

De Architecturas Organizacionais em i* a Arquitecturas Baseadas em Agentes: Uma abordagem orientada a modelos

Carla Silva¹, Pedro Dias², João Araújo², Ana Moreira²

¹ Centro de Ciências Aplicadas e Educação, Universidade Federal da Paraíba,
58297-000, Rio Tinto, Brazil
carla@dce.ufpb.br

² CITI/FCT, Universidade Nova de Lisboa,
2829-516, Caparica, Portugal
pedro.r.dias@gmail.com, {ja, amm}@di.fct.unl.pt

Sumário. O Tropos propõe uma abordagem para o desenvolvimento de sistemas multi-agentes; baseia-se nos conceitos sociais e intencionais oferecidos pelo *framework* de modelação organizacional i*, adotando-o para suportar as várias fases iniciais do desenvolvimento, em particular o desenho arquitectural. Reconhecendo no UML (*Unified Modeling Language*) a capacidade de actuar como uma Linguagem de Descrição Arquitectural, este trabalho especifica, através de uma abordagem orientada a modelos, as transformações necessárias para obter modelos arquitecturais, descritos no perfil de Agência UML, a partir de modelos arquitecturais organizacionais descritos em i* no contexto do Tropos.

Palavras-chave: Desenvolvimento Dirigido por Modelos, Transformações de Modelos, Framework i*, Modelos Arquitecturais, Sistemas multi-Agentes.

1 Introdução

Segundo [1], um agente é um sistema computacional encapsulado que está situado em algum ambiente e é capaz agir de forma flexível e autônoma neste ambiente, a fim de atingir os objetivos para os quais foi projetado. Um sistema multi-agentes (*multi-agent system* – MAS) é um conjunto de agentes que podem desempenhar um ou mais papéis, e que possuem um conjunto bem definido de responsabilidades e visam atingir objetivos. O Tropos [2] oferece uma metodologia baseada em i* [3] para guiar o desenvolvimento de MASs. Esta metodologia tem cinco fases: análise de requisitos iniciais, análise de requisitos finais, desenho arquitectural, desenho detalhado e implementação. A linguagem de modelação i* é utilizada nas 3 primeiras fases, porém, de acordo com o trabalho apresentado em [4], ela apresenta algumas limitações na captura da informação necessária para desenhar a arquitectura de

MASs. Tendo por base o reconhecimento de que o UML 2.0 (*Unified Modeling Language*) [5] suporta a descrição arquitectural de software, este trabalho propõe a automatização da abordagem definida em [6,7,8] e, portanto, apresenta a especificação de regras de transformação de modelos arquitecturais descritos em i* para modelos arquitecturais descritos no perfil de Agência UML definido em [8], seguindo princípios da engenharia orientada a modelos (*Model Driven Engineering – MDE*) [9]. Esta automatização permite tornar mais célere e produtivo o processo de mapear os conceitos i* para os diagramas que compõem o perfil de Agência. Em particular, o objectivo deste trabalho é transformar o modelo de dependência estratégica do i* para o modelo arquitectural definido no perfil UML de forma a obter um desenho arquitectural mais detalhado de acordo com o perfil de Agência [6,7,8].

Existem algumas heurísticas informais, apresentadas em [6], que auxiliam o mapeamento dos modelos i* para um perfil de Agência UML. Este trabalho melhora estas heurísticas e elabora novas heurísticas de mapeamento, para tornar o conjunto de heurísticas mais completo, descrevendo-as usando regras de transformação e mecanismos da MDE. Assim, conseguiremos usufruir as vantagens da MDE e das capacidades de modelação do UML para o desenvolvimento de MAS.

Este artigo está organizado como segue. A secção 2 apresenta a fundamentação teórica deste trabalho. A secção 3 apresenta as heurísticas de mapeamento entre a linguagem i* e a linguagem definida pelo perfil de Agência UML e as respectivas regras de transformação. A Secção 4 apresenta os trabalhos relacionados e, por fim, a secção 5 discute as contribuições do artigo e aponta para alguns trabalhos futuros.

