

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

HEYDE FRANCIELLE DO CARMO FRANÇA

**Uma solução baseada em ontologia
para a prevenção de erros comuns em
modelos de requisitos escritos na
linguagem i***

Goiânia
2016

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

**AUTORIZAÇÃO PARA PUBLICAÇÃO DE DISSERTAÇÃO
EM FORMATO ELETRÔNICO**

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos “reproduzir” e “publicar” conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: Uma solução baseada em ontologia para a prevenção de erros comuns em modelos de requisitos escritos na linguagem i*

Autor(a): Heyde Francielle do Carmo França

Goiânia, 14 de Março de 2016.

Heyde Francielle do Carmo França – Autor

Dr. Renato de Freitas Bulcão Neto – Orientador

HEYDE FRANCIELLE DO CARMO FRANÇA

Uma solução baseada em ontologia para a prevenção de erros comuns em modelos de requisitos escritos na linguagem i*

Dissertação apresentada ao Programa de Pós-Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Renato de Freitas Bulcão Neto

Goiânia
2016

Agradecimentos

Meus sinceros agradecimentos a todos, que direta ou indiretamente, contribuíram para a conclusão deste trabalho. Especialmente minha família, pelo amor, paciência e apoio não só nesta fase mas em todos os momentos da minha vida. Agradeço ao Adriano por estar presente na maior parte desta batalha, com seu amor e companheirismo, nunca deixando que eu desanimasse. E a todos os meus amigos que entenderam todas as vezes que faltei a compromissos sociais para estudar.

Agradeço de coração aos amigos que fiz durante o período do mestrado e que levarei para a vida toda, em especial a Carina Calixto, Edjalma Queiroz e Ernesto Fonseca é muito bom poder contar com a ajuda de vocês, contem sempre comigo também.

Agradeço imensamente ao meu orientador por todo o apoio, paciência, e acima de tudo pelo suporte acadêmico, técnico e psicológico.

Resumo

França, Heyde Francielle do Carmo. **Uma solução baseada em ontologia para a prevenção de erros comuns em modelos de requisitos escritos na linguagem i***. Goiânia, 2016. 86p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

A abordagem de Engenharia de Requisitos Orientada a Metas (do Inglês, GORE) representa as necessidades dos usuários através de metas e intenções, focando em capturar a real intenção dos *stakeholders*. Baseada na técnica GORE, a linguagem de modelagem i* representa metas do sistema e da organização e traz diversas vantagens. Apesar disso, a linguagem i* apresenta problemas relacionados à qualidade dos modelos, que incluem erros típicos de mau uso dos construtores, à presença de ambiguidades na interpretação dos construtores e à complexidade dos modelos resultantes. Assim, o objetivo desta dissertação é apresentar uma solução baseada em ontologia visando a redução de erros comuns em modelos de requisitos construídos na linguagem i*. Para atingir tal objetivo foi realizada inicialmente uma pesquisa bibliográfica, seguida de uma pesquisa experimental para produzir a solução proposta. Esta solução foi implementada realizando a extensão de um ontologia chamada *OntoiStar+*, na qual foram inseridas restrições na linguagem OWL para garantir que os erros frequentes de modelos i* não sejam reproduzidos. Foi realizada também a extensão da ferramenta *TAGOO+* para validação de modelos i* escritos em *iStarML* e conversão para modelos em OWL. Para realização dos testes foram modelados dois domínios diferentes, o *Media Shop* e um sobre universidades, usando estes domínios foram reproduzidos estudos de casos e mensurados os resultados. Os testes realizados em ambas soluções geraram resultados satisfatórios. Os resultados demonstraram uma cobertura de mais de 70% dos erros mais comuns com a extensão da *OntoiStar+* e mais de 80% com a extensão da ferramenta *TAGOO+*.

Palavras-chave

Engenharia de Requisitos Orientada a Metas, Linguagem i*, Istar, Ontologia, OntoiStar, Experimentação.

Abstract

França, Heyde Francielle do Carmo. **An ontology-based solution for prevention of common mistakes in models requirements written in the language i*.** Goiânia, 2016. 86p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

The Goal Oriented Requirements Engineering (GORE) approach represents users' needs through goals with focus on capturing the real intentions of stakeholders. Based on the GORE technique, the i* modeling language represents system's and organization's goals and brings several advantages. Despite that, the i* language faces problems regarding the quality of models, which include typical mistakes of misuse of i* constructs, the presence of ambiguities on the interpretation of those constructs, and the complexity of the resulting i* models. The aim of this work is to present an ontology-based solution for i* models in order to reduce the most well-known errors while constructing such models. To achieve this goal was accomplished initially a literature search, followed by an experimental research to produce the proposed solution. This solution includes the extension of an ontology called OntoiStar+ with OWL restrictions to ensure that frequent mistakes in i* models are not found. Besides, the TAGOOn+ tool was also extended to validate i* models in the iStarML language and convert those to an OWL representation. To perform the tests were modeled two different domains, Media Shop and on universities, using these domains case studies have been reproduced and measured results. Results demonstrate an approximate coverage of 70% of those common errors with extension of OntoiStar+ and more than 80% with extension of TAGOOn+ tool.

Keywords

Goal Oriented Requirements Engineering, Language i*, Istar, Ontology, OntoiStar, Experimentation.

Sumário

Lista de Figuras	7
Lista de Tabelas	9
Lista de Siglas	10
1 Introdução	11
1.1 Contextualização	11
1.2 Motivação	12
1.3 Problema	13
1.4 Objetivo	14
1.5 Materiais e Métodos	14
1.6 Estrutura da Dissertação	15
2 Fundamentação Teórica	16
2.1 Engenharia de Requisitos Orientada a Metas	16
2.2 Linguagem i*	17
2.2.1 Modelo SD	20
2.2.2 Modelo SR	23
2.2.3 Vantagens	26
2.2.4 Erros Comuns em Modelos i*	26
2.3 Ontologias	28
2.4 Considerações Finais	31
3 Trabalhos Relacionados	33
3.1 <i>iStarML</i>	33
3.2 <i>IStar Tool</i>	35
3.3 AIRDoc-i*	36
3.4 OntoiStar+	39
3.5 TAGOOn+	42
3.6 Considerações Finais	43
4 Extensão da ontologia OntoiStar+ e da ferramenta TAGOOn+	45
4.1 Extensão da Ontologia <i>OntoiStar+</i>	45
4.1.1 Tratamento dos Erros	46
4.1.2 Resultados Obtidos	61
4.2 Estendendo a Ferramenta <i>TAGOOn+</i>	62
4.2.1 Inserindo as Regras no <i>TAGOOn+</i>	63
4.2.2 Testes Realizados	65

4.2.3	Resultados Obtidos	70
5	Conclusões e Trabalhos Futuros	71
5.1	Limitações	72
5.2	Trabalhos Futuros	72
5.3	Considerações Finais	72
5.4	Publicações Relacionadas	72
	Referências Bibliográficas	74
A	Catálogo de erros típicos de modelos i*	78
B	Arquivos utilizados para testes da ferramenta <i>TAGOO_{n++}</i>	84

Lista de Figuras

2.1	Principais construtores da linguagem i* [44]	18
2.2	Exemplo de modelagem do domínio <i>Media Shop</i> utilizando a linguagem i* [33]	19
2.3	Tipos de atores descritos na linguagem i* [44]	21
2.4	Dependência de <i>Task</i> [44]	22
2.5	Dependência de <i>Resource</i> [44]	22
2.6	Dependência de <i>Goal</i> [44]	22
2.7	Dependência de <i>Softgoal</i> [44]	23
2.8	Fronteira de um ator [44]	23
2.9	Representação de um ligação do tipo <i>means-end</i> [44]	24
2.10	Exemplo de um relacionamento de decomposição [44]	25
2.11	Passos para desenvolver uma ontologia [31]	29
2.12	Exemplo de utilização da Cardinalidade Mínima [27]	31
3.1	Relacionamento de dependência de meta em <i>iStarML</i>	35
3.2	Relacionamento de dependência de meta em <i>iStarML</i>	36
3.3	O processo <i>AIRDoc-i*</i>	38
3.4	Arquitetura de Desenvolvimento da <i>OntoiStar</i> [21]	39
3.5	Representação gráfica de uma parte da <i>OntoiStar+</i> [21]	41
3.6	<i>TAGOOOn+</i> :Interface Gráfica [21]	42
3.7	<i>TAGOOOn+</i> :Fluxo do processo de transformação [21]	43
4.1	Comparação entre as modelagens correta e incorreta da fronteira de ator[33]	47
4.2	Mensagem de erro: Não permite mais de um ator dentro da fronteira. Fonte: Elaborada pelo autor,2016	48
4.3	Restrição <i>OWL</i> : Trecho que não permite mais de um ator dentro da fronteira. Fonte: Elaborada pelo autor,2016	48
4.4	Mensagem de erro: Não permite mais de um ator dentro da fronteira. Fonte: Elaborada pelo autor,2016	50
4.5	Mensagem de erro: Não permite que haja relacionamentos do tipo <i>InternalElementRelationship</i> para representar uma dependência entre um <i>Actor</i> e um <i>InternalElement</i> . Fonte: Elaborada pelo autor,2016	51
4.6	Mensagem de erro: Não permite que uma meta possa ser ligada diretamente a outra meta. Fonte: Elaborada pelo autor,2016	53
4.7	Mensagem de erro: Não permite que um <i>Depender</i> se origine na fronteira de um ator. Fonte: Elaborada pelo autor,2016	54
4.8	Mensagem de erro: Não permite que um <i>Dependee</i> e um <i>Depender</i> se conectem diretamente a outro ator. Fonte: Elaborada pelo autor,2016	55

4.9	Mensagem de erro: Não permite utilizar um <i>ContributionLink</i> para conectar um <i>IntentionalElement</i> que não seja uma <i>Softgoal</i> . Fonte: Elaborada pelo autor,2016	56
4.10	Mensagem de erro: Não é possível utilizar o <i>MeansEndLink</i> para conectar um <i>InternalElement</i> que não seja uma <i>Task</i> a uma <i>Goal</i> . Fonte: Elaborada pelo autor,2016	57
4.11	Mensagem de erro: Só pode ser utilizado o <i>Occupies</i> para ligar um <i>Agent</i> a um <i>Position</i> . Fonte: Elaborada pelo autor,2016	59
4.12	Mensagem de erro: só pode ser utilizado o <i>Plays</i> para ligar um <i>Agente</i> a um <i>Role</i> . Fonte: Elaborada pelo autor,2016	59
4.13	Mensagem de erro: Só pode ser utilizado o <i>Occupies</i> para ligar um <i>Position</i> a um <i>Role</i> . Fonte: Elaborada pelo autor,2016	60
4.14	Mensagem de erro: Só pode ser utilizado o <i>Occupies</i> para ligar um <i>Actor</i> a um outro <i>Actor</i> . Fonte: Elaborada pelo autor,2016	60
4.15	Mensagem de erro: Só pode ser utilizado o <i>InstanceOf</i> para ligar um <i>Agent</i> a um outro <i>Agente</i> . Fonte: Elaborada pelo autor,2016	61
4.16	<i>TAGOOOn+</i> :Tela do projeto <i>TAGOOOn+</i> [21]	64
4.17	Exemplo de arquivo em <i>iStarML</i> sem erro[21]. Fonte: Elaborada pelo autor,2016	66
4.18	Exemplo do funcionamento do <i>TAGOOOn+</i> . Fonte: Elaborada pelo autor,2016	67
4.19	Exemplo de arquivo em <i>iStarML</i> com erro. Fonte: Elaborada pelo autor,2016	68
4.20	Tela do <i>TAGOOOn</i> exibindo erro. Fonte: Elaborada pelo autor,2016	69
4.21	Exemplo de arquivo em <i>iStarML</i> com erro. Fonte: Elaborada pelo autor,2016	69
4.22	Tela do <i>TAGOOOn+</i> exibindo erro. Fonte: Elaborada pelo autor,2016	70
A.1	Dangling Actor	78
A.2	Internal Actor	78
A.3	Actor Boundary	79
A.4	Actor Dependency	79
A.5	Softgoal	79
A.6	Goal Refinement	80
A.7	Internal Dependency Link	80
A.8	External Dependency Link	81
A.9	Internal Contribution Link	81
A.10	External Contribution Link	82
A.11	Internal Means-End Link	82
A.12	External Means-End Link	83
A.13	External Decomposition Link	83
B.1	Modelo usando dependências (SD)	84
B.2	Modelo usando <i>ielement</i> e <i>ielementLink</i> (SR)	85
B.3	Modelo usando dependências e construtores do Tropos	85
B.4	Modelo usando <i>ielement</i> e <i>ielementLink</i> e construtores do Tropos	86
B.5	Dependência de Serviço no modelo global	86

Lista de Tabelas

3.1	Tags complementares do <i>iStarML</i>	34
3.2	Quadro comparativo entre Trabalhos Relacionados	44
4.1	Condições para que não exista conexão entre <i>InternalElement</i> e <i>Actor Boundary</i> Fonte: Elaborada pelo autor,2016	49
4.2	Condições para que não haja relacionamentos do tipo <i>InternalElementRelationship</i> para representar uma dependência entre um <i>Actor</i> e um <i>InternalElement</i> . Fonte: Elaborada pelo autor,2016	50
4.3	Condições para que uma meta só possa ser decomposta utilizando ligações do tipo <i>MeansEndLink</i> e que só possa ser decomposta em <i>tasks</i> . Fonte: Elaborada pelo autor,2016	52
4.4	Regra para certificar-se que um <i>Depender</i> não se origine na fronteira de um ator. Fonte: Elaborada pelo autor,2016	53
4.5	Regras para certificar-se que um <i>Dependee</i> e um <i>Depender</i> não se conectem diretamente a outro ator. Fonte: Elaborada pelo autor,2016	54
4.6	Restrições para certificar-se que um <i>Dependee</i> e um <i>Depender</i> não se conectem diretamente a outro ator. Fonte: Elaborada pelo autor,2016	55
4.7	Regras inseridas para garantir que um <i>MeansEndLink</i> só conecte <i>Task</i> a <i>Goal</i> . Fonte: Elaborada pelo autor,2016	56
4.8	Restrições inseridas para garantir que as ligações entre atores estejam corretas. Fonte: Elaborada pelo autor,2016	58

Lista de Siglas

AIRDOC-i*	<i>Approach to Improve Requirements Documents for i*</i>
B2C	<i>Business to Customer</i>
BPMN	<i>Business Process Modeling Notation</i>
ER	<i>Engenharia de Requisitos</i>
GMF	<i>Graphical Modelling Framework</i>
GORE	<i>Goal Oriented Requirements Engineering</i>
GQM	<i>Goal Question Metric</i>
MDE	<i>Model Driven Engineering</i>
OWL	<i>Web Ontology Language</i>
RDF	<i>Resource Description Framework</i>
SD	<i>Strategic Dependency Model</i>
SR	<i>Strategic Rationale Model</i>
UFO	<i>Unified Foundational Ontology</i>
UML	<i>Unified Modeling Language</i>
W3C	<i>World Wide Web Consortium</i>
XML	<i>Extensible Markup Language</i>

Introdução

Este capítulo apresenta inicialmente a contextualização e motivação para a realização deste trabalho de pesquisa. Em seguida, descrevem-se o problema de pesquisa encontrado e a proposta de solução para tratá-lo. Faz-se posteriormente uma descrição resumida da metodologia empregada neste trabalho, assim como dos principais resultados obtidos. Por fim, apresenta-se a organização em capítulos desta dissertação de mestrado.

1.1 Contextualização

A Engenharia de Requisitos (ER) é uma fase no desenvolvimento de software na qual é essencial que sejam entendidas e modeladas as necessidades dos usuários, bem como características e particularidades do software a ser desenvolvido. Esta pode ser considerada como a fase mais crítica do processo de desenvolvimento de sistemas. Isso, porque, para construção de um software com qualidade, é preciso que as considerações técnicas estejam em equilíbrio com as sociais e organizacionais desde a modelagem dos sistemas [32].

Um entendimento completo dos requisitos de software é essencial para o sucesso do desenvolvimento do software. Porém, a maioria das técnicas de Engenharia de Requisitos concentra maior parte dos esforços apenas na modelagem e especificação do software, não dando a devida importância para a lógica sobre a composição do sistema, que engloba além do próprio sistema a ser construído, o ambiente da organização que o utilizará.

Suposições incorretas a respeito do ambiente de um sistema podem ser responsáveis por uma grande quantidade de erros na especificação dos requisitos [1]. É necessária muita atenção nas fases iniciais de elicitação de requisitos, como acontece no caso da abordagem GORE (do inglês *Goal Oriented Requirements Engineering*) [41].

A engenharia de requisitos pode representar as necessidades do usuário através de metas e intenções, abordagem esta utilizada na técnica GORE. Essa técnica tem como foco capturar as reais intenções dos *stakeholders*, por meio de metas que estes desejam satisfazer, fornecendo uma maneira natural para estruturar documentos de requisitos com-

plexos. Essas metas fornecem um mecanismo para justificar a existência dos requisitos e facilitar a administração de conflitos entre requisitos [33].

A grande diferença entre as abordagens de engenharia de requisitos tradicionais e a abordagem GORE, é que esta segunda abordagem foca principalmente nas fases iniciais da eliciação de requisitos, com o objetivo de entender o contexto da organização e as metas dos interessados. Explicitando através dessas metas a existência de cada um dos requisitos. A forma de representação destes requisitos também é um pouco diferente, já que na abordagem GORE são apresentados atores organizacionais e relacionamentos estratégicos que estes devem possuir para atingir as suas metas.

A abordagem GORE considera a organização e as metas dos atores como a origem dos requisitos funcionais e não-funcionais, por isso é necessário primeiro obter os objetivos, a fim de elicitar os requisitos. Além de modelar os requisitos do sistema, como nas abordagens tradicionais de engenharia de requisitos, a abordagem GORE modela também as metas, que são a razão porque são necessários requisitos [43].

1.2 Motivação

No contexto da engenharia de requisitos orientada a metas, surgiram diversas abordagens que utilizam essa abstração, dentre elas a *V-Graph* [45], *NFR-Framework* [8], *KAOS* [42] e a linguagem *i** [23], sendo esta última a mais difundida na literatura [9] [12].

A linguagem *i** possui uma estrutura de modelagem conceitual que possibilita identificar motivações, intenções e raciocínios sobre as características de um processo, o que facilita os esforços nas atividades da Engenharia de Requisitos. Por apresentar várias vantagens e ser a mais difundida na comunidade de engenharia de requisitos a *i** foi a linguagem escolhida como objeto de estudo nesta pesquisa.

Na linguagem *i** vários tipos de atores são definidos para modelar situações onde um ator depende de outro para que suas metas sejam alcançadas, tarefas sejam executadas ou recursos sejam fornecidos. O modelo gerado não contempla somente as metas do sistema, mas também os da organização que existem em torno do mesmo [44]. Um modelo gerado utilizando a linguagem *i** descreve os participantes do processo e o contexto organizacional, tornando assim mais fácil visualizar a intenção por trás de cada requisito.

A linguagem *i** se caracteriza por modelar objetivos e as intenções dos atores e suas dependências dentro de um contexto organizacional, permitindo descrever os relacionamentos estratégicos e intencionais em alto nível de detalhamento, trazendo benefícios na identificação e manutenção de requisitos e aumentando a rastreabilidade no caso de mudanças de objetivo.

1.3 Problema

Através de uma revisão bibliográfica sobre a engenharia de requisitos orientada a metas, mais especificamente da linguagem i^* , foram identificados algumas vulnerabilidades que podem prejudicar a qualidade dos modelos gerados. Apesar das vantagens apresentada pela linguagem i^* , vários autores apontam problemas quanto à qualidade dos modelos produzidos nessa linguagem, como: erros típicos de mau uso de construtores, a presença de ambiguidades na interpretação dos construtores e a complexidade dos modelos resultantes [33], [37], [19], [15]. Em suma, os principais erros encontrados em modelos de requisitos utilizando a linguagem i^* envolvem a definição de atores, dependências, relacionamentos entre atores, dentre outros.

O mau uso dos construtores da linguagem normalmente acontece devido a erros de interpretação da sintaxe e semântica do i^* [20]. Para o sucesso dos modelos gerados utilizando a linguagem i^* não basta que sejam bem detalhados, é necessário que estes modelos estejam corretos quanto à sintaxe da linguagem [23]. Para facilitar a identificação desses erros, que podem acontecer em modelos utilizando a linguagem i^* em [33], é descrito um catálogo de erros frequentes da linguagem i^* .

Uma solução viável para esclarecer o significado de cada construtor da linguagem, e assim reduzir erros de interpretação, seria a utilização de uma ontologia. Ontologia é uma descrição conceitual, formal e explícita de um domínio de conhecimento, segundo [3], sendo utilizada dentre outras finalidades, para a criação de um vocabulário consensual e livre de ambiguidades em vários domínios de aplicação.

Ontologias vêm sendo amplamente utilizadas em diversos domínios de aplicação. Por exemplo, em [20] a ontologia UFO (*Unified Foundational Ontology*) é utilizada para alinhar metas e processos de negócios. A ontologia *OntoiStar+* [21] foi criada com o objetivo de representar os conceitos fundamentais da linguagem i^* e os relacionamentos entre esses conceitos.

Foram propostos na literatura trabalhos que exploram a sintaxe e a semântica da linguagem i^* , como a linguagem *iStarML* [14], baseada na linguagem XML, e a ontologia *OntoiStar+* [21], sob a perspectiva da semântica dos construtores. Integrando com a *OntoiStar+* foi desenvolvida a ferramenta *TAGOOOn+* que realiza a conversão de modelos de requisitos utilizando i^* modelados em *iStarML* para modelos em OWL (*Web Ontology Language*) [21]. Porém, nenhuma das soluções trata os erros frequentes, nem mesmo a solução que utiliza uma ontologia para descrever os construtores da linguagem i^* resolve erros típicos em modelos de requisitos i^* , como erros envolvidos na definição de atores, dependências, relacionamentos entre atores, dentre outros.

Percebe-se, então, a necessidade de realizar a extensão desta ontologia com o objetivo de desenvolver uma solução ontológica que trate os erros frequentes da

linguagem i^* . A *OntoiStar+* foi construída utilizando a linguagem OWL, o que permite que sejam acrescentados restrições em sua estrutura para especificar a semântica de cada construtor da i^* .

1.4 Objetivo

O objetivo geral deste trabalho é apresentar uma solução baseada em ontologia para o tratamento de erros comuns em modelos de requisitos construídos na linguagem i^* . Visando atingir este objetivo será reutilizada a ontologia *OntoiStar+*, na qual serão incluídas restrições a respeito da utilização dos construtores que frequentemente são empregados de forma incorreta nos modelos.

Pensando em agregar essa validação dos erros em modelo i^* em ferramentas de modelagem, serão incluídas estas regras na ferramenta *TAGOO+*. Estas regras irão possibilitar a validação dos modelos i^* durante a conversão de modelos em *iStarML* para modelos em OWL.

1.5 Materiais e Métodos

Para possibilitar a redução dos erros frequentes da linguagem i^* e garantir uma maior qualidade nos modelos de requisitos gerados utilizando essa linguagem, foi desenvolvida uma solução baseada em ontologia para o tratamento dos erros de interpretação e mau uso dos construtores da linguagem i^* . A solução foi executada seguindo os seguintes passos:

- Estudo da linguagem i^* para entender seus construtores e verificar quais os erros frequentes relatados na literatura;
- Estudo das ontologias já desenvolvidas, como a UFO e a *OntoiStar+*, e verificação da possibilidade de reutilização;
- Elaboração das restrições OWL para evitar os erros de interpretação e mau uso dos construtores da linguagem i^* e integração dessas regras na ontologia *OntoiStar+*;
- Validação da proposta: após a inserção de cada restrição na ontologia *OntoiStar+* foram realizados testes para verificar se é possível criar um modelo de requisitos com cada erro tratado. Os testes foram realizados utilizando o software *Protégé* [21] e o raciocinador Pellet [34] para apontar os erros. Foram desenvolvidos modelos de testes que descrevem o domínio do *Media Shop* e o de universidades. A escolha do *Media Shop* como domínio para realização dos testes foi devido ao fato de existir vários artigos com exemplos de sua modelagem de requisitos utilizando a linguagem i^* e por este motivo fica evidente os erros que podem acontecer [33],[32], [6],

[7] e o de universidades por já ter sido apresentado por [21] . Como a solução propõe uma redução dos erros de má utilização dos construtores de i^* , o domínio escolhido tem pouca influência nos resultados, já que não serão analisados erros de modelagem relativos a interpretação do domínio e sim quanto a utilização correta dos construtores da linguagem i^* .

- Inserção das restrições na ferramenta *TAGOOOn+* [21] seguindo as regras inseridas na ontologia *OntoiStar+* , para possibilitar a validação de modelos em *iStarML* [5].

1.6 Estrutura da Dissertação

O restante do trabalho está organizado conforme a seguinte estrutura:

1. O capítulo 2 apresenta os conceitos fundamentais de engenharia de requisitos, engenharia de requisitos orientada à metas, ontologia e da linguagem i^* e os erros frequentes em modelos gerados utilizando a linguagem i^* , conhecimentos estes necessários para a contextualização da pesquisa.
2. O capítulo 3 que compara e descreve trabalhos relacionados a esta pesquisa.
3. O capítulo 4 apresenta a primeira parte da solução proposta, mostrando o detalhamento da extensão da ontologia *OntoiStar+*, bem como da ferramenta *TAGOOOn+* e a realização dos testes modelando o domínio *Media Shop* e o de universidades para validação.
4. O capítulo 5 apresenta as conclusões finais a cerca do trabalho apresentado, limitações da proposta e considerações para o desenvolvimento de trabalhos futuros.
5. O apêndice A - Catálogo com os erros mais frequentes no uso da linguagem i^* , referência importante utilizada no desenvolvimento dessa dissertação.
6. E por fim, encontra-se o apêndice B - que apresenta os arquivos em *iStarML* utilizados para validação das alterações realizadas na ferramenta *TAGOOOn+*.

