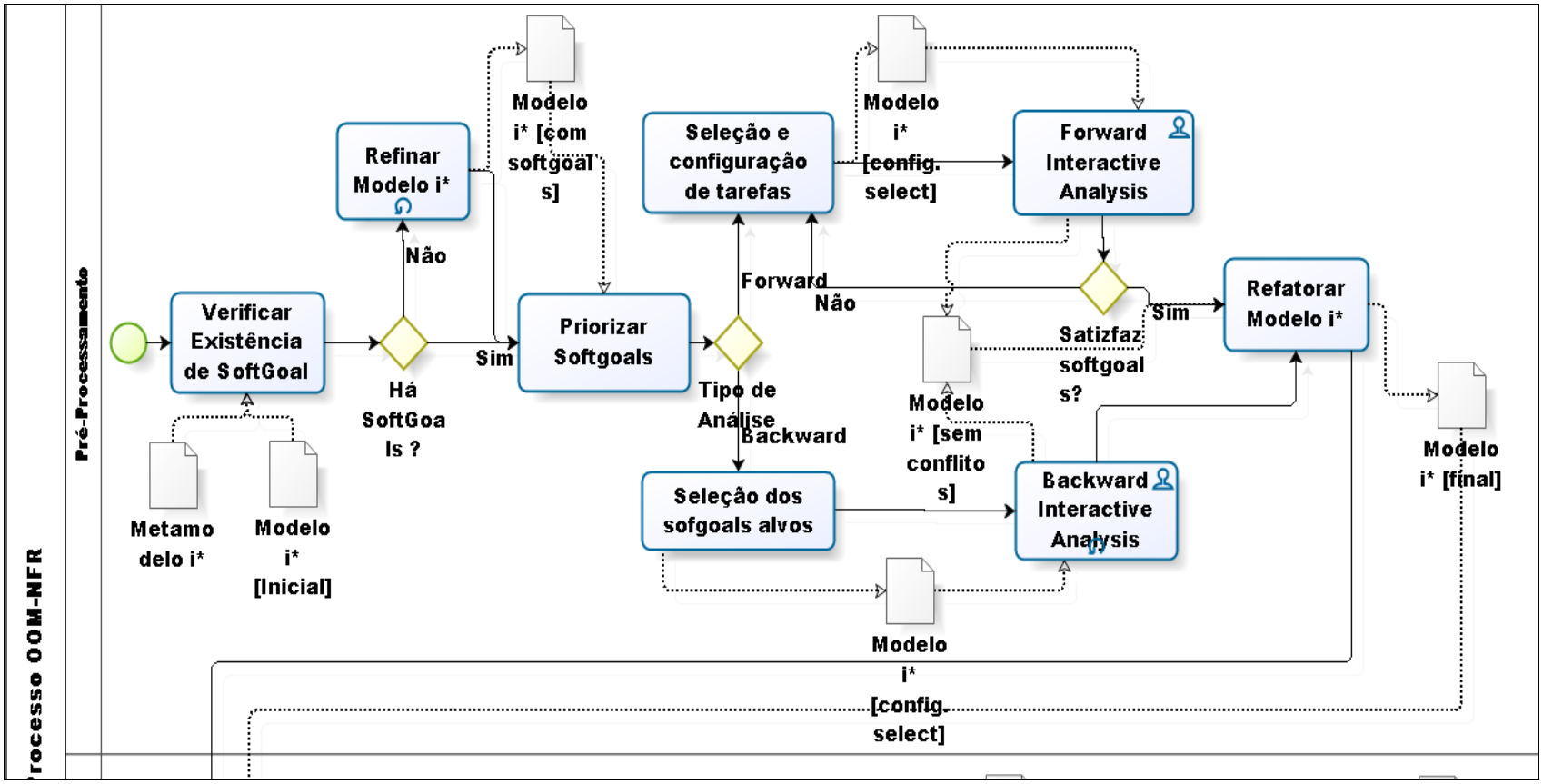


Figura 45. O Pré-Processamento - recorte do OOM-NFR

Sendo assim, há necessidade de se realizar a atividade **Refinar Modelo i*,** a partir da qual o analista modificará o modelo i* de entrada para conter softgoals.

Esta atividade de **Verificar Existência de SoftGoal** recebe também como entrada o **metamodelo i*** wiki (seção 2.3.2), pois, na automatização do processo transformativo OOM-NFR na linguagem QVT, há necessidade de se especificar inicialmente o metamodelo i* em ECORE que vai ser utilizado para processar o XMI (modelo i* de entrada). Caso esse modelo de entrada não seja compatível com o metamodelo declarado, a linguagem QVT não consegue "compilar" o modelo, gerando erro de incompatibilidade na execução processo transformativo, pois, no início do código QVT (ver Apêndice), é declarado o tipo desse metamodelo como sendo do tipo i* wiki. A saída dessa atividade é um modelo i* com softgoals.

- **Priorizar Sofgoals**

  Para que, nas próximas atividades do processo, o analista de requisitos possa melhor interagir com o processo de resolução "Forward/Backward Interactive Analysis", setando o nível de satisfação desejado dos softgoals do modelo e tomando decisões focadas na importância que cada softgoal tem no seu projeto, é necessário que ele antes decida informalmente quais as prioridades que estes softgoals terão de acordo com o que foi negociado com os *stakeholders*. Saber essas prioridades antecipadamente ajudará ao analista a selecionar determinados elementos do modelo i* em detrimento de outros, além de tomar melhores decisões (julgamentos) solicitadas pelo processo de análise NFR. Nas seções 4.2.2, 4.2.3 e 4.2.4, antes da realização de cada análise NFR, primeiramente são definidos quais softgoals serão prioritários e quais não são prioritários para cada cenário de análise. Essa definição das prioridades dos softgoals corresponde a esta atividade **Priorizar Sofgoals.**

- **Escolher Tipo de Análise NFR**

  Nesta atividade, o analista opta pelo tipo de análise NFR que será realizado. A abordagem de tratamento de softgoals utilizada, conforme seção 2.4, pode ser "Forward" ou "Backward" (HORKOFF et al. 2010).

- **Seleção e configuração de tarefas**

Caso o analista opte pelo processo de resolução e tratamento de conflitos direto "Forward" (ver seção 2.4.1), que é bottom-up, a primeira atividade que ele deve fazer é, partindo das folhas do diagrama i*, selecionar uma configuração [**config. selec**t], constituída de tarefas e seus respectivos níveis de satisfação (negado, satisfeito, entre outros). A partir daí, ele poderá rodar o processo de análise, que é a próxima atividade do pré-processamento, e verificar se os softgoals prioritários do projeto serão ou não satisfeitos. Na figura 45, essa atividade gera o objeto de dados **Modelo i*[config. select].**

- **Executar "Forward Interactive Analysis"**

  Nesta atividade é executada a análise interativa para frente, ou seja, o analista executa a abordagem "Forward", implementada através de um SAT-Solver, conforme ilustrado na seção 2.4. Durante esta atividade poderá haver várias interações humanas, caso o resolvedor de fórmula lógica não consiga automaticamente realizar a resolução. Essas interações humanas são necessárias até se obter uma configuração i* que satisfaça aos softgoals prioritários e não haja conflito, alcançando-se o nível de satisfação (satisfatibilidade) desejado. Essa situação é representada pelo ponto de decisão (gateway BPMN) nomeado por: **Satisfaz Softgoals?**

  A partir daí, caso essa situação seja verdadeira, tem-se um **modelo i* com conflitos resolvidos e RNFs priorizados,** representado na figura 45 pelo objeto de dados **Modelo i*[sem conflitos].** Caso a situação seja falsa, o analista deverá novamente repetir a atividade de **seleção de configuração de tarefas** e reexecutar esta atividade **"Forward Interactive Analysis".** Como visto na seção 2.4.1, para modelos i* grandes e com várias alternativas, essa repetição torna-se bastante tediosa, sendo mais indicada, nesses casos, a análise "backward".

  Este **Modelo i*[sem conflitos],** analisado em termos de softgoals, tendo os elementos i* destacados (ver exemplos nas seções seções 4.2.2, 4.2.3 e 4.2.4) com o nível de satisfatibilidade alcançada (satisfeito, não satisfeito, etc), representa uma configuração base encontrada que o modelo OO-Method a ser gerado deverá refletir**.** Esses elementos destacados (rotulados) são todos elementos intencionais do i*, entretanto o processo foca, principalmente, os softgoals e tarefas.

- **Executar "Backward Interactive Analysis"**

  Esta abordagem de resolução e tratamento de conflitos admite também a forma para trás ou "Backward" (seção 2.4), que é top-down e, segundo (HORKOFF et al. 2010), é mais eficiente, completa e automática que a abordagem "forward", no sentido de ter capacidade de responder as questões "Como" e "Por que" das decisões escolhidas durante o processo de resolução. Esta atividade deve ser precedida por outra atividade: **Seleção dos sofgoals alvos** e gera, conforme a figura 45, o objeto de dados **Modelo i*[config. select].**

  Nesta atividade são selecionados inicialmente os softgoals a serem priorizados e tratados os conflitos, rotulando esses softgoals com seus predicados (níveis) de satisfação. Esta atividade parte dos elementos softgoals raízes do diagrama e se propaga para os elementos folhas até se encontrar, iterativamente e interativamente, uma configuração de elementos i* que satisfaçam os softgoals prioritários e esteja sem conflitos.

  .

  Nesta atividade, também é gerado, como saída, um **Modelo i*[sem conflitos]** (figura 45)**,** ou seja, um modelo i* com os conflitos resolvidos e que satisfaça os softgoals prioritários. Esse modelo, onde os elementos i* estão destacados com rótulos do nível de satisfatibilidade alcançada (satisfeito, não satisfeito, etc), será passado para a próxima atividade do pré-processamento**.** Esses rótulos são colocados em todos elementos intencionais do i*, entretanto o processo foca principalmente os softgoals e tarefas. Nas seções 4.2.2, 4.2.3 e 4.2.4 adiante serão ilustrados exemplos detalhados dessa atividade.

- **Refatorar Modelo i*, extraindo Tarefas e Recursos negados da Configuração Selecionada**

  Esta atividade deve refatorar o modelo i* analisado, extraindo os elementos intencionais tarefas e recursos folhas totalmente negados, ou seja, aqueles elementos que estão com o rótulo "**X**" e que, durante o processo de resolução para satisfazer os softgoals prioritários, não foram selecionados pelo analista ou pelo resolvedor SAT-Solver. Esse elemento negado significa que ele não deve estar presente no modelo i* final do pré-processamento e, consequentemente, deve ser removido. Essa atividade não pode deixar o

diagrama i* inconsistente e inválido, por isso, extrai apenas elementos folhas. Os demais elementos do diagrama de i*, necessários à satisfação dos softgoals prioritários e não conflitantes, permanecem no **modelo i* final** sem os respectivos rótulos, pois estes rótulos não serão mais utilizados durante o próximo subprocesso do OOM-NFR: o processo núcleo.

Após essa última atividade do **pré-processamento** OOM-NFR é gerado um **modelo i* sem os elementos negados** que**,** na figura 45, é representado pelo objeto de dados **Modelo i*[final]** para ser finalmente passado para o **processo núcleo** realizar as transformações.

4.2.1 O Modelo Agência Fotográfica em i* wiki

Para melhor exemplificar o pré-processamento do OOM-NFR, foi escolhido um sistema de agência fotográfica para ser modelado. Para fins do propósito do processo MDD deste trabalho, o modelo apresentado será o SR na fase final de requisitos. Esta fase considera que o sistema está no nível mais maduro de requisitos, existindo um ator "sistema" responsável por realizar as metas, os softgoals, tarefas e recursos, além de se relacionar intencionalmente com os demais atores da organização.

A figura 46 mostra o sistema de agência fotográfica completo modelado em i* wiki SR final. Este modelo é um refinamento do modelo SR inicial especificado originalmente em (ALENCAR et al., 2009). Nesse refinamento, foram adicionados softgoals, pois, no modelo original (ALENCAR et al., 2009), não havia softgoals. Isso foi necessário porque o processo MDD de transformação proposto OOM-NFR precisa de softgoals. A razão de se refinar o sistema o sistema de agência fotográfica em i* wiki SR final, foi devido ao fato do modelo SR final ter o **ator sistema** inserido e estar mais próximo, em termos de nível de abstração, do modelo OO-Method. Isso facilita a análise dos softgoals prioritários e as transformações.

Este diagrama do sistema de agência fotográfica, ilustrado na figura 46, foi modelado inicialmente com a ferramenta Organization Modelling Environment (OME, 1998). Entretanto, por ser um diagrama extenso e ter ficado ilegível, quando transposto para este documento, teve de ser redesenhado com a ferramenta da Microsoft Visio (VISIO, 2010) a fim de melhor visualização.

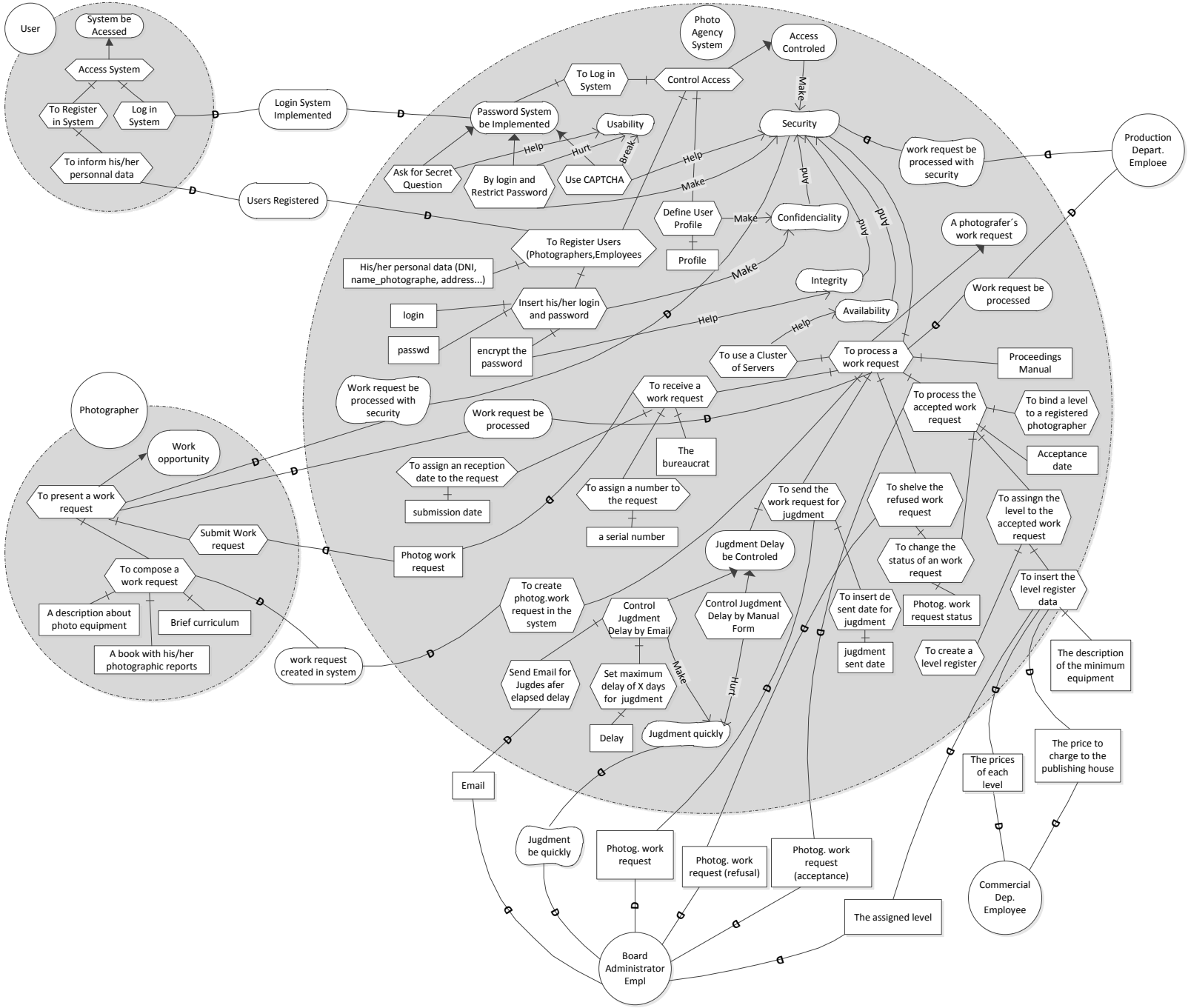


Figura 46. Sistema Agencia Fotográfica. Refinamento de (Alencar, 2009)

- **Requisitos iniciais do Sistema Agência Fotográfica**

O sistema de agência de fotografia, inicialmente especificado em (MARIN, 2008) é dedicado à gestão de reportagens fotográficas e sua distribuição para editoras. Esta agência opera com fotógrafos "freelance" que devem apresentar um pedido para o departamento de produção da agência de fotografia. Este pedido contém: informações pessoais do fotógrafo, uma descrição sobre o equipamento de sua propriedade, um breve curriculum vitae e um livro "do inglês book" com os relatórios fotográficos realizados. Um fotógrafo aceito é classificado em um dos três níveis possíveis nos quais um equipamento mínimo de fotografia é exigido. Para isso, o departamento técnico cria um novo registro para o fotógrafo, e salva-o no arquivo do fotógrafo. Para cada relatório de foto apresentado por um fotógrafo um novo registro com um código seqüencial é criado. Este registro tem o preço que as editoras têm de pagar para a agência, que é estabelecido de acordo com o número de fotos e nível do fotógrafo. Além disso, este registro tem uma anotação descritiva sobre o conteúdo do relatório fotográfico. O departamento comercial estabelece de acordo com o nível dos fotógrafos, o preço que será pago para os fotógrafos e o preço que será cobrado à editora por cada foto.

A partir desta descrição dos requisitos iniciais, foi especificado um novo modelo SR final para atender e validar as transformações propostas nesta dissertação, conforme já ilustrado na figura 46 anterior, que será explicado a seguir:

- **Refinamento do Sistema**

No modelo SR final, conforme ilustrado na figura 46, algumas tarefas anteriormente alocadas aos atores fotógrafo, departamento de produção, departamento comercial, departamento técnico e de julgamento foram alocadas ao ator Sistema Agência Fotográfica (PAS). Os atores que não são sistema precisam fazer registro no sistema e se autenticarem a fim de poder utilizar o sistema de agência fotográfica. As atividades de compor a requisição de trabalho, contendo seu book (relatório de fotos), equipamento e breve currículum é feita via online via sistema que ao final de compor a requisição, submetem-na para ser processada pelo departamento de produção. Via sistema, o dep. de produção ao visualizar um recebimento de uma nova requisição de trabalho fotográfico, a envia para ser tratada e julgada pela setor de análise (Board Administrator) que verifica a requisição e atribui um nível de acordo com o seu equipamento fotográfico, aceitando ou negando essa requisição. O departamento de produção define um prazo máximo de x dias e dispara envia email, após expirado esse prazo, para o setor de análise a fim de que ele proceda a análise o mais rápido possível.

Após isso, caso a requisição tenha sido aceita, o dep. de produção/técnico cria um novo registro para a requisição onde é armazenado o preço, por cada foto, que será pago ao fotógrafo à editora. Esses preços são definidos pelo Departamento Comercial de acordo com o nível do fotógrafo. Caso a requisição tenha sido rejeitada, o dep. de produção procede o arquivamento da mesma.

Atarefa processamento da requisição é decomposta também através do recurso "manual de procedimentos", significando que há uma disponibilização de um manual para os demais atores que a processam.

Por fim, o sistema define um controle de acesso simples onde os atores que utilizam o sistema podem se autenticar via login/senha e terem autorização (permissão) para realização de determinadas tarefas (funcionalidades) a partir de perfis específicos. Ainda, são mostradas algumas atividades com suas respectivas contribuições positivas ou negativas para realização de requisitos não funcionais como segurança, disponibilidade, confidencialidade, integridade e julgamento rápido. O softgoal usabilidade também é mencionado e a tarefa "To use CAPTCHA" (do inglês -*Completely Automated Public Turing test to tell Computers and Humans Apart*) é modelada, mostrando que, se a tarefa "To log by login/passwd" incluir essa

subtarefa de utilizar CAPTCHA , ela estará contribuindo negativamente para a usabilidade, apesar desta mesma tarefa contribuir positivamente para segurança, caracterizando assim um conflito. Sabemos que um dos fatores que mais prejudicam a acessibilidade para deficientes visuais via web, é o recurso CAPTCHA, ou seja, quando sistemas de login solicitam para o usuário digitar as letras de uma imagem exibida para se ter acesso ao mesmo. Isso é um impedimento total para um deficiente visual, pois os leitores de tela atuais do mercado não conseguem processar o conteúdo dessas imagens.

A seguir serão apresentados três exemplos do pré-processamento OOM-NFR, utilizando este modelo de agência fotografia modelado em i* wiki.

4.2.2 Análise NFR 1

Neste primeiro exemplo de Pré-Processamento e Análise NFR serão priorizados três softgoals: **Segurança, Usabilidade e Julgamento Rápido.**

Em todos os exemplos, considerou-se já ter um modelo i* rico em requisitos não-funcionais e o pré-processamento **OOM-NFR** começa a partir da atividade "**Escolher Tipo de Análise NFR"** onde o analista escolhe se a análise vai ser para frente (Forward) ou para trás (Backward). Nesta análise NFR foi escolhida a análise para trás "**Backward**". A razão dessa escolha foi apenas para fins de exemplificação do processo de análise NFR. Entretanto, conforme visto na seção 2.4.1, este tipo de análise é mais completa e tem mais vantagens que a "Forward".

Assim, essa primeira análise pode ser chamada também de configuração 1 a ser gerada com priorização de segurança, usabilidade e rapidez. Intitulou-se essa análise por:

Configuração 1: Prioriza-se Segurança, Usabilidade e Rapidez (Análise Backward)

O objetivo é encontrar, através da análise NFR, uma configuração que satisfaça a todos os softgoals do modelo e que foram escolhidos como prioritários. Esta análise é realizada através dos passos a seguir:

Passo 1:

Neste passo 1 (figura 47), é realizada a atividade "**realizar seleção do(s) softgoal(s) alvos e predicado/nível de satisfação"** do pré-processamento **OOM-NFR.**

Como se observa na figura 47, o analista escolheu segurança, usabilidade e julgamento rápido que são os softgoals prioritários. Como a análise é backward (top-down), foram escolhidos os softgoals que são raízes do diagrama i* agência fotográfica.

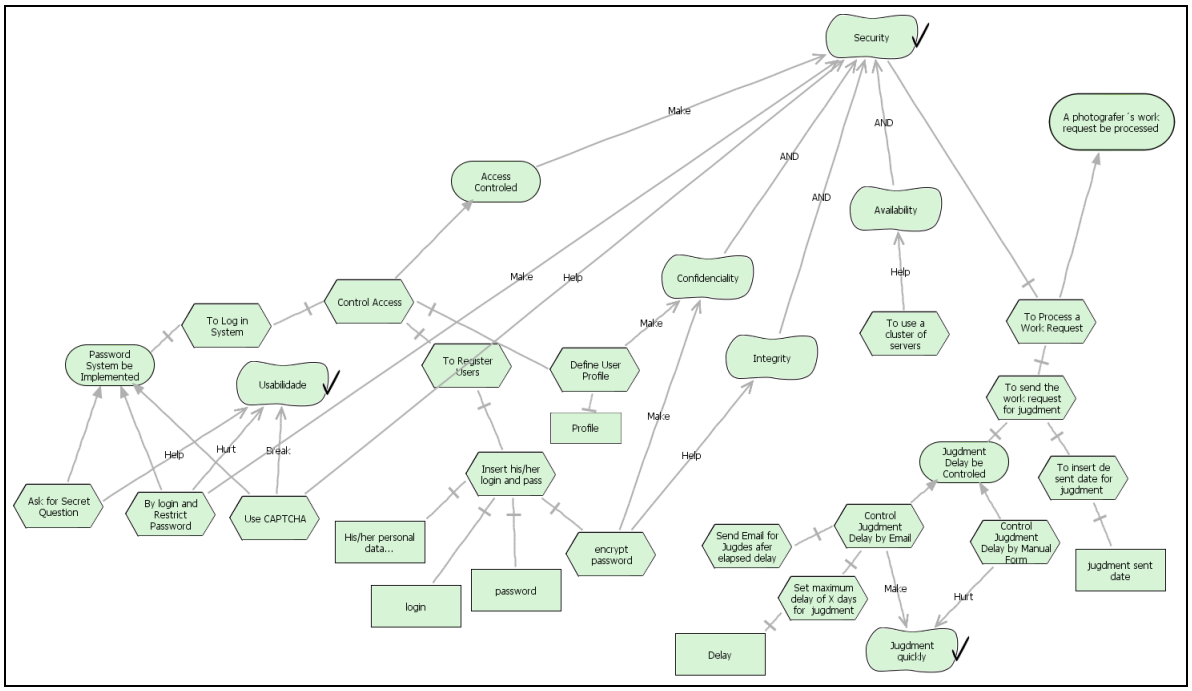


Figura 47. Diagrama i* inicial análise NFR 1 – passo 1

Passo 2:

Nesse passo, o analista aciona a atividade **Executar "Backward Interactive Analysis"** (HORKOFF et al. 2010) através da ferramenta OPENOME (OPENOME, 2010). O resolvedor de fórmula lógica SAT-Solver tenta resolver e encontrar uma configuração i* que satisfaça os softgoals selecionados, ou seja, segurança, usabilidade e rapidez, aplicando as regras de propagação e axiomas descritos na seção 2.4 (formalização do modelo i* e da abordagem). Como o resolvedor não consegue automaticamente encontrar tal configuração, ele gera um diagrama i*, conforme figura 48.

Neste diagrama, está na cor verde e com os rótulos aplicados, o resultado parcial da primeira iteração do que o SAT-Solver conseguiu resolver automaticamente. As atribuições ou propagações que o SAT-Solver não conseguiu resolver automaticamente são destacadas na cor amarela com os respectivos elementos i* e uma tela (figura 49) é exibida, exigindo um julgamento do analista de requisitos.

Esses elementos destacados em cor amarela dizem que o resolvedor, nesta primeira iteração, não conseguiu satisfazer o requisito softgoal usabilidade, aplicando o rótulo de desconhecido (Unknown), denotado por "**?**" no diagrama; e , sugeriu, também realçado na cor amarela, três níveis de satisfação para as tarefas do i* que têm links de contribuições para

usabilidade. Durante a interação, o analista pode aceitar estes níveis de satisfação sugeridos pelo resolvedor ou pode setá-los com quaisquer outros níveis. Isso vai depender de como o analista avalia o nível de satisfação dessas tarefas para atender (satisfazer) o softgoal destino que, neste caso, foi o softgoal usabilidade.

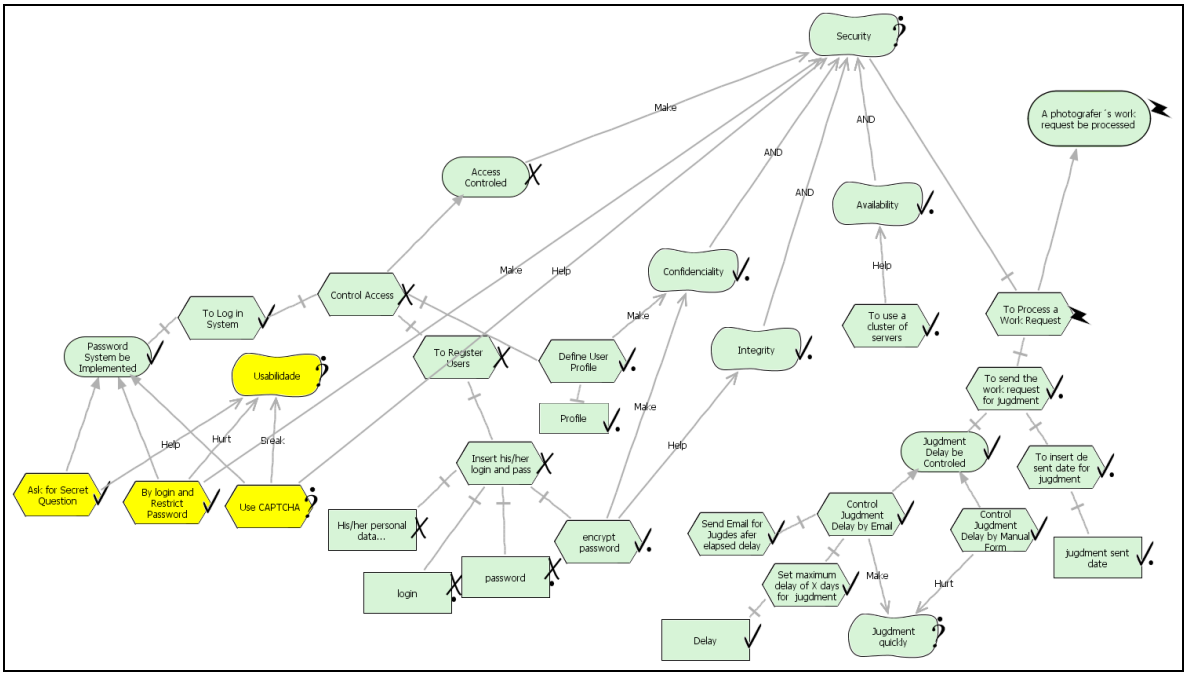


Figura 48. Diagrama parcial análise NFR 1 - passo 2

Na tela inicial de interação (figura 49), o analista, como está priorizando usabilidade, ele nega a tarefa "Use CAPTCHA", pois ela quebra usabilidade; satisfaz a tarefa "Ask for Secret Question" que contribui com Help para usabilidade; e decide satisfazer parcialmente a tarefa "By login and Restrict Password", pois tem contribuição positiva "Make" para Segurança e uma negativa "Hurt" para Usabilidade, ou seja, decidiu um meio-termo (satisfação parcial).

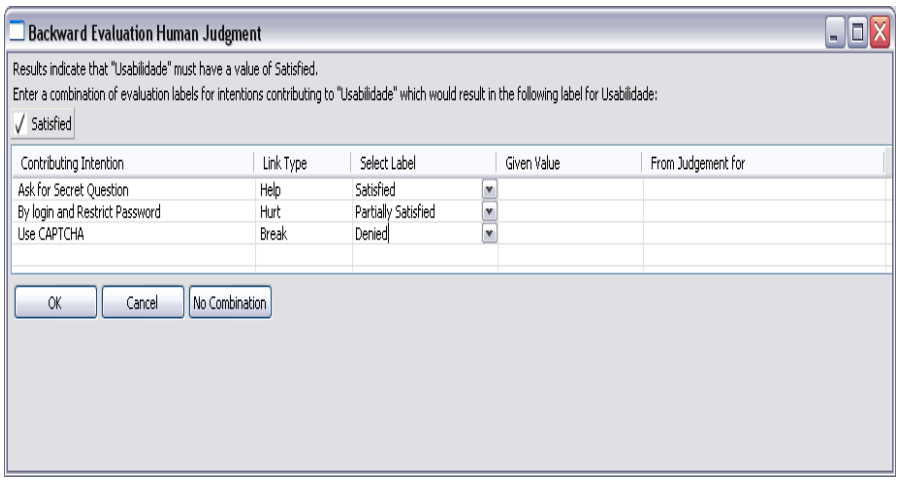


Figura 49. Tela decisão passo análise NFR 1 - passo 2

Essas atribuições específicas foram escolhidas para ter esses valores porque o analista tinha em mente as prioridades a serem alcançadas e satisfeitas.

Uma importante característica da abordagem (HORKOFF et al. 2010) também aparece na figura 50 do diagrama i* parcial gerado pelo SAT-Solver: a detecção de conflito para satisfazer a meta (goal) "**A photografer´s work request be processed**" e também da tarefa "**To process a work request**". Isso ocorre, por que, nesta iteração inicial, Segurança ainda está com valor de satisfação desconhecido (Unknown) e a tarefa "To send the work request for jugdment" ainda está com valor parcialmente satisfeito.

Após tomadas essas decisões, o analista pressiona o botão de OK e resolvedor continua a análise NFR, exibindo o diagrama i* da figura 50 e tela (figura 51) mostrados no passo 3 a seguir:

Passo 3:

Na figura 50, o resolvedor marcou as atribuições feitas pelo analista para usabilidade, conforme passo anterior, e tenta novamente, numa segunda iteração, resolver as prioridades iniciais. Entretanto, o resolvedor ainda não consegue automaticamente resolver Segurança e realça os elementos a ela associada na cor amarela, além de exibir nova tela (figura 51) para solicitar um novo julgamento do analista.

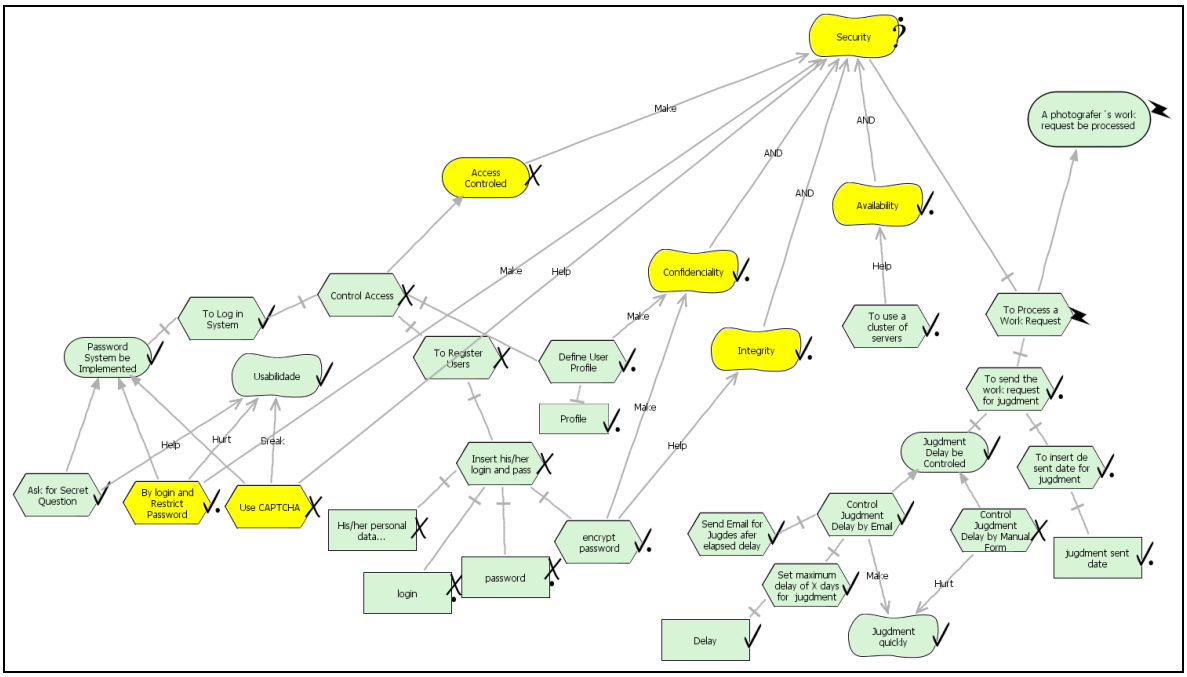


Figura 50. Diagrama parcial análise NFR 1 - passo 3

Neste diagrama, o resolvedor diz que Segurança, que recebe seis (6) contribuições, está ainda indefinida (Unknown) e sugere valores (itens realçados em amarelo) para os elementos que contribuem para Segurança.

Na tela 51, o analista observa a configuração atual do resolvedor e decide satisfazer praticamente todos esses elementos que contribuem para Segurança, exceto a tarefa “By login and Restrict Password”, que decide satisfazer parcialmente; e a tarefa “Use CAPTCHA” que decide negar novamente, pois ela quebra usabilidade.

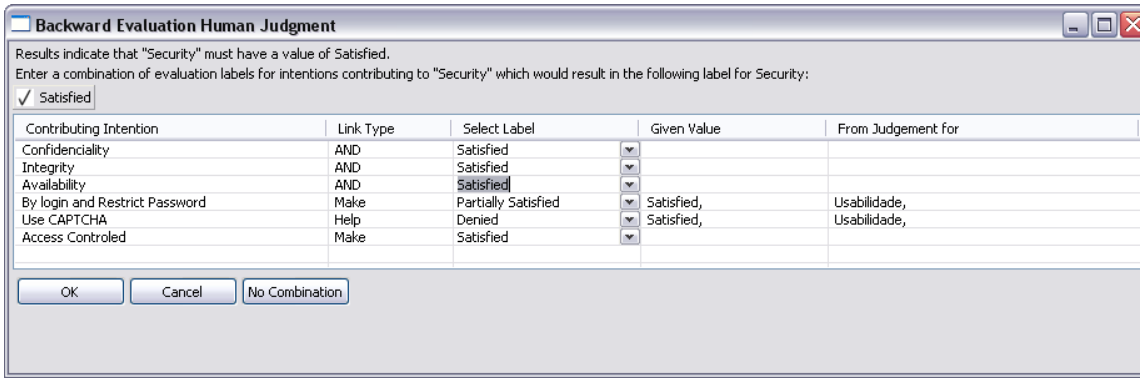


Figura 51. Tela decisão passo análise NFR 1 - passo 3

Após tomada essas decisões, o analista clica em OK e o resolvedor SAT continua o processamento descrito no passo 4:

Passo 4:

Neste passo, o SAT-solver obtém sucesso e, finalmente, exibe o diagrama i* final (figura 52) com a configuração que satisfaz as suas prioridades de segurança, usabilidade e rapidez; e também a tela (figura 53), mostrando que a resolução foi bem-sucedida.

