Unagi: Uma ferramenta para suporte à Modelagem de Requisitos de Sistemas Adaptativos

César Henrique Bernabé[1], **Pedro Pignaton Negri**[1], **Bruno Borlini Duarte**[1], **André Luiz de Castro Leal**[2], **Renata S. S. Guizzardi**[1], **Vítor E. Silva Souza**[1]

[1]Núcleo de Estudos em Modelagem Conceitual e Ontologias (NEMO)
Departamento de Informática – Universidade Federal do Espírito Santo (UFES)
Av. Fernando Ferrari, 514 – Goiabeiras – Vitória, ES – 29075-910

[2]Departamento de Matemática — Universidade Federal Rural do Rio de Janeiro (UFRRJ)
BR-465, Km 7 – Seropédica – Rio de Janeiro, RJ – 23.897-000

{cesar.hber,pedropn,brunoborlini,andrecastr}@gmail.com
rguizzardi@inf.ufes.br, vitor.souza@ufes.br

Abstract. *Adaptative systems can mold themselves in order to suit different states their requirements can assume. Goal Oriented Requirements Engineering (GORE) can be used to support the development of such a system. Zanshin [Souza 2012] is a requirements-based approach to the development of adaptive systems. To support the design process offered by Zanshin, this paper introduces the Unagi tool, which provides a graphical editor for creating models based on GORE.*

Resumo. *Sistemas adaptativos podem se moldar para atender a diferentes estados que seus requisitos podem assumir. A Engenharia de Requisitos Orientada a Objetivos (Goal-Oriented Requirements Engineering ou GORE) pode apoiar o desenvolvimento desse tipo de sistema. Zanshin [Souza 2012] é uma abordagem para desenvolvimento de sistemas adaptativos baseada em requisitos. Para apoiar o processo de design oferecido pelo Zanshin, introduz-se nesse artigo a ferramenta CASE Unagi, que provê um editor gráfico para criação de modelos de requisitos baseados em GORE.*

Vídeo sobre a ferramenta: `https://youtu.be/MpCwjk37iYs`

1. Introdução

Com o avanço da tecnologia dos computadores nas últimas décadas, identificou-se um aumento relativo no que se refere à complexidade de sistemas de software [Andersson et al. 2009], visto que o crescimento da capacidade computacional propiciou maiores possibilidades na área de programação [Brun et al. 2009]. Além disso, à medida que sistemas vão se tornando mais enraizados no nosso dia-a-dia, acabam encontrando os mais diversos contextos de execução e, assim, faz-se importante a elaboração de softwares que possam identificar quando seus requisitos não estão sendo atendidos e, então, adaptarem-se em tempo de execução a essas situações.

Sistemas adaptativos podem assumir diferentes comportamentos de acordo com alterações de contexto, mediante falhas ou mudanças nos requisitos de desempenho [Souza et al. 2012]. Porém, poucas soluções desse tipo consideram a modelagem

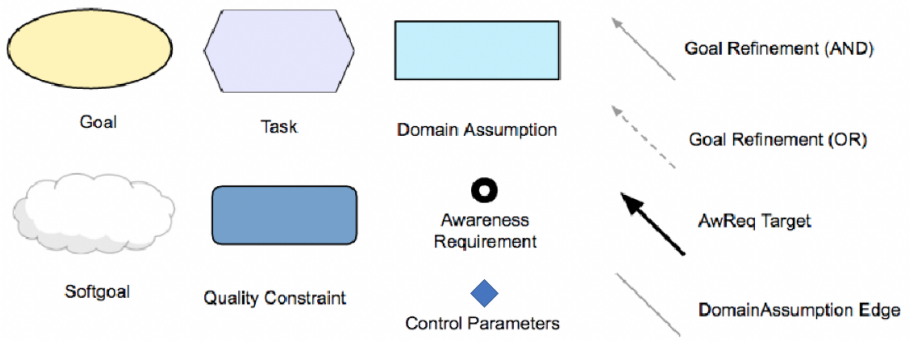

Figura 1. Elementos de um modelo *GORE* e sua representação gráfica.

das características adaptativas desde a fase de Engenharia de Requisitos no processo de software. Dentro desse escopo, destacamos o *Zanshin* [Souza 2012], uma abordagem que baseia-se em modelos de requisitos para projetar características adaptativas em sistemas.