Fundamentação Teórica

Este capítulo reúne os conceitos que fundamentam o trabalho realizado nesta pesquisa, iniciando com a teoria de Engenharia de Requisitos Orientada a Metas. Em seguida, apresenta-se a linguagem de modelagem i^* quanto a sua gramática, notação e os principais erros encontrados em modelos de requisitos utilizando essa linguagem. Por fim, descrevem-se conceitos sobre ontologias, sua importância, uma metodologia de desenvolvimento, bem como a linguagem *OWL* para construção de ontologias.

2.1 Engenharia de Requisitos Orientada a Metas

A engenharia de requisitos corresponde à atividade de entendimento das necessidades dos usuários no contexto do problema a ser resolvido [36]. Ao representar as necessidades do usuário através de metas e intenções, a abordagem *Goal Oriented Requirements Engineering* (GORE) [41] captura as reais intenções dos *stakeholders* por meio de metas que eles pretendem satisfazer. Esta abordagem provê uma forma natural para estruturar documentos de requisitos complexos, através de metas que fornecem um mecanismo para justificar a existência de requisitos e facilitar a administração de conflitos entre requisitos [33].

O característica importante da engenharia de requisitos é a necessidade de compreender o domínio do problema, a fim de formular o modelo de requisitos, compreendendo objetivos, pressupostos e requisitos de domínio. Ao considerar que ambiente e contexto são estáticos, estes poderiam ser entendidos suficientemente bem, de forma a permitir que o modelo de requisitos formulado para a solução seja confiável.

Na prática, contexto e ambiente raramente são estáticos ao longo do tempo, o que dificulta a compreensão dos requisitos. No entanto, a engenharia de requisitos oferece uma gama de técnicas capazes de mitigar ou evitar esses problemas, quando a mudança de contexto acontece de forma lenta o suficiente para permitir que os desenvolvedores avaliem as implicações e tomem as medidas adequadas. Cada vez mais, no entanto, os sistemas estão se tornando muito dinâmicos e estão sujeitos a alterações em períodos curtos e de maneiras que aumentam a dificuldade de compreensão dos requisitos[2].

A engenharia de requisitos lida com elicitação, análise, comunicação e validação de requisitos. Quanto mais cedo os erros forem identificados e corrigidos mais fácil e mais barato será a correção se comparado com uma identificação tardia.

A engenharia de requisitos orientada a metas é focada nas atividades que antecedem a formulação dos requisitos do sistema. As atividades principais que compõem a abordagem GORE são: elicitação, refinamento e análise de metas, e a atribuição de responsabilidade das metas para os agentes. A GORE vê o sistema a ser construído e o seu ambiente como um conjunto de componentes ativos chamados agentes [1].

A modelagem de metas tem como base um modelo de organização humana, que enxerga as pessoas e as organizações como entidades em busca de uma meta ou objetivo. Existem algumas definições para a palavra meta dentro do contexto de engenharia de requisitos: meta é um objetivo de alto nível da empresa, organização ou sistema, ou razão pela qual é necessário um sistema, e guia as decisões por vários níveis dentro da empresa. Meta é definida como um ponto ou objetivo a ser atingido em determinada medida e prazo. Enquanto o objetivo apenas explicita o propósito, intenção ou fim que se deseja alcançar, a meta quantifica e define um prazo, ou seja, uma meta é um objetivo quantificado a ser atingido dentro de um prazo especificado [25].

Segundo [42], uma meta é um objetivo que um sistema deve satisfazer. Pode ser vista também como uma condição a ser alcançada [45]. Qualquer que seja o conceito, as metas podem ser formuladas em diferentes níveis de abstração, variando desde: alto nível, estratégicas, baixo nível e técnicas. Essas abstrações podem ser funcionais, associadas aos serviços a serem prestados, e não funcionais, quando associadas com a qualidade do serviço, tais como: segurança, precisão e desempenho.

Utilizando a abordagem orientada a metas é possível explicitar as metas organizacionais ou *goals* dos *stakeholders*, para que então sejam refinados e extraídos os requisitos funcionais e não funcionais que farão parte do sistema. Portanto, os requisitos funcionais irão representar os requisitos de sistema e os não funcionais serão representados por *softgoals*, um tipo de objetivo que representará algum atributo de qualidade do sistema [41].

Esses elementos e a forma de utilizar os relacionamentos entre eles variam dependendo da abordagem GORE empregada. Existem várias abordagens orientadas a metas, tais como: *NFR Framework*, *Tropos*, *KAOS*, *i** e *GBRAM*. Neste trabalho será utilizada a linguagem *i**.

2.2 Linguagem i*

A linguagem *i** foi desenvolvida com o intuito de poder ser aplicada para modelagem e análise. Para realizar a análise organizacional, os engenheiros de requisitos

modelam os *stakeholders* como atores e suas intenções como metas. Os atores são entidades que possuem metas e podem depender de outros atores para conseguir alcançá-las [22], [45].

O conjunto de dependências criadas pelos atores formam um grafo que representa o ambiente do sistema e o sistema em si. A estrutura conceitual do i* é utilizada para obter uma compreensão mais apurada sobre os relacionamentos da organização, de acordo com os diversos atores do sistema. Além disso, o i* permite compreender as razões envolvidas nos processos de decisão.

Modelos i* detalham os participantes do processo e o contexto que está sendo modelado, o que facilita seu entendimento [22]. A modelagem de *softgoals* traz vantagens como integrá-las aos processos das organizações modeladas e ainda estabelecer os responsáveis por atendê-las [43].

A linguagem cobre as abstrações de tarefas, metas e *softgoals* (traduzidas neste trabalho como meta-flexível), além disso apresenta também abstrações como atores e dependências que permitem analisar os impactos do software na organização. Segue, na Figura 2.1 a ilustração destes construtores básicos da linguagem i*.

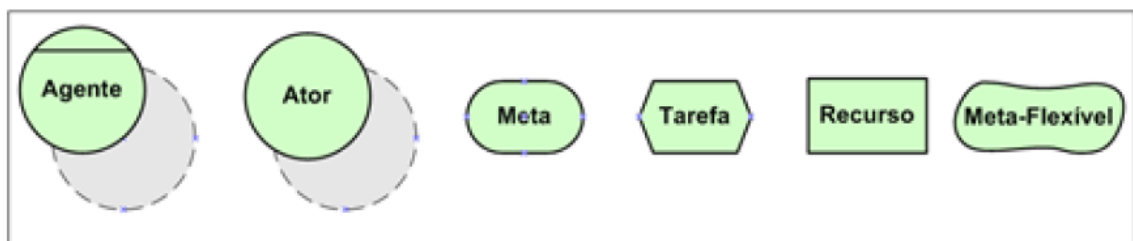


Figura 2.1: Principais construtores da linguagem i* [44]

A contribuição da linguagem i* para a área de engenharia de requisitos está na capacidade de permitir identificar explicitamente motivações, ao modelar o que leva um determinado agente a realizar uma tarefa ou objetivo.

A linguagem i* também facilita o processo de desenvolvimento do sistema mesmo que aconteçam modificações de requisitos ao longo deste, uma vez que permite facilmente identificar e modelar alternativas para os novos requisitos que surgem durante a fase de desenvolvimento do sistema [45]. A linguagem i* é formada por dois modelos: o modelo SD (*Strategic Dependency model*) e o modelo SR (*Strategic Rationale model*).

O modelo SD fornece uma descrição intencional de um processo em termos de uma rede de relacionamentos de dependência entre atores relevantes, ou seja, as relações de dependências externas entre os atores da organização. O modelo SR apresenta uma descrição estratégica do processo, em termos de elementos do processo e das razões que motivam a existência deles.

Segue um exemplo de modelo de requisitos utilizando a linguagem i*, representando o domínio do *Media Shop*. A Figura 2.2 é baseada no exemplo *Media Shop*

adaptado do original [7]. O *Media Shop* é uma loja de venda de variados tipos de itens como: livros, jornais, revistas, CDs, etc. Os Clientes podem usar um catálogo para fazer pedidos, periodicamente atualizado, descrevendo os itens disponíveis. Existe um ator chamado Fornecedor que é responsável pelo abastecimento dos itens do *Media Shop*. Para aumentar a participação no mercado, foi criado também um B2C (*Business to Customer*) para proporcionar vendas pela Internet. Com essa nova opção, o consumidor pode pedir itens pessoalmente, por telefone, ou através da Internet. O sistema foi chamado Medi@ e está disponível na Internet. O objetivo básico do sistema é permitir ao Cliente examinar o catálogo *on-line* e fazer pedidos. Os Clientes em potencial podem pesquisar a loja *on-line* navegando no catálogo ou através de consultas na base de dados de itens.

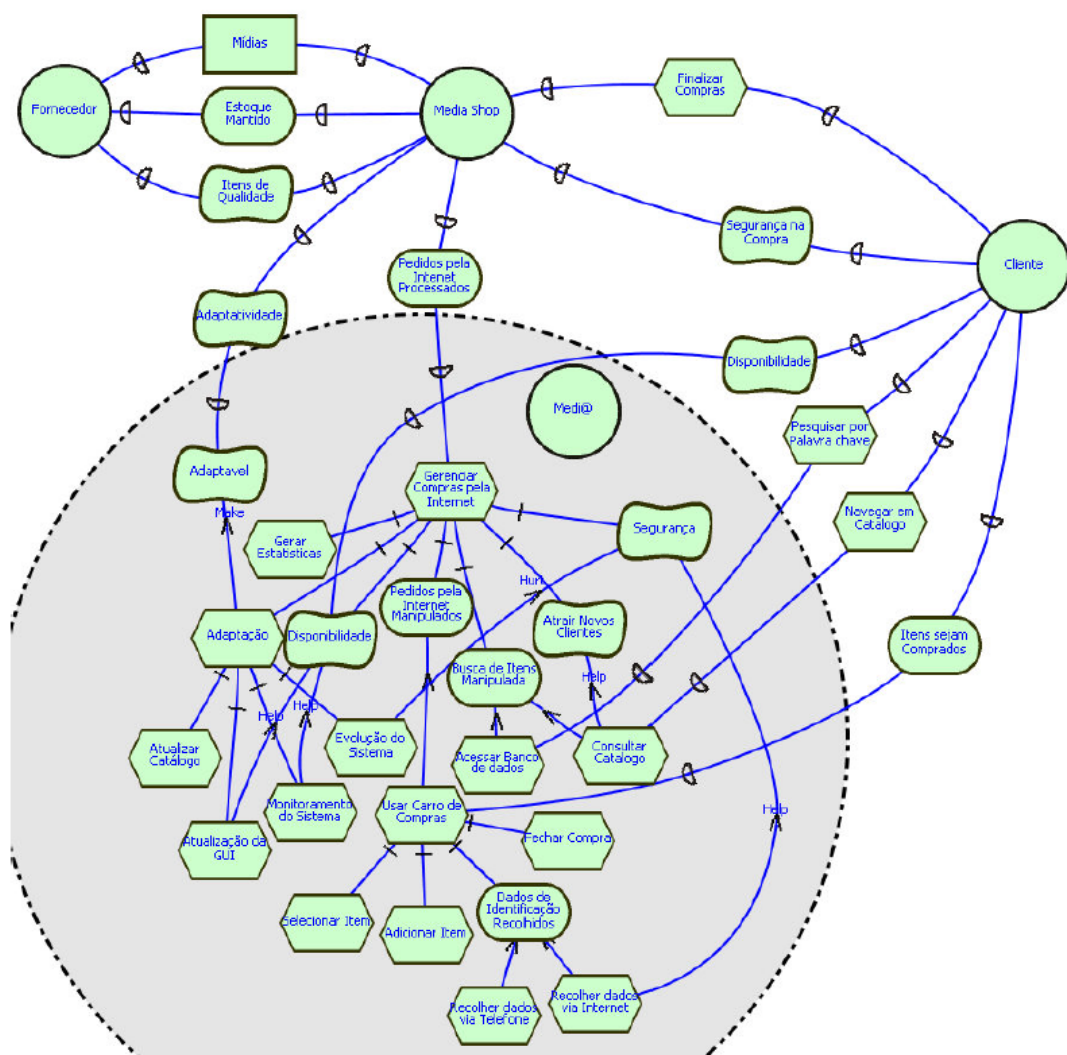


Figura 2.2: Exemplo de modelagem do domínio Media Shop utilizando a linguagem i* [33]

2.2.1 Modelo SD

No modelo SD são capturadas as motivações e os desejos dos atores que fazem parte da organização além de apresentar a rede de relacionamentos do ator. Dado um modelo SD pode-se perguntar que novos relacionamentos entre os atores são possíveis; identificar a viabilidade ou não das dependências; ou ainda relacionar os desejos de um agente com as habilidades do agente do qual depende, para poder explorar as oportunidades disponíveis a esses atores. Modelos SD descrevem as dependências entre os atores, mas não as razões que as motivam.

No modelo SD um agente pode depender de outro para satisfazer uma meta, para executar uma tarefa, fornecer um recurso ou satisfazer uma *softgoal* [26]. As *softgoals* estão relacionadas aos requisitos não funcionais, enquanto as metas, tarefas e recursos estão relacionadas com funcionalidades do sistema [45].

O termo Ator é utilizado para referenciar genericamente qualquer unidade para a qual dependências intencionais possam ser atribuídas. Atores podem ser considerados: intencionais, por possuírem motivações, desejos e razões por trás de suas ações; e estratégicos, quando não focam apenas o seu objetivo imediato, mas quando se preocupam com as implicações de seu relacionamento estrutural com outros atores. Os atores podem ser diferenciados em três tipos especializados de atores: agente, papel e posição.

- Agente (*Agent*): é um ator que possui manifestações físicas concretas. Tanto pode referir-se a humanos quanto a agentes artificiais de software ou hardware;
- Papel (*Role*): representa a caracterização abstrata do comportamento social 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 (*Position*): 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.

Os atores possuem relacionamentos que permitem organizar os mapeamentos desejáveis entre eles. Os relacionamentos são de especialização (*IsA*), agregação (*IsPartOf*), cobertura (*Covers*), atuação (*Plays*) e ocupação (*Occupies*). Quando essa distinção é feita, a análise torna-se mais detalhada e exige uma atenção maior.

- *IsPartOf*: Nessa associação cada papel, posição e agente pode ter sub-partes.
- *IsA*: Essa associação representa uma generalização, com um ator sendo um caso especializado de outro ator. Ambas, *IsA* e *IsPartOf*, 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.
- *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, 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.

Segue na Figura 2.3 um exemplo da utilização correta destes relacionamentos com os tipos de atores descritos pela linguagem i*.

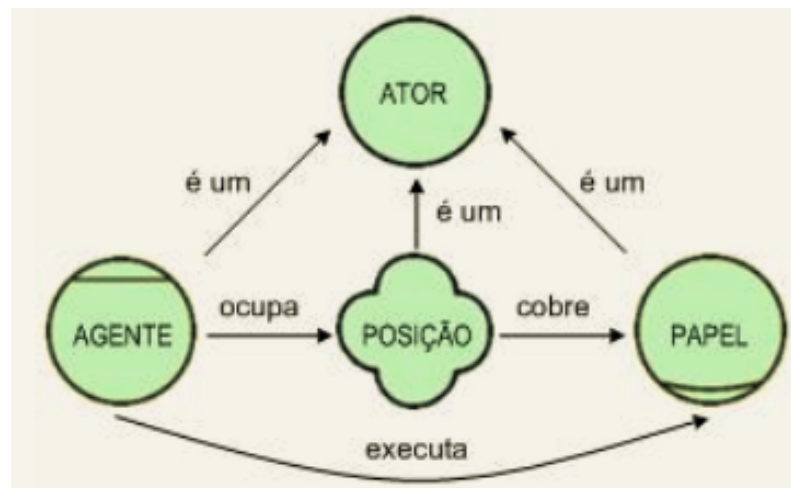


Figura 2.3: Tipos de atores descritos na linguagem i* [44]

Cada dependência em i* é um relacionamento intencional, ou seja, um acordo chamado *dependum*, entre dois atores: o *dependor* e o *dependee*. O *dependor* é o ator que depende de um outro ator (*dependee*) para que o acordo (*dependum*) possa ser realizado. O *dependor* tem metas que não sabe como alcançá-las, não tem recursos para, ou simplesmente não quer realizar, e repassa essa necessidade para outro (o ator *dependee*) que tem a habilidade ou os recursos necessários para isso.

Os relacionamentos de dependência usados em i* descrevem a natureza do acordo e podem ser de quatro tipos: tarefas, recursos, metas ou *softgoals*. Um ator *dependee* satisfaz uma dependência quando disponibilizar o recurso necessário, executar a tarefa que foi requisitada, atender a uma meta, ou satisfazer um *softgoal* do ator *dependor*.

Na dependência de tarefa (*Task*), o *dependee* é requisitado a executar uma dada atividade, sendo informado sobre o que deve ser feito. Contudo, a descrição de uma tarefa em i* não tem por intenção ser uma completa especificação dos passos necessários à execução dessa tarefa, nem há preocupação em se informar o porquê da solicitação de sua realização. Uma tarefa especifica uma forma particular de se fazer algo, elas podem

ser vistas como soluções que provêem operações, processos, representações de dados, estruturação, restrições e meios para atender às necessidades estabelecidas nas metas e *softgoals*. Na Figura 2.4 pode ser observado um exemplo de dependência de tarefa, onde o ator **Cliente** depende do ator **Medi@** para realizar a tarefa **Fazer Pedido**.



Figura 2.4: Dependência de Task [44]

Na dependência de recurso (*Resource*), o ator depende da disponibilidade de uma entidade física ou de uma informação. Por recurso entendemos o produto final de alguma ação, em um processo, que estaria ou não disponível para o ator dependente. Nesse tipo de dependência assume-se que não haja aspectos pendentes a serem tratados ou decisões a serem tomadas. Na Figura 2.5 pode ser observado um exemplo de dependência de recurso, onde o ator **Media Shop** depende do ator **Fornecedor Midia** para fornecer o recurso **Midia**.

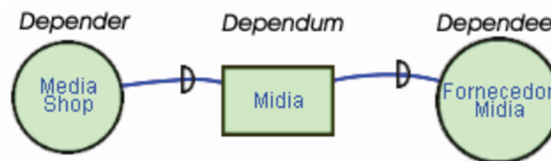


Figura 2.5: Dependência de Resource [44]

Na dependência de meta (*Goal*), um ator depende de outro para realizar uma meta. O *dependee* é livre para tomar as decisões necessárias para satisfazer uma meta, não importando a maneira como haverá satisfação. Na Figura 2.6 pode ser observado um exemplo de dependência de meta, onde o ator **Cliente** depende do ator **Media Shop** para realizar a meta **Comprar itens de Midia**.



Figura 2.6: Dependência de Goal [44]

Uma *softgoal* especifica uma condição ou estado do mundo que um ator gostaria de alcançar, mas que a satisfação não pode ser definida a priori como verdadeira ou falsa, de forma que é o objeto de interpretação ou negociação por parte dos *stakeholders*. Uma dependência de *softgoal* representa uma qualidade que está associada a uma meta,

assim ela pode estar relacionada aos requisitos não funcionais que se deseja que o sistema atenda. Na Figura 2.7 pode ser observado um exemplo de dependência de *softgoal*, onde o ator **Cliente** depende do ator **Bank Cty** para realizar a *softgoal* **Aprovação de Cadastro**.

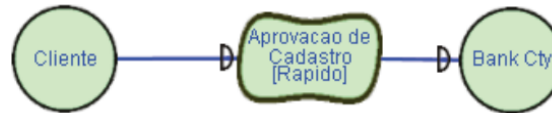


Figura 2.7: Dependência de Softgoal [44]

O analista pode escolher de acordo com a necessidade um destes tipos de dependência para modelar o contexto do projeto. Cada caso tem um propósito e interpretação conforme foi apresentado nos exemplos.

2.2.2 Modelo SR

Enquanto o modelo SD trata apenas dos relacionamentos externos entre os atores, o modelo SR é utilizado para descrever os 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 por trás das dependências entre os atores.

As fronteiras (*Boundary*) dos atores indicam os relacionamentos intencionais de um ator. Todos os elementos dentro da fronteira de um ator são explicitamente desejados pelo ator. Para alcançar estas metas, frequentemente um ator pode depender de intenções de outros atores, representado por uma ligação de dependência atravessando a fronteira de atores [16]. Na Figura 2.8 é ilustrada uma fronteira de ator, percebe-se que internamente a fronteira de um ator não pode existir outro ator, todos os itens internos a fronteira devem contribuir para realizar as intenções daquele ator em questão.

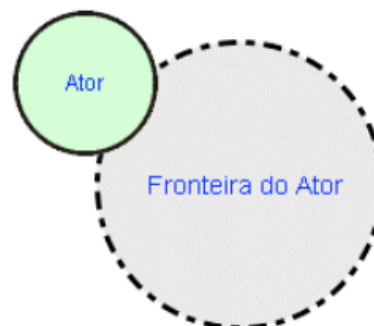


Figura 2.8: Fronteira de um ator [44]

No modelo SR aparecem elementos intencionais para descrever a análise de oportunidades e vulnerabilidades e o tratamento das dependências entre os atores. Os

elementos intencionais podem ser metas, tarefas, recursos, bem como outros tipos de relacionamentos como tipo *means-end* (relação meio-fim, que indica o tipo de meio (*means*) para atingir determinada meta), *decomposition task* e *contribution* [44]. A representação de uma ligação do tipo *means-end* é representada na Figura 2.9, neste exemplo pode-se perceber o uso de uma ligação meio-fim, onde o objetivo do ator **Media Shop** de **Tratar Pedidos dos Clientes** pode ser alcançado através da tarefa **Pedidos Por Telefone**, ou pela tarefa **Pedidos Pela Internet**, ou pela tarefa **Pedidos Na Loja**.

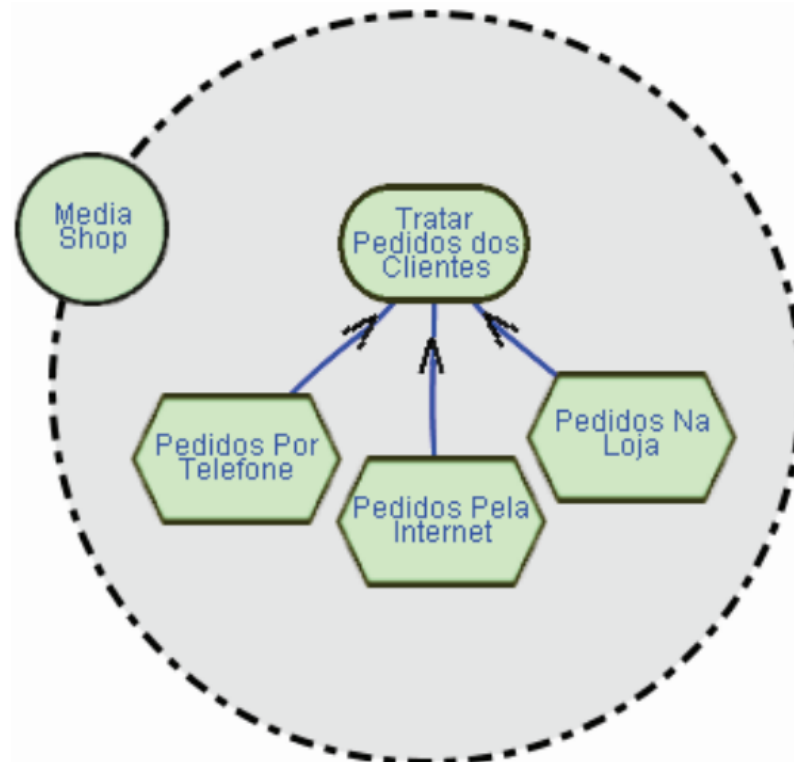


Figura 2.9: Representação de um ligação do tipo *means-end* [44]

Um relacionamento de decomposição-tarefa existe entre uma tarefa e suas partes, mostrando como uma tarefa é executada. Um exemplo de relacionamento de decomposição segue na Figura 2.10, em que pode-se verificar a decomposição da tarefa **Gerenciar Loja da Internet**. A tarefa é inicialmente refinada nas metas **Tratar Pesquisa de Item**, **Tratar Pedidos da Internet**, *softgoal* **Atrair Novos Clientes** e na Tarefa **Produzir Estatísticas**.

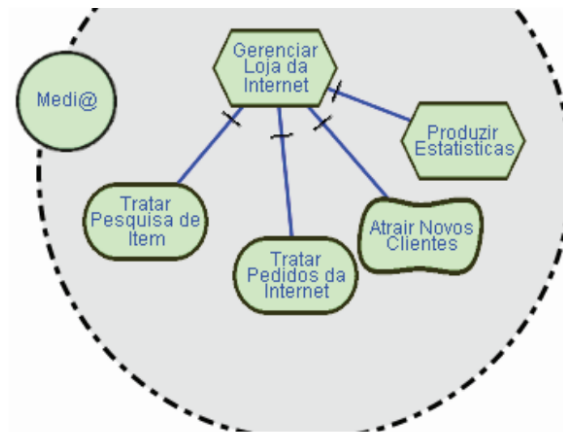


Figura 2.10: Exemplo de um relacionamento de decomposição [44]

- Decomposição Tarefa-Meta (Sub-meta): Neste tipo de decomposição não é especificado como uma é alcançada, permitindo considerar alternativas;
- Decomposição Tarefa-Tarefa (Sub-tarefa): Quando se especifica uma tarefa como um subcomponente de uma tarefa maior;
- Decomposição Tarefa-Recurso: A entidade representada por um recurso normalmente não é considerada um problema por um ator, a principal preocupação é se o recurso estará disponível quando necessário;
- Decomposição de Tarefa- *Softgoal*: pode ser utilizada como um meta de qualidade para a tarefa .

