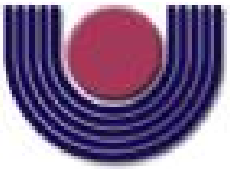


**Unioeste - Universidade Estadual do Oeste do Paraná**
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
***Curso de Bacharelado em Ciência da Computação***

# E4J: Editor *i** para JGOOSE

*Leonardo Pereira Merlin*

**CASCAVEL**
**2013**

**Leonardo Pereira Merlin**

# E4J: Editor *i** para JGOOSE

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Victor Francisco Araya Santander

CASCAVEL
2013

**Leonardo Pereira Merlin**

# E4J: Editor *i** para JGOOSE

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Victor Francisco Araya Santander
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Ivonei Freitas da Silva
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Elder Elisandro Schemberger
Colegiado de Ciência da Computação,
UNIOESTE

Cascavel, 28 de maio de 2013

*Oscar José Merlin Júnior, dedico este trabalho a você, meu irmão e minha referência.*

*"Se você pensa que pode ou se pensa que não pode, de qualquer forma você está certo."*
*Henry Ford*

# AGRADECIMENTOS

Em primeiro lugar, eu gostaria de agradecer ao meu orientador, professor Victor Francisco Araya Santander, pelos conselhos, além do apoio, disponibilidade e interesse no meu trabalho. Apresentou-me oportunidades que eu jamais acreditava que teria novamente. E, além de acreditar nos resultados, sempre me inspirou coragem e ânimo para cumprir meus objetivos.

Também gostaria de agradecer aos outros membros da banca examinadora, os professores Ivonei Freitas da Silva e Elder Elisandro Schemberger. Obrigado por analisar e avaliar cuidadosamente o meu trabalho. Durante as apresentações, contribuiram significativamente com questionamentos e sugestões, de fato, relevantes. E não distante desses, meus sinceros agradecimentos a todos os acadêmicos integrantes do grupo de estudo do Laboratório de Engenharia de Software (LES). Dentre esses, um agradecimento especial aos companheiros Diego Peliser, Leonardo Zanotto Baggio e Rodrigo Trage. Obrigado pelas apresentações e discussões sobre temas de destaque na área de Engenharia de Software. Espero poder retribuir e contribuir para com o grupo.

Quanto aos colegas de moradia, Lucas Inácio, Bruno Belorte, Marcos Schmitt, Nicolas Zaro e Julian Ruiz Diaz, obrigado pela tolerância e bons momentos.

Não posso deixar de agradecer os amigos Fernando Dal Bello, Amadeu Paixão e Felipe Carminati pelas experiências profissionais, oportunidades de um trabalho em equipe bem realizado e um amadurecimento pessoal memorável (nada que uma boa trilha sonora não resolva). Em especial, agradeço ao Carlos Henrique de França, que me ajudou a esclarecer e aguçar minhas pesquisas, além das críticas em numerosas versões de documentos técnicos.

A minha família, pelos conselhos (não ignorados), pelo auxílio e suporte de diversas maneiras. E, Lah (Larissa Torquato de Oliveira e família), este lugar também é de vocês.

A minha companheira, Jamile Merlin, por participar de mais esta fase da minha vida.

À todos, meu sincero agradecimento!

# Lista de Figuras

# Lista de Abreviaturas e Siglas

| | |
|---|---|
| API | *Application Programming Interface* |
| CASE | *Computer-Aided Software Engineering* |
| DFD | Diagrama de Fluxo de Dados |
| EMF | *Eclipse Modelling Framework* |
| ER | Engenharia de Requisitos |
| ES | Engenharia de Software |
| GOOSE | *Goal into Object Oriented Standard Extension* |
| GUI | *Graphical User Interface* |
| IDE | *Integrated Development Environment* |
| ITU | *International Telecommunication Union* |
| JGOOSE | *Java Goal into Object Oriented Standard Extension* |
| OME | *Organization Modelling Environment* |
| ORM | *Object-Relational Mapping* |
| POM | *Project Object Model* |
| POO | Paradigma Orientado a Objetos |
| SD | Modelo de Dependências Estratégicas |
| SR | Modelo de Razões Estratégicas |
| UML | *Unified Modeling Language* |
| XMI | *Xml Metadata Interchange* |
| XML | *eXtensible Markup Language* |

# Sumário

# Resumo

Em discussões pertinentes à engenharia de requisitos, tanto em âmbito acadêmico quanto industrial, ressalta-se que um dos principais desafios da área consiste na integração de modelos organizacionais às demais etapas do processo de engenharia de requisitos.

Alguns trabalhos relacionados já apresentaram técnicas para a realização dessa integração. Nesse sentido, em 2006 foi desenvolvida uma solução computacional denominada JGOOSE (do inglês Java Goal Into Object Oriented Standard Extension), uma ferramenta de auxílio a engenharia de requisitos que proporciona a automatização do mapeamento de modelos e diagramas do framework i* para casos de uso UML.

Apesar das atualizações e melhorias realizada por outros pesquisadores e membros do grupo do Laboratório de Engenharia de Software (LES) da Universidade Estadual do Oeste do Paraná (UNIOESTE - Campus Cascavel/PR), a ferramenta ainda possui uma dependência crítica em relação à elaboração dos dados de entrada. Nas condições atuais é necessário instalar alguma ferramenta específica para produzir o arquivo no formato TELOS.

Desta forma, este trabalho consiste no projeto e desenvolvimento de um editor gráfico de modelos organizacionais i* integrado à ferramenta JGOOSE, visando melhorar suas funcionalidades e diminuir a necessidade de outros softwares para este fim. Uma característica importante desse novo editor é o suporte à especificação iStarML - um formato de arquivo baseado em XML para representação e intercâmbio de modelos i*.

**Palavras-chave:** E4J, JGOOSE, iStarML, Modelagem Organizacional, Engenharia de Requisitos.

# Capítulo 1

# Introdução

Este primeiro capítulo tem como objetivo a apresentação geral do trabalho. É realizada a contextualização e delimitação da pesquisa ao escopo da Engenharia de Software, bem como são destacados os principais objetivos da proposta. Apresenta-se inicialmente, na seção 1.1, o contexto sobre as ferramentas de modelagem organizacional e suas contribuições na área da Engenharia de Requisitos, destacando a influência dessas ferramentas no desenvolvimento de produtos de qualidade. Na seção 1.2, são apresentadas as principais influências e motivações para a realização do trabalho. Em seguida, na seção 1.3, é apresentada a proposta sob uma visão geral e os objetivos norteadores da pesquisa. Na seção 1.4, descreve-se as contribuições esperadas após a finalização deste trabalho. Por fim, na seção 1.5, é apresentada a estrutura geral e a organização do restante desta monografia.

## 1.1 Contexto

Muitas são as opções de ferramentas e técnincas que visam auxiliar engenheiros de requisitos no processo de construção de modelos organizacionais i* [2] [5]. Podendo ser classificadas como ferramentas CASE (*Computer-Aided Software Engineer*) [1] [6], essas ferramentas têm como objetivo o aumento da produtividade e a melhoria da qualidade final dos softwares, através da automatização e gerenciamento de várias fases da Engenharia de Software.

A área de Engenharia de Requisitos (ER), subárea da Engenharia de Software, é responsável por diversas atividades que abrangem os processos de análise, elicitação, especificação, avaliação, ajuste, documentação e evolução dos requisitos de um sistema computacional. É uma

[1] Ferramentas CASE, é toda e qualquer ferramenta baseada em computador que auxilie nas atividades de desenvolvimento de software.

das áreas mais críticas para o sucesso e qualidade de um projeto de software [7]. Pesquisas pertinentes à ER, tanto em âmbito acadêmico quanto industrial, apontam a falta de um entendimento adequado da organização por parte dos responsáveis pela elaboração do documento de requisitos como sendo uma das principais falhas no processo de especificação dos requisitos [8].

Para tentar diminuir os problemas relacionados as fases iniciais do projeto, pesquisas recentes mostram que a comunidade tem buscado estabelecer e utilizar padrões de técnicas, métodos e ferramentas para tratar especificamente da fase inicial de desenvolvimento de software [9]. Pensando nisso, têm-se investido esforços no processo de modelagem organizacional. Este tipo de modelagem visa prover recursos que permitam modelar as intenções, relacionamentos e motivações entre membros de uma organização [10]. Dentre as técnicas de modelagem organizacional, destaca-se a i*, proposto por [11], uma técnica que utiliza a orientação a agentes [12] com enfoque tanto nos desejos e intenções desses agentes, quanto suas dependências [2] [13]. Mais detalhes sobre esta técnica e suas variações serão discutidos no capítulo 2.

Pensando em auxiliar no processo de desenvolvimento de software, alguns trabalhos foram propostos com o intuito de realizar o mapeamento de modelos do *framework* i* para diagramas da UML (do inglês *Unified Modeling Language*). Dentre esses trabalhos, destaca-se o trabalho de Santander [14], que propõe a derivação em casos de uso UML a partir de modelos do *framework* i*. Para apoiar esse processo de derivação, foi desenvolvida a ferramenta JGOOSE (Java Goal into Object Oriented Standard Extension) [15], uma ferramenta que mapeia de forma automática os diagramas i* para casos de uso UML. Essa ferramenta tem como base as diretrizes e passos propostos por Santander [14],

Inicialmente apresentada como GOOSE (*Goal into Object Oriented Standard Extension*) em [16] e [17], em seguida, melhorada e apresentada como JGOOSE por [18], passou também por melhorias com [19] e, atualmente, está sendo aprimorada por [20].