Neste artigo, apresentamos o *Unagi*, uma ferramenta desenvolvida no contexto do *Zanshin*, que provê um ambiente gráfico para modelagem de requisitos usando Engenharia de Requisitos Orientada a Objetivos (*Goal-Oriented Requirements Engineering* ou *GORE*). Esse artigo apresenta os aspectos importantes da ferramenta desenvolvida nesse contexto, bem como uma breve ilustração dos elementos de *GORE* nela usados.

O restante deste artigo está assim dividido: na Seção 2 é resumida a abordagem *Zanshin*; a Seção 3 apresenta o *Unagi* e suas instruções de uso e instalação; na Seção 4 comparamos trabalhos relacionados e, por fim, a Seção 5 conclui o artigo.

2. Referencial Teórico

Zanshin [Souza 2012] é uma abordagem para desenvolvimento de sistemas adaptativos que permite que se especifique, no modelo de requisitos, em quais situações o sistema deve adaptar-se e o que deve fazer para isso. *Zanshin* parte do princípio que sistemas adaptativos possuem, ainda que implícitos ou escondidos, um ciclo de retroalimentação (*feedback loop*) que operacionaliza a adaptação. A proposta do método é fazer com que os elementos desse ciclo se tornem explícitos nos modelos de requisitos utilizados.

A Figura 1 ilustra os elementos comumente utilizados em modelos de objetivos, bem como alguns conceitos introduzidos pela abordagem *Zanshin*. Objetivos (*goals*, representados pelas ovais) são estados do mundo que deseja-se alcançar e podem ser refinados até que sejam operacionalizados por tarefas (*tasks*, hexágonos) que o sistema irá executar ou por pressuposições de domínio (*domain assumptions*, retângulos). Objetivos que não possuem em si critérios exatos de satisfação, denominados *softgoals* (nuvens), são operacionalizados por critérios de qualidade (*quality constraints*, retângulos com cantos arredondados). Refinamentos utilizam lógica Booleana (*AND* ou *OR*) para calcular a satisfação de um elemento "pai" com base nos "filhos".

Além dos elementos tradicionais de *GORE*, os elementos propostos em *Zanshin* são os Requisitos de Percepção (*Awareness Requirements* ou *AwReqs*, círculos) [Souza et al. 2013] e os Requisitos de Evolução (*Evolution Requirements* ou *EvoReqs*, sem representação gráfica) [Souza et al. 2012]. *AwReqs* são requisitos que se referem ao estado assumido por outros requisitos em tempo de execução e são represen-

tados por círculos que apontam para o elemento monitorado. *EvoReqs* descrevem como outros requisitos no modelo devem mudar/evoluir em resposta à falha de um *AwReq*. Por fim, foram adicionados os parâmetros de controle (*control parameters*, losangos) que representam parâmetros do sistema que podem ser reconfigurados para adaptação.

Na Figura 2, pode-se visualizar um exemplo de modelo de requisitos para um sistema de agendamento de reuniões (*meeting scheduler*) [Souza 2012] sendo construído no editor *Unagi*, que será apresentado na próxima seção. O *meeting scheduler* é um sistema responsável por coletar os quadros de horários dos participantes de uma reunião a ser marcada, encontrar uma sala disponível, reservá-la e, por fim, confirmar a presença dos participantes. Nesse contexto, os conceitos de *GORE* são usados para definir o escopo do problema: um objetivo pode ser definido como o estado no qual reuniões são agendadas corretamente e tarefas representam os passos seguidos pelo sistema para cumprir este objetivo, como enviar e-mails aos participantes e efetuar o agendamento da reunião, enquanto pressuposições de domínio representam situações necessárias que se acredita que sejam verdadeiras. Objetivos sem um critério claro de satisfação (*softgoals*), como "boa participação", possuem critérios de qualidade associados que determinam quando são satisfeitos ou não. Qualquer um desses elementos podem ser monitorados pelos requisitos de percepção para garantir que os mesmos sejam atingidos.

