Formalization of Mapping Rules from iStar to Class Diagram in UML

Josenildo Melo, Aêda Sousa, Celso Agra, José Júnior
Departamento de Engenharia da Computação
Universidade de Pernambuco
Recife, Brazil
{jasm, amcs, clasf, jmsj2}@ecomp.poli.br

Jaelson Castro[1], Fernanda Alencar[2]
[1]Centro de Informática
[2]Departamento de Eletrônica e Sistemas
Universidade Federal de Pernambuco
Recife, Brazil
jbc@cin.ufpe.br, fernandaalenc@gmail.com

***Resumo*—Due to tough competition, companies must build solutions (or maintain existed) quickly and effectively, covering the needs of customers without neglecting the quality requirements. To model these solutions, there are several patterns, and the UML (Unified Modeling Language) one of the most used. However, UML is not prepared to capture domain requirements for quality. To achieve this goal, models based on GORE (Goal-Oriented Requirements Engineering) are used as i* (iStar). This paper presents a formalization of i* mapping rules for class diagram in the context of MDD (Model-Driven Development), aiming to create more complete class diagram, where quality requirements are captured. An example is used to illustrate how these formalization rules can be applied.**

Keywords—Transformação entre modelos; i*; Model-Driven Development.

I. INTRODUÇÃO

As empresas precisam responder de forma rápida às novas demandas de mercado, construindo novas soluções ou realizando manutenções nos sistemas existentes. Logo, precisam manter seus processos atualizados e funcionando adequadamente, sem desprezar os requisitos de qualidade [1].

É necessário que ao final da fase de especificação de requisitos, todos os *stakeholders* estejam precisamente cientes das funcionalidades e do comportamento do sistema. Para isso, são propostos e utilizados diversos modelos, destacando-se os modelos da UML (*Unified Modeling Language*).

A UML é eficiente para especificar "o quê" um sistema faz e "como" ele faz algo, mas não o é para especificar o "porquê" ele faz [2]. Ela não é preparada para capturar os requisitos de domínio (*early requirements*), tais como: os porquês da necessidade do sistema pretendido; as alternativas que foram consideradas; as implicações para os vários *stakeholders*; e como direcionar os interesses e preocupações dos *stakeholders* [3]. Para minimizar essas problemáticas, surgiu a Engenharia de Requisitos Orientada a Metas (*Goal-Oriented Requirements Engineering - GORE*)[4]. Em abordagens orientadas a metas, o papel da Engenharia de Requisitos está relacionado à descoberta, à formulação, à análise e aos acordos de "o quê" é o problema a ser resolvido, ao "porquê" o problema deve ser resolvido, e a "quem" cabe a responsabilidade pela resolução desse problema [5]. A necessidade de se ter especificações de requisitos mais precisas que considerem as razões, motivações e intenções capturadas pela abordagem GORE conduziu à proposição inicial de regras de mapeamento de modelos i* (orientado a metas) para diagramas de classe [6] em UML, que, posteriormente, foram estendidas [3]. Desta feita, colaborando para que uma possível transformação automática entre modelos pudesse ser pensada. A formalização das regras de transformação entre esses modelos foi inicializada em [7] e fazendo-se necessária a formalização e teste de todas as regras, de forma a permitir transformações automáticas entre modelos (de i* para diagramas de classes) no contexto do desenvolvimento orientado a modelos (do inglês, *Model Driven Development* - MDD). Este trabalho pretende demonstrar essa formalização. Para isso, o presente artigo está estruturado da seguinte forma: a Seção 2 aborda o referencial teórico; a Seção 3 apresenta a formalização do mapeamento; a Seção 4 apresenta um estudo de caso; a Seção 5 apresenta os trabalhos relacionados; e a Seção 6 apresenta a conclusão.

II. REFERENCIAL TEÓRICO

A seguir, é apresentada uma fundamentação teórica sobre o desenvolvimento orientado a modelos, i*, linguagem de transformação de modelos, e as diretrizes de mapeamento.

A. Desenvolvimento Orientado a Modelos

Usar modelos para projetar sistemas complexos é um padrão em disciplinas tradicionais de engenharia. Não se imagina a construção de um edifício, de uma ponte ou de um carro, sem antes construir uma variedade de modelos e simulá-los. Modelos nos ajudam a entender um problema complexo (e suas possíveis soluções) através da abstração.

Atualmente, o desenvolvimento orientado a modelos (*Model-Driven Development - MDD*) [8] tem se mostrado uma tendência muito bem conceituada [9]. De fato, MDD visa acelerar o desenvolvimento de software, automatizando o desenvolvimento de produtos e empregando modelos reutilizáveis ou abstrações para visualizar o código (ou o domínio do problema). Ao usar os modelos, ou abstrações, podem-se descrever conceitos complexos de forma mais legível do que as linguagens de computação o fazem. Isto

978-1-4673-9272-3/15 $31.00 © 2015 IEEE
DOI 10.1109/SBES.2015.25

melhora a comunicação entre os *stakeholders*, pois os modelos são muitas vezes mais fáceis de entender do que o código [16].

De outro lado, a engenharia de requisitos orientada a metas (*Goal-Oriented Requirements Engineering - GORE*) tem sido reconhecida pela comunidade científica [11] [12] [13] como a melhor maneira para se capturar, em diferentes níveis de abstração, os vários objetivos que o sistema em questão deve atingir. GORE permite uma série de análises, como facilitar o raciocínio sobre o propósito de uma solução e demonstrar a contribuição da solução proposta com a necessidade real [13].

Segundo [11], uma meta é um objetivo que o sistema em questão deve alcançar. Assim, formulações de metas referem-se a propriedades do sistema que devem ser asseguradas. Metas também abrangem diferentes tipos de preocupações: preocupações funcionais (associada com os serviços a serem prestados) e não-funcionais (associadas com a qualidade do serviço, tais como segurança, precisão, desempenho, etc.). Modelos de metas podem ser analisados para demonstrar quais metas realizam outras metas, e quais metas geram conflitos com outras metas [13].

A identificação das metas (*goals*) não é necessariamente uma tarefa fácil. Às vezes, eles são explicitamente estabelecidos pelos *stakeholders* ou em material preliminar disponível para os engenheiros de requisitos. Na maioria das vezes, eles estão implícitos, de modo que uma elicitação de metas precisa ser realizada. A análise preliminar do sistema atual é uma fonte importante para a identificação de metas. Tal análise resulta normalmente numa lista de problemas e deficiências que podem ser formulados com precisão. Negar essas informações produz uma primeira lista de metas a serem alcançadas pelo sistema a ser desenvolvido.