2 Fundamentação Teórica

2.1 O Framework i*

O i* foi desenvolvido para modelar e raciocinar sobre ambientes organizacionais e seus sistemas de informação [10, 11]. Ele oferece uma abordagem de engenharia de requisitos baseada em actores, centrando-se nos *stakeholders* do sistema e nas suas relações. Estas relações mostram como os actores dependem uns dos outros para alcançarem os seus objectivos. O suporte explícito para a modelação de dependências entre múltiplos actores permite introduzir alguma análise social na fase de análise e desenho do sistema. O propósito deste tipo de análise é descobrir os “porquês” dos requisitos (e.g., porque é que um determinado requisito é necessário). Este tipo de estudo, baseado em contextos sociais e organizacionais, permite não só perceber os requisitos do sistema, como também ajudar a prepará-lo para mudanças futuras [11].

2.2 Estilos Arquitecturais Organizacionais

As técnicas de desenvolvimento de software actuais precisam se basear em arquitecturas capazes de evoluir e suportar alterações sem custos elevados. Quer seja para acomodar novos componentes quer para lidar com novos requisitos, estas

modificações ocorrerão mais cedo ou mais tarde. É por isso necessário que as arquitecturas sejam flexíveis e com componentes fracamente acoplados de forma a suportar modificações que permitam obter arquitecturas optimizadas para um conjunto inicial de requisitos.

Alinhando com a demanda actual por arquitecturas flexíveis, o projecto Tropos definiu um catálogo de estilos arquitecturais organizacionais para aplicações multi-agentes cooperativas, dinâmicas e distribuídas. Uma organização pode ser vista como um compromisso social que persegue objectivos colectivos, que controla o seu próprio desempenho e que tem uma fronteira que a separa do seu ambiente. Assim, um sistema de software dentro de uma empresa pode ser analisado nos mesmos trâmites que uma organização.

Estilos arquitecturais organizacionais incluem União Estratégica (*Joint Venture*), Estrutura em 5 (*Structure in 5*), Integração Vertical (*Vertical Integration*), Apropriação (*Co-Optation*), Comprimento de Braço (*Arm's Length*), Tomada de Controlo (*Takeover*), Pirâmide (*Pyramid*), Contratação Hierárquica (*Hierarchical Contracting*), Oferta (*Bidding*) e Estrutura Plana (*Flat Structure*) [10].

A Fig. 1 mostra o desenho arquitectural para o MAS e-news [6, 8], após aplicar o estilo organizacional *Joint-Venture* e representado através do modelo de dependência estratégica (do inglês, *Strategic Dependency – SD*) do *i**. No modelo SD, actores dependem uns dos outros para atingirem objectivos e *softgoals* (estes representam requisitos não-funcionais), realizar tarefas e disponibilizar recursos. Um objectivo representa-se por um oval. As tarefas são representadas por hexágonos. A representação de *softgoals* é feita através de “nuvens”. Os recursos representam-se por rectângulos (na Fig. 1 não estão representados objectivos nem *softgoals*).

Segundo o estilo *Joint-Venture*, existe um actor (*Chief Editor*) que coordena as tarefas e gerencia os recursos partilhados entre os outros actores parceiros (*Editor*, *Webmaster* e *Photographer/Reporter*). Cada um dos actores parceiros controla-se a si próprio numa dimensão local e podem interagir directamente com outros actores para trocas de recursos. No entanto, o *Chief Editor* é o único responsável por operações estratégicas e de coordenação do sistema, e dos seus actores, a um nível global.

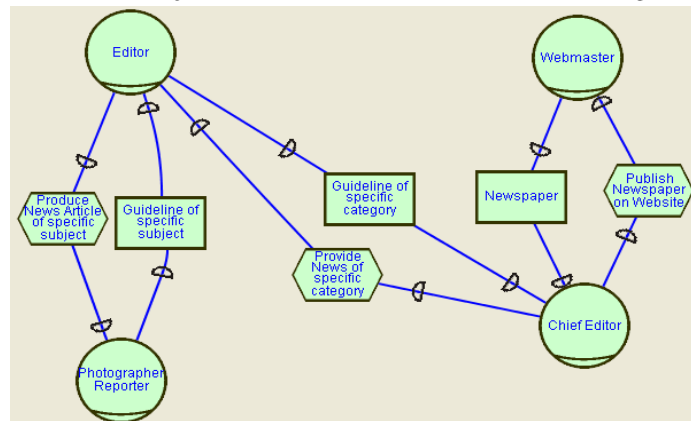


Fig. 1. Modelo arquitectural do sistema e-news em *i**.