O modelo SR também é composto por nós e ligações que, juntos fornecem uma estrutura para expressar as razões envolvidas no processo. Neste modelo são utilizados quatro tipos de nós, que se baseiam nos tipos de dependências do modelo SD: recurso, tarefa, meta e softgoal.

As *Softgoals* podem ter uma satisfação parcial e serem influenciadas por outros elementos. As ligações de contribuição são representadas por ligações com rótulos que indicam o quanto um elemento ajuda ou prejudica a satisfação de uma softgoal. Os rótulos podem ser: *Make*, *Help*, *Some+*, *Some..*, *Hurt* e *Break* [16]. Esses rótulos representam uma contribuição fortemente positiva (*Make*), parcialmente positiva (*Some+* e *Help*), parcialmente negativa (*Hurt* e *Some..*) e fortemente negativa (*Break*).

Para mais detalhes a respeito dos construtores da linguagem i* e sua correta utilização, sugere-se que seja consultado o Guia da linguagem de modelagem i* usado como referência deste trabalho [24]¹.

¹<http://istar.rwth-aachen.de/tiki-index.php?page=iStarQuickGuide>

2.2.3 Vantagens

Sendo i* uma linguagem bastante versátil os seus principais pontos fortes deve-se ao fato de permitir uma análise tanto ao nível do sistema como do nível de interação com o sistema, por parte tanto dos vários utilizadores do sistema como de outros sistemas externos que necessitam de interação com o sistema a ser desenvolvido.

Ao contrário de outras linguagens de análise e modelagem, i* permite especificar o modo de funcionamento dos vários componentes integrantes de um determinado sistema, assim como explicita o “porquê” dessas diversas funcionalidades terem aquele modo específico de funcionamento.

A linguagem i* possibilita ainda a identificação dos vários requisitos e preocupações expressas pelos principais interessados num determinado sistema, fazendo deste modo uma melhor modelagem, integrando nesta os requisitos funcionais e não funcionais expressos pelos *stakeholders*[44]. Oferece, ainda, várias possibilidades de modelagem de um aspecto específico do sistema, dando a hipótese de analisar as várias opções e escolher a mais correta para aquele caso específico, por meio do uso de um sistema de contribuições positivas e negativas [44].

Por ser um *framework* bastante flexível, vem sendo utilizado em contextos variados, entre eles os mais comuns são [4]:

- **Engenharia de Requisitos:** a linguagem i* tem sido bastante usada nas fases iniciais da modelagem dos requisitos.
- **Modelagem de negócio:** com o objetivo de entender melhor a intencionalidade dos processos de negócio permitindo melhorar o planeamento estratégico.
- **Segurança, Confiabilidade e Privacidade:** a modelagem i* pode ajudar a lidar com elementos de segurança, confiabilidade e privacidade, através do estudo dos conflitos de intenções de diferentes entidades sociais.
- **Desenvolvimento de sistemas auto adaptativos:** Sistemas auto adaptativos[4] como o próprio nome sugere, devem se adaptar a mudanças que podem ocorrer em tempo de execução. A linguagem i* tem se mostrado apropriada nas fases iniciais deste tipo de software, pois permite identificar quais os requisitos têm maiores chances de sofrerem mudanças em tempo de execução, tornando possível considerar soluções alternativas e ainda permite identificar quais serão os outros requisitos que poderão ser impactados com esta mudança.

2.2.4 Erros Comuns em Modelos i*

Sendo i* uma linguagem que se encontra em fase crescente tanto de utilização como de desenvolvimento, é natural que apresente algumas falhas tanto a nível estrutural como ao nível de visualização. Em [4], é ressaltado que o crescente surgimento de

variantes da linguagem i* **prejudica a interoperabilidade**, já que fica difícil saber se os conceitos possuem o mesmo significado.

Da mesma forma que outras técnicas de representação de requisitos apresentam algumas falhas e problemas isso também acontece na linguagem i*, onde revisando a literatura sobre o assunto pode-se destacar alguns erros típicos que se deve ao **mau uso da notação i***.

Em [33], são destacados além de alguns erros típicos de mau uso de construtores, **a presença de ambiguidades e a complexidade dos modelos**. Os principais erros típicos são:

1. Atores e relacionamentos entre atores

- [a] Atores sem ligação
- [b] Atores dentro da fronteira de outros atores
- [c] Uso de nomes inadequados em atores
- [d] Atores com ligação de dependência sem usar um "*dependum*"

2. Dependências

- [a] Uso de outras ligações em vez das ligações de dependências
- [b] Uso inadequado de "*dependums*" (*softgoals* como metas, tarefas como *softgoals*, etc.)
- [c] Formação de ciclos entre "*dependums*" de atores diferentes

3. Elementos internos e relacionamentos entre elementos internos

- [a] Deixar elementos sem ligações
- [b] Uso de ligações em situações irregulares (ligação de dependência dentro da fronteira do ator)
- [c] Elementos do SR fora de fronteira do ator correspondente
- [d] Decompor metas em sub-metas ou sub-tarefas
- [e] Decompor *softgoals* em sub-*softgoals* ou sub-tarefas
- [f] Contribuição para metas, tarefas ou recursos
- [g] *Means-Ends* onde uma meta é um meio
- [h] Ligação direta entre elementos internos de dois atores diferentes

Além destes erros típicos, em [10], é explicitada a existência de ambiguidades no conceito de especialização da linguagem i*. Cuesta em [10], inclusive, propõe uma definição formal para o conjunto de operações de especialização aplicáveis na construção de modelos i*, e ainda uma metodologia de utilização desta solução.

Maiores detalhes sobre os erros frequentes em modelos i* podem ser encontrados no Apêndice A.

2.3 Ontologias

Conforme foi descrito anteriormente, um dos problemas da linguagem i^* é a falta de descrição semântica dos construtores. Sendo assim, ontologias podem ser utilizadas para resolver este tipo de problema, na medida em que elas definem um vocabulário de representação, capturam os conceitos e as relações de um domínio, e um conjunto de axiomas, que restringem a sua interpretação [30].

A ontologia modela o conhecimento de um domínio através de entidades e relacionamentos entre elas. O papel da ontologia neste processo é explicitar o vocabulário utilizado e fornecer um padrão para o compartilhamento da informação [3].

As ontologias são reconhecidas como importantes ferramentas conceituais na Ciência da Computação, desde o final dos anos 60, especialmente nas áreas de modelagem conceitual e inteligência artificial [18], tornando-se bastante utilizada por permitir que se faça a explicitação de conceitos, propriedades, relações, funções, restrições e axiomas claramente definidos, possibilitando criar uma representação de um modelo abstrato de algum fenômeno do mundo real [3].

Uma definição bastante citada na literatura é a que considera uma ontologia como uma especificação formal e explícita de uma conceitualização compartilhada, onde especificação formal quer dizer algo que é legível para os computadores [17]. Os principais elementos de uma ontologia são: conceitos (Classes), hierarquia, propriedades dos conceitos (*slots*, atributos), restrições sobre as propriedades (tipo, cardinalidade, etc), relações entre conceitos (igualdade, disjunção, etc), instâncias de conceitos [3].

Duas das principais características da ontologia são a capacidade para alta *expressividade e estruturação*. A expressividade se refere à extensão e facilidade com as quais uma ontologia pode descrever a semântica de um domínio, ou seja, quão numerosos e detalhados são os relacionamentos entre os seus elementos. A estruturação diz respeito ao grau de extensão organizacional ou hierárquica de uma ontologia, ou seja, quão específico é o nível de representação de um determinado conceito [3].

Algumas das vantagens da utilização de uma ontologia são: interoperabilidade entre aplicações de software, conhecimento modelado independente de implementação, capacidade de reusar e estender conceitos de outras ontologias, deduzir novos fatos a partir de fatos declarados no domínio [31].

Para desenvolver uma ontologia é necessário seguir uma metodologia, como a 101, que reúne os sete passos apresentados na Figura 2.11 e que serão detalhados a seguir [31]:

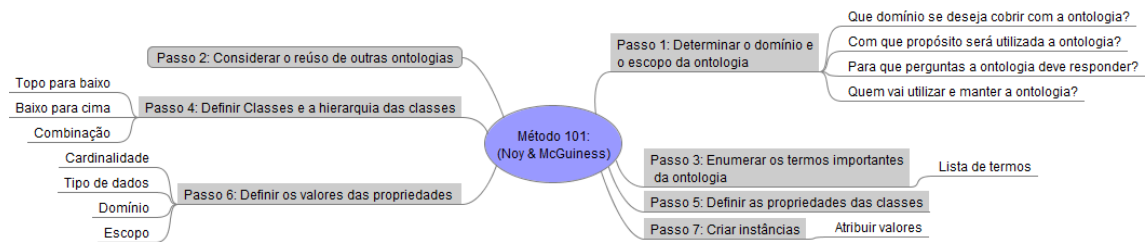


Figura 2.11: Passos para desenvolver uma ontologia [31]

- **Passo 1. Determinar o domínio e abrangência da ontologia:** É Sugerido começar o desenvolvimento de uma ontologia, definindo seu domínio e escopo. Para definir escopo e domínio podem ser utilizadas algumas questões como: Qual o domínio que ontologia irá cobrir? Para que será usada a ontologia? Que tipos de perguntas a ontologia deve fornecer respostas? Quem vai usar e manter a ontologia? Deve-se elaborar também algumas questões de competência, isto é, questões que a ontologia precisa ser capaz de responder.
- **Passo 2. Considerar a reutilização de ontologias existentes:** É importante considerar as ontologias já desenvolvidas e verificar a possibilidade de reutilização.
- **Passo 3. Enumerar termos importantes na ontologia:** Neste passo é recomendado escrever uma lista de todos os termos que pretende-se fazer declarações ou explicitar o significado ao usuário.
- **Passo 4. Definir as classes e hierarquia de classes:** Existem várias abordagens para o desenvolvimento de uma hierarquia de classes, um método sugerido é o *top-Down*, onde primeiro se cria as classes com a definição mais geral dos conceitos e posteriormente se faz a especialização destes conceitos.
- **Passo 5. Definir as propriedades de classes:** Somente as classes não são suficientes para fornecer todas as informações para responder as questões de competência definidas no passo 1. Depois de definir as classes é importante descrever a estrutura interna destes conceitos, definindo as propriedades destas classes.
- **Passo 6. Definir valores das propriedades:** Definir os tipos de valores permitidos, a cardinalidade e outras características das propriedades.
- **Passo 7. Criar instâncias:** O último passo é a criação de instâncias individuais das classes.

A mais recente linguagem de ontologia desenvolvida é a OWL (*Web Ontology Language*), projetada para ser usada por aplicações que necessitem processar o conteúdo de informações, ao invés de somente apresentar a visualização destas informações [35].

O W3C ² recomenda que as pessoas que queiram construir ontologias utilizem a linguagem OWL, com isso, se espera que esta linguagem se torne um padrão.

²O World Wide Web Consortium (W3C) é a principal organização de padronização da World Wide Web

A OWL é uma linguagem para definição e instanciação de ontologias Web. Uma ontologia OWL pode formalizar um domínio, definindo classes e propriedades destas classes, definir indivíduos e afirmações sobre eles e, usando-se a semântica formal OWL, especificar como derivar consequências lógicas, isto é, fatos que não estão presentes na ontologia, mas são vinculados pela semântica [3].

A OWL foi projetada para prover uma linguagem de ontologia que pudesse ser usada para descrever, de um modo natural, classes e relacionamentos entre elas em documentos e aplicações Web. Os termos usados em uma ontologia devem ser escritos de uma maneira que eles possam ser interpretados sem ambiguidade e usados por agentes de software.

Os elementos básicos para construção de uma ontologia OWL são as classes, as instâncias (também chamadas de indivíduos) das classes e os relacionamentos (as propriedades) entre estas instâncias.

- Classes: provêm um mecanismo de abstração para agrupar recursos com características similares, ou seja, uma classe define um grupo de indivíduos que compartilham algumas propriedades. O conjunto de indivíduos que estão associados a uma classe, é chamado de extensão de classe [40].
- Indivíduos: que são instâncias de classes, são definidos com fatos. Um fato é uma declaração que é sempre verdade em um dado domínio. Indivíduos podem ser relacionados usando-se as propriedades da OWL.
- Propriedades: são relações binárias, podem ser usadas para estabelecer relacionamentos entre indivíduos ou entre indivíduos e valores de dados. Estes relacionamentos permitem afirmar fatos gerais sobre os membros das classes e podem também especificar fatos sobre indivíduos. A OWL possui duas categorias principais de propriedades [40]:
 - Propriedades de dados tipados (*datatype properties*): relação entre indivíduos e valores de dados.
 - Propriedades de objetos (*object properties*): relação entre indivíduos.

Propriedades podem ser utilizadas para definir restrições. Em OWL pode-se descrever a classe dos indivíduos que tem pelo menos um, ou no máximo ou exatamente um número específico de relacionamentos com outros indivíduos ou *datatype values* (valores de tipos de dados). Restrições são utilizadas para definir limites para indivíduos que pertencem a uma classe. Podem ser de 3 tipos:

- Restrições que utilizam quantificadores universal e existencial;
- Restrições de cardinalidade(mínima, máxima e exata). Este tipo de restrição foi o mais utilizado neste trabalho. Os tipos de restrição de cardinalidade

são: *maxCardinality*, *minCardinality*, *Cardinality*. As restrições que descrevem essas classes são conhecidas como *Cardinality Restrictions* (Restrições de Cardinalidade). Para uma dada propriedade P, uma *Minimum Cardinality Restriction* (Restrição de Cardinalidade Mínima) especifica o número mínimo de relacionamentos P dos quais um indivíduo deve participar. Uma Restrição Cardinalidade Máxima (*Maximum Cardinality Restriction*) especifica o número máximo de relacionamentos P dos quais um indivíduo pode participar. Uma *Cardinality Restriction* (Restrição de Cardinalidade) especifica o número exato de relacionamentos P dos quais um indivíduo participa. Segue um exemplo na Figura 2.13 de uma restrição do tipo *minCardinality* para facilitar o entendimento. Neste exemplo é apresentada uma restrição de cardinalidade mínima sobre a relação *ensinadoPor* de tal forma que ela deve ocorrer pelo menos uma vez para indivíduos da classe *Disciplina*.

- Restrições que aplicam valores específicos (tipo *hasValue*).

Restrições de cardinalidade

```
<owl:Class rdf:about="#disciplina">
  <rdfs:subClassOf>

    <owl:Restriction>
      <owl:onProperty
        rdf:resource="#ensinadoPor"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
        1
      </owl:minCardinality>
    </owl:Restriction>

  <rdfs:subClassOf>
  </owl:Class>
```

Figura 2.12: Exemplo de utilização da Cardinalidade Mínima [27]

O *Protégé* é uma ferramenta gráfica para edição de ontologias e aquisição de conhecimento [21], que vem sendo usada há algum tempo por especialistas principalmente nas áreas de medicina e indústria para a modelagem de domínios. Esta foi a ferramenta escolhida para ser utilizada nesta pesquisa por suportar a metodologia de implementação de ontologias do Método 101 [31].

2.4 Considerações Finais

Nos últimos anos tem crescido o interesse em criar e/ou utilizar abordagens de engenharia de requisitos orientadas a metas. Neste capítulo foram citadas algumas dessas

abordagens, dando ênfase na linguagem i^* . Através de artigos publicados abordando as características da linguagem i^* , tornou-se óbvia a existência de algumas vulnerabilidades desta linguagem. Para solucionar estes problemas encontrados, uma solução viável é a utilização de uma ontologia para esclarecer o significado de cada construtor da linguagem, bem como a sua utilização correta. Portanto, esta dissertação toma para estudo a linguagem i^* , e a viabilização de um abordagem ontológica utilizando a linguagem OWL para descrever regras, a fim de mitigar os erros documentados neste capítulo.

Trabalhos Relacionados

Este capítulo apresenta conceitos básicos sobre os trabalhos relacionados à proposta deste trabalho. Entre os trabalhos relacionados, podemos citar o *iStarML* [4], *Istar Tool* [28], *AIRDoc-i** [37], *OntoiStar+* [21] e *TAGOOOn+* que serão detalhadas a seguir.

Considerando a grande aceitação da linguagem *i** pela comunidade de engenharia de requisitos e observando que nos últimos anos vêm sendo propostas algumas variantes para a linguagem, percebe-se que existe a necessidade de garantir a interoperabilidade entre variantes da *i**, bem como garantir que os modelos gerados estejam livres dos erros relatados no capítulo anterior. A seguir serão abordados trabalhos que abordam essas duas vertentes, a de integração de variantes do *i** e validação dos modelos gerados. Finalizando com um comparativo entre essas abordagens apresentadas e a solução proposta nesta dissertação.

3.1 *iStarML*

Devido à grande aceitação da linguagem *i**, é fácil encontrar uma grande quantidade de ferramentas que foram criadas com base nos conceitos da *i** ou de variantes dela. A maioria dessas ferramentas possui estruturas e modelos próprios, dificultando o intercâmbio de modelos entre essas ferramentas. Essa incompatibilidade entre ferramentas dificulta o compartilhamento de modelos e bases de conhecimentos comuns.

Para facilitar a interoperabilidade entre essas variações da linguagem e dos modelos gerados, em [5] é proposto um guia de utilização da linguagem *iStarML*, um metamodelo baseado em XML (*Extensible Markup Language*) usado para representar modelos *i**. O principal objetivo desse metamodelo é proporcionar um formato de intercâmbio entre os outros formatos de modelos do *i**. É uma especificação de linguagem que suporta todas as outras definições e especificações das variantes de modelos *i** propostos.

Os conceitos da linguagem i* são transformados em *tags* XML para gerar um arquivo com extensão *.istarmml*. Segue a relação dos principais construtores descritos na linguagem *iStarML* e o seu respectivo significado na linguagem i*:

- *Actor*: um ator representa uma entidade que pode ser uma organização, um departamento, uma pessoa ou software. Também pode ser uma abstração de um ator como um papel ou posição ocupada.
- *Intentional element*: *Intentional element* é uma entidade que pode se relacionar com diferentes atores de acordo com sua rede social ou também para expressar uma necessidade, são exemplos de elementos internos: *task*, *goal*, *softgoal* e *resource*.
- *Dependency*: uma dependência é um relacionamento onde é explicitada a dependência de uma ator (*dependee*) com outro ator (*dependor*) ou com um *Intentional Element*.
- *Boundary*: representa um grupo de *intentional elements*, e representa os objetivos e a visão de escopo relacionadas a um determinado ator.
- *ielementLink*: Um *Intentional element link* representa um relacionamento entre *Intentional elements* e podem ser *decompositionlink*, *contributionlink* e *meansendlink*.
- *actorlink*: representa um relacionamento entre dois atores, pode ser um *isa*, *ispartof*, *instanceof*, *plays*, *occupies* e *covers*.

Na Tabela 3.1 são apresentadas algumas *tags* adicionais que o *iStarML* utiliza e que não existe um conceito correspondente na linguagem i*, juntamente com o seu significado.

Tabela 3.1: Tags complementares do *iStarML*

Conceitos Adicionais	Tag	Significado
Arquivo em linguagem i*	<istarmml>	A tag principal do <i>iStarML</i>
Diagrama	<diagram>	Um diagrama representa um modelo gráfico particular
Gráfico	<graphic>	Representa alguma propriedade gráfica do diagrama em particular

Na Figura 3.1 pode ser observado um exemplo de relacionamento de dependência de meta utilizando a linguagem *iStarML*, cada um dos elementos é representado pela <tag> correspondente. Neste exemplo o ator **Cliente** depende do ator **Media Shop** para realizar a meta **Comprar itens de Midia**.

```
1  <?xml version="1.0"?>
2  <istarml version="1.0">
3    <diagram name="Dependência de Meta">
4      <actor id="05" name="Cliente"/>
5      <actor id="06" name="Media Shop"/>
6      <ielement id="212" name="Comprar itens de Midia" type="goal">
7        <dependency>
8          <depender aref="05"/>
9          <dependee aref="06"/>
10        </dependency>
11      </ielement>
12    </diagram>
13  </istarml>
```

Figura 3.1: Relacionamento de dependência de meta em iStarML

Esta abordagem possibilita a interoperabilidade, mas não ajuda na validação e verificação de inconsistências ou presença de mau uso dos construtores da linguagem, já que a *iStarML* realiza apenas checagens sintáticas [15].

3.2 IStar Tool

Em [28] é proposto o desenvolvimento da ferramenta *IStar Tool*, que foi baseada no uso do framework GMF (*Graphical Modelling Framework*), o qual provê um processo para desenvolvimento baseado em modelos. A criação dos modelos foi fundamentada no estudo detalhado da linguagem *i**. Os modelos criados seguindo o GMF envolveram:

- a definição do metamodelo (modelo de domínio) para diagramas *i**;
- a definição dos modelos gráfico e ferramental;
- a definição do modelo de mapeamento que combina as definições dos modelos de domínio, gráfico e ferramental;
- a criação do modelo de geração que produz a versão executável (código Java) da ferramenta *IStar Tool*.

Um ponto importante na criação desta ferramenta foi a definição, no modelo de mapeamento, das restrições sobre o uso da linguagem *i** de acordo com os guias e boas práticas definidas em [16] e com o catálogo de erros mais comuns levantados em [33]. Nem todos os itens do catálogo de [33] foram contemplados nesta ferramenta, pois alguns deles dependem da interpretação do conteúdo dos elementos. O foco principal foi em validar o conjunto de regras básicas sobre o uso correto das ligações dos elementos na construção de modelos em *i**.

A ferramenta *IStar Tool* é capaz de realizar a verificação dos erros mais comuns encontrados em modelos *i**. Contudo, a verificação de alguns dos erros apresentados em [16], [33] ainda não foi contemplada nesta versão, ficando seu tratamento para trabalhos

futuros, apenas 7 dos 15 erros apresentados em [33] foram tratados, são eles: os erros 1(c), 2(a), 2(c), 3(d), 3(f) e 3(g)(vide anexo A). A verificação de consistência dos modelos foi definida para ser realizada à medida que os modelos estão sendo criados. Portanto, a ferramenta *IStar Tool* só permite a criação de modelos válidos.

Segue abaixo na figura 3.2 a tela da ferramenta *IStar Tool*, nesta imagem é possível visualizar a representação de um relacionamento de dependência. Neste caso o **AtorA** depende do **AtorB** para realizar a **meta**.

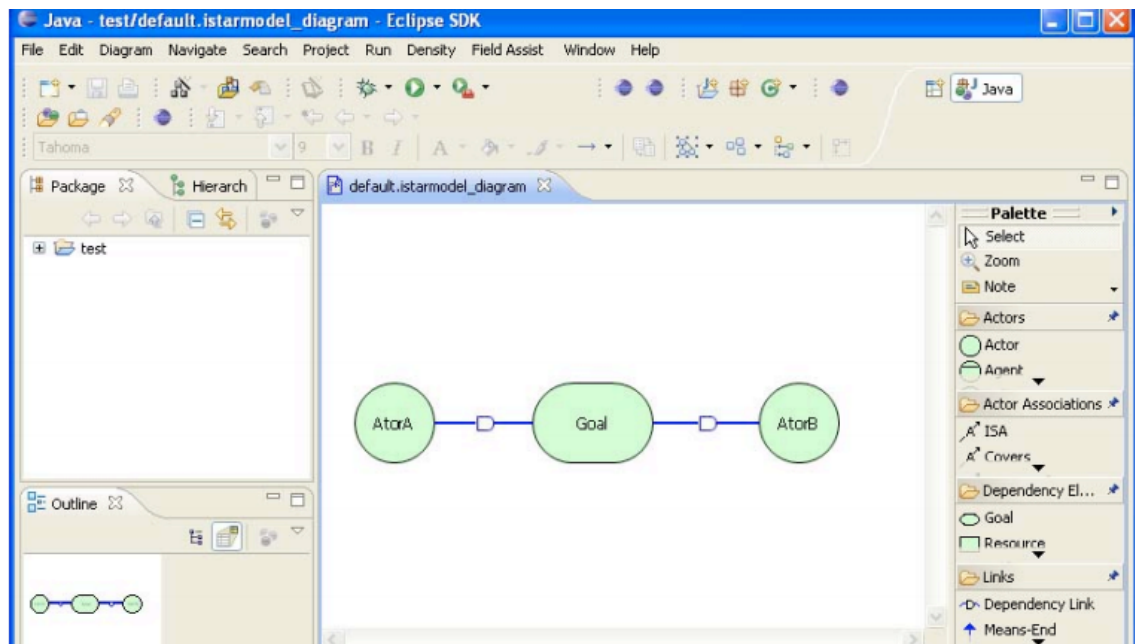


Figura 3.2: Relacionamento de dependência de meta em iStarML

3.3 AIRDoc-i*

Em [39] é proposto um processo baseado em BPMN (*Business Process Modeling Notation*), chamado *Approach to Improve Requirements Documents for i* models* (*AIRDoc-i**), para avaliação de modelos i* através da análise das atividades e dos artefatos do processo. Foi elaborado um processo gráfico em *BPMN* que serve como guia para a detecção de erros sintáticos da linguagem i* em seus modelos. O processo é dividido em duas etapas, a fase de avaliação e a fase de melhoria.

A fase de avaliação possui quatro atividades que são: elaboração do plano de atividade, definir as atividades do GQM (*Goal Question Metric*), coletar os valores das métricas e interpretação das atividades do GQM. Estas atividades têm como objetivo definir a equipe de avaliação, definir a pergunta, a métrica, apontar os erros sintáticos nos modelos i* e coletar o resultado da métrica.

A fase de melhoria possui duas atividades que são identificar e aplicar as correções. A identificação das correções acontece quando o *stakeholder* recorre ao catálogo

de erros frequentes em i^* para encontrar a forma correta de realizar aquela modelagem na qual o erro identificado. A aplicação da correção do modelo i^* acontece na segunda atividade desta fase produzindo um modelo i^* melhorado.