Ou seja, dado como entrada os modelos i*, no formato de arquivo TELOS [21] [22], a ferramenta consegue gerar conforme o template proposto em [23] os casos de uso UML com um bom nível de detalhamento.

Porém, a ferramenta JGOOSE ainda não possui funcionalidades para a produção dos arquivos de entrada da ferramenta. Ainda existe a dependência da ferramenta OME ou, mais

especificamente, ao formato de arquivo TELOS. Nesse contexto, percebe-se a necessidade de se desenvolver um editor de modelos i* integrado à ferramenta JGOOSE, bem como implementar o suporte à especificação do formato de arquivo iStarML [24], uma formato em XML para representação de modelos i* com o propósito de servir como um intercâmbio entre os outros meta-modelos existentes [25].

## 1.2 Motivação

A área de Engenharia de Requisitos está em crescimento e destaque por impactar de forma tão significativa nos resultados finais de um projeto de software. Desta forma, é valido o investimento de esforços para a melhoria de métodos, técnicas ou ferramentas que auxiliem os profissionais da área a aprimorar seu trabalho de forma eficiente. O i* é a base da GRL (*Goal-oriented Requirements Language* ou Linguagem de Requisitos Orientada a Objetivos), que junto à UCM [2] constituíram a URN [3], que passou a ser adotada como um padrão internacional, em novembro de 2008, pela ITU (*International Telecommunication Union*) [26] [27]. Além de se tratar de um padrão internacional, é a técnica de modelagem já justificada pela JGOOSE e seus usuários já estão familiarizados com os conceitos do *framework*.

Além disso, a ferramenta de que trata este trabalho é frequentemente usada por acadêmicos do curso de Ciência da Computação da Universidade Estadual do Oeste do Paraná - Campus Cascavel. Todos os anos, alunos se debatem com problemas apresentados por outros softwares de modelagem i*. Os questionamentos mais comuns estão relacionados à usabilidade e integridade das ferramentas. Isso acarreta em oportunidades para novas soluções computacionais se apresentarem.

Outro fator, não menos importante, é o gosto pessoal pela área de projeto e desenvolvimento de software. Isto ajudou na tomada de decisão quanto ao foco da pesquisa, bem como resultou na implementação de uma API para o iStarML (Veja o apêndice A).

[2] UCM - *Use Case Maps* , em português: "Mapas de Caso de Uso". Uma técnica de engenharia de software baseada em cenários para descrever relacionamentos entre um ou mais casos de uso.

[3] URN - *User Requirements Notation*, em portugês: "Notação Requisitos de Usuário". Notação destinada a elicitação, análise, especificação e validação de requisitos.

## 1.3 Proposta

A ferramenta JGOOSE, no escopo do seu propósito, já atende as principais necessidades do engenheiro de requisitos. Porém, ainda existe a dependência da ferramenta mencionada (OME) para elaborar os modelos organizacionais e exportá-los em arquivo TELOS.

Dessa forma, o objetivo geral deste trabalho consiste em aumentar os recursos e funcionalidades da ferramenta JGOOSE através do desenvolvimento de uma nova interface para edição de modelos do framework i* integrada à JGOOSE. Ou seja, prover aos usuários da ferramenta JGOOSE uma interface gráfica rica em recursos que facilitem o trabalho de modelagem organizacional, visando diminuir a necessidade de usar outros softwares para esse fim.

Como objetivos específicos, têm-se:

- Estudar sobre as ferramentas de modelagem organizacional i* disponíveis à comunidade.
- Estudar sobre o framework i*, bem como suas variações.
- Estudar o formato de arquivo iStarML e incorporá-lo à ferramenta como o formato de arquivo padrão.
- Realizar um estudo sobre a evolução histórica da ferramenta JGOOSE e analisar sua arquitetura na versão 2013.
- Criar e apresentar exemplos de uso da ferramenta proposta, a fim de verificar as principais funcionalidades da ferramenta na versão final.

## 1.4 Contribuições Esperadas

Após a finalização deste trabalho, deseja-se uma maior independência para a JGOOSE e, consequentemente, seus usuários, além de aumentar o destaque na comunidade e promover a adoção da ferramenta para fins de modelagem i*. Por fim, espera-se uma contribuição significativa diante das ferramentas da comunidade i*, trazendo um reconhecimento para o grupo de pesquisa do LES e todos os envolvidos no desenvolvimento e progresso da JGOOSE. Além disso, seria uma nova ferramenta no quadro comparativo do i* Wiki [2].

## 1.5 Estrutura do Trabalho

Basicamente, o restante deste trabalho encontra-se organizado da seguinte maneira: Nos capítulos 2 e 3 são apresentados alguns fundamentos teóricos necessários para uma melhor compreensão da área de estudo de que trata este trabalho. No capítulo 2, os conceitos básicos e as características do *Framework* i*, bem como suas variações, são apresentados. Já no capítulo 3, após um estudo sobre a evolução histórica do software JGOOSE, uma análise e discussão detalhada de sua arquitetura, na versão 2013, é realizada. No capítulo 4, a proposta é detalhada através de uma visão geral do projeto e arquitetura da nova interface. Também é apresentada uma discussão sobre os principais recursos disponíveis aos usuários. Em seguida, no capítulo 5 é apresentado um estudo de caso, mostrando e avaliando a aplicação da ferramenta em um domínio específico. Finalmente, o capítulo 6 reúne as análises e considerações finais sobre os resultados, bem como relata sobre os possíveis trabalhos futuros.

Outra composição importante, do ponto de vista técnico-computacional, é o apêndice A: uma documentação sobre a API desenvolvida durante a fase de implementação.

# Capítulo 2

# Framework i*, Variações e Ferramentas

Neste capítulo são apresentados os conceitos básicos necessários para o entendimento sobre a técnica de modelagem organizacional do framework i*, bem como os trabalhos baseados nessa técnica. Por fim, discute-se sobre algumas ferramentas de modelagem i* e/ou variações. Inicialmente, na seção 2.1, são apresentados os conceitos fundamentais do i* através dos seus dois componentes de modelagem e alguns dos meta-modelos existentes. Na seção 2.2, mostra-se algumas variações ou extensões da proposta inicial do i* em [28]. Em seguida, na seção 2.3, comenta-se sobre a ferramentas OME, uma ferramenta que geram arquivos de entrada para a JGOOSE. Por fim, na seção 2.5, são feitas algumas considerações finais do capítulo.

## 2.1 O Framework i*

O framework i* (pronunciado "i-star" [1]), originalmente proposto por Yu [28], é um framework de modelagem organizacional conceitual. Ou seja, ajuda no desenvolvimento de modelos que auxiliam a análise de sistemas sob uma visão estratégica e intencional de processos que envolvem vários participantes.

O framework i* preocupa-se principalmente com a análise do contexto organizacional e social de um sistema. O sistema, nesse caso, não consiste somente em componentes técnicos, mas também de elementos humanos. Como o framework i* é bastante flexível para representar situações envolvendo interações entre múltiplos participantes, esse framework pode ser utilizado para representar variados contextos organizacionais.

A seguir, têm-se alguns contextos onde a modelagem i* vem sendo aplicada:

[1] O nome i*, pronunciado em inglês "i-star" faz referência ao conceito sobre uma intencionalidade distribuída. No Brasil, são comuns as pronúncias "i-estrela" e "i-star".

**Engenharia de Requisitos**: é uma das áreas de aplicações mais comuns do i*, principalmente nas fases iniciais do processo de engenharia de requisitos (*Early Requirements*) [13] e [29];

**Modelagem de Negócio (Business Modeling)**: estudos na área apresentaram o uso do i* para visualização explícita da intencionalidade por trás dos processos de negócios. Isso ajuda a se obter um melhor entendimento sobre o trabalho, além de facilitar seu replanejamento [30] [31];

**Desenvolvimento Orientado à Objeto**: em [32] e [33] utilizou-se da pUML (precise UML) [34] e da *Object Constraint Language* (OCL) [35] para tratar dos requisitos finais (*Late Requirements*), além de usar o framework i* para os requisitos iniciais;

**Desenvolvimento Orientado à Agentes**: em [36] apresentou-se o uso de agentes com estrutura BDI (*Believe, Desire and Intention*) [37] para realizar análises na fase inicial de requisitos. Já em [38], utilizou-se de Sistemas Multi-Agentes (SMA) para especificar a estrutura organizacional;

**Segurança, Confiabilidade e Privacidade**: a modelagem i* pode ajudar a lidar com elementos de segurança, confiabilidade e privacidade, através do estudo dos conflitos de intenções de diferentes entidades sociais [39];

Segundo [40], pode-se dizer que o i* é um framework de modelagem tanto orientado a agentes quanto orientado a objetivos, pois sua essência é realizada na combinação de agentes/atores e objetivos. Ambos os paradigmas, Orientação à Agentes [41] e Orientação à Objetivos [42], têm apresentado bons resultados em contextos de modelagem organizacional, principalmente em modelagens da fase inicial (*Early Requirements*) do processo de engenharia de requisitos.

O i* é composto por dois components de modelagem: modelo de dependências estratégicas e modelo de razões estratégicas. Esses componentes auxiliam na representação, respectivamente, das dependências entre atores e dos detalhes por atrás das dependências de cada ator. É fundamental conhecer as notações e saber aplicar esses conceitos para se construir um bom modelo organizacional [43]. A seguir, são apresentados os conceitos e notações por trás dos modelos [2].