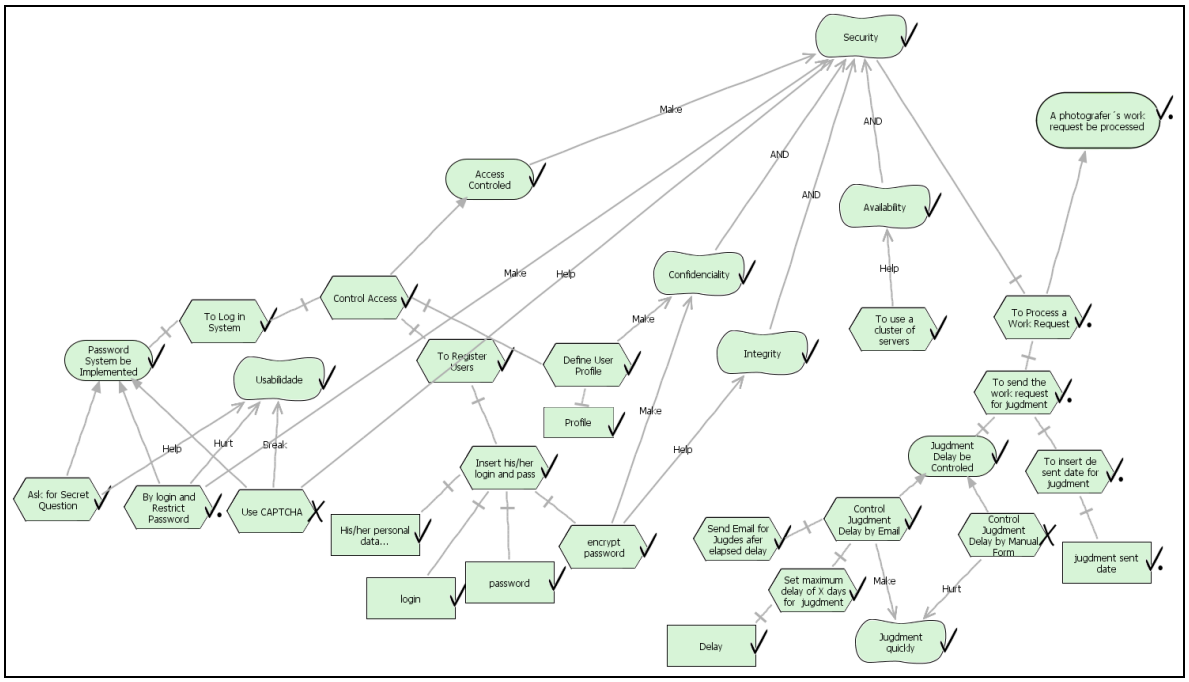


Figura 52. Diagrama i* final análise NFR 1 – passo 4

Este diagrama contém a **configuração 1** da análise NFR obtida com os elementos i* necessários para a próxima atividade do pré-processamento: **Refatorar Modelo i*, extraindo Tarefas e Recursos negados da Configuração Selecionada.**

Após esta última atividade do pré-processamento OOM-NFR, é gerado (refatorado) um modelo i* adequado à aplicação do **Processo Núcleo** que, de fato, realiza as transformações de i* para OO-Method e que será visto na seção 4.3 deste capítulo.

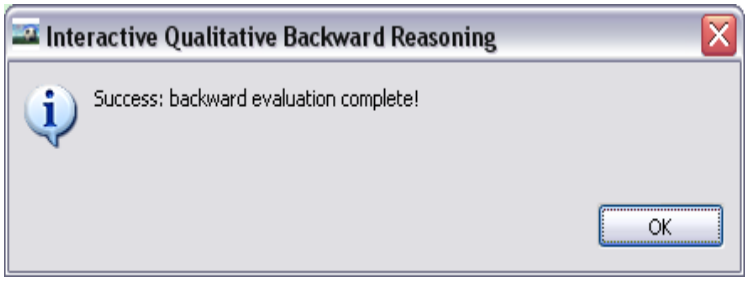


Figura 53. Tela final análise NFR 1 - passo 4

4.2.3 Análise NFR 2

Neste segundo exemplo de Pré-Processamento e Análise NFR será priorizada apenas o softgoal **Segurança.** Usabilidade e Julgamento Rápido da requisição de trabalho do fotógrafo (Work Request) serão quebrados, ou seja, não serão satisfeitos, pois não serão softgoals prioritários.

Nesta análise NFR, para efeito de exemplificação, foi escolhida a análise para frente "**Forward**". Entretanto, conforme seção 2.4.1, esse tipo de análise, para modelos grandes com várias alternativas, torna-se tediosa devido a necessidade de se realizar várias repetições da análise. Isso representa uma desvantagem quando comparada com a "backward"

Essa segunda análise, pode ser chamada de **configuração 2** a ser gerada com priorização apenas de segurança e intitulada como:

Configuração 2: Prioriza-se Segurança, mas quebra-se Usabilidade e Rapidez no Julgamento (Análise Forward)

Passo 1:

Neste passo 1 (figura 54) é realizada a atividade "**Seleção e configuração de tarefas a serem satisfeitas ou negadas"** do pré-processamento **OOM-NFR.**

Como se observa na figura 54, o analista inicialmente escolheu e atribuiu valores aos elementos folhas do diagrama i*, que estão realçados numa tonalidade de verde escuro, para serem submetidos à análise forward (bottom-up).

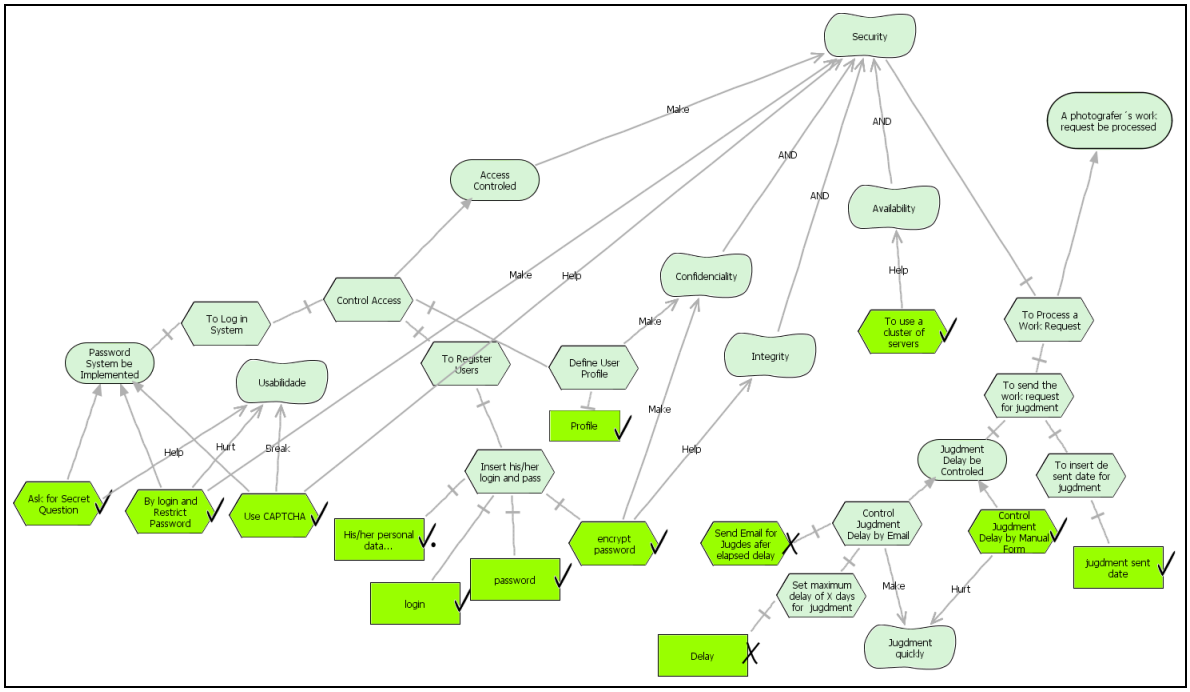


Figura 54. Diagrama i* inicial análise NFR 2 – passo 1

É importante mencionar neste diagrama que, intencionalmente e conhecendo suas prioridades a serem alcançadas, o analista de requisitos decidiu por negar totalmente "Send Email for Jugdes afer elapsed delay" e "Delay", pois acredita que essas atribuições contribuirão para a não satisfação do softgoal rapidez; e decidiu satisfazer totalmente a tarefa "Use CAPTCHA", pois ela quebra a usabilidade.

Após isso, o analista aciona o SAT-Solver com a atividade **Executar "Forward Interactive Analysis** do pré-processamento .

Passo2:

Neste passo, o resolvedor não consegue resolver Usabilidade, após aplicar as regras de propagação, recebendo valores parcialmente satisfeito, parcialmente negado e negado das tarefas "Ask for Secret Question", "By login and Restrict Password" e "Use CAPTCHA", respectivamente. Neste caso, o resolvedor solicita que o analista resolva, esses conflitos existentes, atribuindo um valor final para Usabilidade. Na tela apresentada na figura 55, o analista decide, como de fato era seu propósito inicial, negar Usabilidade.

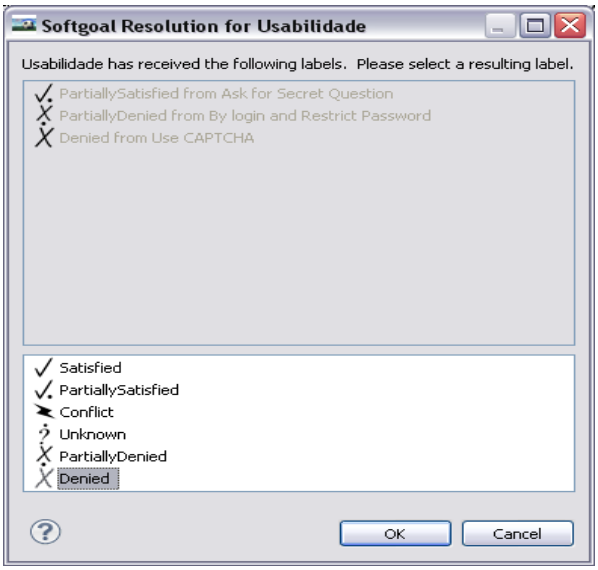


Figura 55. Tela decisão análise NFR 2 – passo 1

Passo 3:

Neste passo, o SAT-solver obtém sucesso e, finalmente exibe o diagrama i* (figura 56) com a configuração que satisfaz Segurança e quebra Usabilidade e Rapidez. Além disso, a tela (figura 57) mostra que a resolução foi bem-sucedida, tempo consumido para análise e número de interações humanas realizadas.

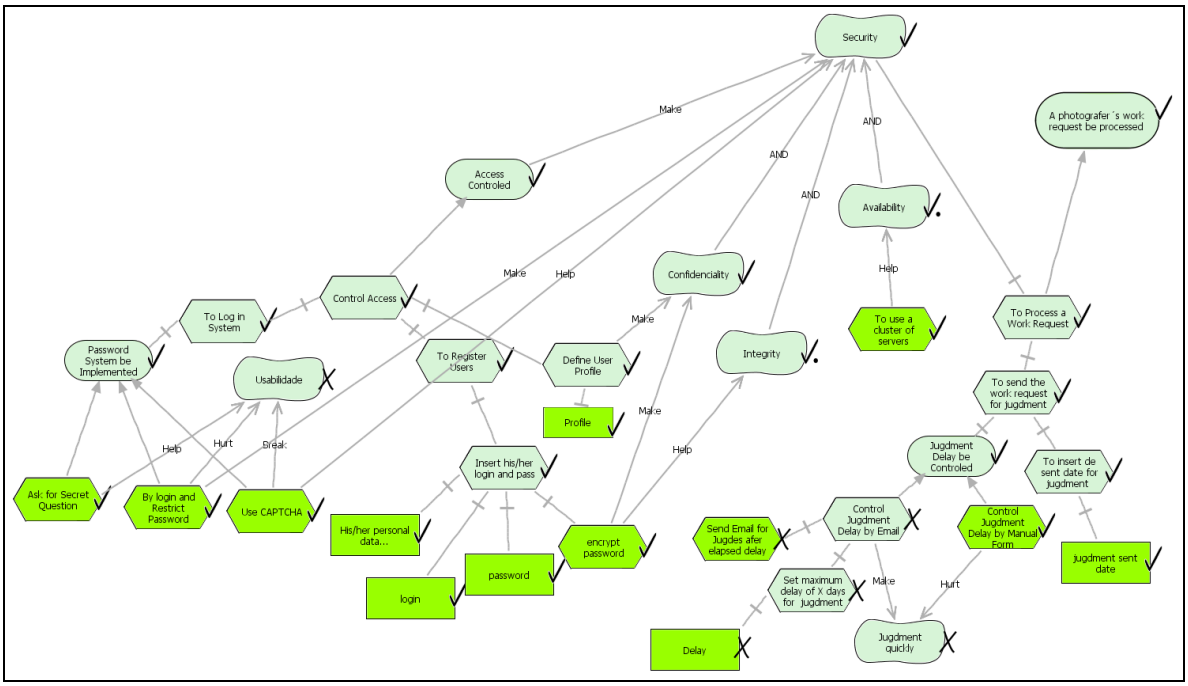


Figura 56. Diagrama parcial análise NFR 2 - passo 3

Essa configuração obtida é refatorada para retirar os elementos i* tarefas/recursos folhas negados e, depois, passar o modelo i* refatorado para o processo núcleo do NFR realizar as transformações.

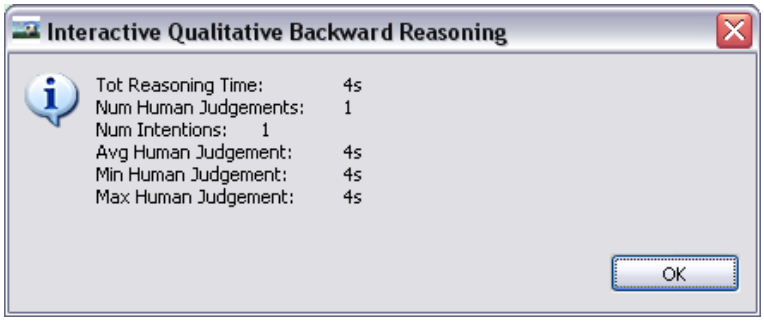


Figura 57. Tela final análise NFR 2 - passo 3

4.2.4 Análise NFR 3

Neste terceiro exemplo de Pré-Processamento e Análise NFR será quebrado, ou seja, não satisfeito o softgoal **Segurança** e serão satisfeitos **Usabilidade e Rapidez** no julgamento da requisição do trabalho do fotógrafo "Work Request", pois serão os RNFs prioritários. Nesta análise NFR foi escolhida a análise para frente "**Forward**".

Assim, essa terceira análise pode ser chamada de **configuração 3** a ser gerada com priorização apenas de **Usabilidade e Rapidez** e intitulada como:

Configuração 3: Quebra-se Segurança, Prioriza-se Usabilidade e Rapidez no Julgamento (Análise Forward)

Passo 1:

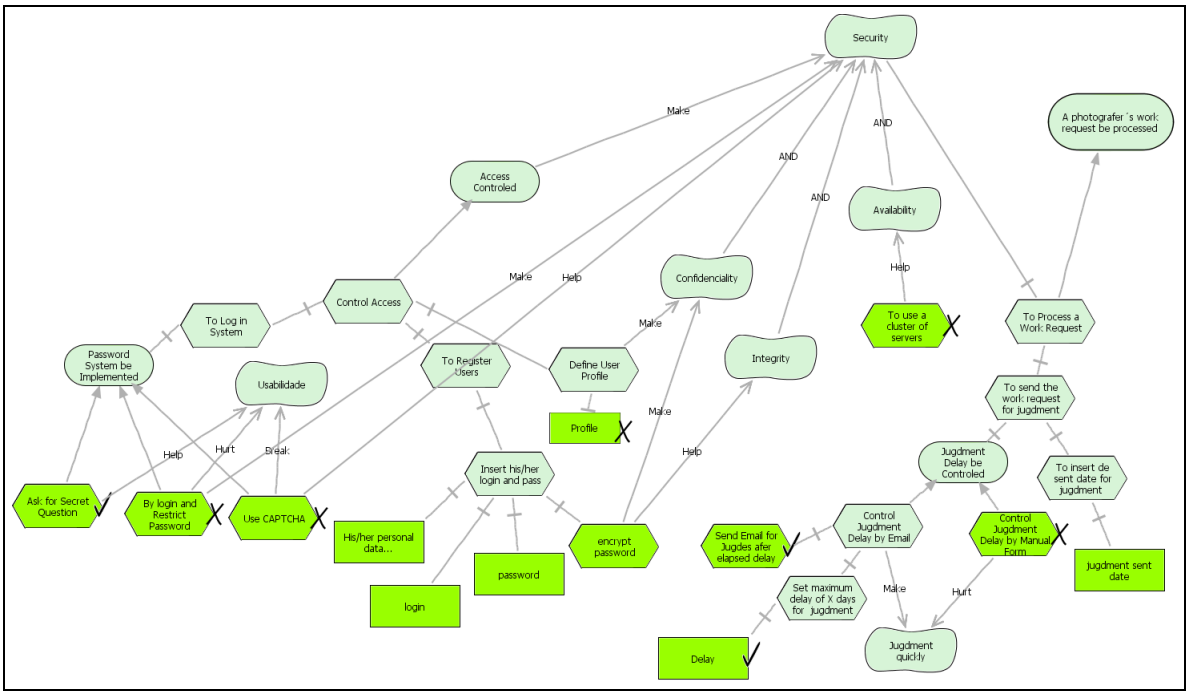


Figura 58. Diagrama i* inicial análise NFR 3 – passo 1

Neste passo 1 (figura 58) é realizada a atividade "**Seleção e configuração de tarefas a serem satisfeitas ou negadas"** do pré-processamento **OOM-NFR.**

Como se observa na figura 58, o analista inicialmente escolheu e atribuiu valores aos elementos folhas do diagrama i*, que estão realçados numa tonalidade de verde escuro, para serem submetidos à análise forward (bottom-up).

É importante mencionar neste diagrama que, intencionalmente e conhecendo as suas prioridades a serem alcançadas, o analista de requisitos decidiu por negar totalmente os seguintes elementos i*: "By login and Restrict Password", "Use CAPTCHA", "Profile", "To use a cluster of servers" e "Control Jugdment Delay by Manual Form" pois acredita que essas atribuições contribuirão para a **não satisfação** do requisito não-funcional **segurança** e **satisfação** de **usabilidade e rapidez**.

Após isso, o analista aciona o SAT-Solver com a atividade **Executar "Forward Interactive Analysis** do pré-processamento .

Passo 2:

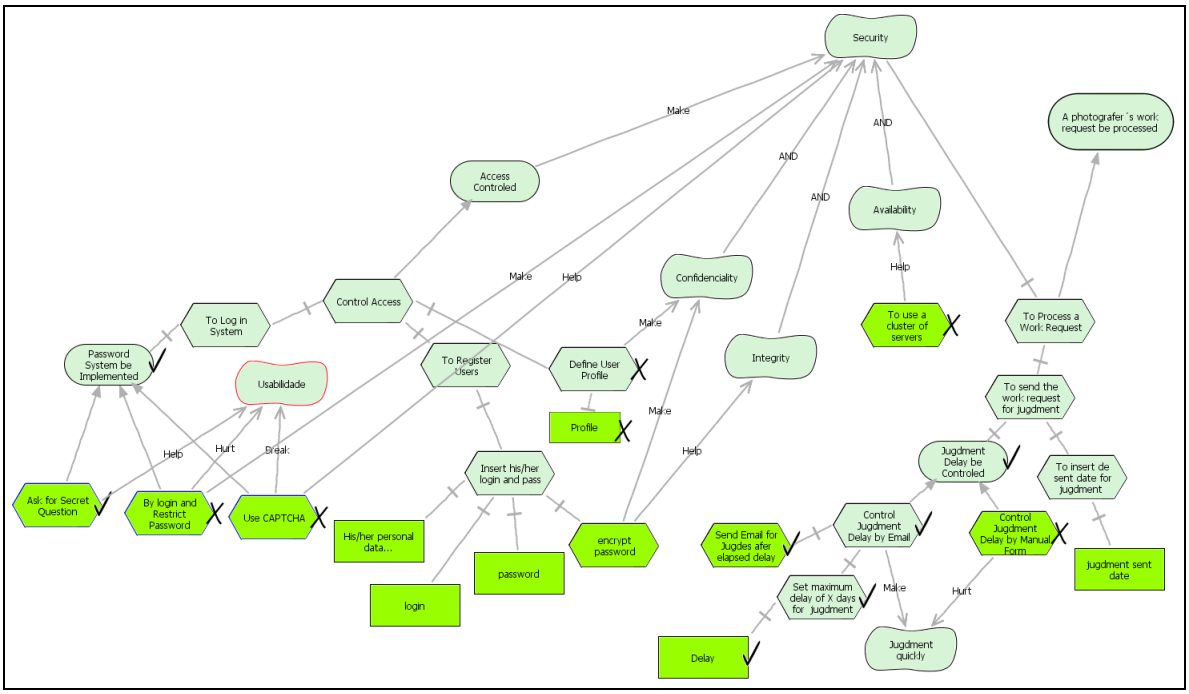


Figura 59. Diagrama parcial análise NFR 3 - passo 2

Neste passo (figura 59), o resolvedor, como não consegue resolver Usabilidade após aplicar as regras de propagação, recebendo valores parcialmente satisfeito das tarefas “Ask for Secret Question”, “By login and Restrict Password” e “Use CAPTCHA” respectivamente; e solicita que o analista resolva, atribuindo um valor final para Usabilidade.

Na Tela (figura 60) a seguir, o analista decide manter esse nível de parcialmente satisfeito para Usabilidade, pois foi a avaliação recebida por “Ask for Secret Question”.

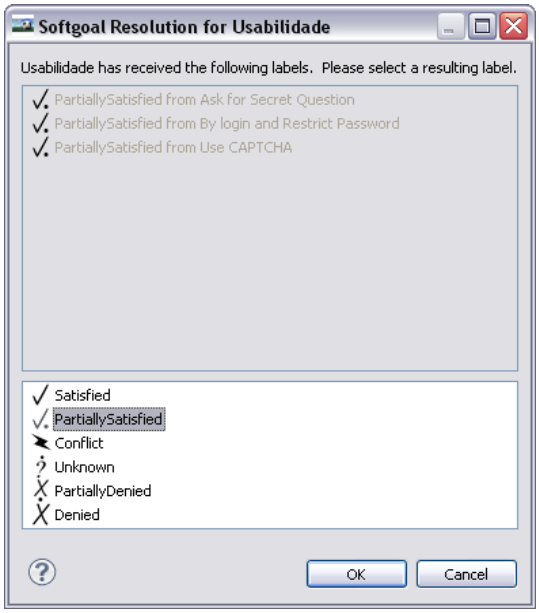


Figura 60. Tela decisão análise NFR 3 - passo 2

Passo 3:

Neste passo, o SAT-solver obtém sucesso e, finalmente exibe o diagrama i* (figura 61) com a configuração que quebra segurança e satisfaz usabilidade e rapidez.

Essa configuração obtida é refatorada para retirar os elementos i* tarefas/recursos negados e, depois, passar o modelo i* refatorado para o processo núcleo do NFR realizar as transformações. Observa-se, nesse caso, que a tarefa "Define User Profile" negada automaticamente pelo resolvedor também será retirada do modelo.

Além disso, a tela (figura 62) mostra que a resolução foi bem-sucedida, tempo consumido para análise e número de interações humanas realizadas.

Vistos esses três exemplos de pré-processamento do OOM-NFR, será apresentado, na próxima seção, o **Processo Núcleo** que é responsável pelas transformações dos elementos i* em OO-Method.

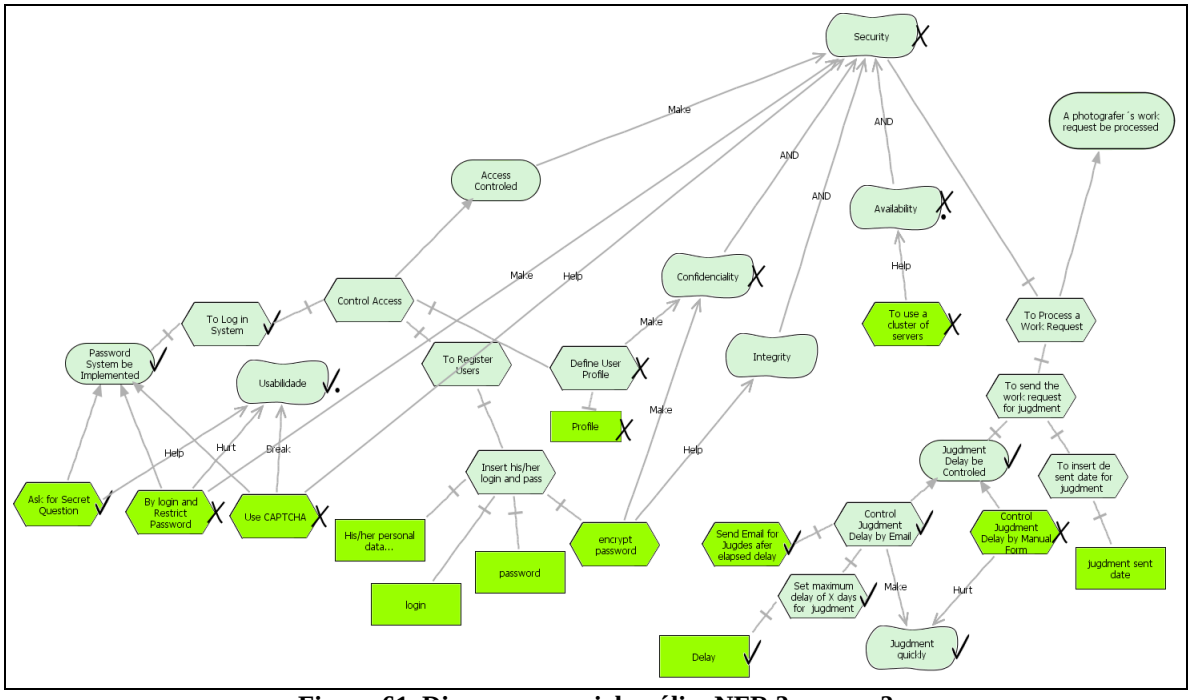


Figura 61. Diagrama parcial análise NFR 3 - passo 3

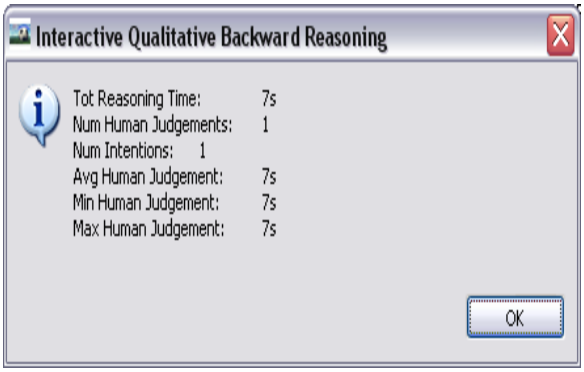


Figura 62. Tela decisão análise NFR 3 - passo 2

4.3 Processando as Regras de Transformação

O subprocesso responsável por realizar o processamento das regras de transformações é o **Processo Núcleo** que, de fato, é um tradutor ou compilador de modelos. Este processo recebe como entrada um modelo i* refatorado com os conflitos resolvidos e softgoals priorizados a partir da análise NFR realizada no **Pré-Processamento.** Além disso, como se observa na figura 63 a seguir, o **Processo Núcleo** recebe também como entrada o **metamodelo do OO-Method**. A partir daí, são realizadas as atividades do Processo Núcleo sobre todos os elementos desse modelo i* passado como entrada.

A figura 63 a seguir mostra o recorte do processo geral OOM-NFR, focando-se o **Processo Núcleo**.

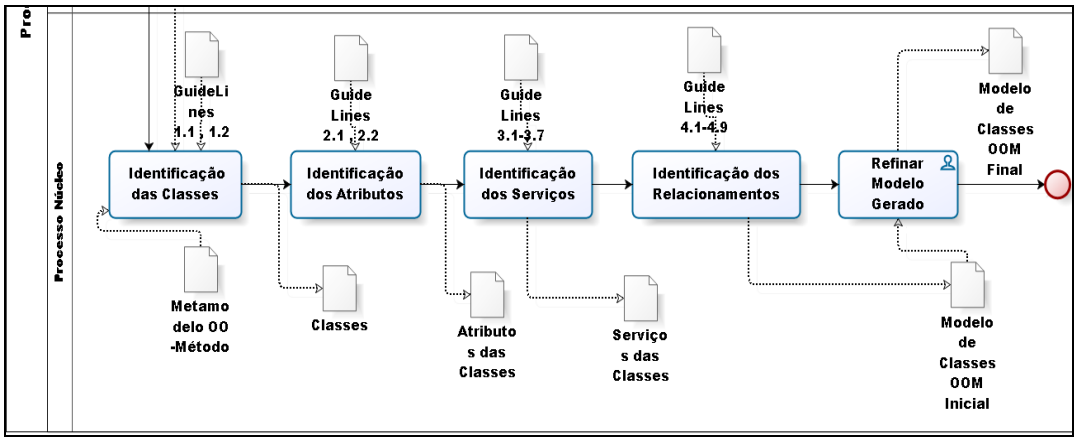


Figura 63. Processo Núcleo – recorte do OOM-NFR

O **Processo Núcleo** consiste em 5 (cinco) atividades descritas a seguir:

- Identificação das Classes;
- Identificação dos Atributos;
- Identificação dos Serviços;
- Identificação dos Relacionamentos;

- Refinamento do Modelo inicial OO-Method gerado.

Cada uma dessas atividades contém as diretrizes "guidelines" de transformação necessárias para identificar ou mapear os elementos do modelo i* nos elementos do modelo objeto do OO-Method: **Classes**, **Atributos**, **Serviços** e **Relacionamentos**. Cabe-se ressaltar que o modelo objeto é apenas um dos modelos do esquema conceitual do OO-Method (ver seção 3.2.1.3) e que o processo OOM-NFR tem como escopo apenas esse **modelo objeto**. Todas as diretrizes de transformação serão detalhadas nas próximas seções e exemplificadas através do sistema Agencia Fotográfica.

A implementação (codificação) completa dessas regras na linguagem de transformação de modelos QVT está na seção apêndice desta dissertação. Neste apêndice, estão também os comentários dos trechos de código QVT que realizam efetivamente as transformações especificadas neste capítulo. A razão de deixar a parte relativa ao código (implementação) no apêndice foi separar especificação de requisitos da implementação dessas diretrizes a fim de facilitar a clareza e não misturar os dois níveis de abstração.

A atividade final do **Processo Núcleo** e também do **OOM-NFR** consiste em refinar o modelo de classes inicial do OO-Method gerado, pois como será visto nas conclusões deste capítulo, há elementos do modelo OO-Method que não são possíveis de serem derivados do modelo i* wiki devido ao "gap semântico" entre esses modelos e suas peculiaridades específicas.

4.3.1 Geração das Classes

Esta atividade trata da identificação e geração das principais classes que devem estar no modelo de classe do OO-Method. Os elementos foco dessa atividade são atores e recursos do modelo i*.

Para simplificar, facilitar a compreensão geral, a aplicação e demonstração de todas as regras de transformações definidas, será considerado o modelo de entrada, o modelo i* da agência de fotografia completo, conforme ilustrado na seção **4.2.1** Entretanto, na seção **4.4** (Resultados) serão mostrados os modelos OO-Method obtidos, tendo como entrada para este **processo núcleo,** os modelos i* refatorados a partir dos três exemplos de análise NFR ilustrados na seção 4.2 anterior.

Os atores são considerados para a geração de classes na classe alvo modelo, porque, por definição, um ator é uma entidade ativa que realiza ações para atingir as metas, utilizando as suas capacidades. Já os recursos podem representar

uma entidade física ou informacional. As entidades físicas são elementos do mundo real que têm várias propriedades. Como, por exemplo, documentos eletrônicos, tais como uma fatura eletrônica. Portanto, o conceito de ator e entidades físicas podem ser combinados com o conceito de objeto, em um modelo de classe, o que implica que esses os elementos i* estão relacionados com a definição de classe. Assim, para esses dois construtores i*, são definidas as seguintes diretrizes ou regras de transformação (GIACHETTI, 2011):

Diretriz 1.1:Um ator i* é transformado em uma classe do modelo de classe destino. O nome da classe gerada é obtida a partir do nome do ator.

No modelo i* exemplo (agência fotográfica), os atores envolvidos são os seguintes:

- **User**
- **Photographer**
- **Production Depart. Employee.**
- **Commercial Dep. Employee**
- **Board Administrator Employee**
- **Photo Agency System**

O ator P**hoto Agency System,** doravante denominado de ator PAS, é um ator diferenciado que tem sob sua fronteira vários elementos i* a serem automatizados.

Assim, esse ator será convertido numa Classe chamada abreviada também como PAS no modelo conceitual OO-Method. Essa classe terá apenas o atributo nome derivado do nome do ator de origem. É importante que o nome do ator sistema no modelo SR final inicie com o substantivo "System" a fim de que, se distingua dos demais atores que não são sistemas. Isso inclusive poderia facilitar na implementação das regras de transformação. A Classe PAS conterá ainda serviços eventos de inicialização e finalização do sistema.

Por fim, serão gerados por "default" e automaticamente os serviços eventos (atômicos) relacionados com a criação, exclusão e modificação de instâncias dessas classes derivadas de atores. Por exemplo, ao se criar a classe Usuário, criar-se-iam também os serviços básicos e indispensáveis a manutenção dessa classe, ou seja, Criar Usuário, Deletar Usuário (Create User, Delete User).

Diretriz 1.2: Um recurso i* que representa uma entidade física é transformada em uma classe do modelo de classe destino. O nome da classe gerada é obtidos a partir do nome do recurso.

No modelo i* do sistema agência fotográfica os recursos físicos envolvidos são os seguintes:

- **work request**
- **Refused work request**
- **Accepted work request**
- **Proceedings Manual**
- **Level**

Deve-se gerar automaticamente create, delete e update, caso não estiverem explícitos no modelo e não forem caputrado pela diretriz 3.7.

Cabe ressaltar que o metamodelo i* wiki utilizado para construir os diagramas a partir da ferramenta IStarTool não metamodela, como deveria ser, o elemento intencional recurso como sendo uma metaclasse com um atributo, por exemplo, "resource type" que pudesse diferenciar um recurso físico de um recurso informativo. Entretanto, considera-se, nos exemplos realizados que tal atributo exista e possa ser processado para fins de transformação.

A lista 1 mostra um trecho da automatização dessas diretrizes 1.1 e 1.2, implementado na linguagem de transformação de modelos QVT.

```
modeltype ISTAR uses istar('http://www.cin.ufpe.br/istar')
where {self.elements->closure(oclIsKindOf(ISTAR::ElementType::SOFTGOAL))->size()>0 };
modeltype OOM uses oo_method('http://www.cin.ufpe.br/oo_method');
transformation IstarOOm(in istar : ISTAR, out OOM);
mapping ISTAR::Model::model2OOMModel() : OOM::Model {
title := self.title;
/*
Mapeamentos que transformam Atores e Recursos Físicos em Classes OO-Method
*/
classes:= self.actors[ISTAR::Actor]->asOrderedSet()->map ator2classe()->
        union(self.getRecursosFisicos()->asOrderedSet()->map recursofisico2classe())->asOrderedSet();
```

Lista 1 Código em QVT para Transformar i* em OO-Method: Diretriz 1.1 e 1.2

A cláusula ***modeltype*** define os metamodelos ECORE do i* wiki (ISTAR) e do OO-Method (OOM). A cláusula ***where*** indica que modelo instância do ISTAR só será processado, se houver elementos softgoals, pois estes são necessários para realização da análise NFR.

A função ***transformation*** da lista 1 especifica que a transformação receberá uma instância de entrada ***istar*** compatível com o metamodelo ISTAR e terá uma instância de saída (resultado) compatível com metamodelo OOM.

A função de mapeamento ***mapping*** é o ponto de entrada de todas as transformações. Essa função, denominada **model2OOMModel(),** mapeia a classe do tipo modelo em i* wiki, representada em QVT por **ISTAR::Model,** em outra classe do tipo modelo em OO-Method representada por **OOM::Model**. Além disso, é no corpo dessa função **mapping** , onde se faz a atribuição do título do modelo e as chamadas das demais transformações para gerar as classes, atributos, relacionamentos de associação e agentes.

Na lista 1, é ilustrada também a geração de classes do modelo OO-M a partir dos atores e recursos físicos do modelo i*, conforme especificação das diretrizes 1.1 e 1.2. As funções que realizam essas transformações são **ator2classe() e recursofisico2classe**() invocadas pela cláusula ***map*** em QVT**.** Assim todas as classes do modelo OO-M são geradas a partir da união do retorno dessas duas funções.

Para mais detalhes da automatização do processo OOM-NFR, uma listagem completa da implementação em QVT está disponível no **apêndice** desta dissertação.

4.3.2 Geração de Atributos

Para cada classe obtida a partir das duas primeiras diretrizes, é necessário identificar os elemento i* que podem ser usados para inferir seus atributos. Para fazer isso, os recursos que representam uma entidade informativa serão o nosso principal alvo, porque eles representam propriedades de recursos físicos ou atores. Assim, esses recursos são transformados em atributos das classes geradas anteriormente, considerando a partir do qual o elemento (ator ou entidade física) a entidade informacional está relacionada. Para a geração de atributo desses atributos as seguintes as diretrizes de transformação são definidas.