Uma vez que um conjunto de metas e requisitos é obtido e validado com as partes interessadas, muitos outros objetivos podem ser identificados pelo refinamento e abstração, apenas perguntando COMO e POR QUE aos objetivos/requisitos já disponíveis [11].

Dentre as abordagens GORE existentes encontra-se o i* [20], atualmente uma das modelagens mais difundidas [14] [15]. Ela tem sido aplicada para modelagem de organizações, processos de negócio e requisitos do sistema, entre outros [11].

*B. i**

A técnica i* [20] fornece a compreensão e os objetivos do ambiente organizacional. O i* oferece um *framework* de modelagem que se concentra nos relacionamentos estratégicos entre atores. O termo "ator" foi utilizado para referir genericamente a qualquer unidade para a qual poderiam ser atribuídas dependências intencionais (quando dois ou mais atores compartilham algum elemento de dependência, como uma tarefa ou recurso). Um ator intencional não se limita a realizar atividades e produzir entidades, mas têm motivações, intenções e justificativas por trás suas ações [16]. Um ator é estratégico quando não está apenas focado em atender ao seu objetivo imediato, mas também está preocupado com as implicações em longo prazo das suas relações estruturais com outros atores [3]. Quando tentamos entender uma organização utilizando técnicas de modelagem padronizadas (como UML), vemos que ela não é capaz de expressar as razões (os "porquês") do processo (motivações, intenções e razões). A ontologia do i* atende a alguns desses conceitos mais avançados. Os participantes do ambiente organizacional são atores com propriedades intencionais, tais como objetivos, crenças, habilidades e compromissos. Esses atores dependem uns dos outros, a fim de terem seus objetivos e suas tarefas executadas [3].

O i* é composto por dois modelos: o Modelo de Dependência Estratégica (SD) e o Modelo de Razão Estratégica (SR).

1) *Modelo de Dependência Estratégica (Strategic Dependence Model – SD)*: Este modelo descreve as relações de dependências externas entre os atores da organização. Graficamente, é representado através de um conjunto de nós e ligações entre eles, onde os nós representam os atores e cada ligação indica uma dependência entre dois atores: o ator que depende que outro ator satisfaça a dependência (*depender*) e o ator de quem se depende, responsável por satisfazer a dependência (*dependee*) [6], [16], [7].

Neste modelo, distinguem-se quatro tipos de dependências [3]:

- Objetivos: são afirmações;
- Recursos: estão relacionados com a noção de entidades;
- Tarefas: dizem respeito às atividades;
- Objetivos-Soft: associados com a noção de requisitos não funcionais.

Em i*, nós também podemos modelar graus diferentes de dependência entre os atores relevantes. Para modelar as sub-unidades de um ator complexo, nós também podemos classificar atores em três tipos diferentes [3]:

- um agente é um ator com uma manifestação física concreta (uma pessoa ou um sistema);
- um papel é uma caracterização abstrata do comportamento de um ator dentro de algum contexto, domínio ou esforço especializado;
- uma posição é uma abstração intermediária entre um agente e um papel. É um conjunto de papéis normalmente executados por um agente.

2) *Modelo de Razão Estratégica (Strategic Rational Model – SR)*: O modelo SR fornece um nível mais detalhado de modelagem por olhar "dentro" dos atores para modelar as relações intencionais internas. Esse modelo é utilizado para [3]:

- descrever os interesses, preocupações e motivações dos processos participantes;
- permitir a avaliação das possíveis alternativas na definição do processo;

- pesquisa mais detalhada das razões existentes por trás das dependências entre os vários atores.

Dois novos tipos de relacionamento são incorporados nesse modelo: *means-end*, que sugere que poderia haver outros meios de alcançar o objetivo, e *task-decomposition*, que descreve o que deve ser feito a fim de executar uma determinada tarefa.

C. Linguagem de Transformação de Modelos

As transformações de modelos são um passo necessário em cada fase da abordagem orientada a modelos. Para isso, linguagens são utilizadas para definir regras que especificam a conformidade entre os modelos, de tal forma que um modelo possa ser especificado a partir de outro modelo. As linguagens possuem uma estrutura específica visando facilitar o desenvolvimento dessas regras. Para este trabalho, foi utilizada a linguagem ATL (*Atlas Transformation Language*) [19].

D. Diretrizes de Mapeamento

As regras de mapeamento propostas por [3] definem as características do i* que podem ser implementadas em diagramas de classes. As diretrizes são demonstradas na Tabela 1. Não faz parte do escopo desse estudo discutir essas regras, e sim a formalização delas.

TABELA I. DIRETRIZES DE MAPEAMENTO

Número	i*	UML
1.1	Agentes, papéis ou posições	Classe.
1.2	Relacionamento ***IS-PART-OF*** entre agentes, papéis ou posições.	Agregação de classe.
1.3	Relacionamento ***IS-A*** entre agentes, papéis ou posições.	Generalização/especialização de classe.
1.4	Relacionamento ***OCCUPIES*** entre um agente e uma posição.	Associação de classe chamada OCCUPIES.
1.5	Relacionamento ***COVERS*** entre uma posição e um papel.	Associação de classe chamada COVERS.
1.6	Relacionamento ***PLAYS*** entre um agente e um papel.	Associação de classe chamada PLAYS.
2.1	Tarefas definidas no modelo SD.	Métodos com visibilidade pública.
2.2	Tarefas definidas no modelo SR.	Métodos com visibilidade privada.
3.1	Recursos definidos no modelo SD.	Classe se essa dependência tem a característica de um objeto.
3.1	Recursos definidos no modelo SD.	Atributo com visibilidade privada em uma classe que representa o ator *dependee*, se essa dependência não pode ser caracterizada como um objeto.
3.2	Recursos (sub-recursos) definidos no modelo SR.	Atributo com visibilidade privada na classe que representa o ator em que o sub-recurso pertence (se esse sub-recurso não pode ser entendido como um objeto).
3.2	Recursos (sub-recursos) definidos no modelo SR.	Um classe independente, caso contrário.
4.1	Objetivos (ou Objetivos-Soft) no modelo SD.	Atributo do tipo Enum com visibilidade pública na classe que representa o *dependee*.
4.2	Objetivos (ou Objetivos-Soft) no modelo SR.	Atributo do tipo Enum com visibilidade pública na classe que representa o ator em que pertence o sub-objetivo.
5	Decomposição de tarefas.	Representada pelas pré e pós-condições (expressas em OCL) da operação pUML correspondente.
6.1	Objetivos (ou Objetivos-Soft) - Objetivos (ou Objetivos-Soft) .	A disjunção (separação) dos valores meios implica nos valores finais.
6.2	Objetivos (ou Objetivos-Soft)-Tarefa, Recurso-Tarefa .	A pós-condição dos meios, que implica o valor do fim.
6.3	Tarefa-Tarefa.	A disjunção da pós-condição dos meios implica na pós-condição do fim.