### 2.1.1 Modelo de Dependências Estratégicas

O modelo de Dependência Estratégica (SD)[2], representa um conjunto de relacionamento estratégicos externos entre os atores organizacionais, formando uma rede de dependências. Fornece uma visão mais abstrata e ampla da organização, sem se preocupar com os detalhes (razões internas) por trás dessas dependências.

#### Atores, Especializações e Fronteira

i. **Ator** pode ser definido como uma entidade (humana ou computacional) que age sobre o meio que está inserido para conquistar seus objetivos, exercitando seu *know-how* [28]. Atores podem ser vistos como uma referência genérica a qualquer unidade que se possa atribuir dependências intencionais. Os atores possuem relações de dependências com outros atores para um determinado fim. Quando existe uma necessidade de maiores detalhes sobre um modelo organizacional, atores podem ser diferenciados em três especializações: agentes, posições e papéis. A Figura 2.2 exemplifica os tipos de atores, enquanto a Figura 2.1 apresenta um exemplo dos possíveis relacionamento entre os tipos de atores. A seguir, descreve-se esses tipos:

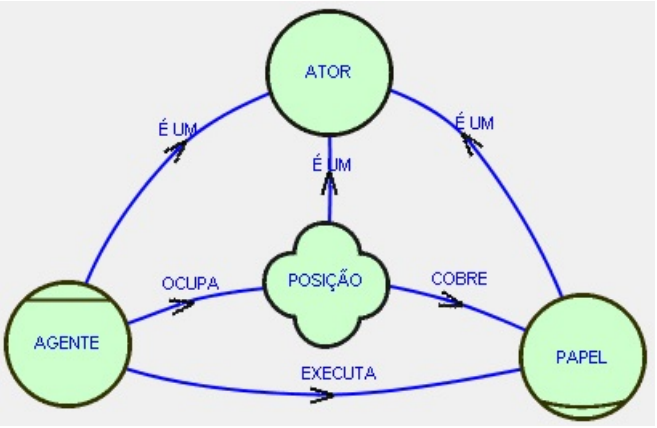
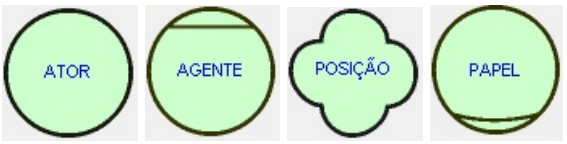



Figura 2.1: Exemplo de relações entre atores [1]

ii. **Agente** é a decomposição de um ator que possui manifestações físicas concretas. Refere-se tanto a humanos quanto a agentes de software ou hardware. Um agente possui dependências independentemente do papel que está executando. As características de um agente

[2]SD, do inglês Strategic Dependency




(a) Ator (b) Agente (c) Posição (d) Papel

Figura 2.2: Atores e especializações.

normalmente não são fáceis de se transferir para outros atores/agentes. São como experiências, habilidades ou, até mesmo, limitações físicas.

iii. **Posição** representa uma abstração intermediária entre um agente e um papel. É o conjunto de papéis tipicamente executados por um agente, ou seja, representa uma posição dentro da organização onde o agente pode desempenhar várias funções (papéis). Diz-se que um agente ocupa uma posição e uma posição cobre um papel.

iv. **Papel** é a caracterização abstrata do comportamento de um ator dentro de determinados contextos sociais ou domínio de informação. Essas características devem ser facilmente transferíveis a outro ator social. As dependências associadas a um papel são aplicáveis independentemente do agente que desempenha o papel.

**Associações entre Atores:** As associações entre os atores são descritas através de links de associação (conforme a Figura 2.3. Essas associações podem ser de seis tipos:

i. ***IS PART OF*** (faz parte de) - Nessa associação cada papel, posição e agente pode ter subpartes. Em *IS PART OF* há dependências intencionais entre o todo e sua parte. Por exemplo, a dependência do todo sobre suas partes para manter a unidade na organização.

ii. ***ISA*** (é um) - Essa associação representa uma generalização, com um ator sendo um caso especializado de outro ator. Ambas, *ISA* e *IS PART OF*, podem ser aplicadas entre quaisquer duas instâncias do mesmo tipo de ator.

iii. ***PLAYS*** (executa) - A associação plays é usada entre um agente e um papel, com um agente executando um papel. A identidade do agente que executa um papel não deverá ter efeito algum nas responsabilidades do papel ao qual está associado, e similarmente, aspectos de

um agente deverão permanecer inalterados mesmo associados a um papel que este desempenha.

iv. ***COVERS*** (cobre) - A associação covers é usada para descrever uma relação entre uma posição e os papéis que esta cobre.

v. ***OCCUPIES*** (ocupa) - Esta associação é usada para mostrar que um agente ocupa uma posição, ou seja, o ator executa todos os papéis que são cobertos pela posição que ele ocupa.

vi. **INS** - Esta associação é usada para representar uma **INS**tância específica de uma entidade mais geral. Por exemplo, quando se deseja representar um agente que é uma instanciação de outro agente.

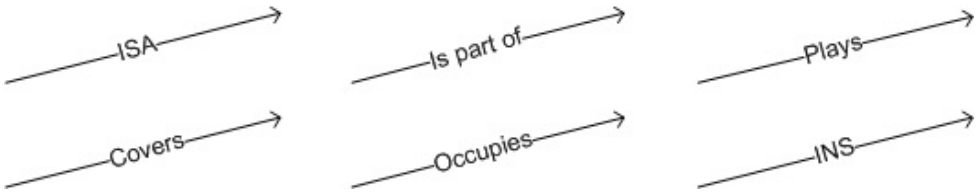



Figura 2.3: Tipos de associações entre atores [2].

**Relação de Dependência** Uma relação de dependência pode ser deifinida como um acordo entre dois atores. Os elementos que compõe uma relação de dependência são:

i. ***Depender***: é o ator dependente, ou seja, o ator que precisa que o acordo (*Dependum*) seja realizado. Esse ator não se importa como o outro ator (*Dependee*) irá satisfazer a necessidade da dependência.

ii. ***Dependum***: é o elemento intermediário, objeto de questionamento e validação, da relação de dependência.

iii. ***Dependee***: é o ator que tem a responsabilidade de satisfazer a relação de dependência.

Dessa forma, pode-se classificar o tipo de uma relação de dependência com base em dos seguintes tipos de *Dependum*:

i. **Objetivo** (*Goal*) - é uma declaração de afirmação sobre um certo estado do mundo. Deve ser de fácil verificação. O *Dependee* é livre para tomar qualquer decisão para satisfazer o objetivo e é esperado que ele o faça. Não importa para o *Depender* como o *Dependee* irá alcançar esse objetivo.

ii. **Tarefa** (*Task*) - é uma atividade a ser realizada pelo *Dependee*. Tarefas podem ser vistas com a realização de operações, processos e etc. Porém, não deve ser uma descrição passo-a-passo ou uma especificação completa de execução de uma rotina.

iii. **Recurso** (*Resource*) - é entidade (física ou informativa) a ser entregue para o *Depender* pelo *Dependee*. Satisfazendo-se esta dependência, o *Depender* está habilitado a usar essa entidade como um recurso.

iv. **Objetivo-Soft** (*Softgoal*) - é semelhante ao Objetivo, porém os critérios de avaliação e verificação são mais subjetivos. O *Depender* pode decidir sobre o que constitui a realização satisfatória do objetivo.

**Ligação de dependência** É uma estritamente a conexão entre os elementos de forma direcionada. Assim, pode-se ter somente duas opções de conexão: início no *Depender*, fim no *Dependum* e início no *Dependum* e fim no *Dependee*. Essa conexão é definida por um segmento contínuo, com a letra "D" sobrescrita, e direcionada da origem para o destino (conforme os exemplos da Figura 2.4).

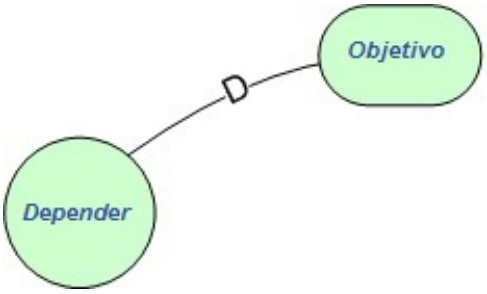
### 2.1.2 Modelos de Razões Estratégicas (SR)

Já o modelo de Razões Estratégicas(SR)[3], representa os detalhes das razões internas que estão por trás das dependências dos atores. Com isso, é possível detalhar os interesses, preocupações e motivações específicas de um ator. Esse tipo de modelo também torna possível a avaliação de alteranativas em definições de processos.

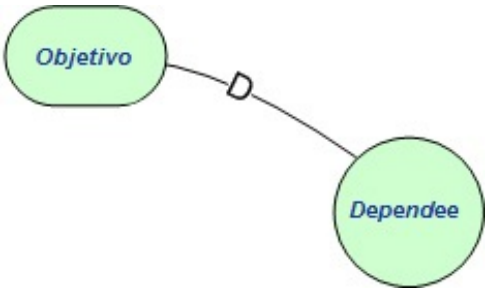
O modelo SR, além de poder conter todos os atores e dependências do SD, "abre-se" os Atores para se detalhar as dependências e expressar as razões. Ou seja, para os atores que precisam ser detalhados, é habilitado o limite da fronteira que deve estar visível e com espaço o suficiente

[3]SR, do inglês Strategic Rationale


(a) Ligação de Dependência.




(b) Ligação de Dependência: do *Depender* para o *Dependum*.

