

UNIVERSIDADE FEDERAL DE PERNAMBBUCO

CENTRO DE INFORMÁTICA

GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

**AVALIAÇÃO DA EFICÁCIA DA FERRAMENTA AGILE TOOL PARA CRIAR FERRAMENTAS DE SUPORTE ÀS EXTENSÕES DA LINGUAGEM I\***

Aline Éllida D’Oleron Vasconcelos

Trabalho de Graduação

Recife

MAIO DE 2013

UNIVERSIDADE FEDERAL DE PERNAMBBUCO

CENTRO DE INFORMÁTICA

Aline Éllida D’Oleron Vasconcelos

**Avaliação da eficácia da ferramenta AGILE Tool para criar ferramentas de suporte às extensões da linguagem i\***

*Trabalho apresentado ao Programa de GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO do CENTRO DE INFORMÁTICA da UNIVERSIDADE FEDERAL DE PER-NAMBUCO como requisito parcial para obtenção do grau de Bacharel em ENGENHARIA DA COMPUTAÇÃO.*

Orientadora: Profa. Dra. Carla Taciana L. L. Silva Schuenemann

**Recife**

**MAIO DE 2013**

**AGRADECIMENTOS**

Agradeço primeiramente a Deus pelo seu infinito amor por mim.

Agradeço de todo o coração

Aos meus pais, Amauri e Lúcia, e a minha irmã Lianne por todo apoio e por serem meus grandes exemplos.

Ao meu namorado e melhor amigo, Filipe, por estar sempre comigo e por estar pronto para me ajudar sempre que preciso.

A minha orientadora, professora Carla Taciana Lima Lourenço Silva Schuenemann, por sua simpatia, paciência e por todo seu o tempo dedicado a me auxiliar durante o trabalho.

Ao professor Josias Paes pela sua prestatividade sempre que foi necessário.

E aos meus amigos e colegas pelo apoio e companheirismo.

**RESUMO**

Falhas ocorridas durante os processos de Engenharia de Requisitos podem afetar a fase de produção e provavelmente a própria vida útil dos *softwares.*Aelicitação, análise e especificação de requisitos, fases iniciais do desenvolvimento de um projeto de *software*, são essenciais para que este seja sistemático e disciplinado, diminuindo assim os riscos de falhas no seu desenvolvimento. A Engenharia de Requisitos Orientada a Objetivos tem como finalidade modelaro problema e basear-se nos objetivos organizacionais para definir os requisitos do sistema a ser desenvolvido, a partir da captura das necessidades dos stakeholders. O framework i\* é uma das diversas abordagens propostas para especificação de requisitos orientada a objetivos. Várias linguagens foram propostas baseadas no i\* e cada uma delas busca satisfazer necessidades específicas de um projeto.Assim, cada uma destas linguagens necessitade ferramentas específicas para sua modelagem gráfica, ocasionando um aumento de custo de desenvolvimento.A abordagem AGILE (Automatic Generation of i\* Languages) propõe um processo e uma ferramenta CASE (Computer-Aided Software Engineering), chamada Agile Tool. Esta ferramenta permite a configuração do metamodelo núcleo, de forma a criar linguagens baseadas no i\* e a geração automática de seus respectivos editores gráficos, diminuindo assim o esforço na construção destes. Até o presente momento a ferramenta havia sido aplicada na construção do editor gráfico para a linguagem i\* Aspectual, porém, é preciso aplicá-la a outras linguagens baseadas no i\* para que seja possível avaliar a eficácia da ferramenta na construção dos seus respectivos editores gráficos. As linguagens escolhidas nesta avaliação foram: i\*-C, i\*-Orthogonal e o i\* com concerned objects.Este trabalho, portanto, apresenta a avaliação da eficácia da ferramenta Agile Tool, uma ferramenta que dá suporte às principais etapas de criação de editores gráficos de linguagens baseadas no i\*.

Sumário

[Capítulo 1 – Introdução: 8](#_Toc354575407)

[1.1.Considerações Iniciais 8](#_Toc354575408)

[1.2.Motivação 9](#_Toc354575409)

[1.3. Objetivos 9](#_Toc354575410)

[Geral: 9](#_Toc354575411)

[Específicos: 10](#_Toc354575412)

[1.4. Estrutura do documento 10](#_Toc354575413)

[Capítulo 2 – Fundamentos teóricos 11](#_Toc354575414)

[2.1. Engenharia de Requisitos 11](#_Toc354575415)

[2.2. Engenharia de Requisitos Orientada a Objetivos 12](#_Toc354575416)

[2.3.Framework i\* 12](#_Toc354575417)

[2.3.1. SD *Model* 14](#_Toc354575418)

[2.3.2. SR *Model* 16](#_Toc354575419)

[2.4. Variantes do i\* 20](#_Toc354575420)

[2.4.1. I\*-c 20](#_Toc354575421)

[2.4.2. I\*-Orthogonal 21](#_Toc354575422)

[2.4.3. I\* com concerned object 21](#_Toc354575423)

[2.5. Abordagem Agile 21](#_Toc354575424)

[2.6. Agile Tool 25](#_Toc354575425)

[2.7. Definição dos metamodelos das linguagens variantes do i\* 30](#_Toc354575426)

[2.7.1. I\*-c 30](#_Toc354575427)

[2.7.2. I\*-orthogonal 34](#_Toc354575428)

[2.7.3. I\* com concerned objects 37](#_Toc354575429)

[Capítulo 3 – Avaliação da eficácia do Agile Tool 41](#_Toc354575430)

[3.1. Utilizando as variantes 42](#_Toc354575431)

[Capítulo 4 – Conclusão 51](#_Toc354575432)

[4.1. Considerações finais 51](#_Toc354575433)

[4.2. Contribuições 51](#_Toc354575434)

[4.3. Trabalhos Futuros 52](#_Toc354575435)

[Referências 53](#_Toc354575436)

# Lista de Figuras

[Figura 2.1 - Especializações de atores e relacionamento entre atores 14](#_Toc354590423)

[Figura 2.2 - Dependência entre atores 14](#_Toc354590424)

[Figura 2.3 - Tipos de relacionamentos entre atores no i\* 15](#_Toc354590425)

[Figura 2.4 - Graus de dependência em i\* 16](#_Toc354590426)

[Figura 2.5 - Ator e sua frontera 17](#_Toc354590427)

[Figura 2.6- Tipos de ligação meio-fim 17](#_Toc354590428)

[Figura 2.7- Tipos de decomposição de tarefas 18](#_Toc354590429)

[Figura 2.8- Metamodelo núcleo para a linuagem i\* original 20](#_Toc354590430)

[Figura 2.9 - Framework GMF e dependências 23](#_Toc354590431)

[Figura 2.10 - Processo de criação de um editor gráfico com o framework GMF utilizando a notação BPMN 24](#_Toc354590432)

[Figura 2.11 - Processo de criação de um editor gráfico utilizando a abordagem AGILE 25](#_Toc354590433)

[Figura 2.12 - Diagrama classes mostrando os elementos de modelagem 26](#_Toc354590434)

[Figura 2.13 - Tela principal do Agile Tool para a configuração de uma base i\* 27](#_Toc354590435)

[Figura 2.14 - Diagrama de classes resumido para a criação de novos elementos de modelagem 28](#_Toc354590436)

[Figura 2.15 - Tela de configuração dos novos elementos de modelagem 29](#_Toc354590437)

[Figura 2.16 - Tela do Agile Tool para configuração de restrições 30](#_Toc354590438)

[Figura 2.17 - Elementos do tipo tarefa ou recurso com cardinalidade [m..n], [0..1] e [1..1] 31](#_Toc354590439)

[Figura 2.18 - Cardinalidade nas ligações do tipo means-end 31](#_Toc354590440)

[Figura 2.19 - Metamodelo para a linguagem i\*-C 33](#_Toc354590441)

[Figura 2.20 - Metamodelo da linguagem de variabilidade para modelo de objetivos i\*-Ortogonal (LIMA, 2011) 35](#_Toc354590442)

[Figura 2.21 - Metamodelo para a linguagem Tropos 39](#_Toc354590443)

[Figura 2.22 Modelo de concerned object (REBOLLAR, 2008). 40](#_Toc354590444)

[Figura 2.23 - Metamodelo para a linguagem i\* com concerned objects 41](#_Toc354590445)

[Figura 3.1 - Metamodelo para a linguagem i\*-C adaptado para o Agile Tool 43](#_Toc354590446)

[Figura 3.2 Configuração do novo elemento *Cardinality* 44](#_Toc354590447)

[Figura 3.3 - Configuração do novo elemento *CardinalityLink* 45](#_Toc354590448)

[Figura 3.4 - Configuração das restrições da ligação *CardinalityLink* 46](#_Toc354590449)

[Figura 3.5 - Editor gráfico gerado para o i\*-C com o elemento de cardinalidade e uma relação de dependência de recurso entre atores 47](#_Toc354590450)

Figura 3.6 - Metamodelo para a linguagem i\*-Orthogonal adaptado para o Agile Tool……..48

[Figura 3.7 - Editor gráfico gerado para o i\*-Orthogonal com o elemento de contexto ligado a uma ligação de dependência de objetivo entre atores 48](#_Toc354590451)

[Figura 3.8 - Editor gráfico gerado para o i\*com concerned obects com o elemento de Concerned object e seus atributos e uma relação de dependência de recurso entre atores 49](#_Toc354590452)

# Lista de Tabelas

[Tabela 2.1 - Restrições existentes na linguagem i\*-C 32](#_Toc354590453)

[Tabela 2.2 - Restrições existentes na linguagem i\*-Orthogonal 36](#_Toc354590454)

[Tabela 2.3 - Restrições existentes na linguagem Tropos 38](#_Toc354590455)

# Capítulo 1 – Introdução:

Este capítulo está dividido em três partes: (i) a primeira apresenta uma visão geral sobre a Engenharia de Requisitos;(ii) a segunda descreve a motivação para o desenvolvimento deste trabalho; (iii) a terceira apresenta os seus objetivos; e por fim, (iv) a quarta detalha a estrutura do documento.

## 1.1.Considerações Iniciais

A análise e especificação de requisitos, fases iniciais do desenvolvimento de um projeto de *software*, são essenciais para que este seja sistemático e disciplinado, diminuindo assim os riscos de falhas no seu desenvolvimento. A ocorrência de falhas durante os processos de ER afetam não somente a fase de produção, mas a própria vida útil dos *softwares*. (BROOKS, 1986).

O objetivo principal da engenharia de requisitos(ER) é descobrir as reais necessidades dos stakeholders, que são as pessoas ou organizações que serão afetadas pelo sistema e tem influência, direta ou indireta, sobre os requisitos do sistema. Para isso, é preciso entender o problema e seu contexto, elicitar os requisitos necessários pra o sistema, analisá-los, documentá-los e validá-los. (KOTONYA; SOMMERVILLE, 1998).

O alvo da Engenharia de Requisitos Orientada a Objetivos é identificar o problema e basear-se nos objetivos pretendidos com o sistema a ser desenvolvido, a partir da captura das necessidades dos stakeholders. Diversas propostas surgiram ao longo do tempo para tentar solucionar o problema da má elaboração dos requisitos dos sistemas, dentre ela está o Framewor i\*, que será utilizado como base nesse trabalho. (YU, E. 1995)

O framework i\* é uma das diversas abordagens para modelagem propostas para especificação de requisitos orientada a objetivos. Com o uso da linguagem i\* (YU, 1997), é possível representar os atores envolvidos no ambiente onde o sistema irá existir, suas intenções e objetivos, assim como os requisitos funcionais e não funcionais do sistema futuro.

