# Ferramentas para Suporte do Mapeamento da Modelagem i* para a UML: eXtended GOOD – XGOOD e GOOSE

Flávio Pereira Pedroza[1], Fernanda M. R. Alencar[1], Jaelson F. B. Castro[2], Fernando R. C. Silva[2], Victor F. A. Santander[3]
[1]Departamento de Eletrônica e Sistemas - Universidade Federal de Pernambuco
[2]Departamento de Informática - Universidade Federal de Pernambuco
[3]Departamento de Informática - Universidade do Oeste do Paraná
Recife, Brazil.
e-mail: flavio.pedroza@caeser.org.br, fmra@ufpe.br, {jbc,frcs}@cin.ufpe.br, vfasantander@unioeste.br

**Resumo.** O bom entendimento dos requisitos organizacionais é vital para o sucesso do desenvolvimento de aplicações na área de engenharia de software. Com a popularização do paradigma da orientação à objeto, a linguagem de modelagem UML (Unified Modeling Language) se tornou padrão para este tipo de desenvolvimento. Porém, a UML ainda não está suficientemente estruturada para suportar a modelagem dos requisitos organizacionais. Faz-se necessário a utilização de outras ferramentas de modelagem. A técnica de modelagem i* (istar) supre esta deficiência, sendo uma técnica bem difundida e aceita. Neste trabalho, apresentamos ferramentas de apoio às regras de mapeamento entre as técnicas i* e UML, o eXtended GOOD (Goals into Object Oriented Development) e o GOOSE (Goal Into Object Oriented Standard Extension) que possibilitam a geração de diagramas UML a partir de modelos i* de forma automática.

**Palavras-chave:** Modelagem i*, Modelagem UML, XMI , Plataforma OME

## 1. Introdução

Qualquer sistema deve atender às necessidades da organização do qual irá fazer parte. Para isto, faz-se necessário entender de forma apropriada os interesses e necessidades da organização. A técnica i* [1] possui papel fundamental para os estágios iniciais do desenvolvimento. Com ela é possível realizar a captura dos requisitos organizacionais que irão influenciar na determinação dos requisitos funcionais do sistema. A captura destes requisitos é crucial para o sucesso do desenvolvimento do sistema. Já a técnica UML [2] se ajusta melhor aos estágios posteriores do desenvolvimento. Apesar de possuir métodos para capturar os requisitos iniciais (diagrama de casos de uso), ela falha em indicar como o sistema atende aos objetivos da organização, quais foram as alternativas consideradas e o porquê das soluções adotadas. Faz-se necessário, então, a associação destas duas técnicas. Com o objetivo de manter a consistência entre o

sistema e o os objetivos da organização e estabelecer o impacto que uma mudança acarreta no sistema e vice-versa, faz-se necessário estabelecer um relacionamento de rastreamento. Para isso, foram propostas diretrizes de mapeamento, que são regras que determinam como os objetos em i* serão transformados nos elementos do diagrama de classes [3, 4] ou no diagrama de casos de uso da UML.
Para auxiliar e automatizar este mapeamento, foram propostas ferramentas capazes de ler o diagrama i* gerado pela ferramenta OME [5] e gerar de forma automática os diagramas da UML. O Artigo está estruturado da seguinte forma: na seção 2 temos uma introdução da técnica de modelagem i*, utilizada para modelagem organizacional; na seção 3 apresentamos as diretrizes de mapeamento utilizadas para gerar os diagramas da UML a partir do modelo i*; na seção 4 vemos as ferramentas CASE de apoio utilizadas para gerar este mapeamento, inclusive as novas ferramentas propostas; finalmente, na seção 5, vemos as considerações finais e propostas de trabalhos futuros.

## 2. Técnica i* (*istar*)

A técnica i* foi desenvolvida como um método de modelagem orientado a agentes, centrado na característica intencional dos mesmos. A técnica de modelagem i* foca nos relacionamentos entre os atores e suas dependências. Os atores são vistos como tendo propriedades intencionais, tais como objetivos, opiniões, habilidades e compromissos. Os atores dependem uns dos outros para cumprir objetivos, realizar tarefas e fornecer recursos. Através da cooperação de outros, um ator pode alcançar objetivos que serão difíceis de serem alcançados sozinho.
O framework i* consiste de dois modelos: o modelo de Dependência Estratégica (**SD** - *Strategic Dependency*), e o modelo de Dependência Racional (**SR** - *Strategic Rationale*).