Os modelos de objetivos que são usados pelo *Zanshin* devem ser especificados usando a plataforma *Eclipse Modeling Framework* (*EMF*). Atualmente este processo é feito por meio da criação de elementos em um editor gerado automaticamente pelo próprio EMF, baseado no metamodelo de *Zanshin*, que exibe os elementos do modelo de uma forma genérica, similar a uma árvore de diretórios em um gerenciador de arquivos. O *Unagi*, vem substituir tal editor genérico por uma ferramenta mais amigável.

O framework *Zanshin* encontra-se disponível para download e uso na plataforma de controle de versão Github. Em sua wiki,[1] o leitor interessado encontrará informações para execução das simulações de sistemas adaptativos, inclusive a versão completa do modelo mostrado na Figura 2, o *Meeting Scheduler*.

3. O Editor Gráfico *Unagi*

Baseado na tecnologia Sirius [Viyovic et al. 2014], que permite definir representações gráficas para elementos de modelos especificados em *EMF*, o *Unagi* traz uma interface gráfica para criação de modelos baseados em *GORE*, provendo assim suporte ao desenvolvimento de modelos de requisitos ao *Zanshin*. A Seção 3.1 descreve o uso da ferramenta, a Seção 3.2 apresenta sua arquitetura e a Seção 3.3 discute como ela foi avaliada.

3.1. Uso da ferramenta

Para exemplificar o uso da ferramenta, apresentamos como estudo de caso o sistema agendador de reuniões (*meeting scheduler*), que ilustra a proposta do *Zanshin* em [Souza 2012], já descrito na Seção 2. O *meeting scheduler* foi escolhido por ser um exemplo conhecido na comunidade de Engenharia de Requisitos [van Lamsweerde et al. 1995] e para que pudéssemos comparar a saída produzida pelo

[1] `https://github.com/sefms-disi-unitn/Zanshin/wiki`.