## 1.2.Motivação

Por ser o i\* uma linguagem orientada a objetivos de amplo uso no meio acadêmico e com uma aplicação crescente na indústria, diversas linguagens foram propostas baseadas nesta. Estas linguagens são variantes do i\*: i\*Wiki (HORKOFF; Yu; Grau, 2013), GRL (AMYOT et al., 2013), Tropos (SUSI; PERINI; MYLOPOULOS, 2005), i\*-c (SILVA; BORBA; CASTRO, 2010), Aspectual i\* (ALENCAR et al., 2005), i\* Ortogonal(LIMA, 2011) entre outras. Cada variante do i\* busca satisfazer necessidades específicas de um projeto, necessitando de ferramentas também específicas para sua modelagem gráfica. Ou seja, para cada nova linguagem proposta, deverá ser criada uma nova ferramenta de modelagem, ocasionando, assim, o aumento do custo de desenvolvimento (Paes et al., 2011).

Paes et al.(2011) relataram que as linguagens baseadas no i\* herdam ou reusam um conjunto comum de conceitos do i\*original. Sendo assim, idealizaram uma abordagem que permitisse definir a estrutura sintática e semântica de uma nova linguagem para a família i\*, baseada no reuso de um conjunto de conceitos do i\* original. Foi então desenvolvida a abordagem AGILE (Automatic Generation of i\* Languages), que propõe um processo e uma ferramenta CASE (Computer-Aided Software Engineering), a qual permite a configuração do metamodelo núcleo, de forma a criar linguagens baseadas no i\* e a geração automática de seus respectivos editores gráficos. No entanto, a ferramenta havia sido aplicada apenas na construção do editor gráfico para a linguagem i\* Aspectual, não sendo suficiente para avaliar a eficácia da abordagem. Sendo assim, é preciso aplicá-la a outras linguagens baseadas no i\* para que seja possível avaliar a eficácia da ferramenta na construção dos seus respectivos editores gráficos.

## 1.3. Objetivos

### Geral:

Avaliar a eficácia da ferramenta Agile Tool, proposta por Paes et al.(2011), aplicando esta ferramenta na construção de editores gráficos para três variantes da linguagem de modelagem i\*.

### Específicos:

* Analizar e/ou definir os metamodelos referentes às linguagens i\*-c, i\* Orthogonal e i\* com *concerned objects*;
* Utilizar a ferramenta Agile Tool na construção de editores gráficos para estas linguagens;
* Levantar as limitações da ferramenta Agile Tool na construção dos editores gráficos para estas linguagens.

## 1.4. Estrutura do documento

Além deste capítulo introdutório, este documento está estruturado da seguinte forma:

Capítulo 2 – Fundamentos teóricos: Neste capítulo, o objetivo será demonstrar com mais detalhes a Engenharia de Requisitos, a Engenharia de Requisitos Orientada a Objetivos,o*Framework* i\*, apresentação das variantes da linguagem i\* utilizadas no trabalho, a abordagem Agile e a ferramenta Agile Tool.

Capítulo 3 – Avaliação da eficácia do Agile Tool: Neste capítulo serámostrada a aplicação do Agile Tool na criação dos editores gráficos das variantes do i\* consideradas neste trabalho e os resultados obtidosdesta aplicação.

Capítulo 4 – Conclusão:Este capítulo apresenta as conclusões finais sobre o trabalho apresentado e as considerações para trabalhos futuros.

# Capítulo 2 – Fundamentos teóricos

Neste capítulo serão apresentadas as fundamentações teóricas a respeito daEngenharia de Requisitos e do*Framework* i\*. Além disto, considerações sobre o estado atual das ferramentas CASE de suporte ao *Framework* i\* são desenvolvidas, uma vez queelas sãoa base para a construção deste trabalho.

## 2.1. Engenharia de Requisitos

No processo de desenvolvimento de sistemas, a engenharia de requisitos é responsável por melhorar a modelagem dos sistemas e a capacidade de analisá-los, possibilitando uma maior compreensão das suas características antes mesmo de ser implementado. A atividade de engenharia de requisitos deve realizar a interação das partes interessadas pelo sistema para que sejam definidos o domínio da aplicação, os serviços que o sistema irá oferecer e suas restrições operacionais.

O processo de engenharia de requisitos é composto pelas seguintes etapas (KOTONYA; SOMMERVILLE, 1998):

* Elicitação de requisitos: onde é utilizado um conjunto de técnicas para descobrir os requisitos do sistema que será desenvolvido, ou seja, quais propriedades o sistema deve apresentar para resolver um problema real(SWEBOK, 2004). Pode ser feita por técnicas de leitura de documentos, entrevistas, e consultas com os stakeholders. No final desta etapa é gerada uma lista com os requisitos identificados.
* Análise e negociação: cada requisito levantado na etapa anterior é analisado por diferentes stakeholders para que possa ser tomada a decisão de quais requisitos realmente são necessários.
* Documentação: todos os requisitos aceitos são documentados de forma compreensível para todos os stakeholders.
* Validação: esta etapa é realizada em conjunto com o cliente para que possíveis problemas no documento de requisitos sejam identificados e corrigidos antes de se iniciar as atividades do ciclo de desenvolvimento.

Para facilitar a comunicação entre os indivíduos envolvidos no projeto do software são utilizados alguns meios, como: diagramas de casos de uso, diagramas de sequência, diagramas i\*.

## 2.2. Engenharia de Requisitos Orientada a Objetivos

A engenharia de requisitos orientada a objetivos(do ingles, Goal-Oriented Requirements Engineering - GORE) foca em atividades que precedem a formulação dos requisitos do sistema de software. As principais atividades presentes na abordagem GOREsão: elicitação de objetivos, refinamento de objetivos e vários tipos de analises de objetivos, e atribuição de responsabilidades a metas para agentes. GORE visa o sistema a ser feito e seu ambiente como uma coleção de componentes ativos chamados agentes. Componentes ativos podem restringir seus comportamentos para garantir as restrições que lhes são atribuídas.Estes componentes são humanos desempenhando certos papéis, dispositivos, e softwares. (LAPOUCHNIAN, 2005)

GORE tem uma perspectiva de engenharia de sistemas mais ampla, comparada aos métodos tradicionais de engenharia de requisitos. Os objetivos são afirmações prescritivas que devem controlar o sistema feito do software TO-BE e seu ambiente. As propriedades de domínio e expectativas sobre o ambiente são capturadas explicitamente durante o processo de elaboração de requisitos, adicionadas a especificação de requisitos de software usual. Além disso, os objetivos fornecem uma justificativa para os requisitos que os operacionalizam.E engenharia de requisitos orientada a objetivos se baseia nos objetivos dos stakeholders, com o propósito de desenvolver um software que realmente satisfaça seus desejos.

Com o crescente uso da engenharia de requisitos orientada a objetivos foram desenvolvidas várias propostas de abordagens GORE para satisfazer necessidades de diversos grupos de pesquisa. Nesse trabalho será abordado uma dessa propostas, o framework i\*.

### 2.3.Framework i\*

O framework i\* é uma abordagemGORE usada para suportar análise e modelagem organizacional. Está centrada em definir relacionamentos entre atores estratégicos, que representam os *stakeholders* envolvidos com o sistema. Os atores dependem uns dos outros para atingir seus objetivos, para realizar suas tarefas e para obter ou fornecer recursos necessários.(YU, 1995).

Este framework consiste em dois modelos: modelo de Dependência Estratégica (SD), para descrever uma configuração particular de relacionamento de dependência entre atores organizacionais, e modelo de Razão Estratégica (SR), para descrever os motivos que os atores possuem em adotar certa configuração.

Em ambos os modelos, um ator é uma entidade ativa que realiza ações para atingir seus objetivos.Este termo é usado para se referir genericamente a qualquer unidade para a qual dependências intencionais podem ser atribuídas (YU, 1995). Os atores podem ser especializados em três tipos:

* Papel: É uma caracterização abstrata do comportamento de um ator dentro de um contexto.
* Agente: É um ator com manifestações físicas concretas, podendo tanto se referir a humanos como agentes artificiais (hardware/software).
* Posição: É um conjunto de papéis tipicamente realizado por um agente. Um agente ocupa uma posição, e, uma posição desempenha um papel.

Existem seis tipos de relacionamentos entre atores (Figura 2.1):

* ISA: é uma generalização, um ator sendo especializado por outro ator.
* IS-part-of: Papéis, posições e agente podem conter subpartes. Podem existir dependências entre o todo e suas partes.
* Plays: Usada entre um agente e um papel, onde o agente executa o papel.
* Covers: Usada para descrever um relacionamento entre uma posição e o papel que ele cobre.
* Occupies: Usado para mostra que um agente ocupa uma posição.
* INS: Representa uma instância específica de uma entidade mais geral. Um agente é uma instanciação de outro agente.

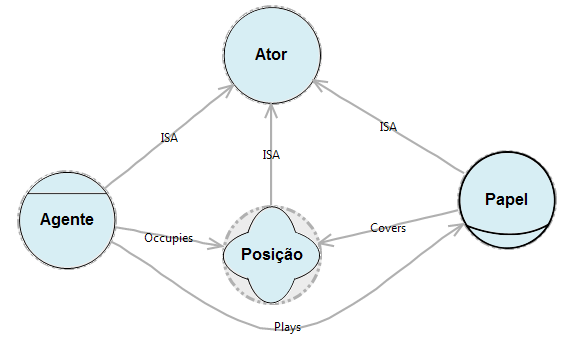


Figura 2.1 - Especializações de atores e relacionamento entre atores

### 2.3.1. SD *Model*

O modelo de dependência estratégica provê uma descrição informal de um processo termo de uma rede de relacionamento de dependência entre atores. Um modelo de processo intencional busca capturar as motivações e intenções por trás das atividades e fluxos em um processo.

Esse modelo utiliza um conjunto de nós e ligações para representar os atores envolvidos no processo e seus relacionamentos.Um ator que depende de outro para atingir seus objetivos é chamado de *depender*. O ator que satisfaz a dependência é chamado de *dependee*. O acordo entre estes dois atores é chamado de *dependum*. (YU, 1995) (Figura 2.2)

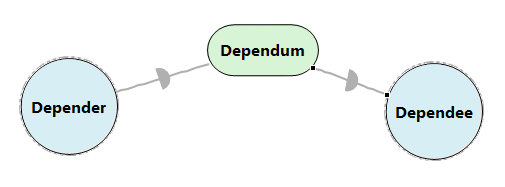


Figura 2.2 - Dependência entre atores

Existem quatro tipos de acordo nos relacionamentos de dependência do i\*: Objetivo (goal), tarefas (task), recursos(resources) e softgoal. Ou seja, num relacionamento entre atores, um dependee pode atender a um objetivo, executar uma tarefa, prover um recurso, ou realizar uma tarefa para alcançar um softgoal. (YU, 1995)

Na dependência de objetivo, o depender depende do dependee para alcançar um certo estado de mundo, e o dependee tem a liberdade de escolher como fazer isso. Com a dependência de objetivo, o depender ganha a habilidade de assumir que um estado de mundo ou uma condição será realizado, mas se torna vulnerável já que o dependee pode falhar ao realizar. (Figura 2.3)

Na dependência de tarefa, o depender depende do dependee para realizar uma atividade. Ela especifica como uma tarefa deve ser feita, mas não diz o porquê. O depender se torna vulnerável, pois o dependee pode não conseguir realizar a tarefa.

Na dependência de recurso, o depender depende do dependee para a disponibilidade de uma entidade, que pode ser física ou uma informação. Estabelecida essa dependência, o depender é capaz de usar essa entidade como um recurso.

Na dependência de softgoal, o depender depende do dependee para realizar um requisito não funcional.

|  |  |
| --- | --- |
|  |  |
|  |  |

Figura 2.3 - Tipos de relacionamentos entre atores no i\*

|  |
| --- |
|  |
|  |
|  |
|  |

O modelo também distingue entre graus de dependência. São eles: Aberto, Compromissado e Crítico.

No relacionamento aberto um depender deseja obter um objetivo, uma tarefa ou um recurso. Uma falha na obtenção de uma delas afetaria a meta do depender de certa forma, mas não acarreará grandes conseqüências. Esta dependência é representada por “O”. (GRAU et al., 2008)

No relacionamento comprometido (commited), se ocorrer uma falha na relação, as intenções do depender serão afetadas, mas não sofrerão conseqüências muito graves. No lado do dependee, essa relação significa que ele tentará o seu melhor para fornecer o que o depender deseja. Graficamente, essa dependência não possui representação.

