Uma solução baseada em ontologia para o tratamento de erros em modelos construídos na linguagem i*

Heyde Francielle do Carmo França[12], Renato de Freitas Bulcão Neto[1]
1 Departamento de Informática – Universidade Federal de Goiás (UFG)
{heydecarmo, renato}@inf.ufg.br
2 Instituto Federal Goiano – Campus Rio Verde (IF Goiano)
heyde.franca@ifgoiano.edu.br

Abstract. The i* language faces problems regarding the quality of requirements models, which include typical mistakes of misuse of its constructs, the presence of ambiguities on the interpretation of those constructs and the complexity of the resulting i* models. This paper proposes the extension of an ontology-based representation for i* models aiming at preventing the most well-known errors while constructing such models. Results demonstrate an approximate coverage of 70% of those common errors in the context of a case study.

Keywords: Requirements Engineering, Goal, i* Language, OntoiStar, OWL.

1 Introdução

A técnica GORE (*Goal-Oriented Requirements Engineering*) captura e representa as reais intenções de *stakeholders* por meio de metas que devem ser satisfeitas, provendo uma maneira natural de estruturação de documentos de requisitos complexos [1]. Dentre as linguagens para modelagem de requisitos baseadas em GORE, destaca-se a linguagem i* dados os benefícios para a modelagem de requisitos como, por exemplo, acrescentar o entendimento das intenções e motivações dos membros da organização envolvida, promover uma rastreabilidade entre as funcionalidades do sistema e os requisitos, e permitir a análise de caminhos alternativos que podem ser seguidos para a satisfação de requisitos [3], [11], [13].

Apesar das vantagens que a linguagem i* apresenta, a literatura aponta problemas quanto à qualidade dos modelos produzidos nessa linguagem: a complexidade dos modelos resultantes, erros típicos de mau uso de construtores e a presença de ambiguidades na interpretação desses construtores [4], [8], [9]. Estes dois últimos problemas estão interrelacionados, pois o mau uso de seus construtores, em geral, acontece por erros de interpretação da sintaxe e da semântica da linguagem i* [11].

Em [5] foi proposta a ontologia *OntoiStar* de forma a não apenas estruturar, mas também fornecer semântica aos construtores da linguagem i*, porém sem tratar erros típicos em modelos i*, como erros envolvidos na definição de atores, dependências e relacionamentos entre atores. Esses erros comumente encontrados estão resumidos em um documento denominado catálogo de erros frequentes da linguagem i* [2], [4].

Este artigo relata a extensão da ontologia *OntoiStar* usando a semântica da linguagem OWL, com vistas a reduzir a ocorrência de erros descritos no catálogo de erros frequentes [2], [4] em modelos de requisitos construídos na linguagem i*. Ontologias podem ser utilizadas para resolver este tipo de problema, pois definem um vocabulário de representação que captura os conceitos e relações de um domínio, e um conjunto de axiomas que restringem a sua interpretação [7]. Para validação da proposta foi utilizado o cenário *Media Shop*, adaptado de [2] e [6], utilizado como ilustração de modelos i* em vários artigos da área.

Este artigo está estruturado como se segue: a seção 2 apresenta a extensão da ontologia *OntoiStar* para tratar erros em modelos i*, e a seção 3 descreve conclusões, limitações e trabalhos futuros.

2 Extensão da ontologia *OntoiStar*

Com base no catálogo de erros frequentes descritos em [2] e discutidos em [4] e [14], foram elaborados e incluídos axiomas e restrições na *OntoiStar* para prevenir a ocorrência desses erros. Para isso, utilizou-se a semântica de construtores OWL e o editor *Protégé* integrado ao raciocinador *Pellet*. Todos os exemplos apresentados seguem a modelagem do cenário *Media Shop* que inclui uma loja de vendas de diferentes tipos de itens (livros, jornais, revistas, cds, etc) que pode ser visto com mais detalhes em [2], [6]. Para fins de implementação das restrições apresentadas a seguir foi utilizada como referência a evolução do i* original[1], o que justifica a configuração empregada no tratamento dos erros 4 e 8 descritos nesta seção.