2.3 Estilos Arquiteturais Organizacionais em UML

De acordo com o trabalho apresentado em [4], o uso do i* como uma Linguagem de Descrição Arquitetural (*Architectural Description Language – ADL*) é inadequado, dado que não consegue modelar algumas características arquiteturais de MASSs, tais como portas, conectores, protocolos e interfaces [4].

Para permitir o uso de ferramentas UML na modelação de MAS, definiu-se em [6,7,8] um perfil UML para MAS, denominado perfil de Agência. Este perfil possibilitou a extensão de algumas metaclasses do UML de forma que alguns de seus diagramas fossem usados para modelar as seis visões do desenho de MAS, a saber: Arquitetural, Comunicação, Ambiental, Intencional, Racional e Acção.

Este trabalho concentra-se apenas no diagrama arquitetural, que é uma extensão ao diagrama de classes do UML, usado para modelar a visão estática dos módulos que compõem um sistema [6,7,8]. Para explicar este diagrama, usaremos o mesmo sistema *e-news* utilizado na secção 2.2. O diagrama da Fig. 2 está definido em termos de *AgentRoles* que precisam de interagir uns com os outros para alcançar os seus *Dependums*. Um *dependum* define a natureza do acordo entre as duas entidades envolvidas no relacionamento. *AgentRoles* possuem *OrganizationalPorts* que permitem a troca de mensagens, através de *AgentConnectors*, entre agentes que realizam os *AgentRoles*.

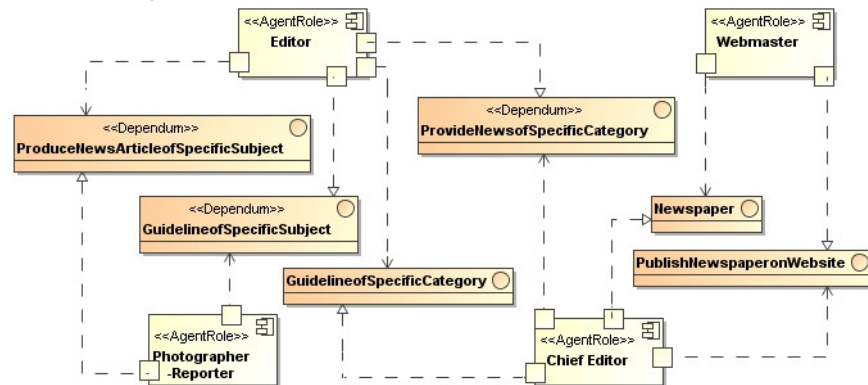


Fig. 2. Modelo arquitetural do sistema *e-news* no perfil de Agência UML

Para transformar modelos arquiteturais descritos em i* (Fig. 1) em modelos arquiteturais descritos de acordo com o perfil de Agência UML [6, 8] (Fig. 2), em [8] foram definidas algumas heurísticas para conduzir o mapeamento dos conceitos definidos nas duas linguagens. Este trabalho apresenta uma evolução destas heurísticas e a sua formalização através de regras de transformação de modelos.

2.4 Desenvolvimento de Software Orientado a Modelos

O Desenvolvimento Orientado a Modelos (do inglês, *Model-Driven Development – MDD*) [9] é uma abordagem da engenharia do software que consiste na aplicação de

do *dependum*. Em última análise, os *Actors* e os *Dependums* são especializações de “*Node*”, que junto ao “*IStarRelationship*” são componentes do “*IStarModel*”.

