César Henrique Bernabé

GORO: uma ontologia sobre requisitos orientados a objetivos.

Vitória, ES

2020

César Henrique Bernabé

GORO: uma ontologia sobre requisitos orientados a objetivos.

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Programa de Pós-Graduação em Informática

Orientador: Prof. Dr. Vítor E. S. Souza

Coorientador: Profa. Dra. Carla T. L. L. S. Schuenemann

Vitória, ES

2020

César Henrique Bernabé
GORO: uma ontologia sobre requisitos orientados a objetivos./ César Henrique Bernabé. – Vitória, ES, 2020-
77 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Vítor E. S. Souza

Dissertação de Mestrado – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Programa de Pós-Graduação em Informática, 2020.

1. Ontologia. 2. Engenharia de Requisitos Orientada a Objetivos. I. Souza, Vítor Estêvão Silva. II. Universidade Federal do Espírito Santo. IV. GORO: uma ontologia sobre requisitos orientados a objetivos.

CDU 02:141:005.7

César Henrique Bernabé

GORO: uma ontologia sobre requisitos orientados a objetivos.

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Trabalho aprovado. Vitória, ES, 16 de março de 2020:

Prof. Dr. Vítor E. S. Souza
Orientador

Prof. Dr. João Paulo A. Almeida
Universidade Federal do Espírito Santo

Prof. Dr. Enyo José T. Gonçalves
Universidade Federal do Ceará

Vitória, ES
2020

Aos meus pais, por me apoiarem e acreditarem na minha capacidade, e aos meus orientadores, por tudo que me ensinam e inspiram.

Agradecimentos

O mestrado foi uma etapa da minha vida que me abriu os olhos. Aqui eu pude perceber o meu apreço pela área acadêmica. Foi também nesta etapa que eu tive a oportunidade de amadurecer, profissional e pessoalmente. Com certeza o maior aprendizado que eu tive, é que eu ainda tenho muito a aprender. Todo esse processo só foi possível graças as pessoas que me cercaram.

Começo agradecendo ao meu orientador, Vítor. Muito obrigado pelos ensinamentos, pela paciência, pela insistência e por sua postura perante o mundo. Você é um exemplo para mim. Agradeço também a minha coorientadora, Carla. Sua atenção aos detalhes e positividade me ajudaram a querer fazer sempre o melhor, sem pensar em desistir.

Agradeço também aos amigos e professores do Nemo, cada um de vocês deu alguma contribuição para eu chegar aqui. Aos professores agradeço pelos ensinamentos, pela qualidade das aulas e do conteúdo ensinado, vocês são ótimos. Aqui cabe um agradecimento especial a professora Renata, muito obrigado por todo conhecimento compartilhado e por todos os conselhos. Aos amigos, obrigado pela ajuda, seja com a pesquisa ou com as cervejinhas do final da semana, é bom saber que posso contar com vocês e vocês comigo.

A minha família, pai, mãe e irmã. Por tudo que vocês fizeram por mim. Obrigado por estarem sempre ao meu lado, por apoiarem minhas escolhas e por me ensinarem o que é o amor. Ao Caíque, obrigado pelo companheirismo, pela paciência e por me apoiar em todos os momentos.

Por fim, aos amigos de fora da Ufes e a todos os que, de alguma forma, contribuíram e me ajudaram nesta etapa. Muito obrigado!

"O amor em primeiro lugar realmente ensina um homem
a acreditar no mundo objetivo fora de si mesmo.
Não só faz do homem o objeto, mas o objeto de um homem."
(Karl Marx)

Resumo

A correta especificação dos requisitos garante que softwares sejam construídos para resolver o problema identificado e previne erros que, se identificados somente em fase de produção, podem apresentar impacto no custo até 90% maior do que em fases anteriores. Diversas abordagens buscam melhorar o processo de elicitação e especificação de requisitos. Dentre elas, destaca-se a Engenharia de Requisitos Orientada a Objetivos (*Goal-Oriented Requirements Engineering*, ou GORE), que apresenta vantagens perante métodos tradicionais de Engenharia de Requisitos.

Diversas abordagens foram propostas desde que a área de GORE começou a ganhar destaque. Entretanto, poucas dessas abordagens foram construídas com base em artefatos bem fundamentados e com preocupação em relação à semântica dos conceitos. Assim, este trabalho apresenta a *Goal-Oriented Requirements Ontology* (GORO), uma ontologia baseada em UFO (*Unified Foundational Ontology*) e integrada à SEON (*Software Engineering Ontology Network*), que busca definir de forma consensual e precisa a semântica usada por abordagens GORE. A GORO pode ser utilizada, por exemplo, para análise ontológica de linguagens existentes de modelagem de objetivos, como interlíngua para conversão entre modelos de linguagens diferentes e como base para construção de novas linguagens.

Palavras-chaves: ontologia, objetivos, requisitos.