Diretriz 2.1: Se um recurso i* representa uma entidade de informação relacionada a um ator do modelo i*, então esse recurso informacional é transformado em um atributo da classe gerada a partir desse ator a que se refere. O nome do atributo é obtido a partir do nome do recurso informacional. Os recursos do modelo que são afetados por essa regra de transformação são:

• **Personnal data (relacionado a User)**
• **Profile (relacionado a User)**
• **Login (relacionado a User)**
• **Password (relacionado a User)**
• **Level (relacionado a Photographer)**
• **Brief curriculum (relacionado a Photographer).**
• **Photo equipment (relacionado a Photographer)**

Diretriz 2.2: Se um recurso i* representa uma entidade informacional relacionado com um recurso físico, então esse recurso informacional é transformado em um atributo da classe gerada do recurso físico. O nome do atributo é obtido a partir do nome do recurso informacional. Assim, os recursos do modelo que são afetados por esta diretriz de transformação são os seguintes:

• **Photog. work request status (relacionado a Work Request)**
• **Serial number (relacionado a Work Request)**
• **Submission date (relacionado a Work Request)**
• **Photog. work request status (relacionado a Work Request)**
• **A book with his/her photographic reports (relacionado a Work Request)**
• **The bureaucrat (relacionado a Work Request)**
• **Jugdment sent date (relacionado a Work Request)**
• **Acceptance date (relacionado a Accepted work request)**
• **Descripton of minimun equipment (relacionado a Level)**
• **Prices of each level (relacionado a Level)**
• **Price to charge the publishing house (relacionado a Level)**

4.3.3 Geração de Serviços

Neste ponto, as tarefas do modelo i* SR e suas decomposições possíveis são inspecionadas (busca de profundidade). Segundo o modelo i*, uma tarefa especifica uma forma particular de fazer alguma coisa. Além disso, a partir da experiência prática, uma tarefa no modelo i * em geral, é responsável pela realização de um objetivo e/ou para a produção de um recurso. Para a identificação dos serviços de classe, considera-se que uma tarefa represente e possa ser mapeada em um serviço no modelo de classe. Um serviço no modelo de

classes descreve um comportamento específico dos objetos de uma classe. Na abordagem OO-Method, um serviço pode ser atômico (Evento) ou uma composição de outros serviços (Transação). Assim, como o modelo i* não oferece informação de modelagem o suficiente para determinar se uma tarefa corresponde a um serviço atômico ou composto, as diretrizes de transformação apresentadas para a geração de serviços só produzem serviços como transações. Só na fase final, ou seja, na atividade de Refinamento do Modelo OO-M gerado é que os eventos são distinguidos das transações.

Além disso, se a tarefa representa um serviço que agrupa outros serviços, então esta tarefa é transformada em uma transação que é composta pelas transações agrupadas. As transações inferidas devem ser refinadas mais tarde, em tempo de design, para indicar os eventos correspondentes. Os eventos relacionados com criação, exclusão e modificação de instâncias de classe são sempre criados por padrão "default" quando as classes são geradas, exceto quando estes serviços são obtidos da aplicação das diretrizes de transformação. Assim, as diretrizes para a identificação de serviços são os seguintes:

Diretriz 3.1: Uma tarefa que só afeta os valores de recursos informacionais relacionada a um ator se transforma em uma transação da classe gerada a partir de o ator correspondente. O nome da transação gerada é inferido a partir do nome da tarefa. Os recursos relacionados são definidos como argumentos da transação. Se a tarefa também é decomposta em subtarefas, então as subtarefas decompostas são incluídas na fórmula da transação gerada.

Exemplos de tarefas afetadas por essa diretriz:

- Relacionadas ao ator **Usuário**
  - **Insert login and password**
  - **Define User Profile**
  - **Encript the password**

- Relacionada ao ator **Fotógrafo**
  - **To bind a level to a registered photographer (argumento: level)**
  - **Production Depart. Employee (não há tarefas geradas)**
  - **Commercial Dep. Employee (não há tarefas geradas)**
  - **Board Administrator Employee (não há tarefas geradas)**

• Relacionada ao ator **Sistema Agência de Fotográfica - "Photo Agency System - PAS**"

- **use CAPTCHA (1)**
- **By Login and Restrict password (2)**
- **Ask for Secret Question (3)**
- **To Register Users in System (personal data) (4)**
  - Esta transação é uma fórmula composta pela tarefa **Insert login and password**
- **To Log in System (5)**
  - Esta transação é uma fórmula composta pelas tarefas 1,2 e 3 (OU inclusivo)
- **Control Access**
  - Composta pela tarefas (4) , (5)
- **Initialize, finalize System (s**erviços default gerados automaticamente**)**

Observa-se que, a maioria das tarefas que, no modelo original SR (Alencar, 2009), estavam alocadas a atores setores organizacionais (Depto. Comercial, Depto Producão, etc), agora, neste modelo refinado SR, fazem parte do ator sistema PAS.

Diretriz 3.2: Uma tarefa que só afeta os recursos informativos relacionados a um recurso físico é transformada em uma transação do recurso físico correspondente. O nome da transação gerada é inferido a partir do nome da tarefa. Os recursos relacionados são definidos como argumentos da transação. Se a tarefa é também decomposta em subtarefas, então as subtarefas decompostas estão incluídas na fórmula da transação gerada.

- Transações associadas ao recurso físico **Work Request**
  - **To assign na reception date (submission date)**
  - **To assign a number (serial number)**
  - **Set Maximum delay (delay)**
  - **To insert the sent date for jugdment (jugdment sent date)**
  - **To change status (photo work request status)**
  - **Assign Level for PhotoRequest(assigned level)**

Obs. Os itens entre parênteses são os argumentos das transações.

• Transação relacionada ao recurso fisico **Level**

- **To insert the level register data (level**, **Descripton of minimun equipment**, **Prices of each level**, **Price to charge the publishing house**)

Estas transações são geradas a partir das tarefas que estão próximo dos recurso informacionais folhas e recebem esses recursos como argumentos.

Diretriz 3.3: Uma tarefa que só afeta um recurso físico é transformada em um serviço do recurso físico correspondente. O nome da transação gerada é inferido a partir do nome da tarefa. Se a tarefa também é decomposta em subtarefas, em seguida, as subtarefas decompostas são incluídas na fórmula da transação gerada.

- Transações relacionadas ao recurso físico **Work Request:**
  - **To receive a work request**
  - **To create a work request**
  - **To change the status of an work request**
  - **To Send Email for Jugdes after elapsed delay (1)**
  - **Set maximum delay of X days for judgment(2)**
  - **To insert de sent date for judgment(3)**
  - **To send work request for judgment – Composta pela tarefa (3)**
  - **Control Judgment Delay by Email**
    - **Transação composta pelas tarefas (1) e (2)**
  - **Control Judgment Delay by Manual Form**

- Transação associada a **Refused work request**
  - **To shelve the refused work request**

- Transações associadas a **Accepted work request**

  - **To Assign level of accepted work request (1)**
  - **To process the accepted work request**: Composta pela transação (1) anterior e da **To bind a level to a registered photographer** do ator Fotógrafo já especificada na Diretriz 3.1

Diretriz 3.4: Uma tarefa que afeta recursos físicos diferentes (distintos) ou afeta recursos informativos relacionados a diferentes recursos físicos é transformada em uma transação da classe gerada a partir do ator que contém tal tarefa (de acordo com a fronteira ator

correspondente). O nome do gerado da transação é inferido a partir do nome da tarefa. Os recursos relacionados são definidos como argumentos da transação. Se a tarefa também é decomposta em subtarefas, então as subtarefas decompostas são incluídas na fórmula da transação gerada.

- **To process a work request** (Work Request , Proceedings Manual)
  - É uma fórmula composta, devido à decomposição de tarefas em i*, pelas tarefas que foram transformadas em transações de maior nível do recurso físico *work request* através da diretriz 3.3 anterior, ou seja, composta pelas transações: **To receive a work request**,**To create a work request**, **To change the status of an work request** e **To send work request for judgment.**

Essa diretriz tem impacto nas diretrizes 4.6 e 4.7 que geram os relacionamentos.

Diretriz 3.5: Uma tarefa que não afeta recursos é transformada em uma transação da classe gerada a partir do ator que contém a tarefa (de acordo com a fronteira do ator correspondente). Se a tarefa também é decomposta em subtarefas, então as subtarefas decompostas são incluídas na fórmula da transação gerada. O nome da transação gerada é inferido a partir do nome da tarefa.

Esta diretriz contempla as tarefas solitárias, ou seja, estão no modelo, mas não geram nem consomem recursos de natureza alguma, restando apenas a opção de gerá-las e associá-las ao ator a que pertencem.

O exemplo mais significativo no modelo i* é a tarefa "**To use a Cluster of Servers**" que é transformada na transação de mesmo nome, passando a pertencer à classe PAS derivada do ator sistema.

Diretriz 3.6: As tarefas que participarem de uma dependência de recurso, onde o recurso corresponde a um recurso físico, são transformadas em transações da classe gerada a partir do recurso dependum. Os nomes das transações geradas são inferidos a partir dos nomes dos correspondentes tarefas. Esta diretriz é complementar a diretriz 3.3. Não se aplica quando a orientação é 3.4 é aplicada às tarefas envolvidas.

Essa diretriz gera múltiplas transações. O exemplo no modelo são as tarefas "Submit work request" e "To receive a work request" sobre o recurso **work request**. Como "To

receive a work request" já foi gerada pela diretriz 3.3, esta regra apenas adiciona mais uma transação, a "**Submit work request**" proveniente do ator Fotógrafo.

Diretriz 3.7: Uma tarefa que está envolvida na geração de um recurso físico é transformada em uma transação de criação da classe gerada a partir do recurso físico correspondente. O nome da transação gerada é inferido a partir do nome da tarefa envolvida. Esta diretriz é complementar à diretriz 3.3. Não é aplicável quando a diretriz 3.4 é usada sobre a tarefa em questão.

Exemplos:

- **Create work request** (serviço evento atômico relacionado a **work request**)
- **Create Level Register** (serviço evento atômico relacionado a **Level**

A lista 2 ilustra a automatização de algumas dessas transformações implementadas na linguagem de transformação de modelos QVT.

```
mapping ISTAR::Actor::ator2classe(): OOM::Class{
        name := self.name;
  attributes:= self.inferir_atributos_ator(self)->map recinfo_ator2attribute();
   services:= self.inferir_servicos_ator(self)->map task_ator2service();
}
 /* Mapeamento que transforma um recurso informativo (lógico) em um Atributo da classe gerada a partir de um ator*/ mapping ISTAR::Element::recinfo_ator2attribute(): OOM::Attribute{
name := self.name;
}
 /* Mapeamento que transforma uma tarefa de um ator em um serviço da Classe gerada do ator. */ mapping ISTAR::Element::task_ator2service(): OOM::Service{
        name := self.name;
        kind:= OOM::ServiceKind::none;
        parameters:= self.getParameters()->map element_resource2parameter();
}
```

Lista 2 Código em QVT para Transformar i* em OO-Method: Atributos e Serviços

Nessa lista 2 de código, está ilustrada a geração de atributos e serviços das classes OO-Method realizada através da função de mapeamento **ator2classe()**. No corpo dessa função, é atribuído o nome da classe a partir do nome do ator e são recuperados os atributos e serviços

relacionados a esse ator, para depois, serem invocadas as funções **recinfo_ator2attribute()** e **task_ator2service()** que, de fato, realizam as transformações.

A lista 2 mostra ainda o mapeamento que transforma uma tarefa de um ator em um serviço da classe gerada a partir desse mesmo ator. Como cada serviço em OO-M tem um nome, um tipo e parâmetros (ver metamodelo OO-Method na seção 3.2.2), então esses elementos são também gerados. A atribuição do tipo do serviço como "**none**" indica que este serviço é uma transação e não um evento de criação ou destruição. Para mais detalhes do código implementado em QVT, ver seção apêndice.

4.3.4 Geração dos Relacionamentos

Nesta atividade, os conceitos básicos considerados de relacionamentos das abordagens orientadas a objeto são: a generalização e associações. No entanto, é importante ressaltar que i* foca principalmente nas representações estratégicas por meio dos elementos intencionais e seus relacionamentos. Portanto, as informações de cada relacionamento do modelo i* devem ser analisadas para se derivar os tipos de relações geradas entre as classes.

O i* oferece um conceito similar ao de generalização para a definição de relacionamento entre os atores, que é a relação é-um (IS-A). Este construtor i* tem correspondência direta no modelo de classe. No entanto, para os recursos, em particular, para aqueles recursos que representam entidades físicas, não há uma clara representação no framework i* para determinar a ocorrência de generalizações. Portanto, é necessário o conhecimento e intervenção do analista para determinar quando generalizações estão presentes entre os recursos físicos transformados em classes de acordo com as diretrizes da atividade 3 anterior. A geração de associações entre as classes é muito limitado em modelos i*, uma vez que o i* wiki não fornece informações suficientes para determinar propriedades como a cardinalidade de uma associação. Neste ponto, é muito importante observar a relação particular que a abordagem OO-Method utiliza para indicar a visibilidade que uma classe tem sobre os serviços ou atributos de outras classes do modelo. Esta relação é chamada relação/relacionamento de agente, e é fundamental para permitir a correta execução dos serviços definidos na classe modelo. As associações que são geradas a partir de um modelo i* não podem distinguir entre as relações que correspondem a associações entre classes, e relacionamentos de agente. Portanto, no modelo de classes gerado, todas as relações inferidas (com exceção de relações de herança) são expressas como associações, que devem ser refinadas (atividade final do processo núcleo do OOM-NFR) para

identificar corretamente associações e as relações de agente. Para o relacionamento de associação deve ser necessário adicionar as informações de cardinalidade, o papel de cada extremidade da associação e o nome da associação. Para as relações de agente, deve ser necessário para adicionar as informações correspondentes à visibilidade sobre os serviços da classe relacionada.

Diretriz 4.1: Se houver um relacionamento de generalização (relação é-um "IS-A") entre dois atores do modelo i* que foram transformados em classes, em seguida, um relacionamento de generalização é definido entre essas classes correspondentes no modelo de classe.

No modelo i* agência fotográfica não há relacionamentos "IS-A". Isso foi feito por que a semântica desses relacionamentos ainda é dúbia e sem consenso entre as várias versões do i*, inclusive o i* wiki. Além disso, de nada adiantaria ter esses elementos no modelo i*, pois ainda não foi especificado no metamodelo OO-Method utilizado para o OOM-NFR (seção 3.3.2), o tipo de relacionamento herança. Assim, apenas os relacionamentos de associações e "agent link" serão gerados em OO-Method. Assim, esta diretriz 4.1 e a 4.2 estão especificadas em QVT como gerando relacionamentos de associações em OO-Method.

Entretanto, apesar dessas limitações, no modelo i* agência fotográfica, por exemplo, poder-se-ia relacionar os atores Photographer, Production Depart. Employee, Commercial Dep. Employee, Board Administrator Employee com um relacionamento IS-A para o ator User. Dessa forma seria gerado a partir desta diretriz 4.1 um relacionamento de Herança em OO-Method da classe User para essas demais classes, possibilitando dessa forma herança de transações e atributos.

Diretriz 4.2: Se um recurso físico do modelo i * é derivado de outro recurso físico, então um relacionamento de generalização é definido do recurso derivada para o recurso original. Esta derivação não é definida explicitamente no modelo i*, portanto é necessária a intervenção do analista de requisitos para indicar quais são os recursos físicos abrangidos por essa condição.

No modelo i* do sitema agência fotográfica, esta situação está presente para os recursos **Refused work request** e **Accepted work request** que seriam derivados de **Work Request,** mas que pelos mesmos motivos da diretriz 4.1 anterior não são gerados os relacionamento de generalização/herança entre essas classes.

Diretriz 4.3: Para as duas classes geradas a partir dos atores i*, se houver qualquer ligação de dependência de tarefa ou recurso entre os dois atores transformados, então uma associação entre as classes correspondentes é gerada automaticamente no diagrama de classes OO-Method.

No modelo exemplo i* agência fotográfica, praticamente todos os atores têm um relacionamento com o ator Sistema, pois existem diversas dependências de recurso ou tarefa entre estes atores, possibilitando a geração de relacionamento de associação entre eles.

Esta regra é uma generalização da 4.4, pois engloba qualquer tipo de dependência entre esses atores. Entretanto ela só gera associações entre atores.

Exemplos de associações geradas por essa diretriz:

- **User e Photo Agency System (PAS)**
- **Photographer e PAS**
- **Production Depart. Employee e PAS**
- **Commercial Dep. Employee e PAS**
- **Board Administrator Employee e PAS**

Diretriz 4.4: Se existe uma ligação de dependência de recursos onde o dependum, os atores depender e dependee foram transformados em classes, então associações são geradas automaticamente entre estas classes.

Esta regra é uma instância da 4.3, só que ela gera associações entre atores e recursos físicos (dependum recurso informacional não geram associações por essa diretriz).

Como exemplo de aplicação dessa diretriz no modelo i* estão os atores **Photographer**, **PAS** (**Photo Agency System**) e a dependência de recurso físico **Work Request**. Logo, são geradas em OO-Method associações entre as classes geradas a partir desses elementos.

Duas associações são geradas:

- **Photographer e Work Request**
- **Work Request e PAS**

Diretriz 4.5: Para uma ligação de dependência de recurso onde o dependum é transformado em um atributo de classe (dependum é recurso informacional) e os atores depender e dependee são transformados em classes, uma associação é gerada entre as classes

geradas dos atores e a classe derivada do recurso físico ao qual está associado esse atributo (recurso informacional).

Como exemplo de aplicação desta diretriz, existe ligação de dependência dos recursos dependum (**Prices for each Level** e **Price to charge the publishing house**) entre os atores **PAS** e **Comercial Dep. Employee** onde esses recursos foram transformados em atributos da classe **Level**. Assim, duas associações devem ser gerada entre as classes **Commercial Dep. Emp**., **PAS** e **Level**.

- associação entre **Commercial Dep. Emp e Level**
- associação entre **Level** e **PAS**

Diretriz 4.6: Para uma classe resultante da transformação de um recurso físico que é definido dentro da fronteira de um ator (recurso interno), uma associação é criada entre esta classe de recurso e a classe resultante da transformação do ator respectivo (aquele que contém o recurso).

As associações derivadas por esta diretriz podem ser novamente geradas a partir da nova Diretriz 4.7, pois nenhum recurso físico está isoladamente (solto) dentro de um ator e geralmente este recurso interno pode ser um dos argumentos de uma tarefa desse ator.

Como exemplo, o recurso físico **Proceedings Manual**, contido na fronteira do ator **Photo Agency System**-PAS, que é transformado em uma classe. Portanto, uma associação é gerada entre as classes derivadas a partir da transformação de **Proceedings Manual** e **PAS** no diagrama de classes em OO-Method.

Como foi introduzido o Ator Sistema, algumas dessas diretrizes do processo original (Alencar, 2009) necessitariam ser refinadas para excluir relacionamentos de associação desnecessários, no modelo final OO-M, entre a classe gerada a partir do ator Sistema e as demais classes derivadas dos demais atores e recursos físicos. Poder-se-ia até definir uma nova diretriz para transformar esses relacionamentos em agregações da classe gerada a partir do Ator Sistema. Algumas outras diretrizes passíveis modificações, por exemplo:

- A regra 4.4 é uma instância (caso específico de 4.3) onde o dependum é recurso. Sendo assim, outra possível modificação seria fundir essas duas diretrizes em uma.

Além das diretrizes até agora vistas, revisadas e validadas com exemplos, em relação ao processo definido em (ALENCAR, 2009), o processo **OOM-NFR** adiciona três novas diretrizes (**4.7, 4.8 e 4.9)** que serão especificadas a seguir:

Diretriz 4.7: Caso uma tarefa interna a um ator foi transformada em transação da classe derivada desse ator (Diretriz 3.4), faz-se a seguinte análise: **Se** essa transação tem argumentos derivados de recursos físicos distintos ou tem argumentos derivados de recursos informativos relacionados a diferentes recursos físicos, **então** é gerada uma associação entre as classes obtidas a partir desses recursos físicos distintos ou entre as classes às quais pertencem os recursos informativos.

Esta nova diretriz identifica a transação **To process a work request** (Work Request, Proceedings Manual) e gera uma associação entre **Work Request e Proceedings Manual**.

Diretriz 4.8: Para qualquer ligação de dependência entre os dois atores depender e dependee transformados em classes, onde exista uma tarefa que fora transformada em serviço de uma classe no lado do Dependee, criar também um **agent link** da classe gerada do Depender para a classe gerada Dependee.

A partir desta nova diretriz são gerados os relacionamentos OO-Method do tipo agent link. Alguns desses relacionamentos são:

- **User e PAS**
  -To Log in System
  -To Register Users in System
- **Photographer e PAS**
  -To process a work request

Como os relacionamentos agent link são entre classes clientes e classes servidoras, além dessas classes envolvidas, deve-se especificar também o(s) serviços da classe servidora para os quais a classe cliente aponta.

Diretriz 4.9: Cada tarefa transformada em serviço que estava dentro da fronteira de um ator transformado em classe, caso essa tarefa pertença a outra classe diferente desse ator, criar um

relacionamento de Agente (agent link) entre esta classe derivada do ator e a classe a qual pertence essa respectiva tarefa.

Esta nova diretriz foi criada pela necessidade de também gerar relacionamentos agent link que incluíssem as classes geradas a partir de recursos físicos. Alguns agent links gerados por esta diretriz:

- **Photographer** e **Work Request (Submit work request)**
- **Production Depart. Employee e Work Request (To send work request for judgment)**
- **Production Depart. Employee e "Refused work request" (To process the accepted work request)**
- **Production Depart. Employee e "Accepted work request" (To process the accepted work request)**
- **Board Administrator Employee e Work Request (To shelve the refused work request)**

4.4 Modelos OO-Method Gerados

Como resultado da aplicação das diretrizes ou regras de transformação especificadas na seção 4.3 anterior, são gerados os modelos objeto (ver seção 3.2.1.3) do OO-Method. Para gerar esses modelos, o Processo Núcleo OOM-NFR recebeu como entrada os modelos i* refatorados na etapa final de Pré-processamento apresentados nas seções 4.2.2, 4.2.3 e 4.2.4. Com isso, o principal objetivo do processo OOM-NFR foi alcançado, ou seja, a partir da análise, tratamento e priorização dos requisitos não-funcionais especificados em i* wiki, obteve-se, como resultado do processo transformativo, diferentes modelos objeto do OO-Method. As figuras a seguir mostram esses modelos com os respectivos comentários.

Na seção de **apêndice** deste trabalho, está listado o código fonte em QVT (QVT, 2009) e os respectivos comentários da automatização dessas transformações realizadas. Esse esforço em automatizar o processo transformativo OOM-NFR na linguagem de transformação de modelos QVT teve como vantagem tornar a atividade de gerar modelos objeto OO-M a partir do i* mais rápida, menos propensa a erros humanos, entre outras vantagens, citadas na introdução deste trabalho, quando se aborda os objetivos de um processo MDD.

Como descrito no apêndice, esta linguagem de transformação de modelos, baseia-se nos metamodelos ECORE das abordagens de origem e destino; recebe uma instância do modelo de origem no formato XMI e gera outra instância do modelo destino também em XMI. Como não havia disponibilidade de editor gráfico para ler e editar o modelo OO-Method gerado no formato XMI através da linguagem de processamento de modelos QVT, os construtores do OO-Method gerados foram desenhados ou editados graficamente através do Eclipse UML2 (ECLIPSE, 2010).

A figura 64 representa o modelo OO-Method gerado a partir da configuração 1 da seção 4.2.2 (primeira análise NFR) denominada:

Configuração 1: Prioriza-se Segurança, Usabilidade e Rapidez (Análise Backward)

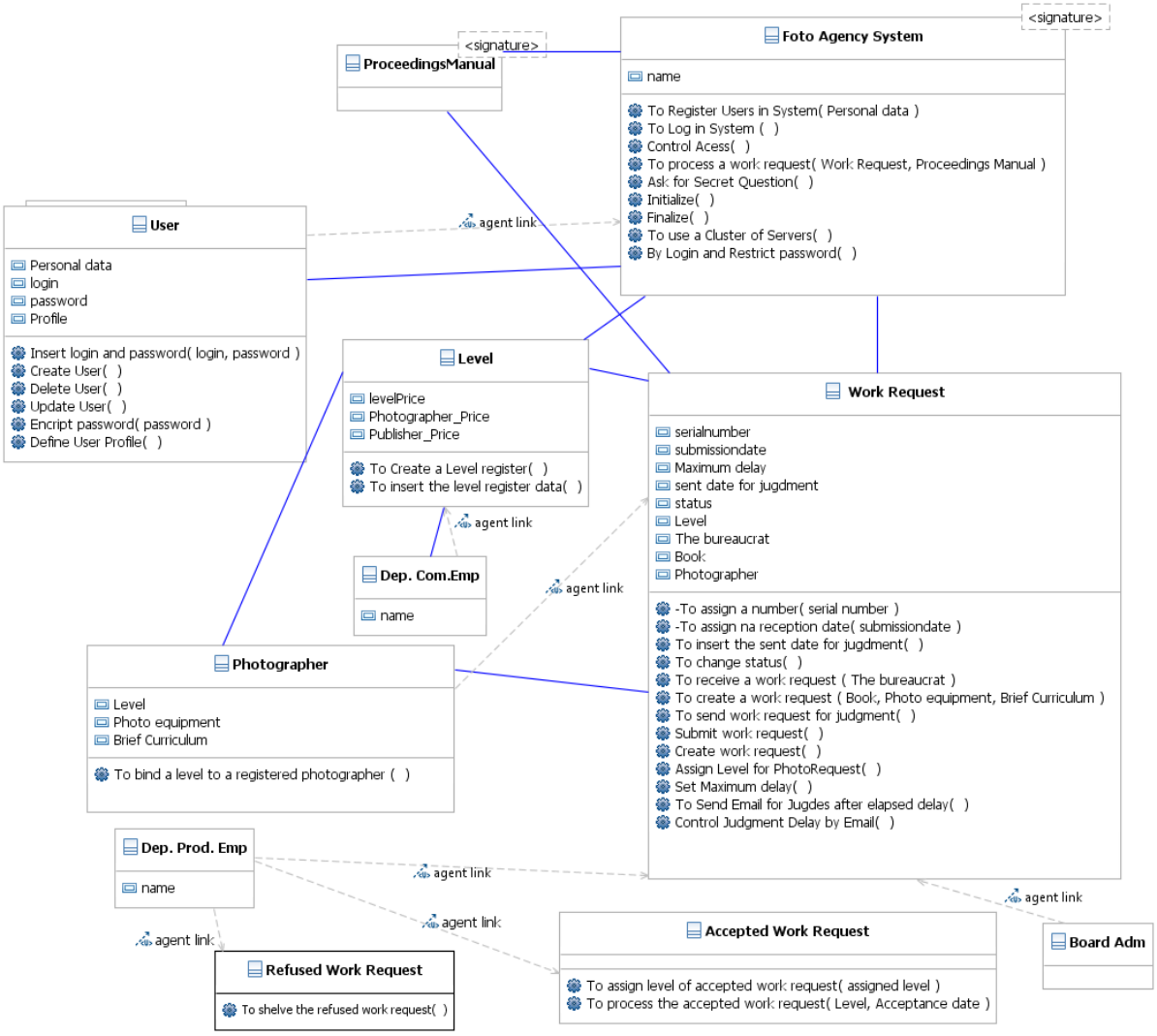


Figura 64. Modelo OO-Method gerado a partir da Configuração 1

Nesta figura 64, percebe-se que foram removidos os serviços:

- **Control Judgment Delay by Manual Form** da Classe **Work Request**
- **Use CAPTCHA** da Classe **Photo Agency System**

A figura 65 a seguir representa o modelo OO-Method gerado a partir da configuração 2 da seção 4.2.3 (segunda análise NFR) denominada:

Configuração 2: Prioriza-se Segurança, mas Quebra-se Usabilidade e Rapidez Julgamento (Análise Forward)

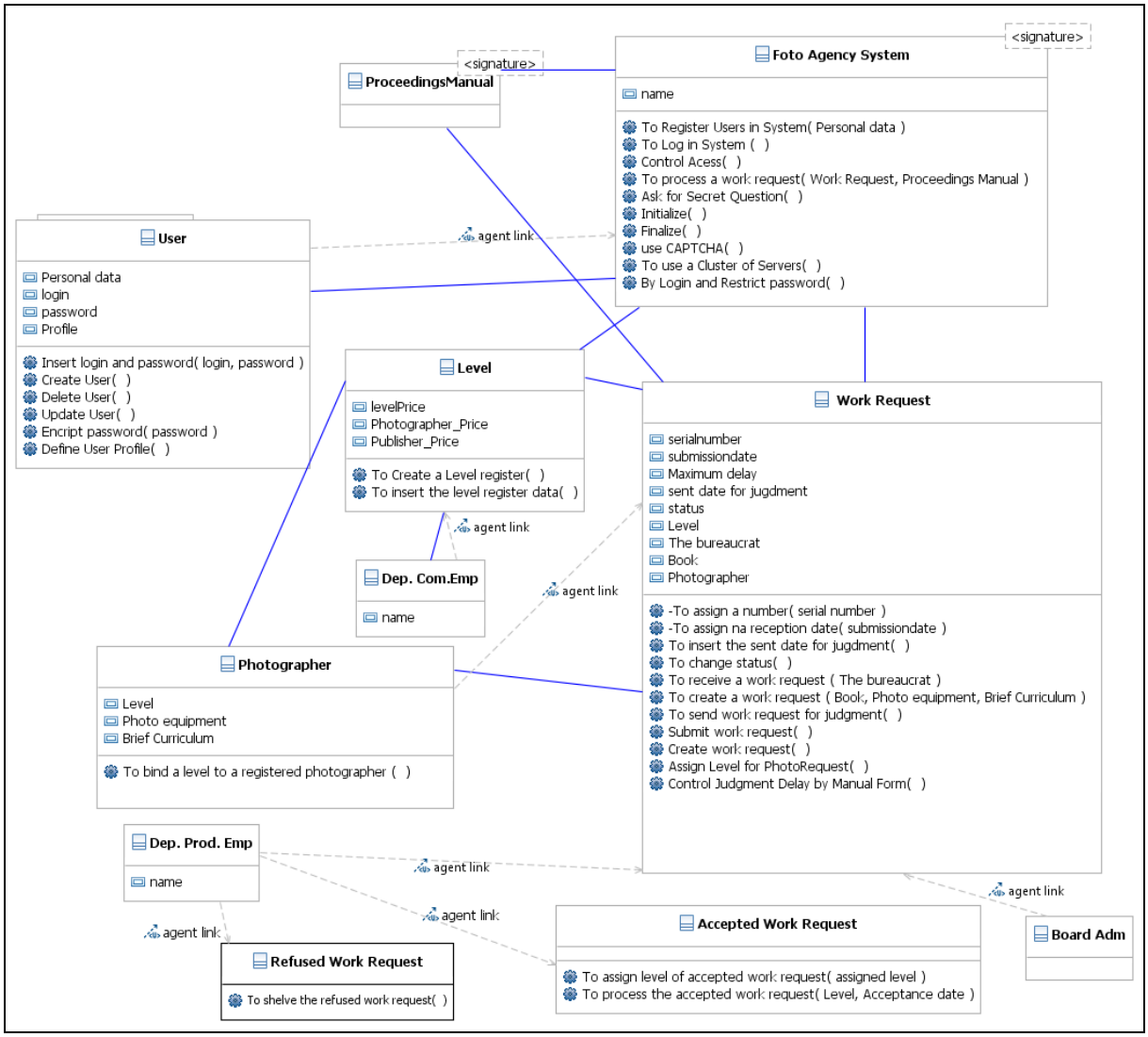


Figura 65. Modelo OO-Method gerado a partir da Configuração 2

Nesta figura 65, observa-se que foram removidos do modelo os seguintes serviços da classe **Work Request**:

- **Control Judgment Delay by Email**

- **To Send Email for Jugdes after elapsed delay**
- **Set maximum delay of X days for jugdment**

Entretanto o serviço **use CAPTCHA** da Classe Sistema **PAS** foi mantido, pois é responsável por quebrar a usabilidade.

Por fim, a figura 66 representa o modelo OO-Method gerado a partir da configuração 3 da seção 4.2.4 (terceira análise NFR) denominada:

Configuração 3: Quebra-se Segurança, Prioriza-se Usabilidade e Rapidez Julgamento (Análise Forward)

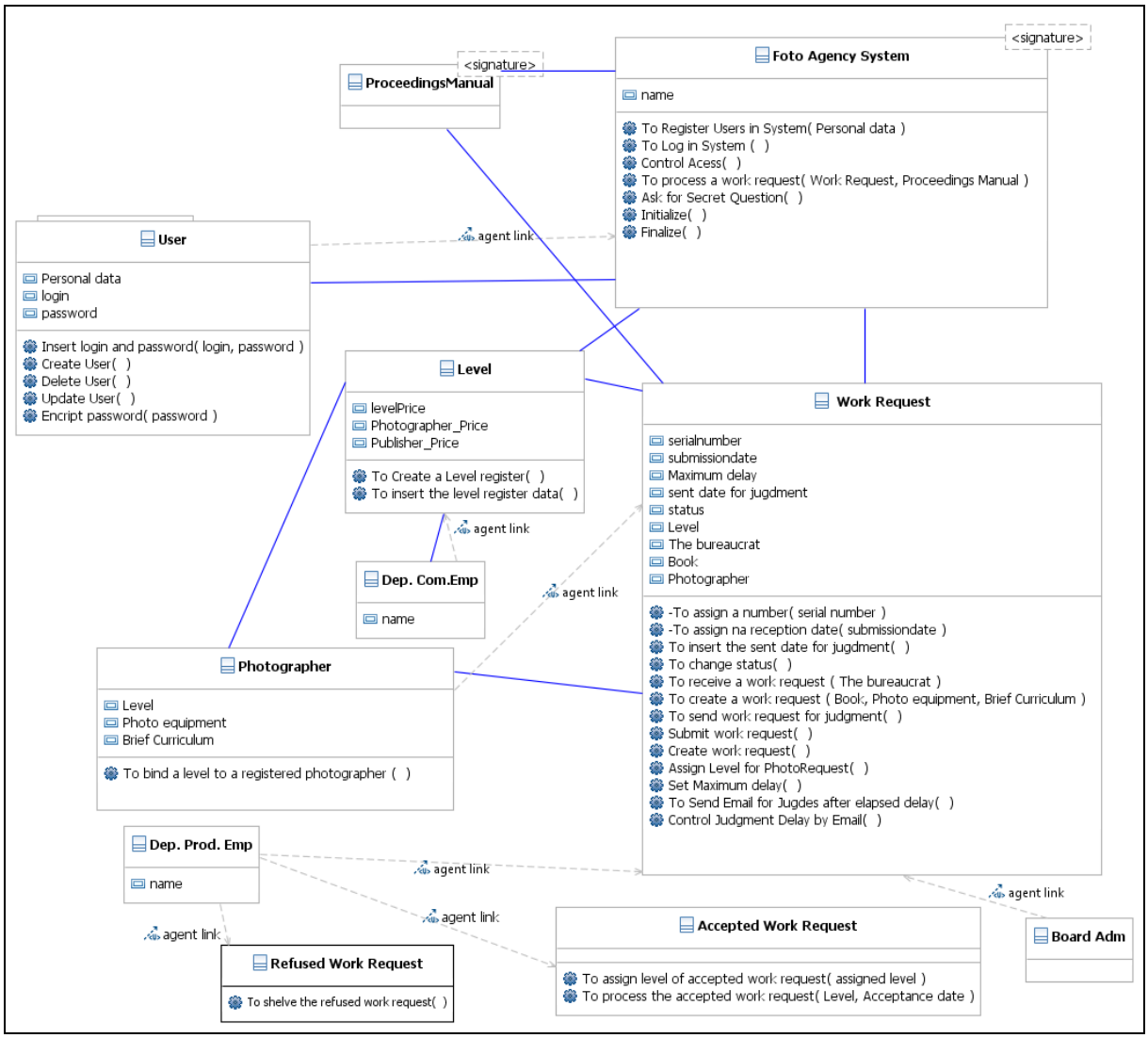


Figura 66. Modelo OO-Method gerado a partir da Configuração 3

Na figura 66, observa-se que:

Estão removidos do modelo os serviços da Classe **Photo Agency System:**

- **Use CAPTCHA**
- **By Login and Restrict password**
- **To Use a Cluster of Servers**

Estão removidos da classe **WorkRequest**:

- **Control Judgment Delay by Manual Form**

Estão removidos da classe **User** (Usuário):

- **Serviço: Define User Profile**
- **Atributo: Profile**

Como se observa em todos os modelos as classes geradas estão com os devidos relacionamentos: associação e agent link. Nota-se também que a maior parte das classes que representam usuários do sistema tem relacionamentos de agent link para com as classes Photo Agency System (ator sistema em i* wiki) e Work Request (principal recurso físico processado pelo modelo). Isso se dá pelo fato dessas classes serem clientes, ou seja, terem visibilidade para os serviços dessas principais classes servidoras Photo Agency System e Work Request.

A geração desses três modelos distintos em OO-Method só foi possível porque o processo OOM-NFR trata, analisa e prioriza os softgoals (requisitos não-funcionais relacionados a produto e interno ao ator sistema) em i* wiki. Isso não ocorre na abordagem de (ALENCAR, 2009), nem nas demais abordagens, conforme mencionado na seção 3.3, onde se faz essa mesma transformação sem contemplar softgoals e se gera apenas um único modelo OO-M a partir do qual não se é capaz de se afirmar se esses softgoals estão satisfeitos (prioridades) e também estão sem conflitos.

Não fosse o pré-processamento (Análise NFR) realizado pelo OOM-NFR, seria gerado apenas um único modelo objeto do OO-Method com todos os serviços gerados a partir das tarefas de i*, pois não seriam considerados os softgoals do modelo de entrada em i*.