### 2.1 Modelo SD

O modelo de Dependência Estratégica é uma rede de relacionamentos de dependência entre atores. O modelo SD foca na captura na estrutura intencional de um processo, ou seja, na captura das motivações e intenções por trás das atividades e fluxos em um processo.
O modelo consiste uma série de nós e *links*. Cada nó representa um ator. Um ator é uma entidade ativa que realiza ações para atingir determinados objetivos. Cada *link* entre dois atores indica que um ator depende do outro para, de alguma forma, cumpri algum objetivo. O ator que depende de outro é chamado de *depender* e o ator responsável por cumpri a dependência é chamado de *dependee*. O objetivo da dependência é denominado *dependum.*
Através da dependência, o *depender* é capaz de alcançar objetivos que não seria capaz de alcançar sozinho. Existem quatro tipos de dependências: dependência de **objetivo** (*goal*), dependência de **tarefa** (*task*), dependência de **recurso** (*resource*) e dependência de **objetivo *soft*** (*soft goal).*
O modelo SD distingue três tipos de atores: **agente**, **papel** e **posição**. Dizemos que um agente ocupa uma posição e que uma posição cobre um papel.

### 2.2 Modelo SR

O modelo de dependência estratégica mostra somente os relacionamentos externos entre os atores. No modelo de razão estratégica (*Strategic Rationale model – SR)*, temos uma representação e uma racionalização mais explícita sobre os interesses dos atores, e de como estes interesses podem ser atendidos ou afetados pelos diversos sistemas.
No modelo SR temos uma visão interna dos atores, proporcionando um nível de detalhamento maior do modelo. Os elementos intencionais (**objetivos**, **tarefas**, **recursos** e **objetivos *soft***) aparecem no modelo SR como elementos internos ligados por relacionamentos meio-fim e decomposição de tarefas. Estes relacionamentos proporcionam uma representação explicita das razões por trás das dependências entre os atores e quais são alternativas por trás dos processos (Figura 1).

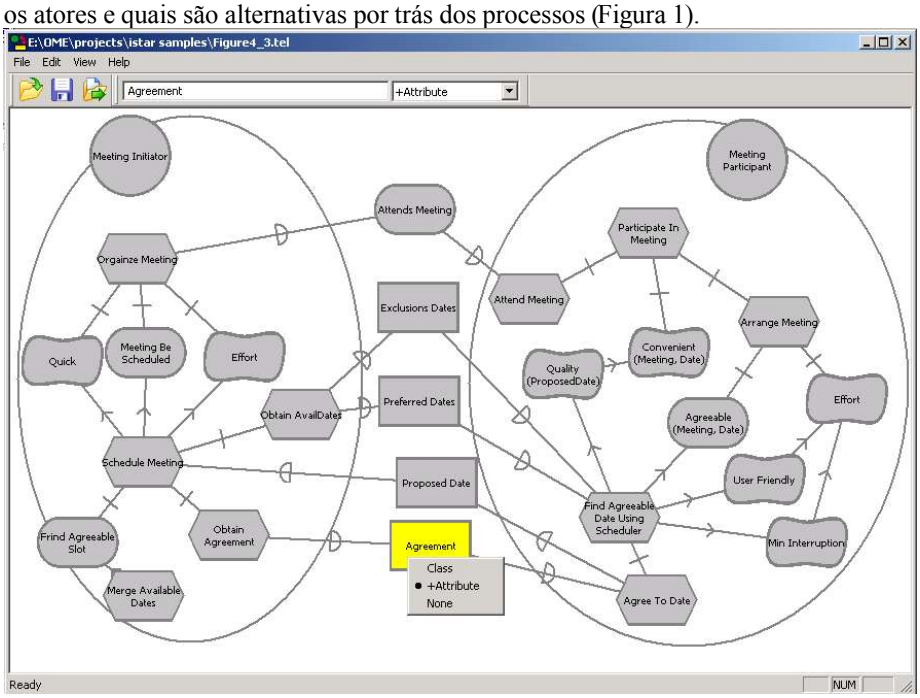


**Figura 1- Modelo SR**

No modelo SR são introduzidas duas classes de ligações: ligações meio-fim e decomposição de tarefas. Uma ligação meio-fim indica um relacionamento entre um fim e um meio para alcançar este fim. O meio é geralmente expresso na forma de uma tarefa, que incorpora a noção de como fazer alguma coisa.
Uma ligação de decomposição de tarefa é utilizada para descreve os componentes de uma tarefa.