Erro 1: Atores dentro da fronteira de outro ator - a classe *ActorBoundary* da *OntoiStar* representa a fronteira de um ator. Para eliminar a possibilidade da existência de mais de um ator dentro dessa fronteira, a classe *ActorBoundary* foi redefinida para que apresente os relacionamentos *has_Actor_Bondaury* e/ou *has_Actor_boundary_elements* com exatamente 0 atores. Assim, o ator do *ActorBoundary* será o único dentro do relacionamento, permitindo apenas relacionamento com elementos da classe *InternalElement*. Após essa redefinição ser inserida na *OntoiStar*, o raciocinador *Pellet* integrado ao *Protégé* notifica que uma inconsistência é detectada ao se tentar reproduzir o erro da Figura 2, ligando-se os atores *Medi@* e *Bank_Cpy* por meio de *has_Actor_Boundary*.

Erro 2: Conexão de um *InternalElement* para um *ActorBoundary* - a classe *ActorBoundary* não deve possuir nenhum relacionamento direto do tipo *contribution* ou *dependency* com um *InternalElement*. As redefinições de classe apresentadas na Tabela 1 evitam a existência de relacionamentos do tipo *DependencyRelationship* ou *InternalElementRelationship* com a classe *ActorBoundary*. As regras foram criadas usando *DependableNode* em vez de *InternalElement*, pois assim garante que não poderão existir esses relacionamentos, nem com itens da classe *InternalElement*, nem com a classe *Actor*.

[1] Disponível em: http://istar.rwth-aachen.de/tiki-index.php?page=iStarQuickGuide

Erro 3: Dependência de ator - deve-se usar *DependencyRelationships* apenas para expressar dependências entre um membro da classe *Actor* e um *InternalElement*; não deve usar relacionamentos do tipo *InternalElementRelationShip* para representar dependências. Para evitar este erro, criaram-se as restrições descritas na Tabela 2. O símbolo * representa as possibilidades de um tipo de construtor, i.e. *source* e *target*.

Tabela 1. Condições para que não haja conexão entre *InternalElement* e *Actor Boundary*

has_Dependency_DependerLink_*-ref exactly 0 DependableNode
has_Dependency_DependeeLink_*-ref exactly 0 DependableNode
has_Dependency_DependumLink_*-ref exactly 0 DependableNode
has_InternalElement_MeansEndLink_*-ref exactly 0 DependableNode
has_InternalElement_AndDecompositionLink_*-ref exactly 0 DependableNode
has_InternalElement_ContibutionLink_*-ref exactly 0 DependableNode

Tabela 2. Condições para que não haja relacionamentos do tipo *InternalElementRelationship* para representar uma dependência entre um *Actor* e um *InternalElement*.

has_InternalElement_MeansEndLink_*-ref exactly 0 InternalElement
has_InternalElement_AndDecompositionLink_*-ref exactly 0 InternalElement
has_InternalElement_ContibutionLink_*-ref exactly 0 InternalElement
has_InternalElement_DecompositionLink_*-ref exactly 0 InternalElement
has_InternalElement_OrDecompositionLink_*-ref exactly 0 InternalElement

Erro 4: Refinando metas - uma meta não pode ser ligada diretamente a outra meta. Esta deve ser decomposta utilizando uma ligação do tipo *MeansEndLink*, e estas ligações só podem ligar tarefas a metas. Seguem na Tabela 3 as redefinições de classes para prevenir que este erro ocorra.