No relacionamento crítico (critical), se ocorrer uma falha na dependência, as intenções do depender serão gravemente afetadas. Essa relação é representada por “X”. (GRAU et al., 2008) (Figura 2.4)

|  |
| --- |
| Figura 2.4 - Graus de dependência em i\* |

### 2.3.2. SR *Model*

O modelo de razão estratégica oferece uma descrição intencional de processos em termos de elementos de processos e a razões por trás deles. O modelo SR descreve os elementos intencionais que são internos ao ator. (YU, 1995) Utilizando o conhecimento representado e organizado por estes conceitos de modelagem, alternativas de processo podem ser geradas e exploradas sistematicamente de modo a ajudar os atores a melhor atender os seus interesses. Graficamente, as intencionalidades de cada ator são representadas dentro de sua fronteira. (Figura 2.5)

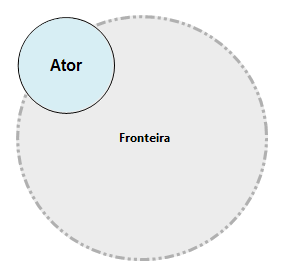


Figura 2.5 - Ator e sua frontera

O modelo SR possui os mesmos tipos de nós do modelo SD: objetivo (goal), tarefa (task), recurso (resource) e softgoal. Existem duas classes de ligações: ligação meio-fim (means-end) e ligação de decomposição de tarefa (task decomposition).

As ligações de meio-fim indicam uma relação entre um fim e um meio de alcançar este fim. Um fim pode ser um objetivo a ser alcançado, uma tarefa a ser realizada, um recurso a ser produzido, ou um softgoal a ser satisfeito. O meio usualmente é descrito como uma tarefa, visto que uma tarefa expressa a idéia de como fazer alguma coisa. Quando um meio for uma tarefa, o fim pode ser um objetivo, tarefa, recurso ou softgoal.Se um meio for um objetivo, obrigatoriamente o fim deverá ser um objetivo.Para que possa existir uma ligação meio-fim entre softgoals, um destes deve ser satisfeito por uma tarefa. (Figura 2.6)

|  |  |
| --- | --- |
|  |  |

Figura 2.6- Tipos de ligação meio-fim

As ligações de decomposição de tarefas podem ligar nós tarefas a objetivos, tarefas, recursos e softgoals. Este tipo de ligação significa que, se uma tarefa for decomposta, ela só pode ser completada se todos os seus componentes ligados estiverem satisfeitos. (Figura 2.7)

|  |
| --- |
| Figura 2.7- Tipos de decomposição de tarefas |

As ligações de contribuição mostram como um elemento, que pode ser um softgoal ou uma tarefa, contribui para que um softgoal seja satisfeito. Existem nove tipos de contribuição. São eles:

* Make: contribuição positiva que é suficiente para que um softgoal seja satisfeito;
* Some +: contribuição positiva, mas possui nível de satisfação desconhecido;
* Help: contribuição parcialmente positiva, já que não consegue satisfazer um softgoal sozinha;
* Unknown: contribuição com influência desconhecida;
* Some -: contribuição negativa, mas possui nível de satisfação desconhecido;
* Hurt: contribuição parcialmente positiva, já que não consegue comprometer um softgoal sozinha;
* Or: contribuição que consegue satisfazer se pelo menos um dos elementos de origem for satisfeito.
* And: contribuição que só consegue satisfazer se todos os elementos de origem forem satisfeitos.
* Break: contribuição negativa que consegue comprometer um softgoal;

Para representar os metamodelos das linguagens i\* utilizamos o diagrama de classe da notação UML. O metamodelo núcleo das linguagens i\* é composto pelos seguintes elementos, e é mostrado na figura 2.8:

* ***Model****,* representa o local onde os elementos serão inseridos, sendo assim, é um repositório de atores, elementos intencionais e relacionamentos;
* ***Compartment****,* caracteriza os atores nos modelo SD e SR. Possui um tipo que pode ser obtido do enumerador *CompartmentType*;
* ***Element****,* pode estar contido em um *Model* ou em um *Compartment*. Os tipos possíveis de *Element* estão no enumerador *ElementType*.
* ***IntentionalElement****,* é uma superclasse que representa todos os tipos de elementos intencionais.
* ***NodeObject****,* é uma superclasse que representa todos os tipos de elementos intencionais e compartimentos. As metaclasses *Compartment* e *IntentionalElement* são especializações dessa classe.
* ***Relationship****,* esta classe foi incluída no metamodelo núcleo da linguagem i\*, assim ela pode ser especializada para representar algum relacionamento que possa ser criado em extensões de linguagem i\*.
* ***ActorLink****,*é uma especialização da metaclasse *Relationship*. Está composta na metaclasse *Model,* uma vez que só pode ser feita entre elementos contidos dentro de um modelo. Possui um atributo do tipo *ActorLinkType* para representar o tipo de relacionamentos entre atores.
* ***DependencyLink****, conecta atores e/ou elementos intencionais.* Deve ser utilizado dentro de um modelo, e possui atributo type do tipo *DependencyLinkType*, que é um *enumeration*. Todo relacionamento contém um source (início a ligação) e um target (destino da ligação).
* ***ContributionLink,*** relacionamento entre elementos intencionais da metaclasse *Element,* e pode ser criado somente dentro de um *Compartment*. Seus tipos estão enumerados no ContributionType.
* ***TaskDecompositionLink e MeansEndLink,*** relacionamento entre elementos intencionais da metaclasse *Element,* e pode ser criado somente dentro de um *Compartment*.

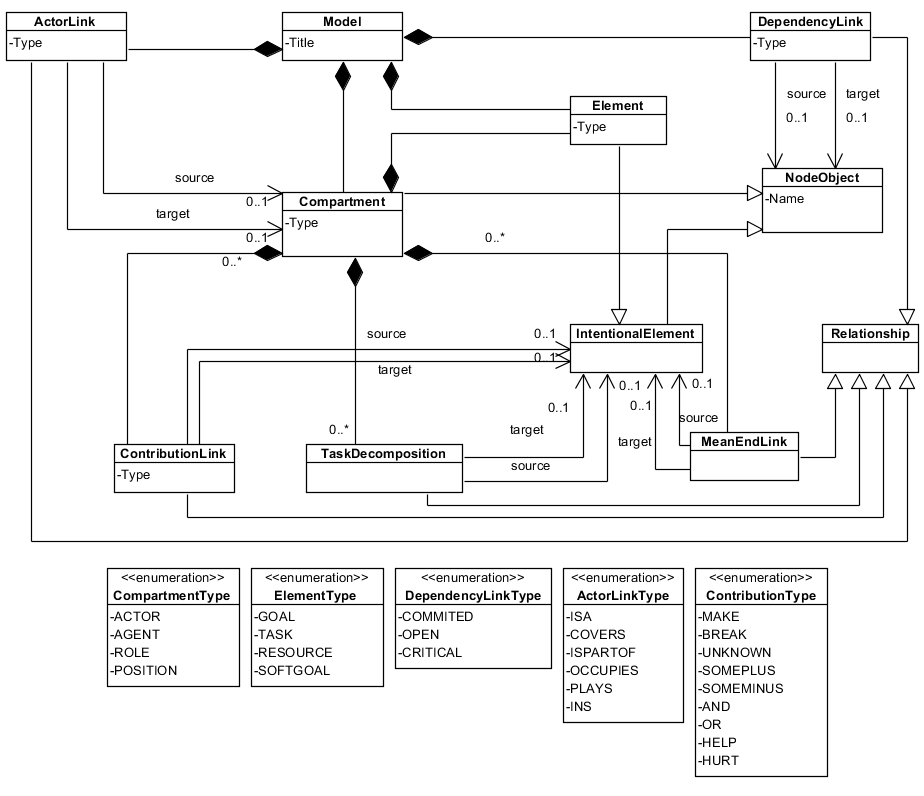


Figura 2.8- Metamodelo núcleo para a linuagem i\* original

### 2.4. Variantes do i\*

Neste trabalho serão estudadas algumas linguagens que foram construídas baseadas no i\* original. Estes dialetos podem conter novos elementos ou apenas modificar algumas restrições de relacionamento do i\* original. A seguir será apresentada uma descrição rápida sobre essas variants, que serão abordadas mais a fundo em seções mais a frente.

### 2.4.1. I\*-c

O I\*-c (i\* com cardinalidade) é uma extensão do i\* original que pode ser usado para fazer a modelagem de LPS (Linha de Produtos de Software). Sendo assim, o modelo deve ser capaz de capturar informações relativas à presença opcional e obrigatória de *features*, queindica uma funcionalidade ou uma característica do sistema, em um produto de LPS. (SILVA; BORBA; CASTRO, 2010)

### 2.4.2. I\*-Orthogonal

O i\*-Orthogonal é uma extensão da linguagem i\*-C (SILVA; BORBA; CASTRO, 2010), definida na abordagem G2SPL.

### 2.4.3. I\* com concerned object

O modelo concerned object é uma extensão do diagrama de atores e diagrama de objetivos do framework Tropos (REBOLLAR, 2008).

### 2.5. Abordagem Agile

O framework i\* é uma linguagem orientada a objetivos em amplo uso no meio acadêmico e com uma aplicação crescente na indústria. Um dos principais desafios em trabalhar com esta linguagem é a diversidade de variantes que foram desenvolvidas a partir dela. Para cada variante, novos elementos de modelagem foram adicionados ao i\* original.Cada uma dessas linguagens busca satisfazer necessidades específicas de um projeto, necessitando assim de ferramentas específicas para sua modelagem gráfica.Ou seja, para cada nova linguagem proposta, deverá ser criada uma nova ferramenta de modelagem, ocasionando, assim, o aumento do custo de desenvolvimento (Paes et al., 2011). Segundo Lucena(2008), essas variações de linguagens baseadas no i\* podem ocasionar alguns problemas:

* Divisão de esforço, pois cada grupo de pesquisa estaria preocupado em desenvolver uma ferramenta para dar suporte ao seu problema específico.
* Novos usuários ficariam desmotivados em adotar o i\* devido a grande variação de linguagens.

Paes et al.(2011) relataram que as linguagens baseadas no i\* herdam ou reusam um conjunto comum de conceitos do i\*original. Foi identificado um núcleo comum entre estas linguagens, onde foram identificados os elementos de modelagem comuns e separados os elementos de modelagem variáveis.

Sendo assim, idealizaram uma abordagem que permitisse definir a estrutura sintática e semântica de uma novalinguagem para a família i\*, baseada no reuso de um conjunto de conceitos do i\* original.

Paes et al.(2011)Surgiu então a abordagem AGILE (Automatic Generation of i\* Languages). Esta abordagem proporciona a geração automática de editores gráficos para linguagens baseadas no i\* utilizando uma ferramenta CASE (Computer-Aided Software Engineering). Esta ferramenta é integrada ao framework GMF(Graphical Modelling Framework), um framework para a geração de editores gráficos baseados em modelos de domínio, utilizando EMF e GEF.O EMF (Eclipse MOdeling Framework) auxilia a especificação de metamodelos e oferece funcionalidades para a geração automática de código, e o GEF(Graphical Eclipse Framework) é utilizado na criação de editores gráficos. O framework GMF é utilizado na construção de editores gráficos baseados na plataforma Eclipse, e é dividido em dois componentes principais: toolinge runtime. O componente tooling disponibiliza uma abordagem orientada a modelos para a geração do editor gráfico, e o componente runtime disponibiliza os elementos e ferramentas básicas para o editor. O GMF, portanto utiliza uma metodologia de desenvolvimento que se concentra na criação de modelos ao invés da geração de código baixo nível, chamada MDD (Model Driven Development).

O diagrama abaixo(Figura 2.9) mostras as dependências entre o editor gráfico gerado, GMF runtime, EMF, GEF e a plataforma Eclipse (<http://www.eclipse.org/articles/Article-Introducing-GMF/article.html>).

