CIN-UFPE

# Desenvolvimento do Syntax Checker para o iStarTool

Trabalho de Graduação

**Átila Valgueiro Malta Moreira**

**Orientadora: Prof.ª Drª. Carla Taciana Lima Lourenço Silva Schuenemann**

Recife – Pernambuco, 2012.

**Desenvolvimento do Syntax Checker para o iStarToll**

Trabalho apresentado à disciplina de Trabalho de Graduação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

## RESUMO

Este trabalho de graduação apresenta a análise dos resultados da avaliação, por uma amostra de usuários, das melhorias no Syntax Checker do iStarTool, desenvolvidas a partir da identificação e correção de alguns problemas estruturais surgidos da inserção do corretor e analisador de modelos i*. O iStarTool é uma ferramenta desenvolvida pelo Laboratório de Engenharia de Requisitos do Centro de Informática da Universidade Federal de Pernambuco e tem como finalidade ser usada na atividade de modelagem de requisitos, com vistas a reduzir o risco de projeto de Softwares e melhorando a sua qualidade final. Acredita-se que as melhorias propostas neste trabalho suavizarão a curva de aprendizagem, facilitando a boa prática da Engenharia de Requisitos.



## ABSTRACT

This graduate work presents the analysis of evaluation results, for a sample of users, of improvements in iStarTool's Syntax Checker, developed from identification and fix of some base problems arising of insertion of fixer and checker of i* models. The iStarTool is a tool developed by Laboratório de Engenharia de Requisitos of Centro de Informática of Universidade Federal de Pernambuco and have as end be used in make models of requeriments, with the objective in reduction of risk in Software Projects and increase end quality of this. It is believed that the improvement proposed in this work soften the learning curve, facilitating good practice in Requirements Engineering.

## AGRADECIMENTOS

Agradeço primeiramente à minha família, em especial meu pai, minha mãe e meu irmão por terem me dado o apoio que precisei durante toda a minha vida até chegar à graduação e, mais fortemente, enquanto precisava ficar finais de semana em casa estudando.

Agradeço também aos colegas e professores que partilharam comigo o período da graduação no Centro de Informática da Universidade Federal de Pernambuco – Cin/UFPE. Dentre todos os professores do CIn/UFPE, quero registrar meu agradecimento especial ao Professor **Jaelson Castro**, com quem pude começar este trabalho e à Professora **Carla Taciana Lima Lourenço Silva Schuenemann** que prosseguiu orientando-me, mesmo durante um período muito atarefado de sua vida, dedicando parte do seu tempo à estruturação, orientações e revisões necessárias à conclusão deste trabalho de graduação.

# Sumário

# Capítulo 1 – Introdução:

O presente capítulo está organizado em quatro partes: inicialmente é apresentada uma visão geral sobre Engenharia de Requisitos (ER) do *Framework i** (YU, 1995), a título de fundamentação teórica; a importância da proposta de trabalho que serviu de motivação para a construção desta monografia; seus objetivos e, por fim, um pouco a respeito de como está estruturado o presente documento.

## 1.1. Considerações Iniciais

A complexidade que envolve a ER tem se constituído motivo de preocupação na abordagem de diferentes autores. Nesse contexto, Brooks (1986) já chamava a atenção para o fato de que a decisão precisa sobre os requisitos de um sistema poderia ser considerada como a fase mais complexa, difícil e custosa de se corrigir no processo de construção de *softwares*.

Mais recentemente, Sommerville (2007) também chamou atenção para a complexidade envolvida na ER. Segundo ele:

> "Talvez o maior problema que enfrentamos no desenvolvimento de sistemas de software grandes e complexos seja o da Engenharia de Requisitos. Ela está relacionada com as definições do que o sistema deve fazer, suas propriedades emergentes desejáveis e essenciais e as restrições quanto à operação do sistema e quanto aos processos de desenvolvimento de *software*" (SOMMERVILLE, 2007, p. 77).

Com base nestes argumentos pode-se afirmar que a ocorrência de falhas durante os processos de ER terminam afetando não somente a fase de produção, mas a própria vida útil dos softwares, conforme previa Brooks (1986). Analisando consequências e soluções para os problemas mais comuns que afligem à ER, Firesmith (2007) destacou as ferramentas de suporte inadequadas e o despreparo dos profissionais.

Além desses, a ênfase restrita aos requisitos funcionais, os requisitos não rastreados ou ausentes, a baixa qualidade dos requisitos, a ocorrência de restrições inapropriadas, a excessiva volatilidade dos requisitos, a verificação inadequada da qualidade dos requisitos e

sua validação inadequada, o gerenciamento inadequado dos requisitos e metodologias inadequadas, foram outros problemas apontados por Firesmith (2007).

Segundo Wiegers (2001 apud FIRESMITH, 2007), aproximadamente 50 por cento de defeitos dos produtos se originam nos requisitos, 80 por cento do esforço de retrabalho num desenvolvimento de projeto pode ser atribuído à má elaboração dos requisitos. Tais defeitos, segundo *Health and Safety Executive* (HSE,1995 apud FIRESMITH, 2007), são a causa de mais de 40 por cento dos acidentes que envolvem os sistemas de segurança crítica, podendo causar destruições e até mesmo mortes.

Levando em conta esses dados, é possível afirmar que existem vantagens em aumentar a ênfase nas fases iniciais (especificação e análise de requisitos) durante o desenvolvimento de softwares. Tal prática irá proporcionar a redução de falhas no processo, os retrabalhos e todas as suas consequências, quer no âmbito do processo em si, quer na sua composição de custos, e ainda sobre o seu resultado final, - o *software*. Para tanto, ao longo do tempo, foram propostas diversas possíveis soluções, dentre elas o Framework i*, que se trata de um diagrama com um conjunto de regras que tem como objetivo ajudar no levantamento dos requisitos do sistema. O Framework i* será apresentado em mais detalhes no decorrer do presente texto.

## 1.2. Motivação

No cenário apresentado na parte anterior, um dos problemas encontrados é o fato das ferramentas existentes para ER geralmente não oferecerem um bom suporte ao usuário (FIRESMITH, 2007). Tal aspecto termina dificultando o processo de aprendizado e utilização de linguagens que permitam modelar os sistemas de forma apropriada.

Com essa preocupação, no Centro de Informática da Universidade Federal de Pernambuco – Cin/UFPE, Santos (2008a) desenvolveu o *iStarTool*, ferramenta cujo objetivo é dar suporte à modelagem i* nas fases iniciais de desenvolvimento de *softwares* e, como parte dela, exatamente no intuito de facilitar o aprendizado da linguagem i*, foi criada uma primeira versão do *Syntax Checker* (MALTA et al., 2011) rudimentar, que precisa ter suas funcionalidades ampliadas para favorecer o seu uso e alcançar o seu propósito.

O *Syntax Checker*, ou validador sintático, se trata de uma ferramenta que valida se determinados modelos seguem as regras de sua linguagem. A maioria das ferramentas computacionais desenvolvidas para públicos grandes apresentam seus checadores sintáticos, como pode ser visto desde compiladores que mostram possíveis erros código até em programas de edição de texto onde o programa pode apontar erros como uma pontuação gramaticalmente errada, para isso são utilizadas as regras da linguagem a ser analisada.

O *iStarTool* é uma ferramenta CASE (*Computer Assisted Software Engineering*). Ferramentas CASE como o próprio nome diz são ferramentas que auxiliam à engenharia de software. Estas ferramentas podem ser melhoradas através de checadores sintáticos similares ao *Syntax Checker*, visto que a validação de diagramas ou estruturas geradas pelas ferramentas CASE pode evitar erros no desenvolvimento de *softwares*.

Estima-se que a finalização do *Syntax Checker* poderá repercutir na diminuição do custo de aprendizado de uso da ferramenta e da linguagem i*, ampliando o número de usuários e, consequentemente, proporcionando melhorias significativas nas boas práticas de modelagem de *software*. Noutras palavras, a iniciativa é direcionada aos desenvolvedores de sistemas complexos, que poderão usar a ferramenta e, com isto, reduzir o custo dos projetos, bem como, durante a vida útil do *software*, reduzir o custo de inserção de novas funcionalidades no mesmo.

## 1.3. Objetivos

### Geral:

Ampliar as funcionalidades do *Syntax Checker* do *iStarTool*, ferramenta de suporte à modelagem de requisitos com a linguagem i*, adotada pelo grupo de pesquisa de Engenharia de Requisitos do Cin-UFPE (LER - Laboratório de Engenharia de Requisitos).

### Específicos:

Facilitar a modelagem de requisitos com diagramas i*;

Submeter a nova verão da ferramenta à avaliação de usuários finais.

## 1.4. Estrutura do documento

Além deste capítulo, de caráter introdutório, o presente documento está estruturado da seguinte forma:

Capítulo 2 – Fundamentação teórica: Neste capítulo, o objetivo será demonstrar com mais detalhes a Engenharia de Requisitos, o *Framework* i*, as Ferramentas CASE que dão suporte ao *Framework* i* e um pouco sobre o *Graphical Modelling Framework* (GMF) para criação de editores gráficos no Eclipse.

Capítulo 3 – *iStarTool*: Aqui será apresentada uma visão mais aprofundada da ferramenta desenvolvida pelo grupo LER-Cin-UFPE, além de uma discussão mais detalhada do *Syntax Checker*.

Capítulo 4 – Implementação de melhorias e avaliação da ferramenta: Neste capítulo será explicado detalhadamente quais as modificações feitas na ferramenta para este trabalho, além dos resultados da avaliação da nova versão da ferramenta pelos alunos de uma turma de graduação.

Capítulo 5 – Conclusão: Este capítulo apresenta as conclusões finais acerca do trabalho apresentado e as considerações para trabalhos futuros.

Apêndice – onde será apresentado o formulário desenvolvido pelo autor para fins de submissão à amostra de usuários que avaliou as melhorias implementadas no *Syntax Checker* do *iStarTool*.

# Capítulo 2 – Fundamentação Teórica

Neste capítulo são apresentadas as fundamentações teóricas a respeito da Engenharia 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 que elas são a base para a construção deste trabalho.

## 2.1. Engenharia de Requisitos

Devido ao fato deste trabalho retratar um *plugin* para uma ferramenta que tem como propósito ajudar engenheiros de *software* a modelar os requisitos de seus projetos, torna-se pertinente apresentar uma breve descrição do que é Engenharia de Requisitos.

Segundo Sommerville (2007):

> "Os requisitos de um sistema são descrições dos serviços fornecidos pelo sistema e suas restrições operacionais. Esses requisitos refletem as necessidades dos clientes de um sistema que ajudam a resolver algum problema, por exemplo, controlar um dispositivo, enviar um pedido ou encontrar informação. O processo de descobrir, analisar, documentar e verificar serviços e restrições é chamado de engenharia de requisitos (RE – *Requeriments Engineering*)" (SOMMERVILLE, 2007, p. 79).

Para se entender melhor a Engenharia de Requisitos, será apresentada uma classificação dos requisitos de software e alguns processos de Engenharia de Requisitos.

### 2.1.1. Classificação de requisitos

Na engenharia de requisitos existe a necessidade de classificar os requisitos em tipos, sendo os principais deles os requisitos funcionais e os requisitos não funcionais.

Os requisitos funcionais, como o próprio nome diz, são requisitos que descrevem a funcionalidade de um sistema. Os requisitos funcionais podem descrever como o sistema deve se comportar e ainda podem descrever como o mesmo não pode se comportar.

Por outro lado, requisitos não funcionais(RNFs) não se encontram ligados a funções específicas do programa. Os requisitos não funcionais geralmente estão ligados a restrições ou

aspectos de qualidade no sistema, que permite descobrir como os requisitos funcionais devem ser implementados.

Um exemplo de requisito funcional e não funcional poderia ser: no caso do funcional possibilitar criar uma conta de acesso ao sistema, e não funcional poderia ser que toda a conta deve ser segura o suficiente para evitar acesso de terceiros.

### 2.1.2. Processos de engenharia de requisitos

Segundo Sommerville (2007), engenharia de requisitos é composta por uma série de processos iterativos e incrementais. Estes processos são:

- Estudo de viabilidade – Nesse processo além de entender a viabilidade tecnológica, deve se estudar se o software realmente contribui para os objetivos dos *stakeholders i.e.*, qualquer pessoa envolvida no processo;
- Elicitação e análise de requisitos – Neste processo é feita uma análise para entender as expectativas e aspirações que os *stakeholders* possuem no sistema;
- Validação de requisitos – Após a elicitação e análise de requisitos, tem-se a validação dos mesmos. Para tanto é muito importante a participação dos *stakeholders;*
- Gerenciamento de requisitos – Como os requisitos costumam mudar muito do começo do processo até o *software* estar pronto, então existe uma fase responsável por gerenciar tais mudanças. Não necessariamente esta fase é executada depois das outras, visto que mudanças podem ocorrer em qualquer momento do processo.

## 2.2. *Framework* i*

O *Framework* i* (leia-se “i-estrela”) foi apresentado pela primeira vez por Eric Yu em sua tese de doutorado em 1995 na Universidade de Toronto.

Segundo Yu (1995) entre suas contribuições estão o aperfeiçoamento na forma como se retratam atributos não funcionais e os processos que envolvem humanos e máquinas. Para tanto, foi necessário trabalhar com dois modelos que serão explicados logo adiante, ainda neste capítulo, quais sejam: o modelo de dependência estratégica (também chamado de SD *Model*, *Strategic Dependency Model*) e o modelo de razão estratégica (também chamado de SR *Model*, *Strategic Rationale Model*).