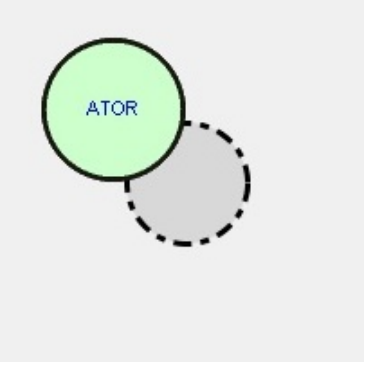



(c) Ligação de Dependência: do *Dependum* para o *Dependee*.

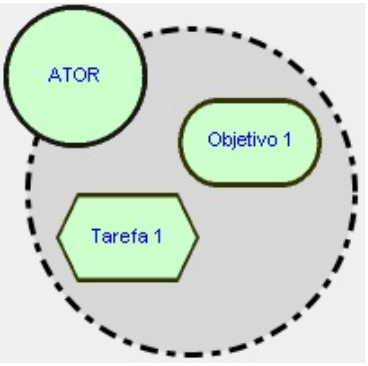
Figura 2.4: Exemplos de ligação de dependência.

para receber os elementos de dependência e/ou ligações internas. Dessa forma, pode-se pensar nos elementos internos ao ator, ou seja, dentro da área de fronteira, como "pertencentes" ao ator. A seguir, são descritos os demais elementos de um modelo SR.

**Fronteira** Uma fronteira indica os limites intecionais de um determinado ator. Todos os elementos dentro dos limites de um ator, são explicitamente desejos ou pretenções desse ator. Uma fronteira é representada por um círculo tracejado e o elemento do ator dessa fronteira deve ser sobreposto a ela, ficando acima do tracejado (conforme a Figura 2.5).




(a) Fronteira Vazia




(b) Fronteira com elementos internos

Figura 2.5: Exemplos de fronteira do ator.

**Ligação de meio-fim (*means-end*)** É representada graficamente por uma seta direcionada ao nó fim, significando o meio para atingir um fim (objetivo, recurso, *softgoal*, ou uma tarefa). A Figura 2.6 exemplifica este tipo de ligação.

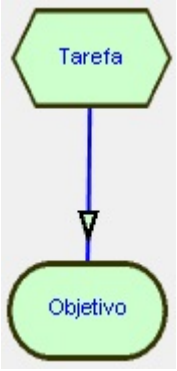



Figura 2.6: Exemplo de ligação meio-fim.

**Ligação de decomposição (*decomposition*):** É responsável por detalhar e expressar da melhor como realizar uma determinada tarefa, através da decomposição em sub-elementos ligados a tarefa principal (superior) através de um segmento de reta cortado. Esses sub-elementos podem ser: metas, tarefas, recursos e objetivos-soft.

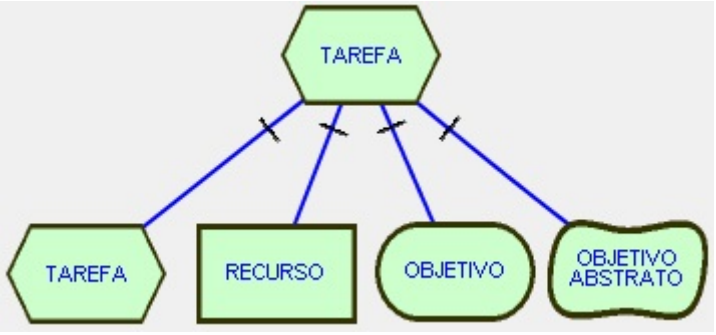



Figura 2.7: Exemplo de ligação de decomposição.

**Ligações de Contribuição (*contribution*):** As ligações de contribuição são para ligar elementos à exclusivamente um objetivo-soft (*softgoal*). Essa ligação ajuda a modelar a forma como os elementos contribuem para a satisfação desse objetivo-soft (*softgoal*). Essas ligações de contribuição, ilustradas na Figura 2.8, podem ser:

i. ***Make***: é uma contribuição positiva, suficientemente forte para satisfazer o objetivo-soft.

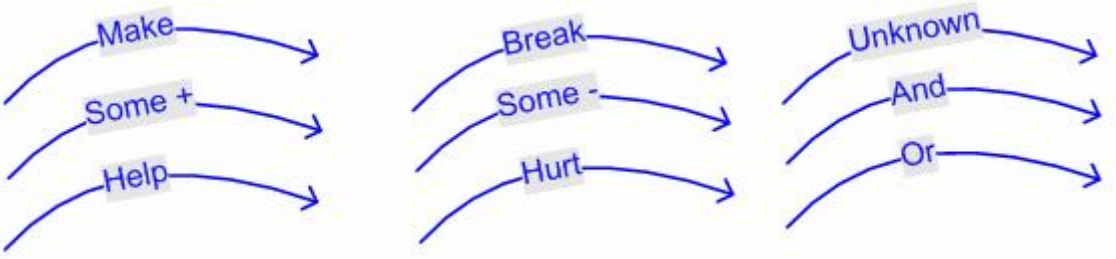



Figura 2.8: Ligações de contribuição [2].

ii. ***Some +***: é uma contribuição positiva, mas cuja força de influência é desconhecida. Pode equivaler a um *make* ou a um *help*.

iii. ***Help***: é uma contribuição positiva fraca, pois não é suficiente para que ela sozinha satisfaça o objetivo-soft.

iv. ***Unknown***: é uma contribuição cuja influência é desconhecida.

v. ***Hurt***: é uma contribuição negativa fraca, porém não é suficiente para que ela sozinha recuse a satisfação de um objetivo-soft.

vi. ***Some -***: é uma contribuição negativa, mas a força de sua influência é desconhecida. Pode equivaler a um *hurt* ou a um *break*.

vii. ***Break***: é uma contribuição negativa, suficientemente forte para rejeitar a satisfação do objetivo-soft.

viii. ***Or***: é uma contribuição onde o objetivo-soft é satisfeito se algum dos descendentes for satisfeitos.

ix. ***And***: é uma contribuição onde o objetivo-soft é satisfeito se todos os descendentes forem satisfeitos.

## 2.2 Variações Baseadas no Framework i*

### 2.2.1 i* Wiki

O i* Wiki, é um projeto criado com o intuito de reunir trabalhos relativos ao i*, de forma colaborativa [2] [44]. Com isso, a comunidade incentiva a colaboração dos usuários do framework, por meio de *feedback* ou mesmo inserção de conteúdo em site oficial. Além disso,

esses usuários podem sugerir alternativas ou extensões sintáticas e semânticas em relação a linguagem utilizada.

Apesar da ampla visão que a comunidade pode ter com os trabalhos divulgados no site, a intenção é fornecer e evoluir uma única versão semântica do i*. Dessa forma, o i* Wiki funciona sobre duas versões do guia para o i* [2]: uma versão estável, servindo de referência para os usuários; outra versão aberta a discussão, acessível aos usuários registrados no site e passível de comentários e sugestões individuais. Além disso, o site reúne um conjunto de Estudos de Casos, Publicações e Eventos relacionados a área de i*.

### 2.2.2 *Goal-Oriented Requirements Language* (GRL)

A GRL é uma linguagem de apoio à modelagem orientada a agentes e objetivos. Assim como o i*, a GRL foca na modelagem dos relacionamentos estratégicos entre atores e seus objetivos. Pode-se pensar como uma alternativa que concentra recursos das metodologias NFR (*Non-Functional Requirements*), i* e Tropos [45]. Outro ponto interessante é que a GRL é escalável, podendo se trabalhar com diferentes níveis de granularidade, em múltiplos diagramas ou visões de um mesmo modelo.

Além disso, uma combinação da GRL com a *Use Case Map* (UCM) deu origem a *User Requirement Notation* (URN) - um padrão internacional do *International Telecommunication Union* (ITU) para notação de requisitos de usuário.

### 2.2.3 Tropos

Tropos é um projeto que foi lançado em 2000 [46], e visa apoiar a construção de sistemas de software orientados a agentes. O projeto reúne um grupo de autores de diversas Universidades no Brasil, Canadá, Bélgica, Alemanha, Itália, etc. O processo de desenvolvimento segundo esta metodologia inicia com um estudo e elaboração de um modelo do ambiente no qual o sistema em desenvolvimento irá operar. Este modelo é refinado até que este represente o ambiente com o sistema em seu contexto. Cada modelo é descrito em termos dos atores observados no ambiente em modelagem, seus objetivos e relacionamentos. A metodologia Tropos oferece um framework que engloba as principais fases de desenvolvimento de software, com o apoio das seguintes atividades: Requisitos Iniciais, Requisitos Finais, Projeto Arquitetural e Projeto

Detalhado.

## 2.3 Ferramenta OME

Atualmente, existem várias ferramentas de modelagem i*. Pode-se encontrar mais de 20 ferramentas referenciadas no site do i* Wiki [2]. Além disso, alguns trabalhos já realizaram comparações sobre ferramentas do framework i*, como em [1](Tabela 6) e no site [2].

O Ambiente de Modelagem Organizacional, tradução literal de OME - Organization Modelling Environment, é um editor gráfico de propósito geral para dar suporte à modelagem orientada a objetivo e/ou orientada a agentes. É uma aplicação Java para *desktop* desenvolvida na Universidade de Toronto [47].

A ferramenta possui recursos que auxiliam o usuário no desenvolvimento e manipulação de modelos i* e NFR (*Non-Functional Requirements*) [48]. Em 2004, o desenvolvimento foi parado e seu código foi portado para a plataforma Eclipse [49], dando origem a sua versão em código aberto chamda OpenOME [50].