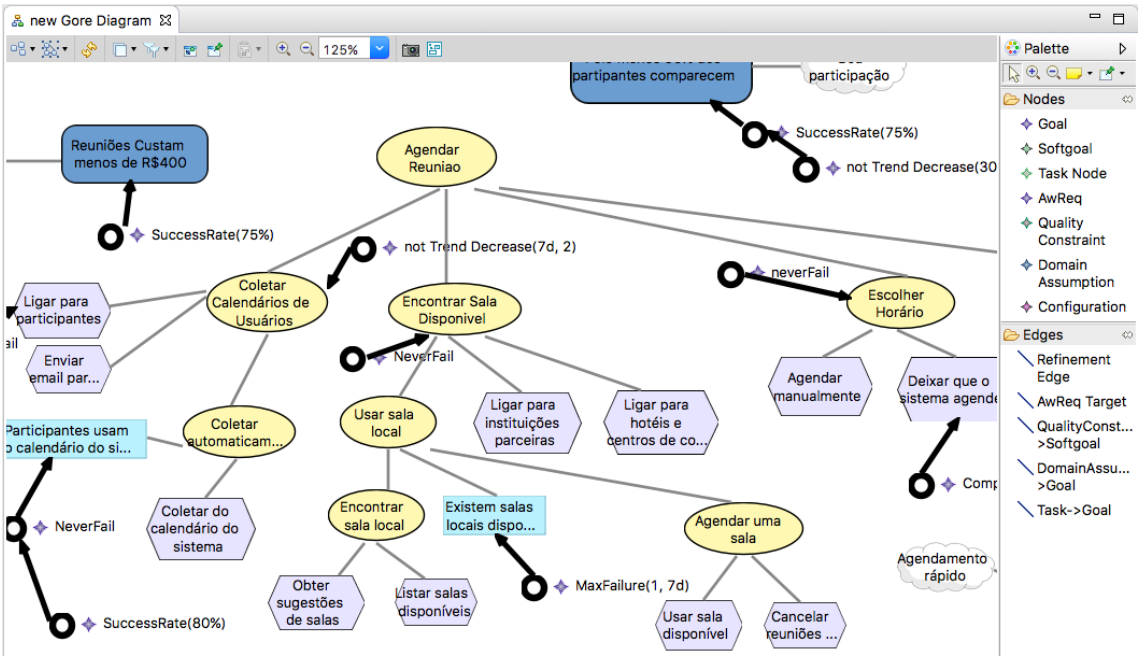

Figura 2. Interface principal do *Unagi* durante a construção do modelo *GORE* do sistema de agendamento de reuniões.

Unagi com os arquivos utilizados pelo *Zanshin* na simulação do sistema, disponível no repositório de controle de versões do framework.[2]

A Figura 2 também mostra a interface principal do *Unagi*. A partir desta tela, o modelador pode incluir elementos do modelo simplesmente arrastando o item desejado da paleta de componentes para a região do diagrama, bem como definir as associações entre os componentes selecionando o tipo de relação disponível na mesma paleta e clicando nos elementos origem e destino, nesta ordem. Nessa contexto, algumas checagem de consistência (aderência ao metamodelo) são feitas em tempo de modelagem. Todos os elementos do modelo original do *meeting scheduler* foram replicados no *Unagi*. As características de cada componente foram definidas por meio da paleta de opções mostrada na parte inferior da tela. Essa paleta mostra configurações específicas para o elemento selecionado, dependendo do seu tipo. Nela é possível definir características como nome, alvo, objetivo pai, dentre outros.

Os *EvoReqs* inerentes a cada *AwReq* são especificados por meio de uma funcionalidade particular a esse tipo: ao clicar sobre um *AwReq*, o editor oferece a opção de criação de um subdiagrama de detalhamento dos elementos, exemplificado na Figura 3. Assim, o usuário pode determinar as estratégias de adaptação próprias a cada *AwReq* usando a mesma lógica de arrastar e soltar.

Construído o modelo gráfico, o usuário pode acionar o conversor do *Unagi*, que transforma os arquivos criados pelo *Sirius* para a especificação do diagrama em arquivos *XML* que podem ser lidos pelo *Zanshin*. Note que o Sirius também salva os arquivos que representam os diagramas criados na ferramenta em formato *XML*. No entanto, a sintaxe destes arquivos é diferente da utilizada pelo EMF e, consequentemente, por *Zanshin*. Por

[2] `https://goo.gl/oooAGp`.

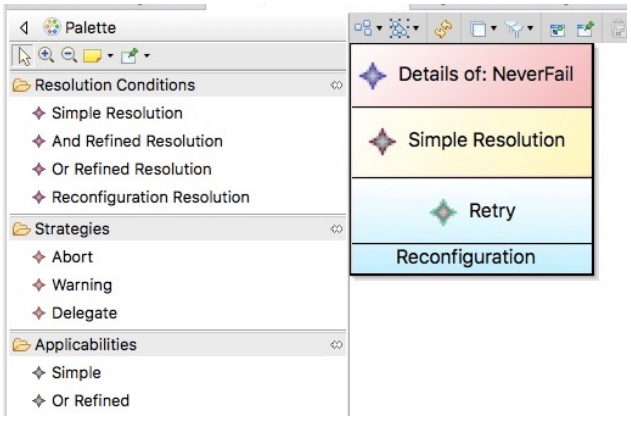

Figura 3. Interface para especificação dos *EvoReqs* de um *AwReq*.