Antes disto, entretanto, convém trazer alguns conceitos do *Framework* i*, fundamentais para a explicação dos modelos SD e SR, que serão tratados.

### 2.2.1. Conceitos do *Framework* i*:

A título de apresentar algumas explicações sobre os elementos que compõem o modelo, notadamente seus principais componentes, neste item são tratados alguns aspectos sobre Atores, suas formas especializadas e relações estabelecidas entre eles.

- Ator (do inglês, *Actor*): Podem ocorrer da forma genérica ou de uma das três formas especializadas.
    - Agente (do inglês, *Agent*): Geralmente se refere a um ator concreto, um exemplo são *stakeholders* ou máquinas físicas.
    - Papel (do inglês, *Role*): Representa características abstratas inseridas em um Ator.
    - Posição (do inglês, *Position*): Representa uma entidade intermediária entre Agente e Papel, podendo ser utilizada para representar a ocupação de um Agente em determinado contexto.

No entanto, o conceito de Ator e das suas especializações só se torna expressivo quando inseridos os elementos de relacionamento. Existem seis tipos de relacionamento entre atores (YU, 1995), representados na Figura 1, a seguir. Estes são:

- ISA (faz referência a *is a*, que em português significa "é um"): representa uma especialização, geralmente usada para diferenciar grupos. Esta ligação só pode ocorrer entre atores do mesmo tipo, ou entre qualquer tipo de Ator e Atores do tipo genérico.
- *Is-part-of* (faz referência a *is part of*, que em português significa "é parte de"): nesta relação pode-se fragmentar um Ator em subpartes especializadas. Esta ligação só pode ocorrer entre atores do mesmo tipo, ou entre qualquer tipo de Ator e Ator do tipo genérico.
- *Plays* (em português pode ser traduzido "como faz"): é utilizado para ligar agentes a seus respectivos papéis.
- *Covers* (em português significa "cobre", do verbo cobrir): é usado para descrever uma relação entre posição e papel.
- *Occupies* (em português significa "ocupa"): esta ligação serve para simbolizar qual o papel que um determinado agente ocupa.

- INS: representa uma instância específica de uma entidade mais geral. Este tipo de ligação ocorre apenas entre Atores do tipo Agente.

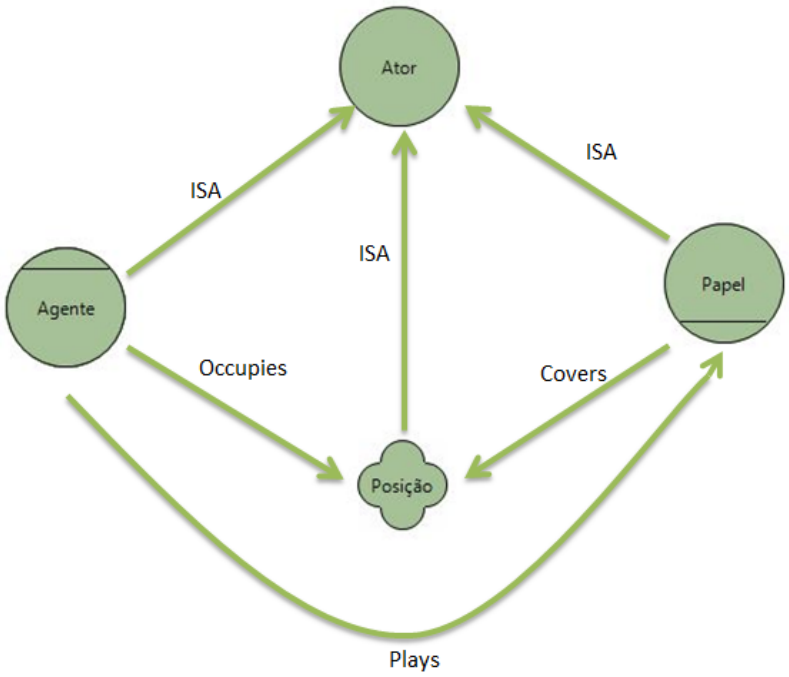



**Figura 1. Representação dos tipos de Atores e de algumas formas de relação entre os mesmos (SILVA JUNIOR, 2011, adaptado pelo autor).**

### 2.2.2. SD *Model*

Com os conceitos básicos do *Framework* tratados pode se começar a descrição dos modelos previamente comentados. No caso, como o título desta subseção propõe, a mesma está dedicada especificamente ao modelo SD.

Segundo Yu (1995) este modelo consiste de um conjunto de nós e ligações. Cada nó representa um ator, e cada ligação entre estes representa uma dependência (Figura 2). O ator que depende é conhecido como ***depender*** e o ator de quem se depende é conhecido como ***dependee***. É também adicionado um objeto no qual a dependência é centrada, sendo este objeto denominado ***dependum***.

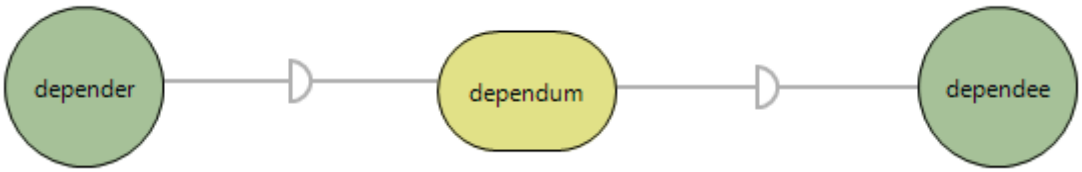



**Figura 2. Representação esquemática de um conjunto de nós e ligações entre atores (SILVA JUNIOR, 2011, adaptado pelo autor).**

Os *dependum* podem ser de quatro tipos, representados na Figura 3:

- *Goal* (que em português significa "objetivo"): Geralmente usado para macros, que são objetivos do ator.
- *Task* (em português significa "tarefa"): Associado ao sentido de tarefa seria similar a executar uma função.
- *Resource* (em português significa "recurso"): Como o próprio nome diz é algo mais concreto.
- *Softgoal*: Esse elemento é uma variação do *goal* para representar coisas menos tangíveis logo, este tipo de elemento tem a capacidade de fornecer um mapeamento melhor para requisitos não funcionais.

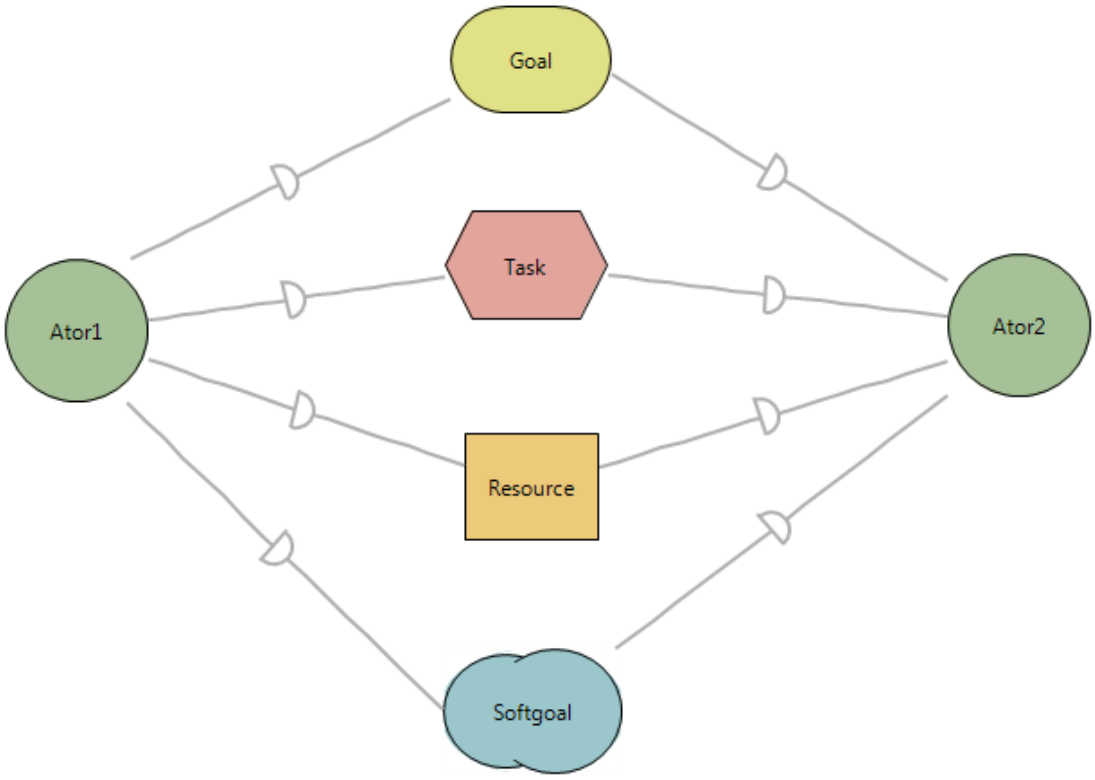



**Figura 3. Representação esquemática dos tipos de *dependum* (SILVA JUNIOR, 2011, adaptado pelo autor).**

Ainda neste modelo existem identificadores de graus de dependência e/ou vulnerabilidade entre atores, representados na Figura 4 a seguir apresentada. Existe um total de três modificadores para as ligações, são eles:

- *Open* (que em português significa "aberta"): esta significa que, em caso de falha, o ator *depender* não será afetado. Não possui marcação de identificação
- *Commited* (que em português significa "comprometida"): esta, em caso de falha, o ator *depender* será afetado, porém não de forma grave. Para representar esta dependência deve se colocar um "O" na ligação.
- *Critical* (que em português significa "crítica"): neste caso, como o próprio nome indica, em caso da dependência não ser cumprida o *depender* é afetado de forma grave. Esta ligação é marcada com um "X".

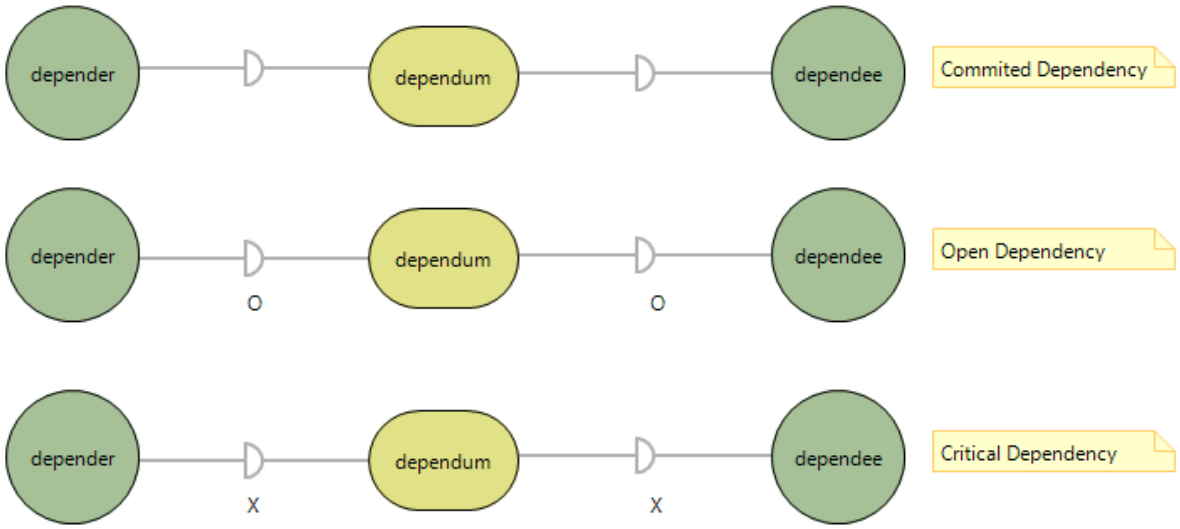



**Figura 4. Identificadores de graus de dependência e/ou vulnerabilidade entre atores (SILVA JUNIOR, 2011, adaptado pelo autor).**

### 2.2.3. SR *Model*

O SD *Model*, embora seja bastante expressivo, ainda não é o suficiente para se passar como cada ator funciona internamente. Objetivando melhorar o poder do *Framework* i* Yu (1995) propôs o SR *Model*. Neste modelo existe um aprofundamento no funcionamento interno de cada ator, para tanto foi inserido o conceito de fronteira do ator, representada na Figura 5, a seguir.

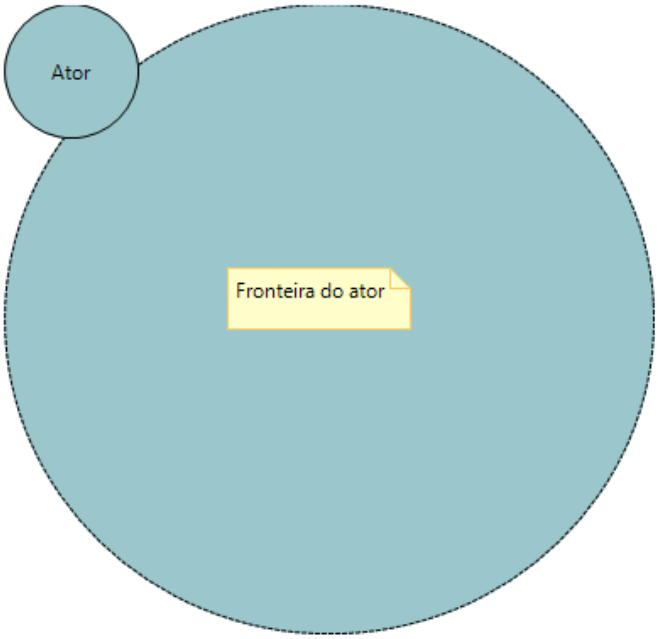



**Figura 5. Representação esquemática do conceito de fronteira do ator (SILVA JUNIOR, 2011, adaptado pelo autor).**

Dentro da fronteira do ator estão colocados elementos baseados nos *dependum* do SD *Model*, quais sejam: *Goal*, *Tasks*, *Resources* e *Softgoals* (Figura 6). Também são inseridas três classes novas de ligações: as *means-ends links* (que em português significa "ligações meio-fins"), as *task decomposition links* (que em português significa "ligações de decomposição de atividades") e as *contribution links* (que em português significa "ligação de contribuição").