III. FORMALIZAÇÃO DO MAPEAMENTO

A. Processo

Tendo como base as regras apresentadas na Tabela 1, foi proposta a formalização do mapeamento dessas regras. O processo é exibido na Figura 1. Ele se inicia com a entrada dos dados obtidos através da ferramenta iStarTool [26]. Nesta ferramenta, os elementos i* são modelados e a ferramenta gera os elementos UML correspondentes em um arquivo XMI. No segundo passo, este arquivo XMI é importado pela ferramenta Eclipse (adaptada com um *plugin* da linguagem ATL) e regras descritas em ATL são aplicadas, gerando um novo arquivo XMI (o modelo de saída). No último passo, esse modelo de saída deve ser importado por uma ferramenta CASE, para que o diagrama de classes possa ser visualizado.

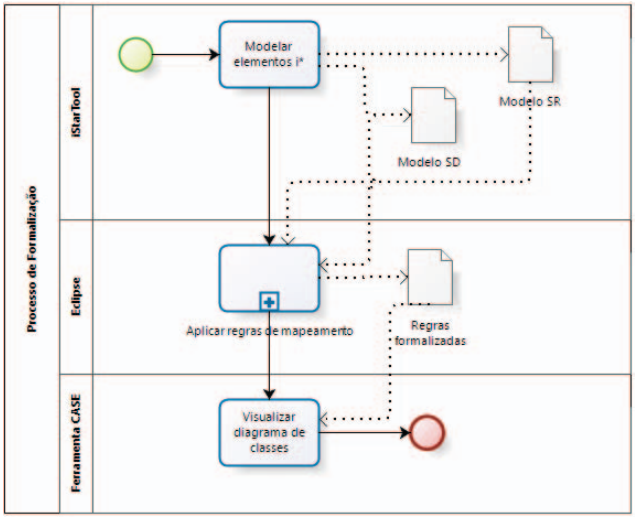

Fig. 1. Processo de formalização.

A atividade "Aplicar regras de mapeamento" é um sub-processo, que é exibido na Figura 2. Esse processo se inicia com a configuração do ambiente Eclipse, onde é executada a transformação dos modelos, e as demais atividades referem-se à formalização das diretrizes.

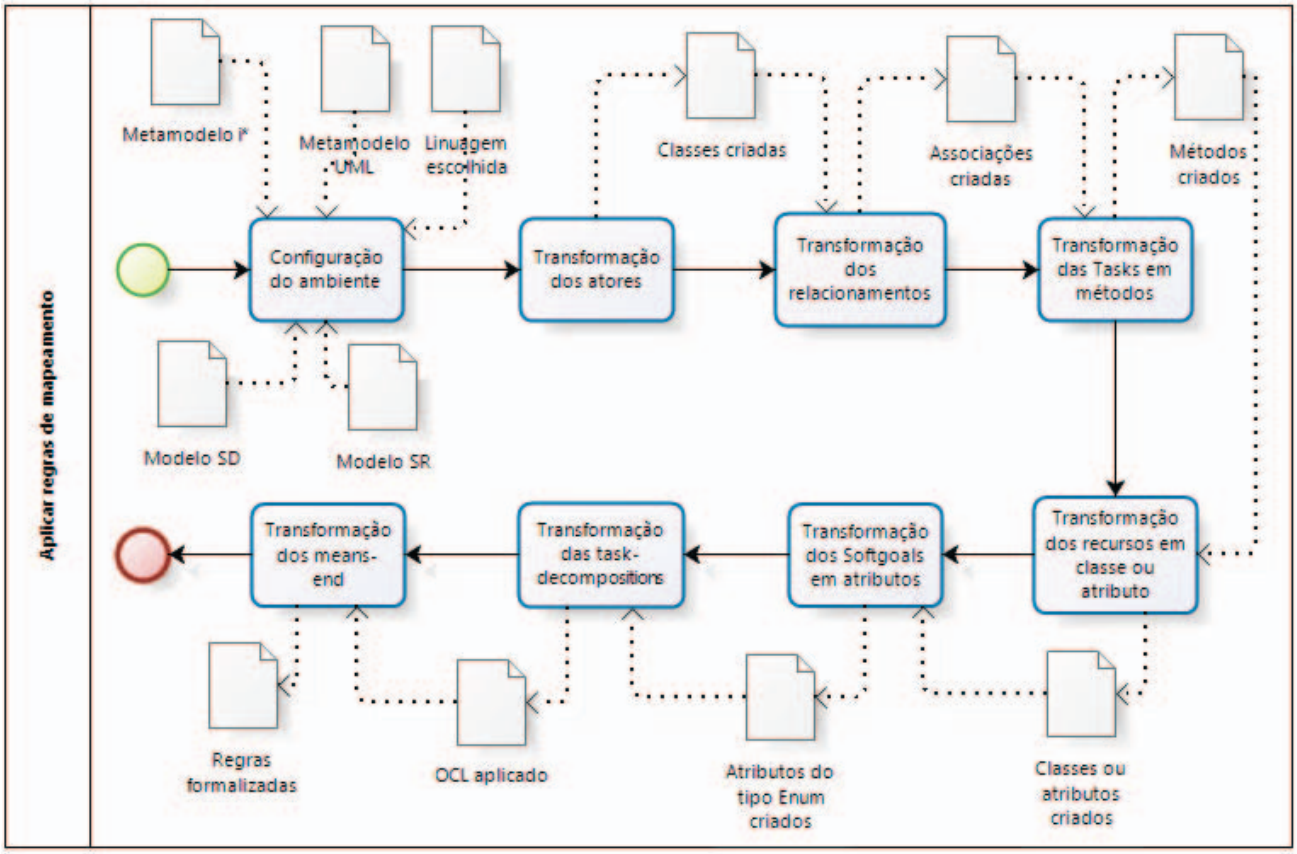

Figura 2. Sub-processo "Aplicar regras de mapeamento".