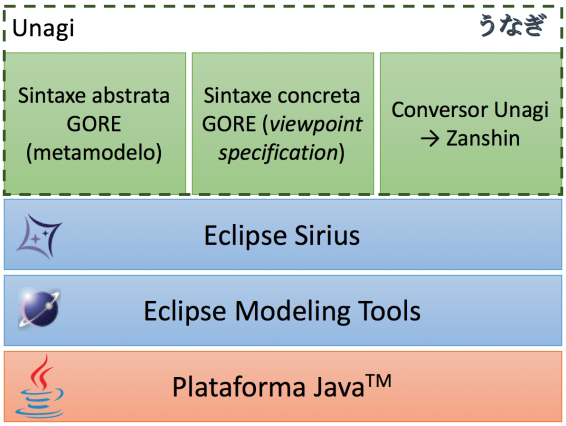

Figura 4. Descrição da arquitetura da ferramenta.

isso se faz necessário o processo de conversão. Para executá-lo, o usuário deve clicar com o botão direito em qualquer área livre do diagrama e selecionar "Converter → Convert Diagram".

Deste modo, o usuário pode importar o arquivo no *Zanshin* e rodar as simulações a partir do diagrama construído, ilustrando a principal proposta do *Unagi*: facilitar a criação de modelos *GORE* para uso no *Zanshin*. Inicialmente só era possível criar esses modelos por meio do editor *EMF*, tarefa que poderia ser trabalhosa e complicada para usuários que não tivessem experiência com esse procedimento, já que além de demandar um trabalho maior para criação, fica difícil visualizar de forma trivial as relações e hierarquias entre os elementos, pois todos são representados em uma lista simples. Como efeito, isso poderia desencorajar desenvolvedores de sistemas adaptativos a usarem o *Zanshin*.

3.2. Arquitetura da ferramenta

O *Unagi* foi construído em cima do plug-in *Sirius*, que por sua vez tem como base as ferramentas de modelagem da plataforma Eclipse™ (dentre as quais se destaca o *Eclipse Modeling Framework*, ou *EMF*), baseada em Java™, como ilustrado na Figura 4. Em sua página Web[3] é possível encontrar instruções de instalação e uso da ferramenta.

[3]https://github.com/hbcesar/unagi2/wiki/

Seguindo o paradigma de Desenvolvimento Orientado a Modelos [Embley et al. 2011], a construção de uma ferramenta de modelagem gráfica utilizando o *Sirius* passa pela definição de dois modelos relacionados ao domínio em questão (em nosso caso, GORE). O modelo da sintaxe abstrata, ou metamodelo, define os elementos que poderão ser criados no modelador gráfico, bem como suas propriedades e relações. O modelo da sintaxe concreta, chamado pelo *Sirius* de *viewpoint specification*, define as características gráficas dos elementos definidos no metamodelo. Ambos são especificados utilizando a sintaxe do *EMF*.

O último componente do *Unagi* é o conversor responsável por transformar o modelo gráfico em um arquivo no formato compatível com o *Zanshin*, também baseado em *EMF*. O conversor foi construído utilizando a linguagem Java™ com o apoio de bibliotecas de manipulação de arquivos *XML*, como as bibliotecas do pacote javax.xml.parsers. Por fim, tem-se a integração do editor gráfico com o conversor por meio da implementação de uma Chamada Java Externa (*External Java Call*), uma funcionalidade do plug-in *Sirius* que permite a invocação de serviços externos contendo código escrito em Java™.

3.3. Avaliação

O arquivo *XML* gerado pelo conversor para o caso de uso utilizado encontra-se disponível no repositório da ferramenta,[4] bem como o arquivos *XML* da simulação original do *Zanshin* estão no repositório do *framework*.[5] O leitor interessado pode comparar os arquivos para verificar que são, de fato, compatíveis. Para avaliar o funcionamento do *Unagi*, usamos o editor para modelar outros dois sistemas utilizados originalmente em [Souza 2012] (despacho de ambulâncias e caixa eletrônico) e comparamos os arquivos gerados pelo editor com os modelos originais, com resultados satisfatórios.