*Means-ends links* pode ser um objetivo a ser satisfeito, um recurso a ser produzido, uma tarefa a ser executada ou também um *softgoal* a ser satisfeito. O meio geralmente é expresso por uma tarefa, mas também pode ser um objetivo ou um *softgoal*. Caso um meio seja expresso por um objetivo, o fim também deve ser um objetivo. Também é possível um *means-end link* entre *softgoal*, desde que algum *softgoal* seja satisfeito por uma *task*.

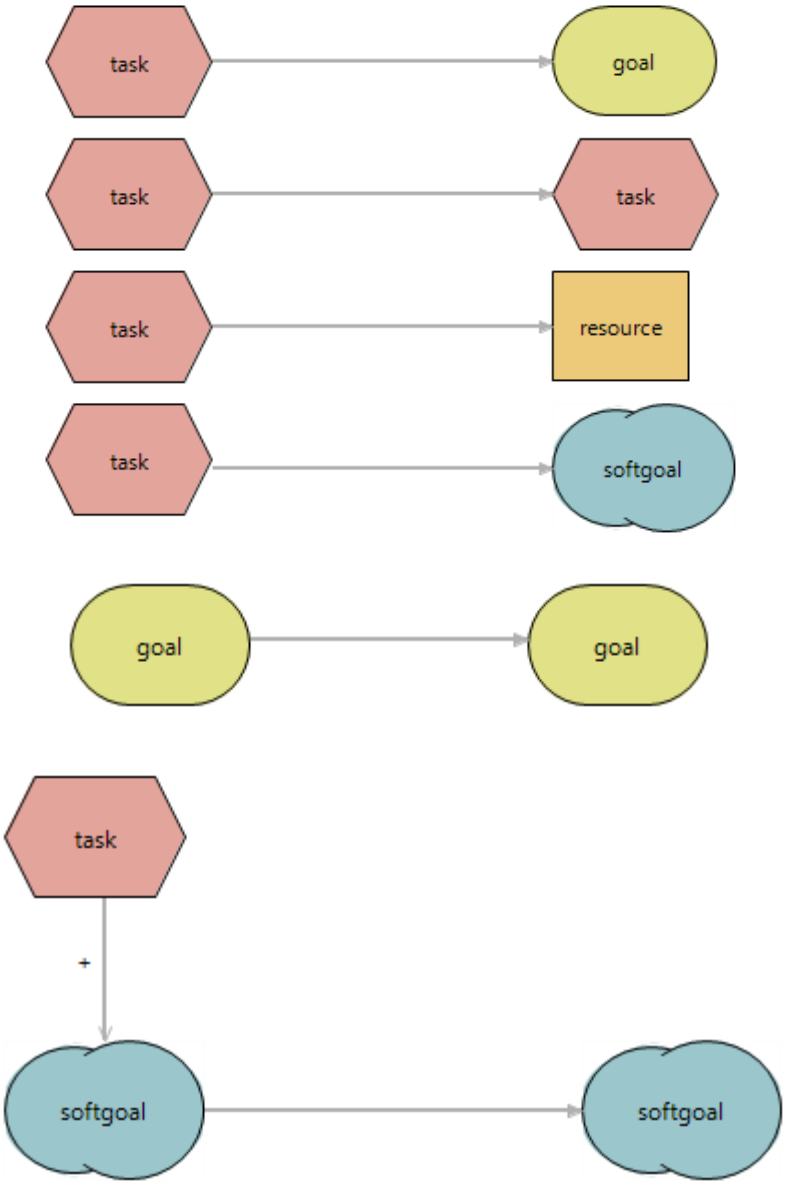



**Figura 6. Representação esquemática da ligação *Means-ends* (SILVA JUNIOR, 2011, adaptado pelo autor).**

*Task decomposition links* como o próprio nome diz é uma ligação específica entre *tasks*. O principal objetivo desta ligação é quebrar *tasks* complexas em blocos mais simples. Podem existir diversos elementos conectados a uma mesma *task* (Figura 7).

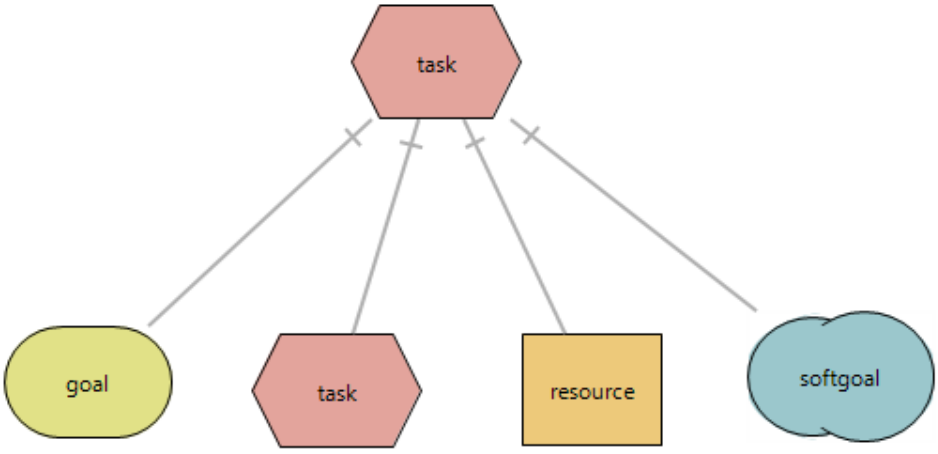



**Figura 7. Representação esquemática da ligação de *Task decomposition* (SILVA JUNIOR, 2011, adaptado pelo autor).**

E, finalmente, as *contribution links* que são usadas nas *softgoals* e permitem medir o grau de satisfação de uma *softgoal*, lembrando que *softgoal* serve para requisitos não funcionais e como isso é abstrato, tal característica pede também uma ligação abstrata. No caso da ligação de contribuição, assim como as de dependência, ela recebe modificadores que permitem expressar a espécie de contribuição com mais facilidade (Figura 8). Estes modificadores são:

- *Make* (que significa em português "fazer"): ligação de aspecto positivo, com força suficiente para satisfazer um *softgoal*.
- *Some* + (que significa em português "algum mais"): ligação de aspecto positivo, com valor de satisfação desconhecido.
- *Help* (que significa em português "ajuda"): ligação de aspecto positivo, fraca.
- *Unknow* (que significa em português "desconhecido"): geralmente é algo que contribui de alguma forma, porém é desconhecido o tipo de contribuição.
- *Hurt* (que significa em português "machuca"): tem aspecto negativo, porém não é forte o suficiente para comprometer uma *softgoal*.
- *Or* (que significa em português "ou"): geralmente se um dos elementos é satisfeito o *softgoal* é satisfeito.
- *And* (que significa em português "e"): nesta ligação o *softgoal* só é satisfeito se todos os elementos de origem forem satisfeitos.

- *Break* (que significa em português “parar”): esta ligação é o oposto simétrico da ligação do tipo *make*.

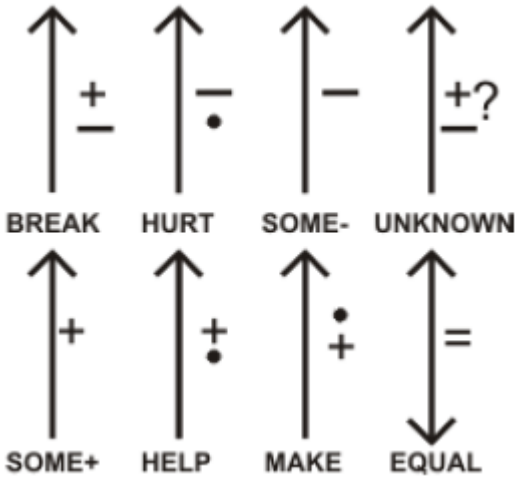



**Figura 8. Representação as contribuições para o *softgoal* (SILVA JUNIOR, 2011).**

## 2.3. Ferramentas CASE de suporte ao i* e principais erros no uso da modelagem i*

Nesta seção serão apresentadas as principais ferramentas CASE de suporte ao i* (daqui por diante mencionadas como ferramentas CASE i*) seguido dos principais problemas que ocorrem na modelagem i* e, finalmente, uma análise aprofundada das ferramentas CASE i* que possuem checadores de sintaxe.

### 2.3.1. Ferramentas CASE i*

Para uma análise das ferramentas i* já disponíveis e uma comparação entre as mesmas, foram consideradas as ferramentas encontradas no i* Wiki (i*Wiki, 2012).

Nesta análise foram avaliadas as ferramentas: *OpenOME*, OME, TAOM4E, *ST-Tool*, J-PRiM, jUCMNav, *SNet Tool*, DesCARTES, *i*-Prefer*, *Measufier* e a ferramenta que possibilitou este trabalho, - a *iStarTool*.

Com o objetivo de facilitar a leitura destas informações, será apresentada a Tabela 1, que contém uma análise inicial das ferramentas disponíveis. Na sequência, considerando as

ferramentas que possuem alguma espécie de checadores sintáticos, será feita uma análise dos respectivos checadores sintáticos.

Desde a criação do *Framework* i* em 1995 por Eric Yu, várias outras linguagens foram propostas tendo como base o Framework i*. Segundo Silva Junior(2011), alguns exemplos de linguagens propostas são:

- i* Wiki: Trata-se de uma versão simplificada voltada ao aprendizado do i*.
- Tropos: Trata-se de uma metodologia para o desenvolvimento de softwares orientados a agentes.
- GRL (do inglês, *Goal-oriented Requiriment Language*): Trata-se de uma linguagem com propósito de amadurecer a modelagem de requisitos não funcionais.
- i*-c: Trata-se de uma modelagem voltada a linha de produto de *software* (LPS).

| Propriedades / Ferramentas | *Framework i** suportado | Permite *SD Model* | Permite *SR Model* | Permite trabalhar paralelamente com SD e *SR Models* | Possui alguma espécie de validação de modelo para o *SD Model* | Possui alguma espécie de validação de modelo para o SR Model | Novas funcionalidades podem ser adicionadas |
|---|---|---|---|---|---|---|---|
| *OpemOME* | Yu'97 | SIM | SIM | SIM | NÃO | NÃO | SIM |
| OME | Yu'97 | SIM | SIM | SIM | NÃO | NÃO | SIM |
| TAOM4E | Tropos | SIM | SIM | SIM | NÃO | NÃO | SIM |
| *ST-Tool* | Tropos<br>Secure Tropos | SIM | SIM | SIM | SIM | SIM | NÃO |
| J-PRiM | Yu'95 | SIM | SIM | SIM | SIM | SIM | SIM |
| jUCMNav | GRL | SIM | SIM | SIM | SIM | SIM | SIM |
| *SNet Tool* | Não informado | NÃO | SIM | NÃO | NÃO | NÃO | NÃO |
| DesCARTES | Yu'95<br>Tropos<br>*Iterative* Tropos | SIM | SIM | SIM | SIM | SIM | SIM |
| *i*-Prefer* | Não informado | SIM | SIM | SIM | SIM | SIM | NÃO |
| *Measufier* | Yu'95 | NÃO | NÃO | SIM | SIM | SIM | NÃO |
| *iStarTool* | Yu'95<br>iStarWiki | SIM | SIM | SIM | SIM | SIM | SIM |

**Tabela 1. Análise comparativa das ferramentas i* disponíveis.**

Conforme pode ser visto na tabela 1, apenas as ferramentas *ST-Tool*, J-PriM, JUCMNav, DesCARTES, *i*-Prefer, Measifier e iStarTool* que possuem checadores sintáticos serão avaliadas mais profundamente ainda neste capítulo.

### 2.3.2. Principais erros que ocorrem no uso do *framework* i*.

Santos (2008b) sistematizou os principais erros que ocorrem em modelagem de diagramas i* agrupando-os em três categorias: atores e relacionamento entre atores; dependências; e, elementos internos e relacionamentos entre elementos internos. Os erros listados por estas categorias são apresentados na Tabela 2, onde nesta dissertação, o autor classificou tais erros já sistematizados por Santos (2008b) em sintáticos e semânticos.

| | Erros que ocorrem em modelagem de diagramas i* | Sintático | Semântico |
|---|---|---|---|
| Atores e relacionamento entre atores | | | |
| 1 | Atores sem ligação | X | |
| 2 | Atores dentro da fronteira de outros atores | X | |
| 3 | Uso de nomes inadequados em atores | | X |
| 4 | Ligar Atores com ligação de dependência sem usar um *dependum* | X | |
| 5 | Ligar um elemento interno ao Ator direto ao Ator | X | |
| Dependências | | | |
| 1 | Uso de outras ligações em vez das ligações de dependências | X | |
| 2 | Uso inadequado de *dependums*. Exemplo: Goal no lugar de softgoal. | | X |
| Elementos internos e relacionamentos entre elementos internos | | | |
| 1 | Deixar elementos sem ligações | X | |
| 2 | Uso de ligações em situações irregulares (ligação de dependência dentro da fronteira do ator) | X | |
| 3 | Elementos do SR fora da fronteira do ator correspondente | X | |
| 4 | Decompor *goals* em *goals* ou *tasks* | X | |
| 5 | Contribuição para *goals*, *tasks* e *resources* | X | |
| 6 | *Means-ends* onde um *goal* é o meio | X | |
| 7 | Ligação direta entre elemento interno de dois atores diferentes | X | |

**Tabela 2. Erros mais comuns cometidos ao elaborar um modelo i* por Santos (2008b) (sistematização entre erros sintáticos e semânticos feita pelo autor).**

Entretanto, como pode ser visto na classificação feita por Santos (2008b), alguns dos erros comuns na linguagem são semânticos[1]. Logo, para a análise ser mais coerente com o

[1] Erros semânticos são a classe dos erros que estão vinculados ao sentido das coisas. (FERREIRA, 1988)

presente trabalho, serão considerados apenas erros sintáticos[2], visto que estes são possíveis de serem tratados pela ferramenta de forma mais simples.