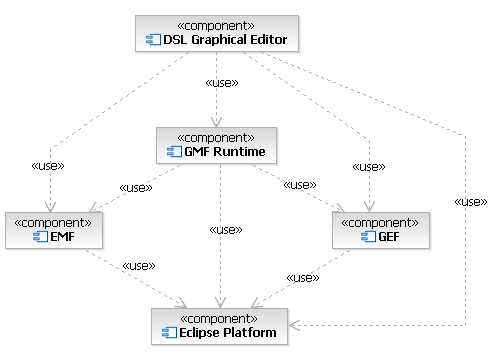


Figura 2.9 - Framework GMF e dependências

Como podemos ver, um editor gráfico GMF depende do componente runtime GFM, mas também faz uso direto do EMF, GEF e da plataforma Eclipse.

Para representar diferentes aspectos de um editor, o framework GMF é dividido em quatro etapas:

**Domain Model**: o metamodelo que queremos usar para criar o editor gráfico descrito usando a linguagem de descrição chamada Ecore(EMF 2010).

**Domain Gen Model**: esse arquivo é usado para gerar o código do modelo de domínio com EMF.

**Graphical Def Model**: esse arquivo PE usado para definir os elementos gráficos para o modelo de domínio.

**Tooling Def Model**: usado para definir a paleta de ferramentas que pode ser usada no editor gráfico.

**Mapping Model**: esse arquivo liga o modelo de domínio, o modelo gráfico e o modelo de ferramentas.

**Diagram Editor Gen Model**: esse arquivo final é usado para gerar o editor gáfico GMF além do código EMF gerado pelo arquivo .genmodel.

Estes passos estão mostrados na figura abaixo (Figura 2.10):

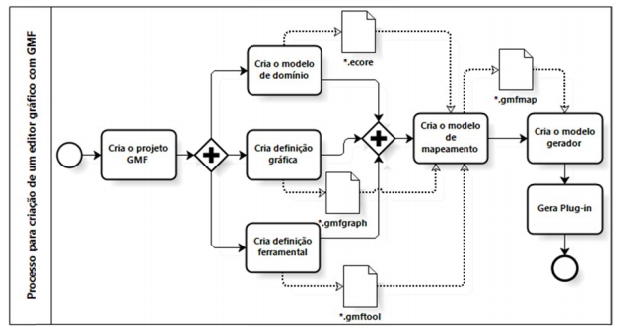


Figura 2.10 - Processo de criação de um editor gráfico com o framework GMF utilizando a notação BPMN

Teoricamente a geração de artefatos é feita automaticamente, utilizando os conceitos de MDD. Porém, na prática surgem alguns problemas que comprometem a idéia de geração automática. Um deles é que a geração do modelo de domínio é realizada manualmente, bem como a forma gráfica dos elementos de modelagem, na parte da definição gráfica. Além de outros problemas listados por Paes et al.(2011).

Com o objetivo de automatizar as atividades do processo de criação de editores gráficos para linguagens baseadas no i\* utilizando o GMF, foi criada a abordagem AGILE. Essa abordagem visa além de automatizar, aumentar a produtividade e diminuir o custo de desenvolvimento de ferramentas desse tipo. Para isto, o foram excluídas do processo as atividades de criação do modelo de domínio, criação da definição gráfica e criação do ferramental, e foram adicionadas as seguintes atividades semi-automatizadas: configuração da base do i\*; criação e configuração de novos elementos de modelagem; e geração automática dos modelos GMF. Foi criada a ferramenta AGILE Tool para automatizar o processo. (Figura 2.11)

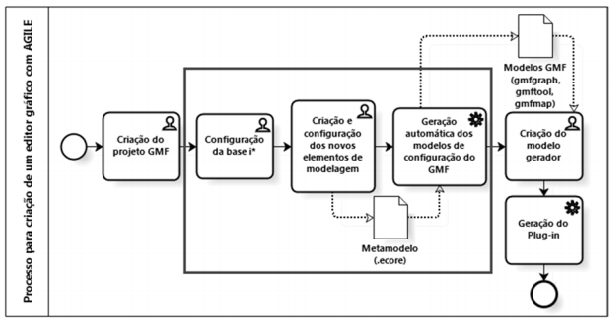


Figura 2.11 - Processo de criação de um editor gráfico utilizando a abordagem AGILE

### 2.6. Agile Tool

A ferramenta AGILE Tool foi desenvolvida para dar suporte a abordagem AGILE. Através de uma interface amigável, a ferramenta dá suporte às principais etapas de criação de linguagens baseadas no i\*.A AGILE Tool é um plugin para o Eclipse, trabalhado conjuntamente com o GMF.

A ferramenta possui três etapas para a definição do metamodelo da linguagem i\* através da ferramenta:

1. Configuração da base i\* para a nova linguagem.
2. Criação, configuração e inserção de novos elementos de modelagem no metamodelo previamente criado.
3. Geração automática das restrições para cada relacionamento existente na linguagem através de código OCL (Object Constraint Language).

Na primeira etapa cada um dos elementos de modelagem existentes no i\* original é mapeado e implementado como entidades, como mostra o diagrama a seguir. (Figura 2.12)

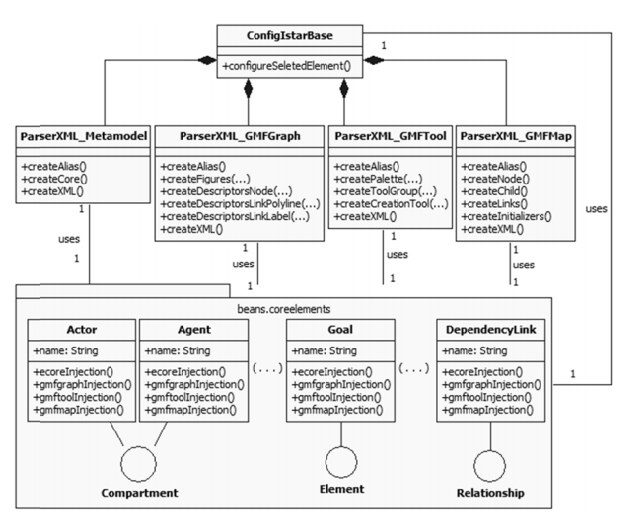


Figura 2.12 - Diagrama classes mostrando os elementos de modelagem

A configuração da linguagem é realizada na tela inicial do Agile Tool. Nela devem ser selecionados todos os elementos da base do i\* que será utilizada para a criação na nova linguagem. (Figura 2.13)

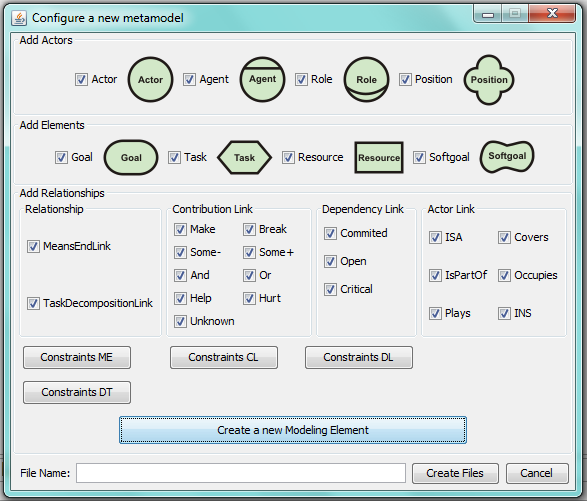


Figura 2.13 - Tela principal do Agile Tool para a configuração de uma base i\*

A segunda etapa, responsável pela criação, configuração e inserção de novos elementos, só pode ser feita após a configuração da base do i\*. Os novos elementos serão inseridos no metamodelo criado na etapa anterior.Existem três tipos de classes de novos elemento, são elas: NewModule, NewElement e NewRelationship, que compõem o pacote newElements. A figura a seguir (Figura 2.14) mostra o diagrama com os possíveis novos elementos.

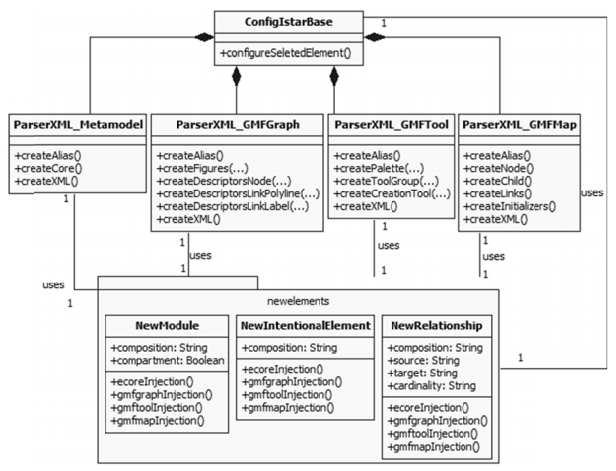


Figura 2.14 - Diagrama de classes resumido para a criação de novos elementos de modelagem

As configuraçãos dos novos elementos é feita na tela de criação e configuração de novos elementos do Agile Tool. Nela serão definidas características importantes dos novos elementos, como: onde o elemento será composto, qual será sua classificação, seu atributos, entre outras. A tela de criação e configuração é mostrada a seguir. (Figura 2.15)

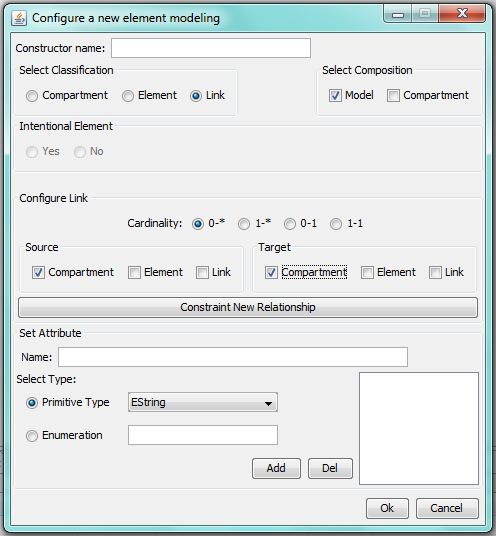


Figura 2.15 - Tela de configuração dos novos elementos de modelagem

A terceira e última etapa é a de geração automática das restrições para cada relacionamento existente. Sempre que uma ligação do i\* original for selecionada ou quando uma nova ligação for criada, serão feitas as restrições de relacionamentos.

Na tela do Agile Tool devem ser selecionadas a fonte (Source) e os possíveis alvos(target) do relacionamento. Automaticamente é gerado o código OCL de acordo com as configurações do usuário. (Figura 2.16)

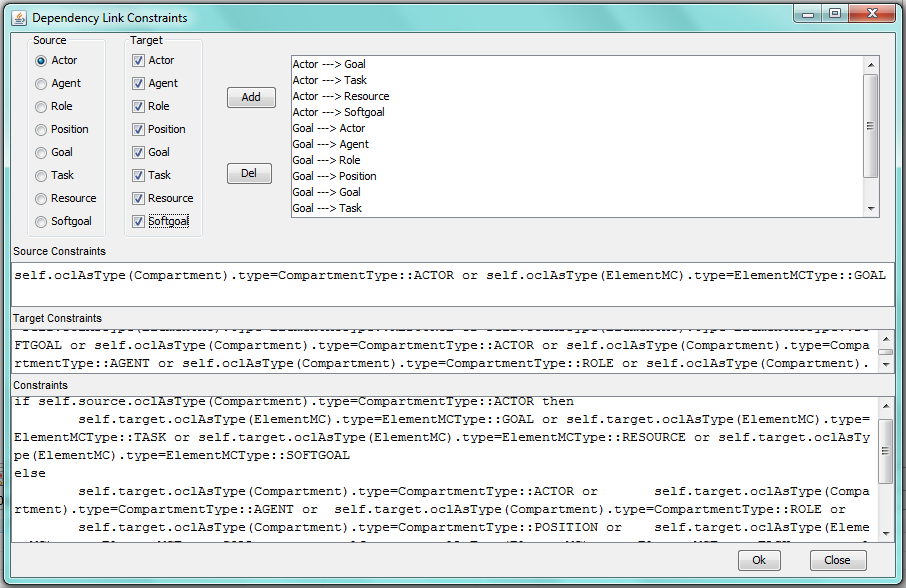


Figura 2.16 - Tela do Agile Tool para configuração de restrições

### 2.7. Definição dos metamodelosdas linguagens variantes do i\*

Neste capítulo serão estudadas as variantes do i\* apresentadas na seção anterior. Serão mostrados os elementos que as compõem, suas restrições e modelaremos o metamodelo para cada uma das variantes.