Erro 5: *Link* de dependência interna - não devem ser realizadas relações de dependência entre um *InternalElement* dentro da fronteira de um ator. Pode haver um *Dependum* que se origina externamente à fronteira do ator ligando-se a um *InternalElement* de dentro da fronteira, mas nunca uma relação de dependência entre dois *InternalElement* internos à fronteira. Para garantir que este erro não ocorra foi acrescentada uma redefinição na classe *ActorBoundary* como mostra a Tabela 4.

Erro 6: *Link* de dependência externa - não se deve usar um *link* de dependência entre dois atores sem um *Dependum*. Ou seja, um ator não pode estar diretamente ligado a outro ator. Para garantir que esse erro não ocorra foram inseridas as seguintes redefinições nas classes *DependeeLink* e *DependerLink*, conforme ilustra a Tabela 5.

Tabela 3. Condições para que uma tarefa só possa ser decomposta utilizando ligações do tipo *MeansEndLink* e que só possa ser decomposta em metas.

has_InternalElement_MeansEndLink_*-ref exactly 0 Resource
has_InternalElement_MeansEndLink_*-ref exactly 0 Softgoal
has_InternalElement_MeansEndLink_*-ref exactly 0 Plan
has_InternalElement_MeansEndLink_*-ref exactly 0 Service
has_InternalElement_MeansEndLink_*-ref exactly 0 Process

Tabela 4. Regra para certificar-se que um *Depender* não se origine na fronteira de um ator.

has_Dependency_DependerLink_source-ref exactly 0 InternalElement

Tabela 5. Regras para certificar-se que um *Dependee* e um *Depender* não se conectem diretamente a outro ator.

has_Dependency_DependeeLink_*-ref exactly 0 Actor
has_Dependency_DependerLink_*-ref exactly 0 Actor

Erro 7: *Link* de contribuição interna - um *ContributionLink* só deve ser usado para conectar um *IntentionalElement* a uma *Softgoal*. Para garantir que esse erro não ocorra foi inserido na classe *ContributionLink* as redefinições da Tabela 6.

Tabela 6. Regras inseridas para garantir que um *ContributionLink* só conecte um *InternalElement* somente a *Softgoal*.

has_InternalElement_ContibutionLink_*-ref exactly 0 Goal
has_InternalElement_ContibutionLink_*-ref exactly 0 Process
has_InternalElement_ContibutionLink_*-ref exactly 0 Service

Erro 8: *Link MeansEnd* interno - os *links* do tipo *MeansEnd* só podem ser utilizados para ligar *Tasks* a *Goals*. Para garantir que este relacionamento seja utilizado de forma correta foram inseridas as seguintes redefinições na classe *MeansEndLink* da ontologia *OntoiStar*, descritas na Tabela 7.

Tabela 7. Regras inseridas para garantir que um *MeansEndLink* só conecte *Tasks* a *Goals*.

has_InternalElement_MeansEndLink_*-ref exactly 0 Resource
has_InternalElement_MeansEndLink_*-ref exactly 0 Softgoal
has_InternalElement_MeansEndLink_*-ref exactly 0 Plan
has_InternalElement_MeansEndLink_*-ref exactly 0 Service
has_InternalElement_MeansEndLink_*-ref exactly 0 Process

Erro 9: Ator sem ligação ou com ligação incorreta - não deve existir no modelo um ator sem uma ligação de dependência ou *ActorRelationship*, e os relacionamentos entre atores devem obedecer às regras:

- *IsA* – relacionamentos apenas entre atores do mesmo tipo;
- *Plays* – relacionamentos apenas entre um agente e um papel;
- *Covers* – relacionamentos apenas entre uma posição e um papel;
- *Occupies* – relacionamentos apenas entre uma posição e um agente; e
- *INS* – relacionamentos apenas entre um agente e outro agente.

Vale ressaltar que as redefinições de classes descritas na Tabela 8 garantem apenas que os relacionamentos estão corretos, mas não garantem que não exista um ator sem ligação. Isto ocorre porque não é possível criar uma regra de obrigatoriedade de existência de um tipo de relacionamento utilizando a expressividade dos construtores da linguagem OWL. Sendo assim, o erro 9 é coberto apenas parcialmente.