Na figura 3.3 é possível visualizar o processo *AIRDoc- i^** , onde cada um dos retângulos na parte superior do quadro representa uma das quatro atividades da fase de avaliação, e cada um dos retângulos na parte inferior da figura representa as atividades da fase de melhoria.

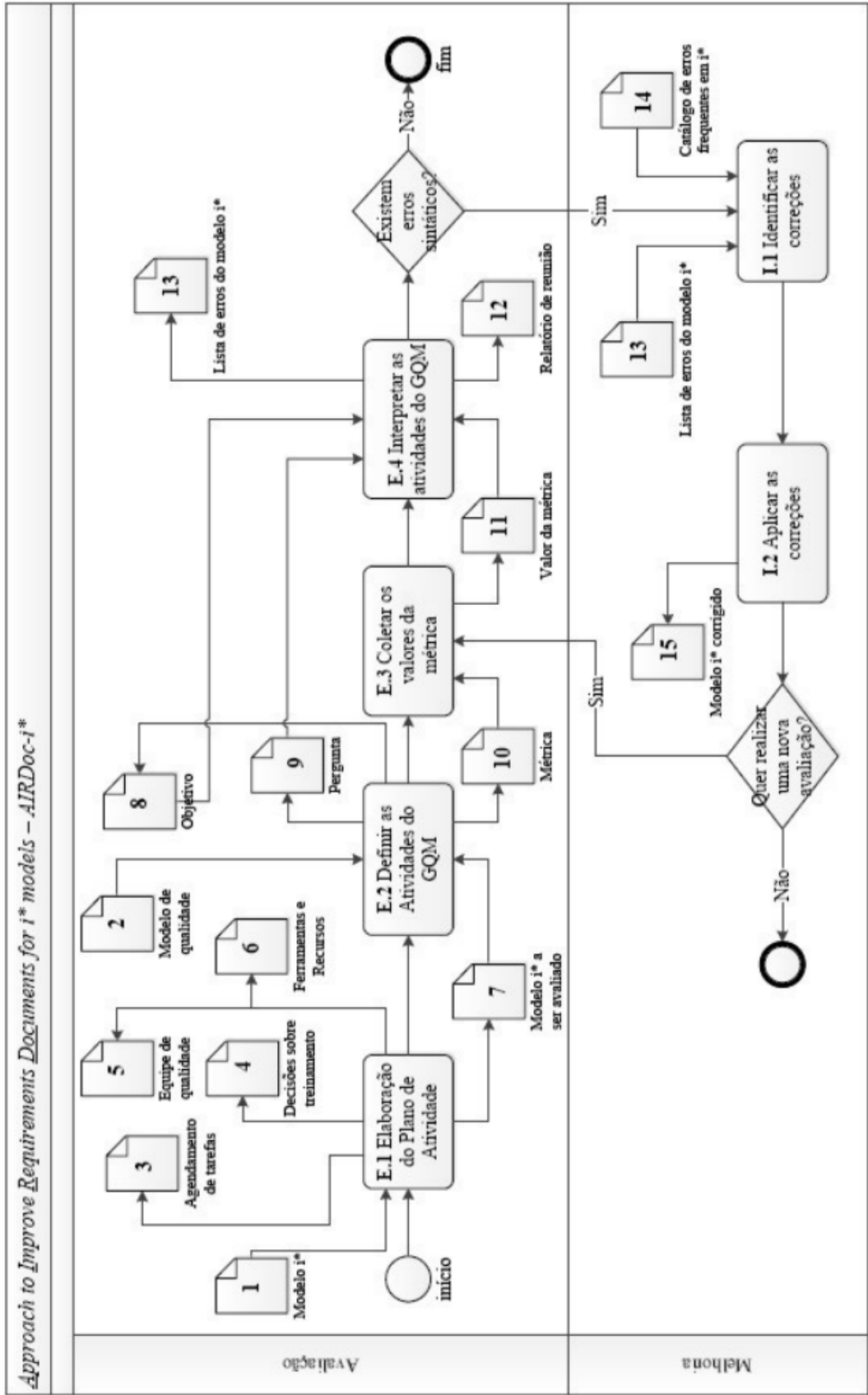


Figura 3.3: O processo AIRDoc-i*

Este modelo possibilita que sejam encontrados e corrigidos os erros em modelos gerados utilizando a linguagem i^* , porém o processo de verificação de erros é manual, dependendo bastante do entendimento de modelagem *BPMN*. Além de que o processo de correção também é manual, o que pode gerar novos erros e necessitar de outra validação.

3.4 OntoiStar+

Em [21] foi desenvolvida a ontologia denominada *OntoiStar*, que modela os construtores da linguagem i^* em uma ontologia utilizando a linguagem *OWL*. Posteriormente com o objetivo de integração e interoperabilidade entre as variantes da linguagem i^* , a autora [21] desenvolveu mais duas ontologias. Sendo uma ontologia com os construtores da variante *Tropos* e outra com os construtores da *Service-Oriented i^** . Foi realizada a integração das três ontologias geradas, criando assim a *OntoiStar+*, para possibilitar a interoperabilidade entre as variantes da linguagem i^* .

O desenvolvimento da ontologia *OntoiStar+* foi realizado seguindo a abordagem MDE (*Model Driven Engineering*). A MDE é uma abordagem de desenvolvimento de software que considera modelos como a regra chave para descrever o sistema a ser desenvolvido. Esta abordagem é baseada em arquitetura em camadas, onde modelos, metamodelos e metametamodelos são representados como M1, M2 e M3 respectivamente.

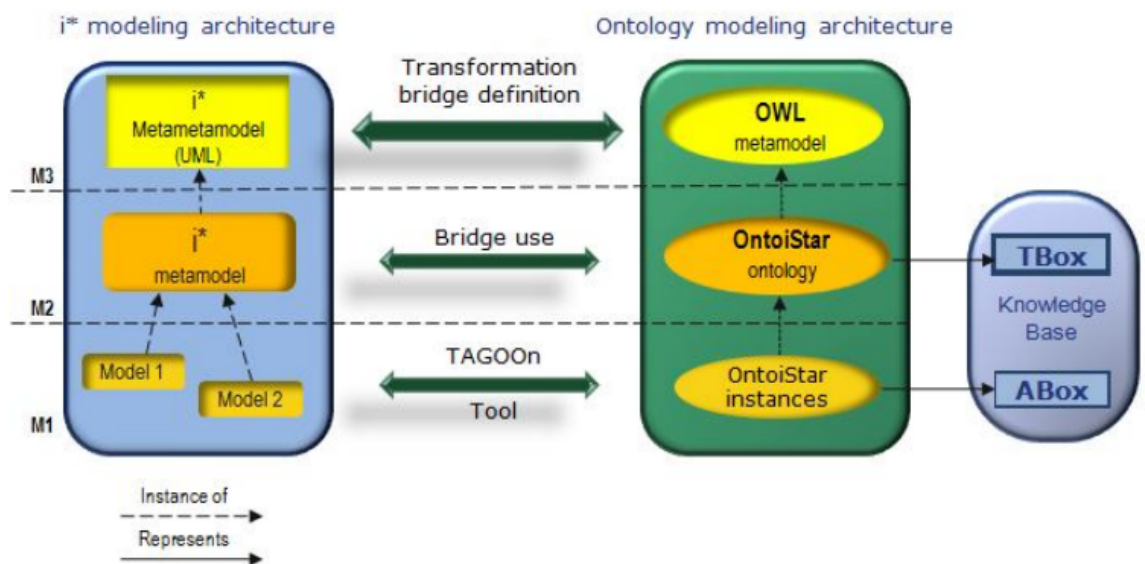


Figura 3.4: Arquitetura de Desenvolvimento da OntoiStar [21]

Na Figura 3.4 podem ser observadas as camadas utilizadas nessa abordagem. Do lado esquerdo está a modelagem da arquitetura da linguagem i^* , onde o metamodelo i^* representado em linguagem *iStarML* está localizado na camada M2, e é descrito pelo metametamodelo representado em UML (*Unified Modeling language*) na camada M3.

Do lado direito está apresentada a arquitetura da solução ontológica desenvolvida em [21], onde o metamodelo da linguagem i^* está na camada M2 e é descrito por um metamodelo em OWL. A ponte de transformação é definida na camada M3, onde contém as regras de mapeamento entre os conceitos do metamodelo i^* , cada classe, associação e conceitos do metamodelo em OWL, como classes e propriedades. Na camada M2 é aplicada a transformação do metamodelo em i^* em uma instância da ontologia *OntoiStar+*. Para possibilitar a transformação automática de um metamodelo i^* em uma instância da *OntoiStar+* foi desenvolvida a ferramenta *TAGOOon+* que será descrita com maiores detalhes na seção seguinte.

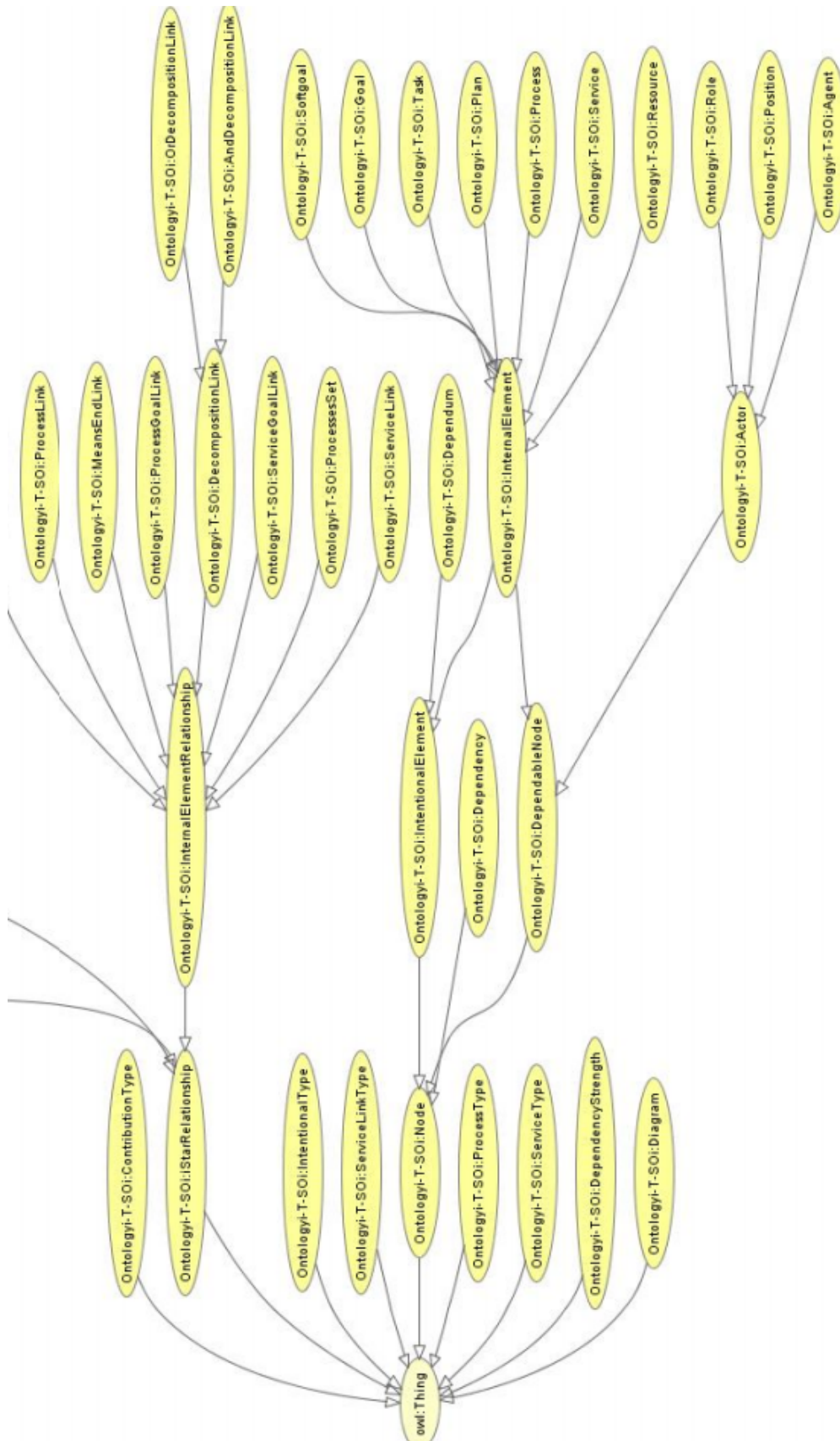


Figura 3.5: Representação gráfica de uma parte da OntoiStar+ [21]

Na Figura 3.5 pode ser observada a representação gráfica de uma parte da *OntoiStar+*, com a descrição de cada conceito da linguagem *i**, *Tropos* e *Service-Oriented i** e relacionamentos possíveis. Porém, como o objetivo da ontologia é apenas modelar os construtores da linguagem, não faz parte da proposta a verificação dos modelos gerados quanto à correta utilização dos construtores, exigindo um conhecimento profundo das regras da linguagem *i**.

3.5 TAGOOn+

A ferramenta **TAGOOn+** (*Tool for the Automatic Generation of Organizational Ontologies*), ou em português ferramenta para geração automática de ontologias organizacionais, foi desenvolvida pela mesma autora da *OntoiStar+* [21], com o objetivo de realizar a transformação automática de um modelo em *iStarML* em uma instância da ontologia *OntoiStar+*. Na figura 3.6 é possível visualizar a interface gráfica do software.

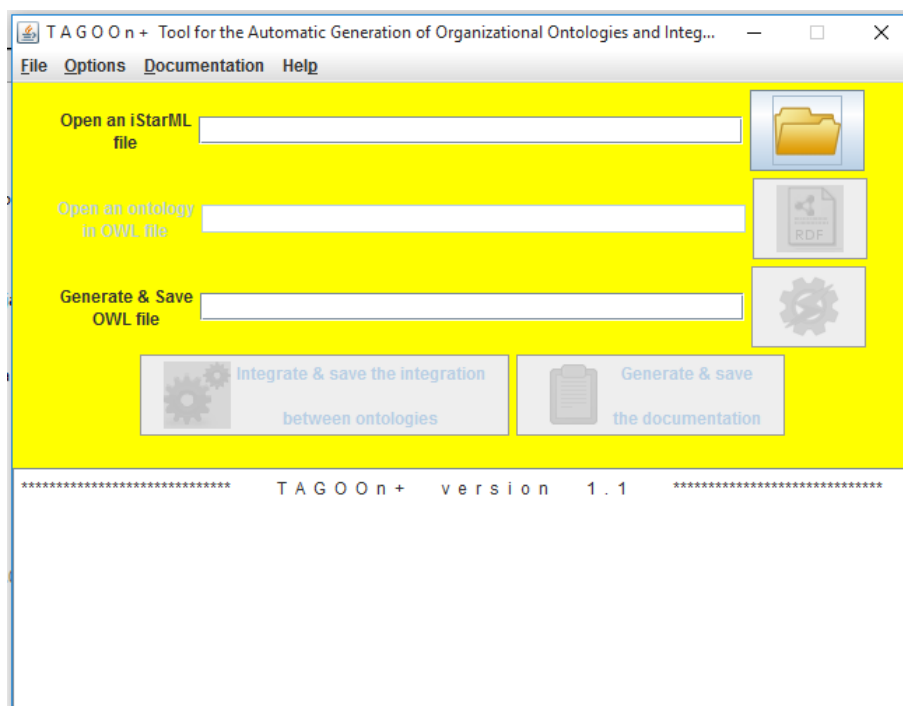


Figura 3.6: TAGOOn+:Interface Gráfica [21]

A ferramenta suporta a transformação automática de modelos expressados em qualquer uma das variantes mapeadas na *OntoiStar+*: *i**, *Tropos* e *Service-oriented i**. O processo de transformação é realizado recebendo como entrada um modelo *i** em formato *iStarML*, e então o *TAGOOn+* faz um mapeamento de cada *<tag>* do *iStarML* para o seu respectivo construtor mapeado em *OWL* segundo a ontologia *Ontoistar+*. Este fluxo do processo de mapeamento do *TAGOOn+* é apresentado graficamente na Figura 3.7.

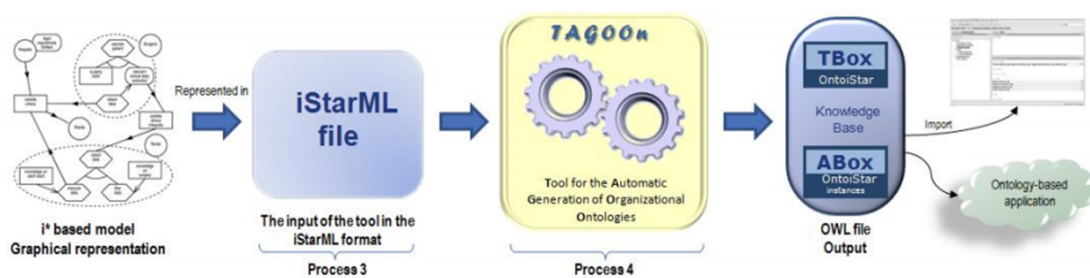


Figura 3.7: TAGOOn+:Fluxo do processo de transformação [21]

O arquivo em *iStarML* é analisado durante a execução de TAGOOn+ a fim de aplicar regras de mapeamento para transformar o modelo em *iStarml* para um modelo em *OWL*. O arquivo *OWL* gerado pode ser importado com um editor de ontologias para modificar a base de conhecimento, consultar e realizar inferências utilizando algum tipo de *plugin* raciocinador.

3.6 Considerações Finais

Embora seja visível que os trabalhos citados anteriormente mostraram resultados significativos, o *Airdoc-i** e *Istar tool* não possibilitam a criação de modelos utilizando variantes da linguagem *i**. Desta forma prejudicando a interoperabilidade de modelos criados com diferentes versões da linguagem.

A abordagem da *iStar tool* é a que mais se assemelha ao trabalho executado nesta dissertação, já que é a única que se preocupa na validação dos modelos gerados. Esta ferramenta faz a validação em tempo de execução, não permitindo que seja criado um modelo contendo os erros que são tratados pela *iStar tool*. Porém, a ferramenta não permite a modelagem ou integração de modelos que utilizem alguma das variantes da linguagem *i**, além de não tratar 8 dos erros listados no catálogo de erros frequentes da linguagem *i**.

Analisando a *OntoiStar+*, TAGOOn+ e a *iStarML* percebe-se que as três soluções permitem utilização de construtores pertencentes a variantes da linguagem *i**, porém a primeira e a segunda soluções não se preocupam em validar a utilização dos construtores, e a terceira apenas faz validações sintáticas.

Tabela 3.2: *Quadro comparativo entre Trabalhos Relacionados*

Características	Airdoc i*	iStar tool	OntoiS- tar+	TA- GOOn+	iStarML
Validação	SIM (manual)	SIM (automática)	NÃO	NÃO	SIM (automática)
Interoperabilidade	NÃO	NÃO	SIM	SIM	SIM
Percentual de erros tratados	(-)	47%	(-)	(-)	(-)
Agrega semântica aos construtores da i*	NÃO	NÃO	SIM	SIM	NÃO
Análise sintática	SIM (Manual)	SIM (automática)	SIM (Parcial)	SIM (Parcial)	SIM (Parcial)

Analisando a Tabela 3.2, percebe-se que apenas a *iStar tool* faz a validação quanto à utilização correta dos construtores da linguagem i*, enquanto as outras abordagens fazem apenas uma validação sintática da utilização destes construtores e mesmo assim na maioria delas apenas uma análise parcial e/ou manual. Avaliando os resultados dos trabalhos relacionados citados, foi percebida a necessidade de uma proposta que tratasse as lacunas deixadas. Sendo que a única destas ferramentas que propõe a validação da utilização dos construtores da linguagem i* conseguiu um sucesso em apenas 47% dos erros relatados na literatura. Com o objetivo de criar uma solução que além de se preocupar com a interoperabilidade das variantes da linguagem i*, se preocupasse ainda com a validação de erros de interpretação dos construtores tanto em relação aos erros sintáticos quanto aos erros semânticos.

Neste sentido a solução proposta nesta dissertação utiliza a ontologia *OntoiStar+*, que além de permitir a interoperabilidade dos modelos i* ainda acrescenta semântica aos construtores dessa linguagem, e acrescenta regras *OWL* para que seja possível realizar a validação da utilização correta destes construtores, reduzindo efetivamente a possibilidade de criar modelos inconsistentes. É realizada ainda uma extensão da ferramenta *TAGOOn+*, permitindo assim que a ferramenta também faça a validação dos modelos i* representados utilizando *iStarML*, tendo como resultado modelos com instâncias *OWL* com menor de incidência de erros.

Extensão da ontologia *OntoiStar+* e da ferramenta TAGOOn+

Neste capítulo é tratado o desenvolvimento da proposta de uma abordagem ontológica para redução de erros em modelos de requisitos utilizando a linguagem *i**. Primeiro, será apresentado o processo de tratamento dos erros frequentes em modelos *i**, através da inserção de restrições em *OWL* na ontologia *OntoiStar+*, desta forma agregando semântica aos construtores da linguagem.

Durante o texto serão apresentados os erros, seguidos das regras *OWL* 2 que foram inseridas na ontologia para mitigar cada erro e posteriormente os testes realizados, utilizando a ferramenta *Protégé* 4.3 ¹, integrado com o raciocinador *Pellet* 2.3.4 para demonstrar que o erro não é repetido após a inserção das regras na ontologia. Em seguida, é apresentada a extensão realizada na ferramenta TAGOOn+, onde foram inseridas condições de validação, de forma a garantir que não fossem gerados modelos com os erros que foram tratados na ontologia *OntoiStar+*. São apresentados ainda os testes realizados para validação da extensão da ferramenta e os resultados obtidos.

4.1 Extensão da Ontologia *OntoiStar+*

A ideia inicial deste trabalho era criar uma ontologia com construtores da *i** para clarificar o significado e o uso correto de cada construtor. Depois, dever-se-iam criar modelos *i** que, tomando essa ontologia como referência, deveriam apresentar menos erros que aqueles modelados seguindo outras abordagens descritas no capítulo 3.

No entanto, pela metodologia de desenvolvimento de ontologias 101 [31], após a definição do escopo, o primeiro passo para criar uma ontologia é a realização de uma pesquisa para verificar a existência de uma ontologia já desenvolvida para que possa ser aproveitada. Nesta pesquisa foi encontrada a ontologia *OntoiStar+* que descreve todos

¹ disponível para download em http://protege.stanford.edu/download/protege/4.3/installanywhere/Web_Installers

os construtores da linguagem *i**, por isso, foi reutilizada para atingir o objetivo deste trabalho.

A ontologia *OntoiStar+*, como já foi citado no capítulo 3, inclui os construtores da linguagem *i** e de duas variantes da linguagem: a *Tropos* e a *Service-Oriented i**. Com o objetivo de reutilizar essa ontologia foram feitas análises dos construtores e dos erros frequentes citados em [33], a partir das quais foi percebido que a ontologia apesar de definir todos os construtores utilizados pela linguagem e suas variantes não possui regras para validação do uso correto destes construtores. Sendo assim, é possível a geração de modelos *i** inconsistentes.

A linguagem *i** possui algumas restrições de utilização de seus construtores, porém para quem está iniciando sua utilização é pouco provável que possua um conhecimento profundo de todas essas regras; mesmo usuários mais experientes estão sujeitos à má interpretação ou ao uso indevido. Essa má utilização dos construtores da linguagem fica óbvia quando em [33] é levantado um catálogo de erros que ocorrem frequentemente em modelos *i**.

Com base nas regras de utilização dos construtores da linguagem *i**, foram elaborados e incluídos axiomas e restrições na ontologia *OntoiStar+* com o objetivo de não permitir que os erros catalogados em [33] pudessem ser modelados, gerando assim modelos mais consistentes. As restrições em OWL inseridas na ontologia, de forma a definir limites para indivíduos que pertencem a uma classe, foram as do tipo restrição de cardinalidade citadas na seção 2.3.

4.1.1 Tratamento dos Erros

Com base no catálogo de erros frequentes descritos em [33] e discutidos em [37] e [38], foram elaborados e incluídos axiomas e restrições na *OntoiStar+* para prevenir a ocorrência desses erros. Para isso, utilizou-se a semântica de construtores OWL e o editor *Protégé* integrado ao raciocinador *Pellet* [34].

Todos os exemplos apresentados nesta seção seguem a modelagem do cenário *Media Shop*, que inclui uma loja de vendas de diferentes tipos de itens (livros, jornais, revistas, cds, etc) que pode ser visto com mais detalhes em [33], [7]. Para fins de implementação das restrições apresentadas a seguir foi utilizada como referência a evolução do *i** original², o que justifica a configuração empregada no tratamento dos erros 4 e 8 descritos nesta seção.

²Disponível em: <http://istar.rwth-aachen.de/tiki-index.php?page=iStarQuickGuide>

Erro 1: Atores dentro da fronteira de outro ator

Conforme recomendação de boas práticas para utilização da linguagem *i**, descritas em [33], [28] e [4], não deve existir um ator dentro da fronteira de outro ator. Na ontologia *OntoiStar+* é definida a classe *ActorBoundary*, que representa a fronteira de um ator.

Para eliminar o erro 1[b] do catálogo de erros, a possibilidade da existência de mais de um ator dentro dessa fronteira redefiniu-se a classe *ActorBoundary* para que ela apresente o relacionamento *has_Actor_Boundary* e/ou *has_Actor_boundary_elements* com exatamente 0 atores. Assim, o ator do *ActorBoundary* será o único dentro do relacionamento, permitindo apenas relacionamento com qualquer elemento da classe *InternalElement*.

Na Figura 4.1, à esquerda, observa-se a modelagem correta, utilizando um único ator dentro da fronteira; neste caso o ator **Medi@**. À direita, na mesma figura, observa-se uma modelagem incorreta, em que dentro da fronteira encontram-se os atores **Medi@** e **Bank_Cpy**, algo que não deveria ser permitido.