4.5 Considerações Finais

Este capítulo apresentou a proposta deste trabalho de pesquisa condensada através do processo OOM-NFR e que objetiva a transformação de um modelo i* para o modelo OO-Method com foco em softgoals. Inicialmente, na seção 4.1 foi ilustrada a visão geral do

processo OOM-NFR e seus principais componentes: o Pré-processamento e o Processo Núcleo.

No Pré-processamento (seção 4.2), foram elencadas as atividades responsáveis por realizar a análise dos requisitos não-funcionais. A primeira atividade é verificar se há softgoals no modelo i* a ser processado. Para exemplificar as etapas do processo OOM-NFR foi escolhido, como modelo de entrada, o sistema de agência fotográfica. A principal atividade do pré-processamento é a análise NFR onde é realizada a julgamento dos requisitos não-funcionais, verificando quais deles devem ser priorizados e satisfeitos. Nessa etapa, o analista pode escolher entre a análise "Forward" ou "Backward" (HORKOFF et al. 2010), conforme sua necessidade (ver vantagens e desvantagens na seção 2.4.1) e interage com mecanismo de resolução de fórmulas formalizadas em i* através de um SAT-Solver implementado na ferramenta OpenOME (OPENOME, 2010). Nessa interação, conflitos são resolvidos, prioridades são analisadas, resultando numa configuração final de modelo i* satisfatória. A atividade final do pré-processamento é refatorar essa configuração final para remover os elementos negados e que não vão ser passados para a próxima etapa do processo OOM-NFR: o processo núcleo.

Para demonstrar esse Pré-processamento foram dados três exemplos de análise NFR baseados no modelo i* agência fotográfica. Cada exemplo analisado tinha prioridades distintas (diferentes) em termos dos softgoals, sendo geradas configurações finais também diferentes para atender a cada conjunto de prioridades definida pelo analista.

O Processo Núcleo, seção 4.3, tem como principal atividade transformar o modelo i* para o modelo OO-Method através da aplicação das diretrizes de transformação. O processo núcleo transformativo do OOM-NFR é dividido basicamente em quatro atividades: geração de classes, geração de atributos, geração de serviços e geração de relacionamentos.

Cada uma dessas quatro atividades são decompostas em diretrizes que definem quais elementos i* serão transformados nos elementos semanticamente equivalentes em OO-Method e também como é realizada essa transformação. Ao longo da definição dessas diretrizes são dados exemplos do modelo i* agência fotográfica em que elas podem ser aplicadas.

A atividade final do processo OOM-NFR é um refinamento do modelo OO-Method incialmente gerado, pois como foi mencionado há transformações que dependem da intervenção do analista para ser concluída.

No próximo capítulo 5, será estudada mais detalhadamente, a aplicação do processo OOM-NFR através de um exemplo de sistema de controle de acesso modelado em i*.

CAPÍTULO 5

5. Exemplo de Aplicação do OOM-NFR

Neste capítulo é estudado um exemplo de aplicação do processo OOM-NFR proposto nesta dissertação, visando melhor ilustrar e demonstrar mais detalhadamente as suas atividades. Este estudo é realizado através de um sistema de informação de controle de acesso.

O capítulo faz inicialmente uma contextualização do sistema escolhido e descrição dos objetivos deste exemplo. Na seção 5.3, é modelado o sistema de controle de acesso em i* wiki e são explicados alguns de seus componentes principais. A partir da seção 5.4 é ilustrado e aplicado o processo de transformação OOM-NFR, descrevendo seus subprocessos básicos: Pré-processamento e Processo Núcleo.

Na etapa de pré-processamento são realizados dois exemplos de análise NFR. Em cada análise são priorizados diferentes softgoals, ou seja, aqueles relacionados a requisitos não-funcionais de produto (SOMMERVILLE, 1998) internos ao ator sistema (ver seção 2.2), resultando, ao fim dessa etapa, modelos i* com os elementos necessários para satisfazer ou não esses softgoals de acordo com as prioridades consideradas.

A seção 5.5 dedica-se ao processo núcleo do OOM-NFR que é responsável pelas atividades que, de fato, realizam a transformação dos elementos i* para elementos do modelo objeto (ver seção 3.2.1.3) do OO-Method. Cabe-se ressaltar que o modelo objeto é apenas um dos modelos do esquema conceitual do OO-Method (ver seção 3.2.1.3) e que o processo OOM-NFR tem como escopo apenas esse **modelo objeto**. Na seção 5.6 são mostrados os modelos objetos do OO-Method gerados a partir das duas análises NFR realizadas. Por fim, é realizado um resumo do capítulo e são tecidas as considerações finais sobre o exemplo desta aplicação do processo OOM-NFR ao sistema de controle de acesso.

5.1 Contextualização

O Serviço Federal de Processamento de Dados (SERPRO) tem como padrão de acesso a sistemas de informação, um sistema de controle de acesso que possibilita autenticação e autorização de usuários. O sistema segue o padrão conhecido como RBAC (Controle de Acesso baseado em Papéis), onde papéis, denominados de perfis, são previamente definidos com a autorização e privilégios requeridos. Além disso, esse sistema ofereçe a qualquer módulo ou aplicação desenvolvida um ponto central de "logon" para seus usuários e funcionalidades de manutenção da infraestrutura de segurança desses aplicativos.

Esse tipo de sistema de informação foi escolhido por apresentar vários softgoals interessantes tais como segurança, desempenho (rapidez), usabilidade, entre outros. que podem ser modelados em i* wiki e priorizados para serem submetidos ao processo de transformação proposto OOM-NFR.

O sistema controle de acesso (SCA), especificado neste trabalho, representa uma visão simplificada do sistema real em produção e utilizada única e exclusivamente para os propósitos acadêmicos desta pesquisa.

Além disso, algumas modificações adicionais foram feitas para melhor se adequar ao principal objetivo deste trabalho que é focar os requisitos não-funcionais. Entretanto, as idéias e funcionalidades aqui apresentadas podem refletir perfeitamente as necessidades de qualquer sistema de informação que verse sobre controle de acesso.

Conceitos elementares do SCA:

- Aplicação: qualquer sistema, subsistema ou módulo gerenciado pelo SCA. Por exemplo, Sistema Financeiro, Sistema de Créditos e Recursos Humanos podem ser aplicações gerenciadas pelo SCA.
- Usuário: qualquer pessoa, que para acessar alguma aplicação específica, precisa autenticar-se (realizar logon) e autorizar-se (possuir um ou mais perfis dessas aplicações para poder ter permissão de utilizá-las).
    - Formas de autenticação:
        - Login/senha
            - Nessa modalidade de logon pode-se utilizar o recurso CAPTCHA, ou seja, aquele recurso muito usado nos sistemas de logon que solicitam para o usuário digitar as letras ou

números de uma imagem exibida para se ter acesso ao mesmo. CAPTCHA contribui negativamente para a usabilidade, mas contribui positivamente para segurança, caracterizando assim um conflito. Isso é um impedimento total de acesso aos sistemas para um deficiente visual, pois os leitores de tela atuais do mercado não conseguem processar o conteúdo dessas imagens.

  - Certificado Digital;
  - Biometria;

- Administrador: usuário específico responsável por:
  - cadastrar qualquer usuário geral e atribuir os perfis desse usuário geral;
  - manter a infraestrutura básica de acesso dos sistemas (aplicações) gerenciadas pelo SCA.
- Perfil: entidade que dá a um usuário autorização para realizar uma ou mais transações de uma aplicação. Geralmente um perfil está associado a uma ou mais transações.
- Transação: entidade única que representa um **caso de uso** de uma aplicação. Em geral, cada caso de uso de uma aplicação é uma transação, ou seja, uma aplicação está associada a várias transações.
- Role: entidade de banco de dados que agrega todos os privilégios de banco relacionados a uma determinada aplicação e necessários para executar todas as transações dessa Aplicação. Geralmente há apenas uma Role de Banco para cada aplicação. Por exemplo, Recursos Humanos tem a role RH_Role que agrega os privilégios de banco necessários para executar todas as transações (casos de uso) do RH. Como essa role é critica em termos de segurança, ela tem sua senha criptografada e mudada periodicamente.

Cada sistema ou aplicação para ser gerenciada pelo SCA precisa que sua infraestrutura básica, ou seja, **Aplicação** (dados gerais da aplicação), **Transações**, **Perfis** e **Roles** seja cadastrada e mantida. Para isso, o SCA possui em seu modelo as tabelas **Aplicação, Transações, Perfis e Roles** para manter essa infraestrutura. Por exemplo, a aplicação Recursos Humanos (RH) para funcionar no SCA, antes de atribuir perfis de RH aos usuários gerais, faz-se necessário cadastrar toda infraestrutura de controle acesso do RH: Dados da Aplicação RH, Transações do RH,

Perfis do RH, Associar Perfis a Transações e cadastrar a Role RH. A necessidade de associar Perfis a Transações é fundamental para aumentar ou reduzir os privilégios. Por exemplo, pode-se definir um perfil chamado **Gerencia_RH** que estaria associado a todas as transações da aplicação RH para que, quando esse Perfil fosse atribuído a um determinado usuário exercendo o papel de "Gerente", este pudesse ter acesso a todas funcionalidades dessa aplicação. Poder-se ia definir também perfis com menos privilégios, por exemplo, o perfil **Consulta_RH** associado apenas às transações (casos de uso) de consulta e que teria apenas autorização de realizar consultas básicas sobre informações RH dos usuários.

- Mecanismo Básico de Autorização: Uma vez o usuário autenticado, para cada transação (caso de uso) da aplicação que ele deseje utilizar é realizada a tarefa de **Autorizar Acesso**. Essa tarefa consiste em verificar os perfis associados ao usuário e as respectivas transações desses perfis para verificar se ele realmente tem ou não autorização de acesso. Após feita essa verificação lógica, caso o usuário obtenha acesso, é concedida a permissão física de banco.
- Mecanismo Básico de Logs e Históricos: Por questões de segurança e visando atender a necessidades de auditorias, o SCA possui um complexo sistema de historiamento de tudo que é acessado e autorizado pelo mesmo. Periodicamente, são realizadas demandas de Auditorias sobre aspectos gerais das Aplicações ou para analisar algum incidente de segurança. Essas Auditorias podem ser realizadas através de consultas já automatizadas ou manuais (que precisam ser particularmente desenvolvidas e implementadas) sobre os Históricos existentes. Dados e resultados das auditorias são consolidados num relatório e são estritamente confidenciais, devendo ser totalmente criptografados.

  Na próxima seção serão delineados alguns dos objetivos da realização deste exemplo.

5.2 Objetivos

O principal propósito deste exemplo é ilustrar como o processo transformação MDD proposto pode ser aplicado a um sistema de informação real e em produção, modelado a nível de requisitos em i* wiki, onde é gerado um modelo inicial a nível de projeto em OO-Method que reflete também os softgoals especificados (requisitos não-funcionais relacionados a produto e interno ao ator sistema).

Alguns outros objetivos específicos podem ser derivados desse objetivo principal:

- Análise, priorização e tratamento de conflitos entre os softgoals;
- Explorar a capacidade de análise estratégica do sistema de controle de acesso (SCA) por estar sendo remodelado na linguagem i* wiki que dá suporte a isso;
- Possibilidade de identificação de possíveis vulnerabilidades para posterior tratamento;
- Possibilidade de otimização e reengenharia dos processos envolvidos no SCA;

5.3 Sistema de Controle de Acesso em i* wiki

O modelo de requisitos original do SCA é especificado em UML: Casos de Usos, Diagramas de Casos de Uso e outros artefatos, conforme processo de desenvolvimento de software utilizado pelo Serpro (PSDS-Processo Serpro de Desenvolvimento de Soluções) baseado no Rational Unified Process (RUP).

Sendo assim, o sistema já estava com seus requisitos especificados e validados em UML, não havendo necessidade de realizar novamente elicitação, negociação e validação. Então, a partir desses artefatos e conhecimento do domínio da aplicação, o SCA foi remodelado em i* wiki a fim de ser utilizado como exemplo de aplicação do processo transformativo MDD proposto. Apenas alguns softgoals foram adicionados para melhor ilustrar o exemplo.

A modelo i* da razão estratégica final " SR final" do SCA, cujos requisitos foram descritos na seção 5.1 (contextualização) mais os softgoals, está representado na figura 67 a seguir. Este diagrama do SCA foi modelado inicialmente com a ferramenta Organization Modelling Environment (OME, 1998). Entretanto, por ser um diagrama extenso e ter ficado ilegível, quando transposto para este documento, teve de ser redesenhado com a ferramenta da Microsoft Visio (VISIO, 2010) a fim de melhor visualização.

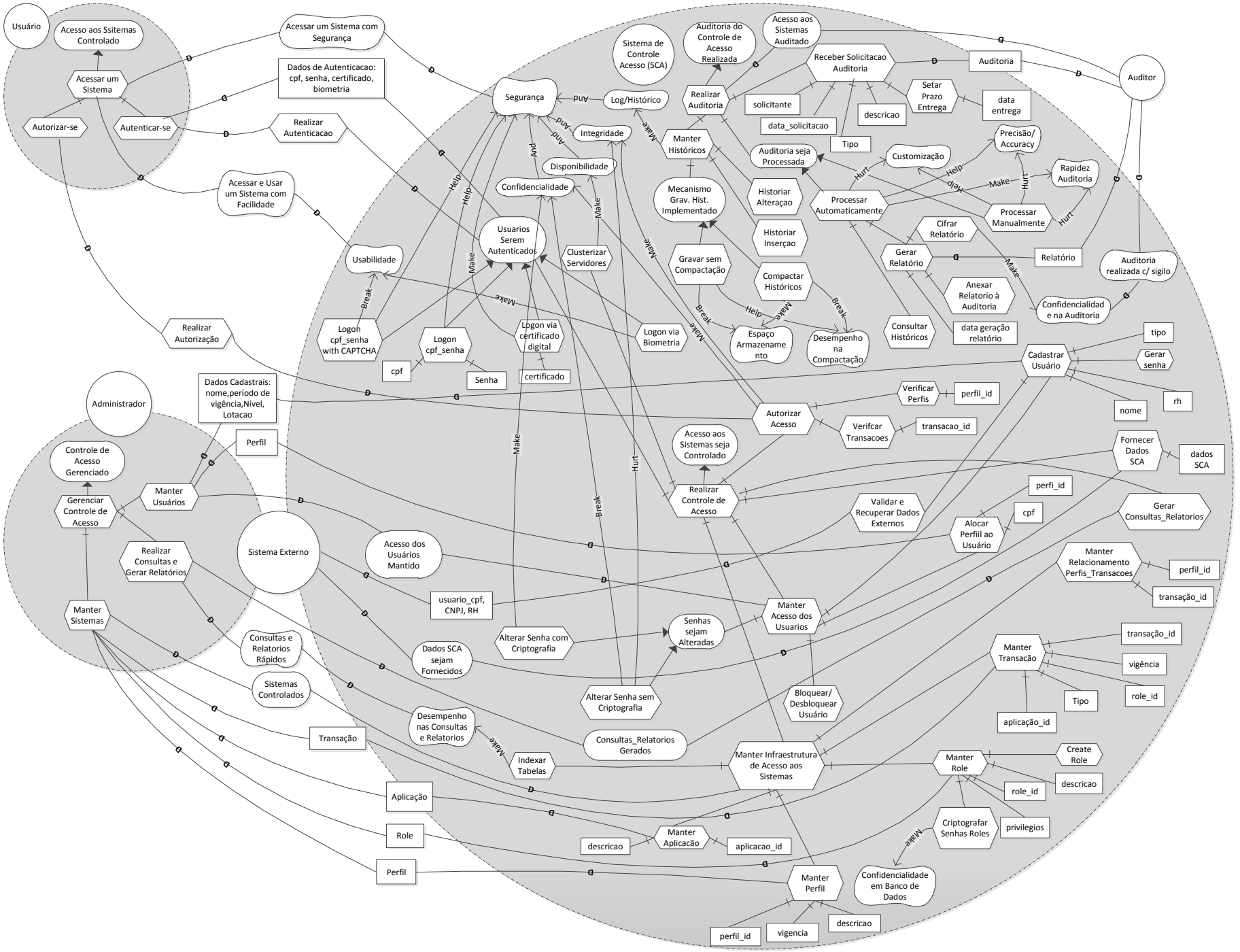


Figura 67. Modelo Sistema de Controle de Acesso (SCA) em i* wiki (diagrama geral)

Neste diagrama, ilustrado na figura 67, estão modelados em i* wiki os conceitos elementares já vistos anteriormente, na seção 5.1, sobre o SCA, tais como: aplicações, transações, roles, perfis, auditoria e atores.

Os principais atores modelados são: Usuário, Administrador, Auditor, Sistema Externo e o ator Sistema de Controle de Acesso (SCA).

Observam-se também vários softgoals: **Usabilidade, Segurança, Espaço Armazenamento Reduzido, Rapidez na Auditoria, Confidencialidade Banco de Dados, Desempenho nas Consultas e Relatórios; e Precisão (Accuracy), Confidencialidade na Auditoria, Customização, Desempenho Gravação de Históricos e Log/Histórico.** Esses são os softgoals modelados para o Sistema de Controle de Acesso (SCA) e que serão utilizados para definição de prioridades e tratamento de conflitos. Várias tarefas modeladas estão contribuindo positivamente e/ou negativamente para satisfação ou não desses softgoals.

O softgoal **Segurança** recebe contribuições "AND" de quatro outros softgoals: confidencialidade, integridade, disponibilidade e Log/Histórico. Há vários elementos i* meio-fim, tais como: O objetivo "Usuários Serem Autenticados" pode ser satisfeito através de quatro mecanismos: Logon cpf_senha with CAPTCHA, Logon cpf_senha (sem CAPTCHA), Logon via Biometria e Logon via certificado digital. Essas tarefas de meio-fim geralmente têm links de contribuição para os vários softgoals modelados. Por exemplo: Logon cpf_senha with CAPTCHA contribui negativamente para usabilidade; já Logon via certificado digital contribui positivamente para usabilidade.

O objetivo "Mecanismo Grav. Hist. Implementado" pode ser viabilizado através de duas formas: Gravar sem Compactação e Compactar Históricos. Já o objetivo "Auditoria seja Processada" pode ver satisfeito também de dos modos: Processar Automaticamente e Processar Manualmente.

Nas seções posteriores serão apresentados e detalhados todos os passos do processo proposto OOM-NFR.

5.4 O Pré-Processamento

Como já visto na seção 4.1, o processo OOM-NFR é constituído de dois outros processos: o pré-processamento e o processo núcleo. Neste exemplo, não será exibidos novamente os diagramas destes processos, detalhando suas atividades específicas já

explicadas no capítulo 4. Será dado foco as atividades básicas desses processos, ou seja, a análise NFR e a transformação de modelos realizada através da aplicação das diretrizes.

5.4.1 Análise NFR 1

Nesta primeira Análise NFR do Pré-Processamento foram priorizados os seguintes softgoals do SCA: **Usabilidade, Segurança, Espaço Armazenamento Reduzido, Rapidez na Auditoria, Confidencialidade na Auditoria, Confidencialidade Banco de Dados e Desempenho nas Consultas e Relatórios**. Os demais softgoals: Desempenho Sistema, Precisão (Accuracy), Customização não são prioritários e serão deixados de lado para que o SAT resolva automaticamente, desde que não conflite com os prioritários escolhidos pelo analista.

Como já foi visto, após passar para o **OOM-NFR** um modelo i* softgoals, inicia-se a atividade "**Escolher Tipo de Análise NFR"** onde o analista define se a análise vai ser para frente "Forward" ou para trás "Backward". Nesta primeira análise NFR foi escolhida a análise para trás "**Backward**". Essa abordagem é mais eficiente, mais automática e completa do que a abordagem "forward", no sentido de ter capacidade de responder as questões "Como" e "Por quê" das decisões escolhidas durante o processo de resolução (HORKOFF et al. 2010). Na análise "backward" (top-down) são selecionados inicialmente os softgoals prioritários e que são raízes do diagrama i* SCA. A partir daí, o resolvedor SAT-Sover tenta iterativamente encontrar uma configuração que satisfaça a esses softgoals.

Para facilitar a compreensão e referência posterior, essa primeira análise composta por essa lista definida de prioridades de softgoals e tipo de análise, será chamada também de **configuração 1.**

De fato, nessa configuração 1, estão a maior parte dos softgoals encontrados no modelo i* SCA que o analista decidiu priorizar de acordo com sua política de segurança ou dentro de um determinado contexto ou circunstância do projeto. Essa configuração pode ser considerada também uma boa configuração para análise NFR e posterior geração do modelo em OO-Method.

Para facilitar a exemplificação do pré-processamento do OOM-NFR no SCA foi realizado um recorte reduzido do modelo i* desse sistema a fim de melhor visualização gráfica dos softgoals e elementos do i* sob interesse da análise. A ferramenta utilizada como suporte para análise foi o OPEN OME. A seguir serão mostrados os passos realizados,

detalhando uma possível análise NFR realizada pelo analista de requisitos com ajuda do SAT-Solver implementado nesta ferramenta.

Passo 1: Seleção dos Requisitos não-funcionais a serem satisfeitos (prioritários)

Neste passo, é realizada a atividade "**realizar de seleção do(s) sofgoal(s) alvos e predicado/nível de satisfação"** do pré-processamento **OOM-NFR.**

Como se observa, na figura 68 a seguir, o analista escolheu os seguintes softgoals para serem priorizados: Usabilidade, Segurança, Espaço Armazenamento Reduzido, Rapidez na Auditoria, Confidencialidade na Auditoria, Confidencialidade Banco de Dados e Desempenho nas Consultas e Relatórios. Como a análise é backward (top-down), foram escolhidos os softgoals que são raízes do diagrama i* SCA.

Passo 2: Seleção das priorizações para Segurança

Quando o analista inicia o processo de resolução, o Sat-solver tenta resolver, escolhendo, de acordo com as regras formalizadas do i*, uma configuração que satisfaça todos os softgoals priorizados. Como não consegue sozinho, exibe a tela (figura 69) a seguir alertando o analista que Segurança deve ser satisfeita e que faça as devidas alocações para que isso seja obtido. Simultanemente mostra essa situação no diagrama da figura 70, destacando em amarelo os elementos que estão sob análise NFR.

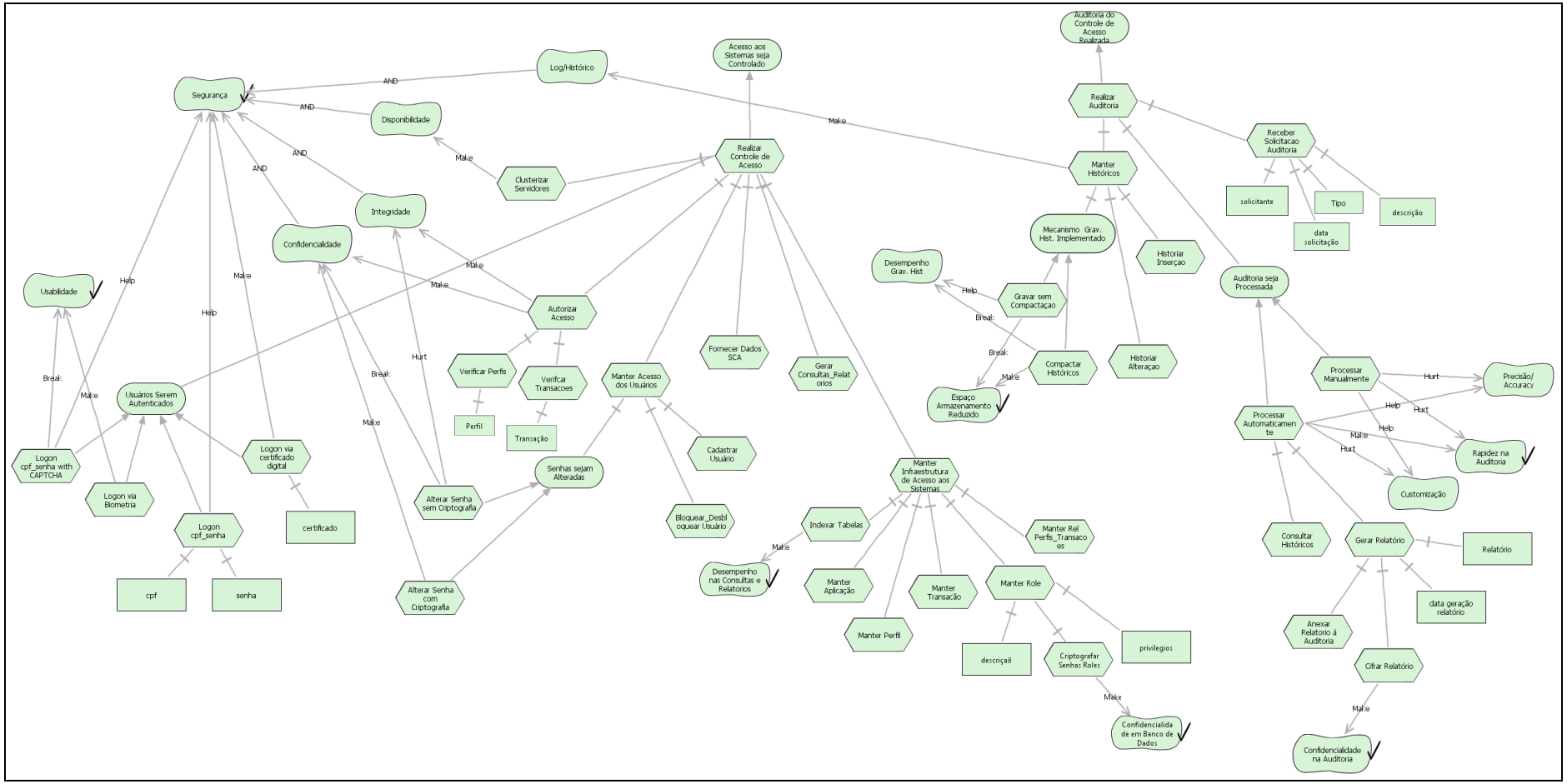


Figura 68. Softgoals prioritários da Análise NFR 1 - Backward

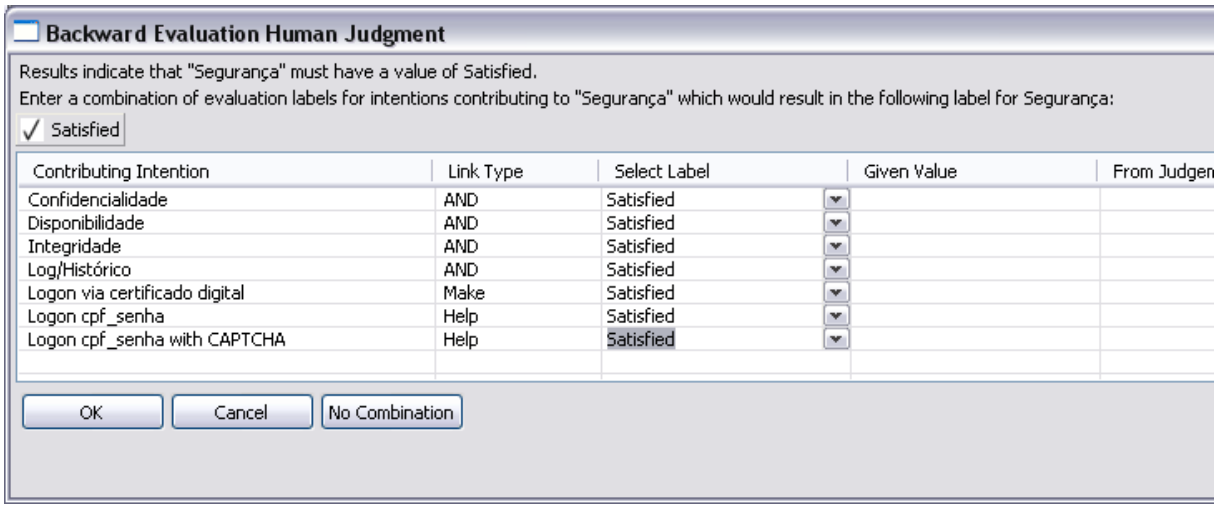


Figura 69. Tela decisão análise NFR 1 - passo 2

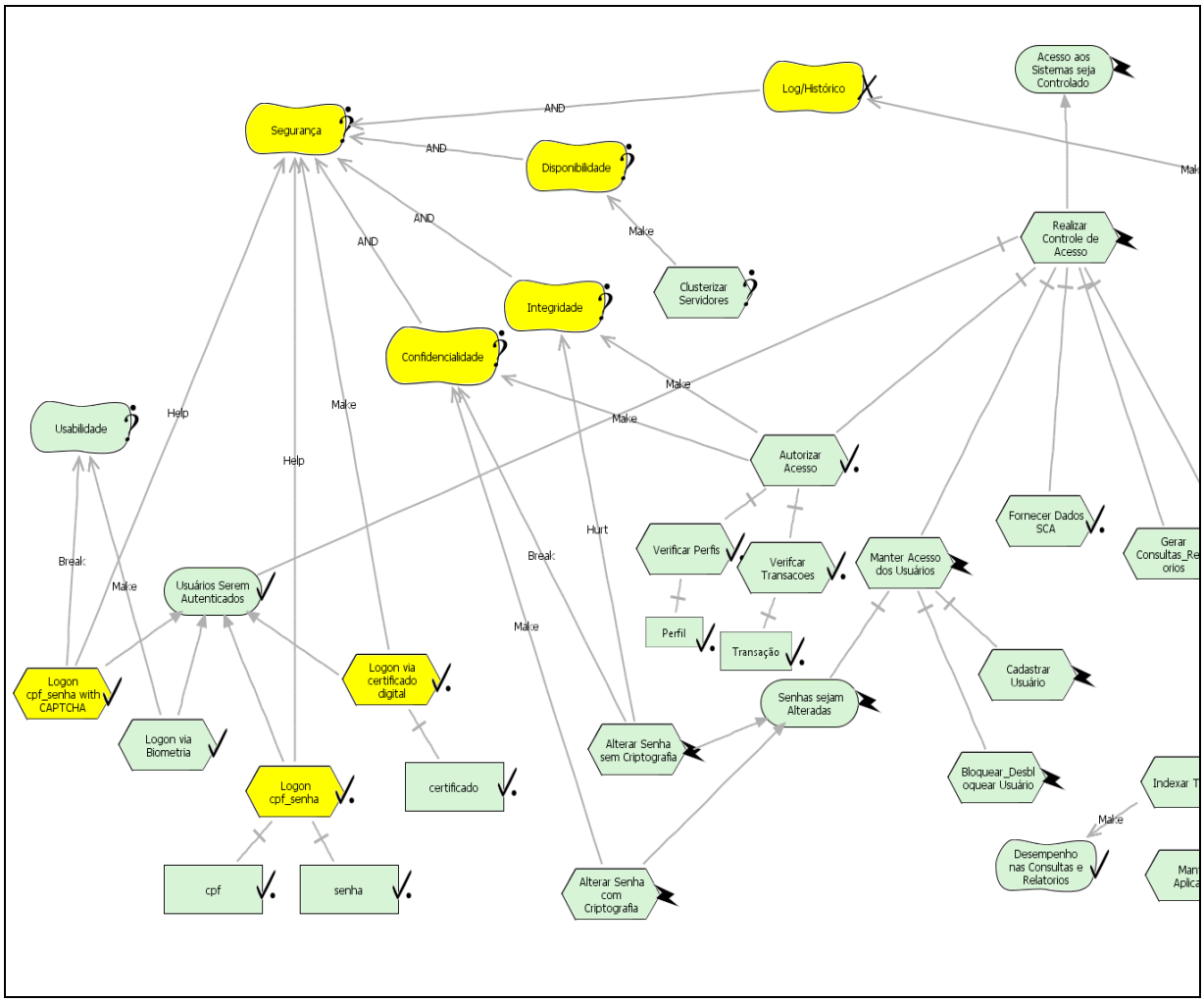


Figura 70. Diagrama parcial análise NFR 1 - passo 2

Como se observa nesse diagrama parcial gerado (figura 70), há vários elementos indefinidos e conflitantes. Isso é normal, pois ainda o resolvedor está em sua primeira iteração.

Passo 3: Resolução de Usabilidade

As figuras 71 e 72 a seguir mostram a resolução feita pelo analista e recorte de diagrama, focando os elementos envolvidos. Como usabilidade é um softgoal prioritário, o analista decide não satisfazer Logon cpf_senha with CAPTCHA que contribui negativamente para usabilidade; e satisfazer Logon via certificado digital que contribui positivamente para esse softgoal.

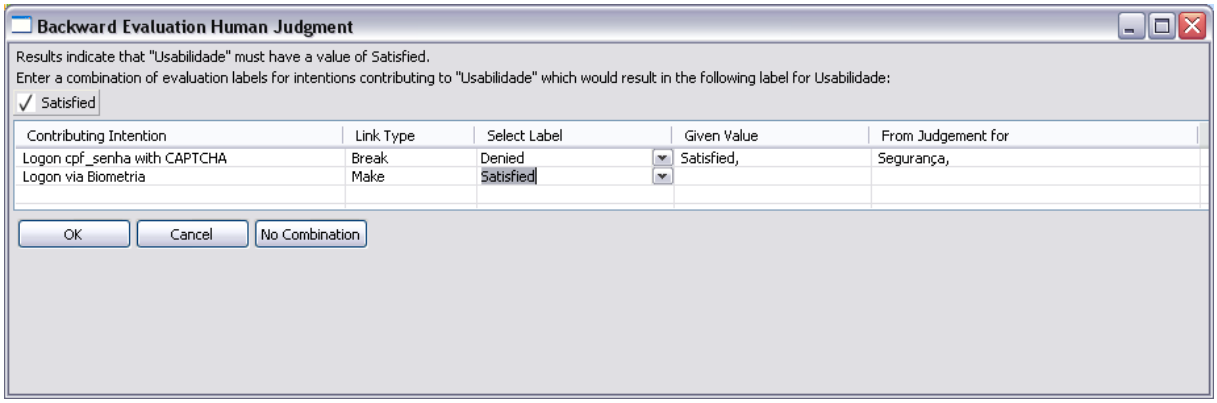


Figura 71. Tela decisão análise NFR 1 - passo 3

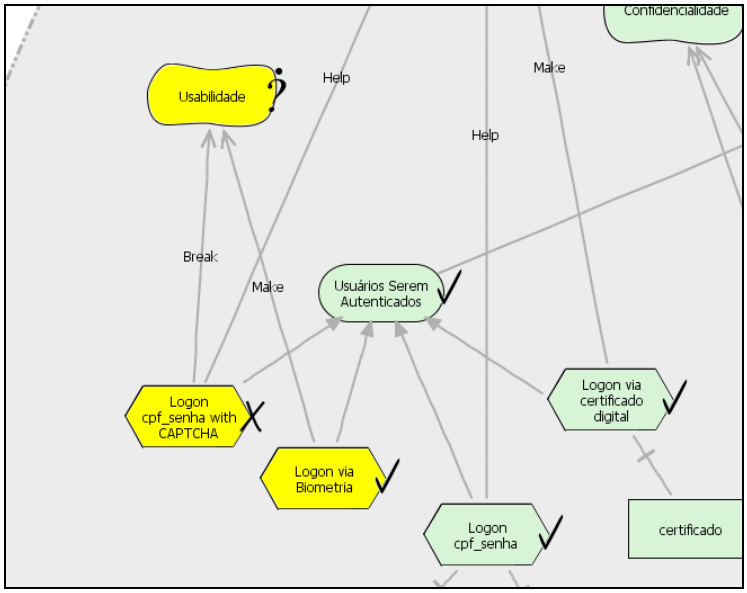


Figura 72. Diagrama parcial análise NFR 1 - passo 3

Passo 4: Analista prioriza Espaço Armazenamento Reduzido

Nas figuras 73 e 74 a seguir são atribuídos os valores para resolução e priorização de Espaço Armazenamento Reduzido e exibido sub-recorte da configuração parcial do modelo nesta etapa de resolução.

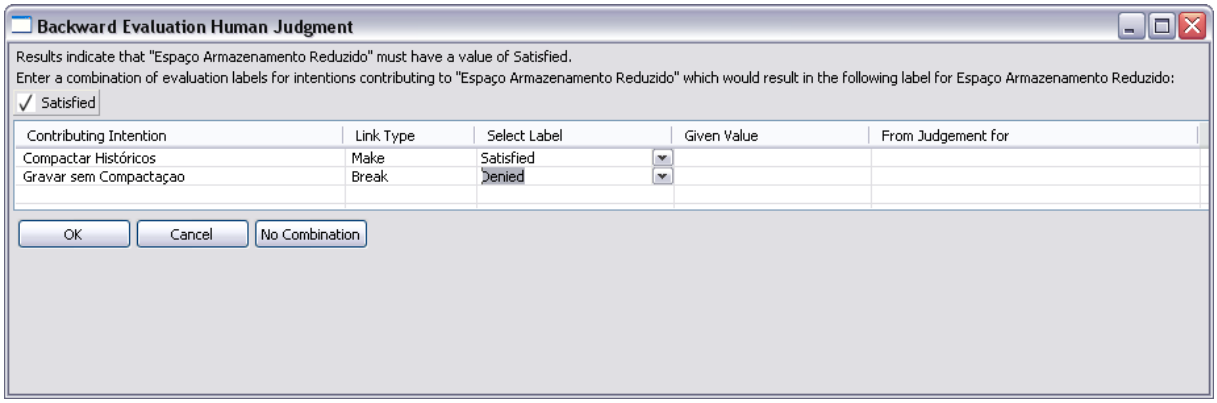


Figura 73. Tela decisão análise NFR 1 - passo 4

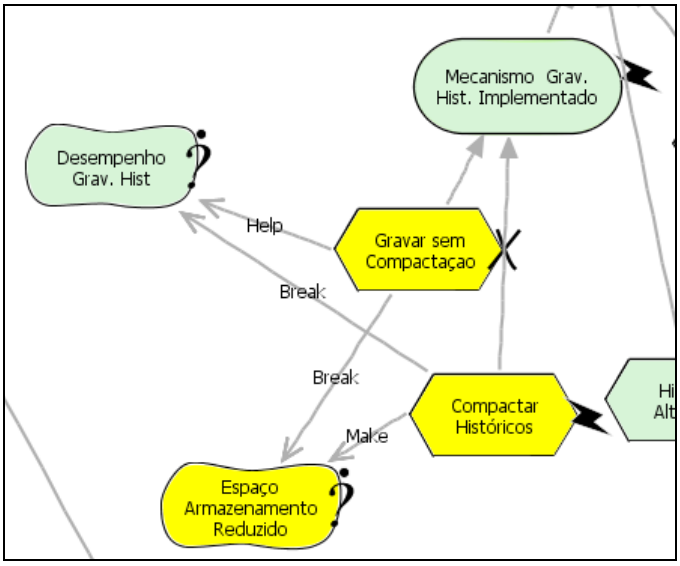


Figura 74. Diagrama parcial análise NFR 1 - passo 4

Passo 5: Rapidez na Auditoria

Nas figuras 75 e 76 a seguir são atribuídos os valores para resolução e priorização **Rapidez na Auditoria** e exibido sub-recorte da configuração parcial do modelo nesta etapa de resolução.