Apesar do seu desenvolvimento ter sido finalizado, ainda existem usuários da OME3. Além disso, a ferramenta possui um manual do usuário online (http://www.cs.toronto.edu/km/ome/docs/manual/manual.html) e é de fácil utilização. A maioria dos recursos i*, por exemplo, estão de acordo com [28].

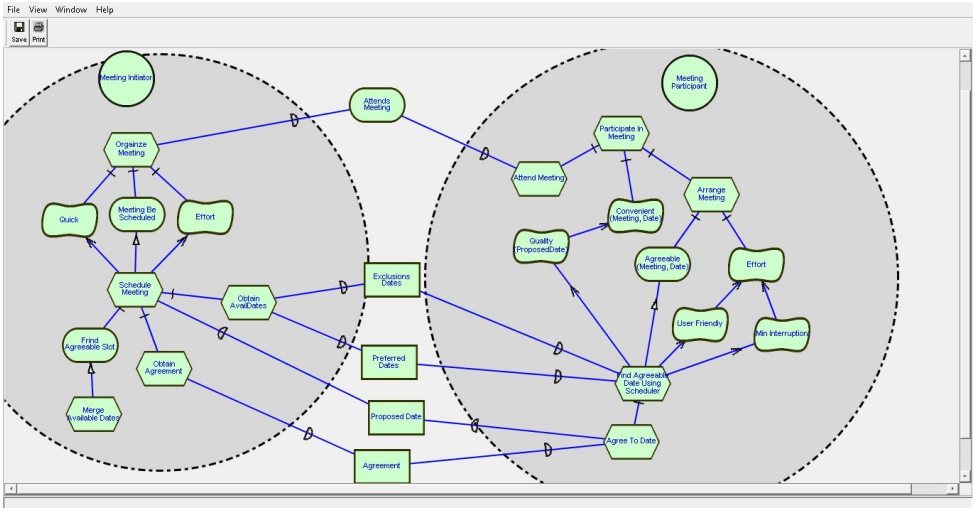



Figura 2.9: Exemplo de modelagem com a ferramenta OME3

Como exemplo, têm-se na Figura 2.9 um dos modelos de exemplos já contidos na ferramenta

OME3 junto à instalação (arquivo: "Meeting-Schedule.tel"). Nessa tela, pode-se observar a barra de ferramentas (*toolbar*) com as opções para criação dos elementos i*: atores, elementos de dependência, links de dependência, links de associação, etc.

Cabe relembrar que essa ferramenta gera os modelos i* no formato TELOS, atualmente aceitos pela ferramenta JGOOSE. Além disso, considerando que a ferramenta apresenta problemas de instabilidade e seu processo de desenvolvimento foi descontinuado, acredita-se que a JGOOSE deva adotar uma nova solução para este fim.

## 2.4 iStarML

Muitas ferramentas foram criadas com base nos conceitos do framework i* ou de variações desse [51] [52]. Isso acabou gerando modelos específicos para cada ferramenta, dificultando o intercâmbio de modelos entre essas ferramentas. Pensando nisso, foi desenvolvida a iStarML. Um meta-modelo baseado em XML (*Extensible Markup Language*) usado para representar modelos i* [53].

O principal objetivo desse meta-modelo é proporcionar um formato de intercâmbio entre os outros formatos de modelos do i*. Ou seja, a especificação iStarML deve suportar todas as outras definições e especificações dos modelos já propostos. Com isso, tudo o que se consegue especificar no formato TELOS, por exemplo, deve-se conseguir também no formato iStarML.

Como prova de conceitos, foi realizado um estudo onde se aplicou o iStarML estritamente para fazer a interconexão entre duas ferramentas diferentes [25]. Nesse estudo, as ferramentas aplicadas foram jUCMNav [54] e a HiME [55].

Porém, existe a preocupação sobre a real adoção da especificação iStarML. Como exemplo, tem-se a promessa da ferramenta OpenOME, que diz estar trabalhando para implementar rotinas de importação/exportação em iStarML [50] [56].

Conforme já mencionado no Capítulo 1, é objetivo dessa pesquisa a implementação e adoção do iStarML como especificação do formato de arquivo padrão. Portanto, detalhes mais específicos sobre a linguagem iStarML serão feitas no Capítulo 4.

## 2.5 Considerações Finais do Capítulo

Neste Capítulo foram apresentados os conceitos gerais do framework i*, bem como as notações da última versão do i* Wiki. Além disso, as variações do i* e seus respectivos metamodelos, são estudados e analisados. Apesar das variações apresentadas, destaca-se a solução de intercâmbio entre diferentes formatos e ferramentas, iStarML. O iStarML foi adotado pelo editor proposto e será discutido melhor no Capítulo 4.

# Capítulo 3

# JGOOSE

Neste capítulo, é apresentada a ferramenta JGOOSE (*Java Goal into Object Oriented Standard Extension*). Inicialmente, na seção 3.1, são apresentados os principais conceitos, objetivos e as diretrizes que norteiam os processos de mapeamento de modelos organizacionais i* para casos de uso UML da ferramenta. Ainda nessa seção, também é apresentado um resumo histórico das versões ao longo dos anos e as principais contribuições de outros autores. Em seguida, na seção 3.2, é analisada a organização estrutural da ferramenta, mostrando os impactos da proposta deste trabalho (E4J) na estrutura e processos da JGOOSE.

## 3.1 Visão Geral

A JGOOSE é uma ferramenta de auxílio no mapeamento de modelos organizacionais para modelos funcionais [15]. Essa ferramenta implementa seus processos guidados pelas diretrizes propostas por Santander [14] e é com base nessas diretrizes que a ferramenta interpreta os modelos organizacionais do framework i* e gera os casos de uso UML, apresentando-os no *template* proposto por Cockburn [23]. Com essa ferramenta, é possível derivar casos de uso com base nas intencionalidades associadas aos atores de um ambiente organizacional.

Na subseção a seguir, apresentaremos as diretrizes e passos da proposta de Santander [14], para posteriormente melhor compreender o funcionamento da ferramenta JGOOSE. E, na subseção seguinte (3.1.2), será apresentado um resumo histórico sobre as principais mudanças já ocorridas no projeto JGOOSE.

### 3.1.1 Diretrizes e Passos

O conjunto de diretrizes e passos propostos por Santander [14] são a essência dos processos realizados pela ferramenta JGOOSE. É com base nessas diretrizes e passos que a ferramenta realiza o mapeamento de modelos organizacionais i* para modelos funcionais de Caso de Uso UML. A seguir, são descritas brevemente essas diretrizes e passos [19]:

1º Passo: Descoberta de atores;

Diretriz 1: todo ator em i* deve ser analisado sob um possível mapeamento para ator em caso de uso.

Diretriz 2: deve-se analisar se o ator em i* é externo ao sistema computacional pretendido, pois atores em caso de uso nunca são partes do sistema. Caso o ator seja externo ao sistema, o mesmo é considerado candidato a ator em Casos de Uso.

Diretriz 3: se o ator i* candidato tiver pelo menos uma dependência com o sistema computacional pretendido, esse deve ser um ator em caso de uso.

Diretriz 4: atores em i* relacionados através do mecanismo ISA, ou seja, com heranças de suas atividades nos modelos organizacionais e mapeados individualmente para atores em casos de uso (aplicando diretrizes 1, 2 e 3), serão relacionados no diagrama de casos de uso através do relacionamento do tipo «*generalization*».

2º Passo: Descoberta de Casos de Uso;

Diretriz 5: para cada ator descoberto para o sistema no 1º passo, devemos observar todas as suas dependências (*dependum*) como *dependee* em relação ao ator que representa o sistema computacional pretendido (*depender*), visando descobrir casos de uso para o ator.

Subdiretriz 5.1: deve-se analisar as dependências do tipo objetivo associadas com o ator e mapeadas diretamente para casos de uso.

Subdiretriz 5.2: deve-se avaliar as dependências do tipo tarefa associadas com o ator. Se um ator depende de outro ator para realizar uma tarefa, deve-se investigar se esta tarefa necessita ser refinada em subtarefas. Este tipo de tarefa pode ser mapeada para caso de uso.

Subdiretriz 5.3: deve-se avaliar as dependências do tipo recurso associadas com o ator. Se um ator depende de outro ator para obter um recurso, por que o mesmo é requerido? Se para esta resposta existe um objetivo, o mesmo será candidato a ser um objetivo de um caso de uso para este ator.

Subdiretriz 5.4: deve-se avaliar as dependências do tipo objetivo-soft associadas com o ator. Normalmente uma dependência do tipo objetivo-soft em i* é um requisito não-funcional associado ao sistema pretendido.

Diretriz 6: analisar a situação especial na qual um ator de sistema (descoberto seguindo as diretrizes do passo 1) possui dependências (como *depender*) em relação ao ator em i* que representa o sistema computacional pretendido ou parte dele (ator $\rightarrow$ *dependum* $\rightarrow$ sistema computacional).

Diretriz 7: classificar cada caso de uso de acordo com seu tipo de objetivo associado: contextual, de usuário ou de subfunção.