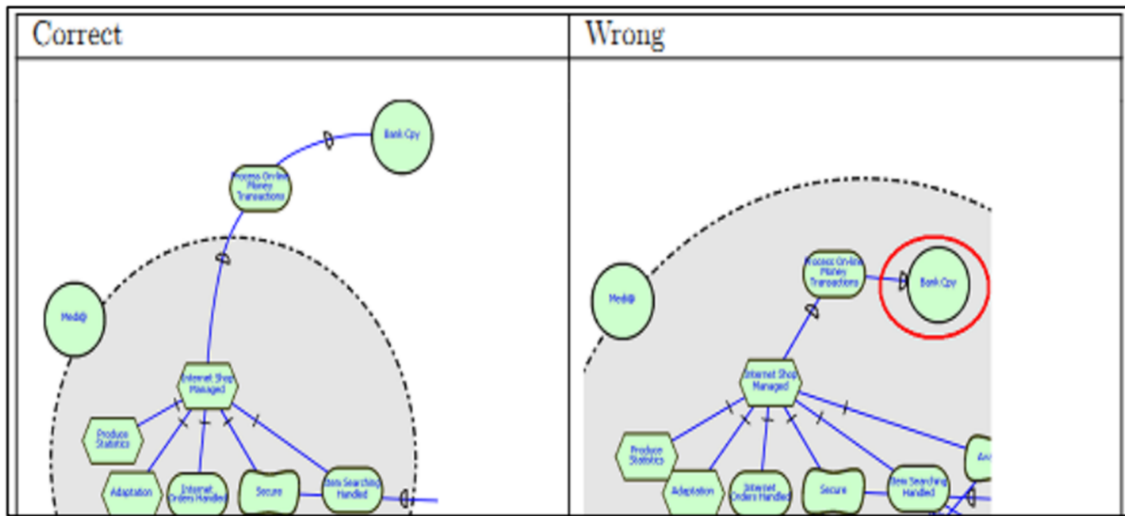


Figura 4.1: Comparação entre as modelagens correta e incorreta da fronteira de ator[33]

Após essa redefinição ser inserida na ontologia *OntoiStar+*, percebe-se que não é mais possível criar uma ligação com mais de um ator dentro da fronteira. Na Figura 4.2 a ferramenta *Protégé* acusa uma inconsistência ao se tentar reproduzir o erro da Figura 4.1 e ligar os atores *Medi@* e *Bank_Cpy* pelo relacionamento *has_Actor_Boundary*.

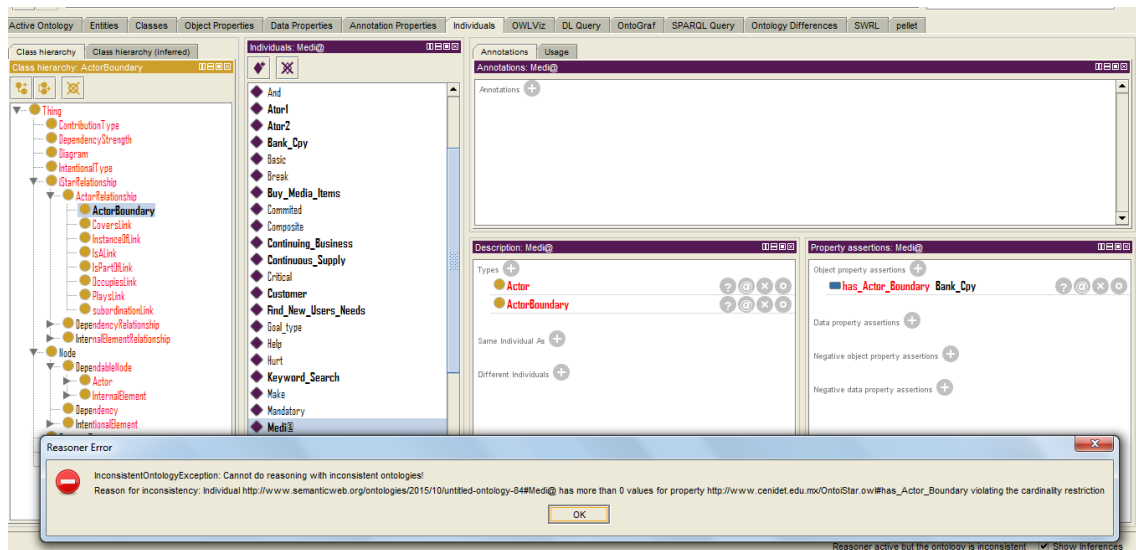


Figura 4.2: Mensagem de erro: Não permite mais de um ator dentro da fronteira. **Fonte:** Elaborada pelo autor, 2016

Segue na Figura 4.3 um trecho da restrição OWL inserida na classe *ActorBoundary* para garantir que o erro não seja reproduzido. Como pode ser observado na figura, foi utilizada a restrição de cardinalidade do tipo **ExactCardinality**, para garantir que dentro da fronteira do ator exista exatamente 0 objetos da classe *Actor*.

```
<SubClassOf>
  <Class abbreviatedIRI="OntoiStar:ActorBoundary"/>
  <ObjectExactCardinality cardinality="0">
    <ObjectProperty
      abbreviatedIRI="OntoiStar:has_Actor_Boundary"/>
    <Class abbreviatedIRI="OntoiStar:Actor"/>
  </ObjectExactCardinality>
</SubClassOf>
```

Figura 4.3: Restrição OWL: Trecho que não permite mais de um ator dentro da fronteira. **Fonte:** Elaborada pelo autor, 2016

Erro 2: Conexão de um *InternalElement* com um *ActorBoundary*

A classe *ActorBoundary* não deve possuir nenhum relacionamento direto do tipo *contribution* ou *dependency* com um *InternalElement*. Assim, foi incluída para a classe *ActorBoundary* as seguintes redefinições de classe para garantir que não ocorram os erros 2[b] e 2[c] do catálogo de erros :

Tabela 4.1: Condições para que não exista conexão entre *InternalElement* e *Actor Boundary* **Fonte:** Elaborada pelo autor, 2016

Propriedade do Objeto	Restrição inserida
has_Dependency_DependLink_source-ref	exactly 0 DependableNode
has_Dependency_DependLink_target-ref	
has_Dependency_DependeeLink_source-ref	
has_Dependency_DependeeLink_target-ref	
has_Dependency_DependumLink_source-ref	
has_Dependency_DependumLink_target-ref	
has_InternalElement_MeansEndLink_source-ref	
has_InternalElement_MeansEndLink_target-ref	
has_InternalElement_AndDecompositionLink_source-ref	
has_InternalElement_AndDecompositionLink_target-ref	
has_InternalElement_ContributionLink_source-ref	
has_InternalElement_ContributionLink_target-ref	

As regras apresentadas na Tabela 4.1 evitam a existência de qualquer relacionamento do tipo *DependencyRelationship* ou *InternalElementRelationship* com a classe *ActorBoundary*. Para atingir este objetivo foi utilizada uma restrição de cardinalidade que especifica o número exato de relacionamentos possíveis entre os objetos especificados, ou seja, os objetos das classes *DependencyRelationship* ou *InternalElementRelationship* devem ter exatamente 0 relacionamento com os objetos da classe *DependableNode*. As regras foram criadas usando *DependableNode* em vez de *InternalElement*, pois assim garante que não poderão existir esses relacionamentos com itens da classe *InternalElement*, nem com a classe *Actor*, que conforme o erro 1 já apresentado também é um relacionamento indevido.

Depois de inseridas as regras citadas na Tabela 4.1 não é mais possível gerar nenhum relacionamento do tipo *contribution* ou *dependency* com um *InternalElement*, como pode ser observado na Figura 4.4, onde é apresentada a tela de erro exibida pelo *Protégé*.

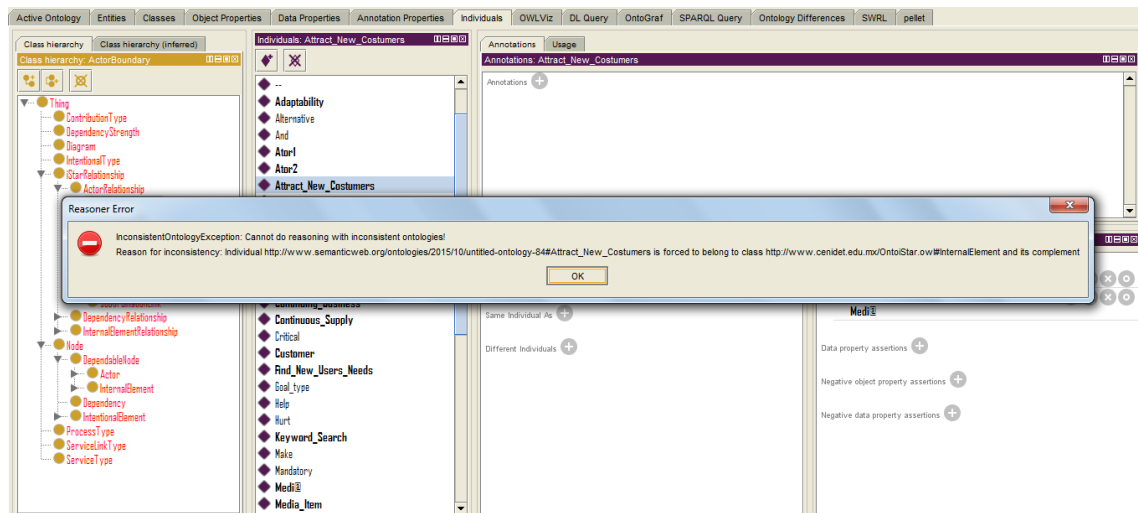


Figura 4.4: Mensagem de erro: Não permite mais de um ator dentro da fronteira. **Fonte:** Elaborada pelo autor, 2016

Erro 3: Dependência de ator

Deve-se usar *DependencyRelationship* somente para expressar dependência entre um membro da classe *Actor* e um *InternalElement*, ou seja não usar nenhum dos relacionamentos do tipo *InternalElementRelationship* para representar dependências. Para garantir que o erro 2[a] do catálogo de erros não ocorra foram acrescentadas à ontologia, na classe *Actor*, as restrições apresentadas na Tabela 4.2:

Tabela 4.2: Condições para que não haja relacionamentos do tipo *InternalElementRelationship* para representar uma dependência entre um *Actor* e um *InternalElement*.
Fonte: Elaborada pelo autor, 2016

Propriedade do Objeto	Restrição inserida
has_InternalElement_MeansEndLink_source-ref	exactly 0 InternalElement
has_InternalElement_MeansEndLink_target-ref	
has_InternalElement_AndDecompositionLink_source-ref	
has_InternalElement_AndDecompositionLink_target-ref	
has_InternalElement_ContributionLink_source-ref	
has_InternalElement_ContributionLink_target-ref	
has_InternalElement_DecompositionLink_source-ref	
has_InternalElement_DecompositionLink_target-ref	
has_InternalElement_OrDecompositionLink_source-ref	
has_InternalElement_OrDecompositionLink_target-ref	

Para evitar o erro 2[a] foi utilizada uma restrição de cardinalidade, ou seja, os objetos das classes *InternalElementRelationship* devem ter exatamente 0 relacionamento com os objetos da classe *InternalElement*. Depois de inseridas essas regras não é mais

possível gerar nenhum relacionamento do tipo *InternalElementRelationship* para representar uma dependência entre um *Actor* e um *InternalElement*, como pode ser observado na Figura 4.5, onde é apresentada a tela de erro exibida pelo *Protégé*.

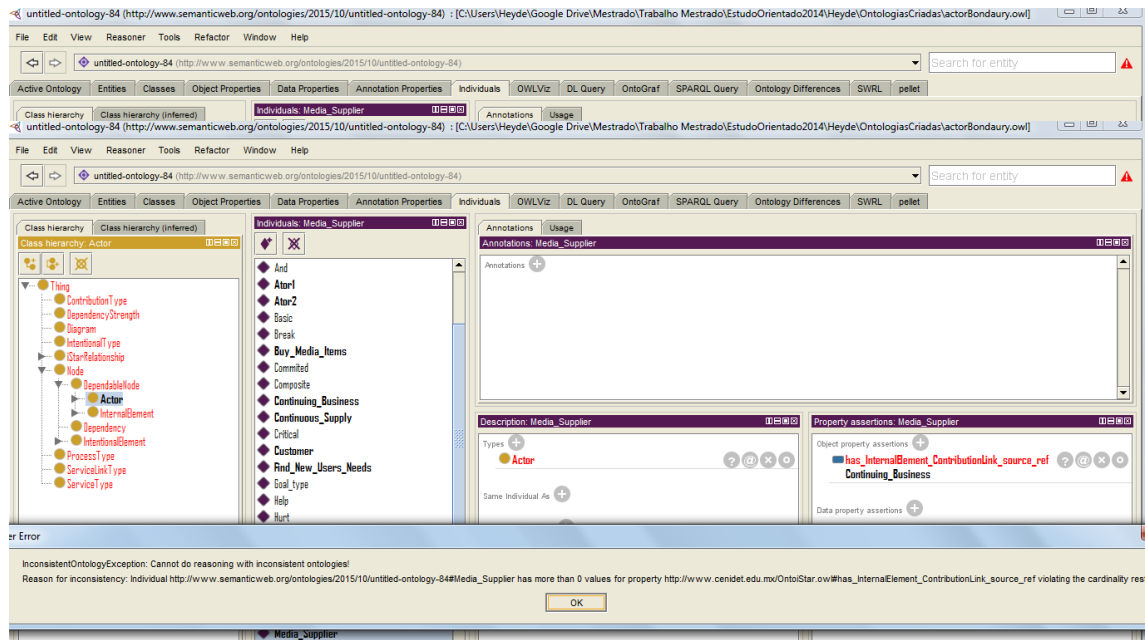


Figura 4.5: Mensagem de erro: Não permite que haja relacionamentos do tipo *InternalElementRelationship* para representar uma dependência entre um *Actor* e um *InternalElement*. **Fonte:** Elaborada pelo autor; 2016

Erro 4: Refinando metas

Uma meta não pode ser ligada diretamente a outra meta. Esta deve ser decomposta utilizando uma ligação do tipo *MeansEndLink*, e estas ligações só podem ligar tarefas a metas. Seguem as redefinições de classes para prevenir que os erros 3 [d] e [e] não ocorram.

Tabela 4.3: *Condições para que uma meta só possa ser decomposta utilizando ligações do tipo MeansEndLink e que só possa ser decomposta em tasks. Fonte: Elaborada pelo autor, 2016*

Propriedade do Objeto	Restrição inserida
has_InternalElement_MeansEndLink_source-ref	exactly 0 Goal
has_InternalElement_MeansEndLink_target-ref	
has_InternalElement_MeansEndLink_source-ref	exactly 0 Resource
has_InternalElement_MeansEndLink_target-ref	
has_InternalElement_MeansEndLink_source-ref	exactly 0 Softgoal
has_InternalElement_MeansEndLink_target-ref	
has_InternalElement_MeansEndLink_source-ref	exactly 0 Plan
has_InternalElement_MeansEndLink_target-ref	
has_InternalElement_MeansEndLink_source-ref	exactly 0 Service
has_InternalElement_MeansEndLink_target-ref	
has_InternalElement_MeansEndLink_source-ref	exactly 0 Process
has_InternalElement_MeansEndLink_target-ref	

Depois de inseridas as restrições de cardinalidade não é mais possível que uma meta possa ser ligada diretamente a outra meta. Estas restrições garantem uma limitação de cardinalidade exata sobre os relacionamentos possíveis entre os objetos especificados, ou seja, os objetos da classe *InternalElement_MeansEndLink* devem ter exatamente 0 relacionamento com os objetos das classes *Goal*, *Resource*, *Softgoal*, *Plan*, *Service* ou *Process*. Como pode ser observado na Figura 4.6, onde é apresentada a tela de erro exibida pelo *Protégé*.

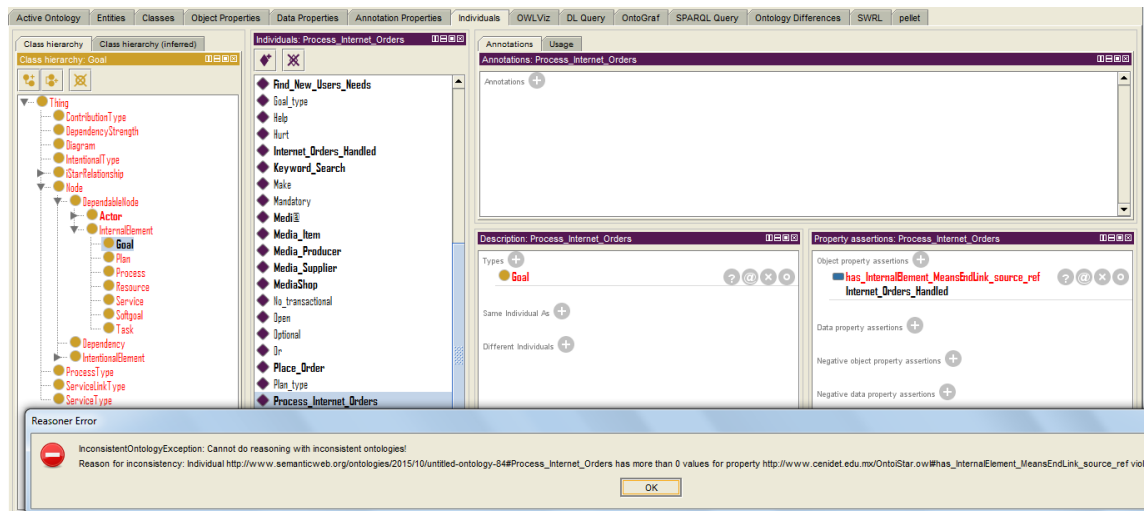


Figura 4.6: Mensagem de erro: Não permite que uma meta possa ser ligada diretamente a outra meta. **Fonte:** Elaborada pelo autor, 2016

Erro 5: Link de dependência interna

Não podem ser usados *links* de dependência dentro da fronteira de um ator. Ou seja, não devem ser realizadas relações de dependência entre um *InternalElement* dentro da fronteira de um ator. Pode haver um *Dependum* que se origina externamente à fronteira do ator ligando-se a um *InternalElement* de dentro da fronteira, mas nunca uma relação de dependência entre dois *InternalElement* internos à fronteira. Para garantir que o erro **3b** do catálogo de erros não ocorra foi acrescentada a seguinte redefinição na classe *ActorBoundary* da ontologia.

Tabela 4.4: Regra para certificar-se que um *Depender* não se origine na fronteira de um ator. **Fonte:** Elaborada pelo autor, 2016

Propriedade do Objeto	Restrição inserida
has_Dependency_DependentLink_source-ref	exactly 0 InternalElement

Após a inserção desta restrição de cardinalidade não é mais possível que um *Depender* se origine na fronteira de um ator, como pode ser observado na Figura 4.7, onde é apresentada a tela de erro exibida pelo *Protégé*. Esta restrição especifica o número exato de relacionamentos possíveis entre os objetos especificados, ou seja, os objetos das classes *has_Dependency_DependentLink_source-ref* devem possuir exatamente 0 relacionamento com os objetos da classe *InternalElement*

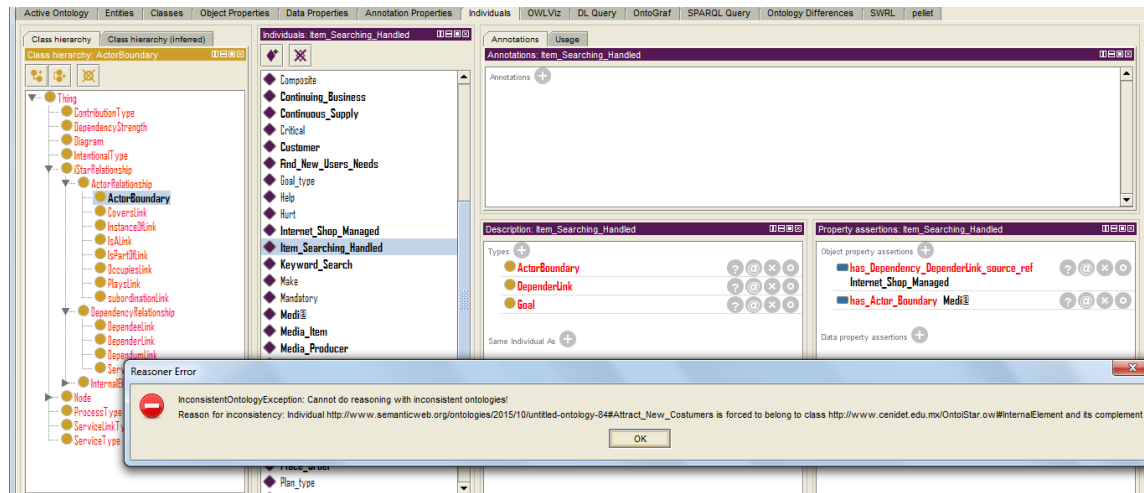


Figura 4.7: Mensagem de erro: Não permite que um *Depender* se origine na fronteira de um ator. **Fonte:** Elaborada pelo autor, 2016

Erro 6: Link de dependência externa

Não se deve usar um *link* de dependência entre dois atores sem um *Dependum*. Ou seja, um ator não pode estar diretamente ligado a outro ator. Para garantir que os erros 1 [d] e 2[c] do catálogo de erros não ocorram, foram inseridas as seguintes redefinições nas classes *DependeeLink* e *DependerLink*:

Tabela 4.5: Regras para certificar-se que um *Dependee* e um *Depender* não se conectem diretamente a outro ator. **Fonte:** Elaborada pelo autor, 2016

Propriedade do Objeto	Restrição inserida
has_Dependency_DependeeLink_source-ref	exactly 0 Actor
has_Dependency_DependeeLink_target-ref	
has_Dependency_DependerLink_source-ref	
has_Dependency_DependerLink_target-ref	

Depois de inseridas estas restrições não é mais possível que um *Dependee* e um *Depender* se conectem diretamente a outro ator, pois a restrição de cardinalidade especifica o número exato de relacionamentos possíveis entre os objetos especificados, ou seja, os objetos das classes *has_Dependency_DependeeLink* devem ter exatamente 0 relacionamento com os objetos da classe *Actor*. Como pode ser observado na Figura 4.8, onde é apresentada a tela de erro exibida pelo *Protégé*.

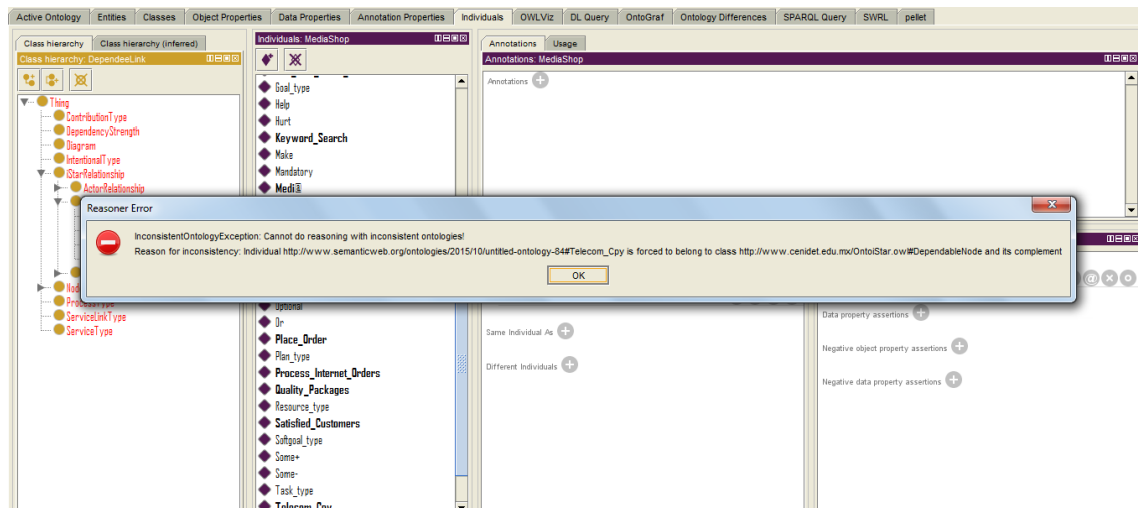


Figura 4.8: Mensagem de erro: Não permite que um *Dependee* e um *Depender* se conectem diretamente a outro ator. **Fonte:** Elaborada pelo autor, 2016

Erro 7: Link de contribuição interna

Um *ContributionLink* só deve ser usado para conectar um *IntentionalElement* a uma *Softgoal*. Para mitigar o erro 3[f] do catálogo de erros e garantir que este construtor seja utilizado de forma correta, foram adicionadas na classe *ContributionLink* as seguintes restrições:

Tabela 4.6: Restrições para certificar-se que um *Dependee* e um *Depender* não se conectem diretamente a outro ator. **Fonte:** Elaborada pelo autor, 2016

Propriedade do Objeto	Restrição inserida
has_InternalElement_ContributionLink_source-ref	exactly 0 Goal
has_InternalElement_ContributionLink_target-ref	exactly 0 Plan
has_InternalElement_ContributionLink_source-ref	exactly 0 Process
has_InternalElement_ContributionLink_target-ref	exactly 0 Resource
has_InternalElement_ContributionLink_source-ref	exactly 0 Service
has_InternalElement_ContributionLink_target-ref	exactly 0 Task

Depois de inseridas essas restrições não é possível utilizar um *ContributionLink* para conectar um *IntentionalElement* que não seja uma *Softgoal*. As restrições de cardinalidade inseridas garantem que os elementos da classe *has_IntentionalElement_ContributionLink* tenham exatamente 0 relacionamentos com os objetos das classes *Goal*, *Plan*, *Process*, *REsource*, *Service* e *Task*. Como pode ser observado na Figura 4.9, onde é apresentada a tela de erro exibida pelo *Protégé*.