A Fig. 3 (b) apresenta o metamodelo do diagrama arquitetural que define que os *AgentRoles* precisam trocar mensagens através de um *AgentConnector* de forma a alcançarem o acordo contratual de fornecimento de serviço entre eles (i.e., o *Dependum*). Uma *OrganizationalPort* determina um ponto de interação distinto entre o *AgentRole* e o seu ambiente. Um *AgentConnectorEnd* é ponto final de um *AgentConnector*, que agrega o *AgentConnector* à *OrganizationalPort*. O *AgentRole*, responsável por oferecer o serviço possui um *OrganizationalPort* e relaciona-se com o *Dependum* através de uma relação *dependee*. O *AgentRole*, que precisa do serviço, tem um *OrganizationalPort* e relaciona-se com o *Dependum* através de uma relação *depend*. A metaclass *Dependum* possui a propriedade *trust* que mostra que o *depend* tem de confiar no *dependee* para delegação de actividades. A propriedade *trust* pode ser de três tipos: total (*full*), média (*medium*) ou nenhuma (*none*), como o definido na enumeração *TrustKind*.

Este trabalho apresenta um processo automatizado capaz de realizar transformações de modelos SD do *i** em modelos arquiteturais de Agência. A razão pela qual utilizamos inicialmente apenas estes dois modelos particulares como modelo origem e modelo destino tem a ver com a simplicidade. Tornar-se-ia muito complexo trabalhar sobre a totalidade dos metamodelos *i** e de Agência. Preferimos dividir os metamodelos em partes menores para que fosse possível conseguir um processo de mapeamento com qualidade, que no futuro poderá ser aplicado ao que resta dos modelos em causa.

De forma a tornar mais completo o processo de transformação, procedemos também à elaboração de novas heurísticas de mapeamento. O objectivo foi tornar o conjunto de heurísticas mais completo e acurado de forma a ser descrito mais facilmente em uma linguagem de definição de regras de transformações de modelos.

3.1 Heurísticas de mapeamento *i** já existentes

Em [6,8] definiram-se algumas heurísticas tendo em conta os elementos do diagrama arquitetural de Agência, tal como apresentado na Tabela 1.

É preciso lembrar que em [8] se encontram as heurísticas propostas para o mapeamento dos modelos *i** para qualquer um dos seis diagramas do perfil de agência (Arquitetural, Comunicação, Ambiental, Intencional, Racional e Acção). Porém, como vamos trabalhar apenas com o diagrama arquitetural, a Tabela 1 só apresenta os mapeamentos correspondentes a este diagrama.

No diagrama arquitetural é introduzido o conceito de confiança (do inglês, *trust*) nas relações do *i**. Isto não se encontra devidamente reproduzido nas heurísticas. O conceito de confiança pode ser visto como dependendo directamente do atributo *degree* das relações. Na secção seguinte é dada uma sugestão para esta limitação.

Outro conceito não considerado no conjunto actual de heurísticas é o de herança. No *i** podem existir relações de herança (*IsA*) entre actores. Essa possibilidade não era considerada nas heurísticas de mapeamento. Tendo em conta as limitações existentes nas heurísticas actuais, importa agora procurar superá-las. Na próxima secção, apresentaremos a nossa proposta para superar essas limitações.

Table 1. Levantamento de um subconjunto das heurísticas definidas em [8].

#	Framework i*	Perfil de Agência	Comentários
#1	Papel (role)	Classe «AgentRole»	Um papel em i* representa-se como um papel desempenhado por um Agente em UML. Diagrama arquitetural.
#2	Tarefa	Classe «MacroPlan»	Representa os meios pelos quais se alcança um goal. Diagrama intencional e arquitetural.
#3	Sub-tarefa	Classe «ComplexAction»	Representa os passos que compõem um «MacroPlan». Diagrama arquitetural.
#4	Softgoal (como parte de uma tarefa)	Classe «Goal»	Cria uma «ComplexAction» responsável por gerar esse goal no «MacroPlan» correspondente à tarefa.
#5	Dependum	Interface «Dependum»	Um dependum pode ser de quatro tipos: goals, softgoals, tarefas e recursos. Diagrama arquitetural.
#6	Depender	Dependência «Depender»	A posição de depender em i* torna-se uma dependência com o mesmo nome. Diagrama arquitetural.
#7	Dependee	Realização «Dependee»	A posição dependee em i* representa-se como uma realização de interface com o mesmo nome. Diagrama arquitetural.
#8	Depender – dependum – dependee	Associação «Conector»	Portas são adicionadas aos agentes para proporcionarem um link para o conector. Diagrama arquitetural.
#9	Goal/Softgoal	Classe «Goal»	Aplica-se apenas a goals / softgoals não decompostos. Representa os objectivos do sistema. Diagrama arquitetural e intencional.