## 3. Mapeamento de i* para UML

As diretrizes de mapeamento determinam as regras pelas quais a associação entre o modelo i* e o modelo UML é feita. A seguir veremos as diretrizes de mapeamento para transformar o modelo i* em um diagrama de classes UML. Um diagrama de classes UML modela a estrutura estática do sistema.

Em seguida, veremos o mapeamento do modelo i* para o diagrama de casos de uso. O diagrama de casos de uso mostra a interações entre casos de uso e os atores. Os casos de uso representam a funcionalidade do sistema. Os atores representam pessoas ou sistemas que interagem com o sistema sendo desenvolvido

## 3.1 Mapeamento de i* para Diagrama de Classes

No total são sete diretrizes: G1 – G7. Infelizmente os diagramas da UML sozinhos não são capazes de modelar todos os aspectos dos requisitos finais, pois não fornece especificações para as limitações dos sistemas (tais como invariantes, pré-condições, etc.). Para isto, adotamos a Object Constraint Language (OCL) [15]. OCL é uma linguagem textual, também parte do padrão OMG, que pode descrever de forma precisa limitações para modelos orientados a objetos. Abaixo, temos as diretrizes que realizam o mapeamento.

**Diretriz G1 -** Relacionada com o mapeamento de atores i*.

- Agentes, papéis ou posições no modelo i* podem ser mapeados em classes UML. Os relacionamentos ***IS-PART-OF, IS-A, OCCUPIES, COVERS*** e ***PLAYS*** são mapeados, respectivamente, como agregação de classes, generalização/especialização de classes, associação de classe denominada *OCCUPIES*, associação de classe denominada *COVERS* e uma associação de classe denominada *PLAYS*.

**Diretriz G2 -** Relacionada com o mapeamento de tarefas i*.

- **Diretriz G2.1** - Uma interface é criada com o nome formado pela junção dos nomes do *dependee* e *depender*. A tarefa definida no modelo SD (*Strategic Dependency*) pode ser mapeada como uma operação nesta interface, que é realizada pela classe que representa o *dependee* (aquele que realiza a tarefa).
- **Diretriz G2.2** - Uma tarefa definida no modelo SR (*Strategic Rationale*) pode ser mapeada como uma operação com visibilidade privada na classe que representa o ator a qual a tarefa pertence.

**Diretriz G3** - Os recursos i* podem ser mapeadas em classes UML.

- **Diretriz G3.1**- Os recursos definidos no modelo SD podem ser mapeados de duas maneiras:
    - um recurso pode ser mapeado em uma classe UML se ele possui características de um objeto, tal qual definido pelo paradigma de orientação à objeto. Um link de dependência entre as classes que representam o *dependee* e o *depender* é criado;
    - um recurso pode ser mapeado como um atributo com visibilidade pública na classe que representa o ator *dependee*, se esta dependência não está caracterizada como um objeto, tal qual definido no paradigma de orientação à objeto.
- **Diretriz G3.2** - Um recurso definido no modelo SR pode ser mapeado como um atributo com visibilidade privada na classe que representa o ator do qual o sub recurso pertence ou como uma classe independente.

**Diretriz G4 -** Relacionada com o mapeamento de objetivos i*.

- **Diretriz G4.1 -** Um objetivo definido no modelo SD pode ser mapeado como um atributo boleano com visibilidade privada na classe que representa o ator.

- **Diretriz G4.2 -** Um objetivo definido no modelo SR pode ser mapeado como um atributo boleano com visibilidade privada na classe que representa o ator a qual o objetivo pertence.

**Diretriz G5 -** Relacionada com o mapeamento de objetivos *soft* i*.

- **Diretriz G5.1 -** Um objetivo *soft* definido no modelo SD pode ser mapeado como um atributo enumerado com visibilidade privada na classe que representa o ator
- **Diretriz G5.2 -** Um objetivo *soft* definido no modelo SR pode ser mapeado como um atributo enumerado com visibilidade privada na classe que representa o ator a qual o objetivo pertence.