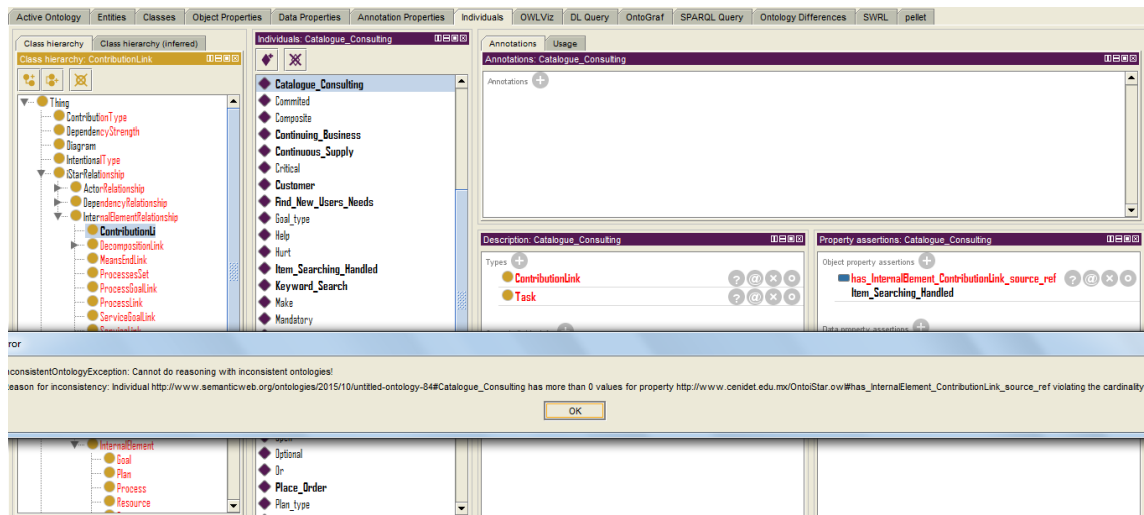


Figura 4.9: Mensagem de erro: Não permite utilizar um *ContributionLink* para conectar um *IntentionalElement* que não seja uma *Softgoal*. **Fonte:** Elaborada pelo autor, 2016

Erro 8: *Link MeansEnd* interno

Os *links* do tipo *MeansEnd* só podem ser utilizados para interligar *Task* a *Goal*. Para garantir que o erro 3[g] não ocorra, e este relacionamento seja utilizado de forma correta foram inseridas as seguintes redefinições na classe *MeansEndLink* da ontologia *OntoiStar+*:

Tabela 4.7: Regras inseridas para garantir que um *MeansEndLink* só conecte *Task* a *Goal*. **Fonte:** Elaborada pelo autor, 2016

Propriedade do Objeto	Restrição inserida
has_InternalElement_MeansEndLink_source-ref	exactly 0 Resource
has_InternalElement_MeansEndLink_target-ref	
has_InternalElement_MeansEndLink_source-ref	exactly 0 Softgoal
has_InternalElement_MeansEndLink_target-ref	
has_InternalElement_MeansEndLink_source-ref	exactly 0 Plan
has_InternalElement_MeansEndLink_target-ref	
has_InternalElement_MeansEndLink_source-ref	exactly 0 Service
has_InternalElement_MeansEndLink_target-ref	
has_InternalElement_MeansEndLink_source-ref	exactly 0 Process
has_InternalElement_MeansEndLink_target-ref	

Após inseridas essas redefinições de classe, não é possível utilizar um *MeansEndLink* para conectar um *internalElement* que não seja uma *Task* a uma *Goal*, como pode ser observado na Figura 4.10, onde é apresentado a tela de erro exibida pelo *Protégé*. AS restrições de cardinalidade inseridas garantem que os objetos da classe

has_InternalElement_MeansEndLink tenham obrigatoriamente 0 relacionamento com os objetos das classes *Resource*, *Softgoal*, *Plan*, *Serve* ou *Process*.

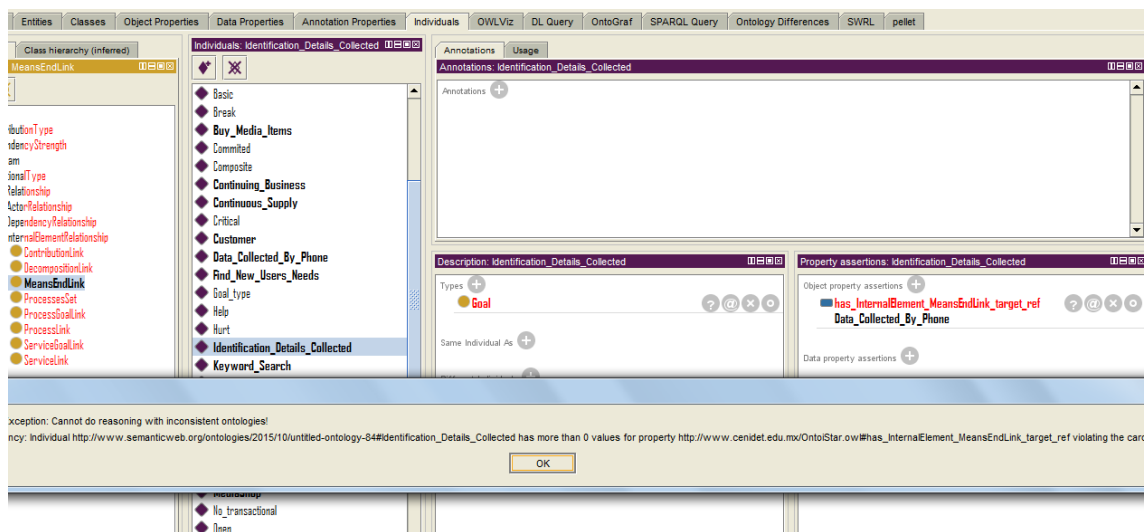


Figura 4.10: Mensagem de erro: Não é possível utilizar o *MeansEndLink* para conectar um *InternalElement* que não seja uma *Task* a uma *Goal*. **Fonte:** Elaborada pelo autor, 2016

Erro 9: Ator sem ligação ou com ligação incorreta

Não deve existir no modelo um ator sem uma ligação de dependência ou *ActorRelationship*, descrito como erro 1[a] no catálogo de erros. Além de não poder existir um ator sem ligações, os relacionamentos entre atores devem seguir as seguintes regras:

- *IsA* : só pode acontecer entre atores do mesmo tipo;
- *Plays*: só pode ocorrer entre um agente e um papel;
- *Covers*: só pode ocorrer entre uma posição e um papel;
- *Occupies*: só pode ocorrer entre uma posição e um agente;
- *INS*: só pode ocorrer entre um agente e outro agente.

Para garantir que essas regras sejam modeladas de forma correta, foram inseridas na ontologia *OntoiStar+* as seguintes redefinições de classes:

Depois de inseridas essas restrições, consegue-se garantir que as regras supracitadas de relacionamentos entre atores sejam utilizadas de forma correta, como pode ser observado nas Figuras 4.11, 4.12, 4.13, 4.14 e 4.15 onde são apresentadas as telas de erro exibidas pelo *Protégé* para cada um dos erros. As restrições de cardinalidade inseridas são do tipo que restringe o número exato de relacionamento que estas classes podem possuir. Desta forma é garantido que os

Tabela 4.8: Restrições inseridas para garantir que as ligações entre atores estejam corretas. **Fonte:** Elaborada pelo autor, 2016

Propriedade do Objeto	Restrição inserida
has_Actor_IsALink_source-ref	exactly 1 Actor
has_Actor_IsALink_target-ref	
has_Actor_PlaysLink_source-ref	exactly 1 Role
has_Actor_PlaysLink_target-ref	
has_Actor_PlaysLink_source-ref	exactly 0 Agent
has_Actor_PlaysLink_target-ref	
has_Actor_PlaysLink_source-ref	exactly 0 Position
has_Actor_PlaysLink_target-ref	
has_Actor_CoversLink_source-ref	exactly 0 Role
has_Actor_CoversLink_target-ref	
has_Actor_CoversLink_source-ref	exactly 1 Agent
has_Actor_CoversLink_target-ref	
has_Actor_CoversLink_source-ref	exactly 0 Position
has_Actor_CoversLink_target-ref	
has_Actor_OccupiesLink_source-ref	exactly 0 Role
has_Actor_OccupiesLink_target-ref	
has_Actor_OccupiesLink_source-ref	exactly 1 Agent
has_Actor_OccupiesLink_target-ref	
has_Actor_OccupiesLink_source-ref	exactly 0 Position
has_Actor_OccupiesLink_target-ref	
has_Actor_InstanceOfLink_source-ref	exactly 1 Actor
has_Actor_InstanceOfLink_target-ref	

objetos das classes *has_Actor_IsALink*, *has_Actor_PlaysLink*, *has_Actor_CoversLink*, *has_Actor_OccupiesLink*, *has_Actor_InstanceOfLink* possuam exatamente 0 relacionamentos com os objetos das classes *Actor*, *Role*, *Agente*, *Position*.

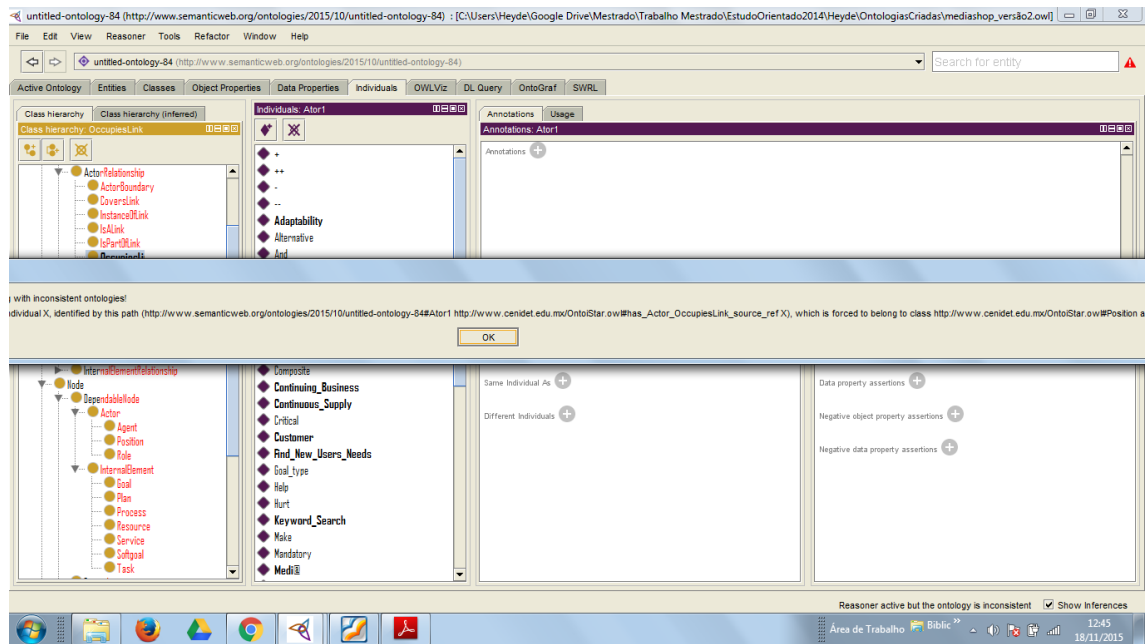


Figura 4.11: Mensagem de erro: Só pode ser utilizado o *Occupies* para ligar um *Agent* a um *Position*. **Fonte:** Elaborada pelo autor, 2016

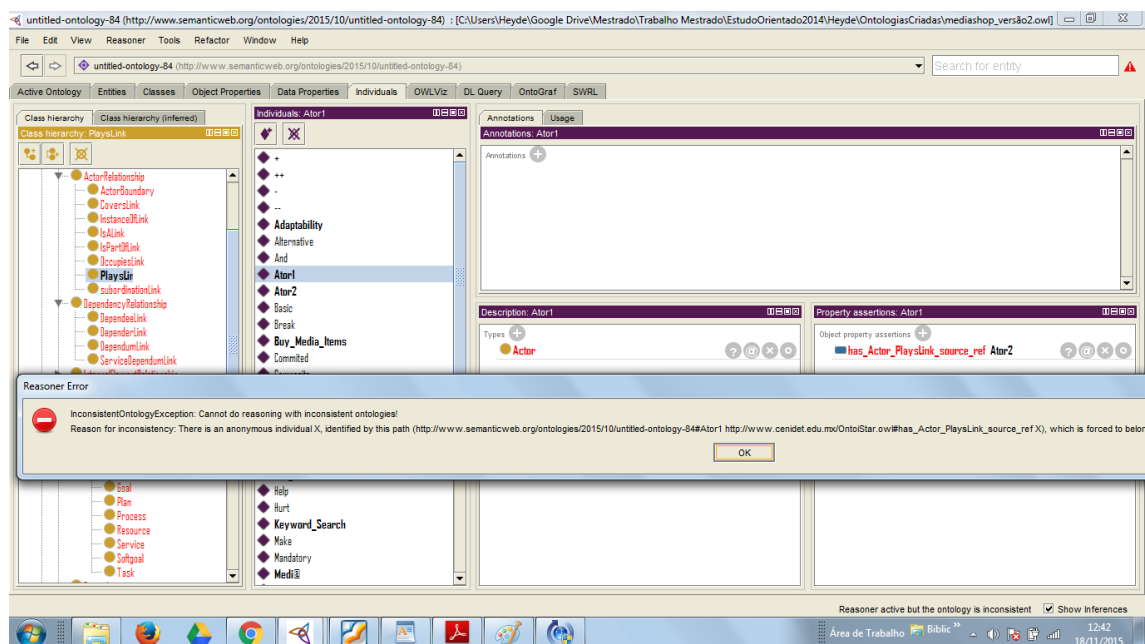


Figura 4.12: Mensagem de erro: só pode ser utilizado o *Plays* para ligar um *Agente* a um *Role*. **Fonte:** Elaborada pelo autor, 2016

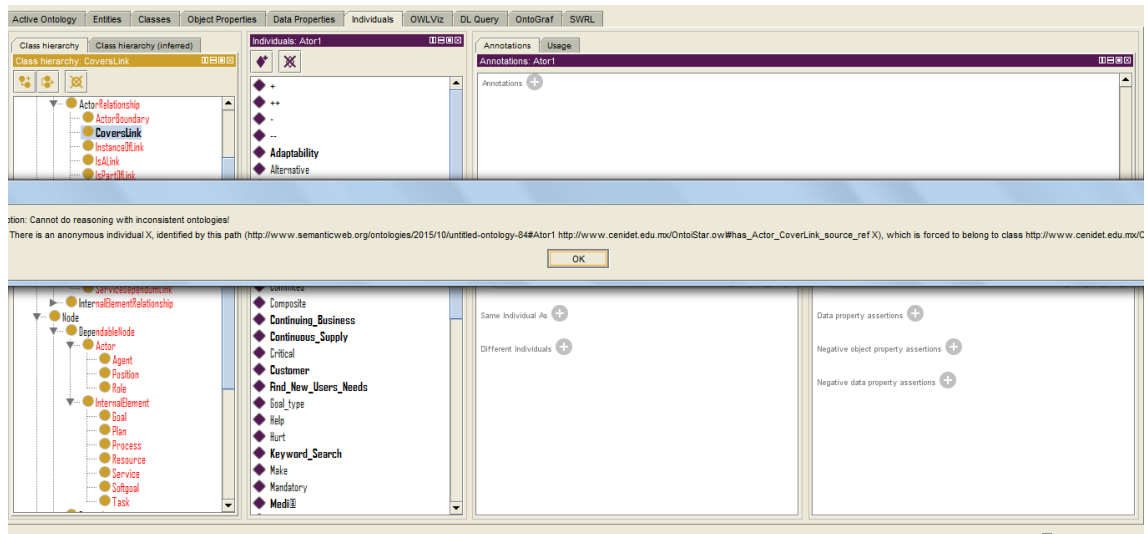


Figura 4.13: Mensagem de erro: Só pode ser utilizado o *Occupies* para ligar um *Position* a um *Role*. **Fonte:** Elaborada pelo autor, 2016

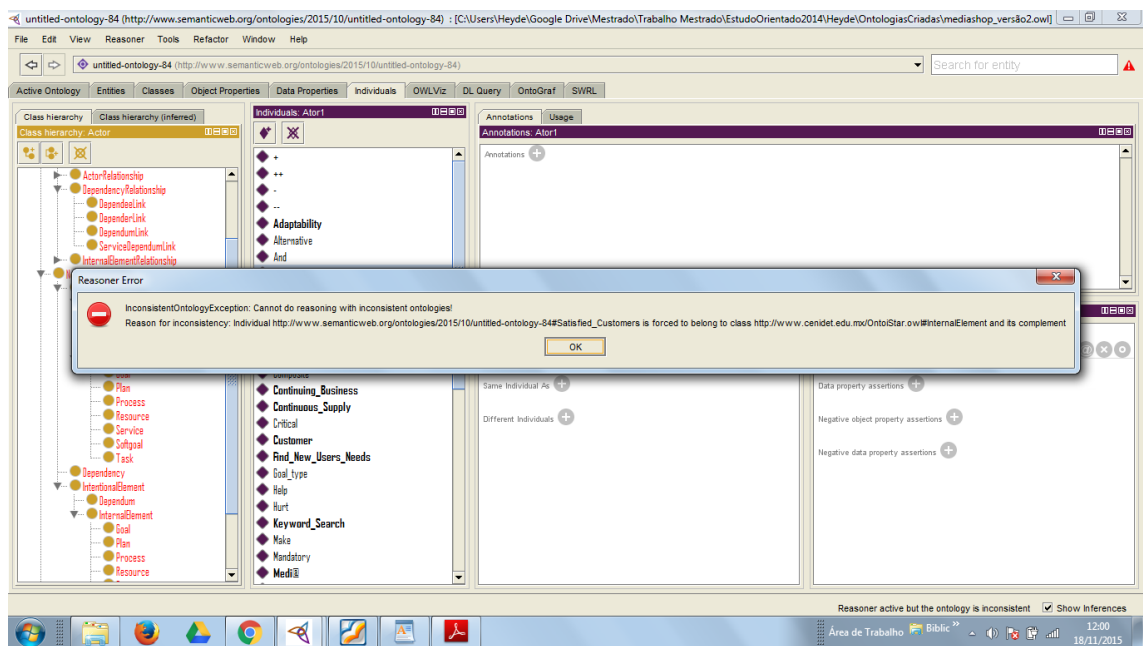


Figura 4.14: Mensagem de erro: Só pode ser utilizado o *Occupies* para ligar um *Actor* a um outro *Actor*. **Fonte:** Elaborada pelo autor, 2016

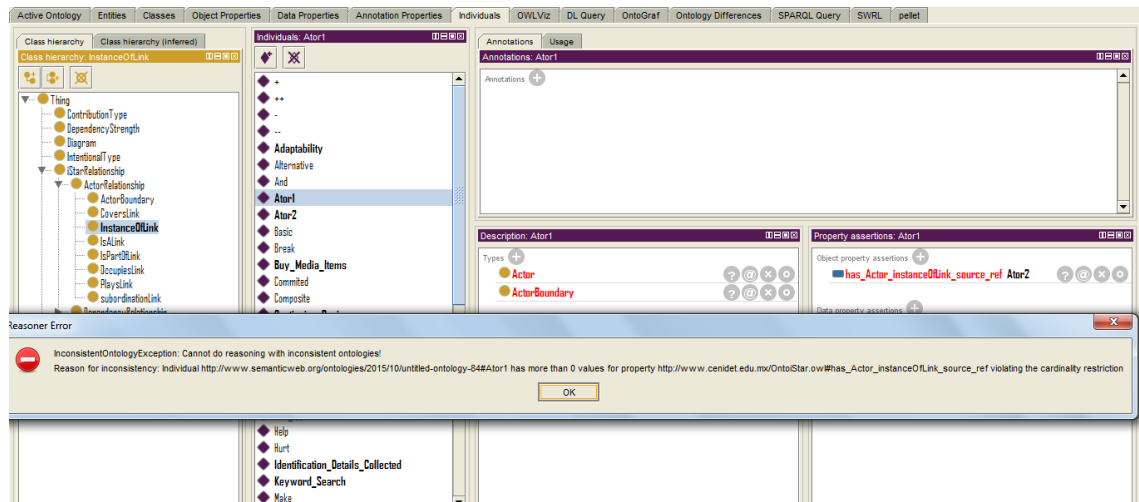


Figura 4.15: Mensagem de erro: Só pode ser utilizado o *InstanceOf* para ligar um Agent a um outro Agente. **Fonte:** Elaborada pelo autor, 2016

No entanto, essas restrições só conseguem garantir que os relacionamentos estão sendo utilizados de forma correta, mas não é possível garantir que não exista um ator sem ligação. Sendo assim, o erro 9 é coberto apenas parcialmente.

Apesar de mesmo com a inserção das regras não conseguir validar o erro **ator sem ligação**, o impacto é minimizado já que este é o erro mais fácil de ser verificado visualmente no modelo, basta verificar se existe algum construtor do tipo *Actor* sem nenhuma ligação com outro construtor.

4.1.2 Resultados Obtidos

Os resultados obtidos na etapa de validação das restrições inseridas na ontologia *OntoiStar+* foram satisfatórios. Os testes foram realizados utilizando a *OntoiStar+* estendida e a ferramenta *Protégé* 4.3 e o raciocinador *Pellet* 2.3.4, onde após a inserção de cada regra, era tentado criar um modelo com o erro listado no catálogo.

Para cada erro foram realizados testes com dois domínios distintos, o *Media Shop* e um domínio de universidade seguindo o exemplo [21] e que é apresentado em mais detalhes no Apêndice B. Nestes testes foram realizadas tentativas de gerar modelos com cada um dos erros tratados e verificado que após a inserção das regras isso não era mais possível. De forma que quando se tenta criar um modelo de requisitos como uma instância da *OntoiStar+* contendo algum dos erros relatados na seção de tratamento de erros, com o auxílio das restrições inseridas e utilizando um raciocinador (*reasoner*) que o *Protégé* possui como plugin para validação da ontologia, o próprio raciocinador já exibe uma mensagem para o usuário identificando o erro. Nos testes realizados o raciocinador utilizado foi o *Pellet* [34].

Os erros 1 a 8 foram tratados na ontologia *OntoiStar+* e não puderam ser reproduzidos após a extensão realizada, mostrando assim que as restrições inseridas realmente inibem a geração de modelos com estes erros. Já no caso do erro 9, após a inserção das regras relatadas anteriormente, os relacionamentos *IsA*, *Plays*, *Covers*, *Occupies* e *INS* ficam livre de erros, porém apenas com as alterações realizadas não é possível garantir que não exista um ator sem ligação. Para que esse erro não impacte o modelo de requisitos, recomenda-se que, após a validação, seja gerada a visualização gráfica do modelo e verificado se existe algum ator sem nenhuma conexão e, caso exista, que o modelo seja revisado.

Com a inserção das restrições em OWL 11 dos 15 erros listados no catálogo de erros frequentes em modelos i* foram tratados. Os erros que ainda permanecem sem tratamento são aqueles mais subjetivos ou que necessitam de uma expressividade maior do que a disponível pela linguagem OWL, como: erros de atores sem ligação, uso de nome inadequado de atores, elementos sem ligações e ligação direta entre elementos internos de dois atores diferentes.

A validação foi realizada utilizando apenas dois domínios diferentes, embora apenas um domínio já seria suficiente para demonstrar a efetividade da solução proposta com a extensão da *OntoiStar+*. Isto se deve ao fato de que as restrições OWL inseridas na ontologia servem para garantir a correta utilização dos construtores da linguagem e não para garantir que a modelagem do domínio escolhido está correta. Ou seja, com esta solução é garantido que o modelo construído utilizando a linguagem i* não contém 11 dos erros frequentes, porém mesmo assim a modelagem dos requisitos pode não estar correta. Desta forma o domínio escolhido não influencia nos resultados.

Para tratar os erros de elementos e atores sem ligação seria necessária a representação de uma hierarquização dos relacionamentos, para que fosse possível garantir que um ator ou elemento tivesse pelo menos um relacionamento pertencente a hierarquia. Para garantir que não exista ligação direta de elementos internos de atores diferentes seria necessário a possibilidade de verificar se os atores em cada relacionamento são diferentes. Com as restrições da linguagem OWL utilizadas nesta solução não foi possível realizar tais validações. Em relação ao erro de nome inadequado de atores, é indicado que não sejam utilizados verbos, abreviações e nomes que ultrapassem o tamanho do ícone o que torna inviável de evitar este erro apenas com as restrições em OWL.

4.2 Estendendo a Ferramenta TAGOOn+

Conforme já discutido no capítulo 1, as soluções desenvolvidas nesta pesquisa tiveram como principal objetivo possibilitar a geração de modelos de requisitos utilizando a linguagem i* livres de erros de má interpretação de seus construtores. Uma das etapas

desta pesquisa foi estender a ferramenta *TAGOOOn+*, de modo a também contribuir com a interoperabilidade entre modelos gerados com as variantes da linguagem *i**.

Para iniciar esta etapa, foi realizado um estudo da ferramenta *TAGOOOn+* para posteriormente possibilitar a inserção das regras de validação em seu código, garantindo assim que os modelos gerados utilizando a ferramenta estejam livres dos erros que foram validados usando as regras OWL na seção anterior.

A seguir serão apresentadas mais informações sobre o funcionamento e desenvolvimento da ferramenta *TAGOOOn+* e de como foi realizada sua extensão. Sendo apresentados também testes de validação dos modelos gerados pela ferramenta após a extensão, resultados obtidos e as considerações finais.

4.2.1 Inserindo as Regras no *TAGOOOn+*

A ferramenta *TAGOOOn+* [21] foi desenvolvida para realizar a transformação automática de modelos baseados em *i** para uma instância da ontologia *OntoiStar+*.

Após o entendimento das funcionalidades do *TAGOOOn+*, foram iniciadas as alterações de forma a acrescentar as restrições inseridas na *OntoiStar+*. O objetivo dessas alterações é permitir que o *TAGOOOn+* seja capaz de validar os arquivos em *iStarML* de forma que só sejam gerados arquivos OWL correspondentes, caso o modelo esteja utilizando os construtores da linguagem *i** de forma correta. Caso contrário, a ferramenta deverá apresentar uma tela de erro para que o usuário o corrija antes de gerar novamente o modelo.