### 2.7.1. I\*-c

Como foi visto na seção 2.4.1, o i\* com cardinalidade tem o objetivo de capturar as variabilidades de linhas de produtos de software. Para isso, foi adicionado o atributo de cardinalidade a alguns elementos de modelagem para permitir a captura de mais informação sobre uma variabilidade de uma LPS. (SILVA; BORBA; CASTRO, 2010)

Os elementos que podem ter cardinalidade são os elementos do tipo recurso e tarefa, visto que representam funcionalidades e características de um sistema. (Figura 2.17)

|  |
| --- |
|  |
| Figura 2.17 - Elementos do tipo tarefa ou recurso com cardinalidade [m..n], [0..1] e [1..1] |

A ligação do tipo meio-fim também pode receber o atributo de cardinalidade, podendo ser dos tipos *ou-inclusivo* (isto é, com cardinalidade <1-k>), *ou-exclusivo* (cardinalidade <1-1>) e *ou com cardinalidade* (cardinalidade <i-j>).(Figura 2.18)

|  |  |  |
| --- | --- | --- |
| **Ligação meio-fim com cardinalidade <i..j>** | **Ligação meio-fim com cardinalidade <1..k>** | **Ligação meio-fim com cardinalidade <1..1>** |
|  |  |  |

Figura 2.18 - Cardinalidade nas ligações do tipo means-end

A tabela abaixo (Tabela 2.1) mostra todas os tipos de ligações existentes no modelo i\*-c.

Tabela 2.1 - Restrições existentes na linguagem i\*-C

|  |  |
| --- | --- |
| Restrições do I\*-c | |
| Ligação de dependência | Atores → Elementos intencionais  Elementos intencionais → Atores  Elementos → Elementos intencionais |
| Ligação Ator (ISA) | Ator → Ator  Agente → Agente  Posição → Posição  Papel → Papel |
| Ligação Ator (Is Part Of) | Ator → Ator  Agente → Agente  Posição → Posição  Papel → Papel |
| Ligação Ator (Covers) | Posição → Papel |
| Ligação Ator (Occupies) | Agente → Posição |
| Ligação Ator (Plays) | Agente → Papel |
| Ligação Ator (INS) | Agente → Agente |
| Meio-fim | Tarefa → Objetivo  Tarefa com cardinalidade→ Objetivo |
| Meio-fim com Cardinalidade | Objetivo → Objetivo  Tarefa → Objetivo |
| Decomposição de Tarefas | Tarefa com cardinalidade→ Tarefa com cardinalidade  Tarefa com cardinalidade→ Todos os elementos intencionais |
| Ligação de contribuição | Softgoal → Softgoal  Tarefa → Softgoal  Objetivo → Softgoal |
| Tipos de contribuição | Make, Break, SomePlus,  SomeMinus, Help, Unknown, Hurt, And, Or |

Para representar o metamodelo do i\* com cardinalidade, foi adicionada a seguinte metaclasse no metamodelo do i\* original:

* ***Cardinality***, define a cardinalidade dos elementos intencionais e do relacionamento *MeansEndLink*, onde o valor mínimo deve ser menor ou igual ao valor máximo, e quando os valores forem iguais eles devem ser diferentes de zero.

A partir do metamodelo núcleodas linguagens *i\**(PAES, LIMA, *et al.*, 2011), foi criado o metamodelo para a linguagem i\*-C, mostrado na figura a seguir.A metaclasse Cardinality éum atributo de um *IntentionalElement* ou de um *MeandEndLink*.

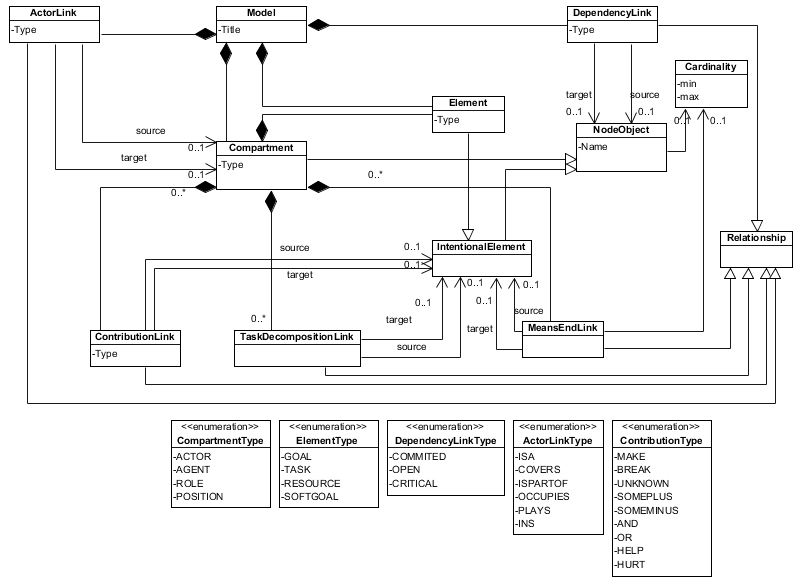
******

Figura 2.19 - Metamodelo para a linguagem i\*-C

### 2.7.2. I\*-orthogonal

O i\*-Orthogonal é uma extensão para a linguagem i\*-c. Essa extensão incorpora alguns construtores de variabilidade que estavam faltando, tornando possível capturar e analisar variabilidade organizacional (LIMA, 2011).

Foram aplicadas as seguintes modificações ao i\*-c:

* Exclusão do relacionamento means-end entre objetivos;
* Inclusão de cardinalidade a todos os elementos que representam pontos de variação para indicar se são obrigatórios ou opcionais.
* Reestruturação da análise sobre as ligações entre pontos de variação e variantes, excluindo as propriedades booleanas isFeature e isGrouped; e criação das metaclasses Cardinality e FeatureGroup (e suas extensões – DependencyFeatureGroup e MeansEndFeatureGroup);
* A linguagem permite que um dependum do tipo objetivo seja uma ponto de variação e,assim, captura variabilidade externa.
* Inclusão de atores do modelo SD como variantes, uma vez que o ator do sistema pode ser relacionado a eles através de um ponto de variação. Também foram adicionadas cardinalidade às dependências.
* Inclusão de restrições de variabilidades (dos tipos “requires” e “excludes”) entre elementos intencionais e/ou atores, ou seja, entre pontos de variação e variantes.

Com o objetivo de criar os construtores de variabilidade, alguns relacionamentos e outras metaclasses(Figura 2.20) foram adicionadas no metamodelo núcleo do i\*.

* ConstraintLink é usado para representar restrições de variabilidade e possui dois tipos: requires e excludes. Esse relacionamento deve ter um source e um target do tipo NodeObject.
* FeatureGroup foi criada para agrupar as dependências de variabilidades. Esta metaclasse possui as extensões DependencyFeatureGroup e MeansEndFeatureGroup.
* Cardinality foi incluída para representar a propriedade de NodeObject e FeatureGroup. Possui valores mínimos e máximos que fazem a distinção de dependências de variabilidades opcionais, obrigatórias ou alternativas.
* Context é uma metaclasse que permite o i\* orthogonal ter notações de contexto. Essas anotações são usadas para determinar se certo relacionamento será ativado em um produto. O contexto só pode estar relacionado a um Relationship e não a um NodeObject, pois analisando quais relacionamentos são ativados é também possível determinar quais elementos também estarão presentes no modelo da aplicação.

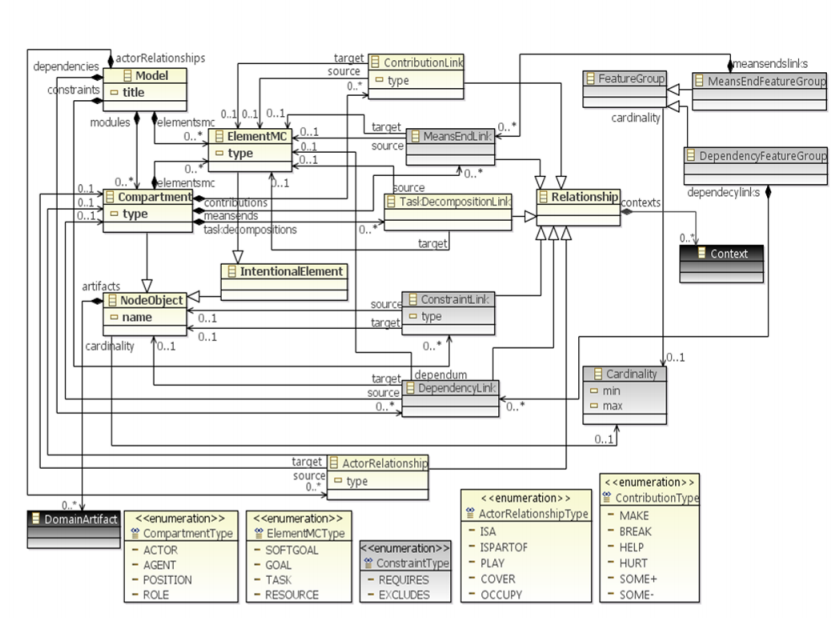


Figura 2.20 - Metamodelo da linguagem de variabilidade para modelo de objetivos i\*-Ortogonal (LIMA, 2011)

A tabela 2.2 mostra todos os tipos de ligações existentes no modelo i\* orthogonal.

Tabela 2.2 - Restrições existentes na linguagem i\*-Orthogonal

|  |  |
| --- | --- |
| Restrições do I\* orthogonal | |
| Ligação de dependência | Atores → Elementos intencionais  Elementos intencionais → Atores  Elementos → Elementos intencionais |
| Ligação Ator (ISA) | Ator → Ator  Agente → Agente  Posição → Posição  Papel → Papel |
| Ligação Ator (Is Part Of) | Ator → Ator  Agente → Agente  Posição → Posição  Papel → Papel |
| Ligação Ator (Covers) | Posição → Papel |
| Ligação Ator (Occupies) | Agente → Posição |
| Ligação Ator (Plays) | Agente → Papel |
| Ligação Ator (INS) | Agente → Agente |
| Meio-fim | Tarefa → Objetivo  Tarefa com cardinalidade→ Objetivo  Recurso → Objetivo  Recurso com cardinalidade→ Objetivo |
| Meio-fim com Cardinalidade | Objetivo → Objetivo  Tarefa → Objetivo |
| Decomposição de Tarefas | Tarefa → Objetivo  Tarefa → Tarefa  Tarefa → Recurso |
| Ligação de contribuição | Softgoal → Softgoal  Tarefa → Softgoal  Objetivo → Softgoal  Recurso → Softgoal |
| Tipos de contribuição | Make, Break, SomePlus,  SomeMinus, Help, Unknown, Hurt, And, Or |
| Cardinalidade de elementos intencionais | [0..1] (Opcional)  [1..1] (Obrigatório) |

### 2.7.3. I\* com concerned objects

O modelo concerned object é uma extensão do diagrama de atores e diagrama de objetivos do framework Tropos. Nesta subseção será abordada primeiramente a modelagem Tropos e em seguida o modelo de concerned object (REBOLLAR, 2008).