**Diretriz G6** - Relacionada com o mapeamento de relacionamentos de decomposição de tarefa no modelo i*.

A decomposição de tarefa no modelo SR é representada através de pré e pós-condições (definidas em OCL) da operação correspondente em UML. A pré-condição é a conjunção (conector AND OCL) das pré-condições das tarefas. A pós-condição é a conjunção (conector AND OCL) de todas: (i) pós-condições das tarefas, (ii) atributos boleanos dos recursos; (iii) atributos boleanos dos objetivos; (iv) atributos enumerados do objetivo *soft*.

**Diretriz G7** - Relacionada com o mapeamento de relacionamentos de meio-fim no modelo i*. Análise de meio-fim é representada através de disjunções de todos os possíveis meios de alcançar o fim.

- **Diretriz G7.1** - Se o fim é uma tarefa e os meios são tarefas então a disjunção das pós-condições das tarefas meio implica a pós-condição da tarefa fim.
- **Diretriz G7.2** - Se o meio é uma tarefa e o fim é um objetivo, um objetivo *soft* ou um recurso, o valor do fim é igual a disjunção das pós-condições das tarefas meio.
- **Diretriz G7.3** - Se o fim é um objetivo (*soft*) e os meios são objetivos (*soft*) então a disjunção dos valores dos meios implica o valor fim.

São criados também *links* de associações entre as classes do diagrama para representar os relacionamentos existentes no modelo i* (*depender, dependee* e *dependum*).
O diagrama de classes é apenas um dos diagramas da UML. Outro diagrama bastante utilizado e o diagrama de casos de uso.

## 3.2 Mapeamento de i* para Diagrama de Casos de Uso

A partir dos modelos organizacionais SD e SR deriva-se o diagrama de casos de uso para um sistema alvo da organização. Inicialmente, são descobertos os atores para o diagrama de Casos de Uso em UML e posteriormente, são descobertos os casos de uso para estes atores, bem como os fluxos principal e alternativos (cenários primário e secundários) dos casos de uso descobertos. A seguir apresentamos as diretrizes que possibilitam o mapeamento.

***1º Passo da Proposta: Descoberta de atores***

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

**Diretriz 2:** inicialmente, deve-se analisar se o ator em i* é externo ao sistema computacional pretendido. Caso o ator seja externo ao sistema, o mesmo é considerado candidato a ator em Casos de Uso;

**Diretriz 3:** deve-se garantir que o ator em i* é candidato a ator em Caso de Uso, através da seguinte análise:

> **SubDiretriz 3.1:** verificar se existe pelo menos uma dependência do ator analisado em relação ao ator em i* que representa o sistema computacional pretendido;

**Diretriz 4:** atores em i*, relacionados através do mecanismo IS-A 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 <<generalização>>.

***2º Passo da Proposta: Descoberta de Casos de Uso***.

**Diretriz 5:** para cada ator descoberto para o sistema pretendido no passo 1, devemos observar todas as suas dependências (dependum) do ponto de vista do ator como dependee, em relação ao ator sistema computacional pretendido (sistema computacional → dependum → ator), visando descobrir casos de uso para o ator analisado;

> **SubDiretriz 5.1:** deve-se avaliar as dependências do tipo objetivo associadas com o ator;
>
> **SubDiretriz 5.2:** deve-se avaliar as dependências do tipo tarefa, associadas com o ator;
>
> **SubDiretriz 5.3:** deve-se avaliar as dependências do tipo recurso, associadas com o ator;
>
> **SubDiretriz 5.4:** deve-se avaliar as dependências do tipo objetivo-soft, associadas com o ator;

**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 → dependum → sistema computacional).

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

***3º Passo da Proposta: Descoberta e descrição do fluxo principal e alternativos dos Casos de Uso.***

**Diretriz 8:** analisar cada ator e seus relacionamentos no Modelo SR para extrair informações que possam conduzir à descrição de fluxos principal e alternativos, bem como pré-condições e pós-condições dos casos de uso descobertos para o ator.