3.2 Novas heurísticas de mapeamento

Um dos propósitos deste trabalho é enriquecer as heurísticas resumidas na seção anterior, tornando-as mais claras e propondo novas heurísticas (Figuras 4, 5 e 6).

Degree – Trust		
Diagrama SD	Diagrama Arquitetural	Descrição
Dependência Crítica	Trust = Full Degree = Critical	Valor do atributo trust e degree do elemento Dependum. Diagrama arquitetural
Dependência Aberta	Trust = None Degree = Open	Valor do atributo trust e degree do elemento Dependum. Diagrama arquitetural
Dependência Normal	Trust = Medium Degree = Committed	Valor do atributo trust e degree do elemento Dependum. Diagrama arquitetural
Se uma dependência, independentemente do seu tipo, for marcada como “Critical”, o atributo degree no elemento Dependum terá a mesma designação e o atributo trust no mesmo elemento é definido como “Full”. Se essa dependência estiver representada como “Open”, o atributo degree no elemento Dependum terá a mesma designação e o atributo trust do mesmo elemento é designado como “None”. Caso contrário, é fixado o valor de trust como sendo “Medium” e o valor de degree é comprometido. Mesmo assim, esta heurística pode ser vista como um padrão geral aplicável à maioria dos sistemas. Pode-se dar o caso em que todas as dependências entre actores, que não só as críticas, exijam um grau de confiança “Full”.		

Fig. 4. Heurística *degree-trust*.

Entre essas características encontramos regras que modelam a relação que existe entre os conceitos de grau (*degree*) e confiança (*trust*) (Fig. 4). Já a Fig. 5 trata de uma heurística que mapeia os diferentes tipos de relações de dependência (objetivo, *softgoal*, tarefa, recurso) existentes em i*. Por último, a Fig. 6 define uma heurística para a noção de herança.

Tipo de Dependência – DependumKind		
Diagrama SD	Diagrama Arquitectural	Descrição
Dependência de Goal	Valor “goal” na metaclassa de enumeração DependumKind	Uma dependência de “goal” implica que o valor goal esteja contido dentro dos valores da metaclassa de enumeração DependumKind. Diagrama arquitectural
Dependência Softgoal	Valor “softgoal” na metaclassa de enumeração DependumKind	Uma dependência de “softgoal” implica que o valor softgoal esteja contido dentro dos valores da metaclassa de enumeração DependumKind. Diagrama arquitectural
Dependência de Tarefa	Valor “task” na metaclassa de enumeração DependumKind	Uma dependência de tarefa implica que o valor task esteja contido dentro dos valores da metaclassa de enumeração DependumKind. Diagrama arquitectural
Dependência Recurso	Valor “resource” na metaclassa de enumeração DependumKind	Uma dependência de “resource” implica que o valor resource esteja contido dentro dos valores da metaclassa de enumeração DependumKind. Diagrama arquitectural
No modelo SD existe quatro tipos de dependência: dependência de goal, de softgoal, de recurso e de tarefa. Estas dependências têm o nome do tipo de dependum (DependumKind) que as compõem. Assim, pode-se definir um mapeamento directo entre os tipos de dependum existentes em i* para uma metaclassa de enumeração DependumKind no diagrama arquitectural.		

Fig. 5. Heurística tipo de dependência

3.1.3 Regras de transformação

O processo de automatização dos mapeamentos indicados anteriormente é feito recorrendo à linguagem de transformação de modelos ATL (*Atlas Transformation Language*) [15].