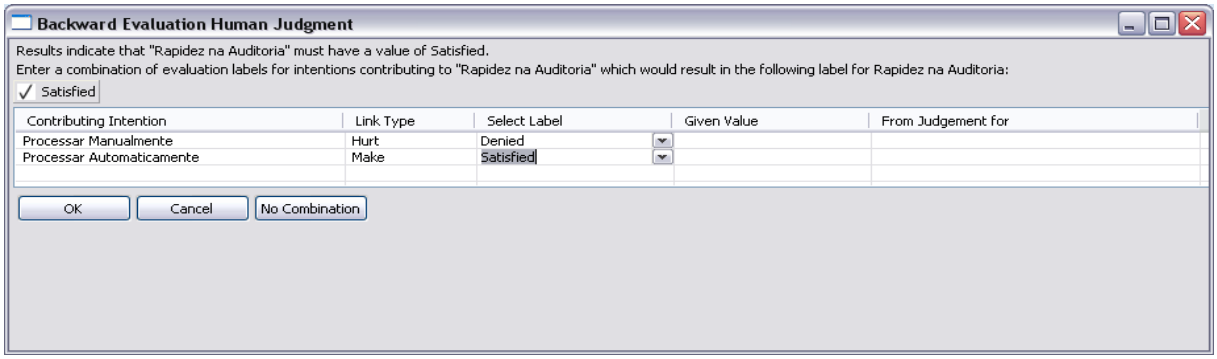


Figura 75. Tela decisão análise NFR 1 - passo 5

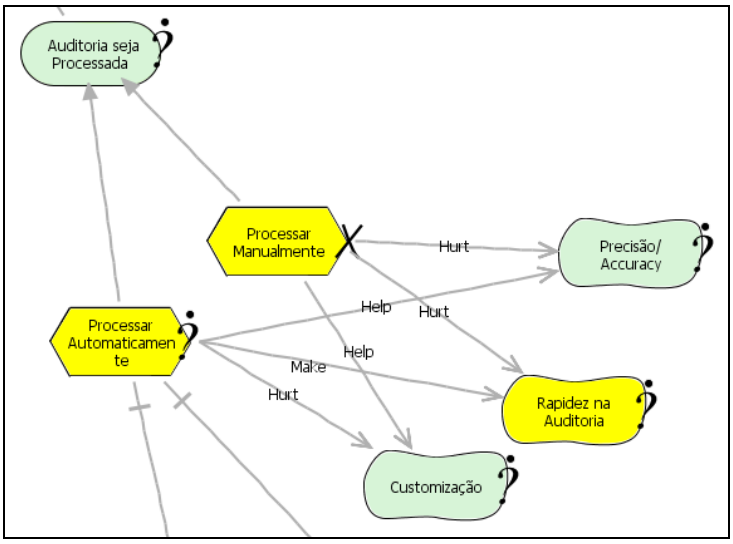


Figura 76. Diagrama parcial análise NFR 1 - passo 5

Passo 6: Resolução do RNF Confidencialidade

Nas figuras 77 e 78 a seguir são atribuídos os valores para resolução e priorização **Confidencialidade** e exibido sub-recorte da configuração parcial do modelo nesta etapa de resolução.

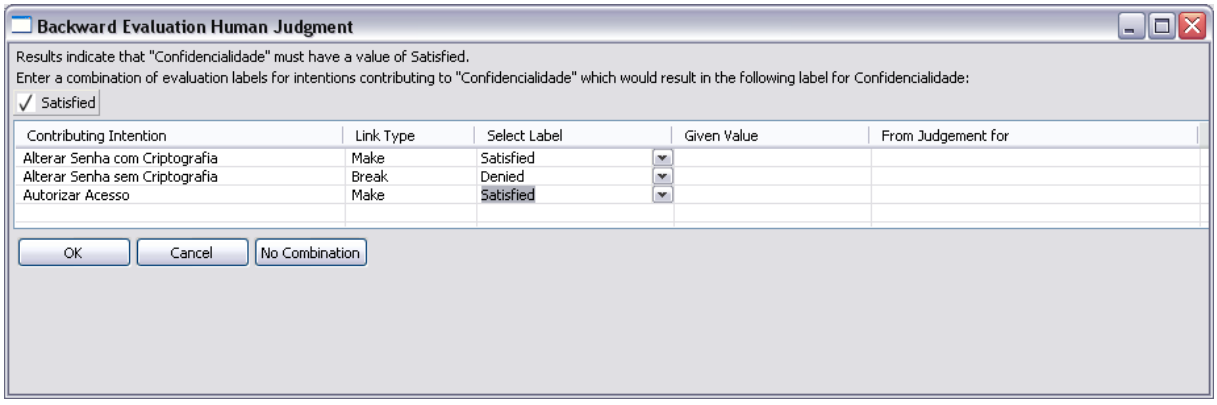


Figura 77. Tela decisão análise NFR 1 - passo 6

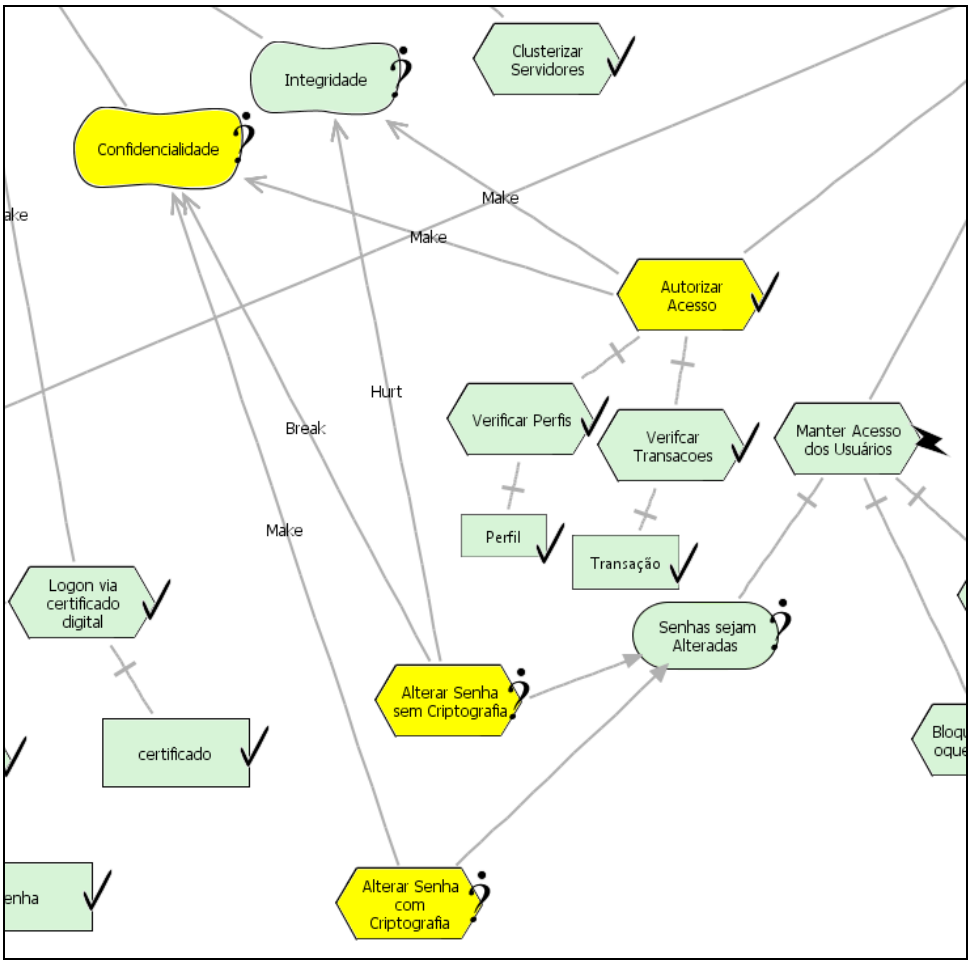


Figura 78. Diagrama parcial análise NFR 1 - passo 6

Neste diagrama alguns elementos ainda estão indefinidos e conflitantes, pois o processo de resolução ainda está nas suas iterações parciais e não foi finalizado.

Passo 7: Resolução de Integridade

Nas figuras 79 e 80 a seguir são atribuídos os valores para resolução e priorização do softgoal **Integridade** e exibido sub-recorte da configuração parcial do modelo nesta etapa de resolução.

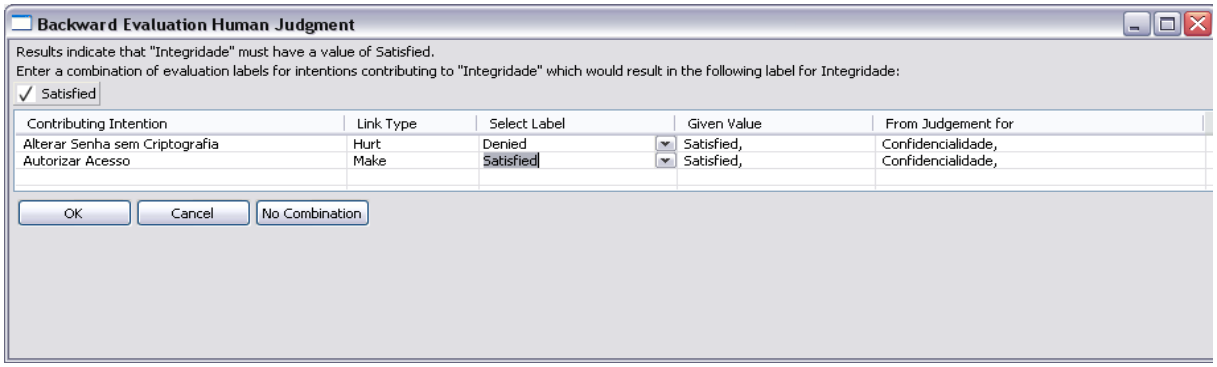


Figura 79. Tela decisão análise NFR 1 - passo 7

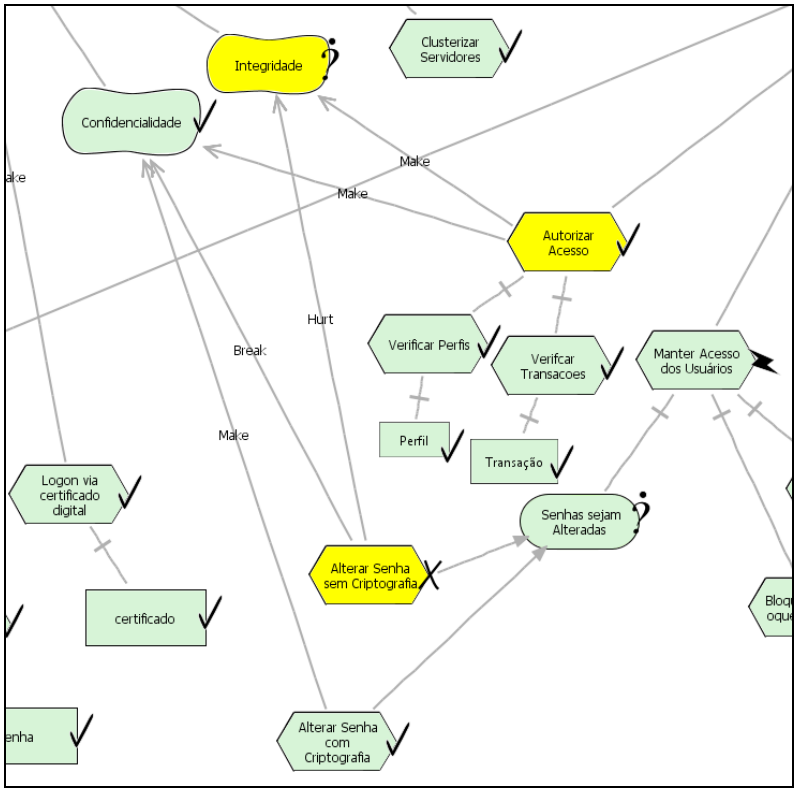


Figura 80. Diagrama parcial análise NFR 1 - passo 7

Nesta última iteração do Sat-Solver o processo de análise NFR é concluído, exibindo-se mensagem de sucesso (figura 81) a seguir e diagrama i* final obtido.

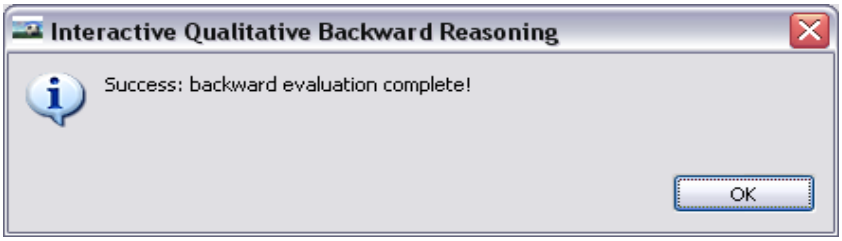


Figura 81. Tela final análise NFR 1 - passo 7

Após esta última atividade do pré-processamento OOM-NFR, o diagrama final gerado (figura 82) é refatorado para remover os elementos negados (tarefas) e ser submetido ao **Processo Núcleo** que, de fato, realiza as transformações de i* para OO-Method.

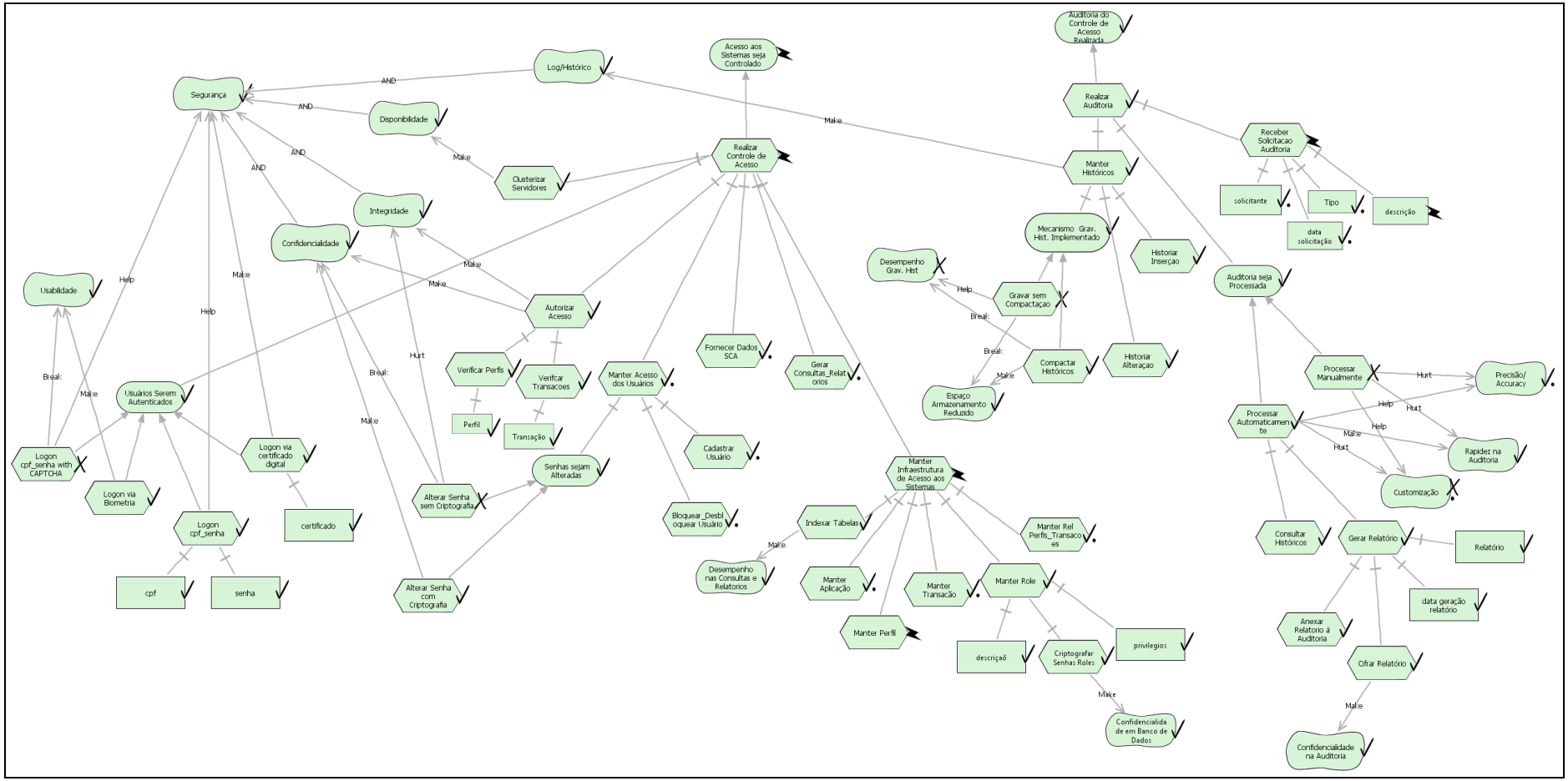


Figura 82. Diagrama i* final da Análise NFR 1 - Backward

Mesmo tendo ainda elementos com rótulo "**Z"** neste diagrama, o processo de resolução dá sucesso, pois os softgoals da configuração inicial escolhidos pelo analista como prioritários já foram satisfeitos.

Para melhor visualização e não sobrecarregar este trabalho com várias telas ou diagramas desnecessários, apenas os diagramas com a configuração inicial e com a configuração final da resolução serão exibidos o recorte total analisado. Os demais diagramas dos passos intermediários serão mostrados sub-recortes desse recorte total, destacando apenas os elementos que estão em foco no momento da análise.

5.4.2 Análise NFR 2

Nesta segunda Análise NFR do Pré-Processamento serão priorizados apenas seguintes softgoals do SCA: **Confidencialidade na Auditoria, Customização e Desempenho Gravação de Históricos**. Os demais softgoals não são prioritários e, para demonstração de exclusão de um maior número de tarefas no modelo final gerado, as tarefas são rotuladas e analisadas para não os satisfazerem e não conflitarem com os prioritários. Assim, os softgoals Usabilidade, Segurança, Espaço Armazenamento Reduzido, Rapidez na Auditoria, Confidencialidade Banco de Dados e Desempenho nas Consultas e Relatórios; e Pecisão (Accuracy) não são prioritários. A análise NFR foi escolhida a análise para frente "**Forward**". Essa análise é "bottom-up", ou seja, parte-se dos elementos folhas para o(s) elemento(s) raiz(es). Essa abordagem tem como desvantagem ser mais trabalhosa (ver seção 2.4.1), pois o analista, caso o SAT-Sover não obtenha sucesso no processo de resolução para uma configuração inicial setada, ele deve entrar manualmente com outras configurações diferentes para submeter novamente ao resolvedor. Entretanto, a análise "forward" pode ser mais eficiente, caso o analista já tenha um conhecimento dos requisitos não-funcionais a serem priorizados e conheça também as regras de propagração e semântica do i*.

Para facilitar a compreensão e referência posterior, esta segunda análise com essa lista de prioridades de requisitos não-funcionais definidas e tipo de análise, será chamada também de **configuração 2.** Esta configuração 2 tenta quebrar ou não satisfazer a maior parte dos requisitos não-funcionais relevantes encontrados no modelo i* SCA e que o analista decidiu não priorizá-los para fins deste segundo experimento realizado. A seguir, são mostrados os passos da resolução.

Passo1: Seleção dos elementos iniciais a serem resolvidos

Na figura 83 a seguir, o analista seleciona inicialmente as tarefas do modelo do sistema SCA em i*, colocando os devidos rótulos de satisfação total (escolha) e não satisfação total (não-escolha ou negação) pra ser submetido ao processo "forward" de resolução. Como já foi visto, no capítulo 4, esse processo se propaga das folhas para as raízes do diagrama.

Como se observa, nessa configuração inicial, estão negadas várias tarefas i*:Logon via Biometria, Logon via certificado digital, Clusterizar Servidores, Verificar Perfis, Verifcar Transacões, Indexar Tabelas, Criptografar Senhas Roles, Processar Automaticamente, Compactar Históricos,etc.; e selecionadas (satisfeitas) outras como: Logon cpf_senha with CAPTCHA, Alterar Senha sem Criptografia, Gravar sem Compactaçao e Processar Manualmente. Com essa configuração definida, o analista vai executar o processo de resolução para verificar se realmente vai atender aos seus requisitos não-funcionais com as devidas prioridades pré-estabelecidas.

Passo 2: Analista verifica conflito para satisfazer segurança

Logo após o analista começar a executar o processo de análise NFR, é exibido uma tela (figura 84) e diagrama parcial obtido (figura 85), mostrando que não conseguiu resolver o softgoal **segurança (realçado em vermelho)** e solicita seja atribuído algum valor coerente. O analista seleciona, como de fato é, um rótulo de conflito.

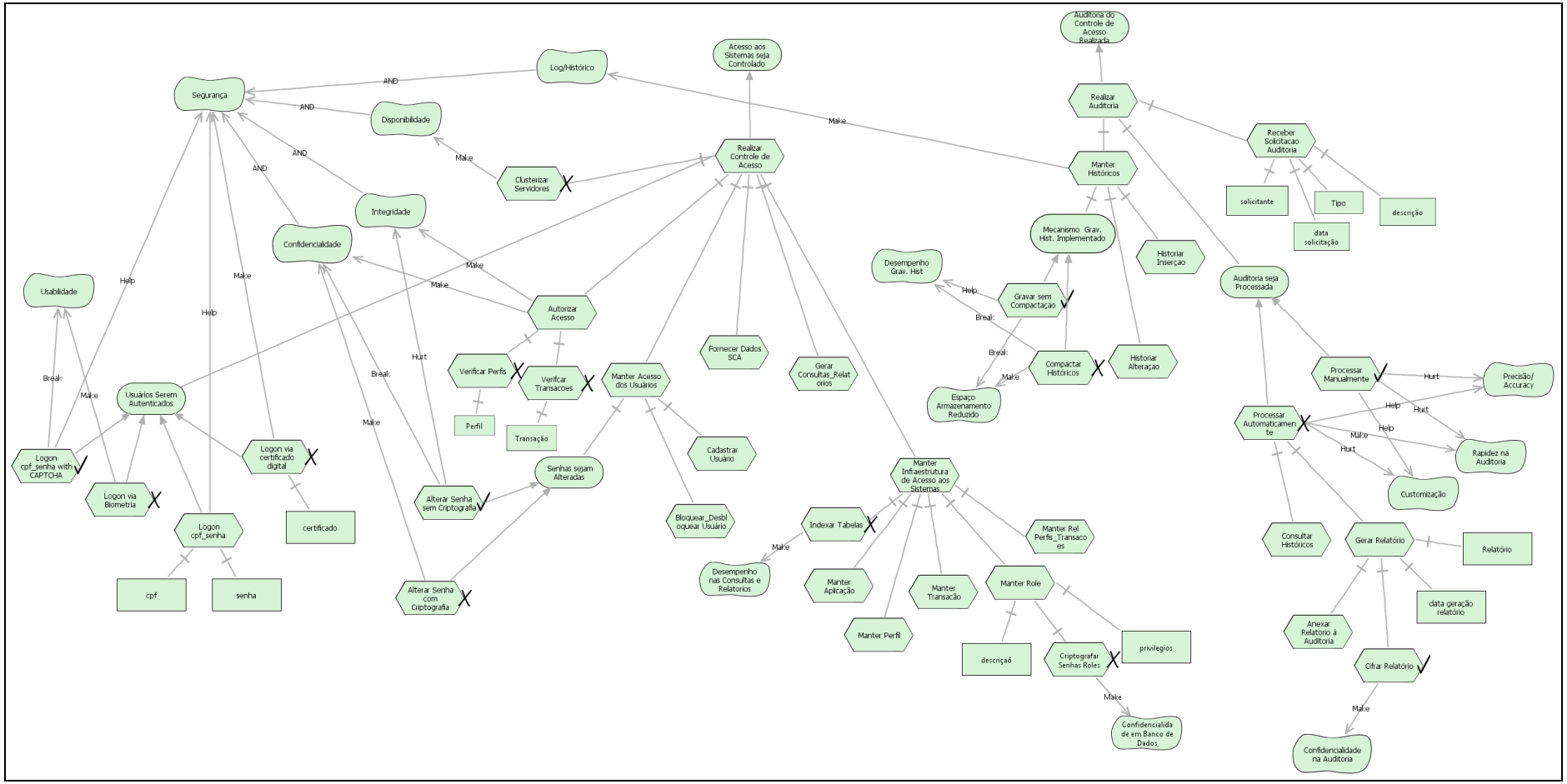


Figura 83. Elementos iniciais da Análise NFR 1 - Forward

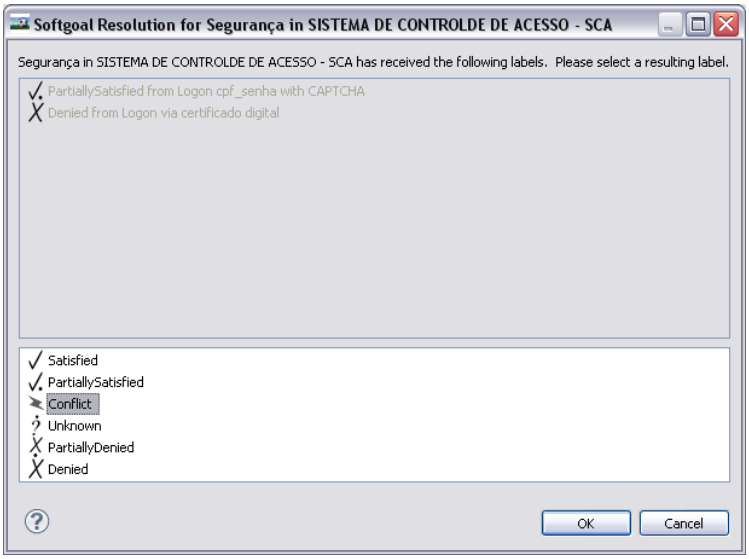


Figura 84. Tela decisão análise NFR 2 - passo 2

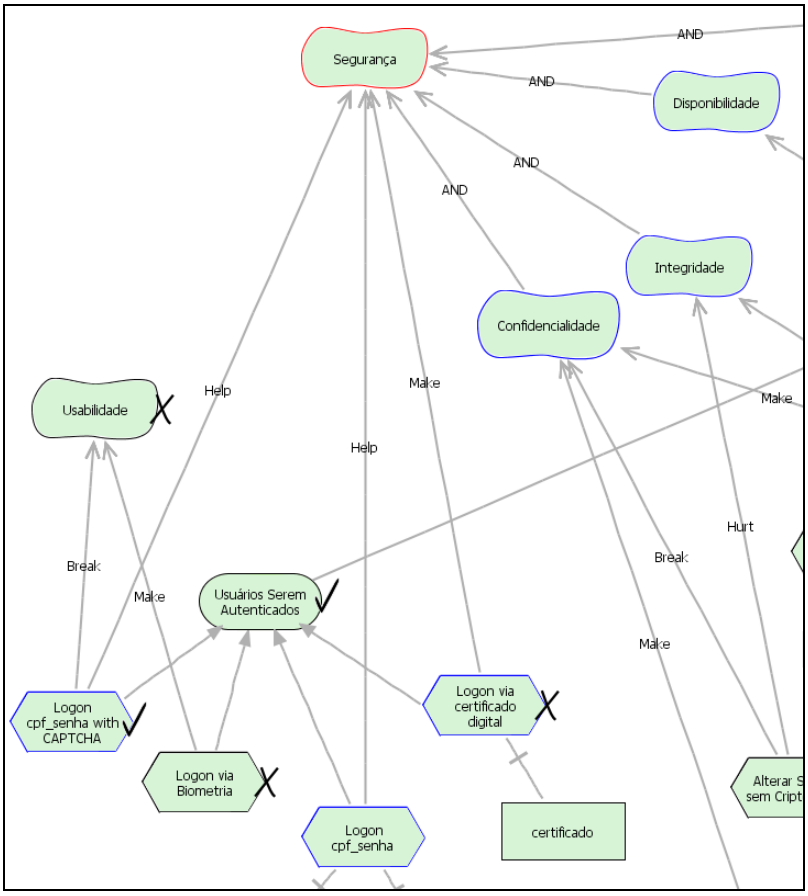


Figura 85. Diagrama parcial análise NFR 2 - passo 2

Passo 3: Analista prioriza Desempenho na gravação de históricos

Nas figuras 86 e 87 a seguir são atribuídos os valores para resolução e priorização do RNF **Desempenho na gravação de históricos** e exibido sub-recorte da configuração parcial do modelo nesta etapa de resolução.

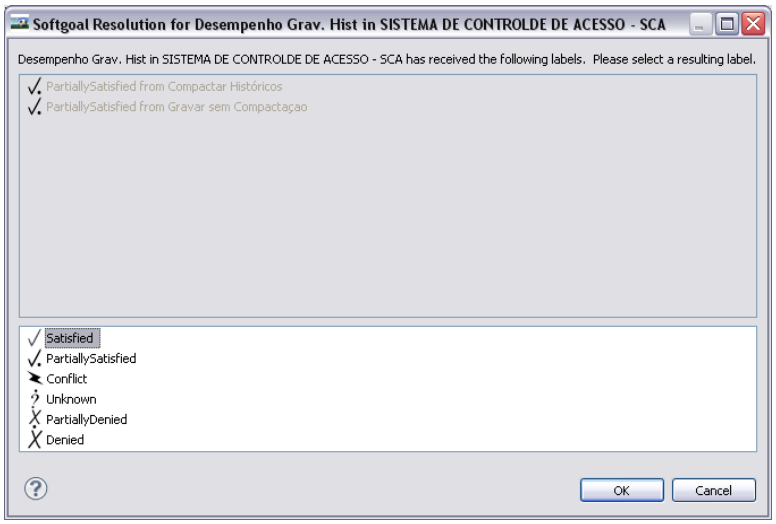


Figura 86. Tela decisão análise NFR 2 - passo 3

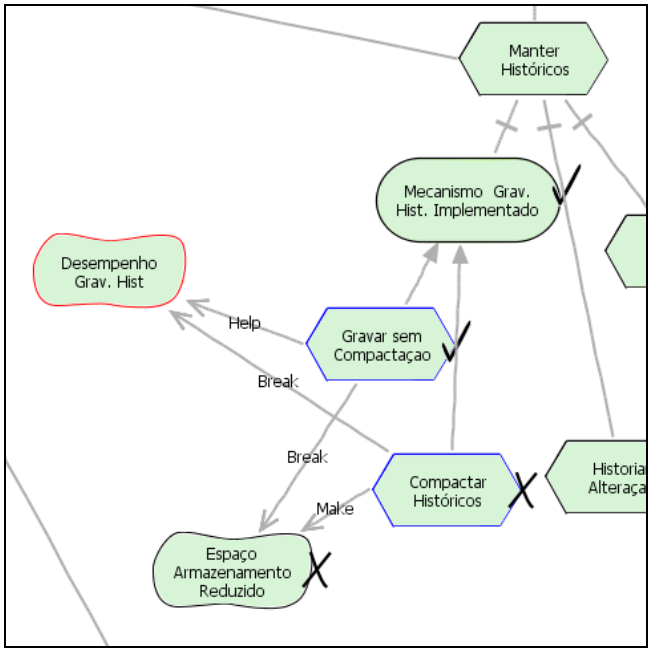


Figura 87. Diagrama parcial análise NFR 2 - passo 3

Passo 4: Analista prioriza Customização na Auditoria

Nas figuras 88 e 89 a seguir são atribuídos os valores para resolução e priorização do RNF **Customização na Auditoria** e exibido sub-recorte da configuração parcial do modelo nesta etapa de resolução.

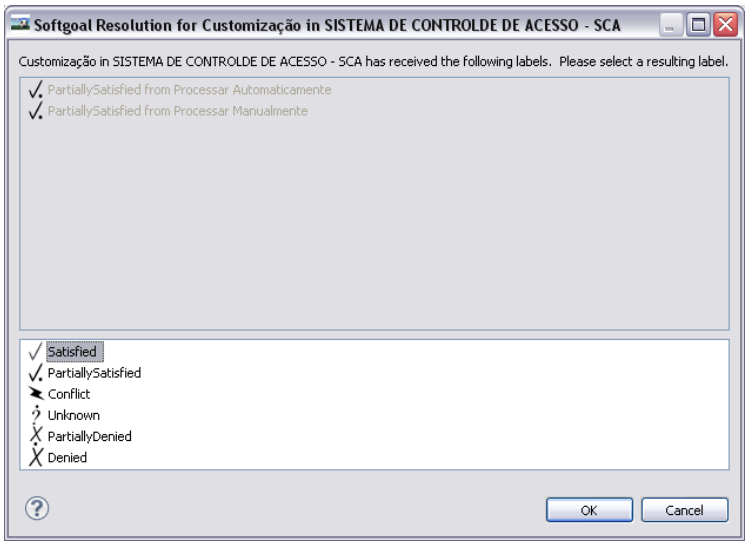


Figura 88. Tela decisão análise NFR 2 - passo 4

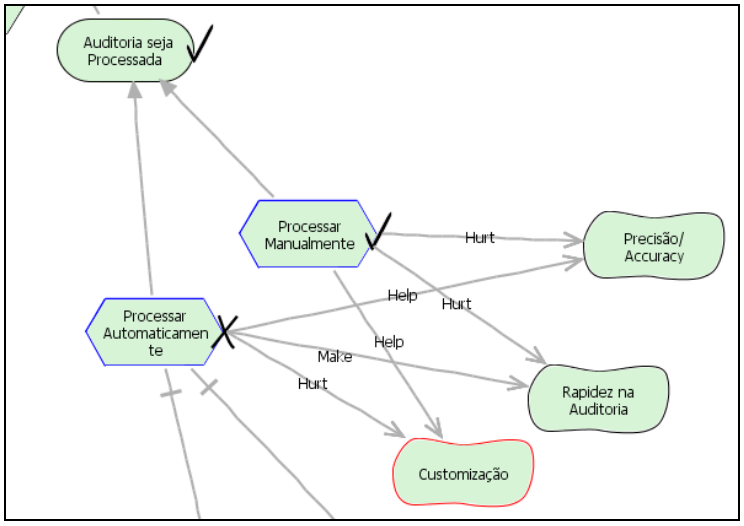


Figura 89. Diagrama parcial análise NFR 2 - passo 4

Passo 5: Analista quebra (nega) Precisão na Auditoria

Nas figuras 90 e 91 a seguir são atribuídos os valores para resolução e priorização do softgoal **Precisão na Auditoria** e exibido sub-recorte da configuração parcial do modelo nesta etapa de resolução.

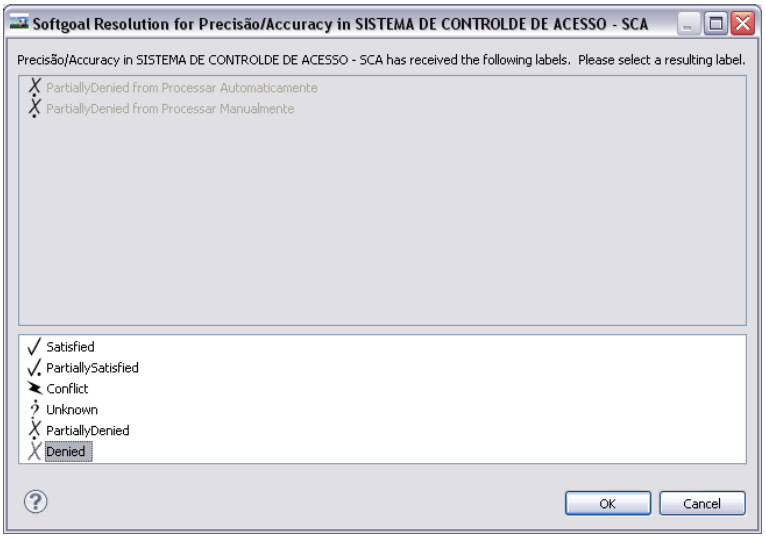


Figura 90. Tela decisão análise NFR 2 - passo 5

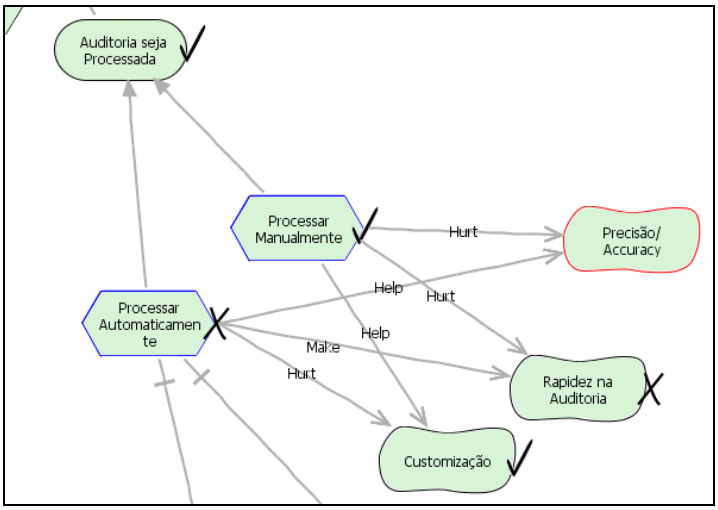


Figura 91. Diagrama parcial análise NFR 2 - passo 5

Passo 6: Analista decide novamente não priorizar segurança

Nesta última iteração, ilustrada nas figuras 92 e 93 a seguir são atribuídos os valores para resolução e priorização do softgoal **Segurança**, exibindo-se o sub-recorte da configuração parcial do modelo nesta etapa de resolução.