> **Diretriz 8.1:** analisar os sub-componentes em uma ligação de decomposição de tarefa em um possível mapeamento para passos na descrição do cenário primário (fluxo principal) de casos de uso.
>
> **Diretriz 8.2:** analisar ligações do tipo meio-fim em um possível mapeamento para passos alternativos na descrição de casos de uso.
>
> **Diretriz 8.3:** analisar os relacionamentos de dependências de sub-componentes no modelo de Razões Estratégicas em relação a outros atores do sistema. Estas dependências podem originar pré-condições e pós-condições para os casos de uso descobertos.

**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. Cada passo de um caso de uso deve ser analisado para verificar a possibilidade do mesmo ser refinado em um novo caso de uso.

**Diretriz 9.1:** Inicialmente deve-se averiguar qual é o objetivo que se quer atingir com a ação que o passo representa na descrição do caso de uso;
**Diretriz 9.2:** Descoberto o objetivo, deve-se averiguar a necessidade de decompor/refinar o objetivo para que o mesmo possa ser alcançado;
**Diretriz 9.3:** Um novo caso de uso será gerado a partir do objetivo analisado, se pudermos definir os passos que representam a necessidade de interações adicionais entre o ator e o sistema para se atingir o sub-objetivo desejado;
**Diretriz 9.4:** Adicionalmente, pode-se encontrar novos objetivos e cenários com base na observação dos obstáculos de objetivos já descobertos.

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

# 4. Ferramentas de Apoio

As ferramentas CASE (Computer Aided Software Engineering) fornecem suporte automatizado para muitos dos métodos existentes de análise e projeto de sistemas. Elas fornecem um meio ambiente que automatiza grande parte das tarefas repetitivas e que consomem bastante tempo do desenvolvedor. Existem várias ferramentas diferentes para os diferentes aspectos do desenvolvimento de um sistema. Contudo elas não fazem parte de um ambiente integrado. Apesar dos benefícios do uso de ferramentas CASE de forma individual, elas são mais úteis quando trabalham juntas e de uma forma integrada. Na próxima seção descrevemos a ferramenta OME que suporta a modelagem dos requisitos iniciais ou organizacionais.

## 4.1 Organization Modeling Environment - OME

OME (Organization Modeling Environment) é uma ferramenta de análise e modelagem orientada a objetivos e/ou agentes. Fornece ao usuário uma interface gráfica para o desenvolvimento de modelos. Atualmente, a ferramenta suporta dois frameworks de modelagem: i* e NFR.
A ferramenta foi projetada de modo que possibilita aos usuários estender a ferramanta para suporta novos frameworks. Além disso, fornece uma API Java que permite a criação de plugins para adicionar novas funcionalidades a ferramenta.
O OME3 é basicamente composto de duas partes: o núcleo (*kernel*) e os *plugins*. O *kernel* possui uma arquitetura em camadas, composta de três módulos principais, cada um composto de várias classes Java e/ou ferramentas externas. *Plugins* são classes externas que implementas funções específicas do *framework*. Os modelos gerados são salvos utilizando-se a sintaxe Telos [6].

### 4.2 Ferramenta XGOOD – eXtended Goal Into Object Oriented Development

A ferramenta XGOOD é uma versão aprimorada da ferramenta GOOD [3] (*Goal Into Object Oriented Development)*. Assim como o GOOD, o XGOOD trata-se também de uma ferramenta para auxiliar o mapeamento do modelo i* diretamente para o digrama de classes em UML, contendo, porém, algumas funcionalidades em relação à ferramenta proposta e desenvolvida anteriormente, tais como:

- adoção do formato XMI [7] (suportando tanto as versões 1.0 e 1.1) para representar os modelos - sendo um padrão adotado e reconhecido pela OMG [8] facilita o compartilhamento dos modelos entre diversas ferramentas CASE (Rational Rose, ArgoUML, Poseidon, MagicDraw UML, etc), não se restringindo à uma única ferramenta case, como em a anterior.
- Opção de selecionar as diretrizes de mapeamento adequadas, ou seja, quais elementos serão mapeados - possibilita ao usuário excluir certos elementos capturados segundo a técnica i*, mas que não são computacionais no modelo UML.
- Inclusão de novas diretrizes de mapeamento - o XGOOD suportará mais diretrizes relacionadas a novos elementos capturados pelos modelos i* (ex.: papel, posição e agente).