B. *Metamodelos*

A automação do processo de mapeamento é feita utilizando a linguagem de transformação ATL, que opera sob os modelos presentes na arquitetura de metamodelagem MOF. A Figura 3 apresenta a arquitetura MOF para esse estudo.

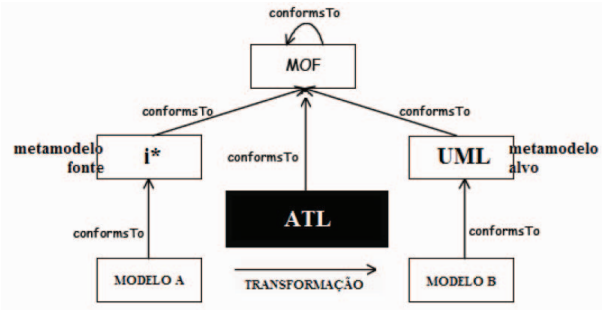

Fig. 3. Arquitetura MOF para o mapeamento (Fonte: [7], p. 50).

Para que sejam efetuadas as transformações, faz-se necessário o uso de metamodelos. Um metamodelo é um modelo que define os conceitos e relacionamentos que podem ser criados em um modelo. Portanto, um modelo sempre está de acordo com um metamodelo. Os metamodelos usados neste trabalho foram definidos usando Ecore[1], uma linguagem para construção de metamodelos baseada no framework EMF[2]. Os metamodelos utilizados neste trabalho podem ser visualizados em [25].

C. *Abordagem de Transformação*

Demonstraremos agora como foram feitas as formalizações. O código ATL pode ser visualizado em [25].

De acordo com a Tabela 1, a diretriz **1.1** menciona que um ator (seja ele um agente, papel ou posição) deve ser transformado em uma classe. Tal transformação é exibida na Figura 4, que mostra os elementos antes (em i*) e depois da transformação (em UML).

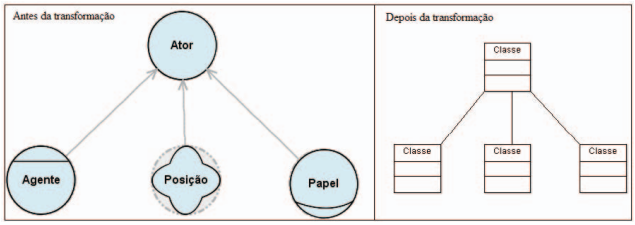

Fig. 4. Transformação de atores em classes.

[1] http://www.eclipse.org/modeling/emft/?project=ecoretools
[2] http://www.eclipse.org/modeling/emf/

A diretriz **1.2** determina que o relacionamento IS-PART-OF em i* seja transformado em uma agregação de classes (quando a existência de uma classe faz sentido, mesmo sem a existência da outra classe). Tal transformação é exibida na Figura 5, que mostra um relacionamento entre um agente e uma posição e a transformação em uma agregação. A formalização desta transformação foi feita em conjunto com a transformação das associações, e será explicada mais adiante.

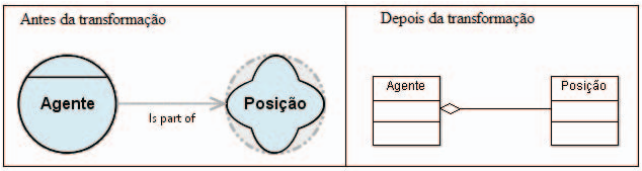

Fig. 5. Transformação do relacionamento IS-PART-OF em agregação.

A diretriz **1.3** determina que um relacionamento IS-A entre agentes, papéis ou posições deve ser transformado em uma generalização de classes (quando se cria uma superclasse encapsulando estrutura e/ou comportamento comum a várias subclasses, semelhante à ideia de herança em orientação a objetos). Essa transformação é exibida na Figura 6, onde é exibido um relacionamento IS-A entre dois agentes e o resultado após a transformação.

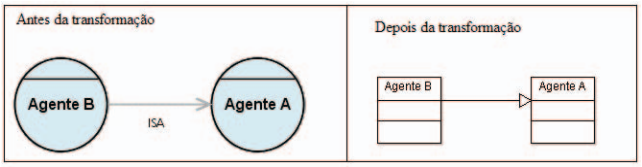

Fig. 6. Transformação do relacionamento IS-A em generalização.

As diretrizes **1.4**, **1.5** e **1.6** determinam que relacionamentos *occupies*, *covers* e *plays* sejam transformados em associações de classes, e cada associação deve ter o nome do relacionamento correspondente. Essa transformação é exibida na Figura 7, que mostra cada relacionamento com a sua associação correspondente. A formalização da diretriz **1.2** (que determina que o relacionamento IS-PART-OF seja transformado em uma agregação de classes) foi feita em conjunto com essa formalização dos relacionamentos porque, na linguagem ATL, o que diferencia uma associação de uma agregação é apenas o valor de um atributo chamado *aggregation*: caso o valor seja a *string* "*shared*", significa que o atributo refere-se a uma agregação. Caso o valor seja a string "*none*", o atributo refere-se a uma associação.

A diretriz **2.1** afirma que as tarefas (*tasks*) definidas no modelo SD devem se tornar métodos com visibilidade pública, enquanto a diretriz **2.2** afirma que as tarefas definidas no modelo SR devem se tornar métodos com visibilidade privada. A Figura 8 mostra esses elementos antes e depois da transformação. Para realizar essa formalização, faz-se necessário detectar quando um elemento faz parte de um modelo SD ou SR. A ferramenta iStarTool soluciona esse problema, pois ao criar o código XMI do modelo SR, os elementos (que estão dentro de uma fronteira de um ator) são criados dentro da *tag* referente a esse ator.