Tanto a descrição da análise que foi realizada no presente trabalho de graduação, como a apresentação mais detalhada da *iStarTool* e do *Syntax Checker* proposto serão desenvolvidas nos capítulos seguintes.

### 2.3.3. Análise aprofundada das ferramentas.

Para alcançar o objetivo final deste trabalho de conclusão de curso, fez-se necessária uma análise mais aprofundada das outras ferramentas atualmente disponíveis para a mesma finalidade.

Conforme já apresentado na Tabela 1 deste documento apenas as ferramentas ST-Tool, J-PriM, JUCMNav, DesCARTES, *i*-Prefer*, *Measifier* e *iStarTool* possuem checadores sintáticos de modelo e, portanto, serão avaliadas mais profundamente.

Cumpre registrar que se tomou por referência a data 18 de março de 2012, quando se tentou realizar os *downloads* das ferramentas constantes na Tabela 1. Naquela ocasião, e noutras tentativas posteriores até a fase de aplicação dos questionários desta pesquisa (procedimento metodológico que será descrito no Capítulo 4 a seguir), as ferramentas ST-*Tool*, J-PriM, DesCARTES e *Measifier* apresentavam-se corrompidas ou com o *link* de *download* quebrado. Isto impossibilitou incluí-las no presente estudo comparativo apresentado na Tabela 3 a seguir que, dada esta razão, contemplou apenas as ferramentas JUCMNav e *i*-Prefer*. Na Tabela 3 pode ser visto como as ferramentas tratam os diferentes erros cometidos durante a modelagem de diagramas i*.

[2] Erros sintáticos são a classe de erros que podem ser analisados sem necessidade de verificação de sentido da linguagem, visto que são erros de construção e disposição de elementos segundos as regras da gramática. (FERREIRA, 1988)

| Principais erros sintáticos (SANTOS, 2008b) | JUCMNav | *i*-Prefer* | *iStarTool* (Sem *syntax checker*) | *iStarTool* (Com *syntax Checke rudimentar*) |
|---|---|---|---|---|
| Atores e relacionamento entre atores | | | | |
| Atores sem ligação | Permite | Permite | Permite | Permite, mas avisa |
| Atores dentro da fronteira de outros atores | Permite | Permite | Não permite | Não permite |
| Ligar Atores com ligação de dependência sem usar um *dependum* | Não permite | Não permite | Permite | Permite, mas avisa |
| Ligar um elemento interno ao Ator direto ao Ator | Não permite | Permite | Permite | Permite, mas avisa |
| Elementos internos e relacionamentos entre elementos internos | | | | |
| Deixar elementos sem ligações | Permite | Permite | Permite | Permite |
| Elementos internos e relacionamentos entre elementos internos | | | | |
| Deixar elementos sem ligações | Permite | Permite | Permite | Permite |
| Uso de ligações em situações irregulares (ligação de dependência dentro da fronteira do ator) | Permite | Permite | Permite | Permite |
| Elementos do SR fora da fronteira do ator correspondente | Permite | Permite | Não permite | Não permite |
| Decompor *goals* em *goals* ou *tasks* | Permite | Permite | Não permite | Não permite |
| Contribuição para *goals*, *tasks* e *resources* | Permite | Permite | Não permite | Não permite |
| *Means-ends* onde um *goal* é o meio | Permite | Permite | Permite | Permite, mas avisa. |
| Ligação direta entre elemento interno de dois atores diferentes | Permite | Permite | Permite | Permite |

**Tabela 3. Comportamento das ferramentas com relação aos erros na elaboração de modelos i*, (removidos os erros semânticos e a análise por ferramentas foi feita pelo autor).**

As características evidenciadas na Tabela 3 permitem, com bastante clareza, se inferir como a inserção do *Syntax Checker* rudimentar permitiu o aumento no grau de maturidade da ferramenta *iStarTool*. Entretanto, é possível perceber que o *iStarTool* ainda permite que alguns erros passem sem ao menos avisar o ocorrido. Esses erros são: deixar elementos sem

ligações, na categoria “Elementos internos e relacionamentos entre elementos internos”; Deixar elementos sem ligações, Uso de ligações em situações irregulares (ex.: ligação de dependência dentro da fronteira do ator) e Ligação direta entre elementos internos de dois atores diferentes, da categoria “Elementos internos e relacionamentos entre elementos internos”.

A inclusão desses checadores sintáticos no *Syntax Checker* do *iStarTool* se constituiu em atividades realizadas no presente trabalho de graduação, conforme será explicado adiante.

## 2.4. GMF – *Graphical Modelling Framework*

Boa parte das ferramentas vistas na seção anterior foram construídas usando o *framework* de modelagem gráfica da *Eclipse Foundation*. Este *framework*, conhecido como GMF, também foi usado para a construção do *iStarTool*.

Segundo Silva Júnior (2011):

> “O *framework* de modelagem gráfica (*Graphical Modelling Framework* – GMF) (GMF, 2010) é um projeto da *Eclipse Foundation* (ECLIPSE, 2010) cujo propósito é fornecer uma infraestrutura para o desenvolvimento de editores gráficos baseados nos *frameworks* – também do Eclipse – EMF (*Eclipse Modeling Framework*) (EMF, 2010), que auxilia a especificação de metamodelos e provê funcionalidades para a geração automática do código Java respectivo, e GEF (*Graphical Eclipse Framework*) (GEF, 2010) utilizado para a criação de editores gráficos genéricos” (SILVA JÚNIOR, 2011, p. 84).

GMF é um *plug-in* para a plataforma Eclipse, que fornece uma série de ferramentas baseadas em modelos Ecore. Este modelo Ecore é o meta-modelo de uma linguagem e contém as principais regras, elementos e ligações da linguagem a ser trabalhada, além de definir a sintaxe de uma linguagem de modelagem. GMF possui também um *pipeline* de trabalho bem definido como pode ser visto em sua interface apresentada na Figura 9 a seguir.

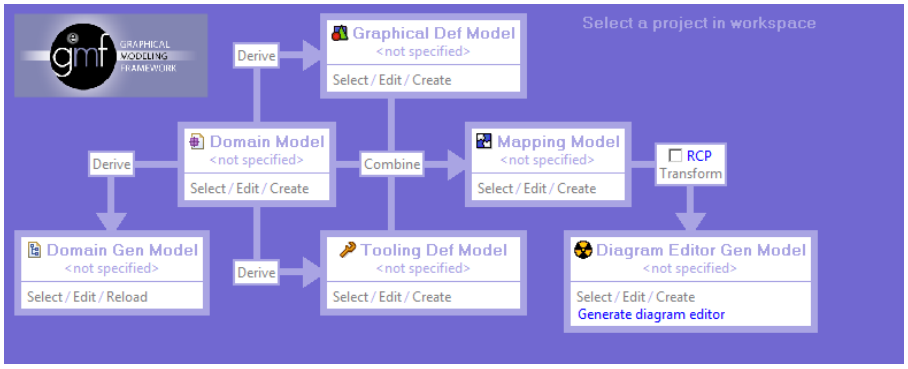



**Figura 9. Imagem da *Dashboard* do GMF (GMF, 2012)**

Na Figura 9 pode-se notar que o *pipeline* de produção do GMF possui passos bem distintos e, muitos deles, com dependências. Este *pipeline* foi transformado por Silva Júnior (2011) em um modelo *Business Process Modeling Notation* (BPMN, 2011) visando facilitar o entendimento, conforme é apresentado na Figura 10. É possível observar ainda que em GMF a construção da ferramenta gráfica começa obrigatoriamente pela criação do Projeto GMF (Figura 10).

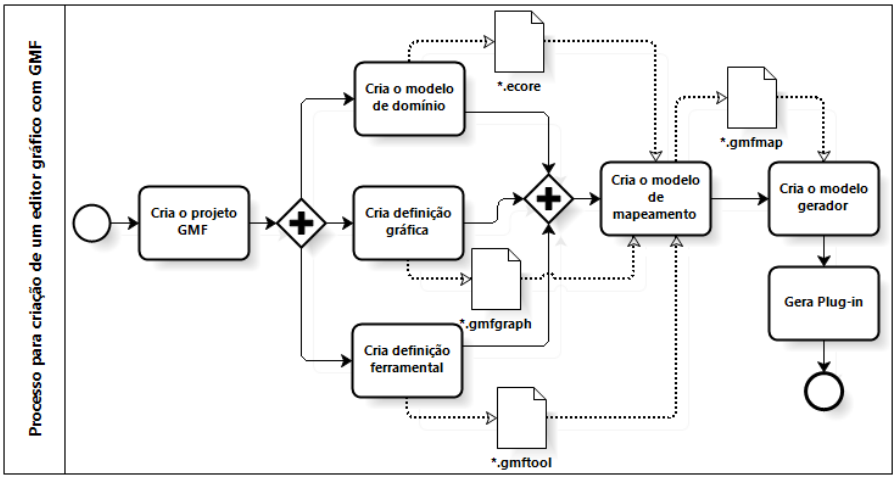



**Figura 10. BPMN descrevendo o processo de criação de ferramentas gráficas usando o *Plug-in* GMF (SILVA JÚNIOR, 2011).**

Após a criação do projeto GMF, os próximos passos são criar o modelo de domínio, a definição gráfica e a definição da ferramenta. Destes três passos citados, a geração do modelo de domínio deve merecer uma atenção especial, justamente por se tratar de um trabalho menos automático.

A geração do modelo de domínio tem como resultado o modelo Ecore, o qual pode gerar tanto definição gráfica quanto a definição da ferramenta de forma simplificada. Após a

construção do modelo Ecore deve-se trabalhar no melhoramento das definições gráficas geradas pelo GMF, para então utilizar a funcionalidade de gerar a ferramenta final.

# Capítulo 3 – *iStarTool*

Este capítulo tem por objetivo a apresentação da ferramenta *iStarTool*, bem como do *Syntax Checker* rudimentar: *plugin* que foi construído sobre o *iStarTool* e alvo principal deste trabalho, que tem por objetivo geral, conforme já explicitado no primeiro capítulo, ampliar as funcionalidades deste *Syntax Checker*.

## 3.1. Visão geral da ferramenta

A ferramenta *iStarTool* é uma ferramenta CASE, que permite a criação de modelos i*. Ela foi desenvolvida como um *plugin* da plataforma Eclipse pela então aluna de Mestrado Bárbara Siqueira Santos sob a orientação do Professor Jaelson Freire Brelaz de Castro, do Centro de Informática da Universidade Federal de Pernambuco – CIN/UFPE (Santos, 2008a).

A ferramenta, por se tratar de um *plugin* para a plataforma Eclipse, tem uma interface mais familiar aos desenvolvedores, visto que a plataforma de desenvolvimento Eclipse é bastante difundida. Segundo Santos (2008a), para a construção do *iStarTool* (Figura 8) foi utilizado o *framework* de modelagem gráfica GMF (*Graphical Modelling Framework*), projeto da *Eclipse Foundation*, como já foi dito no capítulo anterior, que tem como propósito fornecer uma infraestrutura para desenvolvimento de editores gráficos.

A Figura 11 a seguir apresenta uma imagem do *iStarTool* (MALTA, 2011). Nela é possível observar os cinco painéis assinalados e descritos a seguir:

Número 1, que é o *Stage*, local onde é possível criar e editar modelos i*;

Número 2, que é a paleta de ferramentas, na qual podem ser encontrados tanto os elementos, quanto as ligações que constituem o modelo;

Número 3, onde está o *Outline*, que possibilita ser visualizada uma miniatura do modelo que está sendo trabalhado, aspecto bastante útil, sobretudo quando se está trabalhando com modelos muito grandes;

Número 4, conhecido como propriedades, onde se pode mudar nome, tipo e outras propriedades; e, finalmente,

Número 5, situado na parte superior esquerda da imagem, que é o ativador do *Syntax Checker*, que será apresentado em detalhe mais adiante.