O XML Metadata Interchange (XMI) é um padrão que permite expressarmos objetos através do uso da eXtensible Markup Language (XML) [9], o formato universal para a representação de dados na World Wide Web. Seu principal objetivo é permitir o fácil intercâmbio de metadados entre ferramentas de modelagem (baseadas na UML – Unified Modeling Language) e repositórios de metadados (baseados no MOF – Meta Object Facility [10]). Utiliza o DTD (Document Type Definition) e, mais recentemente na versão 2.0, o XSD (XML Schema Definition) como meio de validar e manter a consistência dos modelos. Foi adotado em 1999 e atualmente se encontra na versão 2.0. O XMI integra três padrões:

1. XML - eXtensible Markup Language, um padrão do W3C standard;
2. UML - Unified Modeling Language, um padrão da OMG para modelagem orientada a objetos;
3. MOF - Meta Object Facility, um metamodelo e repositório de metadados da OMG;

O propósito do XMI é resolver certas questões que surgem com bastante freqüência no decorrer de um desenvolvimento de software:

1. **Combinar ferramentas em um ambiente heterogêneo**: o XMI resolve o problema de combinara várias ferramentas fornecendo um formato padrão flexível e facilmente interpretável para as informações.
2. **Trabalhando em um meio ambiente distribuído**: o uso do XMI facilita a troca de dados e modelos pela Internet. Por usar o formato XML, os dados podem ser disponibilizados como simples páginas em um servidor Web;

Na Figura 2 vemos o princípio de funcionamento da ferramenta. O arquivo "*.tel" contendo o modelo i* é lido pela ferramenta, é gerando então o arquivo XMI (contendo o modelo de classes UML). Adicionando ao arquivo XMI às informações do diagrama, temos o diagrama de classes completo, que pode ser lido por diversas ferramentas CASE.

A ferramenta foi desenvolvida em C++, através do ambiente de desenvolvimento Visual C++ 6.0, tendo-se um primeiro protótipo em teste. Funciona em Windows e atualmente já está implementando as diretrizes G1, G2, G3 e G4. Atualmente são suportadas as ferramentas CASE Rational Rose [11], Telelogic Tau [12] e MagicDraw [13].

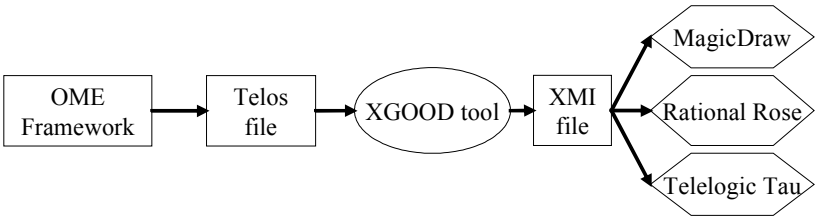


**Figura 2 - Princípio de funcionamento da nova ferramenta XGOOD**

Na Figura 1 vemos a tela da ferramenta (em sua versão mais recente) mostrando como é feita a seleção da diretriz adequada, para um recurso (*Agrement*) modelado segundo a técnica i*.

Muitos elementos do modelo i* possuem mais de uma opção de mapeamento. Além disso, alguns elementos não devem fazer parte do diagrama de classes. Em vista disso, é dada ao usuário a opção de selecionar a diretriz de mapeamento adequada para cada elemento. No total, estão disponíveis oitos opções possíveis para a seleção dos mapeamentos: **Class, Interface, +Attribute, +bool Attribute, -Method, -Attribute, -bool Attribute, None**. uso da ferramenta é bastante intuitivo é simples. Basicamente, existem três passos que o usuário deve executar:

- *File >> Open* – Abre um arquivo "*.tel" contendo o modelo i*;
- Selecionar a diretriz de mapeamento adequada para cada elemento (Figura 1).
- *File >> Export...* – Exportar o modelo i* para o modelo de classes correspondente (possibilidade de escolha entre os formatos "*.xmi" versão 1.0 ou versão 1.1).