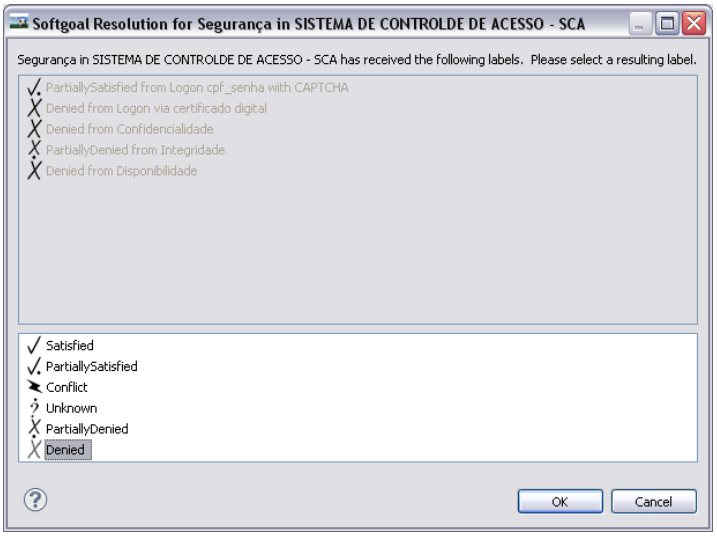


Figura 92. Tela decisão análise NFR 2 - passo 6

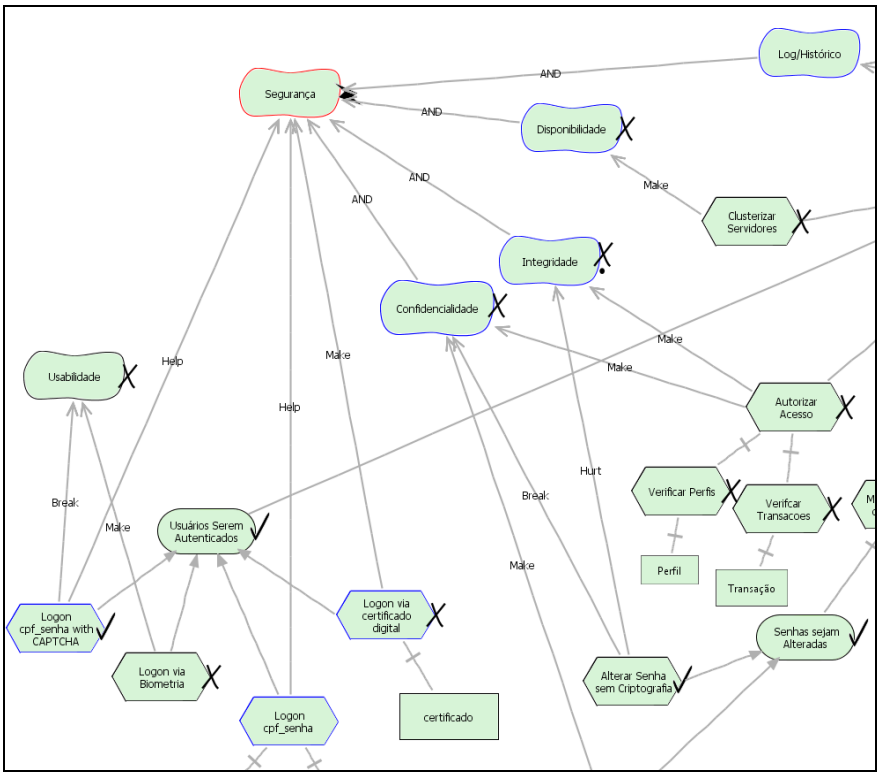


Figura 93. Diagrama parcial análise NFR 2 - passo 6

Após esse passo, o processo exibe mensagem de sucesso, na resolução “Forward”, seguido do modelo final resultado da análise, conforme figura 94 a seguir. A figura mostra que todos os softgoals que deveriam ser priorizados e escolhidos inicialmente pelo analista de requisitos foram satisfeitos. Após essa etapa de pré-processamento, segue-se a refatoração desse diagrama final gerado a fim de ser submetido ao processo núcleo transformativo.

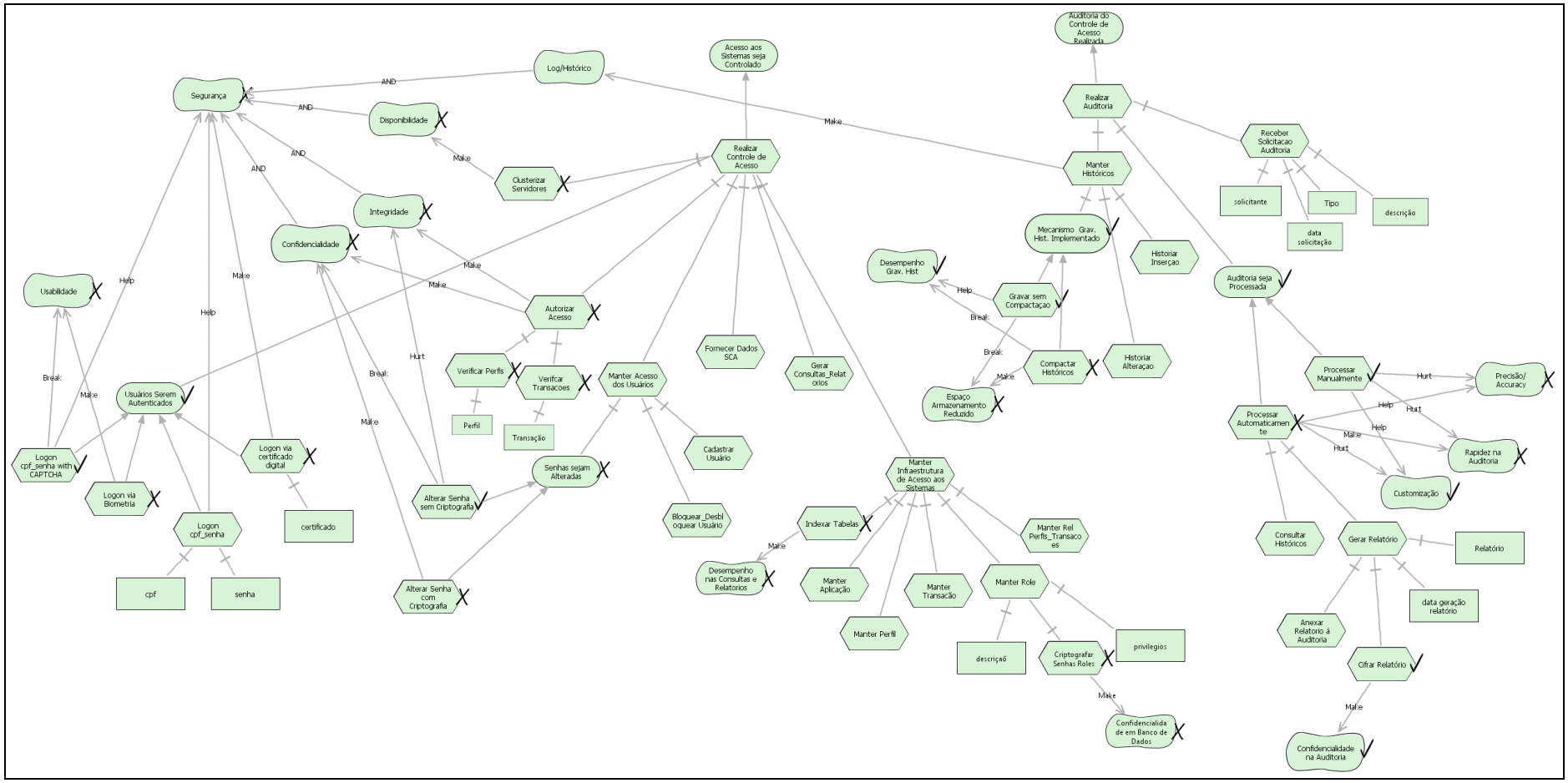


Figura 94. Diagrama i* final da Análise NFR 2 - Forward

5.5 Processando as Regras de Transformação

Como já visto na seção 4.3, o subprocesso responsável por realizar o processamento das regras de transformações é o **Processo Núcleo.** Esse processo recebe como entrada um modelo i* refatorado com os conflitos resolvidos e softgoals (requisitos não-funcionais relacionados a produto e interno ao ator sistema) priorizados a partir da análise NFR realizada no **Pré-Processamento. O processo núcleo** é guiado por diretrizes que definem o mapeamento dos elementos i* para o **modelo objeto** do OO-Method. Cabe-se ressaltar que o modelo objeto é apenas um dos modelos do esquema conceitual do OO-Method (ver seção 3.2.1.3) e que o processo OOM-NFR tem como escopo apenas esse modelo objeto.

O Sistema de Controle de Acesso, modelado em i* wiki, será submetido ao processo transformativo núcleo do OOM-NFR que aplicará as diretrizes de geração de classes, atributos, serviços e relacionamentos especificados nas seções a seguir. Cabe ressaltar que essas diretrizes são automatizadas e implentadas na linguagem de transformação de modelos QVT, conforme exposto no apêndice deste trabalho.

5.5.1 Geração das Classes

Para derivar do modelo i* SCA as respectivas classes em OO-M são aplicadas as diretrizes 1.1 e 1.2 vistas no capítulo 4. A seguir, as definições dessas diretrizes e as classes mapeadas a partir delas:

Diretriz 1.1: Um ator * i é transformado em uma classe do modelo de classe destino.O nome da classe gerada é obtida a partir do nome do ator.

- **Usuário**
- **Administrador**
- **Auditor**
- **Sistema de Controle de Acesso (SCA)**
- **Sistema Externo**

Diretriz 1.2: Um recurso i* que representa uma entidade física é transformada em uma classe do modelo de classe destino. O nome da classe gerada é obtidos a partir do nome do recurso.

Como visto no diagrama SCA da seção 5.3, os seguintes recursos físicos, e portanto, transformados em classes são:

- **Perfil**

- **Transação**
- **Role**
- **Aplicação**
- **Auditoria**

5.5.2 Geração de Atributos

Para cada classe obtida das duas primeiras diretrizes anteriores, é necessário identificar os elemento i* que podem ser usados para inferir seus atributos. Para fazer isso, os recursos que representam uma entidade informativa serão o nosso principal alvo, porque eles representam propriedades de recursos físicos ou atores. Assim, esses recursos são transformados em atributos das classes geradas anteriormente, considerando a partir do qual o elemento (ator ou entidade física) a entidade informacional está relacionada. Para a geração de atributo desses atributos as seguintes as diretrizes de transformação são definidas.

A seguir apresentamos as respectivas diretrizes de geração dos atributos em OO-Method e os respectivos atributos gerados.

Diretriz 2.1: Se um recurso i* representa uma entidade de informação relacionada a um ator do modelo i *, então esse recurso informacional é transformado em um atributo da classe gerada a partir desse ator a que se refere. O nome do atributo é obtido a partir do nome do recurso informacional. Os recursos do modelo que são afetados por essa regra de transformação são:

- Relacionados à classe **Usuário**
    - **cpf**
    - **nome**
    - **senha**
    - **certificado**
    - **biometria**
    - **rh**
    - **tipo**
    - **CNPJ_empresa**

- **Administrador (não há atributos gerados)**
- **Auditor (não há atributos gerados)**

- **Sistema de Controle de Acesso (SCA) (não há atributos gerados)**
- **Sistema Externo (não há atributos gerados)**

Nota-se que para as classes Administrador, Auditor, SCA e Sistema Externo não foram gerados atributos, pois no modelo i* não havia recursos informacionais relacionados aos seus respectivos atores.

Diretriz 2.2: Se um recurso i* representa uma entidade informacional relacionado com um recurso físico então esse recurso informacional é transformado em um atributo da classe gerada do recurso físico. O nome do atributo é obtido a partir do nome do recurso informacional. Assim, os recursos do modelo que são afetados por esta diretriz de transformação são os seguintes:

- **Perfil**
  - **perfil_id**
  - **descricao**
  - **vigência**
- **Transação**
  - **Transação_id**
  - **tipo**
  - **vigência**
  - **role_id**
  - **aplicação_id**
- **Role**
  - **role_id**
  - **descrição**
  - **privilegios**
- **Aplicação**
  - **Aplicação_id**
  - **descrição**
- **Auditoria**
  - **solicitante**
  - **Data solicitacao**
  - **relatório**
  - **Tipo**

- **Data geração relatório**

5.5.3 Geração de Serviços

Nesta fase do processo transformativo já se tem geradas as classes e seus atributos, faltando ainda gerar seus os serviços que, segundo, o metamodelo OO-M, são agregados das classes. Essa geração é obtida partir das diretrizes a seguir:

Diretriz 3.1: Uma tarefa que só afeta os valores de recursos informacionais relacionada a um ator se transforma em uma transação da classe gerada a partir de o ator correspondente. O nome da transação gerada é inferido a partir do nome da tarefa. Os recursos relacionados são definidos como argumentos da transação. Se a tarefa também é decomposta em subtarefas, então as subtarefas decompostas são incluídas na fórmula da transação gerada.

- **Usuário**
  - **Alocar Perfil ao Usuário (CPF,perfi_id)**
  - **Bloquear_Desbloquear Usuário**
  - **Cadastrar Usuário**
  - **Gerar Senha**
  - **Alterar Senha com Criptografia**
  - **Alterar Senha sem Criptografia**
- **Administrador (não há transação gerada)**
- **Auditor (não há transação gerada)**
- **Sistema de Controle de Acesso (SCA)**
  - **Logon cpf_senha with CAPTCHA**
  - **Logon cpf_senha**
  - **Logon via certificado digital**
  - **Logon via Biometria**
  - **Verificar Perfis - 1**
  - **Verifcar Transacoes - 2**
  - **Autorizar Acesso - 3 (transação composta por 1 e 2)**
  - **Manter Acesso dos Usuarios - 4**
  - **Manter Infraestrutura de Acesso aos Sistemas - 6**
  - **Realizar Controle de Acesso(transação composta por 3, 4 , 5 e 6)**

- **Compactar Históricos - 6**
- **Historiar Alteraçao - 7**
- **Historiar Inserçao - 8**
- **Manter Históricos - 9**
- **Receber Solicitacao Auditoria - 10**
- **Realizar Auditoria (transação composta por 9 e 10)**
- **Cifrar Relatório - 11**
- **Gerar Relatório - 12 (transação composta por 11)**
- **Consultar Históricos - 13**
- **Processar Automaticamente - 14 (transação composta por 12 e 13)**
- **Processar Manualmente**
- **Fornecer Dados SCA**
- **inicializar, finalizar Sistema – transações geradas automaticamente associadas eventos default.**

- **Sistema Externo**
  - **Validar e Recuperar Dados Externos**

Diretriz 3.2: Uma tarefa que só afeta os recursos informativos relacionados a um recurso físico é transformada em uma transação do recurso físico correspondente. O nome da transação gerada é inferido a partir do nome da tarefa. Os recursos relacionados são definidos como argumentos da transação. Se a tarefa é também decomposta em subtarefas, então as subtarefas decompostas estão incluídas na fórmula da transação gerada.

- **Auditoria**
  - **Setar Prazo Entrega**
  - **Anexar Relatorio à Auditoria**

No modelo SCA poderia ter várias outras transações geradas a partir dessa diretriz, caso fossem modeladas novas tarefas que estivessem relacionadas com a produção ou consumo de recursos informativos, ou seja, se para cada recurso informativo modelado, houvesse um tarefa, por exemplo, para atribuir um valor para tal recurso e também para consumir (consultar) esse mesmo recurso informativo. Se isso fosse feito, por exemplo, para um recurso físico com três recursos informativos relacionados, haveria seis tarefas (três para setar esses recursos e mais três para ler esses mesmos recursos informativos). Se fosse seguida essa lógica similar a orientação a objeto que padroniza a definição de uma classe, atributos e

seus métodos dessa forma, o modelo i* ficaria enorme, dificultando sua visualização e demonstração.

Diretriz 3.3: Uma tarefa que só afeta um recurso físico é transformada em um serviço do recurso físico correspondente. O nome da transação gerou é inferida a partir do nome da tarefa. Se a tarefa também é decomposta em subtarefas, em seguida, as subtarefas decompostas são incluídas na fórmula da transação gerada.

- **Perfil**
  - **Manter Perfil**
- **Role**
  - **Manter Role**
  - **Criptografar Senhas Roles**
- **Aplicação**
  - **Manter Aplicação**
- **Transação**
  - **Manter Transação (transacao id, role id, aplicacao id)**

Diretriz 3.4: Uma tarefa que afeta recursos físicos diferentes (distintos) ou afeta recursos informativos relacionados a diferentes recursos físicos é transformada em uma transação da classe gerada a partir do ator que contém tal tarefa (de acordo com a fronteira ator correspondente). O nome do gerado transação é inferida a partir do nome da tarefa. Os recursos relacionados são definidos como argumentos da transação. Se a tarefa também é decomposta em subtarefas, então as subtarefas decompostas são incluídas na fórmula da transação gerada.

- **Manter Relacionamento Perfis_Transacoes (perfil id, transaçao id)**

Diretriz 3.5: Uma tarefa que não afeta recursos é transformada em uma transação da classe gerada a partir do ator que contém a tarefa (de acordo com a fronteira do ator correspondente). Se a tarefa também é decomposta em subtarefas, então as subtarefas decompostas são incluídas na fórmula da transação gerada. O nome da transação gerada é inferido a partir do nome da tarefa.

- **Clusterizar Servidores**
- **Indexar Tabelas**

- **Gerar Consultas_Relatórios**

Diretriz 3.6: As tarefas que participarem de uma dependência de recurso, onde o recurso dependum corresponde a um recurso físico, são transformadas em transações da classe gerada a partir do recurso dependum. Os nomes das transações geradas são inferidos a partir dos nomes dos correspondentes tarefas. Esta diretriz é complementar a diretriz 3.3. Não é utilizada quando a orientação é 3.4 é aplicada às tarefas envolvidas.

Esta diretriz gera as tarefas que estão dentro dos atores externos ao ator sistema (SCA) e têm depedum (recursos físicos) associados à outras tarefas dentro do ator sistema.No caso do SCA, as transações desses atores externos já foram contempladas nas diretrizes 3.3 e 3.4.

Diretriz 3.7: Uma tarefa que está envolvida na geração de um recurso físico é transformada em uma transação de criação da classe gerada a partir do recurso físico correspondente. O nome da transação gerada é inferido a partir do nome da tarefa envolvida. Esta diretriz é complementar à diretriz 3.3. Não é aplicável quando a diretriz 3.4 é aplicada sobre a tarefa em questão.

A única transação gerada por essa diretriz é **Create Role** da Classe **Role.**

5.5.4 Geração dos Relacionamentos

Até esta etapa, já estão geradas as classes, seus atributos e serviços. Entretanto, as classes estão soltas no modelo, ou seja, estão desconectadas umas às outras, pois estão faltando os relacionamentos entre elas. De acordo com o metamodelo OO-Method, especificado para operar com o processo OOM-NFR proposto, há apenas dois tipos básicos de relacionamentos definidos: associação e “agent link” já vistos no capítulo 3, seção 3.2.2. Sendo assim, as diretrizes 4.1 desta seção não gerarão relacionamentos de herança em OO-Method. Entretanto, na implementação dessa diretriz utilizando a linguagem de transformação de modelos QVT, conforme está listado no código fonte e comentários expostos no apêndice deste trabalho, ela é transformada em relacionamentos de asssociação OO-Method.

Diretriz 4.1: Se houver um relacionamento de generalização (relação é-um “IS-A”) entre dois atores do modelo i* que foram transformados em classes, em seguida, um

relacionamento de generalização é definido entre essas classes correspondentes no modelo de classe.

Caso tivessem sido modelados relacionamentos IS-A entre atores no sistema SCA, os atores Administrador e Auditor poderiam estar associados com um relacionamento de herança para com a classse Usuário gerada. Entretanto, relacionamentos IS-A não foram modelados, por terem ainda uma semântica imprecisa na abordagem i* wiki. Também, não está modelado no metamodelo OO-Method o relacionamento de herança. Para não deixar essa diretriz sem automatização, ela foi implementada em QVT (ver apêndice) como relacionamento de associação OO-Method.

Diretriz 4.2: Se um recurso físico do modelo i* é derivado de outro recurso físico, então um relacionamento de generalização é definido do recurso derivada para o recurso original. Esta derivação não é definida explicitamente no modelo i*, portanto é necessária a intervenção do analista de requisitos para indicar quais são os recursos físicos abrangidos por essa condição.

No modelo i* SCA não foram identificados elementos que pudessem gerar relacionamentos através dessa diretriz. Na implementação dessa diretriz 4.2 em QVT (ver apêndice) só é implementada a assinatura da função e seu retorno é nulo, demostrando que essa diretriz não é factível de automatização.

Diretriz 4.3: Para as duas classes geradas a partir dos atores i*, se houver qualquer ligação de dependência entre os dois atores transformados, então uma associação entre as classes correspondentes é gerada automaticamente no diagrama de classes OO-Method.

As associações geradas:

- **Usuário e SCA**
- **Administrador e SCA**
- **Auditor e SCA**
- **Sistema Externo e SCA**

Diretriz 4.4: Se existe uma ligação de dependência de recursos onde o dependum, os atores depender e dependee foram transformados em classes, então associações são geradas automaticamente entre estas classes.

Associações geradas:

- **Auditor e Auditoria**
- **Auditoria e SCA**
- **Administrador e Perfil**
- **Perfil e SCA**
- **Administrador e Transação**
- **Transação e SCA**
- **Administrador e Aplicação**
- **Aplicação e SCA**
- **Administrador e Role**
- **Role e SCA**

Diretriz 4.5: Para uma ligação de dependência de recurso onde o dependum é transformado em um atributo de classe (dependum é recurso informacional) e os atores Depender e dependee são transformados em classes, uma associação é gerada entre as classes geradas dos atores e a classe derivada do recurso físico ao qual está associado esse atributo (recurso informacional).

Esta regra, no modelo SCA, aparece na ligação de dependência onde o **dependum Relatório** e atributo da classe Auditoria. A dependência é entre Auditor e o SCA.
Segundo essa diretriz, seriam geradas as associações abaixo e que já foram criadas na diretriz 4.4, mas que poderiam, dependendo do diagrama i* modelado, não estarem criadas, por exemplo, caso não houvesse no modelo, por uma questão de modelagem ou omissão, uma dependência de recurso fisico entre Auditor, Auditoria e SCA; então as associações derivadas dessa dependência não teriam sido geradas na diretriz 4.4, não ocorrendo assim redundância. Na ocorrência de redundância, a automatização do processo OOM-NFR não criaria novamente a associação.

- **Auditor e Auditoria**
- **Auditoria e SCA**

Diretriz 4.6: Para uma classe resultante da transformação de um recurso físico que é definido dentro da fronteira de um ator (recurso interno), uma associação é criado entre esta classe de recurso e a classe resultante da transformação do ator respectivo (aquele que contém o recurso).

No modelo SCA não há recurso físico totalmente interno aos atores, pois os recursos físicos estão fora das fronteiras desses atores. Os que estão visualmente internos estão por uma questão meramente gráfica com o objetivo de reduzir o tamanho e facilitar a visualização do modelo. Por exemplo, todos os recuros físicos do modelo (Perfil, Transaçao, Role, Aplicaçao, Auditoria) deveriam estar visualmente fora das fronteiras dos atores. Há sim, muitos recursos informativos internos, mas que já foram transformados em atributos. Logo, não há associação gerada por essa diretriz.

Diretriz 4.7: Caso uma tarefa interna a um ator foi transformada em transação da classe derivada desse ator (Diretriz 3.4), faz-se a seguinte análise: **Se** essa transação tem argumentos derivados de recursos físicos distintos ou tem argumentos derivados de recursos informativos relacionados a diferentes recursos físicos, **então** é gerada uma associação entre as classes obtidas a partir desses recursos físicos distintos ou entre as classes às quais pertencem os recursos informativos.

Dessa Diretriz são geradas as associações:

- **Usuário e Perfil** a partir da tarefa **Alocar Perfil ao Usuário**(cpf, perfil_id);
- **Perfil e Transação** a partir da tarefa **Manter Relacionamento Perfis_Transacoes** (cod. perfil,cod. transaçao);
- **Transação e Aplicação** a partir da tarefa **Manter Transação**(transacao_id, role_id, aplicacao_id);
- **Transação e Role** a partir da tarefa **Manter Transação**(transacao_id, role_id, aplicacao_id);

Diretriz 4.8: Para qualquer ligação de dependência entre os dois atores depender e dependee transformados em classes, onde exista uma tarefa que fora transformada em serviço de uma classe no lado do Dependee, criar também um **agent link** da classe gerada do Depender para a classe gerada Dependee.

Esta diretriz gera o agents links:

- **Usuário e SCA (**Logon via certificado digital, Logon cpf_senha)
- **Administrador e SCA (**Manter Infraestrutura de Acesso aos Sistemas**)**
- **Auditor e SCA** (**Receber Solicitacão Auditoria)**
- **SCA e Sistema Externo (Validar e Recuperar Dados Externos)**

Diretriz 4.9: Cada tarefa transformada em serviço que estava dentro da fronteira de um ator transformado em classe, caso essa tarefa pertença a outra classe diferente desse ator, criar um relacionamento de Agente (**agent link)** entre esta classe derivada do ator e a classe a qual pertence essa respectiva tarefa.

Esta diretriz gera os seguintes relacionamentos agent link:

- **Administrador e Perfil (Manter Perfil)**
- **Administrador e Aplicação (Manter Aplicação)**
- **Administrador e Transação (Manter Transação)**

5.6 Modelos OO-Method Gerados

Como resultado da aplicação das diretrizes ou regras de transformação especificadas na seção 5.5, os modelos objetos (ver seção 3.2.1.3) do OO-Method são gerados. Para obter esses modelos, o Processo Núcleo OOM-NFR recebeu como entrada os modelos i* refatorados na etapa final de Pré-processamento apresentados nas seções 5.4.1 e 5.4.2.

Nas figuras 95 e 96 são mostrados esses modelos com os respectivos comentários.

A figura 95 representa o modelo objeto do OO-M gerado a partir da configuração 1 (primeira análise NFR) descrita na seção 5.4.1 deste capítulo.

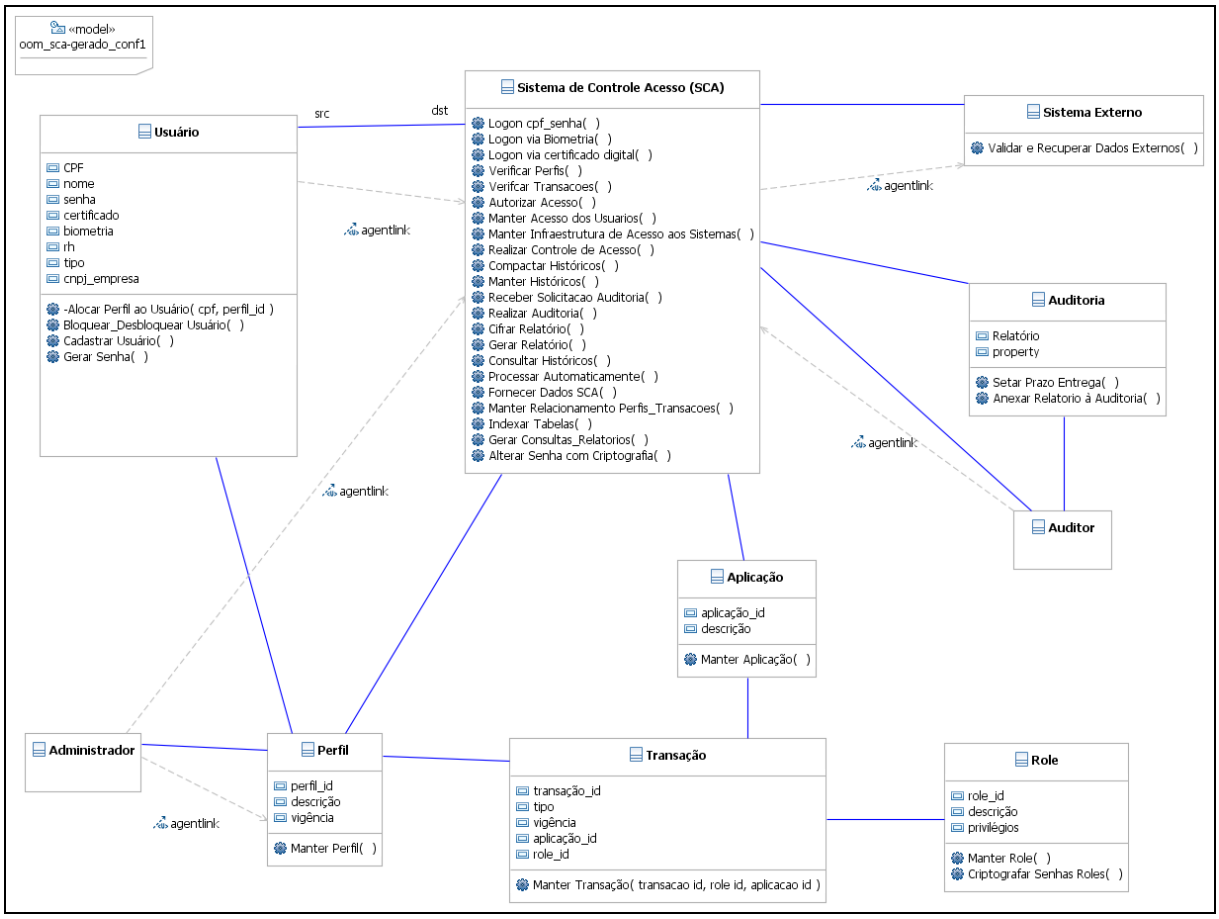


Figura 95. Modelo OO-Method gerado a partir da Configuração 1

Neste modelo gerado, observa-se que os seguintes serviços da classe Sistema de Controle de Acesso (SCA) estão ausentes:

- **Logon cpf_senha with CAPTCHA**
- **Alterar Senha sem Criptografia**
- **Processar Manualmente**
- **Gravar sem Compactação**

Isso ocorreu porque a configuração 1 prioriza os seguintes softgoals: **Usabilidade, Segurança, Espaço Armazenamento Reduzido, Rapidez na Auditoria, Confidencialidade na Auditoria, Confidencialidade Banco de Dados e Desempenho nas Consultas e Relatórios.** Além disso, durante o processo de análise NFR 1 (seção 5.4.1) as tarefas a partir das quais esses serviços derivaram, foram sendo negadas para que esses RNFs de Usabilidade, Segurança, Rapidez, etc fossem satisfeitos e ficassem sem conflitos. De fato, foi a última atividade do Pré-processamento (ver seção 4.2), ou seja, a refatoração do modelo final gerado

pela análise NFR) que removeu essas tarefas negadas e passou esse modelo i* refatorado para o Processo Núcleo realizar as transformações.

Os serviços das demais classes foram mantidos, pois não tiveram impactos nos RNFs priorizados a partir da análise NFR. Os relacionamentos de associações e agent link gerados entre as classes também estão ilustrados.

A figura 96 representa o modelo OO-Method gerado a partir da configuração 2 (segunda análise NFR) descrita na seção 5.4.2 deste capítulo

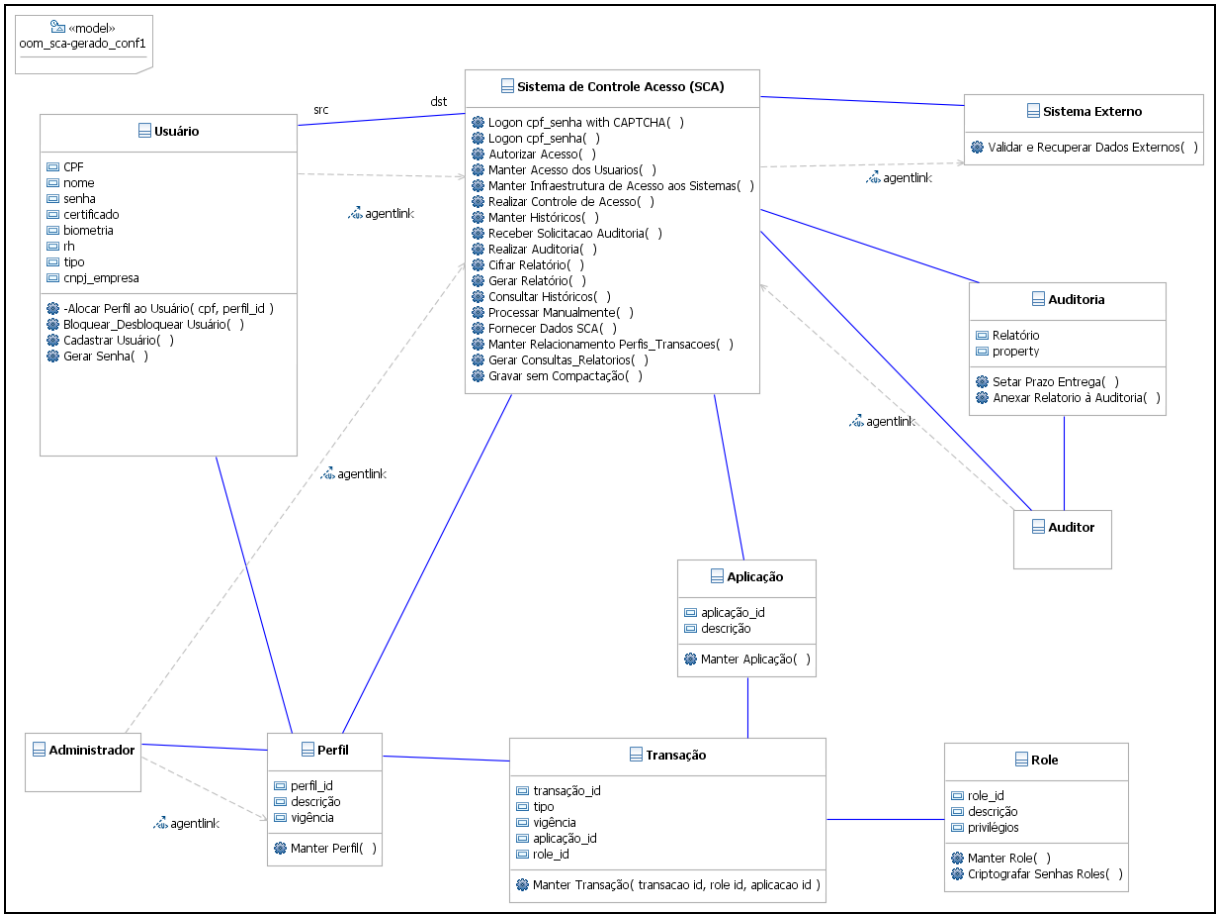


Figura 96. Modelo OO-Method gerado a partir da Configuração 2

Neste modelo gerado, observa-se que os seguintes serviços da classe Sistema de Controle de Acesso (SCA) estão ausentes:

- **Logon via Biometria**
- **Logon via certificado digital**
- **Clusterizar Servidores**
- **Verificar Perfis**
- **Verifcar Transacoes**

- **Indexar Tabelas**
- **Criptografar Senhas Roles**
- **Processar Automaticamente**
- **Compactar Históricos**

Isso ocorreu porque, nesta segunda Análise NFR, foram priorizados apenas os seguintes softgoals do SCA: **Confidencialidade na Auditoria, Customização e Desempenho Gravação de Históricos.** Os demais softgoals (Usabilidade, Segurança, Espaço Armazenamento Reduzido, Rapidez na Auditoria, Confidencialidade Banco de Dados e Desempenho nas Consultas e Relatórios; e Pecisão/Accuracy) não são prioritários e, ao longo do processo de resolução, foram sendo negados totalmente (não satisfeitos) juntamente com suas tarefas relacionadas. Esse grande número de tarefas está ausente no modelo OO-Method gerado, pois elas não contribuem para satisfazer, priorizar e deixar sem conflitos os três softgoals considerados prioritários (**Confidencialidade na Auditoria, Customização e Desempenho Gravação de Históricos**).

Por fim, os relacionamentos de associações e agent link gerados entre as classes também estão ilustrados na figura 96.

5.7 Considerações Finais

Neste capítulo foi realizado um exemplo completo de aplicação do processo de transformação OOM-NFR a um sistema de informação real chamado Sistema de Controle de Acesso (SCA). O Sistema de Controle de Acesso foi contextualizado e os principais objetivos a serem atingidos com o experimento foram descritos.

Na seção 5.3 foi mostrado e detalhado o SCA modelado em i* wiki que fora utilizado como entrada do processo transformativo. Na seção 5.4 foi descrito o pré-processamento responsável por realizar a análise dos softgoals. Para demonstrar esse pré-processamento foram dados dois exemplos de análise NFR baseados no modelo i* SCA. Cada exemplo analisado tinha prioridades distintas em termos dos softgoals, sendo geradas configurações finais também diferentes para atender a cada conjunto de prioridades definida pelo analista.