4. Trabalhos relacionados

A ferramenta *GATO* [Pimentel 2015] também permite gerar arquivos *EMF* para o *Zanshin* a partir do modelo criado pelo usuário em uma ferramenta gráfica, porém baseada na Web, ao contrário do *Unagi*, que é voltado para *desktop*. Ambas trabalham com uma linguagem *GORE* e suportam as extensões de adaptação propostas por *Zanshin*. Por ser construída na plataforma Web, *GATO* pode parecer mais simples e intuitiva a priori. O *Unagi*, porém, permite que o utilizador crie subdiagramas para descrever os elementos envolvidos no ciclo de retroalimentação (de adaptação), o que permite um melhor gerenciamento de diagramas de grandes proporções. Em *GATO*, as propriedades dos elementos de adaptação e evolução devem ser especificados na forma de tabela simples. Ademais, os parâmetros recorrentes a cada *AwReq* são específicos e, portanto, em *Unagi* estão representados na paleta de elementos, o que restringe o usuário ao correto uso dos mesmos, enquanto *GATO* oferece apenas uma caixa de texto na qual os parâmetros devem ser especificados, aumentando as chances de erros devido a especificação incorreta, seja por erro de digitação ou por indicação de um parâmetro não existente. Além disso, *GATO* só oferece a opção de exportar para o *Zanshin* em um XML contendo o metamodelo do diagrama, porém não exporta o XML com refinamentos e com as instruções de adaptação, por exemplo.

Existem algumas ferramentas de modelagem *GORE* de propósito geral (i.e., não direcionadas a construção de sistemas adaptativos) mas, ainda assim, relacionadas.

[4] `https://github.com/hbcesar/unagi`.

[5] `https://goo.gl/sNHht1`.

TAOM4e [Morandini et al. 2007] visa apoiar a elicitação e modelagem de requisitos baseada na metodologia *Tropos*. Por englobar o processo de elicitação de requisitos até a analise em tempo de execução, é justo que o *TAOM4e* seja comparado ao conjunto *Unagi* + *Zanshin*. A principal diferença entre eles é que enquanto *TAOM4e* é baseada em *Tropos*, *Unagi* + *Zanshin* não são baseados em nenhuma linguagem específica, mas nos conceitos comuns às várias abordagens *GORE*.

A ferramenta *OpenOME* [Horkoff et al. 2011] é voltada ao desenvolvimento de modelos na linguagem $i^\star$ e também é fundamentada em *EMF*, essa ferramenta possui interface muito semelhante ao *Unagi*, como os recursos de arrastar elementos a partir de uma paleta de opções, bem como apresenta um processo de instalação e configuração muito parecidos. Entretanto, destoa da proposta principal do *Unagi*, pois não oferece suporte a modelagem de sistemas adaptativos, como os elementos do ciclo de retroalimentação propostos por *Zanshin* [Souza et al. 2012] e usados em *Unagi*.

5. Conclusões

Os aspectos importantes da ferramenta *Unagi* podem ser divididos em dois: a interface de modelagem e o conversor para arquivos escritos com sintaxe adotada por *Zanshin* [Souza 2012]. Na primeira, temos como principal ponto positivo o fato de a interface focar inteiramente nos elementos de *GORE* e nos elementos de adaptação, adotando a mesma simbologia proposta pelo *framework*. Portanto, espera-se que usuários já familiarizados com *Zanshin* sintam-se confortáveis com a ferramenta em seus primeiros momentos de uso e que novos usuários possam rapidamente se habituar tanto com o *framework* quanto com a ferramenta *CASE*.

O conversor integrado à ferramenta tem, como único objetivo, transformar o modelo atual e todos os submodelos de *AwReqs* em arquivos *XML* com sintaxe específica para uso no *Zanshin*. Tal funcionalidade facilita bastante o processo de especificação do modelo de objetivos para uso com o *framework*.

Encontra-se em desenvolvimento: (a) a integração efetiva do *Zanshin* com o *Unagi*, de modo que este último permita que o usuário chame o framework direto do editor de modelos ao fim da edição do diagrama (atualmente ele precisa executar o *Zanshin* em uma instância separada); e (b) a revisão dos meta-modelos do *Unagi* (e, consequentemente, do *Zanshin*), baseando-os em ontologias bem fundamentadas como a *Runtime Requirements Ontology* [Duarte et al. 2016] e a *Goal-Oriented Requirements Ontology* [Negri et al. 2017]. Tal trabalho visa garantir ainda mais que a ferramenta promova a construção de modelos válidos e que façam sentido no mundo real.