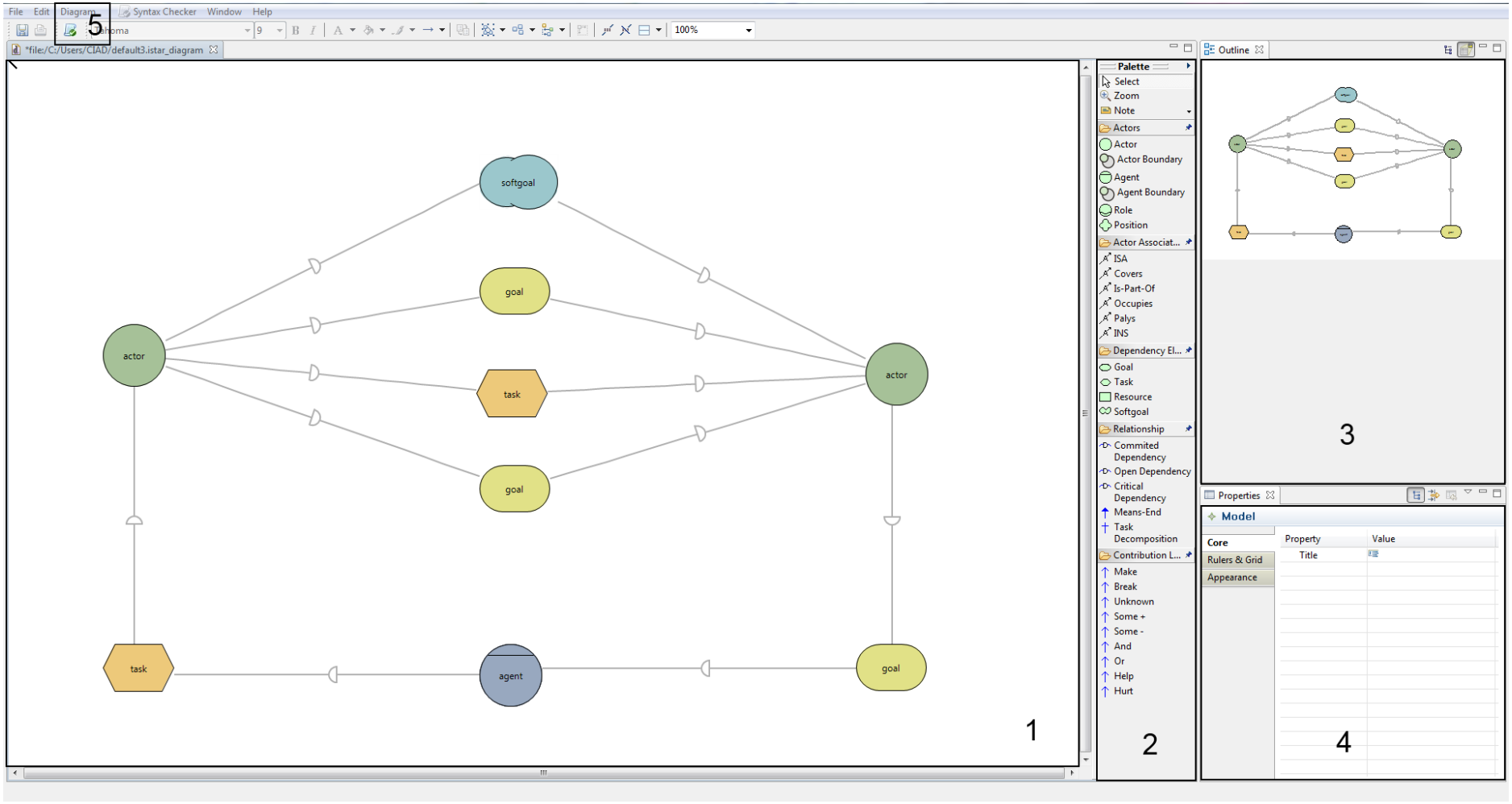



**Figura 11. Imagem do *iStarTool* (MALTA, 2011).**

Entre as diversas possibilidades que a ferramenta oferece, o seu uso já permite trabalhar com ferramentas M2M (*Model to Model*), assim transformando os modelos i* gerados pelo *iStarTool* noutros modelos.

Seguindo a perspectiva de checagens que a ferramenta realiza, Santos (2008a) faz referência a Santos (2008b) quando escreve:

> "A ferramenta *iStarTool* é capaz de realizar a verificação dos erros mais comuns encontrados em modelos i*. Contudo, a verificação de alguns dos erros apresentados em Grau (2008) e Santos (2008) ainda não foi contemplada nesta versão, ficando seu tratamento para trabalhos futuros. A verificação de consistência dos modelos foi definida para ser realizada à medida que os modelos estão sendo criados. Portanto, a ferramenta *iStarTool* só permite a criação de modelos válidos" (Santos, 2008a, p. 94).

O presente trabalho tem então por objetivo, aumentar a cobertura de erros sintáticos que não foram resolvidos por Santos (2008a).

## 3.2. Problemas encontrados e solucionados na ferramenta

O *iStarTool* tendo sido desenvolvido em 2008, por Santos (2008a), usou a versão de *plug-in* GMF disponível na época que, por conseguinte, teve que passar por diversas atualizações. Em decorrência disto, no momento em que foram iniciadas as melhorias do *iStarTool*, tornou-se necessária a correção de algumas funcionalidades que apresentavam falhas. Neste momento avaliou-se que o próprio desenvolvimento do *iStarTool* não era tão simples, dado que a maioria das bibliotecas utilizadas no GMF antigo sequer funcionavam nos computadores mais recentes, que utilizam arquitetura de processador de 64 bits[3].

Outro problema foi a utilização de bibliotecas soltas pelo Sistema Operacional - SO para ativar algumas funcionalidades da ferramenta. Este foi o primeiro passo: encapsular todas as bibliotecas[4] dentro do projeto, procedimento que permitiu que o projeto pudesse ser desenvolvido e passado para vários computadores de forma prática e eficiente.

---

[3] O *iStarTool* foi desenvolvido em computadores de arquitetura de 32 bits.

[4] Bibliotecas ou biblioteca de classes é um conjunto de rotinas que pode ser usado em vários projetos (DEITEL, 2007, p. 134).

No momento em que todas as bibliotecas estavam encapsuladas dentro do projeto, foi dado início à substituição pelas suas versões mais novas. Esta atividade não foi tão simples como se planejou, visto que, ao mesmo tempo em que se substituía alguma biblioteca, notava-se que isto gerava muitos erros no projeto, fato decorrente de chamadas de métodos[5] que não existiam mais nas bibliotecas novas.

Além de problemas de mau funcionamento, a ferramenta teve, durante a substituição de suas bibliotecas, a necessidade de troca em algumas de suas bibliotecas gráficas o que resultou noutros problemas que também foram corrigidos. Um deles foi o posicionamento errado de todos os *Labels*[6] dos elementos do modelo i*.

## 3.3. Geração de *plug-in* para o *iStarTool*

Finalizadas as correções apresentadas na seção anterior foi dado início ao objeto principal deste trabalho, ou seja, a construção dos checadores sintáticos. No GMF as alterações feitas no projeto foram feitas através da construção de *plug-in*. Tais *plug-ins* utilizam como linguagem de programação a linguagem Java. Para tanto existem vários tipos de *plug-ins* que podem ser usados.

Entre os diversos tipos de *plug-ins* existem os que ficam sempre ativos como se fossem inseridos numa nova rotina[7] rodando paralela à rotina principal do programa, sendo trigados no momento em que ocorre alteração no modelo e os que mudam a interface da IDE (*Integrated Development Environments*)[8] para inserir funcionalidade a ela, esta IDE no caso do iStarTool é o Eclipse.

Durante o trabalho foram criados dois *plug-ins* que juntos permitem a validação dos modelos criados pelo *iStarTool*. Para diferencia-los, durante este trabalho, eles serão denominados de *Syntax Checker* e o outro de *Syntax Warnings*, este último tendo sido criado como uma especialização do primeiro.

---

[5] Método descreve os mecanismos que realmente realizam suas tarefas (DEITEL, 2007, p. 58).
[6] Labels é normalmente como são chamados campos de linha única com texto (DEITEL, 2007, p. 330).
[7] Rotina é o mesmo que fluxo de um programa.
[8] IDE, em portugês ambiente de desenvolvimento integrado, oferece muitas ferramentas que suportam o processo de desenvolvimento de software, incluindo editores para escrever e editar programas e depuradores para localizar erros de lógica em programas. (DEITEL, 2007, p. 9)

Para o *Syntax Checker* foi utilizada uma modificação na interface da IDE, que neste caso se trata do Eclipse, para adicionar um botão que serve como ativador de sua funcionalidade. Já o *Syntax Warnings* foi construído utilizando um verificador de mudança sobre o modelo i* trabalhado. Estes dois *plug-ins* terão suas funcionalidades melhor especificadas a seguir.

### 3.3.1. *Syntax Checker*

O *Syntax Checker*, - objeto central deste trabalho, como seu próprio nome indica está vinculado a erros sintáticos e não semânticos, e consegue cobrir todos os erros sintáticos apresentados por Santos (2008b), como pode ser visto na Tabela 2, que foi apresentada no capítulo anterior deste trabalho. Segundo Malta (2011) o *Syntax Checker* está dividido em duas partes: o *Syntax Warnings* e o *Syntax Checker*, ambas apresentadas nas seções a seguir.

#### *3.3.1.1. Syntax Warnings*

O *Syntax Warnings* é um sistema de alertas intrusivo, que tem como objetivo manter o usuário avisado de possíveis erros maiores. Ele funciona alertando o usuário em tempo real sobre a ocorrência de algum erro (MALTA, 2011).

#### *3.3.1.2. Syntax Checker*

O *Syntax Checker*, por outro lado, é bem menos intrusivo. Característica que permite ao mesmo alertar, além dos erros cobertos pelo *Syntax Warnings*, erros que são gerados naturalmente durante a modelagem, como por exemplo, o caso já referenciado por Santos (2008b), quando se refere a um ator sem relacionamentos.

#### *3.3.1.2.* Erros não cobertos pelo *Syntax Checker* rudimentar

O *Syntax Checker*, como já foi dito anteriormente, se encontrava no começo deste trabalho numa versão rudimentar, como pode ser visto na Tabela 3 do Capítulo 2, mais especificamente na subseção 2.3.3. Estas falhas podem ser revistas na Tabela 4 a seguir.

| Principais erros sintáticos (SANTOS, 2008b) | *iStarTool Com Syntax Checker rudimentar* |
|---|---|
| Elementos internos e relacionamentos entre elementos internos | |
| Deixar elementos sem ligações | Permite |
| Elementos internos e relacionamentos entre elementos internos | |
| Deixar elementos sem ligações | Permite |
| Uso de ligações em situações irregulares (ligação de dependência dentro da fronteira do ator) | Permite |
| Ligação direta entre elemento interno de dois atores diferentes | Permite |

**Tabela 4. Simplificação da Tabela 3 para facilitar a leitura das falhas no *Syntax Checker* rudimentar.**

Trabalhar as falhas constantes na primeira versão do *Syntax Checker* do *iStarTool*, ampliando e melhorando a sua capacidade de cobertura, foram atividades desenvolvidas pelo autor durante a realização do presente trabalho. Então uma nova versão do *Syntax Checker* do *iStarTool* foi proposta e submetida a avaliação por uma amostra de usuários, que responderam um questionário que subsidiou a análise de resultados e conclusões apresentadas ao final deste documento. Cumpre, antes disso, apresentar a contribuição deste trabalho, no Capítulo 4, a seguir.

# Capítulo 4 – Implementação de novas funcionalidades e avaliação da ferramenta

Este capítulo é dedicado ao relato dos procedimentos utilizados na evolução da ferramenta, desde o *Syntax Checker* rudimentar até a versão atual, e no levantamento de opiniões de usuários finais em relação ao *Syntax Checker* do *iStarTool*, na versão proposta neste trabalho. Esta versão, incorporou novas verificações sintáticas do modelo às apresentadas no *Syntax Checker* original desenvolvido por Malta (2011).

O trabalho do autor teve seu início no ano de 2011 como trabalho de iniciação científica, sob a orientação do Professor Jaelson Castro (UFPE-CIn). O trabalho em questão partiu de uma listagem de erros de validação do modelo i* identificados por Santos (2008b) e consistiu no desenvolvimento da primeira versão do *Syntax Checker* do *iStarTool*. No entanto, esta versão foi julgada rudimentar e que precisaria ter suas funcionalidades evoluídas. As novas funcionalidades foram incorporadas em 2012 pelo autor, sob a orientação da Professora Carla Taciana Lima Lourenco Silva Schuenemann (UFPE-CIn).

A Tabela 5 apresenta uma análise comparativa entre três versões do iStarTool, considerando a presença ou não do Syntax Checker, bem como versões diferentes do mesmo. Também encontra-se neste capítulo um levantamento de opiniões de usuários a respeito desta nova versão de *Syntax Checker* os quais responderam a um questionário construído especificamente para este propósito, e que consta no Apêndice A do presente trabalho, conforme explicado no plano de experimentação a seguir.

| Principais erros sintáticos (SANTOS, 2008b) | *iStarTool* (sem *Syntax Checker*) | *iStarTool* (com *Syntax Checker rudimentar*) | *iStarTool* (com *Syntax Checker 2012*) |
|---|---|---|---|
| Atores e relacionamento entre atores | | | |
| Atores sem ligação | Permite | Permite, mas avisa | Permite, mas avisa |
| Atores dentro da fronteira de outros atores | Não permite | Não permite | Não permite |
| Ligar Atores com ligação de dependência sem usar um *dependum* | Permite | Permite, mas avisa | Permite, mas avisa |
| Ligar um elemento interno ao Ator direto ao Ator | Permite | Permite, mas avisa | Permite, mas avisa |
| Elementos internos e relacionamentos entre elementos internos | | | |
| Deixar elementos sem ligações | Permite | Permite | Permite, mas avisa |
| Elementos internos e relacionamentos entre elementos internos | | | |
| Deixar elementos sem ligações | Permite | Permite | Permite, mas avisa |
| Uso de ligações em situações irregulares (ligação de dependência dentro da fronteira do ator) | Permite | Permite | Permite, mas avisa |
| Elementos do SR fora da fronteira do ator correspondente | Não permite | Não permite | Não permite |
| Decompor *goals* em *goals* ou *tasks* | Não permite | Não permite | Não permite |
| Contribuição para *goals*, *tasks* e *resources* | Não permite | Não permite | Não permite |
| *Means-ends* onde um *goal* é o meio | Permite | Permite, mas avisa. | Permite, mas avisa. |
| Ligação direta entre elemento interno de dois atores diferentes | Permite | Permite | Permite, mas avisa |

**Tabela 5. Erros sintáticos mais comuns cometidos ao elaborar um modelo i* por Santos (2008b), (sistematização por ferramentas feitas pelo autor).**

Em relação às informações presentes na Tabela 5, o Apêndice B do presente documento apresenta imagens das telas do *Syntax Checker* em funcionamento após as modificações recentemente introduzidas que podem auxiliar na visualização do funcionamento da ferramenta.