3º Passo: Especificação de Caso de Uso;

Diretriz 8: analisar cada ator e seus relacionamentos no modelo de Razões Estratégicas (SR) para extrair informações que possam conduzir à descrição de fluxos principais e alternativos, bem como, pré-condições e pós-condições dos casos de uso descobertos para o ator. Para isso precisamos analisar os subcomponentes em uma ligação de decomposição de tarefa mapeando-os para passos na descrição do cenário primário (fluxo principal) de casos de uso. Também devemos analisar ligações do tipo meio-fim mapeando os meios para passos alternativos na descrição de casos de uso.

Diretriz 9: investigar a possibilidade de derivar novos objetivos de casos de uso a partir da observação dos passos nos cenários (fluxos de eventos) dos casos de uso descobertos.

Diretriz 10: Desenvolver o diagrama de casos de uso utilizando os casos de uso descobertos e os relacionamentos do tipo «*include*», «*extend*» e «*generalization*» usados para estruturar as especificações dos casos de uso.

### 3.1.2 Resumo Histórico

Desde a sua primeira versão [15], a ferramenta JGOOSE passou por várias melhorias e aprimoramentos. Mudanças essas que variam desde a refatoração de código fonte (Classes e *Packages* Java) até alterações na interface gráfica do usuário [3]. A seguir, é apresentado um resumo sobre a origem da JGOOSE e as principais alterações já realizadas na ferramenta.

- **GOOSE** - A *Goal into Object Oriented Standard Extension* (GOOSE), foi a ferramenta que antecedeu à JGOOSE. Implementada com a linguagem de programação *Object Pascal* e com a extensão *Rational Rose* por Marcelo Brischke [17], possuía dependências das ferramentas OME e Rational Rose. Devido a dependência dessa última, que é uma solução proprietária, a GOOSE foi distribuída sob uma licença educacional. Analisando sob aspectos técnicos e funcionais, a ferramenta não contemplava todas as diretrizes propostas por Santander [14]. Mais especificamente, as diretrizes 5.7, 7 e 9 não eram satisfeitas pelo processo do sistema [17].

- **JGOOSE versão 2006** - Desenvolvida por André Abe Vicente [15], a nova implementação passou a ser na linguagem Java e, com isso, foi atribuído o nome de JGOOSE (*Java Goal into Object Oriented Standard Extension*). Por ser em uma nova linguagem de programação, todo o projeto teve que ser re-implementado em Java. Entre outros aspectos, a solução continuou dependente da ferramenta OME, contudo não utilizava mais a extensão Rational Rose. Além disso, essa versão contou com a implementação da subdiretriz 5.4, aperfeiçoamento de outras subdiretrizes e novas funcionalidades de auxílio ao usuário no processo de mapeamento [15].

- **JGOOSE versão 2011** - Melhorada por Mauro Brischke [3], a nova versão contempla a implementação de três diretrizes faltantes: 8, 9 e 10. Também foi fruto desse trabalho a implementação da exportação dos casos de uso no formato XMI, melhorando a comunicação com outras ferramentas como a StarUML. Nessa versão, foi implementado soluções que permitiram o usuário da ferramenta a realizar um refinamento manual dos casos de uso gerados, bem como visualizar graficamente os casos de uso na forma de imagens estáticas.

- **JGOOSE versão 2013** - Essa versão está em fase de desenvolvimento por Diego Peliser

[20] e visa a correção de alguns *bugs* na aplicação das diretrizes e a otimizações de código (algoritmos), bem como melhorias na interface gráfica e na documentação.

## 3.2 Projeto e Arquitetura

Conforme o resumo histórico, a JGOOSE sofreu várias alterações ao longo dos anos. Entretanto, nenhuma dessas alterações influenciaram de forma significativa o fluxo de dados tradicional da ferramenta, mantendo como base as diretrizes e passos propostos por Santander [14]. A figura 3.1 foi apresentada por Brischke [3] como sendo a represetanção da arquitetura da ferramenta JGOOSE.

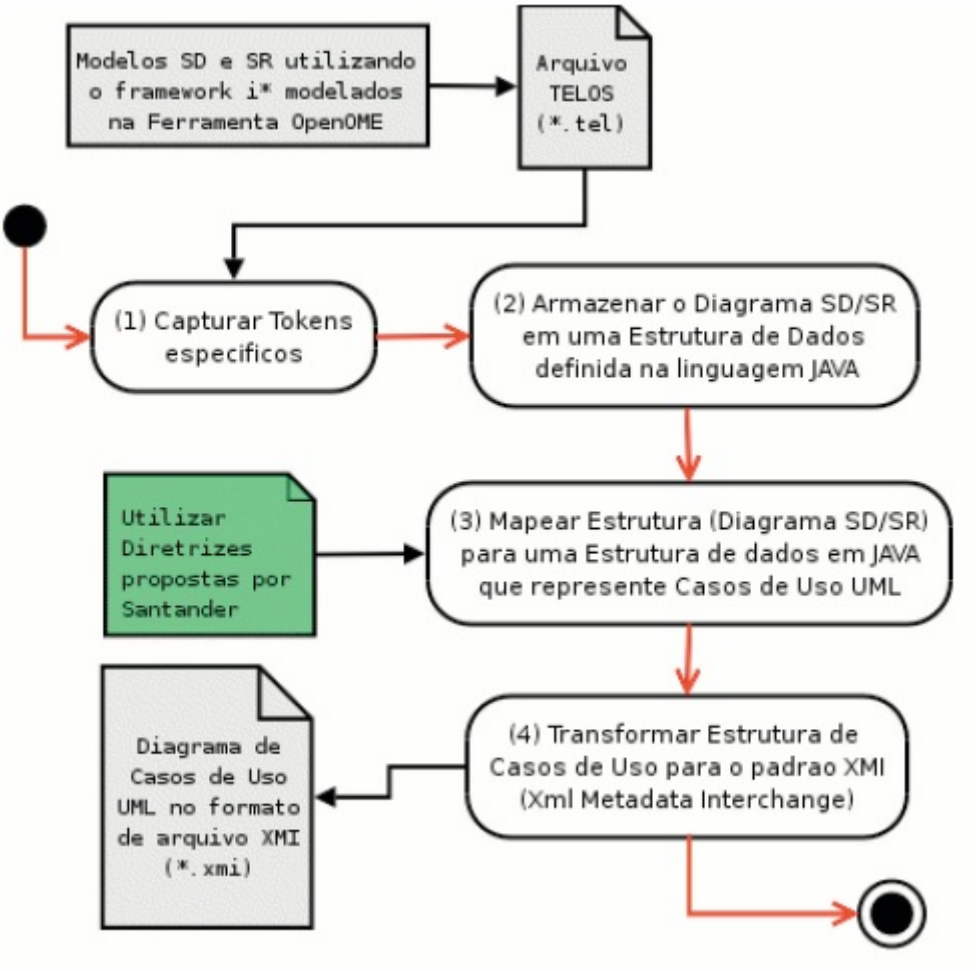



Figura 3.1: Figura apresentada por Brischke como "Arquitetura da ferramenta JGOOSE Demonstrando o seu Funcionamento" [3].

A figura 3.1 foi analisada e refeita sobre os conceitos de DFD (*Data Flow Diagram*) nos

modelos de Gane e Sarson [4]. Antes de apresentar o novo diagrama sobre o fluxo de dados da JGOOSE, precisamos entender os elementos do DFD usados nesse diagrama.

Nesse tipo de modelo (DFD), temos os seguintes elementos (apresentados na figura 3.2):

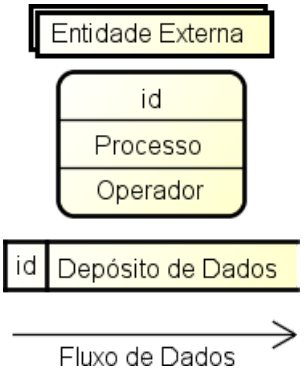



Figura 3.2: Elementos da notação Gane e Sarson [4] de DFD.

- **Entidade Externa** - representam elementos de fora do sistema, mas que comunicam-se com ele. São fontes de entradas de dados para o sistema ou destinos para as saída dos dados e contém apenas o *nome* da entidade na sua representação.

- **Processo** - representa uma transformação dos dados de entrada em dados de saída. Contém um *id*entificador único, um *nome* do processo e o *operador* do processo.

- **Depósito de Dados** - representa um repositório de dados no sistema, podendo ser tanto em banco de dados quanto em arquivos. Contém um *id*entificador único e o *nome* do depósito de dados.

- **Fluxo de Dados** - representam canais através do qual passam as informações. Aplica-se uma etiqueta para representar os dados que "passam" por esse canal.

Em modelos de DFD, existem os chamados *diagramas de contexto DFD*, que expressam, em mais alto nível de abstração, a interação entre *entidades externas* e somente um *processo* do sistema [4]. Esse único processo deve resumir e representar as funções principais do sistema. Com base nesses conceitos, foi construído um diagrama de contexto DFD (figura 3.3) para representar o fluxo de dados entre a *entidade externa* OME3, o *processo* realizado pelo *operador* JGOOSE e a *entidade externa* StarUML.

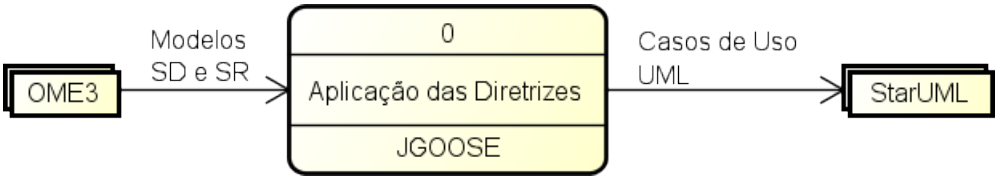



Figura 3.3: Diagrama de contexto da Ferramenta JGOOSE.