Após o pré-processamento, na seção 5.5 foi exemplificado o processo núcleo que tem como principal atividade transformar o modelo i* para o modelo objeto do OO-Method através da aplicação das diretrizes de transformação. O processo núcleo transformativo do

OOM-NFR é dividido basicamente em quatro atividades: geração de classes, geração de atributos, geração de serviços e geração de relacionamentos.

Por fim, na seção 5.6 são exibidos os modelos objetos do OO-Method gerados para cada diagrama final obtido a partir de cada uma das análises NFR realizadas.

A partir do objetivo maior que fora transformar o modelo i* de requisitos do Sistema de Controle de Acesso num modelo de mais baixo nível (OO-Method) alguns outros objetivos específicos foram alcançados, tais como:

- Análise, Priorização e tratamento de conflitos dos softgoals;
- Exploração da capacidade de análise estratégica do SCA por estar sendo remodelado na linguagem organizacional i* wiki que dá suporte a responder "o quê", "o como" e " o porquê" dos elementos modelados;
- Identificação de vulnerabilidades na satisfação dos softgoals de segurança que são críticos para o SCA;
- Otimização e reengenharia dos processos envolvidos no SCA para melhor satisfação das suas metas e seus RNFs;

No próximo capítulo, serão apresentadas as conclusões gerais desse trabalho, as considerações e limitações gerais, contribuições e trabalhos futuros.

CAPÍTULO 6

6. Conclusões e Trabalhos Futuros

Neste capítulo, resumem-se os principais resultados, contribuições, limitações, lições aprendidas através desta pesquisa. Alguns trabalhos futuros são propostos.

No presente contexto atual da Engenharia de Software, onde o desenvolvimento dirigido a modelos é uma realidade em vários domínios de aplicação, este trabalho propõe transformações de modelos de requisitos em i* para modelos no nível de projeto em OO-Method com foco em requisitos não-funcionais. Parte-se do princípio que modelos de requisitos são fundamentais para se obter software de qualidade. Entretanto, a maior parte dos modelos de requisitos existentes documenta apenas os requisitos funcionais, negligenciando os requisitos não-funcionais.

Várias abordagens, mencionadas na seção 3.3, realizam transformações de modelos em i* para OO-Method, mas nenhuma delas considera os softgoals, no sentido de realizar uma análise prévia destes a fim de que o modelo OO-M gerado reflita as prioridades desses softgoals e que eles estejam também sem conflitos.

A partir dessas motivações e visando resolver esses problemas, foi proposto um processo transformativo de i* para OO-Method, chamado OOM-NFR, com base em softgoals, considerando suas possíveis prioridades e conflitos existentes. O processo OOM-NFR foca apenas softgoals do modelo i* internos ao ator sistema, uma vez que esses softgoals podem ser considerados requisitos não-funcionais de produto (SOMMERVILLE, 1998) e passíveis de operacionalização (ver seção 2.2).

O restante deste capítulo está organizado com se segue: Seção 6.1 indica as principais contribuições obtidas com esta dissertação. A Seção 6.2 relata as limitações e restrições realizadas. Enquanto a Seção 6.3 expõe as lições aprendidas e as limitações para conduzir a pesquisa. Para finalizar, a Seção 6.4 anuncia o que pode ser feito como trabalhos futuros.

6.1 Contribuições

Em termos gerais, as contribuições obtidas a partir desta dissertação estão diretamente relacionadas com os objetivos alcançados e expostos na seção 1.3, ou seja:

- **Definição do Processo OOM-NFR que transforma modelos i* em modelos objeto do OO-Method e analisa, antes da transformação, softgoals em termos de conflitos, prioridades e satisfação.**

  A principal contribuição foi o processo OOM-NFR, um processo transformativo que analisa os softgoals (requisitos não-funcionais relacionados a produto e interno ao ator sistema) de modelos i*, em termos de prioridades e conflitos, e gera distintos modelos objeto (PIM) em OO-Method que refletem esses softgoals.

  A geração de distintos em OO-Method derivados de um único modelo i* só foi possível porque o processo OOM-NFR trata, analisa e prioriza os softgoals (requisitos não-funcionais relacionados a produto e interno ao ator sistema) em i* wiki. Isso não ocorre na abordagem de (ALENCAR, 2009), nem nas demais abordagens, conforme mencionado na seção 3.3. Nessas abordagens, faz-se essa mesma transformação sem contemplar esse tipo de análise de softgoals, gerando-se apenas um único modelo OO-M, a partir do qual não se é capaz de se afirmar se esses softgoals estão satisfeitos (priorizados) e também estão sem conflitos. Não fosse o pré-processamento (Análise NFR) realizado pelo OOM-NFR, seria gerado apenas um único modelo objeto do OO-Method com todos os serviços gerados a partir das tarefas de i*, pois não seriam considerados os softgoals do modelo recebido como entrada em i*. O processo OOM-NFR pode ser visto didaticamente como dois subprocessos: o pré-processamento e processo núcleo. A primeira atividade deste primeiro subprocesso é verificar se há softgoals no modelo i* a ser processado. A principal atividade do pré-processamento é a análise de softgoals, verificando-se quais deles devem ser priorizados e satisfeitos. Ao longo do processo de resolução, conflitos são resolvidos, prioridades são analisadas, resultando numa configuração final de modelo i* satisfatória. A atividade final do pré-processamento é refatorar essa configuração final para remover os elementos negados e que não vão ser passados para a próxima etapa do processo OOM-NFR: o processo núcleo.

O processo núcleo, por sua vez, tem como principal atividade transformar o modelo i* para o modelo OO-Method através da aplicação das diretrizes de transformação. O processo núcleo transformativo do OOM-NFR é dividido basicamente em quatro atividades: geração de classes, geração de atributos, geração de serviços e geração de relacionamentos. Cada uma dessas quatro atividades é decomposta em diretrizes que definem quais elementos i* serão transformados nos elementos semanticamente equivalentes em OO-Method e também como é realizada essa transformação. Mais detalhes e exemplos das atividades do processo OOM-NFR ver capítulo 4.

- **Especificação de novas Diretrizes de transformação**

  Além de revisão das diretrizes já existentes em (ALENCAR, 2009), novas diretrizes foram adicionadas: **4.7, 4.8 e 4.9** conforme visto na seção 4.3.4

- **Automatização e Implementação do OOM-NFR em QVT**

  Outra contribuição importante deste trabalho foi a automatização do processo transformativo proposto na linguagem de transformação de modelos QVT, tornando a atividade de gerar um modelo OO-M a partir do i* mais rápida, menos propensa a erros humanos, dentre outras vantagens, descritas nos capítulos 1 e 4 ao se abordar o processo de desenvolvimento de software orientado a modelos (MDD). Detalhes dessa automatização do OOM-NFR, utilizando QVT, ver na seção apêndice.

- **Criação de um metamodelo ECORE OO-Method**

  Para realizar a transformações via metamodelos, conforme recomenda o padrão MDA, foi necessário criar um metamodelo Eclipse ECORE (ver seção 3.3.2) para representar os principais contrutores (metaclasses) do OO-Method (mais detalhes ver seção 3.3.2). O padrão ECORE é compatível com o UML EMOF.

  Este metamodelo ECORE, desenvolvido com o Eclipse GMF, metamodela apenas o **modelo objeto** (ver seção 3.2.1.3) do modelo conceitual do OO-Method. Os elementos desse metamodelo do OO-Method, necessários ao processo transformativo OOM-NFR, foram obtidos e adaptados a partir de (GIACHETTI, 2010). Entretanto, esse metamodelo, no padrão ECORE, é uma contribuição do autor desta pesquisa, pois a linguagem de transformação QVT requer os metamodelos nesse padrão para poder processá-los.

- **Modelagem do Sistema Agência Fotográfica no modelo i* SR final**
  Para melhor ilustrar o modelo i* wiki e atender aos propósitos de tratamento de requisitos não-funcionais deste trabalho, o modelo i* inicial do Sistema de Agência Fotográfica foi totalmente remodelado, sendo gerado os requisitos finais (late i* SR) onde foram adicionados o ator sistema – Photo Agency System (PAS) e vários softgoals necessários ao processo transformativo. Ressalta-se que o modelo Agência Fotográfica estava na formade requisitos iniciais (early i* SR, isto é sem o ator sistema) e totalmente sem requisitos não-funcionais (ALENCAR, 2009). A nova especificação dos sistemas (em i* SR final) visou facilitar e viabilizar as transformações, pois este tipo de modelo está mais enriquecido de detalhes operacionais, ou seja, mais próximo e passível de conversão num modelo de nível de projeto como é o OO-Method. Com este modelo foram exemplificadas e aplicadas todas as regras de transformação presentes no processo núcleo do OOM-NFR e realizada também a análise NFR do pré-processamento.
- **Validação do Processo OOM-NFR através de um exemplo de aplicação**
  Visando dar um exemplo mais concreto e real de aplicação do processo OOM-NFR, foi especificado em i* wiki um Sistema de Controle de Acesso (mais detalhes ver seção 5.3). Esse sistema, chamado SCA, tem vários softgoals como segurança, desempenho e usabilidade, etc; e foi especificado a partir de sua documentação UML (caso de usos) existente, representando um modelo simplificado de sistema real e em produção utilizado para fins desta pesquisa. Todas as atividades do processo OOM-NFR foram detalhadas e ilustradas a partir do SCA.

6.2 Limitações

Em toda pesquisa científica, seja qualitativa ou quantitativa, para se atingir objetivos específicos, geralmente, identificam-se limitações e considerações ideais. As principais limitações observadas ou encontradas nesta dissertação estão descritas a seguir:

- **Metamodelo OO-Method**

O modelo conceitual do OO-Method é composto por 4 submodelos: modelo de classes ou objeto, modelo funcional, modelo dinâmico e modelo de apresentação (seção 3.3.1). Nesta pesquisa foi metamodelado apenas o modelo objeto (modelo de classe). Ainda, no modelo objeto do OO-Method, alguns elementos não foram metamodelados como, os relacionamentos de herança, agregação e composição já que os modelos i* a serem utilizados como entrada do processo foram restritos a não usar elementos de relacionamento entre atores i* (IS-A, IS-PART-OF, etc). Esses relacionamentos ainda possuem uma semântica imprecisa e por esta razão não foram considerados. Esta foi uma das simplificações consideradas.

- **Metamodelo i* wiki da ferramenta IStarTool**

  Para se obter os modelos Agência Fotográfica e SCA foi utilizado a ferramenta IStarTool (ISTARTOOL, 2009), que tem o metamodelo do i* descrito em ECORE (ver seção 2.3.2).

  Este metamodelo é muito simples e facilita o trabalho de edição gráfica dos modelos criados, mas carece ainda de elementos que facilitem transformações "Model to Model" para OO-Method. Uma dessas dificuldades é o uso de "enumerations" para representar os tipos de elementos intencionais. Com "enumerations" fica inviável a criação de atributos extras, principalmente para o elemento intencional "recurso" que precisa de atributos para fazer a distinção entre um recurso físico e um recurso informativo. O ideal seria criar metaclasses distintas para cada elemento intencional do i* para então criar atributos extras dessas metaclasses necessários ao processo transformativo. Esta é uma limitação decorrente do metamodelo e ferramenta IStarTool. Outra restrição é quanto à versão atual dessa ferramenta que possibilita a edição de alguns relacionamentos SD ou SR incompatíveis com o modelo i* wiki, acarretando a edição de modelos incorretos e, consequentemente transformações incorretas. Para o processo OOM-NFR se fez a suposição de que os modelos i* de entrada estavam corretos, não impactando assim nas transformações realizadas nem introduzindo complexidade ao processo para tratar esses erros. Mesmo se fosse utilizado a ferramenta OpenOME e seu metamodelo, alguns desses problemas ainda persistiriam, principalmente pela ausência de atributos extras das metaclasses necessários na transformação para

OO-M. Trabalhos futuros podem realizar uma melhor análise da ferramenta OpenOME, visando levantar as suas características e de seu metamodelo para saber se ela é mais adequada a automatização do processo OOM-NFR do que a ferramenta IStarTool. Uma nova versão da ferramenta de modelagem IStarTool, incorporando algum processo de avaliação de modelos i*, poderia garantir uma melhor qualidade dos modelos i* especificados.

- **Transformações Realizadas**

  Algumas limitações foram observadas no processo transformativo, as quais se destacam.

  - Inviabilidade de gerar cardinalidade nos relacionamentos de associação, pois não há como inferir cardinalidade a partir do modelo i* wiki. Impossibilidade de gerar direção (in-entrada, out-saída) nos parâmetros dos serviços devido a ausência de elementos em i* dos quais possam se inferir tais primitivas.
  - Inviabilidade de transformação automática via algumas diretrizes que geram relacionamentos herança e agent link, pois dependem da interferência do analista de requisitos (atividade manual).

- Perda de alguns elementos i* relevantes após o processo de transformação, ou seja, a partir do modelo de classe OO-Method gerado, não é possível saber quais destes elementos OO-Method estavam relacionados com: depender, dependee e dependuum; serviços que estavam associados a quais elementos i* em um relacionamento "means-end", etc.

- **Limitações de QVT**

  Algumas limitações e deficiências da linguagem QVT (QVT, 2009), como

  - Ferramenta de *debug* e *trace* (registro das transformações realizadas ou erros obtidos) precária;
  - Composição de modelos não é feita;
  - Composição de funções de transformações não é feita;
  - Impossibilidade de transformações bidirecionais, etc.
  - Interação com os modelos estritamente com OCL.

  Essas limitações são decorrentes, principalmente, porque QVT ainda está evoluindo e não alcançou o nível de maturidade desejado. Entretanto, a maior parte das dificuldades apresentadas na implementação das diretrizes de

transformação estão relacionadas com a insuficiência de informações do metamodelo i* de origem, conforme já se mencionou; e, também devido ao "gap" semântico e questões de interoperabilidade entre os modelos i* e OO-Method.

- **Limitações dos RNFs e Método de Resolução**

  Os requisitos não-funcionais são complexos e têm uma enorme classificação. Assim, para melhor visualizar os resultados do processo OOM-NFR foram considerados somente RNFs de produto (SOMMERVILLE, 1998) e que estão relacionados aos softgoals do modelo i* internos ao ator sistema (ver seção 2.2). Isso porque esses são factíveis de terem uma solução de software, ou seja, mais facilmente implementáveis (operacionalizados) através de tarefas, métodos codificados. Entretanto, o processo apresentado nesta dissertação pode também contemplar e analisar RNFs externos (SOMMERVILLE, 1998), desde que também sejam softgoals internos ao ator sistema e operacionalizados, conforme mencionado na seção 2.2.

  O Processo de Pré-processamento do OOM-NFR utiliza o algoritmo de resolução Zchaff SAT solver (MAHAJAN et al., 2004). Um problema SAT é NP-Completo. Sendo assim, esse resolvedor de fórmulas lógicas pode nunca parar (tentar resolver indefinidamente uma fórmula); pode parar e não gerar ou encontrar uma configuração possível que satisfaça as priorizações e trate os conflitos; e pode chegar a configurações distintas para as mesmas prioridades, dependendo do que analista decidir durante as interações na análise NFR.

Observa-se que, algumas dessas limitações foram, na verdade, simplificações visando facilitar, melhor conduzir e gerenciar o estudo da transformação de modelos. Outras limitações foram decorrentes das ferramentas e recursos utilizados.

6.3 Lições Aprendidas

A partir das limitações e dificuldades encontradas durante a especificação, implementação e validação do processo OOM-NFR, pode-se citar algumas lições aprendidas:

- Deve-se considerar limitações ou restrições para realização das transformações: algumas serão plenamente automatizáveis, outras semi-automatizáveis e outras manuais.

- Deve-se definir os elementos que interoperam entre os modelos a serem transformados, ou seja, definir através de quais elementos dos metamodelos de origem e destino se pode obter equivalência semântica. Assim, é necessário conhecer profundamente os modelos (sintaxe concreta) e metamodelos (sintaxe abstrata) das linguagems de origem e de destino.
- Deve-se dominar metamodelagem e tecnologias de transformação usando metamodelos, conhecendo seus potencias e também suas limitações.

6.4 Trabalhos Futuros

Face às limitações e ao escopo desta dissertação, algumas sugestões de trabalhos futuros complementares são apresentadas:

- **Realizar estudos de caso**

  Como cada aplicação modelada em i* tem caraterísticas e softgoals distintos, é interessante que sejam realizados experimentos mais formais, tal como estudo de casos, que possam melhor validar o processo OOM-NFR e possibilitar deteção e correção de erros, num processo contínuo de melhoria.

- **Estender a ferramenta IStarTool**

  Algumas características básicas poderão ser integradas à ferramenta IStarTool, visando realizar as atividades propostas pelo OOM-NFR e evitar o uso de outras ferramentas externas de apoio:

  - Implementar algoritmo de resolução SAT-solver zchaf (MAHAJAN et al., 2004), pois foi utilizado o implementado na ferramenta OPENOME para ilustrar os exemplos. Isso evitaria a necessidade de reescrever o modelo em outra ferramenta e faria com que IStarTool tivesse capacidade de resolução ou raciocínio;
  - Implementar refatoração do modelo i* final, gerado após a fase final do pré-processamento do OOM-NFR
  - Integrar à ferramenta IStarTool as trasformações i* para OO-M implementadas em QVT.

- **Estender o metamodelo OO-Method utilizado**

  Como o metamodelo utilizado foi um subconjunto reduzido do modelo conceitual do OO-Method, pode-se futuramente estender esse metamodelo para incluir mais

elementos OO-M desde que esses novos elementos adiconados tenham alguma equivalência semântica com os construtores i* e sejam passíveis de transformação.

- **Estender o metamodelo i* wiki**

  Há muito a fazer quando se trata em estender o metamodelo i* wiki. O metamodelo i* utilizado, que é o mesmo da ferramenta de edição IStarTool, carece de várias características: novas metaclasses, novos atributos, novas associações que facilitem a implantação e automação, via linguagem (QVT, ATL, etc), das diretrizes de trasformações.

  Entretanto essa extensão não deve ser arbitrária e descontrolada, sob o risco de, até mesmo, descaracterizar o i* como modelo de requisitos. Alguns exemplos dessas possíveis extensões são:

  - definir os elementos intencionais Recursos, Tarefas, como metaclasses próprias e não como uma metaclasse única **Element** *que possui o atributo* enumeração **ElementType** onde se referencia a todos elementos intencionais;
  - definir explicitamente no metamodelo quais dos elementos intencionais, num link de dependência, é o depender, dependumm e dependee. Isso otimizaria enormemente a implementação das regras, pois não haveria necessidade de se recuperar dois pares separados de link de dependência, saber quem é a fonte "source" e alvo "target", para depois inferir depender,dependuum e dependee. Para resolver isso talvez houvesse a necessidade de se definir uma nova metaclasse Link de Dependência (DependencyLink) que tivesse atributos adicionais referenciando mais construtores do i* wiki;
  - definição de atributos extras, por exemplo, para diferenciar recursos físicos de lógicos, atores sistema de atores não sistema, etc.

- **Desenvolver editor gráfico OO-Method que importe o modelo xmi gerado, valide e o exiba graficamente.**

  Este recurso facilitaria a visualização gráfica do modelo OO-M gerado e daria, inclusive, suporte a posteriores refinamentos necessários.

- **Fornecer mecanismos de Verificação ou Validação dos modelos i* wiki**
  Isso evitaria passar modelos para o processo OOM-NFR com erros.

- **Definição de Métricas para aferir a qualidade do processo transformativo**
  Um trabalho como este ajudaria em muito a fornecer informações tais como: eficiência do processo, percentagem de elementos transformados, erros ou falhas de transformações encontradas, entre outras.

- **Testar o Processo Núcleo (Diretrizes de Transformações) sob a ótica da Engenharia de Testes**

  Para se testar eficientemente um software, faz-se necessário definir um projeto de teste, especificando casos de testes que são executados dinamicamente para verificar a existência de "bugs" ou defeitos. Testes não afirmam que um software esteja correto ou sem defeitos, mas garantem, no mínimo, que as funcionalidades (transformações) especificadas estão produzindo as saídas corretas a partir das respectivas entradas.

- **Análise formal detalhada de interoperabilidade e equivalência semântica entre os modelos i* e OO-Method**

  Apesar deste trabalho utilizar recursos informais, experiência e conhecimento técnico sobre esses modelos, há espaço para um estudo mais formal sobre a equivalência entre essas duas linguagens, a fim de melhor compreendê-las sobre esse prisma, trabalhar com provas e verificar possibilidades e limitações mais fundamentadas.

Percebe-se que podem ser definidos vários trabalhos futuros e complementares a esta pesquisa. Todavia, esta dissertação representa uma relevante contribuição pela proposta do OOM-NFR, no sentido de realizar transformações do modelo de requisitos i*, de alto nível de abstração, para o modelo OO-Method, com base nos requisitos não-funcionais, ou seja, softgoals do modelo i* internos ao ator sistema.

Através do OOM-NFR, consegue-se inserir modelos de requisitos i* num processo de desenvolvimento de software MDD para OO-M, considerando a análise de softgoals (priorização, satisfação e resolução de conflitos). Essa análise poderia contribuir, inclusive, para facilitar a adoção do i* pela indústria de software. Além disso, esse processo transformativo OOM-NFR é automatizado, implementando-se as transformações na linguangem de transformações de modelos QVT (ver apêndice) e tornando a atividade de gerar um modelo OO-M a partir do i* mais rápida e menos propensa a erros humanos.

Referências

ALENCAR, F. Mapeando a Modelagem Organizacional em Especificações Precisas. (Tese de Doutorado) em Ciência da Computação, Universidade Federal de Pernambuco, Centro de Ciências Exatas e da Natureza, Recife, Brasil, 1999.

ALENCAR, F; PEDROZA F.; CASTRO J. et al. New mechanisms integration of organizational requirements and object oriented modeling. In Proceedings of the VI workshop on requirements engineering. Proceedings of the VI workshop on requirements engineering (WER'03). , Piracicaba - SP, Brasil, p.109-123, 2003.

ALENCAR, F.; MARÍN, B.; GIACHETTI, G.; PASTOR, O.; CASTRO, J.; PIMENTEL, J.H. From i* Requirements Models to Conceptual Models of a Model Driven Development Process. In: PoEM 2009. Springer LNIBP, 2009.

ALENCAR, F.; MARÍN, B.; GIACHETTI, G.; SANTOS, EMANUEL.at al. From i* to OO-Method: Problems and Solutions. Proceedings of 4th International i* Workshop, Hammamet, Tunisia, 2010a.

ALENCAR, F.; MARÍN, B.; GIACHETTI, G.; PASTOR, O.; CASTRO, J.: Beyond Requirements: An Approach to Integrate i* and Model-Driven Development. In CIBSE 2010b.

BOEHM, B. Characteristics of Software Quality. North Holland Press, 1978

BIZAGI. Businnes Process Modeler tool, 2009, <http://www.bizagi.com/>. Acesso em março 2011.

BPMN. Businness Process Modeling Notation, 2009. <http://www.omg.org/bpmn/>. Acesso em julho 2009.

BRESCIANI, P.; GIORGINI, P.; GIUNCHIGLIA, F.; MYLOPOULOS, J.; PERINI, A. "Tropos: An Agent-Oriented Software Development Methodology". Autonomous Agents and Multi-Agent Systems, p.203–236, 2004.

CASTRO, J.;ALENCAR, F.;CYSNEIROS, G. Closing the GAP Between Organization Requirements and Object Oriented Modeling. 2000.

CASTRO, J.;ALENCAR, F.;CYSNEIROS, G.; MYLOPOULOS J. Integrating Organizational Requirements and Object Oriented Modeling. 2001.

CASTRO, J.; KOLP, M.; Mylopoulos, J. Towards Requirements Driven Information Systems Engineering : The Tropos Project. Journal: Information Systems. Holanda: , v.27, (6) p.365 - 389, 2002.

CASTRO, J. Modelagem Organizacional. Apresentação CIN/LER, 2008.

CYSNEIROS, L.M.; Leite, J.C.S.P. Integrating Non-Functional Requirements into data model. 4th International Symposium on Requirements Engineering, Ireland,1999.

CHUNG, L.; NIXON, B.; YU, E.; MYLOPOULOS, J. Non-Functional Requirements in Software Engineering. 1. ed. [S.l.]: Springer, 1999.

DAVIS, A. Software Requirements: Objects Functions and States. Prentice Hall, 1993.

ECLIPSE. Eclipse Projects, 2009. <http://www.eclipse.org/projects/> Acesso em julho 2010.

ECLIPSE ATL DOCUMENTATION 2009.<http://www.eclipse.org/m2m/atl/doc/>. Acessado em julho 2009.

EMF. Eclipse Modeling Framework, 2009. Disponivel em: <http://www.eclipse.org/modeling/emf>. Acesso em: Fevereiro 2011.

GIACHETTI, G. Supporting Automatic Interoperability in Model-Driven Development Processes. (Tese de Doutorado) Universidad Politecnica, Valencia, Canada. 2011.

GIACHETTI, G.; MARIN, B; PASTOR, O. Integration of domain-specific modeling laiu UML through UML profile extension mechanism. Int J Computer Sei Appl 6(5), 2009.

GIACHETTI, G.; VALVERDE, F.; Pastor, O.: Improving Automatic UML2 Profile Generation for MDA Industrial Development. In: 4th International Workshop on Foundations and Practices of UML (FP-UML) – ER Workshop. LNCS, p.113–122. Springer, 2008.

GRAU, G. et al. i* Guide, 2008. Disponivel em: <http://istar.rwth-aachen.de/>. Acesso em: Fevereiro 2011.

GRONBACK, R. Eclipse Modeling Project: a Domain-Specific Language (DSL) Toolkit, Addison Wesley, 2009.

HITACH, M. O. MDA and System Design. Presentation at MDA Information Day, OMG Technical Meeting, April, 2002.

HORKOFF, J.; Yu, E. Finding Solutions in Goal Models: An Interactive Backward Reasoning Approach.Proceedings of Conceptual Modeling, 2010.Conf. on Conceptual Modeling, Vancouver, Bc, Canada, November, p.1-4, 2010.

ISTARTOOL. IStar modeling Tool support, 2009. Disponível em: <http://portal.cin.ufpe.br/ler/Projects/IStarTool/>. Acessado em janeiro 2011.

JACKSON; M. Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices. Addison Wesley, 1995.

KELLER, S.E. et al. Specifying Software Quality Requirements with Metrics in Tutorial System and Software Requirements Engineering IEEE Computer Society Press p.145-163, 1990.

KELLY, S.; TOLVANEN; J-P. Domain-Specific Modeling:Enabling Full Code Generation. Wiley-IEEE Society Press, 2008

KOTONYA, G.; SOMMERVILLE, I. Requirements Engineering: Processes and Techniques. New York, NY, USA: John Wiley & Sons, Inc., 1998.

KOSKO B.; Fuzzy Engineering, Prentice-Hall, 1997.

LAMSWEERDE, A. V. Requirements Engineering in the Year 00: A Research Perspective. In: Keynote paper, Proc. ICSE'2000 - 22nd Intl. Conference on Software Engineering. IEEE Computer Society Press, 2000.

LER. Lab. de Engenharia de Requisitos. < http://portal.cin.ufpe.br/ler>. Acessado em abril 2011.

MAHAJAN,Y.S.; FU, Z.; MALIK, S.: Zchaff 2004: An Efficient SAT solver. In: H. Hoos,H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 360–375. Springer, Hei-delberg, 2005.

MAIRIZA, D.; DIDAR, Z. et al. Na Investigation into the Notion of Non-Funcional Requirements. SAC 2010.

MARÍN, B.; GIAGCHETTI, G.; PASTOR, O.: The Photography Agency: A case study of the OO-Method Approach. Technical Report DSIC-II/13/08, Universidad Politécnica de Valencia, Valencia, Spain, 2008.

MARTINEZ, A. Conceptual schemas generation from organizational models in na automatic software production process. (Phd Thesis) Valencia University of Technology. [S.l.]. 2008.

MDA. Model Driven Architecture, 2003. Disponivel em: <http://www.omg.org/mda>. Acesso em: agosto 2011.

MORGAN, T. 2002. Business rules and information systems – aligning IT with business goals.Addison-Wesley, Reading, MS, 2002.

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

NUSEIBEH,B.; KRAMER ,J.; FINKELSTEIN, A. "A Frameworkfor Expressing the Relationships Between Multiple Views in Requirements Specifications", IEEE Transactions on Software Engineering, Vol. 20 No. 10, p.760-773, October 1994.

OLIVANOVA. OlivaNova Care Technologies, 2005. http://www.care-t.com. Acesso em: Agosto 2010.

OME. Organization Modelling Environment, 1998. http://www.cs.toronto.edu/km/ome. Acesso em: Março 2011.

OMG. Object Management Group, 1997. Disponivel em: <http://www.omg.org/>. Acesso em: Fevereiro 2011.

OPENOME. 2010. Disponível em: < https://se.cs.toronto.edu/trac/ome/wiki>. Acessado em Agosto 2011.

PAES, J. AGILE: Uma Abordagem para Geração Automática de Linguagens i*. (Dissertação de Mestrado), Centro de Informática, Universidade Federal de Pernambuco, Recife, Brasil, 2011.

PASTOR ,O. Diseño y desarrollo de un entorno de producción automática de software basado en el modelo orientado a objetos. PhD Thesis, DSIC, Universidad Politécnica de Valencia,Valencia, 1992.

PASTOR, O.; Hayes, F.; Bear, S. OASIS: an OO specification language. In: Proc CAISE-92 Conf, Springer, Berlin Heidelberg New York, LNCS 593, p.348–363, 1992.

PASTOR, O.; MOLINA, J. Model-Driven Architecture in Practice: A software Production Environment Basel on Conceptual Modeling. [S.l.]: Springer, 2007.

PASTOR, O.; GIACHETTI, G. Intentional Perspectives on Information Systems Engineering, p 257-275, Springer, 2010.

QVT. Query,View,Transformation, 2009. ECLIPSE QVT DOCUMENTATION http://eclipse.org/m2m/qvto/doc. Acessado em julho 2009.

ROMA, G. A Taxonomy of Current Issues in Requirements Engineering. IEEE Computer Vol. 18, p.14-23, No. 4 Apr. 1985.

SAALTINK, M. The Z/EVES System. 19th International Conference on the Z Formal Method (ZUM), Reading, UK, LNCS 1212, p.72-88, 1997.

SAATY, T.L. Relative Measurement and Its Generalization in Decision Making Why Pairwise Comparisons are Central in Mathematics for the Measurement of Intangible Factors The Analytic Hierarchy/Network Process. Rev. R. Acad.Cien. p.102, 2 251-318, 2008.

UML. Unifield Modeling Language, 2000. Disponivel em: <http://www.omg.org/uml>. Acesso em: Fevereiro 2011.

VISIO. Microsoft VISIO, 2010. Disponivel em: <http://office.microsoft.com/pt-br/visio>. Acesso em: Fevereiro 2012.

YU, E. Modelling Strategic Relationships for Process Reengineering. (Tese de Doutorado) University of Toronto. Toronto, ON, Canada, 1995.

APÊNDICE– Automatização do OOM-NFR em QVT

O processo OOM-NFR possui basicamente dois módulos ou componentes: o pré-processamenteo e o processo núcleo. Destes, apenas o processo núcleo, que define as diretrizes de transformações dos elementos i* para OO-Method, conforme mostrado no capítulo 4, é que foi automatizado (implementado), ou seja, codificado na linguagem de transformação de modelos QVT (QVT, 2009).

Neste apêndice está listada a implementação (codificação) das diretrizes especificadas no capítulo 4 desta dissertação e que geram as classes, atributos, serviços e relacionamentos do OO-Method. Ainda neste apêndice, estão também os comentários dos trechos de código QVT que realizam efetivamente as transformações especificadas. A razão de deixar a parte de código neste Apêndice foi separar especificação de requisitos da implementação dessas diretrizes a fim de facilitar a clareza e não misturar os dois níveis de abstração.

O pré-processamento é praticamente a análise NFR que é realizado como apoio do sat-solver implementado na ferramenta OPENOME. Cabe-se mencionar que a implementação do pré-processamento na ferramenta de modelagem IstarTool, a partir da qual foram obtidos os metamodelos de entrada e modelos i* exemplos, não estava no escopo deste trabalho, ficando como mencionado na seção 6.4 como trabalhos futuros. Entretanto algumas atividades do pré-processamento podem ser facilmente implementadas em QVT tais como verificação da existência de requisitos não-funcionais no modelo de entrada e refatoração da configuração final obtida com a análise NFR.

QVT (QVT, 2009) é acrônimo para Query, View e Transformation, um padrão OMG para linguagem de transformação de modelos, usando metamodelos. A linguagem QVT, na plataforma Eclipse, é basicamente constituída pela linguagem imperativa QVTO-QVT Operational, OCL–Object Constraint Language e a biblioteca padrão (Standar Library).

Inicialmente a figura 97 mostra a tela de "run configurations" do Eclipse. Nesta tela estão especificados os seguintes arquivos:

1. **IstarOOmEspec.qvto**: módulo transformador (código fonte em QVT)
2. **IstarOOmEspec.qvtotrace**: arquivo de rastreabilidade gerado pelo compilador de modelos QVT para cadas transformação realizada.
3. **agenciafotoistar.xmi**: instância i* wiki de entrada, ou seja, modelo i* de entrada

4. **agenciafoto_oom.xmi**: instância OO-Method de saída (modelo OO-M saída)

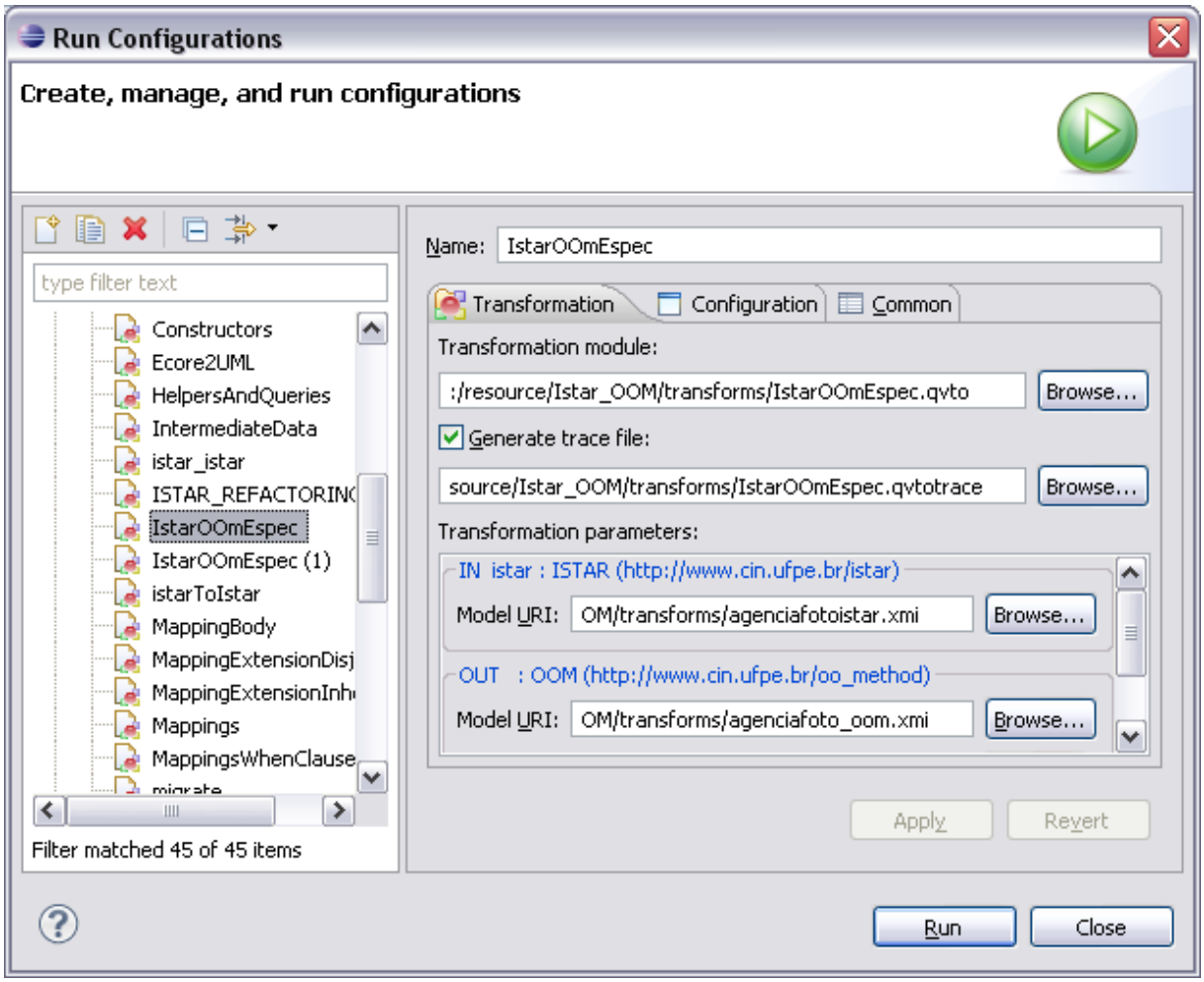


Figura 97. Tela do Eclipse Run Configurations

Os parâmetros IN istar: ISTAR e OUT: OOM representam os metamodelos i* wiki e OO-Method respectivamente. A partir desta tela é possível executar "Run" a Transformação.

A seguir está é exibida a listagem do código implementado em QVT do módulo tradutor (IstarOOmEspec.qvto). Esse módulo pode ser visto como um compilador de modelos. As regras/diretrizes de transformação especificadas no capítulo 4 estão aqui implementadas. Antes de se invocar as funções de mapeamento "map" que efetivamente transformam elementos do modelo i* wiki em elementos do modelo OO-M (funções que têm, como assinatura (parâmetros), os respectivos elementos dos metamodelos e, como corpo, o mapeamento propriamente dito), geralmente são realizadas consultas "queries" que obtêm os elementos do i* necessários às transformações.