Tabela 8. Regras para garantir ligações corretas entre atores.

has_Actor_ IsALink _*-ref exactly 1 Actor
has_Actor_ PlaysLink _*-ref exactly 1 Role
has_Actor_ PlaysLink _*-ref exactly 0 Agent
has_Actor_ PlaysLink _*-ref exactly 0 Position
has_Actor_ CoversLink _*-ref exactly 0 Role
has_Actor_ CoversLink _*-ref exactly 1 Agent
has_Actor_ CoversLink _*-ref exactly 0 Position
has_Actor_ OccupiesLink_*-ref exactly 0 Role
has_Actor_ OccupiesLink_*-ref exactly 1 Agent
has_Actor_ OccupiesLink_*-ref exactly 0 Position
has_Actor_ InstanceOfLink_*-ref exactly 1 Actor

A Tabela 9 resume as restrições acrescentadas à *OntoiStar* em uma representação em lógica de 1ª ordem: Ax (instância da classe *Actor*) e conceitos *Internal Elements* como T (*Task*), R (*Resource*), G (*Goal*), S (*Softgoal*), PL (*Plan*), SE (*Service*), PR (*Process*), IE (*InternalElement* ou (T ∨ R ∨ G ∨ S ∨ PL ∨ SE ∨ PR)).

Tabela 9. Representação axiomática referente a cada um dos erros tratados.

Erro 1	A1 ^ (*has_ActorBoundary* ∨ *has_Actor_boundary_elements*) ^ (T ∨ R ∨ G ∨ S ∨ PL ∨ SE ∨ PR ^ ┐ A_2)
Erro 2	A1 ^ (*has_ActorBoundary*) ^ (┐ (*DependencyRelationship* ∨ *InternalElementRelationship))*^ (T ∨ R ∨ G ∨ S ∨ PL ∨ SE ∨ PR ^ A_2)
Erro 3	A1 ^ (*DependencyRelationship* ^ (┐ *InternalElementRelationship))* ^ (T ∨ R ∨ G ∨ S ∨ PL ∨ SE ∨ PR ^ ┐ A_2)
Erro 4	T1 ^ (*has_InternalElement_MeansEndLink**) ^ (G ∨ ┐ (T ∨ R ∨ S ∨ PL ∨ SE ∨ PR ^ A_2))
Erro 5	ActorBoudary ^ ┐ (*has_Dependency_DependerLink_source-ref*) ^ (T ∨ R ∨ G ∨ S ∨ PL ∨ SE ∨ PR)
Erro 6	*(DependeeLink* ∨ *DependerLink*) ^ *has_Dependency_DependeeLink** ^ (┐ A_1)
Erro 7	*ContributionLink* ^ *has_InternalElement_ContributionLink** ^ (S ^ ┐ (T ∨ R ∨ G ∨ PL ∨ SE ∨ PR)
Erro 8	*MeansEndLink* ^ *has_InternalElement_MeansEndLink** ^ (G ^ ┐ (T ∨ R ∨ S ∨ PL ∨ SE ∨ PR)
Erro 9	A1 ^ *has_Actor_ IsALink** ^ (A2 ^ ┐ (Agent ∨ Role ∨ Position) A1 ^ *has_Actor_ PlaysLink* * ^ (Role ^ ┐ (Agent ∨ A2 ∨ Position) A1 ^ *has_Actor_ CoversLink* * ^ (Role ^ ┐ (Agent ∨ A2 ∨ Position) A1 ^ *has_Actor_ OccupiesLink* * ^ (Position ^ ┐ (Agent ∨ A2 ∨ Role) A1 ^ *has_Actor_ InstanceOfLink* * ^ (Agent ^ ┐ (Position ∨ A2 ∨ Role)

3 Conclusões e trabalhos futuros

Há trabalhos que exploram a sintaxe e a semântica da linguagem i*, como a iStarML [12], baseada em XML, e a *OntoiStar*, baseada em OWL. No entanto, essas soluções não resolvem erros típicos em modelos i*, como aqueles envolvidos na definição de atores, dependências, relacionamentos entre atores.