Para melhor representar o fluxo de dados da JGOOSE, um diagrama com maiores detalhes nos processos foi construído e apresentados na figura 3.4. Esse diagrama visa representar a "antiga estrutura" da figura 3.1 na forma de DFD tradicional, mostrando a *entidade externa* OME3, os *processos* da atual JGOOSE e *entidade externa* StarUML.

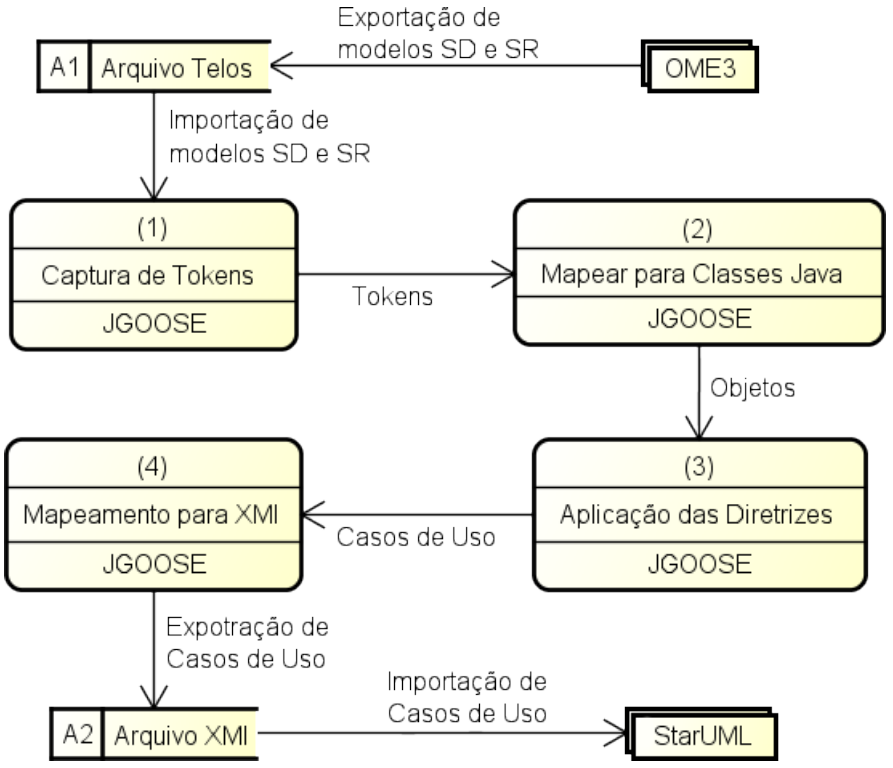



Figura 3.4: Diagrama de Fluxo de Dados da Ferramenta JGOOSE.

### 3.2.1 Impacto da E4J

Para realizar a integração da E4J com a JGOOSE, serão necessárias algumas modificações do fluxo de dados original da JGOOSE. Conforme a proposta deste trabalho, que trata da manipulação de diagramas SD e SR, os processos "(1) - Captura de Tokens - JGOOSE" e "(2) - Mapear para Classes Java - JGOOSE" deverão ser gerenciados pela E4J. Dessa forma, um novo

diagrama de fluxo de dados com a E4J, é apresentado a seguir (figura 3.5):

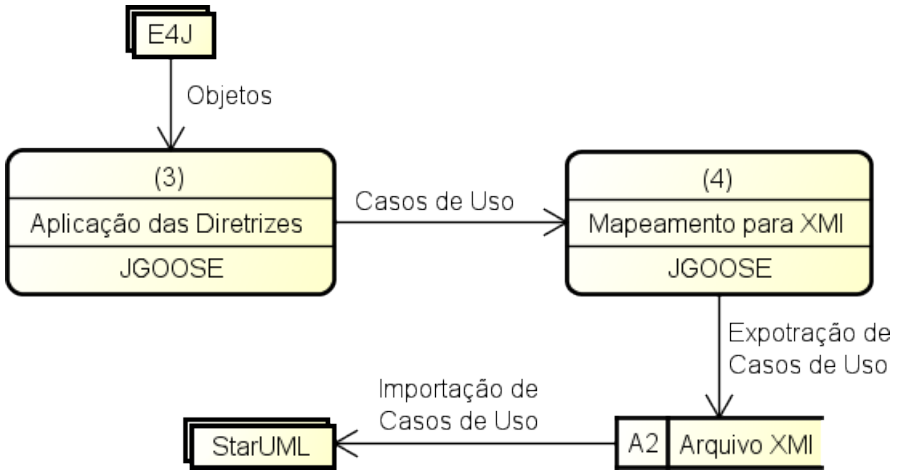



Figura 3.5: Novo DFD da ferramenta JGOOSE, após integração com E4J.

Após a integração entre as duas ferramentas, não será mais possível executar diretamente a JGOOSE, pois essa passa a ser um módulo interno da E4J (mais detalhes dessa integração serão apresentados na seção 4.2.3 do capítulo 4). O processo de mapeamento se iniciará dentro da E4J e, após as rotinas de mapeamento da estrutura em iStarML para a estrutura legada da JGOOSE, a linha de execução do programa passa a ser de responsabilidade da JGOOSE. Somente após o fechamento da interface da JGOOSE é que a linha de execução retorna ao gerenciamento da E4J.

Com isso, um novo diagrama de contexto DFD, onde a E4J passa a ser o processo principal, é apresentada na figura 3.6.

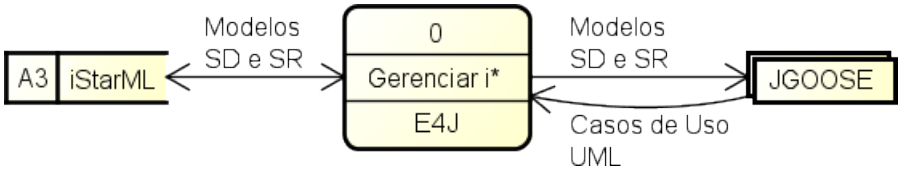



Figura 3.6: Novo diagrama de contexto com a E4J sendo o processo principal.

## 3.3 Considerações Finais do Capítulo

Considerando os processos que serão de responsabilidade da E4J, é conveniente pensar na incorporação da JGOOSE à E4J, pois é com a E4J que se inciam os processos que antecedem

o mapeamento da JGOOSE. Dessa forma, a JGOOSE passará a cuidar especificamente das rotinas (diretrizes e passos) de mapeamento da estrutura (objetos e classes) i* para a estrutura de casos de uso UML, enquanto a E4J trata dos processos iniciais de manipulação dos diagramas organizacionais. Em trabalhos futuros, pode-se estudar a viabilidade da alteração de algumas estruturas e rotinas da JGOOSE para usar as mesmas estruturas (objetos e classes) que a E4J. Vale ressaltar que a integração **não afetará** a forma com a qual a JGOOSE realiza suas funções de mapeamento. Apenas deixará de executar suas rotinas de interpretação dos modelos organizacionais feitos em TELOS.

# Capítulo 4

# Proposta

## 4.1 Visão Geral

### 4.1.1 Conceitos e Considerações Iniciais

## 4.2 Projeto e Arquitetura

### 4.2.1 Modelagem i*

### 4.2.2 Arquitetura Modular

### 4.2.3 Maven

## 4.3 Desenvolvimento

## 4.4 Recursos

### 4.4.1 Internacionalização (i18n)

### 4.4.2 UndoManager

### 4.4.3 API iStarML

## 4.5 Testes e Validação

### 4.5.1 Testes Unitários

## 4.6 Considerações Finais do Capítulo

# Capítulo 5

# Exemplos de Uso

## 5.1 Resultados e Análises

## 5.2 Considerações Finais do Capítulo

# Capítulo 6

# Considerações Finais

## 6.1 Contribuições

## 6.2 Trabalhos Futuros

# Apêndice A

# iStarML API

## A.1 Introdução

...

## A.2 Projeto e Arquitetura

...

## A.3 Documentação

...

# Referências Bibliográficas


[1] SANTOS, B. S. *IStar Tool-Uma proposta de ferramenta para modelagem de i**. Tese (Dissertação de Mestrado) — Centro de Informática, Recife, PE, outubro 2008.

[2] AACHEN, R. *i* Wiki*. 2013. Acessado em: 04 de junho de 2013. Disponível em: <http://istarwiki.org/>.

[3] BRISCHKE, M. *Melhorando a Ferramenta JGOOSE*. Cascavel - PR: [s.n.], Dezembro 2012.

[4] GANE, C. P.; SARSON, T. *Structured systems analysis: tools and techniques*. [S.l.]: McDonnell Douglas Systems Integration Company. ISBN 0930196007. 1977.

[5] GRAU, G. et al. A comparative analysis of i* agent-oriented modelling techniques. In: *Proceedings of the Eighteenth International Conference on Software Engineering and Knowledge Engineering, SEKE*. [S.l.: s.n.], p. 5–7. 2006.

[6] CASE, A. F. Computer-aided software engineering (case): technology for improving software development productivity. *ACM SIGMIS Database*, ACM, v. 17, n. 1, p. 35–43, 1985.

[7] KOTONYA, G.; SOMMERVILLE, I. *Requirements engineering: processes and techniques*. [S.l.]: J. Wiley. (Worldwide series in computer science). ISBN 9780471972082. 1998.