**Modelagem Tropos**

Tropos é uma metodologia de engenharia de software orientado a agentes que provê uma linguagem de modelagem baseada no paradigma multi-agentes(SUSI et al., 2005). As quatro primeiras fases da metodologia Tropos são baseadas no framework i\*, mas ela apresenta algumas modificações. O nome do elemento Tarefa (Task) mudou para Plano (Plan), bem como o elemento Objetivo (Goal) para Hardgoal. A ligação meio-fim sofreu algumas modificações que serão mostradas na tabela x. No modelo Tropos não existe mais o conceito de decomposição de tarefas. Agora, a decomposição pode ser do tipo And ou Or. Em ambas as decomposições podem existir o relacionamento de hardgoal para hardgoal, softgoal para softgoal e plano para plano. As ligações de contribuição agora também podem existir de hardgoal para hardgoal, plano para hardgoal, softgoal para hardgoal. Os tipos de contribuição foram reduzidos para: help, break, hurt e make.

O framework tropos utiliza a seguinte representação gráfica para representar o meio organizacional:

* ***Diagrama de ator***, que é uma representação gráfica onde os atores e seus objetivos, e as dependências entre os atores são mostradas. Esse modelo enfatiza os aspectos estáticos de uma organização.
* ***Diagrama de objetivos,*** que é uma representação gráfica onde os objetivos, planos e dependencias entre cada ator são analizados com mais detalhes.

Tabela 2.3 - Restrições existentes na linguagem Tropos

|  |  |
| --- | --- |
| Restrições do Tropos | |
| Ligação de dependência | Atores → Elementos intencionais  Elementos intencionais → Atores  Elementos → Elementos intencionais |
| Ligação Ator (ISA) | Ator → Ator  Agente → Agente  Posição → Posição  Papel → Papel |
| Ligação Ator (Is Part Of) | Ator → Ator  Agente → Agente  Posição → Posição  Papel → Papel |
| Ligação Ator (Covers) | Posição → Papel |
| Ligação Ator (Occupies) | Agente → Posição |
| Ligação Ator (Plays) | Agente → Papel |
| Ligação Ator (INS) | Agente → Agente |
| Meio-fim | Plano → Hardgoal  Plano → Softgoal  Recurso → Hardgoal  Recurso → Softgoal  Hardgoal → Hardgoal  Hardgoal → Softgoal  Softgoal → Hardgoal  Softgoal → Softgoal |
| Decomposição-AND | Hardgoal → Hardgoal  Softgoal → Softgoal  Plano → Plano |
| Decomposição-OR | Hardgoal → Hardgoal  Softgoal → Softgoal  Plano → Plano |
| Ligação de contribuição | Plano → Hardgoal  Plano → Softgoal  Hardgoal → Hardgoal  Hardgoal → Softgoal  Softgoal → Hardgoal  Softgoal → Softgoal |
| Tipos de contribuição | Make, Break, Help, Hurt |

A partir do metamodelo núcleodas linguagens *i\**(PAES, LIMA, *et al.*, 2011), foi criado o metamodelo para a linguagem Tropos, mostrado na figura a seguir.

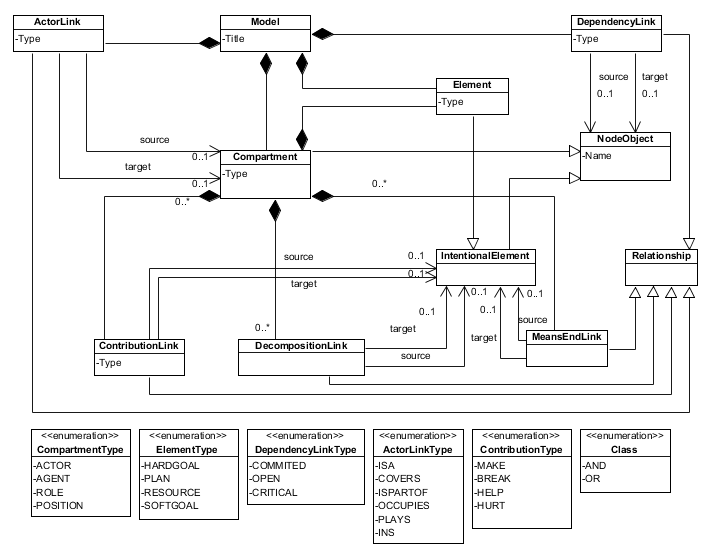


Figura 2.21 - Metamodelo para a linguagem Tropos

**O Modelo de concerned object**

Como foi falado no início desta seção, o modelo de concerned object (ou objeto de interesse) é uma extensão do diagrama de ator e o diagrama de objetivo do framework Tropos. O foco dessa extensão é em descrever a informação relevante do software a ser criado através da identificação de objetos de interesse nos elemenetos de cada modelo organizacional (REBOLLAR, 2008).

Os objetos de interesse podem estar ligados aos elementos do tipo Resource ou Plan. Eles são representados por um círculo, e seus atributos são representados por pequenos círculos associados a eles. A extensão do diagrama de ator envolve a representação dos objetos de interesse associados as dependências de planos e recursos entre os atores.

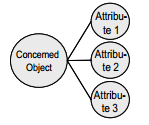
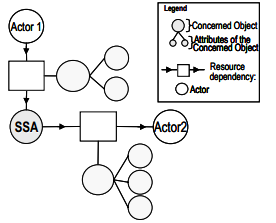


Figura 2.22 Modelo de concerned object(REBOLLAR, 2008).

Para podermos representar o metamodelo da modelagem com concerned objects (Figura 2.23) criamos as seguintes metaclasses:

* ***ConcernecdObject***, que representa o objeto de interesse e pode estar ligado aos elementos do tipo Resource ou Plan.
* ***ConcernedLink***, representando o relacionamento entre o concerned object e os elementos intencionais.
* ***AttribConcerned****,* para representaros atributos do converned object.
* ***LinkAttribConcerned***, que é a ligação entre o concerned object e seus atributos.

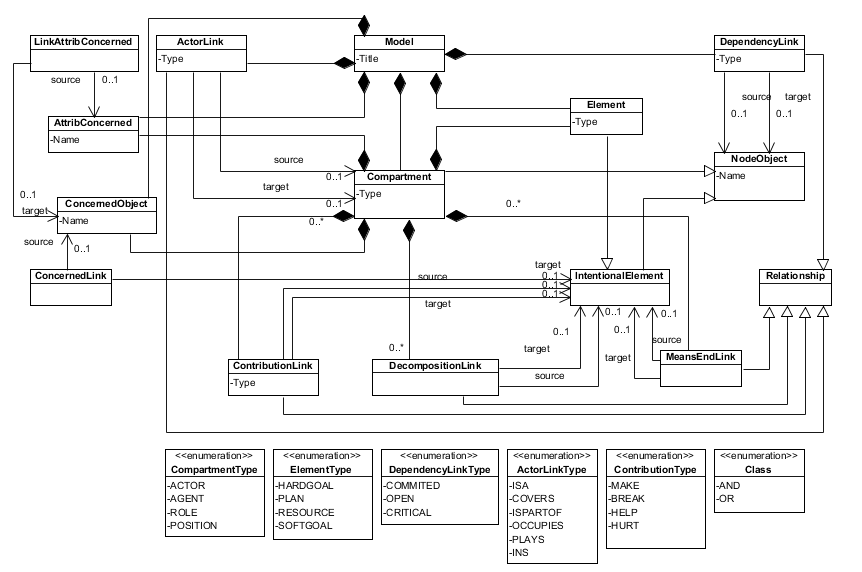


Figura 2.23 - Metamodelo para a linguagem i\* com concerned objects

# Capítulo 3 – Avaliação da eficácia do Agile Tool

Neste capítulo detalharemos a aplicação da ferramenta Agile Tool para a geração de editores gráficos para as variantes do i\* original apresentadas no capítulo anterior. Dessa forma será possível avaliar a eficácia da ferramenta.

Inicialmente foi feita configuração do i\* original que servirá como base para as novas variantes da linguagem. Após a criação da base i\*, já foi possível observar que a versão atual da ferramenta não suporta a geração do modelo SR, apesar de sua interface conter as opções para a configuração deste modelo. Isto impediu a aplicação da ferramenta na construção de um editor gráfico capaz de modelar relacionamentos dentro da fronteira dos atores, que são as ligações de meio-fim, decomposição de tarefas e ligações de contribuição. Dessa forma, o trabalho se resumiu a avaliar a ferramenta Agile Tool na sua capacidade de criar editores que pudessem modelar apenas elementos presentes no modeloSD das linguagens variantes do i\*. A proposta inicial era de avaliar o Agile Tool com as seguintes linguagens: i\* wiki, Tropos, GRL e i\*-c. Com o aprofundamento do estudo dessas linguagens, foi concluído, através de seus metamodelos, que as três primeiras possuíam o modelo SD sem muitas variações. A maioria das diferenças entre elas estariam justamente no modelo SR. Como não seria possível avaliar a ferramenta com relação à modelagem dos modelos SR suportados por estas variantes do i\*, foi preferível eliminar da avaliação as linguagens i\*wiki, Tropos e GRLe adicionar duas linguagens onde os modelos SD possuem mais variações. Sendo assim, o trabalho irá avaliar o Agile Tool usando as liguagens i\*-c, i\*-orthogonal e i\* com concerned objects, apresentados no capítulo 2.

### 3.1. Utilizando as variantes

Utilizando o metamodelo apresentado na subseção 2.7.1 para o i\*-c, observamos que existe a metaclasse Cardinality. Este elemento define a cardinalidade dos elementos intencionais e do relacionamento*MeansEndLink*. Como o relacionamento *MeansEndLink* não faz parte do modelo SD, o foco foi apenas no elementos intencionais. Precisaríamos então definir os possíveis tipos de cardinalidade para estes elementos. Numa visão por alto, seria necessário criarmos um atributo do tipo *enumeration*nos elementos intencionais para definir os tipo de cardinalidade: obrigatório ou opcional.Observando a estururada ferramenta Agile Tool, pudemos concluir que, uma vez que a base do i\* tenha sido finalizada, não é possível alterar as configurações dosseus elementos. Isto significa dizer que não era possível adicionar um novo atributo nos elementos intencionais (no caso, tarefa e recurso) que haviam sido configurados (e reusados) da base do i\*. Sendo assim, tentamos criar novos elementos de modelagem do tipo *Element*(por exemplo, tarefa com cardinalidade, recurso com cardinalidade) que poderiam receber os atributos de cardinalidade. A ferramenta permitiu, através de sua interface, a configuração dos novos elementos de modelagem do tipo *Element* e seus atributos do tipo *Enumeration* contendo os possíveis tipos de cardinalidade. Porém, após toda a configuração,o GMF não consegue fornecer o modelo gerador que é usado para gerar o código executável do editor gráfico, impedindo a criação do mesmo. Esta foi a segunda limitação identificada no Agile Tool.

Numa tentativa de superar esta limitação, foi necessário adaptar o metamodelo do i\*-c da seção 2.7.1 às restrições impostas pela ferramenta Agile Tool (Figura 2.24). Assim, remodelamos o metamodelo do i\*-c, adicionando mais duas metaclasses:

* CardinalityLinkElement, a ligação entre o elemento Cardinality e os elementos intencionais.
* CardinalityLinkME, a ligação entre o elemento Cardinality e ligações do tipo MeansEndLink.

Agora, um novo elemento chamado *Cardinality*pode estar composto tanto na metaclasse*Model* e como na metaclasse*Compartment*. Assim, este elemento pode estar ligado a um elemento intencional através da ligação *CardinalityLinkElement*, e também a uma relação do tipo MeansEndLink através da ligação CardinalityLinkME. A figura a seguir mostra o novo metamodelo para o i\*-c.