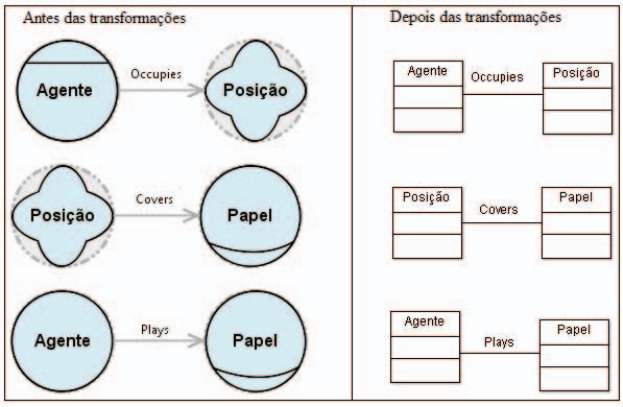

Fig. 7. Transformação dos relacionamentos *occupies*, *covers* e *plays* em associações.

A diretriz **3.1** refere-se a recursos (*resources*) feitos no modelo SD, enquanto a diretriz **3.2** refere-se a recursos feitos no modelo SR. A **3.1** determina que o recurso deve ser transformado em uma classe caso essa dependência tenha a característica de um objeto. Se a dependência não tiver essa característica, deve ser criado um atributo com visibilidade privada na classe que representa o ator *dependee*. Já a diretriz **3.2** diz que os recursos devem ser transformados em atributos com visibilidade privada na classe que representa o ator que o sub recurso depende (se esse sub recurso não pode ser entendido como um objeto. Em caso contrário, deve ser criada uma classe). A ferramenta iStarTool não oferece uma opção para indicar se um elemento (neste caso, um recurso) tem a característica de um objeto. Para atingir nosso objetivo, é necessário inserir manualmente uma *tag* dentro do arquivo XMI. Definimos que seria criada uma *tag* chamada *object* que deverá ter o valor "1" para indicar que esse elemento possui as características de um objeto. Caso haja qualquer outro valor nessa *tag* (ou se essa *tag* estiver vazia), então o elemento não possui a característica.

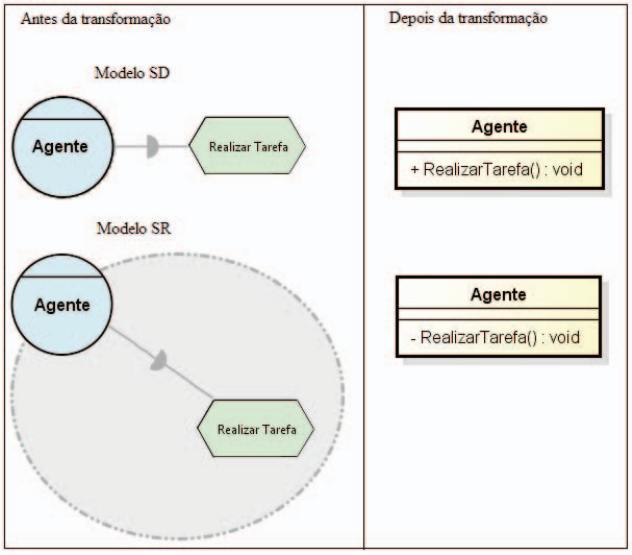

Fig. 8. Transformação de tarefas em métodos.

A diretriz **4.1** afirma que um objetivo (*goal*) ou objetivo-soft (*softgoal*) no modelo SD deve ser transformado em um atributo do tipo *enum* com visibilidade pública na classe que representa o *dependee*. A diretriz **4.2** afirma que o elemento (objetivo ou objetivo-soft) deve ser transformado em um atributo com visibilidade pública na classe que representa o ator em que o sub objetivo pertence. A Figura 9 mostra essa transformação. Note que as definições no diagrama de classes para o resultado dessas transformações são iguais.

A diretriz **5** é relacionada a decomposição de tarefas (*task decomposition*), que descrevem o que deve ser feito para realizar uma determinada tarefa. Segundo a diretriz, a decomposição, ao ser transformada, deve ser representada por pré e pós-condições (expressas em OCL) da operação UML correspondente. Essa representação será feita conectando uma nota (um comentário) em uma classe. Nessa nota será colocada a instrução OCL. A Figura 10 demonstra essa transformação.

As diretrizes **6.1**, **6.2** e **6.3** referem-se aos meios-fim (*means-end*), que sugere o que poderiam ser outros meios para alcançar o objetivo (ou seja, oferece alternativas). Elas especificam que:

- **6.1**. Relacionamentos entre objetivos (ou objetivos-soft) e objetivos (ou objetivos-soft): a disjunção (separação) dos valores meios implica em valor final;
- **6.2.** Relacionamentos entre objetivo (ou objetivo-sof) e tarefa, ou entre recurso e tarefa: a pós condição da tarefa (meios) implica o valor final;
- **6.3.** Relacionamentos entre tarefa e tarefa: a disjunção da pós-condição dos meios implica a pós-condição do fim.

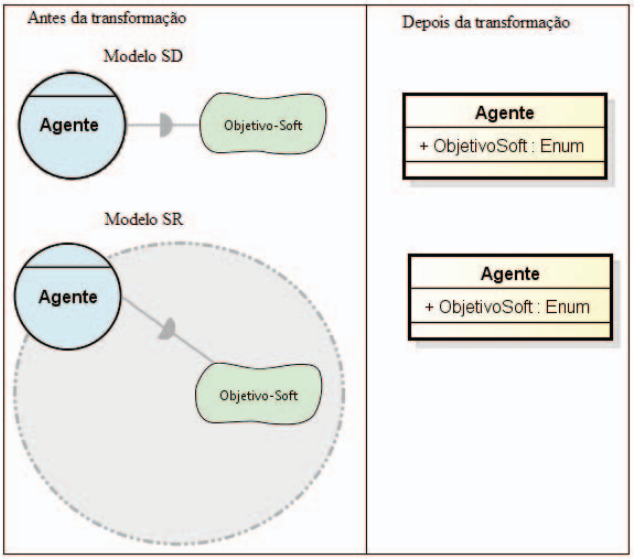

Fig. 9. Transformação dos objetivos-soft.