## 4.1. Plano de avaliação

Para avaliar a ferramenta desenvolvida, foi organizado o seguinte plano:

a) Apresentação e disponibilização da ferramenta a um conjunto de alunos da turma matriculada na Disciplina Especificação de Requisitos e Validação de Sistemas (IF716) – UFPE/CIn, durante o 1º semestre letivo do ano de 2012;
b) Construção de um questionário (apresentado no Apêndice A) para fins de coleta de opinião dos usuários a respeito da ferramenta apresentada. Os usuários em questão responderam o questionário após o período de duas semanas de uso da ferramenta. Para a construção deste questionário, foram levadas em consideração as abordagens presentes na literatura especializada, destacando, entre outros autores: Rodrigues (2007), Thiollent (1997), Laville (1999), Goldenberg (1997), Oliveira (2005) e Appolinário (2004). Este último define questionário como:

> "I. Técnica estruturada para coleta de dados; II. Tipo de instrumento de pesquisa que consiste num conjunto de perguntas escritas que devem ser respondidas pelos sujeitos. Para Malhotra (2001), "o principal ponto fraco da elaboração de um questionário é a ausência de teoria. Como não existem princípios científicos que garantem um questionário ótimo ou ideal, sua concepção é uma habilidade que se adquire com experiência"(p.254). Outros autores têm ressaltado a importância da ordem das questões e como sua formulação pode induzir padrões de resposta que se caracterizam como um viés no questionário" (p. ex.: SCHWARZ, 1999; SHWARZ; SUDMAN, 1995). (APPOLINÁRIO, 2004, p. 168).

Para Laville (1999) os questionários são utilizados para saber a opinião de uma população, representada por uma amostra, à qual se submete uma série de perguntas sobre o tema visado, sendo tais perguntas escolhidas em função de hipóteses. Para cada uma dessas perguntas, oferece-se ao interrogado uma opção de respostas, definida a partir de indicadores, pedindo-lhes que assinalem a que corresponde à sua opinião.

Rodrigues (2007) chama atenção para a importância da brevidade das indagações, isenção, clareza e organização que devem permear a formatação desse tipo de instrumento de coleta de informações na pesquisa. Para

Thiollent (1997), tais aspectos, quando bem observados, permitirão a montagem de questionários "sob medida" para o atendimento dos objetivos finais de uma pesquisa.

Mirian Goldenberg (1997) ressaltava vantagens dos questionários entre os instrumentos de coleta de informações utilizados nas pesquisas acadêmicas, dentre as quais o menor dispêndio, maior facilidade de aplicação, possibilidade de ser aplicado a um grande número de pessoas ao mesmo tempo, além do aspecto de que os pesquisados se sentem mais livres para exprimir suas opiniões, dentre outras vantagens.

Para a definição dos indicadores que estarão associados às possíveis respostas, o pesquisador deve se valer de escalas, as quais, segundo Amaro et al. (2005), podem ser de quatro tipos: Escala de Likert, VAS (Visual Analogue Scales), Escala Numérica e Escala Guttman. A respeito delas os autores comentam:

- ✓ A **Escala de Likert** apresenta uma série de cinco proposições, das quais o inquirido deve selecionar uma, podendo estas ser: concorda totalmente, concorda, sem opinião, discorda, discorda totalmente. As respostas são cotadas (+2, +1, 0, -1, -2) ou pontuadas (1 a 5);
- ✓ **VAS** (*Visual Analogue Scales*) é um tipo de escala que advém da escala de Likert apresentando os mesmos objetivos, entretanto um formato diferente. Tal escala se baseia numa linha horizontal com 10 cm de comprimento apresentando nas extremidades duas proposições contrárias (por exemplo: útil – inútil);
- ✓ A **Escala Numérica** deriva da escala anterior na qual a linha se apresenta dividida em intervalos regulares; e,
- ✓ A **Escala de Guttman** apresenta um conjunto de respostas hierarquizadas, implicando no fato de que, quando o inquirido concordar com uma das opções, o mesmo estará concordando com todas as que se encontram numa posição inferior na escala (AMARO et al., 2005).

Dados os objetivos da presente pesquisa, estruturaram-se os questionários cujas opções de respostas tiveram por base a Escala de Likert, por ser aquela

que metodologicamente melhor atenderia aos propósitos finais deste estudo. Tal escala é amplamente usada em pesquisas de opinião por ser de simples interpretação, fácil de ser respondida e analisada.

Desta maneira, as respostas das questões presentes no questionário serão dadas em termos de conceitos/notas, variando de “definitivamente sim” a “definitivamente não”, onde: 1. Definitivamente sim; 2. Provavelmente; 3. Indeciso; 4. Provavelmente não; e, 5. Definitivamente não.

c) Recepção, tabulação e análise dos dados coletados.

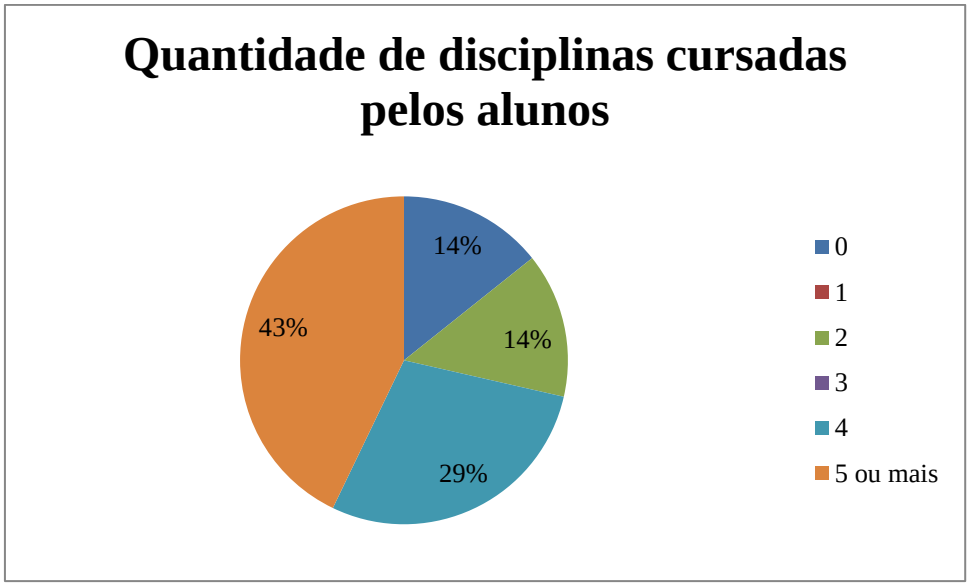
O questionário utilizado nesta pesquisa foi aplicado por meio de formulários web e foi estruturado em sete perguntas que objetivaram levantar, dentre outros aspectos, o grau de interesse dos componentes da amostra na área de Engenharia de Requisitos, como avaliam a ferramenta *iStarTool* e a ajuda prestada pelos *popups* nela introduzidos pelo *Syntax Checker*. Além disto, a facilidade de uso da ferramenta, a complexidade do seu processo de instalação e o grau de satisfação por parte dos usuários foram outros aspectos avaliados pelo questionário aplicado.

Os formulários distribuídos dispunham ainda de um espaço reservado à emissão de outras impressões dos usuários a respeito da ferramenta em estudo.

## 4.2. Resultado da avaliação

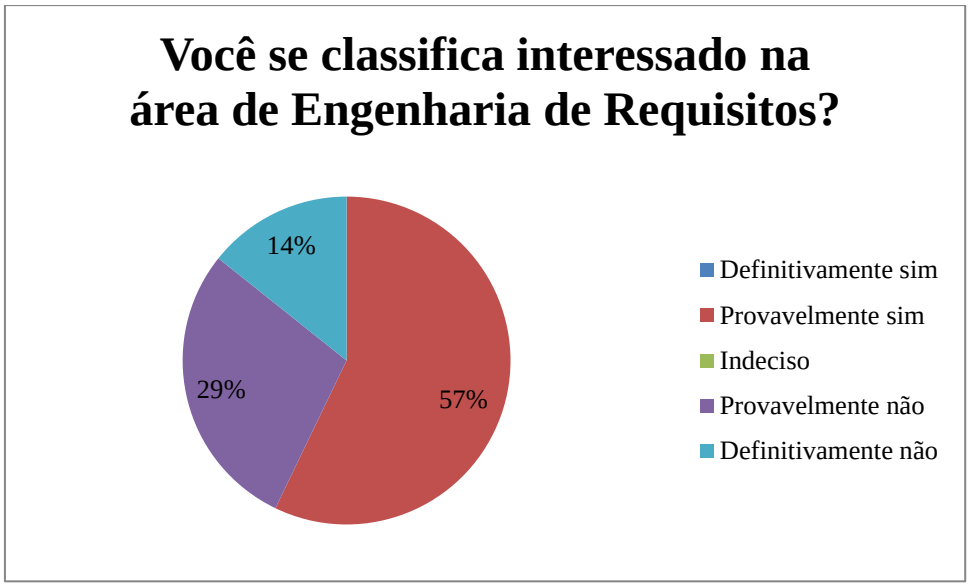
Nessa seção será apresentado o resultado da avaliação, seguido de algumas assertivas construídas pelo autor.

- Primeira Pergunta:
  - Nesta primeira pergunta, o objetivo foi descobrir quantas disciplinas estavam sendo cursadas durante o período em que o aluno usou a ferramenta avaliada. Este ponto foi levantado pelo autor como forma de conhecer um pouco sobre as pessoas que estavam avaliando a ferramenta.



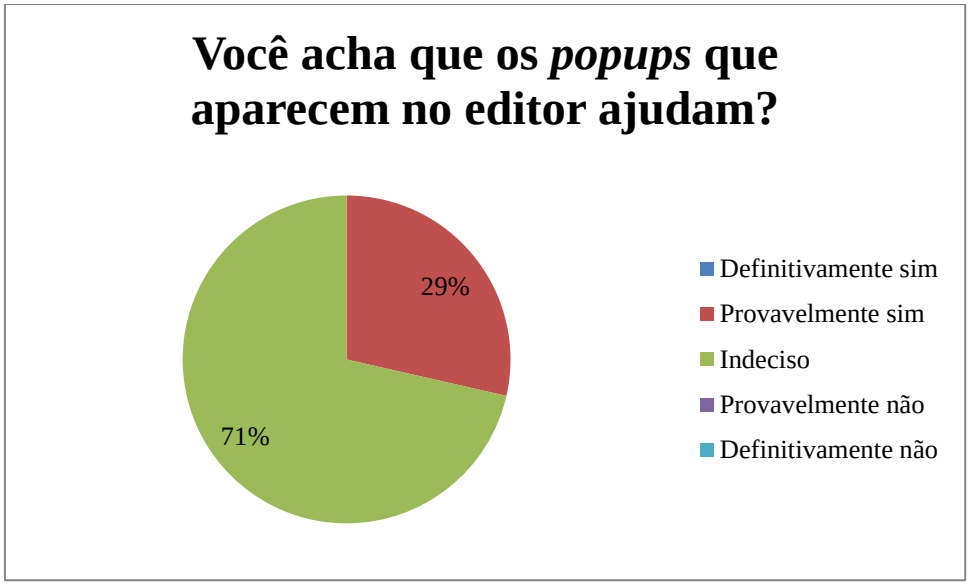

Com esta informação pode-se entender que quase 75% dos alunos tinham pelo menos um turno de aulas todos os dias.

- Segunda Pergunta:
  - Esta pergunta objetivou descobrir o interesse dos alunos pela área específica de Engenharia de Requisitos. Assim como a primeira, esta informação foi importante para conhecer um pouco melhor sobre a população em questão.



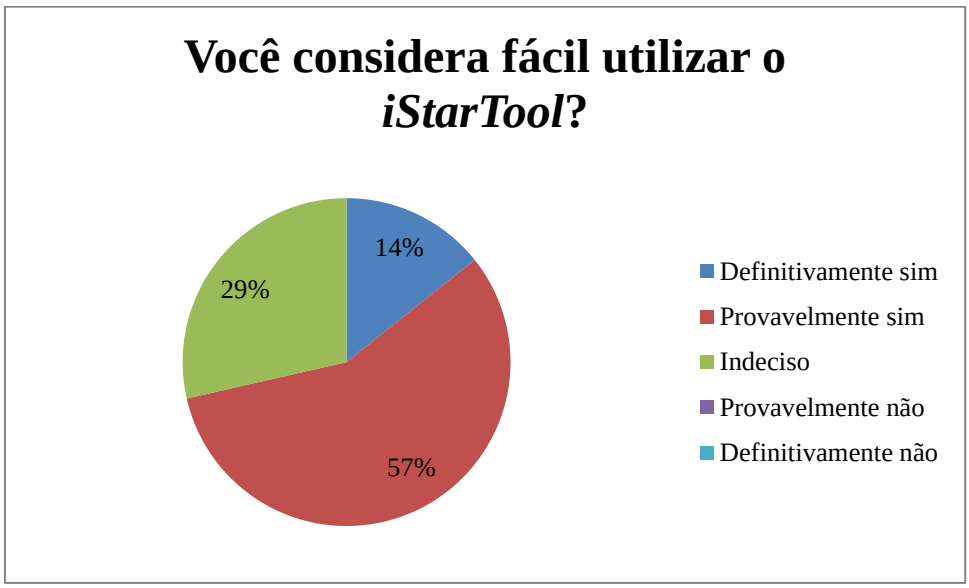

Com esta informação pode-se inferir que quase metade dos alunos que cursaram a disciplina de Engenharia de Requisitos não apresentava muito interesse pela área, mesmo se levando em conta que 57% dos entrevistados chegaram a admitir que “provavelmente” têm interesse.