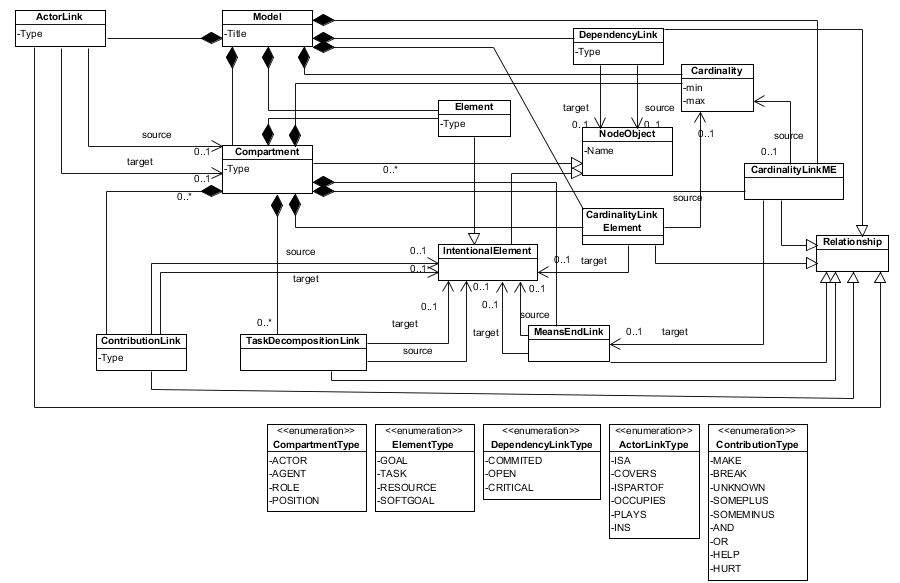


Figura 24 - Metamodelo para a linguagem i\*-C adaptado para o Agile Tool

Utilizando este novo metamodelo para o i\*-c, vemos que deve ser criado um novo elemento chamado *Cardinality*(no metamodelo apresentado na seção 2.7.1, *Cardinality* era um atributo de um *IntentionalElement* ou de um *MeandEndLink*). Este elemento define a cardinalidade dos elementos intencionais e do relacionamento *MeansEndLink*. Como o relacionamento *MeansEndLink* não faz parte do modelo SD, o foco foi apenas no elementos intencionais.Foi visto também na seção 2.7.1 que os elementos intencionais que podem apresentar cardinalidade são os elementos *Resource* e *Task*. Sendo assim, o elemento *Cardinality* só poderá se relacionar com os elementos *Resource* e *Task.*

Foi possível criar, com o uso da ferramenta (Figura 3.2), o novo elemento (*Cardinality*) dentro do modelo SD. A restrição do relacionamento entre o elemento Cardinality e os elementos *Task* e *Resource* deveria ser realizada quando fosse criado um novo elemento de ligação entre eles. A ferramenta Agile Tool permite a configuração desta nova ligação(Figura 3.3) e de suas restrições (Figura 3.4), que chamamos de *CardinalityLinkElement*. Porém, após concluir a configuração, ocorre o mesmo problema relatado anteriormente, e o editor gráfico não pode ser gerado.

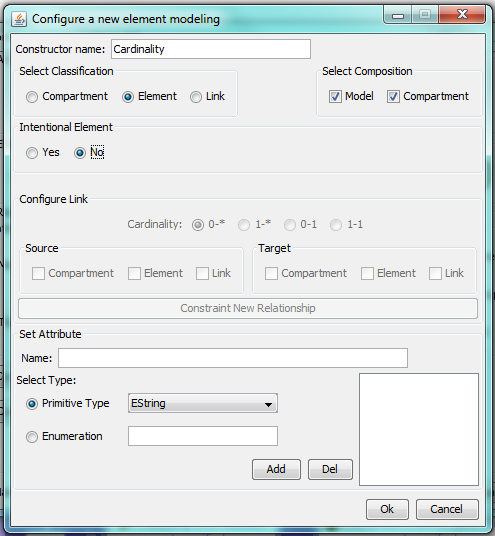


Figura 25 Configuração do novo elemento *Cardinality*

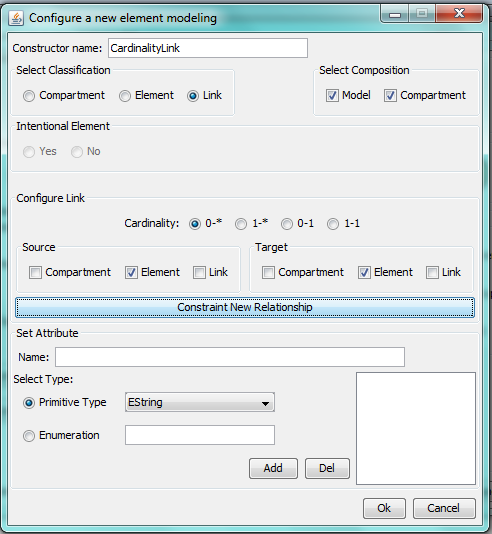


Figura 26 - Configuração do novo elemento *CardinalityLink*

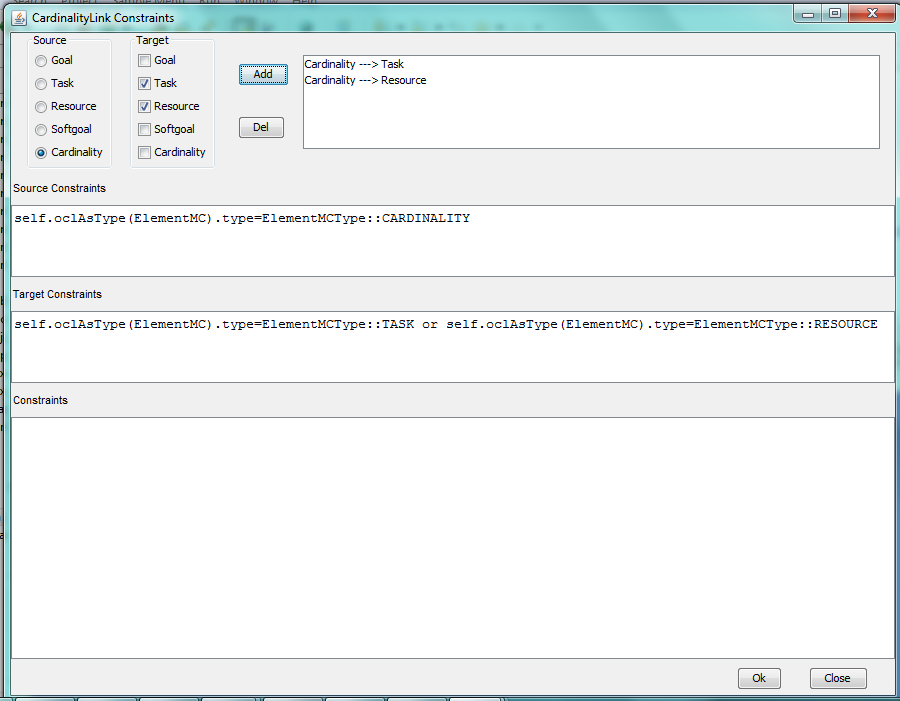


Figura 27 - Configuração das restrições da ligação *CardinalityLink*

Para tentar descobrir em que parte ocorria o problema, primeiramente tentamos gerar o editor gráfico apenas utilizando o novo elemento *Cardinality*, além da base do i\*. Observamos que a ferramenta foi gerada corretamente, diponibilizando o elemento*Cardinality* na barra de ferramentas do editor gráfico. Comparando com o procedimento feito antes da tentativa de descobrir o problema, podemos ver que, como a única configuração feita após a inclusão do novo elemento de modelagem do tipo *Element*(elemento *Cardinality*) foi a inclusão do elemento do tipo *Link*(elemento *CardinalityLinkElement*), concluímos que a ferramenta possui limitações quanto a criação de novos relacionamentos.

Como podemos ver na figura 3.5, quando o editor gráfico conseguiu ser criado,foi possível inserir o elemento de cardinalidade na tela. Este é criado na forma gráfica retangular e a cardinalidade é inserida pelo usuário.O elemento de cardinalidade está solto, poisa ferramenta não consegue criar o novo tipo de relacionamento para ligá-lo ao elemento intencional *resource*. Nesta mesma figura se encontra uma releção de dependência de recurso entre dois atores. Podemos identificar uma falha na ligação de dependência, já que não é possível saber o sentido do relacionamento, pois a ligação não possui a forma gráfica padrão utilizada pelo framework i\*, visto no capítulo 2.

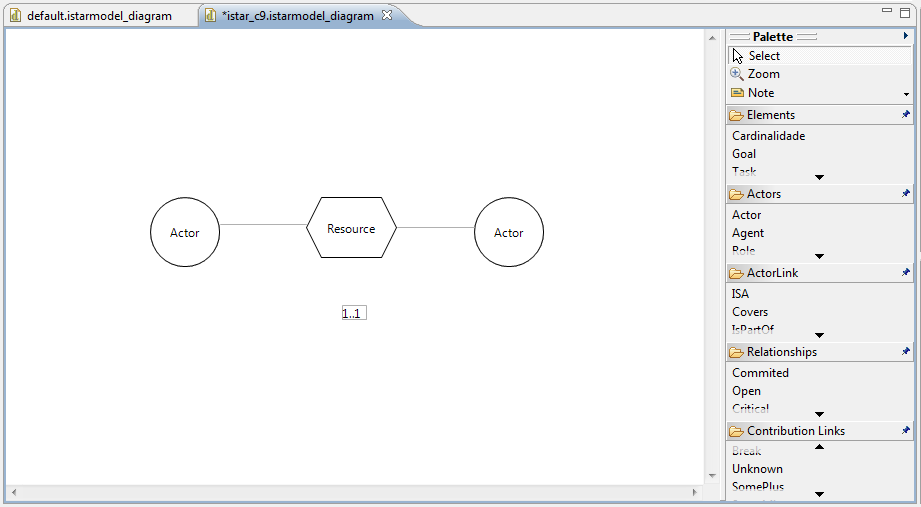


Figura 28 - Editor gráfico gerado para o i\*-C com o elemento de cardinalidade e uma relação de dependência de recurso entre atores

Analogamente a criação dos novos elementos para a linguagem i\*-c, foram criados os novos elementos para o i\*-orthogonal e i\* com concerned objects. Segundo o metamodelo i\*-orthogonal, apresentado na subseção 2.7.2, além do elemento de cardinalidade, um novo elemento de modelagem do tipo *Element* foi adicionado, chamado *Context*. Este elemento é usado para determinar se certo relacionamento será ativado em um modelo. Segundo o metamodelo apresentado, este elemento é composto na metaclasse Relationship. O Agile Tool não suporta a criação de novos elementos compostos dentro de um Relacionamento. Sendo assim, foi necessário remodelar o metamodelo do i\*-Orthogonal para tentar avaliar a criação de seu editor gráfico usando a ferramenta(Figura). Agora existe uma metaclasse *ContextLink* que é um relacionamento entre um elemento do tipo *Link* e o elemento *Context*. A metaclasse *Context* agora não é mais composta na metaclasse Relationship, e sim nas metaclasses *Model* e *Compartment*.

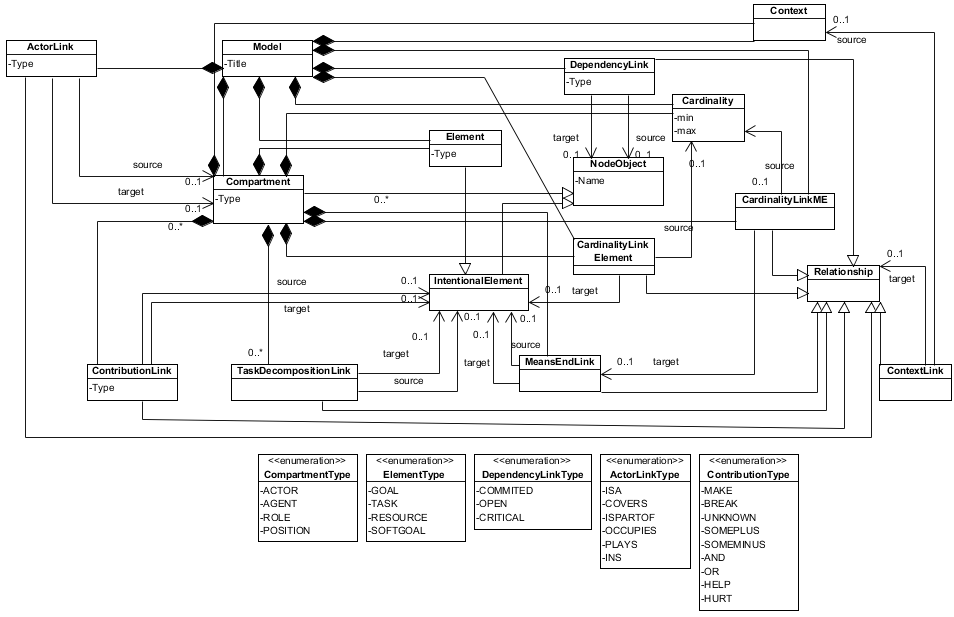


Figura 29 - Metamodelo para a linguagem i\*-Orthogonal adaptado para o Agile Tool

A ferramenta Agile Tool permitiu a configuração de todos os elementos inseridos. Porém, quando foi tentado gerar o editor gráfico, o GMF não consegue fornecer o modelo gerador que é usado para gerar o código executável para sua criação. Executando somente a configuração do elementos *Context*, o editor consegue ser criado, como mostrado na figura a seguir.