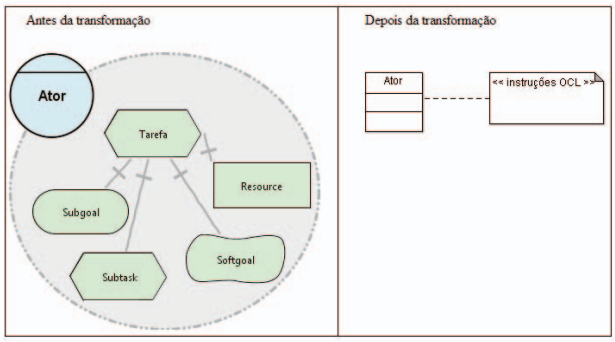

Fig. 10. Transformação da decomposição de tarefas (*task decomposition*).

Essas formalizações foram feitas de forma semelhante à diretriz **5**, onde foi criada uma nota (um comentário) conectada à classe do ator, e nessa nota foi colocada a instrução OCL. A Figura 11 mostra essa transformação.

IV. Estudo de Caso

Para ilustrar o uso da transformação entre modelos, o estudo de caso escolhido é um sistema definido por [3] e representado por uma loja de CDs especializada que permite realizar pedidos através de um catálogo *online*. Suponha uma situação em que um cliente deseja comprar CDs e vai a uma loja especializada. Se o cliente não pode encontrar seu título preferido, a loja pode notificá-lo após a chegada de novas unidades. A loja decidiu melhorar os seus serviços, com a encomenda de um novo sistema de software (SmartCD) para lidar com pedidos, bem como proporcionar um catálogo *online*. Na Figura 12, temos o modelo inicial *Strategic Dependence* (SD) do estudo de caso da loja de CD.

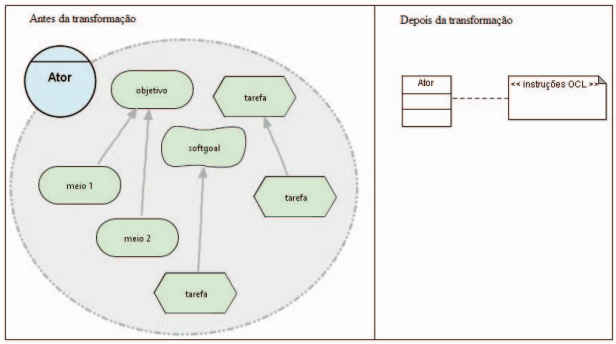

Fig. 11. Transformação dos meios-fim (means-*end*).

Nesta fase inicial de captura de requisitos nós identificamos três posições: *Client*, *Store Management* e *SmartCD*. Este último ator corresponde tanto ao sistema de software a ser desenvolvido, como ao tratamento de encomendas, às notificações das chegadas de CD e o fornecimento do catálogo online. As dependências podem ser encontradas na Figura 12.

Na Figura 13, nós concentramos nossa especificação sobre a posição SmartCD. Este é o sistema de informação que será desenvolvido no futuro. Como podemos ver, nós usamos cinco tipos de relacionamentos (*occupies*, *covers*, *play*, *is -part-of* e

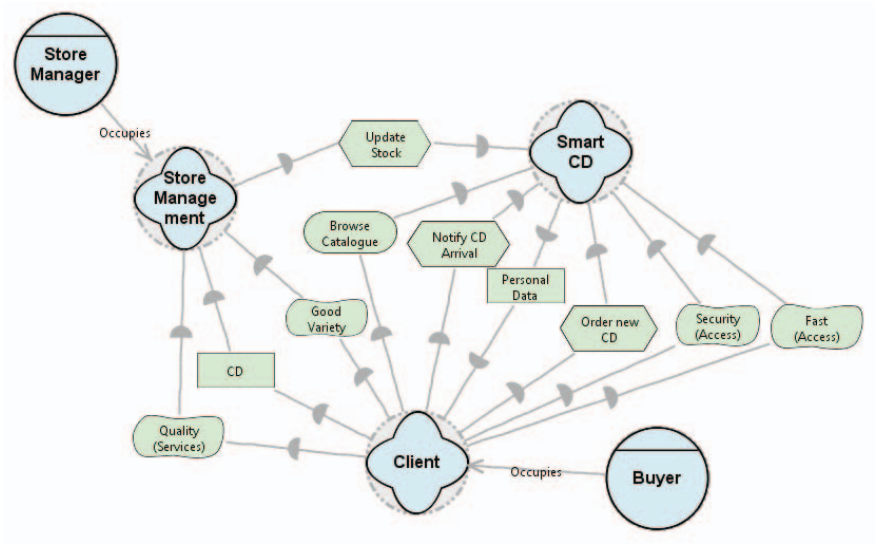

Fig. 12. O modelo *Strategic Dependence (SD)*.

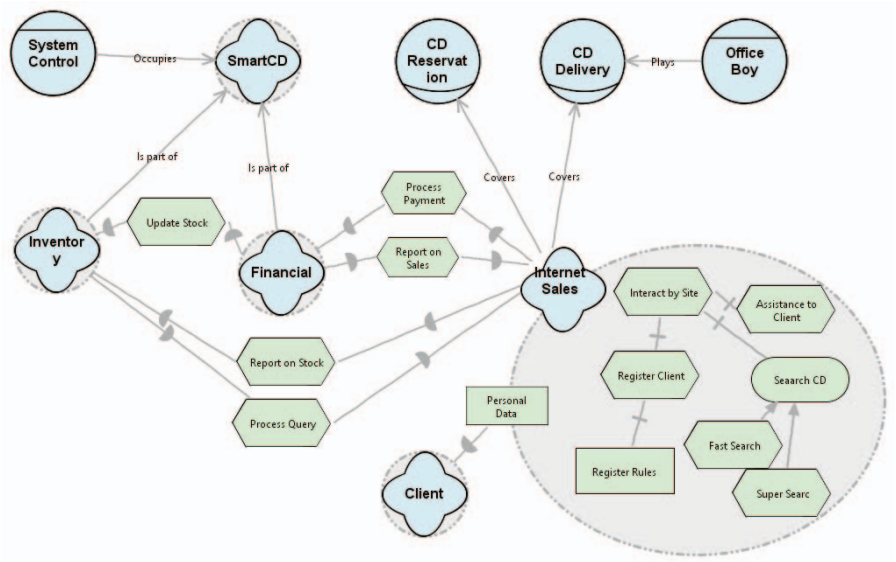

Fig. 13. O modelo *Strategic Rational (SR)* do SmartCD.