Como primeira regra (Fig. 7) temos o mapeamento de um actor para um *AgentRole*. Esta regra pega no conceito de Actor do i* e transforma-o directamente num *AgentRole* do Diagrama Arquitectural. Esta passagem é feita apenas dando o nome da entidade original à de destino.

Herança		
Diagrama SD	Diagrama Arquitectural	Descrição
Relação IsA	Relação ISA	Mapeamento directo da generalização. Diagrama arquitectural
A noção de herança é idêntica em ambos os diagramas. Se, no modelo SD, um Actor é uma especialização de outro, os AgentRoles correspondentes terão a mesma relação de herança entre si.		

Fig. 6. Heurística para herança

Esta transformação serve também para dar suporte às transformações que envolvam o conceito de herança. Desta forma, se um actor for uma especialização de outro, ele terá a informação sobre a sua relação *isa*. Essa informação é passada automaticamente para o modelo destino através já especificação `isa <- ss.isa`.

```
rule Actor2AgentRole {
  from
    ss : IStarSD!Actor
  to
    t : UMLArquitetural!AgentRole(
      name <- ss.Name,
      isa <- ss.isa
    ) }
```

Fig. 7. Regra de transformação de Actor para *AgentRole*

A segunda regra (Fig. 8) mapeia um *DependeeLink* do modelo SD para um *Dependee* no modelo arquitectural. Neste caso, a transformação consiste em mapear as ligações da entidade de origem para a entidade de destino.

```
rule DependeeLink2Dependee {
  from
    s : IStarSD!DependeeLink
  to
    t : UMLArquitetural!Dependee(
      dde <- s.target,
      oardee <- s.dee
    ) }
```

Fig. 8. Regra de transformação de *DependeeLink* para *Dependee*

Uma terceira regra (Fig. 9) transforma um *DependerLink* num *Depender*. Esta transformação ocorre na mesma linha de raciocínio da anterior.

```
rule DependerLink2Depender {
  from
    s : IStarSD!DependerLink
  to
    t : UMLArquitetural!Depender(
      ddr <- s.source,
      oarder <- s.der
    ) }
```

Fig. 9. Regra de transformação de *DependerLink* para *Depender*

Como quarta transformação (Fig. 10), tem-se o mapeamento do *dependum* do modelo SD para o *dependum* do modelo arquitectural. O processo aqui é um pouco mais elaborado. A entidade *Dependum* contém vários atributos que importam tratar aquando da transformação. O atributo *type* toma o valor que o *dependum* no modelo original tiver dentre os valores definidos por *dependumKind* (i.e., *goal*, *softgoal*, tarefa ou recurso). O atributo *degree* terá o valor que o *dependum* no modelo original tiver dentre os valores definidos por *degreeKind* (i.e., aberta, crítica e comprometida). Tem-se um mapeamento directo para o atributo *name* e para as relações com o *dependum*.

Estas regras de transformação foram aplicadas ao modelo SD do sistema e-news, apresentado na Fig. 1, resultando no modelo arquitectural apresentado na Fig. 2. É importante ressaltar que os modelos apresentados nas Figuras 1 e 2 são representados

através da sintaxe concreta da linguagem i* e da linguagem do perfil de agência. A sintaxe concreta define a forma gráfica ou textual de representação do modelo. Já a sintaxe abstracta de um modelo representa as instâncias das metaclasses contidas do metamodelo que define a linguagem de modelação. De facto, são os modelos abstractos que são utilizados no processamento automático de transformação de modelos. Por outro lado, a sintaxe concreta é a interface oferecida para o utilizador construir modelos. Os modelos abstractos de origem e destino, para o estudo de caso do sistema e-news, não foram apresentados no artigo por questão de espaço, mas podem ser encontrados com detalhe em [14].

```

rule DependumSD2DependumArq {
  from
    s : IStarSD!Dependum
  to
    t : UMLArquitetural!Dependum(
      type <- s.dependumKind,
      degree <- s.degreeKind,
      name <- s.name,
      odde <- s.source,
      oddr <- s.target,
      trust <- if (s.degreeKind = 'Committed') then
'Medium'
                else
                if (s.degreeKind = '') then 'None'
                else 'Full'
                endif
            endif
    ) }