- Terceira Pergunta
  - A terceira pergunta faz referência ao sistema intrusivo de ajuda ao usuário, chamado anteriormente de *Syntax Warning*, e que se manifesta por meio de *PopUps*.



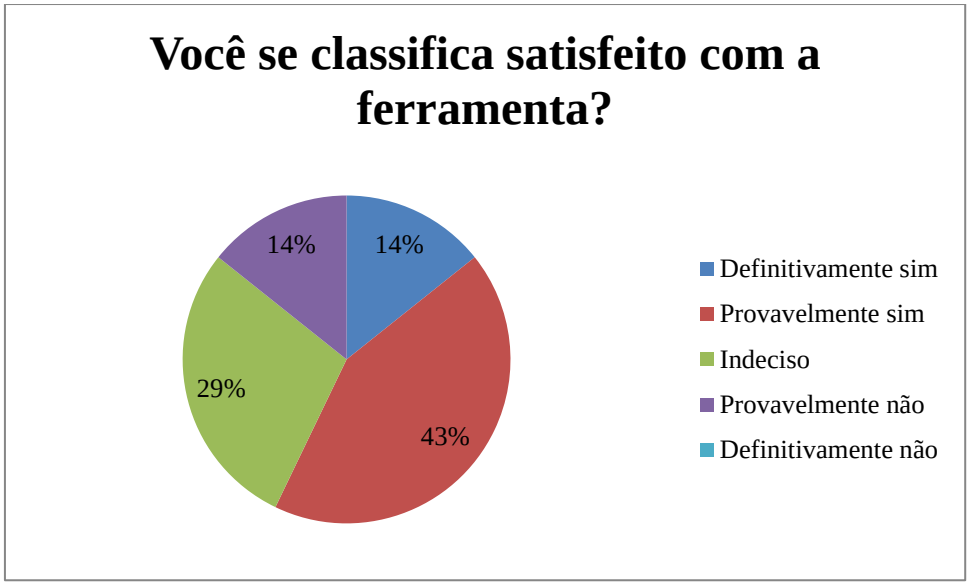

Embora seja um sistema intrusivo, o mesmo alcançou uma boa avaliação, ou pelo menos não apresentou uma má avaliação dos usuários. Entretanto nenhum dos alunos que respondeu ao questionário se sentiu realmente ajudado pelos *popups*.

- Quarta Pergunta
  - Nesta pergunta foi avaliado o grau de satisfação quanto à usabilidade da ferramenta.



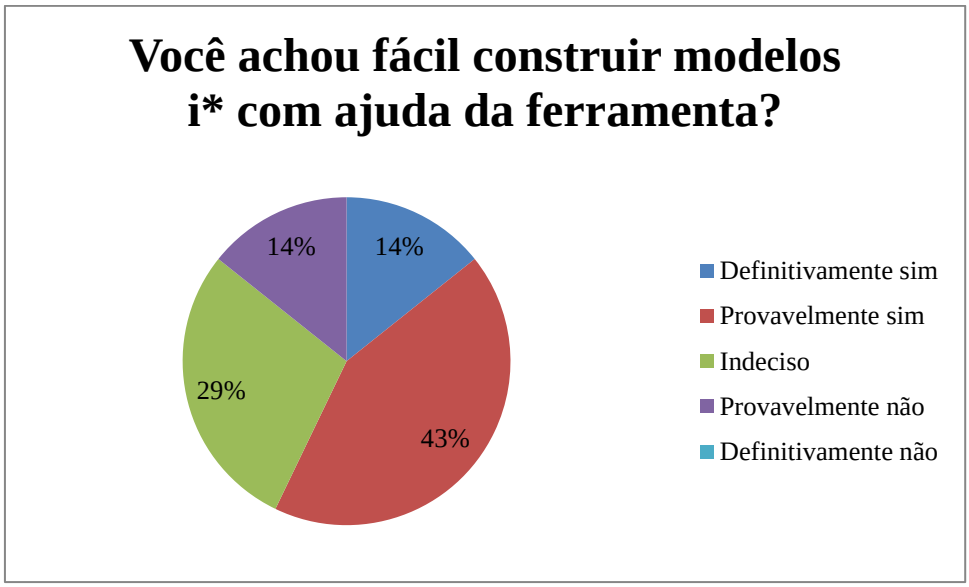
Novamente as respostas foram positivas para as perguntas.

- Quinta Pergunta
  - Esta pergunta está mais preocupada em descobrir como o usuário se sente em relação à ferramenta.



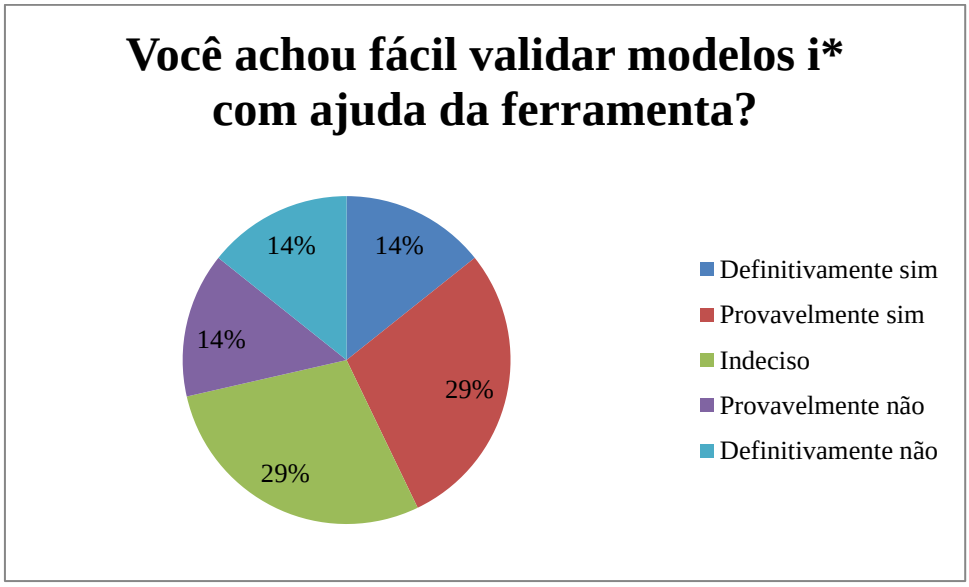

A maioria das respostas foi positiva, entretanto não foi unânime.

- Sexta Pergunta
  - A sexta pergunta tinha o intuito de avaliar quão boa a ferramenta era para cumprir seu propósito específico.




As respostas mostraram algo preocupante, a ferramenta tem apenas pouco mais de 50% de aprovação.

- Sétima Pergunta
  - A sétima pergunta tinha como objetivo avaliar os checadores sintáticos de modelo perante usuários finais.




O resultado foi bom, visto que apenas um pouco mais de 25% acharam que a ferramenta não colaborou.

- Campo aberto
  - Ninguém respondeu nada no campo aberto.

O resultado do questionário foi, em geral, bom para este trabalho. Entretanto, de todos os alunos que receberam o questionário para responder, apenas sete responderam. Portanto não é uma amostra grande o suficiente para eliminar mínimos ou máximos locais.

# Capítulo 5 – Conclusão

Neste último capítulo são apresentadas algumas considerações finais sobre a realização do presente trabalho, as contribuições que o mesmo proporcionou, quer para a formação acadêmica do autor, para o estado da arte de ferramentas CASE ou ainda do ponto de vista de elementos motivacionais para estudos futuros. Os aspectos limitantes enfrentados durante a sua realização também são comentados neste capítulo de conclusão.

Após esta conclusão, são apresentadas as referências bibliográficas que serviram de base para as discussões presentes neste estudo e, finalmente, a título de apêndice, o Questionário desenvolvido pelo autor e aplicado para coleta de opiniões sobre a ferramenta desenvolvida, bem como as telas que ilustram o funcionamento do *syntax Checker* da *iStarTool*.

## 5.1. Considerações Iniciais

A título de considerações iniciais, cumpre destacar que a oportunidade de realizar este trabalho foi muito importante do ponto de vista de formação acadêmica, pelo exercício realizado a título de método de pesquisa e desenvolvimento de trabalho científico.

Durante a realização do mesmo, foi possível vivenciar um pouco dos processos com os quais o profissional se envolve no planejamento de pesquisa, identificação de problemas, revisão bibliográfica, metodologia de pesquisa, desenvolvimento de editores gráficos, aplicação de instrumento de coleta de opiniões, análise e interpretação de resultados e construção de conclusões.

## 5.2. Contribuições

A principal contribuição deste trabalho consiste na construção de uma ferramenta de suporte ao aprendizado e a modelagem de diagramas i*, em especial a funcionalidade de checagem sintática de modelos. Outras contribuições geradas durante este trabalho foram:

- Levantamento dos principais erros que ocorrem durante a construção de modelos i* e sua separação entre semântico e sintático. Para tanto foi utilizado o material construído por Santos (2008b);
- Comparação das ferramentas CASE, que dão suporte à modelagem i*; e,
- Elaboração e aplicação de questionário para avaliar o iStarTool, principalmente as funcionalidades relacionadas ao Syntax Checker. O processo de avaliação foi acompanhado de apresentação em sala de aula da ferramenta desenvolvida para os alunos da disciplina Especificação de Requisitos e Validação de Sistemas do Cin/UFPE.

## 5.3. Limitações

Dentre as limitações enfrentadas na construção do presente trabalho a principal que cumpre destacar diz respeito à complexidade do trato de problemas de natureza semântica, conforme já explicitado na penúltima seção do capítulo 2. Tal propriedade requereria um maior tempo para ser adequadamente abordada e implementada no sistema.

No que concerne ao questionário construído e aplicado junto a usuários, a limitação identificada se referiu à pequena quantidade de pessoas que o responderam, fato que impede concluir uma análise mais apurada da ferramenta. Mesmo assim, cabe destacar o fato das respostas terem sido, em sua maioria, positivas, o que permitiria, de certa maneira, avaliar que as atualizações da ferramenta vêm alcançando o seu objetivo enquanto melhoria da mesma.

## 5.4. Trabalhos Futuros

Considerando que não foram abordados erros semânticos, a inserção de correção semântica se constitui numa primeira indicação para novos trabalhos voltados à ferramenta. Outros possíveis trabalhos futuros são:

- Melhoria da interface do iStartool, visto que esta ainda apresenta falhas que podem gerar erros na interpretação dos modelos;
- Inserção de uma funcionalidade que transforme modelos SR em SD e SD em SR, sem que para isso seja necessária a criação de modelos separados; e,

- Construção de tutoriais interativos que ajudem no aprendizado e desenvolvimento de modelos i*.

Diante do exposto, é possível concluir que ainda há trabalhos que podem ser realizados para que a ferramenta alcance uma maior maturidade e, nesta direção, avalia-se que no presente trabalho existem contribuições que poderão ser levadas em conta na construção de novas versões da ferramenta, notadamente: o formato do questionário desenvolvido; os resultados que o mesmo permitiu apurar, mesmo a despeito do pequeno número de pessoas que o responderam; e o próprio verificador de erros (*Syntax Checker*).

## Referências

AMARO, Ana; PÓVOA, Andreia e MACEDO, Lúcia. A arte de fazer questionários. Faculdade de Ciências da Universidade do Porto, 2005. Disponível em: http://nautilus.fis.uc.pt/cec/esjf/wp-content/uploads/2009/11/elab_quest_outros.doc.. Acesso em 08/04/2012.

APPOLINÁRIO, Fabio. Dicionário de metodologia científica: um guia para a produção do conhecimento científico. São Paulo: Atlas, 2004.

BPMN. **Business Process Modeling Notation.** Version 2.0. January, 2011. Disponível em: http://www.omg.org/spec/BPMN/2.0, acesso em 20/05/2012.

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 em 10/02/2012.

FIRESMITH, Donald. **Common Requirements Problems, TheirNegative Consequences, and the Industry Best Practices to Help SolveThem**. Software Engineering Institute, U.S.A. Disponível em: http://www.jot.fm/issues/issue_2007_01/column2.pdf. Acesso em 12/02/2012.

FREITAS, Henrique; OLIVEIRA, Mírian; ZANELA, Amarolinda Costa e MOSCAROLA, Jean. **Método de pesquisa survey**. Disponível em: http://sphinxbrasil.net.br/revista/wp-content/blogs.dir/7/files/2011/12/AR-GT-A-metodo-pesquisa-survey.pdf, Acesso em: 22/03/2012.