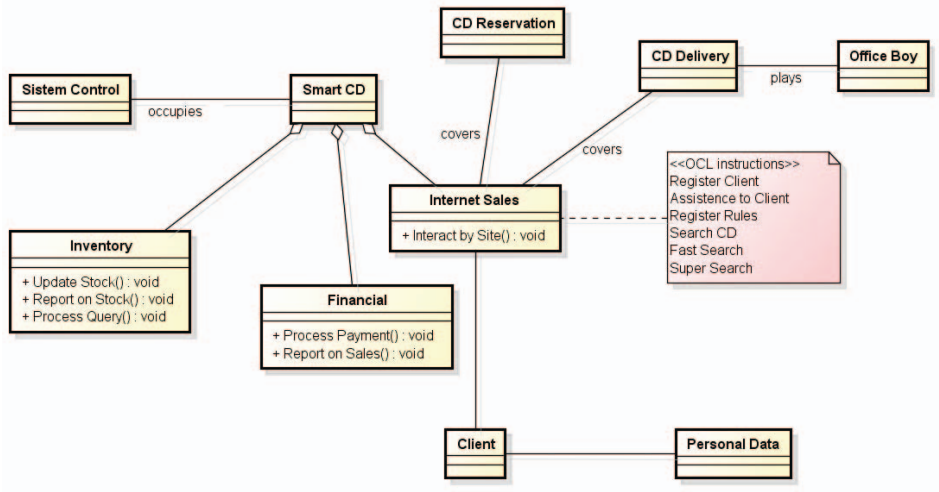

Figura 14. Diagrama de classes do sistema SmartCD.

is-a). O primeiro é, respectivamente, entre um agente (*SystemControl*) e uma posição (*SmartCD*). O segundo é entre uma posição (*InternetSales*, o único ator que possui fronteira) e um conjunto de papéis (*CD Reservation* e *Cd Delivery*). O terceiro é entre um agente (*Office Boy*) e um papel (*CD Delivery*). Papéis, posições e agentes podem ter subpartes. Isso é expresso pelo quarto relacionamento, "IS-PART-OF". Assim, a posição *SmartCD* consiste de *InternetSales*, *Inventory* e *Financial*. O quinto relacionamento, "IS-A", representa uma generalização / especialização conceitual entre agentes, posições ou papéis. Esta construção não é usada na Figura 13. Após a transformação entre os modelos, foi gerado o diagrama de classes que pode ser visualizado na Figura 14. Na transformação, todos os agentes, papéis e posições são transformados em classes (diretriz 1). Os relacionamentos *is-part-of* são transformados em agregações entre a classe *SmartCD* e as classes *Inventory*, *Financial* e *Internet Sales* (diretriz 2). Os relacionamentos *occupies*, *covers* e *plays* são transformados em associações (com o mesmo nome) entre as classes correspondentes, conforme as diretrizes 1.4, 1.5 e 1.5.

As tarefas existentes foram transformadas em métodos com visibilidade privada (por ser um modelo SR) nas classes correspondentes, conforme as diretrizes do grupo 2. O recurso *Personal Data* foi transformado em uma classe, por conter características de um objeto, conforme determina as diretrizes do grupo 3.

A posição *Internet Sales*, como citado anteriormente, é o único ator que possui fronteira. Dentro dela há uma tarefa (*Interact by Site*) e alguns *links* de decomposição. Um desses *links* pertence a um objetivo (*goal*) chamado *Search CD*, e esse objetivo possui dois *links* meios-fim (*Fast Search* e *Super Search*). Esses *links* de decomposição e de meios-fim foram escritos em um comentário associado à classe *Internet Sales*, conforme as diretrizes 5, 6.1, 6.2 e 6.3.

V. Trabalhos Relacionados

Existem diversas pesquisas que lidam com transformação de modelos. Dentre os estudos que se aproximam do nosso, citamos alguns a seguir.

Richard [22] propôs um método formal para descrição de requisitos, e a análise de requisitos foi realizada utilizando KAOS (outro método de análise de requisitos orientado a metas, similar à GORE utilizada neste estudo).

Sun [23] propôs uma técnica de transformação de modelos em domínio orientado a serviços. Uma transformação entre modelos de metas e modelos de projeto foi realizada, no entanto, ao invés de diagramas de classe UML, Sun usou modelos de processos como modelos-alvo.

Lucena [24] propôs um método para transformar modelos i* em modelos arquitetônicos Acme. O modelo-fonte e o modelo-alvo são diferentes dos nossos. Além disso, Lucena propôs um método para melhorar as técnicas de modelação, a fim de obter um modelo-alvo que está pronto para ser transformado. Em nossa abordagem, o modelo-alvo já é o diagrama de classes.

Aguilar [27] propõe uma abordagem para usar i* para melhorar a manutenção de aplicações web. Esta abordagem foi feita utilizando a linguagem QVT. No entanto, as diretrizes precisam ser finalizadas.

Alencar [28] propõe diretrizes para gerar, a partir de um modelo de requisitos i*, um modelo conceitual que é usado como entrada de um processo MDD para geração de produtos de software. No entanto, essas diretrizes ainda não foram formalizadas.

Filho [29] propõe um conjunto de relações de rastreabilidade entre os modelos organizacionais expressos em i* e casos de uso e diagramas de classe de sistemas de software expresso em UML, e uma abordagem para gerá-los automaticamente. Nossa abordagem é diferente, pois propõe uma formalização no contexto de MDD.

VI. Conclusão

Elicitação de requisitos é essencial para que um sistema seja desenvolvido com todas as características e funcionalidades necessárias, e a aplicação de modelos podem ajudar a visualizar o sistema antes que seja iniciada a sua construção. Diversos modelos existem, sendo a UML um dos mais utilizados. Entretanto, a UML não captura todas as necessidades do sistema, indicando "como" um sistema deve ser feito e não "por quê" deve ser feito. Dentre as abordagens que se preocupam com as necessidades do sistema, destaca-se a i*.

O objetivo deste trabalho foi a criação de diagramas de classes mais completos, abrangendo as características do i* e baseando-se em regras criadas anteriormente.

Como trabalhos futuros, está prevista a criação de uma ferramenta para automatizar todo o processo de transformação.

Referências

[1] T. C. Pereira, F.M.R. Alencar, J.R.F. Silva, and J.F.B. Castro, "Requisitos Não-Funcionais em Modelos de Processos de Negócio: Uma Revisão Sistemática", Proceedings do IX Simpósio Bras. Sist. Informação, 2013.