```

Fig. 10. Regra de transformação de Dependum para Dependum

4 Trabalhos Relacionados

Em [16] há um trabalho que usa o Tropos como metodologia para construir MASs complexos e foca a transformação automática da decomposição de tarefas do i* para um diagrama de actividades do UML. Trata-se de uma abordagem baseada numa linguagem de transformação declarativa baseada nos conceitos de definição de padrões, regras de transformação e relações de rastreio. Similarmente, em [17] sugere-se a ideia de um ambiente de desenvolvimento orientado a modelos que abarque todo o processo de desenvolvimento de MASs do Tropos. Porém, em ambos os trabalhos, os modelos de origem e destino eram diferentes dos modelos utilizados no presente trabalho.

O trabalho descrito em [18] foca o desenvolvimento de MASs usando uma abordagem orientada a modelos capaz de suportar a constante mudança destes sistemas e que permita o mapeamento entre modelos de desenho e modelos de implementação. Tal como na nossa proposta, também aqui se descrevem os modelos de origem e de destino e as regras de transformação. É um processo similar ao nosso, porém trabalha-se em um nível mais baixo de abstracção.

Apesar de relevantes, os trabalhos apresentados em [17, 18] focam nas fases mais adiantadas do processo de desenvolvimento de software. A nossa abordagem propõe a

transformação de modelos nas fases iniciais do Tropos. O trabalho apresentado em [16] propõe transformações entre modelos de requisitos em i^* para modelos de requisitos em UML, enquanto que a nossa abordagem propõe transformações de modelos de requisitos em i^* para modelos de desenho arquitetural em um perfil de Agência UML.

5 Conclusões e trabalhos futuros

A abordagem apresentada neste artigo representa um passo em frente rumo à automatização do processo de transformação de modelos arquiteturais descritos em i^* para modelos arquiteturais descritos no perfil de Agência UML.

Com esta abordagem podemos afirmar que é possível juntar o melhor de dois mundos. Podemos nos beneficiar da capacidade da framework i^* para modelar o software usando conceitos sociais e intencionais e, assim, adequada para as fases de requisitos iniciais e finais no desenvolvimento MAS. Podemos também usufruir das capacidades do UML como ADL para tornar o uso do Tropos mais popular, visto que o uso da UML como ADL tem sido cada vez mais frequente.

O uso da MDE na abordagem proposta trás uma série de vantagens. Por exemplo, parte-se de um modelo totalmente independente de plataforma, como o modelo SD do i^* , que são portáveis e podem ser reutilizados na geração de diferentes modelos específicos de plataforma (*Platform Specific Models* - PSMs). Na nossa proposta, o o modelo arquitetural de Agência foi escolhido para ser o PSM.

A automatização proposta pretende facilitar a realização da fase de desenho (ou projeto) arquitetural do Tropos. Esta automatização foi alcançada pela definição dos metamodelos origem e destino em Ecore e pela especificação de um conjunto de regras de transformação de conceitos entre os dois metamodelos, utilizando a linguagem de transformação ATL em ambiente Eclipse.

O processo de automatização apresentado não se esgota no mapeamento entre os diagramas SD e arquitetural. É preciso também mapear os conceitos presentes em todos os outros diagramas definidos no i^* e no perfil de Agência UML.

Para além da definição de heurísticas de mapeamento e especificação de transformações entre as linguagens de modelagem manipuladas na abordagem apresentada, importa também prover ferramentas capazes de produzir os modelos gráficos (i.e., a sintaxe concreta do modelo) para estas linguagens. Uma ferramenta para criação de modelos i^* , construída com base no GMF [13], é o Istar Tool [19]. Ainda é preciso construir uma ferramenta para a criação de modelos do perfil de Agência UML. Também é preciso validar as regras de transformação propostas através de sua aplicação no desenvolvimento de sistemas multi-agentes reais.