A extensão proposta à *OntoiStar* verifica a consistência de modelos i* à medida em que são criados, não permitindo a criação de modelos com aproximadamente 70% dos erros supracitados. Soma-se ainda a vantagem dessa solução também suportar a interoperabilidade entre variantes da linguagem i*.

Os erros do catálogo que não foram contemplados são mais subjetivos (ex.: uso de nome indequado para ator) e/ou mais difíceis de tratar apenas com a expressividade dos construtores da linguagem OWL. Em função dessa limitação, pretende-se realizar como trabalho futuro a utilização de regras SWRL (*Semantic Web Rule Language*) [10], que estendem a capacidade de expressão e de inferência da linguagem OWL.

AGRADECIMENTOS

Ao Programa Institucional de Capacitação Docente do Instituto Federal de Educação, Ciência e Tecnologia Goiano (PICSS IF Goiano), pelo financiamento da pesquisa.

Referências bibliográficas

1. Lamsweerde, A.V.: Goal-oriented Requirements Engineering: A Guided Tour. In: Fifth IEEE International Symposium on Requirements Engineering, 249-262, (2001)
2. Santos, E.B.: Uma Proposta de Métricas para Avaliar Modelos i*. Dissertação (Mestrado). UFPE, (2008)
3. Horkoff, J., Yu, E.: Detecting Judgment Inconsistencies to Encourage Model Iteration in Interactive i* Analysis. In: Fifth International i* Workshop, p. 20-25, (2011)
4. Souza, C., Souza, C., Alencar, F.M.R., Castro, J., Figueiredo, E., Cavalcanti, P.: Avaliação de Modelos i* com o Processo Airdoc-i. In: ERBR13, 1, (2013)
5. Najera K., Martinez, A., Perini, A., Estrada, H.: An Ontology-Based Methodology for Integrating i* Variants. In: 6h International i* Workshop, 1-6, (2013)
6. Castro, J., Kolp, M., Mylopoulos, J.: Towards Requirements-driven Information Systems Engineering: The Tropos Project. Information Systems, Elsevier Science Ltd., Oxford, UK, v. 27, n. 6, 365-389 (2002)
7. Nardi, J. C., Falbo, R. A.: Uma ontologia de Requisitos de Software. In: CIbSE, 111-124 (2006)
8. Nunes, C. M.: Uma Linguagem de Domínio Específico para a Framework i*. Dissertação (Mestrado). Universidade Nova de Lisboa (2009)
9. Malta, Á., Soares, M., Santos, E., Paes, J., Alencar, F.M.R., Castro, J.: iStarTool: Modeling Requirements Using the i* Framework. In: CEUR Workshop Proceedings, 163-165 (2011)
10. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean. M.: SWRL: A Semantic Web Rule Language, Available online at http://www.w3.org/Submission/SWRL/
11. Guizzardi, R.S.S., Franch, X., Guizzardi, G.: Applying a foundational ontology to analyze means-end links in the i∗ framework. In: Meeting of the RCIS, IEEE, 1-11 (2012)
12. Cares C., Franch, X.: iStarML: Principles and Implications. In: 5th International i* Workshop, 8-13, (2011)
13. Yu, E.: Modelling Strategic Relationships for Process Reengineering. Tese (Doutorado). Canadá: University of Toronto, Department of Computer Science (1995)
14. Souza, C., Alencar, F. M. R., Guedes, G., Soares, M., Souza, C., Ramos, R. A., Castro, J.: AIRDoc-i*: um processo para avaliação de modelos i*. In: 15th Workshop em Engenharia de Requisitos (WER), 1-4, (2012)