Após a exportação é gerado um arquivo ".xml" que pode ser importado em várias ferramentas diferentes (Figura 3)

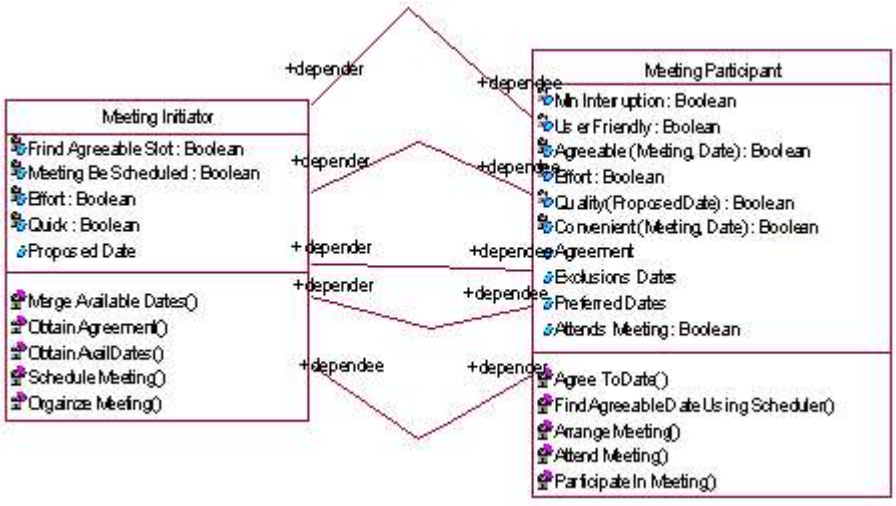


**Figura 3 - Diagrama classes gerado pela ferramenta.**

Além disso, é dada ao usuário a opção de salvar as diretrizes de mapeamento, para que ele não tenha que selecionar todas as diretrizes para cada elemento novamente:

- *File >> Save* – Salva as seleções de mapeamento (diretrizes de mapeamento) do modelo i*;

**4.2.1 Integração com a Ferramenta OME.** É possível estender a funcionalidade da ferramenta OME através de plugins. O módulo plugins é constituído por um conjunto de classes que se encarregam da maioria das manipulações que podemos realizar no modelo e na visão. Fornecem um meio fácil de estender a funcionalidade da ferramenta.
Cada plugin deve implementar a interface Java *OMEPlugin* e cada método do plugin deve implementar a interface Java *PluginMehtod*. Para que um plugin seja carregado com o modelo, é necessário que a classe compilada esteja no diretório OME3\program\plugins\.
Na Figura 4 podemos visualizar a integração coma ferramenta OME. São criados dois novos botões na barra de ferramentas.
Clicando no botão "XGOOD – Guidelines" ou "XGOOD – Export", o usuário pode selecionar as regras de mapeamento de cada elemento e gerar o arquivo ".xml" contendo o diagrama de classes.


**Figura 4 - Ferramenta XGOOD incorporada a ferramenta OME**

## 4.3 Ferramenta GOOSE

O GOOSE (*Goal Into Object Oriented Standard Extension*) é um protótipo de uma ferramenta que realiza a varredura das descrições dos requisitos organizacionais modelados em i* contendo os modelos SD e SR (coletados em um meio ambiente no qual o sistema a ser desenvolvido será usado) em um diagrama de casos de uso da UML. A ferramenta GOOSE lê e interpreta o arquivo ".tel", que contém o modelo i*, e gera o diagrama de casos de uso, através das diretrizes de mapeamento mostradas na seção 3.2.
A ferramenta GOOSE é uma extensão da ferramenta Rational Rose [11]. Para desenvolver a ferramenta foi utilizada a linguagem Rational Rose Scripting. Através desta linguagem é possível acessar as funcionalidades da ferramenta Rational Rose.
Após a instalação da ferramenta no Rational Rose (detalhes da ferramenta não são mostrados aqui, já que nossa intenção é meramente introduzir a ferramenta), é possível acessá-la através do menu principal do Rational Rose clicando em *Tools* e em seguida *GOOSE*.