Agradecimentos. Este trabalho foi parcialmente apoiado pelos órgãos de fomento à pesquisa CNPq, CAPES e GRICES.

Referencias

1. Wooldridge, M. An Introduction to Multiagent Systems. England. pp.: 15-103. 2002.
2. Castro, J., Kolp, M., Mylopoulos, J. Towards Requirements Driven Information Systems Engineering: The Tropos Project. Information Systems. Holanda, v.27, n.6, 365 - 389 (2002)
3. Yu, E., Modeling Strategic Relationships for Process Reengineering, Ph.D. thesis, Department of Computer Science, University of Toronto (1995)
4. Castro, J., Silva, C., Mylopoulos, J. Modeling Organizational Architectural Styles In UML. In: The 15th Conference On Advanced Information Systems Engineering (CAiSE'03), LNCS, vol. 2681, pp.111 – 126, Springer Verlag, Klagenfurt/Velden (2003)
5. Object Management Group (2005) Unified Modeling Language (UML): Superstructure. Version 2.0, Available: www.omg.org/docs/formal/05-07-04.pdf.
6. Silva, C., Araújo, J., Moreira, A., Castro, J., Tedesco, P., Alencar, F., Ramos, R. Modeling Multi-Agent Systems using UML. In: XX Brazilian Symposium of Software Engineering, pp. 81-96, Sociedade Brasileira de Computação, Florianópolis, Brasil (2006)
7. Silva, C., Araújo, J., Moreira, A., Castro, J., Alencar, F., Ramos, R. Organizational Architectural Styles Specification. In Jornadas de Ingeniería del Software y Bases de Datos, Barcelona, Espanha (2006)
8. Silva C., Separating Crosscutting Concerns in Agent Oriented Detailed Design: The Social Patterns Case. Tese de Doutorado. Centro de Informática da Universidade Federal de Pernambuco, Brasil (2007)
9. Kleppe, A., Warmer, J., Bast, W. MDA explained The Model Driven Architecture: Practice and Promise, Addison-Wesley (2007)
10. Yu, E., Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering, presented at Requirements Engineering, Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97), pp. 226-235, Washington D.C., USA (1997)
11. Kolp, M., Giorgini, P., Mylopoulos, J.: Information Systems Development through Social Structures. Proceedings of the 14th international conference on Software engineering and knowledge engineering (SEKE'02), ACM, pp. 183 – 190, Ischia, Italy (2002)
12. Eclipse Modeling Framework (EMF), <http://www.eclipse.org/modeling/emf/>, Último acesso em Outubro de 2010.
13. Graphical Modeling Framework (GMF), <http://www.eclipse.org/modeling/gmf/>, Último acesso em Outubro de 2010.
14. Dias, P. De arquiteturas organizacionais em i* para arquiteturas baseadas em Agentes – Uma abordagem orientada a modelos. Dissertação de Mestrado. Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia, Departamento de Informática (2008)
15. Jouault, F. and Kurtev, I. Transforming models with ATL. In Proceedings of Satellite Events at the MoDELS 2005 Conference, LNCS, vol. 3844, pp. 128--138. Springer (2006)
16. Perini, A. and Susi, A. Automating Model Transformations in Agent-Oriented Modeling. In Agent-Oriented Software Engineering VI, LNCS, vol. 3950. Springer, Berlin/Heidelberg, pp. 167-178 (2006)
17. Bertolini, D., Delperio, L., Mylopoulos, J., Novikau, A., Orlor, A., Penserini, L., Perini, A., Susi, A., Tomasi, B. A Tropos Model-Driven Development Environment. CAiSE 2006 Forum proceedings, LNCS, vol. 4001, Springer Verlag (2006).
18. Da Silva, V, DeMaria, B., Lucena, C. A MDE-Based Approach for Developing Multi-Agent Systems. In Electronic Communication of the European Association of Software and Technology (EASST): Software Evolution through Transformations, vol. 3 (2006)
19. IStarTool Project: A Model Driven Tool for Modeling i* models. Disponível em: <http://portal.cin.ufpe.br/ler/Projects/IstarTool.aspx>, Jul (2009)