[8] LAMSWEERDE, A. V. Requirements engineering in the year 00: A research perspective. In: ACM. *Proceedings of the 22nd international conference on Software engineering*. [S.l.], p. 5–19. 2000.

[9] SCHNEIDER, F.; BERENBACH, B. A literature survey on international standards for systems requirements engineering. *Procedia Computer Science*, Elsevier, v. 16, p. 796–805, 2013.

[10] MASON, G. L. A conceptual basis for organizational modelling. *Systems Research and Behavioral Science*, Wiley Online Library, v. 14, n. 5, p. 331–345, 1997.

[11] YU, E. Modeling organizations for information systems requirements engineering. In: IEEE. *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*. [S.l.], p. 34–41. 1993.

[12] YU, E. Agent orientation as a modelling paradigm. *Wirtschaftsinformatik*, Springer, v. 43, n. 2, p. 123–132, 2001.

[13] YU, E. Towards modelling and reasoning support for early-phase requirements engineering. In: IEEE. *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on*. [S.l.], p. 226–235. 1997.

[14] SANTANDER, V. F. A. *Integrando Modelagem Organizacional com Modelagem Funcional*. Tese (Tese de Doutorado) — Universidade Federal de Pernambuco, Recife - PE, dezembro 2002.

[15] VICENTE, A. A. *JGOOSE: Uma ferramenta de Engenharia de Requisitos para Integração da Modelagem Organizacional i* com a Modelagem Funcional de Casos de Uso UML.* Cascavel - PR: [s.n.], Dezembro 2006.

[16] PEDROZA, F. P. et al. Ferramentas para suporte do mapeamento da modelagem i* para a uml: eXtended GOOD XGOOD e GOOSE. In: *Proceedings of the VII Workshop on Requirements Engineering-WER*. [S.l.: s.n.], v. 4, p. 164–175. 2004.

[17] BRISCHKE, M. *Desenvolvimento de uma Ferramenta para Integrar Modelagem Organizacional e Modelagem Funcional na Engenharia de Requisitos*. Cascavel - PR: [s.n.], Novembro 2005.

[18] VICENTE, A. A. et al. JGOOSE:a requirements engineering tool to integrate i* organizational modeling with use cases in uml JGOOSE: Una herramienta de ingenierÍa de requisitos para la integraciÓn del modelado organizacional i* con el modelado de casos de uso en uml. *Ingeniare. Revista chilena de ingeniería*, SciELO Chile, v. 17, n. 1, p. 6–20, 2009.

[19] BRISCHKE, M.; SANTANDER, V. F. A.; SILVA, I. F. da. Melhorando a ferramenta JGOOSE. In: *XV Workshop de Engenharia de Requisitos*. [S.l.: s.n.]. 2012.

[20] PELISER, D. Aprimorando a ferramenta JGOOSE. Projeto de Iniciação Científica (em andamento) - PIBIC. Abril 2013.

[21] MYLOPOULOS, J. et al. Telos: Representing knowledge about information systems. *ACM Transactions on Information Systems (TOIS)*, ACM, v. 8, n. 4, p. 325–362, 1990.

[22] KOUBARAKIS, M. et al. *Telos: Features and formalization*. [S.l.]: Computer Science Institute, Foundation of Research and Technology, Hellas. 1989.

[23] COCKBURN, A. *Writing effective use cases*. [S.l.]: Addison-Wesley Reading. 2001.

[24] CARES, C. et al. *iStarML Reference's Guide*. [S.l.], 2007.

[25] COLOMER, D. et al. Model interchange and tool interoperability in the i* framework: a proof of concept. In: *Proc. of the 14th Workshop on Requirements Engineering*. [S.l.: s.n.], p. 369–381. 2011.

[26] AMYOT, D. Introduction to the user requirements notation: learning by example. *Computer Networks*, Elsevier, v. 42, n. 3, p. 285–301, 2003.

[27] AMYOT, D.; MUSSBACHER, G. *URN: Towards a new standard for the visual description of requirements*. 2003.

[28] YU, E. S.-K. *MODELLING STRATEGIC RELATIONSHIPS FOR PROCESS REENGINEERING*. Tese (Doutorado) — University of Toronto, 1995.

[29] MAIDEN, N. A. et al. Model-driven requirements engineering: synchronising models in an air traffic management case study. In: SPRINGER. *Advanced Information Systems Engineering*. [S.l.], p. 368–383. 2004.

[30] YU, E. S.; MYLOPOULOS, J.; LESPÉRANCE, Y. Al models for business process reengineering. *IEEE expert*, IEEE, v. 11, n. 4, p. 16–23, 1996.

[31] KOLP, M.; GIORGINI, P.; MYLOPOULOS, J. Organizational patterns for early requirements analysis. *Lecture Notes in Computer Science*, Springer, v. 2681, p. 617–632, 2003.

[32] CASTRO, J.; ALENCAR, F.; CYSNEIROS, G. Closing the gap between organizational requirements and object oriented modeling. *Journal of the Brazilian Computer Society*, SciELO Brasil, v. 7, n. 1, p. 05–16, 2000.

[33] CASTRO, J. F. et al. Integrating organizational requirements and object oriented modeling. In: IEEE. *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*. [S.l.], p. 146–153. 2001.

[34] EVANS, A.; KENT, S. Core meta-modelling semantics of uml: the puml approach. In: *«UML»'99—The Unified Modeling Language*. [S.l.]: Springer. p. 140–155. 1999.

[35] WARMER, J.; KLEPPE, A. *The object constraint language: getting your models ready for MDA*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc. 2003.

[36] BRESCIANI, P. et al. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, Springer, v. 8, n. 3, p. 203–236, 2004.

[37] RAO, A. S.; GEORGEFF, M. P. et al. BDI agents: From theory to practice. In: SAN FRANCISCO. *Proceedings of the first international conference on multi-agent systems (ICMAS-95)*. [S.l.], p. 312–319. 1995.

[38] BASTOS, L. R.; CASTRO, J. F. Enhancing requirements to derive multi-agent architectures. In: *Proceedings of WER*. [S.l.: s.n.], p. 127–139. 2004.

[39] YU, E.; LIU, L. Modelling trust for system design using the i* strategic actors framework. In: *Trust in Cyber-societies*. [S.l.]: Springer. p. 175–194. 2001.

[40] YU, E.; GIORGINI, P.; MAIDEN, N. *Social modeling for requirements engineering*. [S.l.]: Mit Press. 2011.

[41] MAO, X.; YU, E. Organizational and social concepts in agent oriented software engineering. In: *Agent-Oriented Software Engineering V*. [S.l.]: Springer. p. 1–15. 2005.

[42] LAMSWEERDE, A. van. Goal-oriented requirements enginering: a roundtrip from research to practice [enginering read engineering]. In: IEEE. *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International*. [S.l.], p. 4–7. 2004.

[43] WEBSTER, I. et al. A survey of good practices and misuses for modelling with i* framework. In: *Proc. of the VIII Workshop on Requirements Engineering, WER*. [S.l.: s.n.], v. 5, p. 148–160. 2005.

[44] LEUF, B.; CUNNINGHAM, W. The wiki way: quick collaboration on the web. Addison-Wesley Professional, 2001.

[45] REGEV, G.; WEGMANN, A. Where do goals come from: the underlying principles of goal-oriented requirements engineering. In: IEEE COMPUTER SOCIETY. *Proceedings of the 13th IEEE International Conference on Requirements Engineering*. [S.l.], p. 253–362. 2005.

[46] MYLOPOULOS, J.; KOLP, M.; CASTRO, J. Uml for agent-oriented software development: The tropos proposal. In: *The Unified Modeling Language. Modeling Languages, Concepts, and Tools*. [S.l.]: Springer. p. 422–441. 2001.

[47] YU, E.; YU, Y. *Organization Modelling Environment*. 2013. Consultado na INTERNET: http://www.cs.toronto.edu/km/ome/, 2013.

[48] CHUNG, L. et al. Non-functional requirements. *Software Engineering*, 2000.

[49] FOUNDATION, E. *Eclipse*. 2013. Consultado na INTERNET: http://www.eclipse.org/, 2013.

[50] HORKOFF, J.; YU, Y.; YU, E. Openome: an open-source goal and agent-oriented model drawing and analysis tool. In: *CEUR proceedings of the 5th international i* workshop (iStar 2011)*. [S.l.: s.n.], p. 154–156. 2011.

[51] CARES, C.; FRANCH, X. Towards a framework for improving goal-oriented requirement models quality. 2009.

[52] CARES, C. et al. Towards interoperability of *i** models using istarml. *Computer Standards & Interfaces*, Elsevier, v. 33, n. 1, p. 69–79, 2011.

[53] CARES, C. et al. istarml: An xml-based model interchange format for i*. In: *Proc. 3rd Int. i* Workshop, Recife, Brazil*. [S.l.: s.n.], v. 322, p. 13–16. 2008.

[54] KEALEY, J. et al. Integrating an eclipse-based scenario modeling environment with a requirements management system. In: IEEE. *Electrical and Computer Engineering, 2006. CCECE'06. Canadian Conference on*. [S.l.], p. 2432–2435. 2006.

[55] LóPEZ, L.; FRANCH, X.; MARCO, J. Hime: hierarchical i* modeling editor. *Revista de Informática Teórica e Aplicada*, v. 16, n. 2, p. 57–60, 2009.

[56] LAUE, R.; STORCH, A. Adding functionality to openome for everyone. In: *CEUR Proceedings of the 5th International i* Workshop (iStar 2011)*. [S.l.: s.n.], v. 766, p. 169–171. 2011.