[2] G.A.A.C. Filho, A. Zisman, G. Spanoudakis, "A Traceability Approach for i* and UML Models", 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, 2003.

[3] F.M.R. Alencar, F. Pedroza, J.F.B. Castro, R.C.O. Amorim, "New Mechanisms for the Integration of Organizational Requirements and Object Oriented Modeling", VI WORKSHOP DE ENGENHARIA DE REQUISITOS (2003), Proceedings do Workshop de Engenharia de Requisitos, Piracicaba-SP , 2003.

[4] J.F. Castro, F.M.R. Alencar, G.A.C. Filho, J. Mylopoulos, "Integrating organizational requirements and object oriented modeling", Proceedings do Fifth IEEE International Symposium on Requirements Engineering, pp. 146-153, 2001.

[5] J. Pimentel, M. Lucena, J. Castro, C. Silva, E. Santos, F. Alencar, "Deriving software architectural models from requirements models for adaptive systems: the STREAM-A approach".

[6] F.M.R. Alencar, "Mapeando a Modelagem Organizacional em Especificações Precisas", PhD Thesis, Universidade Federal de Pernambuco, Recife, Brazil, 1999.

[7] U.A. Oliveira, "Uma metodologia DSDM para Integração de Requisitos organizacionais e UML no Desenvolvimento de Sistemas de Agentes

Utilizando i* (i-star)", Monografia, Universidade Federal de Sergipe, São Cristóvão-SE, Brazil, 2009.

[8] B. Selic, "The pragmatics of model-driven development", IEEE Software, vol. 20, no. 5, pp. 19–25, Sep-2003.

[9] J.M. Vara, V.A. Bollati, Á. Jiménez, E. Marcos, "Dealing with Traceability in the MDD of Model Transformations", vol. 40, no. 6, pp. 555–583 (2014).

[10] S. Teppola, P. Parviainen, J. Takalo, "Challenges in Deployment of Model Driven Development", Fourth International Conference on Software Engineering Advances, pp. 15–20, 2009.

[11] A. Van Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour", Proceedings do Fifth IEEE International Symposium on Requirements Engineering, pp. 249–262, 2001.

[12] G. Regev, A. Wegmann, "Where do Goals Come from: the Underlying Principles of Goal-Oriented Requirements Engineering The Use of Goals in GORE methods", 13th IEEE International Conference on Requirements Engineering (RE'05), pp. 353–362, 2005.

[13] D. Quartel, W. Engelsman, H. Jonkers, M. Van Sinderen, "A Goal-Oriented Requirements Modelling Language for Enterprise Architecture", 2009 IEEE International Enterprise Distributed Object Computing Conference, pp. 3-13, 2009.

[14] R. Monteiro, J. Araújo, V. Amaral, M. Goulao, P. Patrício, "Model-Driven Development for Requirements Engineering: The Case of Goal-Oriented Ap-proaches", 2012 Eighth International Conference on the Quality of Information and Communications Technology, pp. 75–84, 2012.

[15] E. Yu, D. Amyot, G. Mussbacher, X. Franch, J. Castro, "Practical Applications of i* in Industry : The State of the Art", 21st IEEE International Requirements Engineering Conference, pp. 366–367, 2013.

[16] E. Yu, "Why Agent-Oriented Requirements Engineering", Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality, Pisa, Italy. E. Dubois, A.L. Opdahl, K. Pohl, eds. Presses Universitaires de Namur, pp. 15-22, 1998.

[17] F. Wanderley, D.S. Silveira, J. Araújo, A. Moreira, "Transforming Creative Re-quirements into Conceptual Models", Seventh IEEE International Conference on Research Chalenges in Information Science, Paris, France, 2013.

[18] OMG - Object Management Group, "Unified Modeling Language (UML) Super-structure Specification", version 2.0, In: http://www.omg.org/cgi-bin/doc?formal/05-07-04. [accessed Dec 2014].

[19] F. Jouault, I. Kurtev, "On the Architectural Alignment of ATL and QVT", Proceedings of the 2006 ACM Symposium on Applied Computing, ACM Press, 2006.

[20] Yu, E.: Modelling Strategic Relationships for Process Reengineering, PhD Thesis, University of Toronto, Toronto, Canada (1995).

[21] Metamodelo i* (Openome), In: https://se.cs.toronto.edu/trac/ome/browser/trunk/workspace/openome_model/model/openome_model.ecore. [accessado em Fevereiro de 2015]

[22] B. Richard. A deidealisation semantics for KAOS. In Proceedings of the 2010 ACM Symposium on Applied Computing, pages 267-274. ACM, March 2010.

[23] Z. Sun, J. Wang, K. He, S. Xiang, and D. Yu. A Model Transformation Method in Service-Oriented Domain Modeling. In Proceedings of the 2010 21st Australian Software Engineering Conference, pages 107-116. IEEE Computer Society, 2010.

[24] M. Lucena, J. Castro, C. Silva, F. Alencar, and E. Santos. Stream: a strategy for transition between requirements models and architec- tural models. In Proceedings of the 2011 ACM Symposium on Applied Computing, pages 699-704. ACM, 2011.

[25] Formalization of iStar Mapping Rules for Class Diagram. In: http://goo.gl/5MDf63. [acessado em 08/07/2015]

[26] Á. Malta, M. Soares, E. Santos, J. Paes, F. Alencar, and J. Castro, "iStarTool: Modeling requirements using the i; framework," CEUR Workshop Proc., vol. 766, no. iStar, pp. 163–165, 2011.

[27] J. Aguilar, I. Garrigós, J.-N. Mazón, and J. Trujillo, "An MDA approach for goal-oriented requirement analysis in Web engineering," J. Univers. Comput. Sci., vol. 16, no. 17, pp. 2475–2494, 2010.

[28] F. Alencar, G. Giachetti, and O. Pastor, "From i* requirements models to conceptual models of a model driven development process," Pract. Enterp. Model. Second IFIP WG 8.1 Work. Conf. PoEM 2009, Stock. Sweden, Novemb. 18-19, 2009, Proc., pp. 99–114, 2009.

[29] G. A. A. C. Filho, A. Zisman, and G. Spanoudakis, "A Traceability Approach for i * and UML Models," in 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, 2003.