Cada trecho do código está com seu respectivo comentário, explicando brevemente o que faz e a que diretrizes de transformação está relacionado.

```
/*
* Módulo tradutor IstarOOmEspec.qvto
* Transforma uma instância i* wiki em OO-Method
* Este Módulo representa o Processo Núcleo do OOM-NFR
*/

/*
* Definição dos metamodelos ECORE do i* wiki(ISTAR) e do OO-Method (OOM)
* A cláusula "where" indica que modelo instância do ISTAR só é processado
* se houver elementos softgoals
*/
modeltype ISTAR uses istar('http://www.cin.ufpe.br/istar')
where {self.elements->closure(oclIsKindOf(ISTAR::ElementType::SOFTGOAL))-
>size()>0 };
modeltype OOM uses oo_method('http://www.cin.ufpe.br/oo_method');



/*
* Assinatura da transformação, dizendo que uma instância de entrada,
* compatível com o metamodelo ISTAR, será utilizada e uma instância
* de saída,compatível com OOM será obtida como resultado
*/
transformation IstarOOm(in istar : ISTAR, out OOM);



/*
* MetaClasse temporária que representa uma GenericDependencyLink - GDL
* Esta classe é fundamental para derivar relacionamentos OO-Method e alguns
* serviços. No metamodelo ISTAR, um DependencyLink é tem a estrutura
* NodeObject-|)NodeObject, onde NodeObject pode ser Ator ou Elemento Intenc
* Uma GDL tem estrutura Ei-|)Em-|)Ef, onde Ei,Em,Ef podem ser Ator ou
* Elemento Intencional.Com esse conceito de GDL,é possível saber se o link
* de dependencia é do tipo SD ou SR , quem é o Depender,Dependee
* e Dependum,Se o link de dependência é de recurso, tarefa,etc.
* AEi - A Qual Ator pertence Ei(Elemento Inicial),
* AEf - A Qual Ator pertence Ef(Elemento Final) e o Em (Elemento do meio)
* Uma vez montada todas GDLs do Modelo ISTAR, Todo processo de  transfor-
* mação e aplicação das diretrizes fica bastante facilidado.
*/
intermediate class GDL {
      Ei:NodeObject;
      Em:NodeObject;
      Ef:NodeObject;
      AEi:Actor;
      AEf:Actor;
      is_SD:Boolean;
      };
/*
* Classes temporárias que representam tuplas de elementos ISTAR  visando
  transformações especificadas nas diretrizes de relacionamentos
*/
intermediate class TuplaRecFis_Ator{
rf:ISTAR::Element;
at:ISTAR::Actor;
}
intermediate class TuplaRecfis_RecFis{
rfo:ISTAR::Element;
rfd:ISTAR::Element;
```

```
}
intermediate class TuplaAtor_Tarefa_AgentLink{
ator_depender:ISTAR::Actor;
tarefa_recfis_ator_dependee:ISTAR::Element;
}

/*
* Conjunto dos elementos do Modelo ISTAR,indispensáveis para transformações
*/
property elementos_modelo:OrderedSet(ISTAR::Element)=null;
property links_modelo:OrderedSet(ISTAR::DependencyLink)=null;
property gdls:OrderedSet(GDL)=null;
property associacoes_atores:OrderedSet(ISTAR::ActorAssociation)=null;
property atores_modelo:OrderedSet(ISTAR::Actor)=null;
property recursos_fisicos:OrderedSet(ISTAR::Element)=null;
property recursos_info:OrderedSet(ISTAR::Element)=null;


/*
* Ponto de entrada  de todo processo transformativo(tradutor )
* parâmetro istar recebe todos elementos raízes do modelo de entrada
* do tipo ISTAR::Model e invoca o primeira transformação model2OOMModel
*/
 main() {
istar.rootObjects()[ISTAR::Model]->map model2OOMModel();
// this.atores_modelo:=getAtores();

}


/* Transformação que inicializa os conjuntos a serem utilizados
* posteriormente nas consultas e mapeia a classe do tipo ISTAR::Model em
* OOM::Model,atribuindo os repectivos títulos e as principais agregações
* raízes do modelo OOMque serão transformadas: classes, associations e
* agents. Nesta Transformação inicial são chamadas as demais transformações
* responsáveis por gerar as classes,associações e agents.
* Como elements são agregações do modelo e também de um ator, para o
* processamento desses elements do modelo nos mapeamentos subsequentes,
* faz-se necessário passá-lo como argumento de entrada.
* o operador union é utilizado para unir as duas sequências geradas para
* formar  as classes do modelo OO-M
*/
mapping ISTAR::Model::model2OOMModel() : OOM::Model {
init{
this.atores_modelo:=self.getAtores();
this.elementos_modelo:=self.getElementosModelo();
this.links_modelo:=self.getLinksModelo();
this.gdls:=self.monta_todas_GDLs();
this.associacoes_atores:=self.get_Associacoes_Atores();
this.recursos_fisicos:=self.getRecursosFisicos();
this.recursos_info:=self.getRecursosInfo();
      }

title := self.title;

/*
Mapeamentos que transformam Atores e Recursos Físicos em Classes OO-Method
*/
classes:= self.actors[ISTAR::Actor]->asOrderedSet()->map
ator2classe(self.elements)->
```

```
            union(self.getRecursosFisicos()->asOrderedSet()->map
recursofisico2classe())->asOrderedSet();

/*
* Mapeamentos que transformam Relacionamentos ISA ISTAR e os Relacionamen-
* tos inferidos pelas Diretrizes 4.1 a 4.7 em Associações de Classes OO-M.
* Como nem nos modelos de entrada ISTAR,nem no metamodelo OO-M são
* utilizados construtores de Herança, eles foram considerados associações
*/
associations:= self.getRelacionamentos_ISA()-> map
AtorAssociation2Association->
            union( self.getHerancaRecFisicos->map
RecFisHeranca2Association())->
            union( self.getRelacionamentosGDLs_4_3->map GDL2Association())-
>
            union( self.getTuplasRecFis_Ator()->map
recfis_ator2Association())->
            union(self.getTuplasRecFis_RecFis_4_7->map
recfis_distintos2Association() )->asOrderedSet();

/*
* Mapeamentos que  Geram  os Relacionamentos AgentLink do OO-M e
* Especificados pelas diretrizes 4.8 e 4.9
  */
agents:= self.getTuplasAgentLink4_8()->map Tupla4_8toAgentLink()->
                  union(self.getTuplasAgentLink4_9()->map
Tupla4_9toAgentLink()) ;

}

/*
* Mapeamento dos atores em classes e chamada dos mapeamentos para gerar
* os atributos e serviços desses atores transformados em classes OO-M
* Antes dos Mapeamentos dos atributos e serviços são invocadas as queries
* para inferí-los a partir do modelo ISTAR
*/
mapping ISTAR::Actor::ator2classe(): OOM::Class{
      name := self.name;
   attributes:= self.inferir_atributos_ator(self)->map
recinfo_ator2attribute();

   services:= self.inferir_servicos_ator(self)->map task_ator2service();
}

/*
* Mapeamento que transforma um recurso informativo(lógico)
* em um Atributo da classe gerada a partir de um ator
*/
mapping ISTAR::Element::recinfo_ator2attribute(): OOM::Attribute{
      name := self.name;
}

/*
* Mapeamento que transforma uma tarefa de um ator em um serviço
* da Classe gerada do ator.
* Cada serviço OO-M tem um nome, um tipo (kind) e parametros
* Os parámetros são obtidos a partir dos recursos informativos
* que fazem parte da decomposição da respectiva tarefa.
*/
mapping ISTAR::Element::task_ator2service(): OOM::Service{
      name := self.name;
```

```
      kind:= OOM::ServiceKind::none;
      parameters:= self.getParameters()->map element_resource2parameter();
}

/*
* Mapeamento que transforma recursos informativos de uma tarefa  em seu
* parâmetro(argumento).Cada parâmetro tem um nome e uma direção que está
* setada por default com "in" entrada já que o modelo ISTAR  não possui
* contrutores que possibilitem inferir esta direção.
*/
mapping ISTAR::Element::element_resource2parameter(): OOM::Parameter{
      name:=self.name;
      direction:= OOM::DirectionKind::in;
      }

/*
* Mapeamento de recurso físico em classe OO-M.
* Cada classe é composta de atributos e serviços
*/
mapping ISTAR::Element::recursofisico2classe(): OOM::Class{
      name := self.name;
    attributes:= self.inferir_atributos_recurso_fisico(self)->map
recinfo_recursofisico2attribute();
      services:= self.inferir_servicos_recurso_fisico(self)->map
task_recursofisico2service();
  }

/*
* Mapeamento que transforma um recurso informativo(lógico)  em atributo de
* uma Classe Gerada a partir de um Recurso Físico.
*/
mapping ISTAR::Element::recinfo_recursofisico2attribute(): OOM::Attribute{
      name := self.name;
}

/*
* Mapeamento que transforma uma tarefa em um serviço
* da Classe gerada a partir de um recurso físico
*/
mapping ISTAR::Element::task_recursofisico2service(): OOM::Service{
      name := self.name;
      kind:= OOM::ServiceKind::none;
      parameters:= self.getParameters()->map element_resource2parameter();
}

/*
* Mapeamento que transforma uma Associação ISA de Ator ISTAR em uma
Associação
* de Classes OO-M
*/
mapping
ISTAR::ActorAssociation::AtorAssociation2Association:OOM::Association{
source:=self.source.map ator2classe();
target:=self.target.map ator2classe();
}

/*
* Mapeamento que transforma um relacionamento Herança entre Recurso Físicos
em uma Associação
* de Classes OO-M
*/
```

```
mapping TuplaRecfis_RecFis::RecFisHeranca2Association():OOM::Association{
source:=self.rfo.resolve(OOM::Class);
target:=self.rfd.resolve(OOM::Class);
}

/*
* Mapeamento que a partir de uma metaclasse GlobalDependencyLink - GDL
* gera em uma associação de classes OO-M
* Este Mapeamento representa a diretriz 4.3
*/
mapping GDL::GDL2Association():OOM::Association{
      source:=self.AEi.resolve(OOM::Class);
    target:=self.AEf.resolve(OOM::Class);
}

/*
* Mapeamento para diretrizes 4.4,4.5 e 4.6
*
*/
mapping TuplaRecFis_Ator::recfis_ator2Association():OOM::Association{
source:=self.rf.resolve(OOM::Class);
target:=self.at.resolve(OOM::Class);
}

/*
* Mapeamento que deriva uma asssociação OO-M a partir da Diretriz 4.7
*/
mapping
TuplaRecfis_RecFis::recfis_distintos2Association():OOM::Association{
source:=self.rfo.resolve(OOM::Class);
target:=self.rfd.resolve(OOM::Class);
}
/*
* Mapeamento que converte a tupla inferida pela Diretriz 4.8 em
* Agent Link OO-M
*/
mapping TuplaAtor_Tarefa_AgentLink::Tupla4_8toAgentLink():OOM::AgentLink{
source:=self.ator_depender.resolve(OOM::Class);
target:=self.tarefa_recfis_ator_dependee.resolve(OOM::Service);
}

/*
* Mapeamento que converte a tupla inferida pela Diretriz 4.9 em
* Agent Link OO-M
*/
mapping TuplaAtor_Tarefa_AgentLink::Tupla4_9toAgentLink():OOM::AgentLink{
source:=self.ator_depender.resolve(OOM::Class);
target:=self.tarefa_recfis_ator_dependee.resolve(OOM::Service);
}

/*
* Consulta que infere os atributos dos atores
*/
 query ISTAR::Element::inferir_atributos_ator(in ator_alvo:ISTAR::Actor) :
OrderedSet(ISTAR::Element){

// Retorna Element do tipo Recurso
 var elem_atributos:OrderedSet(ISTAR::Element);
 ator_alvo.elements->forEach(element){
if element.is_recursoinformativo() and
element.is_relacionado_ao_ator(ator_alvo)then{
```

```
elem_atributos+=element;
}endif;
};

this.elementos_modelo->forEach(element){
if element.is_recursoinformativo() and
element.is_relacionado_ao_ator(ator_alvo)then{
elem_atributos+=element;
}endif;
};
 return  elem_atributos;
}

/*
* A seguir as consultas que populam as propriedades inicializadas
* no primeiro mapping das transformações
*this.atores_modelo:=self.getAtores();
*this.elementos_modelo:=self.getElementosModelo();
*this.links_modelo:=self.getLinksModelo();
*this.associacoes_atores:=self.get_Associacoes_Atores();
*/
query ISTAR::Model::getAtores() : OrderedSet(ISTAR::Actor) {
      return self.actors[ISTAR::Actor]->asOrderedSet()
}

query ISTAR::Model::getLinksModelo() : OrderedSet(ISTAR::DependencyLink) {
      return self.links[ISTAR::DependencyLink]->asOrderedSet()
}

query ISTAR::Model::getElementosModelo() : OrderedSet(ISTAR::Element) {
      return self.elements[ISTAR::Element]->asOrderedSet()
}

query ISTAR::Model::get_Associacoes_Atores() :
OrderedSet(ISTAR::ActorAssociation) {
      return self.associations[ISTAR::ActorAssociation]->asOrderedSet()
}


/*
* Consultas (queries) responsáveis por saber se um recurso é informativo
* , físico, tarefa ou está relacionado a um determinado ator
* Estas consultas serão chamadas em vários outros mapeaamentos
*/
query ISTAR::Element::is_recursoinformativo() : Boolean {
      return self.type.oclIsKindOf(ISTAR::ElementType::RESOURCE)and
self.isinformativo='true'
}

query ISTAR::Element::is_recursofisico() : Boolean {
      return self.type.oclIsKindOf(ISTAR::ElementType::RESOURCE) and
self.isinformativo='false'
}

query ISTAR::Element::is_tarefa() : Boolean {
      return self.type.oclIsKindOf(ISTAR::ElementType::TASK)
}


/*
* Consulta para saber se elemento recurso está associado ao ator alvo
```

```
*
*/
query ISTAR::Element::is_relacionado_ao_ator(in ator_alvo:ISTAR::Actor) :
Boolean {
      return self.type->('RESOURCE')and ator_alvo.stereotype-
>includes('related')
}


/* consulta para saber se se o recurso informativo está relacionado ao
*  recurso físico alvo
*/
query ISTAR::Element::is_relacionado_ao_recurso_fisico(in
recurso_alvo:ISTAR::Element) : Boolean {
      return self.type->('RESOURCE')and recurso_alvo.stereotype-
>includes('related')

}


/* Consulta para gerar serviços de um ator transformado em classe
* Este mapeamento invoca as diretrizes 3.1, 3.4 e 3.5 definidas para
* realizar essas transformações.
*
*/
query ISTAR::Element::inferir_servicos_ator(in ator_alvo:ISTAR::Actor) :
OrderedSet(ISTAR::Element){

return self.diretriz_3_1_tarefas_rec_info_ator(ator_alvo)->
                  union(self.diretriz_3_4_tarefas_rec_fisico(ator_alvo))->

      union(self.diretriz_3_5_tarefas_internas_ator(ator_alvo))-
>asOrderedSet()

}


/*
* Consulta da diretriz 3.1 para recuperar as tarefas i* relacionadas a
* recursos informativos de um ator
*/
query ISTAR::Element::diretriz_3_1_tarefas_rec_info_ator(in
ator_alvo:ISTAR::Actor):OrderedSet(ISTAR::Element){

var elem_servicos:OrderedSet(ISTAR::Element);
 ator_alvo.elements->forEach(element){
if element.is_tarefa() and element.tarefa_afeta_ator(ator_alvo)then{
elem_servicos+=element;
}endif;
};

this.elementos_modelo->forEach(element){
if element.is_recursoinformativo() and
element.is_relacionado_ao_ator(ator_alvo)then{
elem_servicos+=element;
}endif;
};
 return  elem_servicos;
}

/* Consulta que recupera os elementos i* para implementar a diretriz 3.4*/
```

```
query ISTAR::Element::diretriz_3_4_tarefas_rec_fisico(in
ator_alvo:ISTAR::Actor) :OrderedSet(ISTAR::Element) {

var elem_servicos:OrderedSet(ISTAR::Element);
var  conta_rec_fis :Integer:=0;
this.atores_modelo->forEach(ator){
      ator.elements->forEach(element){
      if element.is_tarefa() and element.tarefa_afeta_recursos_fisicos()-
>size()>=1
            and element.ElementopertenceAtor(ator_alvo) then{
            elem_servicos+=element;
      }endif;
      };
};

this.elementos_modelo->forEach(element){
if element.is_tarefa() and element.tarefa_afeta_recursos_fisicos()-
>size()>=1
      and element.ElementopertenceAtor(ator_alvo) then{
            elem_servicos+=element;
      }endif;
};

this.atores_modelo->forEach(ator){
      ator.elements->forEach(element){
      if element.is_tarefa()and element.ElementopertenceAtor(ator_alvo)
then{
            element.tarefa_afeta_recursos_informativos()->forEach(recinfo){
             this.recursos_fisicos->forEach(recfis){
              if recinfo.is_relacionado_ao_recurso_fisico(recfis)then{
              conta_rec_fis:=conta_rec_fis+1;
               }endif;
            };

            if conta_rec_fis>0 then{
               elem_servicos+=element;
               conta_rec_fis:=0;
            }endif;
            };

      }endif;

      };
};

this.elementos_modelo->forEach(element){

      if element.is_tarefa()and element.ElementopertenceAtor(ator_alvo)
then{
            element.tarefa_afeta_recursos_informativos()->forEach(recinfo){
             this.recursos_fisicos->forEach(recfis){
              if recinfo.is_relacionado_ao_recurso_fisico(recfis)then{
              conta_rec_fis:=conta_rec_fis+1;
               }endif;
            };

            if conta_rec_fis>0 then{
               elem_servicos+=element;
               conta_rec_fis:=0;
            }endif;
            };
```

```
      }endif;

};
return elem_servicos;

}


/*
* Consulta que recupera os elementos i* para aplicar a diretriz 3.5
* Uma tarefa que não afeta recursos é transformada em um transação da
*classe gerada a partir do ator que contém a tarefa.
*/
query ISTAR::Element::diretriz_3_5_tarefas_internas_ator(in
ator_alvo:ISTAR::Actor):OrderedSet(ISTAR::Element){

var elem_servicos:OrderedSet(ISTAR::Element);
ator_alvo.elements->forEach(element){
if element.is_tarefa() and self.ElementopertenceAtor() and not
(self.tarefa_afeta_recursoinformativo())then{
elem_servicos+=element;
}endif;
};
return  elem_servicos;
}

/*
* Consultas necessárias para geração das classes
*/
query ISTAR::Element::tarefa_afeta_recursoinformativo(in
recurso_alvo:ISTAR::Element) : Boolean {

return self.type->('TASK')and recurso_alvo.stereotype->includes('related')
}

query ISTAR::Element::tarefa_afeta_recursos_informativos() :
OrderedSet(ISTAR::Element) {

var rec_info_afetados:OrderedSet(ISTAR::Element);

if self.is_tarefa() then{

      this.recursos_fisicos->forEach(element){
      if self.is_relacionado_ao_recurso_fisico(element)then{
  rec_info_afetados+=element;
      }endif;
      };
}endif;

return rec_info_afetados;

}

query ISTAR::Element::tarefa_afeta_recursos_fisicos() :
OrderedSet(ISTAR::Element) {

var rec_fisicos_afetados:OrderedSet(ISTAR::Element);

if self.is_tarefa() then{
```

```
      this.recursos_fisicos->forEach(element){
      if self.is_relacionado_ao_recurso_fisico(element)then{
  rec_fisicos_afetados+=element;
      }endif;
      };
}endif;

return rec_fisicos_afetados;

}

query ISTAR::Element::tarefa_afeta_ator(in ator_alvo:ISTAR::Actor) :
Boolean {
      return self.type->('TASK')and ator_alvo.stereotype-
>includes('related')
}


/*
* Consultas para obter recursos físicos
*/
query ISTAR::Model::getRecursosFisicos() : OrderedSet(ISTAR::Element) {
      return self.elements[ElementType::RESOURCE]-
>select(is_recursofisico())->asOrderedSet();
var rec_fisicos:OrderedSet(ISTAR::Element);
rec_fisicos+= OrderedSet{object ISTAR::Element{name:='Work
Request';type:=ElementType::RESOURCE},
object ISTAR::Element{name:='Proceedings
Manual';type:=ElementType::RESOURCE},
object ISTAR::Element{name:='Refused  work
request';type:=ElementType::RESOURCE},
object ISTAR::Element{name:='Accepted work
request';type:=ElementType::RESOURCE},
object ISTAR::Element{name:='Level';type:=ElementType::RESOURCE}
};
return      rec_fisicos;
}

/*
* Consultas para obter recursos informacionais
*/
query ISTAR::Model::getRecursosInfo() : OrderedSet(ISTAR::Element) {
var elem_atributos:OrderedSet(ISTAR::Element);
//Poderia tb ser implementado como:
// self.elements[ElementType::RESOURCE] - getRecursosFisicos()
this.atores_modelo->forEach(ator){
      ator.elements->forEach(element){
if element.is_recursoinformativo()then{
            elem_atributos+=element;
      }endif;
      };
};

this.elementos_modelo->forEach(element){
if element.is_recursoinformativo() then{
elem_atributos+=element;
}endif;
};
 return  elem_atributos;
}
```

```
/* Consulta  para gerar atributos de um recurso físico*/
query ISTAR::Element::inferir_atributos_recurso_fisico(in
recurso_alvo:ISTAR::Element) : OrderedSet(ISTAR::Element){

var elem_atributos:OrderedSet(ISTAR::Element);
var ator: ISTAR::Actor;
this.atores_modelo->forEach(ator){
      ator.elements->forEach(element){
      if element.is_recursoinformativo() and
element.is_relacionado_ao_recurso_fisico(recurso_alvo)then{
            elem_atributos+=element;
      }endif;
      };
};

this.elementos_modelo->forEach(element){
if element.is_recursoinformativo() and
element.is_relacionado_ao_recurso_fisico(recurso_alvo)then{
elem_atributos+=element;
}endif;
};
 return  elem_atributos;
}


/*
* Consulta para recuperar todos serviços relacionados a um recurso físico
*
*/
query ISTAR::Element::inferir_servicos_recurso_fisico(in
recurso_alvo:ISTAR::Element) : OrderedSet(ISTAR::Element){


            return  self.diretriz_3_2_tarefas_rec_fisico(in
recurso_alvo:ISTAR::Element)->
                  union(self.diretriz_3_3_tarefas_rec_fisico(in
recurso_alvo:ISTAR::Element))->
                  union(self.diretriz_3_6_tarefas_rec_fisico(in
recurso_alvo:ISTAR::Element))->
                  union(self.diretriz_3_7_tarefas_rec_fisico(in
recurso_alvo:ISTAR::Element))->asOrderedSet()

}

/* Consulta que recupera elementos i* p/implementar a diretriz 3.2 */
query ISTAR::Element::diretriz_3_2_tarefas_rec_fisico(in
recurso_alvo:ISTAR::Element) :OrderedSet(ISTAR::Element) {

var elem_servicos:OrderedSet(ISTAR::Element);


 var ator: ISTAR::Actor;
this.atores_modelo->forEach(ator){
      ator.elements->forEach(element){
      if element.tarefa_afeta_recursoinformativo(recurso_alvo) then{
            elem_servicos+=element;
      }endif;
      };
};
```

```
this.elementos_modelo->forEach(element){
if element.tarefa_afeta_recursoinformativo(recurso_alvo) then{
            elem_servicos+=element;
      }endif;
};
 return  elem_servicos;
}


/* Consulta que recupera elementos i* p/implementar a diretriz 3.3*/
query ISTAR::Element::diretriz_3_3_tarefas_rec_fisico(in
recurso_alvo:ISTAR::Element) :OrderedSet(ISTAR::Element) {

var rec_fisicos_afetados:OrderedSet(ISTAR::Element);
var elem_servicos:OrderedSet(ISTAR::Element);

this.atores_modelo->forEach(ator){
      ator.elements->forEach(element){
      if element.is_tarefa() and element.tarefa_afeta_recursos_fisicos()-
>size()=1 then{
            elem_servicos+=element;
      }endif;
      };
};

this.elementos_modelo->forEach(element){
if element.is_tarefa() and element.tarefa_afeta_recursos_fisicos()-
>size()=1 then{
            elem_servicos+=element;
      }endif;
};

return elem_servicos;
}

/* Consulta que recupera elementos i* p/ implementar a diretriz 3.6 */
query ISTAR::Element::diretriz_3_6_tarefas_rec_fisico(in
recurso_alvo:ISTAR::Element) :OrderedSet(ISTAR::Element) {

var gdl:GDL;
var elem_servicos:OrderedSet(ISTAR::Element);

this.gdls->forEach(gdl){
if gdl.is_SD and gdl.Em.oclIsKindOf(ISTAR::ElementType::RESOURCE)
      and gdl.Em.is_recursofisico() and gdl.Em.name=recurso_alvo.name then{

elem_servicos+=gdl.Ei;
elem_servicos+=gdl.Ef;
}endif;
};
return elem_servicos
}

/* Consulta que retornaria as tarefas relacionadas aos recursos físicos que
*seriam transformadas em serviços atômicos OO-M,ou seja, serviços de
*criação conforme diretriz 3.7
* Não dá para implementar ao menos que modifique o metamodelo istar para
* inserir mais um atributo de Tarefa, por exemplo, tipo_tarefa, para dizer
* se é de criação ou não. Está apenas especificada e retornando null.
*/
```

```
query ISTAR::Element::diretriz_3_7_tarefas_rec_fisico(in
recurso_alvo:ISTAR::Element) :OrderedSet(ISTAR::Element) {
return null
}

/* Consulta para saber se uma tarefa tem subtarefas*/
query ISTAR::Element::tarefa_tem_subtarefas() : Boolean {
return self.tarefa_recupera_subtarefas()->notEmpty();
}

/* consulta que recupera todas subtarefas de uma tarefa */
query ISTAR::Element::tarefa_recupera_subtarefas() :
OrderedSet(ISTAR::Element) {
var subtarefas:OrderedSet(ISTAR::Element);
this.atores_modelo->forEach(ator){
ator.decompositionsTask->forEach(td){
if self.is_tarefa() and td.source.name=self.name then{
 if td.target.is_tarefa()then{
 subtarefas+=td.target;
 }endif;
}endif;
};
};
return subtarefas
}

/* consulta que recupera os recursos informativos de uma tarefa */
query ISTAR::Element::tarefa_recupera_recursoinformativo() :
OrderedSet(ISTAR::Element) {
var recinfo:OrderedSet(ISTAR::Element);
this.atores_modelo->forEach(ator){
ator.decompositionsTask->forEach(td){
if self.is_tarefa() and td.source.name=self.name then{
 if td.target.is_recursoinformativo() then{
 recinfo+=td.target;
 }endif;
}endif;
};
};
return recinfo
}

/*
* consulta que retorna os relacionamentos ISA entre atores para serem
* transformadas em associações OO-M.
*/
query ISTAR::Model::getRelacionamentos_ISA():
OrderedSet(ISTAR::ActorAssociation){
return self.associations[ISTAR::ActorAssociationType::ISA]->asOrderedSet()
}

/*
* Obtém Tuplas de recursos físicos do modelo ISTAR onde é possível inferir
* Herança para transformar em herança OO-M
* Está query e mapeamento, segundo Diretriz 4.2, depende de intervenção
* do analista de requisitos, por isso apenas sua assinatura foi
* especificada, retornando null.
*/
query ISTAR::Model::getHerancaRecFisicos() :Sequence(TuplaRecfis_RecFis) {
return null
}
```

```
/*
* Consulta que obtem o conjunto ordenado de objetos
* Generic Dependency link do tipo recurso ou tarefa para serem
* transformados em Associações OO-M
*/
query ISTAR::Model::getRelacionamentosGDLs_4_3: OrderedSet(GDL){

var gdls_4_3:OrderedSet(GDL);
this.gdls->forEach(gdl_temp){
      if
gdl_temp.oclAsType(ISTAR::Element).type.oclIsKindOf(ISTAR::ElementType::TASK) or

gdl_temp.oclAsType(ISTAR::Element).type.oclIsKindOf(ISTAR::ElementType::RESOURCE)
        then{
             gdls_4_3+=gdl_temp;
             }endif;
};
return  gdls_4_3;
}

/*
* Consulta que une os conjuntos de objetos tuplas Recurso Fisico x Ator
* para serem transformados em Asssociações OO-M
*/
query ISTAR::Model::getTuplasRecFis_Ator():OrderedSet(TuplaRecFis_Ator){
return self.getTuplasRecFis_Ator_4_4()->
                  union(self.getTuplasRecFis_Ator_4_5())->
                  union(self.getTuplasRecFis_Ator_4_6())->asOrderedSet()
}

/* Consulta que retorna as tupas da diretriz 4.4 */
query ISTAR::Model::getTuplasRecFis_Ator_4_4():
OrderedSet(TuplaRecFis_Ator) {
var recfis_ator1:TuplaRecFis_Ator;
var recfis_ator2:TuplaRecFis_Ator;

var lista_tuplas:OrderedSet(TuplaRecFis_Ator):=null;

 this.gdls->forEach(gdl){
          if gdl.Em.oclAsType(ISTAR::Element).is_recursofisico()
            and gdl.Em.oclAsType(ISTAR::Element).resolveone(OOM::Class)->notEmpty() then{
             recfis_ator1.rf:=gdl.Em;
             recfis_ator1.at:=gdl.AEi;
             recfis_ator2.rf:=gdl.Em;
             recfis_ator2.at:=gdl.AEf;
             lista_tuplas+=recfis_ator1;
             lista_tuplas+=recfis_ator2;
             }endif;
             };

 return lista_tuplas;

}

/* Consulta que retorna as tupas da diretriz 4.5 */
query ISTAR::Model::getTuplasRecFis_Ator_4_5():
OrderedSet(TuplaRecFis_Ator) {
```

```
var recfis_ator1:TuplaRecFis_Ator;
var recfis_ator2:TuplaRecFis_Ator;
var lista_classes_rec_fis: Sequence(OOM::Class);
var classe_do_rec_info:OOM::Class;
var lista_atributos_classe:Sequence(OOM::Attribute);
var lista_tuplas:OrderedSet(TuplaRecFis_Ator):=null;

 this.gdls->forEach(gdl){
          if gdl.Em.oclAsType(ISTAR::Element).is_recursoinformativo()
            and gdl.AEi.resolveone(OOM::Class)->notEmpty()
            and gdl.AEf.resolveone(OOM::Class)->notEmpty() then{


      lista_classes_rec_fis:=gdl.Em.oclAsType(ISTAR::Element).resolve(OOM::
Class);
            lista_classes_rec_fis->forEach(classe){
            classe.attributes->forEach(atrib){
                  if atrib.name=gdl.Em.oclAsType(ISTAR::Element).name then{
                  classe_do_rec_info:=classe;
                  break;
                  }endif;
            };
            };


      recfis_ator1.rf:=classe_do_rec_info.invresolve(ISTAR::Element::recurs
ofisico2classe,OOM::Class);
            recfis_ator1.at:=gdl.AEi;

      recfis_ator2.rf:=classe_do_rec_info.invresolve(ISTAR::Element::recurs
ofisico2classe,OOM::Class);
            recfis_ator2.at:=gdl.AEf;
            lista_tuplas+=recfis_ator1;
            lista_tuplas+=recfis_ator2;
            }endif;
            };

 return lista_tuplas;


}

/* Consulta que retorna as tupas da diretriz 4.6 */

query
ISTAR::Model::getTuplasRecFis_Ator_4_6():OrderedSet(TuplaRecFis_Ator){

var recfis_ator:TuplaRecFis_Ator;
var lista_tuplas:OrderedSet(TuplaRecFis_Ator):=null;

 this.atores_modelo->forEach(ator){
      ator.elements->[ElementType::RESOURCE]->forEach(elem_res){
            if
elem_res.resolveone(ISTAR::Element::recursofisico2classe(),OOM::Class)-
>notEmpty() then{
            recfis_ator.rf:=elemen.res;
            recfis_ator.at:=ator;
            lista_tuplas+=recfis_ator;
            }endif;
            };
      };
```

```
 return lista_tuplas;

}


/*
* Consulta que monta todas Generic Dependency Link para ser atribuída
* a this.gdls. Este conjunto de GDLs é fundamental para processamento
* das Diretrizes que geram os relacionamentos
*/
query ISTAR::Model::monta_todas_GDLs(): OrderedSet(GDL){

var gdl:GDL;
var gdls:OrderedSet(GDL);
var Dlo:=ISTAR::DependencyLink;
var Dld:=ISTAR::DependencyLink;
var dl:=ISTAR::DependencyLink;
self.links->forEach(dl){
 Dlo:=dl;
 self.links->forOne(dl|dl<>Dlo){
       if dl.source.name=Dlo.target.name then{
            Dld:=dl;
       }endif;
      }
      }

if Dlo.source.type.oclIsKindOf(ISTAR::Actor) and
Dld.target.type.oclIsKindOf(ISTAR::Actor) then{

 gdl.Ei:=Dlo.source;
 gdl.Em:=Dlo.target;
 gdl.Ef:=Dld.target;
 gdl.AEi:=gdl.Ei;
 gdl.AEf:=gdl.Ef;
 gdl.is_SD:=true;
 gdls+=gdl;
}else{
 gdl.is_SD:=false;
 gdl.Ei:=Dlo.source;
 gdl.Em:=Dlo.target;
 gdl.Ef:=Dld.target;
 gdl.AEi:=AQualAtorpertenceElemento(Dlo.source,self.actors);
 gdl.AEf:=AQualAtorpertenceElemento(Dld.target,self.actors);
}endif;
return gdls;
}

/* Consultas de Apoio*/
query AQualAtorpertenceElemento(in e:ISTAR::Element):ISTAR::Actor{

var ator:ISTAR::Actor:=null;
var ator_achado:ISTAR::Actor:=null;
var elem:ISTAR::Element:=null;

this.atores_modelo->forEach(ator){
 ator.elements->forOne (elem| elem.name=e.name){
 ator_achado:=ator;
};
};
return ator_achado;
```

```
}

query ISTAR::Element::ElementopertenceAtor(in a:ISTAR::Actor):Boolean {

var pertence:Boolean:=false;
 a.elements->forOne (elem| elem.name=a.name){
 pertence:=true;
};
return pertence;
}

/*Consulta que geram objetos tuplas recurso físico X recurso fisico
* para mapeamento especificado na diretriz 4.7
*/
query
ISTAR::Model::getTuplasRecFis_RecFis():OrderedSet(TuplaRecfis_RecFis){

var recfis_recfis:TuplaRecfis_RecFis;
var lista_tuplas:OrderedSet(TuplaRecfis_RecFis):=null;
var gdl:GDL;
var gdl_transformadas:OrderedSet(GDL);

 gdl_transformadas:=gdl.resolve(GDL::GDL2Association():OOM::Association);
      gdl_transformadas->forEach(gdl_temp){
            recfis_recfis.rfo:=gdl_temp.Ei;
            recfis_recfis.rfd:=gdl_temp.Ef;
            lista_tuplas+=recfis_recfis;
            };

 return lista_tuplas

}


/*
* Consulta que gera as tuplas AtorXTarefa para serem mapeadas
* em AgentLink OO-Method a partir da Diretriz 4.8
*/
query
ISTAR::Model::getTuplasAgentLink4_8():OrderedSet(TuplaAtor_Tarefa_AgentLink
){
//Só vai com resolve
 var ator_tarefa:TuplaAtor_Tarefa_AgentLink;
 var lista_tuplas:OrderedSet(TuplaAtor_Tarefa_AgentLink):=null;
 var gdl:GDL;
 var gdl_transformadas:OrderedSet(GDL);

 gdl_transformadas:gdl.resolve(GDL2Association():OOM::Association);
      gdl_transformadas->forEach(gdl_temp){
            if gdl_temp.Em.resolveone(task_ator2service(): OOM::Service)-
>notEmpty() then{
            ator_tarefa.ator_depender:=gdl_temp.AEi;
            ator_tarefa.tarefa_recfis_ator_dependee:=gdl_temp.Ef;
            lista_tuplas+=ator_tarefa;
            }endif;

            };

return lista_tuplas
}
```

```
/*
* Consulta que gera as tuplas AtorXTarefa para serem mapeadas
* em AgentLink OO-Method a partir da Diretriz 4.9
*/
query
ISTAR::Model::getTuplasAgentLink4_9():OrderedSet(TuplaAtor_Tarefa_AgentLink
){

var ator_tarefa:TuplaAtor_Tarefa_AgentLink;
var lista_tuplas:OrderedSet(TuplaAtor_Tarefa_AgentLink):=null;

 this.atores_modelo->forEach(ator){
      ator.elements->[ElementType::TASK]->forEach(elem_task){
            if element_task.resolveone(task_recursofisico2service():
OOM::Service)->notEmpty() then{
            ator_tarefa.ator_depender:=ator;
            ator_tarefa.tarefa_recfis_ator_dependee:=elem_task;
            lista_tuplas+=ator_tarefa;
            }endif;
            };
      };

 return lista_tuplas
}

/* Consulta que retorna conjunto de recursos informativos de uma Task
*  a partir de sua decomposição para serem transformados em parametros
*/
query ISTAR::Element::getParameters():OrderedSet(ISTAR::Element) {
return self.tarefa_recupera_recursoinformativo()

}

/* FIM da IMPLEMENTAÇÃO QVT do Módulo Tradutor que automatiza o PROCESSO
NÚCLEO do OOM-NFR */
```