Conforme mencionamos no início do artigo, o uso de sistemas adaptativos aparece como solução para a atual conjuntura enfrentada pela Engenharia de Software para o desenvolvimento de sistemas complexos e distribuídos. O *Unagi* tem como essência o apoio ao desenvolvimento de sistemas desse tipo, pois acredita-se que esse campo de pesquisa tornar-se-à cada vez mais promissor.

Referências

[Andersson et al. 2009] Andersson, J., De Lemos, R., Malek, S., and Weyns, D. (2009). Modeling dimensions of self-adaptive software systems. In *Software engineering for self-adaptive systems*, pages 27–47. Springer.

[Brun et al. 2009] Brun, Y., Serugendo, G. D. M., Gacek, C., Giese, H., Kienle, H., Litoiu, M., Müller, H., Pezzè, M., and Shaw, M. (2009). Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems*, pages 48–70. Springer.

[Duarte et al. 2016] Duarte, B. B., Souza, V. E. S., Leal, A. L. d. C., Falbo, R. A., Guizzardi, G., and Guizzardi, R. S. S. (2016). Towards an Ontology of Requirements at Runtime. In *Proc. of the 9th International Conference on Formal Ontology in Information Systems*, volume 283, pages 255–268, Annecy, France. IOS Press.

[Embley et al. 2011] Embley, D. W., Liddle, S. W., and Pastor, O. (2011). Conceptual-Model Programming: A Manifesto. In Embley, D. W. and Thalheim, B., editors, *Handbook of Conceptual Modeling*, pages 3–16. Springer Berlin Heidelberg.

[Horkoff et al. 2011] Horkoff, J., Yu, Y., and Eric, S. (2011). OpenOME: An Open-source Goal and Agent-Oriented Model Drawing and Analysis Tool. *iStar*, 766:154–156.

[Morandini et al. 2007] Morandini, M., Nguyen, D. C., Perini, A., Siena, A., and Susi, A. (2007). Tool-supported development with tropos: The conference management system case study. In *International Workshop on Agent-Oriented Software Engineering*, pages 182–196. Springer.

[Negri et al. 2017] Negri, P. P., Souza, V. E. S., Leal, A. L. d. C., Falbo, R. A., and Guizzardi, G. (no prelo, 2017). Towards an Ontology of Goal-Oriented Requirements. *Ibero-American Conference on Software Engineering.*

[Pimentel 2015] Pimentel, J. H. C. (2015). *Systematic design of adaptive systems: control-based framework*. PhD thesis, Universidade Federal de Pernambuco, Pernambuco, PE, Brasil.

[Souza 2012] Souza, V. E. S. (2012). *Requirements-based Software System Adaptation*. Phd thesis, University of Trento, Italy.

[Souza et al. 2012] Souza, V. E. S., Lapouchnian, A., and Mylopoulos, J. (2012). (Requirement) evolution requirements for adaptive systems. In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 155–164. IEEE Press.

[Souza et al. 2013] Souza, V. E. S., Lapouchnian, A., Robinson, W. N., and Mylopoulos, J. (2013). Awareness requirements. In *Software Engineering for Self-Adaptive Systems II*, pages 133–161. Springer.

[van Lamsweerde et al. 1995] van Lamsweerde, A., Darimont, R., and Massonet, P. (1995). Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt. In *Proc. of the 2nd IEEE International Symposium on Requirements Engineering*, pages 194–203, York, England. IEEE.

[Viyovic et al. 2014] Viyovic, V., Maksimovic, M., and Perisic, B. (2014). Sirius: A rapid development of DSM graphical editor. In *Intelligent Engineering Systems (INES), 2014 18th International Conference on*, pages 233–238. IEEE.