Para realizar a extensão do *TAGOOOn+* foi necessário utilizar as mesmas ferramentas que foram empregadas no seu desenvolvimento, conforme descrito abaixo:

- O código da ferramenta *TAGOOOn+* foi desenvolvido em linguagem Java na IDE Eclipse. Para realizar as alterações necessárias foi utilizada a versão do *Eclipse Java Mars*.
- A JDK (*Java Development Kit*) utilizada foi a versão 1.8.0_71. JDK é um kit que contém, além do compilador e de outras ferramentas, uma biblioteca de classes completa de utilitários de pré-construção que o ajuda a realizar tarefas de desenvolvimento de aplicativos mais comuns.
- *Protégé*: é uma ferramenta gratuita, de código aberto, utilizado para edição e criação de ontologias. Permite a exportação da ontologia criada para vários formatos, incluindo OWL.

Segue na Figura 4.16 a tela do Eclipse com o projeto do *TAGOOOn+* aberto, onde podem ser observados os principais métodos da ferramenta. Inicialmente, foi realizado um estudo detalhado de cada método para entender sua funcionalidade e descobrir quais

os métodos que necessitariam de alterações para atender as restrições acrescentadas na *OntoiStar+*.

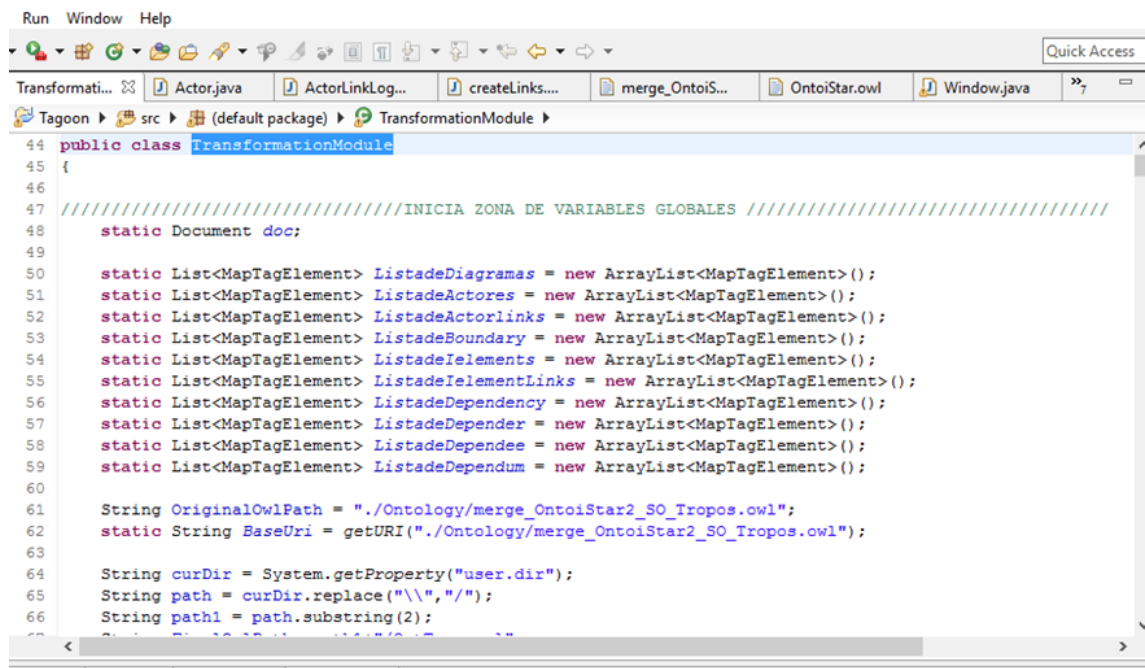


Figura 4.16: TAGOOn+: Tela do projeto TAGOOn+ [21]

Após o entendimento das classes e métodos do software TAGOOn+, ficou evidente que as alterações necessárias deveriam ser feitas nos métodos da classe *TransformationModule*, que é a responsável por transformar cada *tag* do *iStarML* em seu correspondente em OWL. Analisando esta classe, foram realizadas alterações nos métodos abaixo relacionados:

- *LocateAndInstanceDependency*: neste método foi acrescentada uma condição de teste para tratar o erro 5, ou seja, para verificar se não existe dependência entre dois atores. Caso exista algum relacionamento de dependência entre dois atores no modelo em *iStarML* é gerada uma mensagem de erro para o usuário, informando que este tipo de relacionamento só é permitido entre elementos do tipo *goal*, *softgoal*, *task*, *plan* ou *resource*.
- *LocateAndInstanceIelements*: Para garantir que não ocorra o erro 1, ou seja não deve existir um ator dentro da fronteira de outro ator, foi acrescentada neste método uma condição de teste para verificar se o *ActorBoundary* é um elemento do tipo: *goal*, *softgoal*, *task*, *plan*, *resource* ou *process*. Caso contrário, é gerada uma mensagem de erro para notificar o usuário que aquele tipo não é permitido.
- *LocateAndInstanceDependums*: neste método foi acrescentada uma condição de teste para garantir que não ocorram os erros 2, 3 e 6 relatados na seção de tratamentos de erros, verificando se não existe dependência direta entre dois atores,

e se o relacionamento *dependum* só permite a ligação entre *internalelements*. Caso contrário, é gerada uma mensagem indicando o erro para que o usuário o corrija.

- *LocateAndInstanceBTWActorActorLink*: neste método foi acrescentada uma condição de teste para verificar se os relacionamentos entre os tipos de atores estão corretos. Garantindo que não aconteça o erro 9 relacionado na seção de tratamento de erros quanto aos tipos de relacionamentos entre atores. Caso exista algum relacionamento errado, será gerada uma mensagem de erro para que o usuário corrija o relacionamento em questão.
- *LocateAndInstanceBTWElementLink*: neste método foi acrescentada uma condição de teste para verificar se os relacionamentos do tipo *decomposition*, *contribution* e *MeansEndLink* estão corretos, garantindo que não ocorram os erros 6, 7 e 8. Caso exista algum relacionamento incorreto, é gerada uma mensagem de erro para que o usuário o corrija.

4.2.2 Testes Realizados

Após a inserção de cada condição nos métodos supracitados, foram realizados testes para verificar se a alteração iria garantir que não fossem gerados modelos OWL contendo os erros relatados na seção de tratamento de erros. Para realizar tais testes foram criados arquivos *iStarML* apresentando os erros listados no catálogo de erros frequentes e verificando em seguida se o TAGOOn+ iria recusar o arquivo com erro e apresentar a mensagem de erro para que o usuário corrija o erro destacado. Seguem alguns exemplos dos arquivos *iStarML* utilizados para realizar os testes:

Para ilustrar melhor o funcionamento do TAGOOn+, será apresentado na Figura 4.17 um exemplo de arquivo *iStarML* sem nenhum erro e em seguida na Figura 4.18 a tela do TAGOOn+ informando que o arquivo OWL foi gerado com sucesso.

```

1  <?xml version="1.0"?>
2  <istarmal version="1.0">
3    <diagram name="Strategy Dependency CENEDEI">
4      <actor id="05" name="Student">
5        <boundary>
6          <ielement id="104" name="Analyze courses" type="service"/>
7          <ielement id="105" name="Authorize schedule" type="actor"/>
8        </boundary>
9      </actor>
10     <actor id="06" name="Thesis advisor"/>
11     <actor id="09" name="Department chair"/>
12     <ielement id="212" name="Department chair 1" type="softgoal">
13       <dependency>
14         <depender aref="05"/>
15         <dependee aref="06"/>
16       </dependency>
17     </ielement>
18     <ielement id="213" name="Choose courses" type="goal">
19       <dependency>
20         <depender aref="05"/>
21         <dependee aref="06"/>
22       </dependency>
23     </ielement>
24     <ielement id="214" name="Proposed schedule" type="resource">
25       <dependency>
26         <depender aref="05"/>
27         <dependee aref="06"/>
28       </dependency>
29     </ielement>
30     <ielement id="216" name="Request authorization" type="task">
31       <dependency>
32         <depender aref="05"/>
33         <dependee aref="09"/>
34       </dependency>
35     </ielement>
36   </diagram>

```

Line	Reason
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	

Figura 4.17: Exemplo de arquivo em iStarML sem erro[21].

Fonte: Elaborada pelo autor, 2016

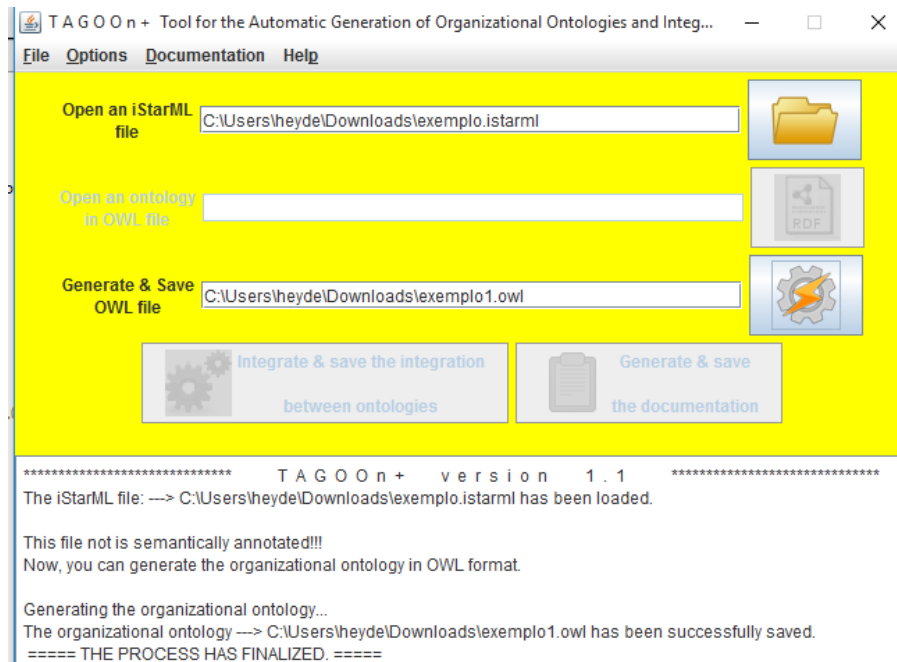


Figura 4.18: *Exemplo do funcionamento do TAGOOn+. Fonte: Elaborada pelo autor, 2016*

Segue na Figura 4.19 um exemplo de arquivo *iStarML* com erro e em seguida a tela de execução do TAGOOn+ com a mensagem de erro. Neste exemplo é apresentado o erro 1 (Atores dentro da fronteira de outro ator), pode ser visto destacado em vermelho que no arquivo apresenta um ator dentro da fronteira de outro ator.

```

1  <?xml version="1.0"?>
2  <istarml version="1.0">
3    <diagram name="Strategy Dependency CENEDEI">
4      <actor id="05" name="Student">
5        <boundary>
6          <ielement id="104" name="Analyze courses" type="service" />
7          <ielement id="105" name="Authorize schedule" type="actor" />
8        </boundary>
9      </actor>
10     <actor id="06" name="Thesis advisor" />
11     <actor id="09" name="Department chair" />
12     <ielement id="212" name="Department chair 1" type="softgoal">
13       <dependency>
14         <depender aref="05" />
15         <dependee aref="06" />
16       </dependency>
17     </ielement>
18     <ielement id="213" name="Choose courses" type="goal">
19       <dependency>
20         <depender aref="05" />
21         <dependee aref="06" />
22       </dependency>
23     </ielement>
24     <ielement id="214" name="Proposed schedule" type="resource">
25       <dependency>
26         <depender aref="05" />
27         <dependee aref="06" />
28       </dependency>
29     </ielement>
30     <ielement id="216" name="Request authorization" type="task">
31       <dependency>
32         <depender aref="05" />
33         <dependee aref="09" />

```

Figura 4.19: Exemplo de arquivo em iStarML com erro. *Fonte:* Elaborada pelo autor, 2016

Na Figura B.5 podemos perceber que o TAGOOn+ recebe o arquivo iStarML com erro e tenta gerar o arquivo OWL, porém quando encontra o erro retorna a mensagem destacando o que está errado no arquivo para que o usuário possa corrigir e tentar gerar novamente o arquivo de forma correta.

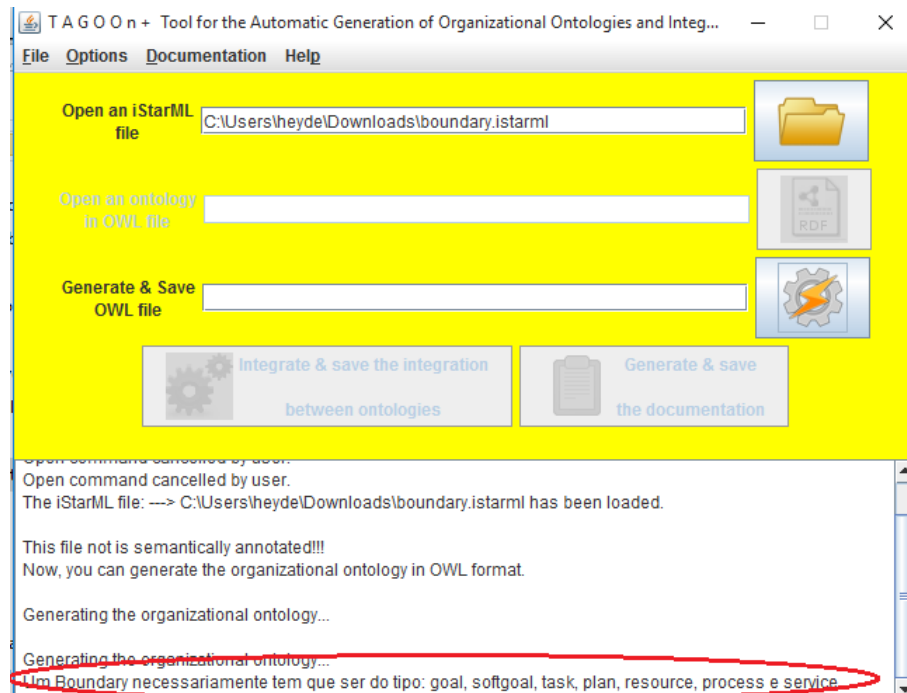


Figura 4.20: Tela do TAGOOn exibindo erro. *Fonte:* Elaborada pelo autor, 2016

Neste exemplo da Figura 4.21 é apresentado o erro 2 (Conexão de um *InternalElement* para um *ActorBoundary*), pode ser visto no arquivo que ele apresenta um *ielementLink* dentro da fronteira de outro ator.

```

1 <?xml version="1.0"?>
2 <istarm version="1.0">
3   <diagram name="Strategy Rationale CENEDEI">
4     <actor id="03" name="Student">
5       <boundary>
6         <ielementLink type="decomposition" value="and"/>
7         <ielement id="66" name="Request authorization of department chair" type="task"/>
8         <ielement id="70" name="Request courses to take" type="task"/>
9         <ielement id="73" name="Exchange Bank receipt" type="task"/>
10        <ielement id="76" name="Take position in queue" type="task"/>
11        <ielement id="77" name="Receive bank receipt" type="task"/>
12        <ielement id="79" name="Pay grant in bank" type="task"/>
13        <ielement id="80" name="register" type="task"/>
14        <ielementLink type="decomposition" value="and">
15          <ielement ref="66"/>
16          <ielement ref="70"/>
17          <ielement ref="73"/>
18          <ielement ref="76"/>
19          <ielement ref="79"/>
20        </ielementLink>
21      </boundary>
22      <ielement id="81" name="Register in master of PHD program" type="goal">
23        <ielementLink type="means-end">
24          <ielement ref="80"/>
25        </ielementLink>
26      </ielement>
27    </actor>
28  </diagram>
29 </istarm>

```

Figura 4.21: Exemplo de arquivo em iStarML com erro. *Fonte:* Elaborada pelo autor, 2016

Nesta Figura 4.22 podemos perceber que o TAGOOn+ recebe o arquivo *iStarML* com erro e tenta gerar o arquivo OWL, porém quando encontra o erro retorna a mensagem destacando o que está errado no arquivo para que o usuário possa corrigir e tentar gerar novamente o arquivo de forma correta.

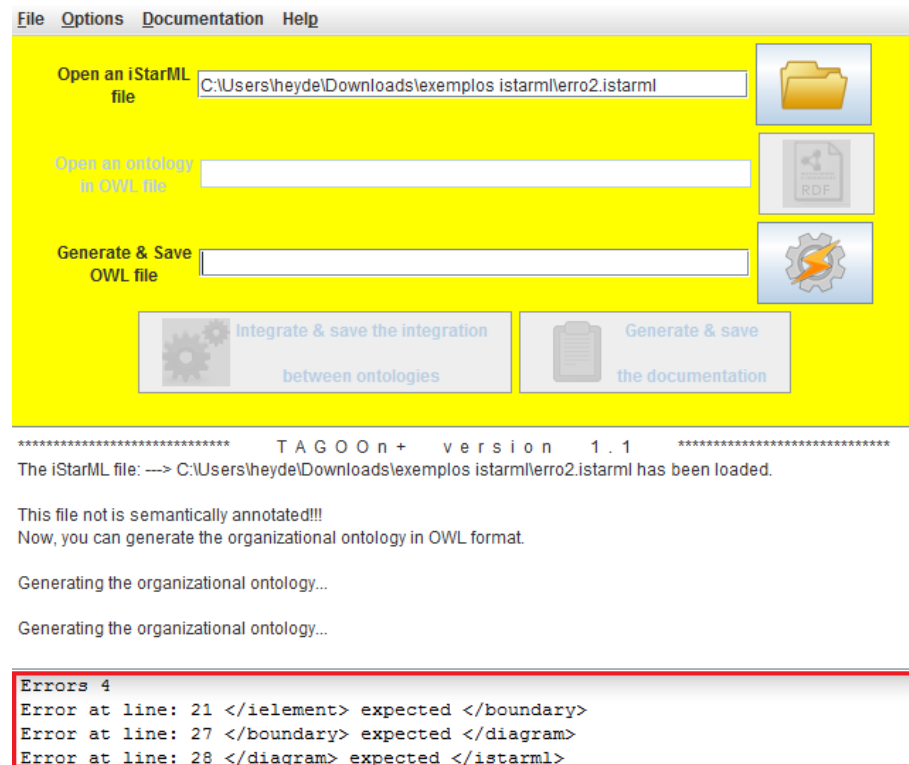


Figura 4.22: Tela do TAGOOn+ exibindo erro. *Fonte:* Elaborada pelo autor, 2016

4.2.3 Resultados Obtidos

Os resultados obtidos na etapa de validação das restrições inseridas no TAGOOn+ foram satisfatórios. Nenhum dos erros tratados na ontologia *OntoiStar+* e posteriormente na ferramenta TAGOOn+ podem ser reproduzidos após a extensão realizada em ambas. De forma que quando é inserido um modelo em *iStarML* contendo algum dos erros relatados na seção de tratamento de erros na ferramenta TAGOOn+, com o auxílio das restrições inseridas, a ferramenta localiza o erro e exibe uma mensagem para o usuário identificando o erro e solicitando que o arquivo em *iStarML* seja corrigido.

No caso do erro 9 (ator sem ligação) ou 1[a] conforme o catálogo de erros ele foi tratado na ferramenta TAGOOn+, assim não é possível gerar modelos com este erro. Os erros não tratados foram apenas aqueles que possuem uma interpretação mais subjetiva, como: uso de nomes inadequados em atores, elementos sem ligações e ligação direta entre elementos internos de dois atores diferentes.

A grande vantagem da realização da extensão desta ferramenta é que a utilizando é possível validar modelos *i** já previamente desenvolvidos em *iStarML*, transformando estes em uma instância válida da *OntoiStar+*. A partir disto então é possível utilizar o *Protégé* e o raciocinador *pellet* para continuar validando possíveis incrementos e alterações no modelo.

Conclusões e Trabalhos Futuros

A linguagem *i** traz algumas vantagens para a fase de modelagem de requisitos e vem sendo muito utilizada no meio acadêmico e bem aceita pela comunidade de engenharia de requisitos, porém, devido à má interpretação de seus construtores, existem erros que podem acontecer com frequência em modelos gerados. Tendo em vista que esses erros podem prejudicar a qualidade dos modelos gerados utilizando a linguagem *i**, nesta dissertação foi apresentada uma solução baseada em ontologia e restrições OWL com o objetivo de minimizar a quantidade de erros frequentes de modelos *i**.

Este trabalho estende a ontologia *OntoiStar+* e a ferramenta *TAGOOOn+* com axiomas e restrições visando a reduzir a quantidade de erros frequentes de modelos *i**, erros estes discutidos amplamente na literatura[33], [37] e [38].

Desta forma, a extensão realizada na ontologia *OntoiStar+* verifica a consistência dos modelos *i** à medida em que são criados através da utilização das restrições OWL inseridas na mesma, não permitindo a criação de modelos com os erros mencionados. Soma-se ainda a vantagem dessa solução também suportar a interoperabilidade entre variantes da linguagem *i**.

Foi realizada adicionalmente a integração dessa versão estendida da ontologia *OntoiStar+* com a ferramenta *TAGOOOn+* [21], [29], que permite transformar modelos *i** escritos na linguagem *iStarML* em modelos baseados na estrutura e semântica da ontologia *OntoiStar+*. A ferramenta *TAGOOOn+* foi estendida para incorporar as novas regras incluídas na *OntoiStar+*, o que traz como benefício principal a verificação automática de erros de modelos *i** em *iStarML*.

Como resultado do trabalho, apenas 4 dos 15 não foram contemplados pela extensão da *OntoiStar+* que, segundo a Figura 1, são: 1(a, c) e 3(a, c). Isto se deve ao fato desses erros serem mais subjetivos (por ex. uso de nome inadequado para ator) e/ou mais difíceis de tratar apenas com a expressividade dos construtores da linguagem OWL. Vale ressaltar que o erro 1(a) do catálogo de erros foi tratado parcialmente, pois a regra garante que os relacionamentos estarão corretos, mas não garante que no modelo não existirá um ator sem ligação.

5.1 Limitações

Apesar desta dissertação ter trazido alguns resultados significativos, pode-se perceber algumas limitações, como:

- Necessidade de uma validação da ontologia *OntoiStar+* estendida por especialistas em *i** e ontologias;
- Realização de um estudo sobre o catálogo de erros a fim de verificar se existem erros além dos já catalogados;
- Estudo de uma alternativa para tratar os erros que não puderam ser mitigados.

5.2 Trabalhos Futuros

Em função das limitações apresentadas neste trabalho, pretende-se realizar como trabalhos futuros:

- a utilização de regras SWRL (*Semantic Web Rule Language*)[11] para contemplar os erros que não foram possíveis de tratar com restrições OWL. SWRL é uma linguagem que estende a capacidade de expressão da linguagem OWL, bem como a sua capacidade de inferência;
- viabilizar a validação da ontologia *OntoiStar+* por especialistas;
- realizar um estudo em modelos *i** para verificar a existência de erros frequentes ainda não listados no catálogo de erros.

5.3 Considerações Finais

A solução proposta nesta dissertação atendeu ao seu objetivo principal que era criar uma solução ontológica para reduzir os erros frequentes em modelos *i**.

Para verificação da eficácia da solução, foram realizados testes utilizando os domínios *Media Shop* e de Universidades. A solução se mostrou satisfatória tratando 70% dos erros frequentes relatados na literatura. Vale ressaltar que os erros que não foram atendidos são aqueles que tratam questões mais subjetivas, o que torna difícil criar uma restrição para validá-lo.

5.4 Publicações Relacionadas

Resultados parciais da extensão da Ontologia *OntoiStar+* foram relatados em um artigo [13] submetido e aprovado no XIX Congresso Iberoamericano da Engenharia de Software (Cibse2016), mais especificamente no WER 2016 que é a décima nona

edição do Workshop em Engenharia de Requisitos (Qualis B3). O trabalho completo desenvolvido nesta dissertação, incluindo a extensão da *OntoiStar+* e da *TAGOOOn+* e a validação nos dois domínios citados, será submetido para o *Journal of Computer Science* (<http://thescipub.com/journals/jcs/>) em inglês.

Referências Bibliográficas

- [1] ALMEIDA-JUNIOR, O. A. **Engenharia de requisitos para sistemas auto-adaptativos**. Master's thesis, 2013. Universidade Federal de Pernambuco - Centro de informática.
- [2] BENCOMO, N.; WHITTLE, J.; SAWYER, P.; FINKELSTEIN, A.; LETIER, E. **Requirements reflection: requirements as runtime entities**. In: *Software Engineering, ACM/IEEE 32nd International Conference on*, volume 2, p. 199–202, May 2010.
- [3] BREITMAN, K. G. **Web Semântica : A Internet do Futuro**. LTC, 2011.
- [4] CARES, C. **From the i* Diversity to a Common Interoperability Framework**. PhD thesis, Universitat Politècnica de Catalunya-Barcelona Tech, 2012.
- [5] CARLOS CARES, XAVIER FRANCH, A. P. E. A. S. **istarml the i* mark-up language: References guide**, 2007.
- [6] CASTRO, J.; ALENCAR, F. **Uso de modelagem social na engenharia de requisitos**. Jornada de Atualização de Informática - JAI 2013. Maceió-AL, Brazil, 2013.
- [7] CASTRO, J., K. M. M. J. **Towards requirements-driven information systems engineering: the tropos project**. *Information Systems Elsevier Science Ltd. Oxford, UK*, 27(6):365–389, 2002.
- [8] CHUNG, L., N. B. Y. E.; MYLOPOULOS, J. **Non-Functional Requirements in Software Engineering**. Springer US, 2011.
- [9] CONEJERO, J. M.; FIGUEIREDO, E.; GARCIA, A.; HERNÁNDEZ, J.; JURADO, E. **On the relationship of concern metrics and requirements maintainability**. *Information and Software Technology*, 54(2):212 – 238, 2012.
- [10] CUESTA, L. L. **The Notion of Specialization in the I* Framework**. PhD thesis, Universitat Politècnica de Catalunya-Barcelona Tech, Barcelona, 2013.
- [11] DERMEVAL D., VILELA J., B. I. I. C. J. I. S. B. P. S. A. **Applications of ontologies in requirements engineering:a systematic review of the literature**. *Requirements Engineering (London. Print)*, v. 1, p. 1-33, 2015.