GOLDENBERG, Mirian. A arte de pesquisar: como fazer pesquisa qualitativa em Ciência Sociais. Rio de Janeiro: Record, 1997.

i*WIKI. **Available i* Tools**. Disponível em: http://istar.rwth-aachen.de/tiki-index.php?page=i*+Tools. Acesso em: 20/02/2012.

LAVILLE, Christian e DIONNE, Jean. A construção do saber: manual de metodologia da pesquisa em ciência humanas/ Tradução Heloísa Moteiro e Francisco Settineri. Porto Alegre: Artmed, UFMG, 1999.

MALTA, Átila; SOARES, Monique; SANTOS, Emanuel;PAES, Josias e CASTRO, Jaelson. **iStarTool: Modeling requirements using the i* framework**. Universidade Federal de Pernambuco – UFPE, Centro de Informática, Recife, Brazil.

OLIVEIRA, Maria Marly de, 1942. Como fazer pesquisa qualitativa. Recife: Bagaço, 2005.

RODRIGUES, Rui Martinho. Pesquisa Acadêmica: Como facilitar o processo de preparação de suas etapas. São Paulo: Atlas, 2007.

ROJAS, Ricardo Arturo Osorio (2001), *El Cuestionario*; [online] [consult 2004-11-22]; Disponível em http://www.nodo50.org/sindpitagoras/Likert.htm. Acesso em: 08/04/2012.

SANTOS, Bárbara Siqueira. IStarTool – **A proposal of tool for modeling i***. (in portuguese, Uma proposta de ferramenta para modelagem de i*). Master's thesis - Universidade Federal de Pernambuco, Centro de Informática, Brasil, 2008a. Disponível em:

http://portal.cin.ufpe.br/ler/Our%20Publications/Dissertations/BarbaraSantos2008.pdf. Acesso em 12/02/2012.

SANTOS, Emanuel Batista dos. **Uma proposta de métricas para avaliar modelos i*** / Emanuel Batista dos Santos. – Recife: O Autor, 2008. xi, 117 folhas : il., fig., tab. Dissertação (mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2008b.

SILVA JUNIOR, Josias Paes da. **AGILE: uma abordagem para geração automática de linguagens i*** / Josias Paes da Silva Junior - Recife: O Autor, 2011. xv, 131 folhas: il., fig., tab. Dissertação (mestrado) - Universidade Federal de Pernambuco. CIn, Ciência da Computação, 2011.

SOMMERVILLE, Ian. **Engenharia de software**, 8ª edição/ Ian Sommerville; tradução:selma Shin Shimizu Melnikoff, Reginaldo Arakaki, Edílson de Andrade Barbosa; revisão técnica: Keichi Kirama – 8ª ed. – São Paulo: Pearson Addison-Wesley, 2007.

THIOLLENT, Michel, 1947. Pesquisa-ação nas organizações. São Paulo: Atlas, 1997.

YU, E. **Modelling Strategic Relationships for Process Reengineering**. Tese (Doutorado) - University of Toronto, Department of Computer Science, Canadá, 1995.

GMF - Graphical Modelling Framework. Web-site. Disponível em: < http://www.eclipse. org/modeling/ gmp/>. Último acesso em: 15 de Abril de 2012.

DEITEL, H.M. **Java: como programar**; revisão técnica: Fábio Lucchini – 6ª ed. – São Paulo: Pearson Prentice Hall, 2007.

FERREIRA, Aurélio Buarque de Holanda. **Dicionário Aurélio Básico**. 1ª Edição - J.E.M.M. Editora 1988.

# Apêndice A. Questionário aplicado junto aos Usuários do IStarTool

Prezado(a) Usuário(a),

Pergunta 1: Antes de tudo gostaríamos de saber quantas disciplinas você cursou (ou está cursando) no mesmo semestre em que esteve (ou está) matriculado em Engenharia de Requisitos.

Alternativas: 0, 1 , 2 , 3 , 4 , 5 ou mais

Para as próximas questões considere as seguintes alternativas:

1. Definitivamente sim
2. Provavelmente sim
3. Indeciso
4. Provavelmente não
5. Definitivamente não

Pergunta 2: Você se classifica interessado na área de Engenharia de Requisitos?

Pergunta 3: Você classifica a ferramenta iStarTool boa para o aprendizado?

Pergunta 4: Você acha que os popups que aparecem no editor ajudam?

Pergunta 5: Você considera fácil utilizar o iStarTool?

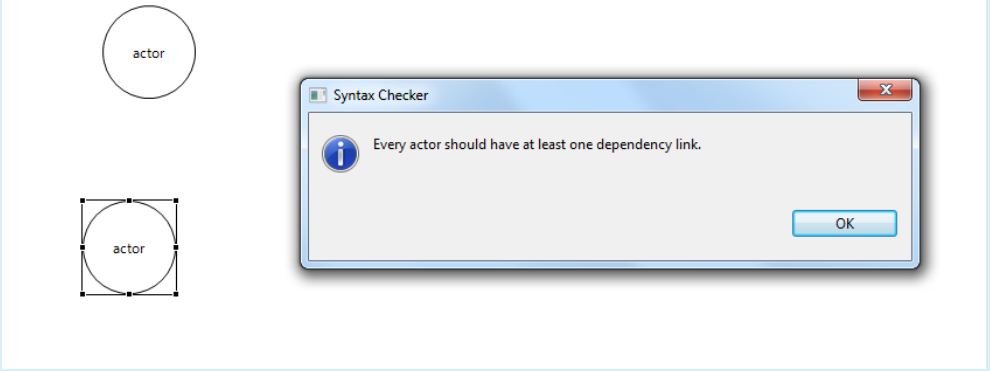
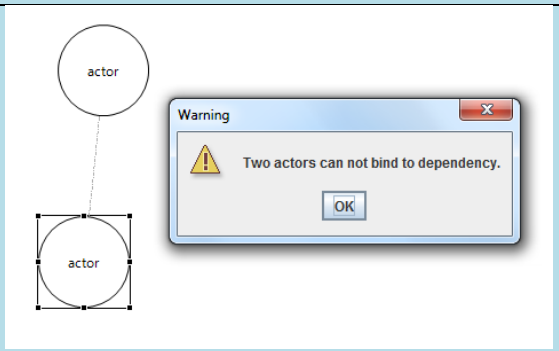
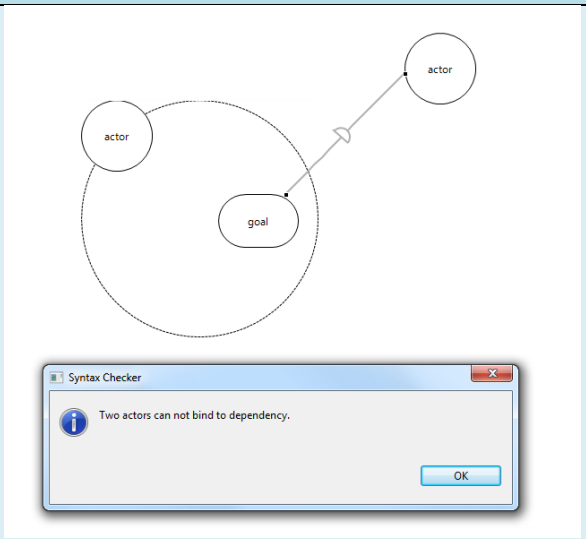
Pergunta 6: Você se classifica satisfeito com a ferramenta?

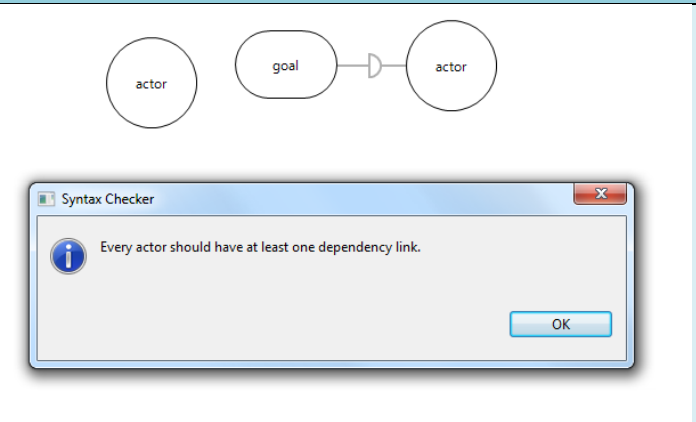
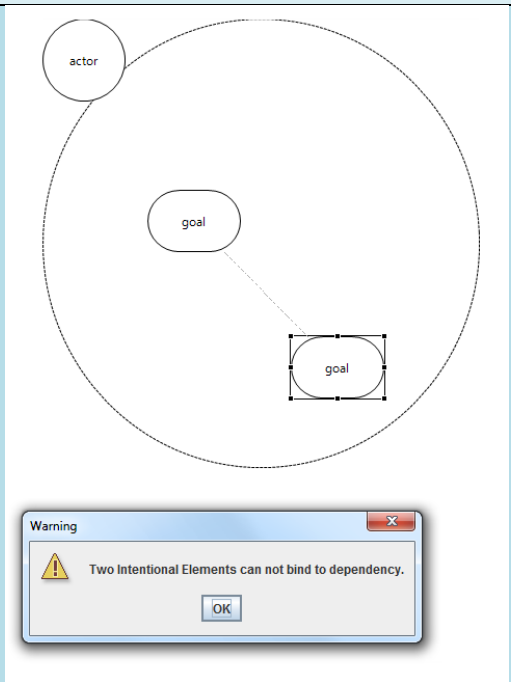
Pergunta 7: Você achou fácil construir modelos i* com ajuda da ferramenta?

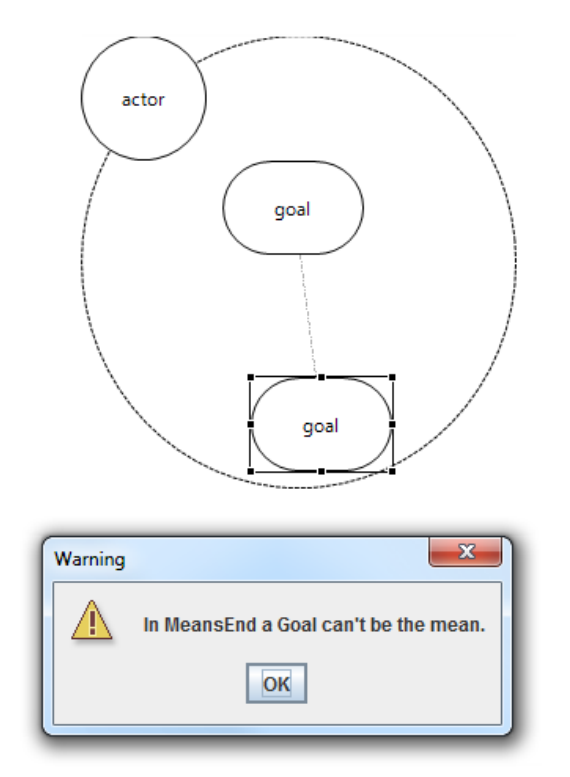
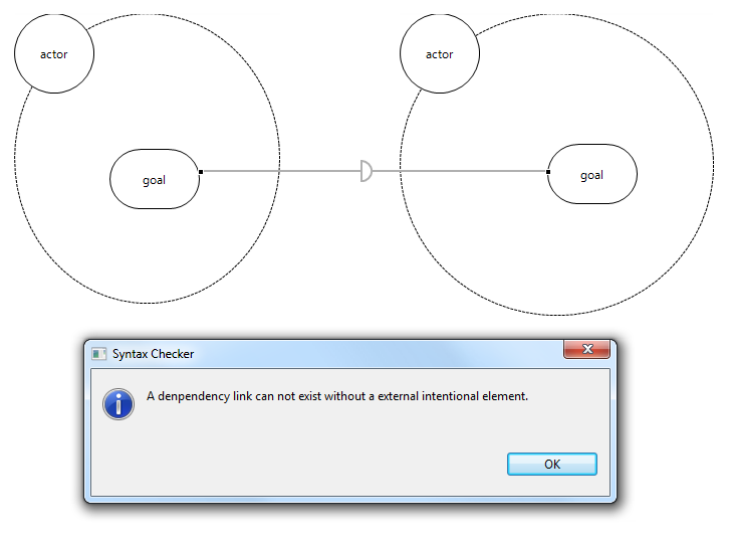
Pergunta 8: Você achou fácil validar modelos i* com ajuda da ferramenta?

Caso deseje registrar alguma contribuição que não tenha sido abordada nas perguntas acima formuladas, utilize o espaço a seguir.

# Apêndice B. Telas do *Syntax Checker* em funcionamento.

| Erros que ocorrem em modelagem de diagramas i* | | |
|---|---|---|
| Atores e relacionamento entre atores | | |
| 1 | Atores sem ligação |  |
| 2 | Atores dentro da fronteira de outros atores | Não Permite |
| 3 | Ligar Atores com ligação de dependência sem usar um *dependum* |  |
| 4 | Ligar um elemento interno ao Ator direto ao Ator |  |

| Elementos internos e relacionamentos entre elementos internos | | |
|---|---|---|
| 1 | Deixar elementos sem ligações |  |
| 2 | Uso de ligações em situações irregulares (ligação de dependência dentro da fronteira do ator) |  |
| 3 | Elementos do SR fora da fronteira do ator correspondente | Não Permite |
| 4 | Decompor *goals* em *goals* ou *tasks* | Não Permite |
| 5 | Contribuição para *goals*, *tasks* e *resources* | Não Permite |

| | | | |
|---|---|---|---|
| 6 | *Means-ends* onde um *goal* é o meio | |  |
| 7 | Ligação direta entre elemento interno de dois atores diferentes | |  |