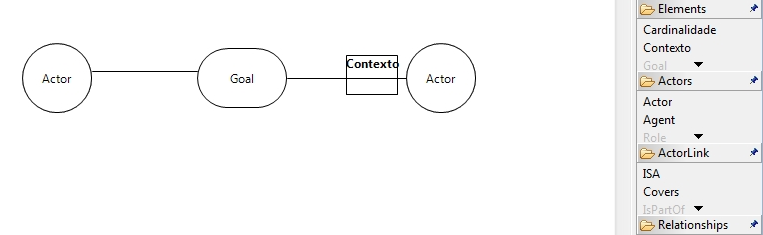


Figura 30 - Editor gráfico gerado para o i\*-Orthogonal com o elemento de contexto ligado a uma ligação de dependência de objetivo entre atores

Finalmente, observando o metamodelo do i\* com c*oncerned object*, apresentado na subseção 2.7.3, devemos criar quatro novos elementos, sendo dois deles elementos de modelagem do tipo Element, representados pelas metaclasses *ConcernedObject* e *AttribConcerned* e dois do tipo Link, representando as metaclasses de relacionamento *ConcernedLink* e*LinkAttribConcerned*. O metamodelo da subseção 2.7.3 já foi criado de forma que pudesse se avaliado utilizando a ferramenta Agile Tool. Como ocorreu nas configurações dos elementos das outras linguagens utilizadas anteriormente, os elementos do tipo *Element* conseguem ser gerados a partir do AgileTool. Porém, a configuração dos elementos do tipo *Link* comprometem a geração do editor gráfico.

Executando somente a configuração do elementos *ConcernedObject* e *AttribConcerned*, o editor consegue ser criado, como mostrado na figura abaixo. Podemos perceber que os novos elementos são criados na forma gráfica retangular, diferentemente da forma gráfica circular que é a forma proposta pelo i\* com *concerned object*.

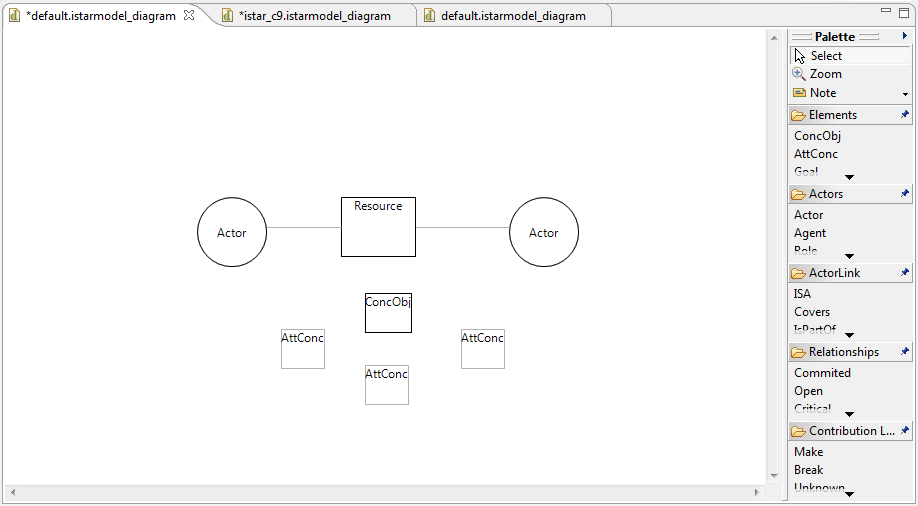


Figura 31 - Editor gráfico gerado para o i\*com concerned obects com o elemento de Concerned object e seus atributos e uma relação de dependência de recurso entre atores

Realizados os testes possíveis com a ferramentaAGILE Tool, foram identificadas as seguintes limitações:

* A ferramenta não suporta a criação do modelo SR, apesar de sua interface permitir as configurações para isso;
* Modelos que possuem a configurações de elementos com atributos do tipo *Enumeration* não conseguem tero editor gráfico gerado;
* Não é possível gerar o editor gráfico com modelos que possuem novos elementos do tipo relacionamento;
* A forma gráfica das ligações de dependência é diferente da forma proposta pelo framework i\*, de forma que não é possível identificar qual o sentido do relacionamento;
* A forma gráfica dos novos elementos do tipo *Element* são sempre no formato retangular, dificultando a distinção entre os elementos do modelo.

Podemos concluir que a versão atual da ferramenta Agile Tool não atende completamente ao seu propósito que é a geração de editores de gráficos para linguagens variantes do i\* a partir da configuração dos elementos presentes nestas linguagens.

## Capítulo 4 – Conclusão

Neste último capítulo são apresentadas as considerações finais sobre este trabalho, as contribuições que este trabalho proporcionou, as suas principais limitações e os trabalhos futuros.

## 4.1. Considerações finais

O presente trabalho teve seus objetivos alcançados a partir do uso da ferramenta Agile Tool para a criação de editores gráficos para as linguagens i\*-C, i\*-Orthogonal e i\* com concerned objects. Para cada linguagem, foram feitas suas devidas configurações no Agile Tool e, em seguida, as tentativas de geração dos seus respectivos editores gráficos. Para cada falha na geração do editor, procurou-se identificar quais etapas de criação causavam os problemas. Após vários testes, foi possível avaliar a ferramenta no que diz respeito à sua eficácia na criação de editores gráficos para linguagens baseadas no i\*.

## 4.2. Contribuições

O escopo deste trabalho engloba a avaliação daferramenta Agile Tool na criação de editores gráficos para linguagens baseadas no i\*. Para isso, as seguintes atividades foram realizadas:

* Estudo das linguages i\*-wiki e Tropos, embora não tenham sido usados na avaliação por só se diferenciarem do i\* original no que diz respeito ao modelo SR;
* Estudo das linguagens i\*-c, i\*-orthogonal e i\* com concerned objects e a criação dos metamodelos para as linguagens i\*-c, i\*-orthogonal e i\* com concerned objects a partir do metamodelo núcleo das linguagens *i\**(PAES, LIMA, *et al.*, 2011);
* Estudo da ferramenta Agile Tool;
* Uso da ferramenta Agile Tool na criação de editores gráficos para as linguagens supracitadas;
* Documentação e análise das limitações apresentadas pela ferramenta na criação dos editores gráficos.

Este trabalho também contribuiu consideravelmente com o desenvolvimento acadêmico e científico da autora, pois permitiu que a mesma estudasse sobre a criação de editores gráficos utilizando a tecnologia Eclipse, a definição de metamodelos para linguagens de modelagem, a ferramenta Agile Tool e diversas linguagens baseadas no i\*.

## 4.3. Trabalhos Futuros

A partir da utilização da ferramenta Agile Tool na criação de editores gráficos para diversas linguagens baseadas no i\*, algumas limitações foram identificadas, conforme apresentado no final do capítulo 3.

Trabalhos futuros incluem a criação de uma nova versão da ferramenta Agile Tool que considere:

* O desenvolvimento da funcionalidade no Agile Tool que permita que os editores gerados sejam capazes de criar modelo SR;
* Correções no Agile Tool para permitir a geração de editores gráficos para modelos que possuam elementos com atributos do tipo *Enumeration*(ex. Valores mínimos e máximos da cardinalidade de um elemento) e/ou novos tipos de relacionamentos (ex. As ligações que relacionam os *concerned objects* aos elementos intencionais);
* A criação das formas gráficas padrão para todos os elementos baseados no framework i\* (ex. Ligações de dependência);
* O desenvolvimento da funcionalidade no Agile Tool que permita a escolha de formas gráficas mais sofisticadas para a criação de novos elementos de modelagem (ex. Os concerned objects).

Do exposto, é possível concluir que as limitações levantadas poderão ser corrigidas em uma versão futura da ferramenta e esta nova versão poderá ser novamente avaliada seguindo a metodologia apresentada neste trabalho.

# Referências

ALENCAR, F.; CASTRO, J.; LUCENA, M.; SANTOS, E.; SILVA, C.; ARAÚJO, J.; MOREIRA, A. Towards Modular i\* Models. In Requirement Engineering Track, at 25th ACM symposium on AppliedComputing, SAC 2010, Sierre, Switzerland: ACM Press, 2010. p. 292-297.

AMYOT, D.; HORKOFF, J.; GROSS, D.; MUSSBACHER, G. A Lightweight GRL Profile for i\* Modeling. In: Proceedings of RIGIM, 2009. p.254-264.

BROOKS Jr., Frederick P. No Silver – Essence and Accident in Software Engineering. Disponível em: <http://people.eecs.ku.edu/~saiedian/Teaching/Sp08/816/Papers/Background-Papers/no-silver-bullet.pdf>.Acesso em22/04/2012.

Introducing the GMF Runtime. Web-site.Disponível em: <<http://www.eclipse.org/articles/Article-Introducing-GMF/article.html>>

GRAU, G.; HORKOFF, J.; YU, E.; ABDULHADI,S. i\* Guide. 2008. Web-site. Disponível em: <http://istar.rwth-aachen.de>. Acesso em: 22 jan. 2013.

LAPOUCHNIAN, Alexei. Goal-Oriented Requirements Engineering: An Overview of the Current Research, 2005.

LIMA, Carlos Diego Quirino. E-SPL – Uma Abordagem para a Fase de Requisitos na Engenharia de Domínio e na Engenharia de Aplicação com Modelos de Objetivos, 2011.

LUCENA, M. ; SANTOS, E.; SILVA, C.; ALENCAR, F.; SILVA, M.; CASTRO, J. Towards a unified metamodel for i\*. In: RCIS´08- Second IEEE International Conference on Research Challenges in Information Science, 2008, Marrakech,Marrocos. Proceedings of Second IEEE International Conference on Research Challenges in Information Science, 2008. RCIS 2008, p. 237-246, 2008.

PAES, J.; [CASTRO](http://dblp.dagstuhl.de/pers/hd/c/Castro:Jaelson_Brelaz_de.html), J.; [SANTOS](http://dblp.dagstuhl.de/pers/hd/s/Santos:Emanuel.html), E.; [LIMA](http://dblp.dagstuhl.de/pers/hd/l/Lima:Carlos.html), C.; SILVA, C. An Approach to Generate Tools for i\* Languages. [SBES, 2011](http://dblp.dagstuhl.de/db/conf/sbes/sbes2011.html#PaesCSSL11). p. 243-252.

SILVA, C.; BORBA, C.; CASTRO, J. G2SPL: Um Processo de Engenharia de Requisitos Orientada a Objetivos para Linhas de Produtos de Software. In: Proceeding of the 13th Workshop on Requirements Engineering, 2010. p. 1-11.

SUSI, A.; PERINI, A.; MYLOPOULOS, J.The Tropos metamodel and its use. In: Informatica, vol. 29, 2005. p. 401–408.

SWEBOK. Guide to the Software Engineering Body of Knowledge. 2004 Version. A project of the IEEE Computer Society Professional Practices Committee. Disponível em: http://www.swebok.org.

YU, E. Towards modelling and reasoning support for early-phase requirements engineering. Requirements Engineering, Proceedings of the Third IEEE International Symposium on, 1997.