- [12] DURAN-LIMON, H. A.; CASTILLO-BARRERA, F. E.; LOPEZ-HERREJON, R. E. **Towards an ontology-based approach for deriving product architectures.** In: *Proceedings of the 15th International Software Product Line Conference, Volume 2, SPLC '11*, p. 29:1–29:5, New York, NY, USA, 2011. ACM.
- [13] FRANCA, H. F. C.; BULCAO-NETO, R. **Uma solução baseada em ontologia para o tratamento de erros em modelos construídos na linguagem i*.** XIX Congresso Iberoamericano da Engenharia de Software (Cibse2016), 2016.
- [14] FRANCH, X.; MATE, A.; TRUJILLO, J.; CARES, C. **On the joint use of i* with other modelling frameworks: A vision paper.** In: *Requirements Engineering Conference (RE), 19th IEEE International*, p. 133–142, Aug 2011.
- [15] GIANCARLO GUIZZARDI, XAVIER FRANCH, R. G.; WIERINGA, R. **Using a foundational ontology to investigate the semantics behind the concepts of the i* language.** *Proceedings of the 6th International i* Workshop (iStar 2013)*, CEUR Vol-978, p. 13–18, Jun 2013.
- [16] GRAU, G.; CARES, C.; FRANCH, X.; NAVARRETE, F. **A comparative analysis of i*agent-oriented modelling techniques.** In: *Proceedings of the Eighteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2006)*, San Francisco, CA, USA, July 5-7, 2006, p. 657–663, 2006.
- [17] GRUBER, T. R. **A translation approach to portable ontology specifications.** *Knowl. Acquis.*, 5(2):199–220, June 1993.
- [18] GUIZZARDI, G. **Ontological foundations for structural conceptual models.** PhD thesis, Enschede, 2005.
- [19] GUIZZARDI, G.; AO PAULO ALMEIDA, J.; GUIZZARDI, R. S. S.; BARCELLOS, M. P.; FALBO, R. **Ontologias de fundamentação, modelagem conceitual e interoperabilidade semântica**, 2011. Iberoamerican Meeting of Ontological Research, p. paper 6.
- [20] GUIZZARDI, R.; FRANCH, X.; GUIZZARDI, G. **Applying a foundational ontology to analyze means-end links in the i* framework.** In: *Research Challenges in Information Science (RCIS), 2012 Sixth International Conference on*, p. 1–11, May 2012.
- [21] HERNANDEZ, K. M. N. **An ontology-based approach for integrating i* variants.** Master's thesis, 2011. Centro Nacional de Investigación y Desarrollo Tecnológico Departamento de Computación - Cuernavaca, Morelos, México.

- [22] HORKOFF, J.; ELAHI, G.; ABDULHADI, S.; YU, E. **Reflective analysis of the syntax and semantics of the i* framework**. In: Song, I.; Piattini, M.; Chen, Y.-P.; Hartmann, S.; Grandi, F.; Trujillo, J.; Opdahl, A.; Ferri, F.; Grifoni, P.; Caschera, M.; Rolland, C.; Woo, C.; Salinesi, C.; Zimanyi, E.; Claramunt, C.; Frasincar, F.; Houben, G.-J.; Thiran, P., editors, *Advances in Conceptual Modeling Challenges and Opportunities*, volume 5232 de **Lecture Notes in Computer Science**, p. 249–260. Springer Berlin Heidelberg, 2008.
- [23] HORKOFF, J., Y. E. **Detecting judgment inconsistencies to encourage model iteration in interactive i* analysis**. *Fifth International i* Workshop*, p. 20–25, 2011.
- [24] JENNIFER HORKOFF, ERIC YU, G. G. **i* guides**, 2006.
- [25] LAPOUCHNIAN, A. **Goal-Oriented Requirements Engineering: An Overview of the Current Research**. Technical report, Department of Computer Science, University of Toronto, Toronto, Canada, Juni 2005.
- [26] LEAL, A. L. C. **Análise de Conformidade de Software com Base em catálogos de Requisitos não funcionais: Uma abordagem baseada em sistemas multi agentes**. PhD thesis, PUC- Rio, Rio de Janeiro Brasil, 2014.
- [27] LOSCIO, B. **Dados, integração de dados e dados interligados**. Workshop de Introdução a Engenharia de Ontologias e Web Semantica, UFPE(2012), 2012.
- [28] MALTA, A., E. A. **istartool: Modeling requirements using the i* framework**. *CEUR Workshop Proceedings*, p. 163–165, 2011.
- [29] NAJERA K., MARTINEZ, A. P. A. E. H. **An ontology-based methodology for integrating i* variants**. *Sixth International i* Workshop*, p. 1–, 2013.
- [30] NARDI, J. C.; DE ALMEIDA FALBO, R. **Uma ontologia de requisitos de software**. In: Castro, J.; Cernuzzi, L.; Gordillo, S. E., editors, *ClbSE*, p. 111–124, 2006.
- [31] NOY, N. F.; MCGUINNESS, D. L. **Ontology development 101: A guide to creating your first ontology**. <http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html>, 2001.
- [32] SANTOS, B. S. **Istar tool - uma proposta de ferramenta para modelagem de i***. Master's thesis, 2008. Universidade Federal de Pernambuco - Centro de informática.
- [33] SANTOS, E. B. **Uma proposta de métricas para avaliar modelos i***. Master's thesis, 2008. Universidade Federal de Pernambuco - Centro de informática.

- [34] SIRIN, E.; PARSIA, B.; GRAU, B. C.; KALYANPUR, A.; KATZ, Y. **Pellet: A practical owl-dl reasoner**. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2), 2007.
- [35] SMITH, M., M. D. L. **Owl web ontology language**, 2014. Guide. <http://www.w3.org/TR/2004/REC-owl-guide-20040210>.
- [36] SOMMERVILLE, I. **Software Engineering 9**. Pearson Education, 2011.
- [37] SOUZA, C.; SOUZA, C.; ALENCAR, F.; CASTRO, J.; CAVALCANTI, P.; SOARES, M.; GUEDES, G.; FIGUEIREDO, E. **Avaliação de modelos i* com o processo airdoc-i***. <http://ceur-ws.org/Vol-1005/erbr2013-submission-23.pdf>, 2013.
- [38] SOUZA, C.; SOUZA, C.; ALENCAR, F.; CASTRO, J.; SOARES, M.; GUEDES, G. **Airdoc-i*: um processo para avaliação de modelos i***. 15th Workshop em Engenharia de Requisitos (WER), 2012.
- [39] TALLABACI, G.; SILVA SOUZA, V. **Engineering adaptation with zanshin: An experience report**. In: *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), ICSE Workshop on*, p. 93–102, May 2013.
- [40] TEIXEIRA, M. **Owl -(web ontology language)**. <http://www3.ceunes.ufes.br/downloads/2/mariateixeira-EC.Introdu2011>.
- [41] VAN LAMSWEERDE, A. **Goal-oriented requirements engineering: a guided tour**. In: *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, p. 249–262, 2001.
- [42] VAN LAMSWEERDE, A. **Requirements engineering in the year 00: A research perspective**. In: *Proceedings of the 22Nd International Conference on Software Engineering, ICSE '00*, p. 5–19, New York, NY, USA, 2000. ACM.
- [43] WERNECK, V. M. B.; DE PADUA ALBUQUERQUE OLIVEIRA, A.; DO PRADO LEITE, J. C. S. **Comparing gore frameworks: i-star and kaos**. In: *Ibero-American Workshop of Engineering of Requirements*, Val Paraiso, Chile, July 2009.
- [44] YU, E. S. **Modelling strategic relationships for process reengineering**. PhD thesis, 1996. <http://portal.acm.org/citation.cfm?id=269793>.
- [45] YU, Y.; LEITE, J.; MYLOPOULOS, J. **From goals to aspects: discovering aspects from requirements goal models**. In: *Requirements Engineering Conference. Proceedings. 12th IEEE International*, p. 38–47, Sept 2004.

Catálogo de erros típicos de modelos i*

Seguem os erros típicos de modelos i* levantados em [33]:

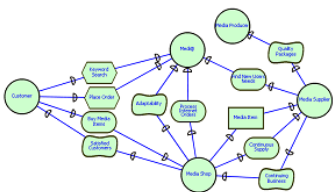

01 - Dangling Actor	Actor without link to another Actor
Details	An Actor without link to another Actor does not contribute with information to model
Questions	Are there Actors without dependencies or structural actor's links (i.e., Is-A, Is-part-of, Covers, Occupies, Plays)?
Correct	Wrong
	

Figura A.1: *Dangling Actor*

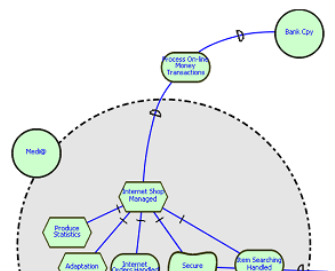
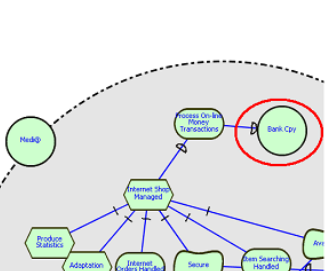
02 - Internal Actor	Do not include an Actor within another Actor
Details	Actors are active and autonomous entities that should be modeled separately. Only intentional elements can be inside an actor (eg. Goal, Softgoal, Task or Resource)
Questions	Is there more than one actor inside the same boundary?
Correct	Wrong
	

Figura A.2: *Internal Actor*

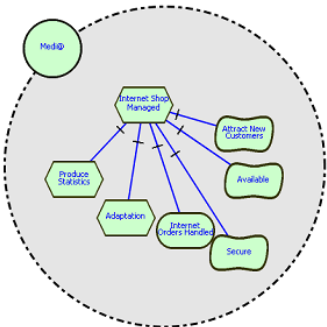
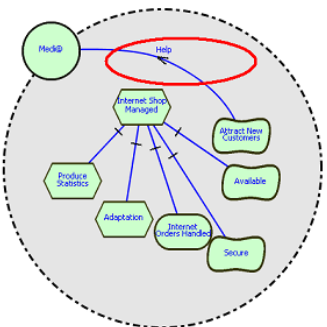
03 - Actor Boundary	Internal element connection to Actor boundary
Details	Do not connect internal elements (goals, tasks, etc.) to the actor boundary
Questions	Is there any internal element connected to the actor boundary?
Correct	Wrong
	

Figura A.3: Actor Boundary

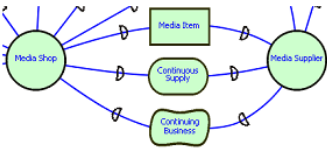
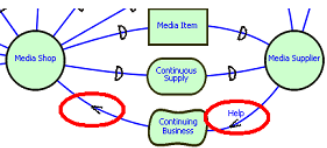
04 - Actor Dependency	Use only the dependency symbol (D) to denote a dependency link
Details	Do not use other types of links (Means-end, Contribution, Decomposition) to denote dependency links.
Questions	Is there any Actor Dependency which does not use the Dependency symbol?
Correct	Wrong
	

Figura A.4: Actor Dependency



05 - Softgoal	Softgoals are distinct from Goals
Details	Softgoals are special Goals that have no clear cut achievement state. There are different levels of satisfaction (e.g., Satisfied, Partially Satisfied, Denied, etc.) They are used to capture quality issues or constraints such as: security, performance, etc.
Questions	Is there any Goal with adverbs or adjectives in their label? The condition expressed in label could be partially satisfied or not completely achieved?
Correct	Wrong
	

Figura A.5: Softgoal

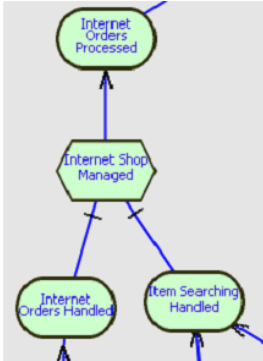
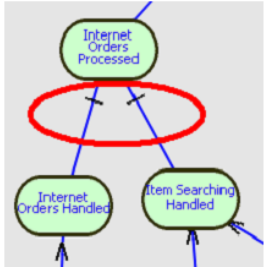
06 - Goal Refinement	Linking of Goals in SR models
Details	In SR models: <ul style="list-style-type: none"> - Goals cannot be directly linked to other Goals - Goals cannot be present at both ends of Decomposition, Contribution, or Means-Ends links
Questions	Are there Goals linked to other Goals?
Correct	Wrong
	

Figura A.6: Goal Refinement

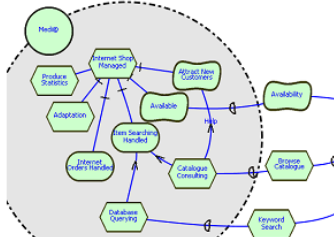
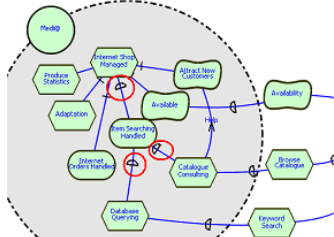
07 - Internal Dependency Link	Do not use dependency links inside an actor
Details	Dependency links cannot be used to connect two intentional elements inside the same actor. It can be used to connect an internal element to a dependum, but not to connect two internal elements.
Questions	Is there any internal element that is connected to an other internal element using a dependency link?
Correct	Wrong
	

Figura A.7: Internal Dependency Link

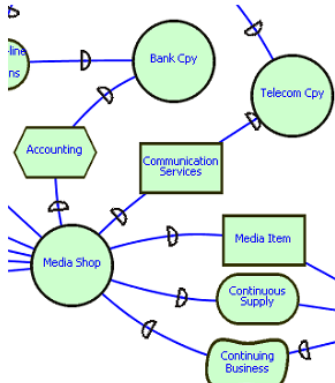
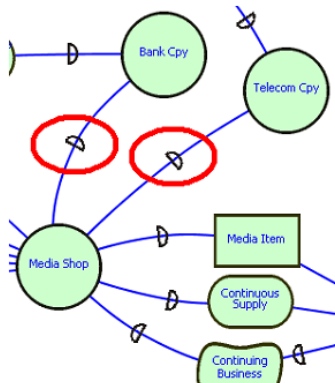
08 - External Dependency Link	Do not use a dependency link between two actors without showing the dependum
Details	The Dependency link should contain a Dependum. Extending the Dependency Link from the Depender to the Dependee without showing the Dependum does not convey what the dependency is about.
Correct	Wrong
	

Figura A.8: *External Dependency Link*

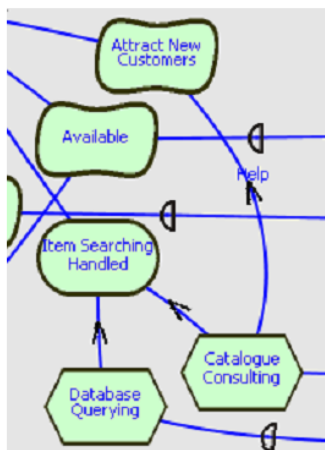

09 - Internal Contribution Link	A contribution link should only be used to connect from an intentional element (of any type) to a softgoal
Details	Contribution links are not allowed from any element to a goal, only to softgoals
Questions	Is there any Goal receiving Contribution links?
Correct	Wrong
	

Figura A.9: *Internal Contribution Link*

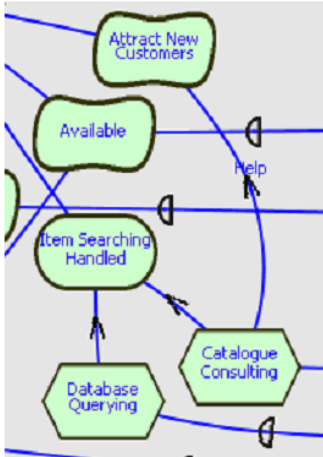
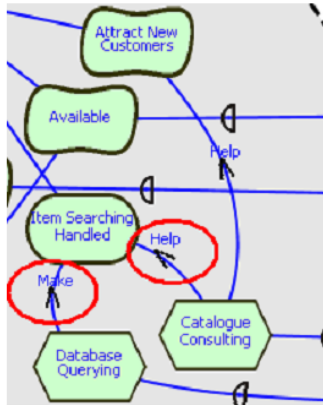
10 - External Contribution Link	Do not use Contribution Links between actors
Details	Contribution is present in SR models and need be drawn inside actor's boundary.
Questions	Is there any contribution link to element outside actor's boundary? Is there any contribution link between an internal element and an element outside actor's boundary?
Correct	Wrong
	

Figura A.10: *External Contribution Link*

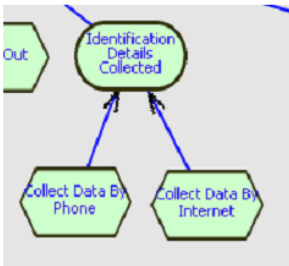
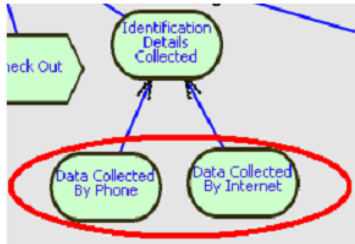
11 - Internal Means-End Link	Means-Ends are only used to link a Task to a Goal
Details	The only place where a Means-Ends link can be used is from a Task to a Goal
Questions	Is there any Goal as Means in a Means-ends link?
Correct	Wrong
	

Figura A.11: *Internal Means-End Link*

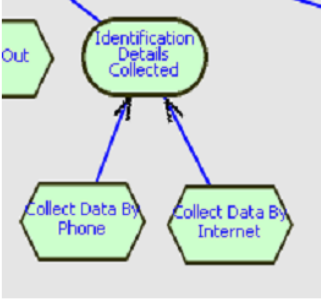
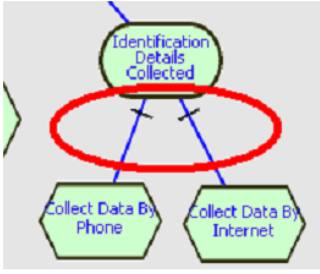
12 - External Means-End Link	Do not extend Means-Ends link beyond the boundaries of actors
Details	Means-Ends Link need to be drawn within an actor and not between actors
Questions	Is there any Means-Ends link outside actor's boundary? Is there any Means-Ends link between an internal element and an element outside actor's boundary?
:browse confirm wa Correct	Wrong
	

Figura A.12: *External Means-End Link*

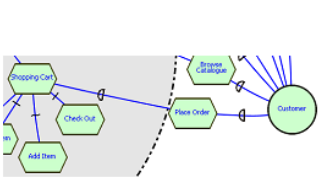
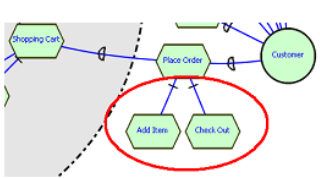
13 - External Decomposition Link	Do not extend Decomposition links beyond the boundaries of actors.
Details	Decomposition Links need be drawn within an actor, and not between actors
Questions	Is there any Decomposition link outside actor's boundary? Is there any Decomposition link between a Task inside actor's boundary and an external element?
Correct	Wrong
	

Figura A.13: *External Decomposition Link*

Arquivos utilizados para testes da ferramenta *TAGOO++*

Seguem os arquivos em *iStarML* utilizados para realizar os testes da ferramenta *TAGOO++*. São apresentados abaixo os modelos de requisitos representados em *istarmml* de 2 domínios, o domínio do *Media Shop* e o domínio de Universidade que Karen Najera [21] também utilizou na sua dissertação:

```
1 <?xml version="1.0"?>
2 <istarmml version="1.0">
3   <diagram name="Strategy Dependency CENEDEF">
4     <actor id="05" name="Student"/>
5     <actor id="06" name="Thesis advisor"/>
6     <actor id="09" name="Department chair"/>
7     <ielement id="212" name="Choose appropriated courses" type="softgoal">
8       <dependency>
9         <depender aref="05"/>
10        <dependee aref="06"/>
11      </dependency>
12    </ielement>
13    <ielement id="213" name="Choose courses" type="goal">
14      <dependency>
15        <depender aref="05"/>
16        <dependee aref="06"/>
17      </dependency>
18    </ielement>
19    <ielement id="214" name="Proposed schedule" type="resource">
20      <dependency>
21        <depender aref="05"/>
22        <dependee aref="06"/>
23      </dependency>
24    </ielement>
25    <ielement id="216" name="Request authorization" type="task">
26      <dependency>
27        <depender aref="05"/>
28        <dependee aref="09"/>
29      </dependency>
30    </ielement>
31  </diagram>
```

Figura B.1: Modelo usando dependências (SD)

```

1  <?xml version="1.0"?>
2  <istaml version="1.0">
3    <diagram name="Strategy Rationale CENEDEI">
4      <actor id="03" name="Student">
5        <boundary>
6          <ielement id="66" name="Request authorization of department chair" type="task">
7            <ielement id="70" name="Request courses to take" type="task">
8              <ielement id="73" name="Exchange Bank receipt" type="task">
9                <ielement id="76" name="Take position in queue" type="task">
10               <ielement id="77" name="Receive bank receipt" type="task">
11               <ielement id="79" name="Pay grant in bank" type="task">
12               <ielement id="80" name="register" type="task">
13               <ielementLink type="decomposition" value="and">
14                 <ielement iref="66">
15                 <ielement iref="70">
16                 <ielement iref="73">
17                 <ielement iref="76">
18                 <ielement iref="79">
19               </ielementLink>
20             </ielement>
21             <ielement id="81" name="Register in master of PHD program" type="goal">
22               <ielementLink type="means-end">
23               <ielement iref="80">
24             </ielementLink>
25             </ielement>
26           </boundary>
27         </actor>
28       </diagram>
29     </istaml>

```

Figura B.2: Modelo usando *ielement* e *ielementLink*(SR)

```

1  <?xml version="1.0"?>
2  <istaml version="1.0">
3    <diagram name="Strategy Dependency CENEDEI">
4      <actor id="05" name="Student">
5      <actor id="06" name="Thesis advisor">
6      <actor id="09" name="Department chair">
7      <ielement id="212" name="Choose appropriated courses" type="softgoal">
8        <dependency>
9          <depender aref="05">
10          <dependee aref="06">
11        </dependency>
12      </ielement>
13      <ielement id="213" name="Choose courses" type="goal">
14        <dependency>
15          <depender aref="05">
16          <dependee aref="06">
17        </dependency>
18      </ielement>
19      <ielement id="214" name="Proposed schedule" type="resource">
20        <dependency>
21          <depender aref="05">
22          <dependee aref="06">
23        </dependency>
24      </ielement>
25      <ielement id="216" name="Request authorization" type="plan">
26        <dependency>
27          <depender aref="05">
28          <dependee aref="09">
29        </dependency>
30      </ielement>
31    </diagram>
32  </istaml>

```

Figura B.3: Modelo usando dependências e construtores do Tropos

```

1 <?xml version="1.0"?>
2 <istarml version="1.0">
3   <diagram name="goal model CENEDEI">
4     <actor id="03" name="Student">
5       <boundary>
6         <ielement id="66" name="Request authorization of department chair" type="plan">
7           <ielement id="70" name="Request courses to take" type="plan">
8             <ielement id="73" name="Exchange Bank receipt" type="plan">
9               <ielement id="76" name="Take position in queue" type="plan">
10                <ielement id="77" name="Receive bank receipt" type="plan">
11                  <ielement id="79" name="Pay grant in bank" type="plan">
12                    <ielement id="80" name="register" type="plan">
13                      <ielementLink type="decomposition" value="and">
14                        <ielement iref="66">
15                          <ielement iref="70">
16                            <ielement iref="73">
17                              <ielement iref="76">
18                                <ielement iref="79">
19                                  </ielementLink>
20                                </ielement>
21                              <ielement id="81" name="Register in master of PHD program" type="goal">
22                                <ielementLink type="means-end">
23                                  <ielement iref="80">
24                                    </ielementLink>
25                                  </ielement>
26                                </ielement>
27                              </ielement>
28                            </ielement>
29                          </ielement>
30                        </ielementLink>
31                      </ielement>
32                    </ielement>
33                  </ielement>
34                </ielement>
35              </ielement>
36            </ielement>
37          </ielement>
38        </boundary>
39      </actor>
40    </diagram>
41  </istarml>

```

Figura B.4: Modelo usando *ielement* e *ielementLink* e construtores do Tropos

```

1 <?xml version="1.0"?>
2 <istarml version="1.0">
3   <diagram name="Global model CENEDEI">
4     <actor id="05" name="Student">
5       <actor id="06" name="Thesis advisor">
6         <boundary>
7           <ielement id="104" name="Analyse courses" type="service">
8             <ielement id="105" name="Authorize schedule" type="service">
9               <ielement id="106" name="Propose courses" type="service">
10                </ielement>
11              </ielement>
12            </ielement>
13          </boundary>
14        </actor>
15        <ielement id="304" name="Choose courses" type="goal">
16          <dependency>
17            <depender aref="05">
18              <dependee aref="06" iref="104">
19                </dependency>
20              </dependee>
21            </depender>
22          </dependency>
23        </ielement>
24        <ielement id="305" name="Authorize schedule" type="goal">
25          <dependency>
26            <depender aref="05">
27              <dependee aref="06" iref="105">
28                </dependency>
29              </dependee>
30            </depender>
31          </dependency>
32        </ielement>
33      </diagram>
34    </istarml>

```

Figura B.5: Dependência de Serviço no modelo global