# 5 Conclusões e Trabalhos Relacionados

Sugerimos nesse trabalho que a captura dos requisitos deve ser feita em diferentes níveis de abstração (indo da fase de captura de requisitos iniciais – relacionada com os requisitos organizacionais - até a fase final de captura – relacionada com os requisitos funcionais). Além disso, argumentamos que a UML sozinha não é adequada para lidar com os diferentes tipos de análises e razões que são necessárias durante a fase de captura de requisitos. Logo, faz-se necessário o uso de técnicas complementares: a técnica i*, para a captura dos requisitos iniciais e a modelagem do negócio; e a UML, para a captura dos requisitos finais.

Todavia, o uso de duas técnicas diferentes, sugere a necessidade de uma ferramenta para promover a integração dessas (através das diretrizes propostas), suportando o rastreamento e as mudanças entre os modelos.

A ferramenta e a técnica para realizar a integração dos requisitos iniciais e dos requisitos posteriores foram propostas em [3]. Posteriormente, as diretrizes foram aprimoradas e refinadas em [4]. Neste trabalho, buscamos dar suporte as novas diretrizes e suprir algumas deficiências da ferramenta anterior:

- Permitir a seleção dos elementos mapeados.
- Adoção do padrão XMI, que tem se tornado um padrão de fato para troca de dados entre ferramentas CASE;
- Independência de ferramentas CASE. A nova ferramenta pode ser utilizada independente da ferramenta utilizada para modelar o diagrama de classes;
- Suporte a mais elementos do modelo i* ;

A nova ferramenta, sem dúvida, representa uma evolução em relação a anterior.

Apresentamos também as diretrizes e a ferramenta para a geração do diagrama de casos de uso, mais um dos vários diagramas da UML. O próximo passo é tentar aprimorar as regras de mapeamento existentes e incorporar as diretrizes para geração do diagrama de classes e diagrama de casos de uso em única ferramenta.

## 6. Referências


[1] Yu, Eric, "Modelling Strategic Relationships for Process Reengineering", PhD thesis, University of Toronto, Departament of Computer Science, 1995.

[2] Rumbaugh, J., Jacobson, I. and Booch, G, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

[3] Filho, G. A. de A. C, "Ferramenta para o Suporte do Mapeamento da Modelagem Organizacional em i* para UML", tese de mestrado, Universidade Federal de Pernambuco, Centro de Informática, 2001.

[4] Alencar, F. M. R., Pedroza, F. P., Castro, J.F. B. and Amorim, R. C. O, "New Mechanisms for the Integration of Organizational Requirements and Object Oriented Modeling", *Proc. VI Workshop em Engenharia de Requisitos*, Piracicaba, Novembro de 2003, pp. 109-123.

[5] OME Organization Modelling Environment. Disponível em: http://www.cs.toronto.edu/km/ome/. Acessado em: 02 de nov. 2004

[6] Mylopoulos, J., Borgida, A., Telos. Representing Knowledge About Information Systems, ACM Transactions on Information Systems, vol. 8, out. 1990, pp. 325-362..

[7] Object Management Group OMG XML Metadata Interchange (XMI) Specification v1.2. Disponível em:

http://www.omg.org/technology/documents/formal/xmi.htm. Acessando em: 02 nov. 2004.

[8] Object Management Group (OMG). Disponível em: http://www.omg.org/. Acessado em: 02 nov. 2004.

[9] Extensible Markup Language (XML) 1.0. Disponível em: http://www.w3.org/TR/2004/REC-xml-20040204/. Acessado em: 18 out. 2004.

[10] Object Management Group: "MetaObjectFacility (MOF) SpecificationSpecification", 2002. Disponível em: http://www.omg.org/technology/documents/formal/mof.htm. Acessado em: 02 nov. 2004.

[11] Rational Rose. Disponível em: http://www.ibm.com/rational/. Acessado em: 02 nov. 2004.

[12] MagicDraw UML. Disponível em: http://www.magicdraw.com/. Acessado em: 02 nov. 2004.

[13] Telelogic Tau. . Disponível em: http://www.telelogic.com/. /. Acessado em: 02 nov. 2004.

[14] Williams, Al, *MFC Black Book*, Coriolis Group, Scottsdale, 1997.

[15] Object Constraint Language Specification. In: OMG Unified Modeling Language Specification. v. 1.5. Mar. 2003. p. 6-1 – 6-48. Disponível em: http://www.omg.org/technology/documents/formal/uml.htm. Acessado em: 02 nov. 2004.