Abstract

Correctly specifying requirements ensures that the right software is built to solve the right problem, and prevents errors that, if identified only in the development phase, can cost up to 90% more than if they were identified in earlier phases. Several approaches seek to improve the requirements elicitation process. Among them, the Goal-Oriented Requirements Engineering (GORE) offers advantages over traditional Requirements Engineering methods.

Several approaches have been proposed since GORE emerged. However, almost none of these approaches were built on well-grounded artifacts or concerning the semantics of constructs. Thus, this work presents the *Goal-Oriented Requirements Ontology* (GORO), an ontology based on the *Unified Foundational Ontology* (UFO) and integrated with the *Software Engineering Ontology Network* (SEON). GORO intends to precisely define the semantics used by GORE approaches. GORO can be used, for example, for modeling languages analysis, as an interlanguage for model conversion and as a basis for building new GORE languages.

Keywords: ontology, goal, requirements.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

UFO	Unified Foundational Ontology
GORE	Goal Oriented Requirements Engineering
GORO	Goal Oriented Requirements Ontology
ASMP	Assumptions Ontology
NFRO	Non-functional Requirements Ontology
COVR	Common Ontology of Value and Risk
SPO	Software Process Ontology
RSRO	Reference Software Requirements Ontology
FR	Functional Requirement
NFR	Non-functional Requirement
NFRF	Non-functional Requirement Framework
KAOS	Keep All Objects Satisfied ou KnowledgeAcquisition in autOmated Specification
GBRAM	Goal-Based Requirements Acquisition Method
GSN	Goal Structure Notation
ER	Engenharia de Requisitos
GCT	GORE Conversion Tool
SABiO	Systematic Approach for Building Ontologies

Sumário

1 Introdução

A construção de um software é um processo delicado que precisa ser planejado de forma precisa. É muito importante entender bem as necessidades dos *stakeholders* para construir a solução correta para um determinado problema. Estima-se que o custo da correção de erros durante a fase de requisitos seja 90% menor do que o custo estimado para a correção de erros na fase de produção (NASA; SPACE, 2018).

Diversas abordagens surgem para melhorar a qualidade dos requisitos elicitados para um software. Dentre elas, destaca-se a Engenharia de Requisitos Orientada a Objetivos (GORE, do inglês, *Goal-Oriented Requirements Engineering*). Esta abordagem baseia-se no conceito de objetivo como um estado que se deseja atingir (LAMSWEERDE, 2001). Em torno deste conceito, propõe-se o uso de outros construtos, como tarefas, qualidades e conflitos, que permitem que objetivos sejam refinados e descritos com precisão, para então serem negociados e validados.

GORE apresenta diversas vantagens em relação à Engenharia de Requisitos (ER) tradicional. Objetivos, por exemplo, explicam as razões por trás de requisitos, pois consideram não somente o software *per se*, mas também o contexto no qual ele está inserido. Objetivos também proveem critérios claros para verificação da satisfação de requisitos, são úteis para identificação e negociação de conflitos, são utilizados para manter registro do raciocínio usado para se refinar requisitos de nível mais alto em requisitos de nível mais específico e são ótimas ferramentas para análise de soluções alternativas ao mesmo problema (LAMSWEERDE, 2001).

1.1 Motivação

Dadas as vantagens trazidas por GORE, diversos *frameworks*, métodos e linguagens de modelagem foram propostos nos últimos 25 anos (HORKOFF et al., 2016). Entretanto, poucas propostas foram definidas levando em consideração a semântica dos construtos propostos. Assim, diversas abordagens GORE, como KAOS (DARDENNE; LAMSWEERDE; FICKAS, 1993) e *i** (YU, 1996), apresentam conceitos com definições sobrecarregadas, ambíguas ou vagas. Esses conceitos serão discutidos ao longo deste texto.

Em outras palavras, mesmo assumindo cada vez mais importância dentro da área de Engenharia de Requisitos, GORE carece de um artefato para definir conceitos comuns a esta área de forma bem fundamentada, consensual e precisa. Entende-se que uma conceituação é consensual quando esta é aceita por especialistas de domínio de uma determinada comunidade, e então utilizada por esta comunidade.

Por outro lado, ontologias vem sendo utilizadas com sucesso como instrumento de conceituação consensual para comunicação humana, provendo semântica e interoperabilidade entre entidades com naturezas distintas (GUARINO; OBERLE; STAAB, 2009; GUIZZARDI, 2007). Se apresentam, portanto, como possível ferramenta para suprir a lacuna identificada ao se analisar as diferentes linguagens GORE propostas na literatura, visto que diversas linguagens, ao não considerarem a semântica de seus construtos, acabam tendo problemas em relação aos seguintes fatores (MOODY, 2009):

- **Déficit de construto:** não há um construto que pode ser utilizado para representar um determinado conceito;
- **Sobrecarga de construto:** um único construto é utilizado para representar múltiplos conceitos;
- **Redundância de construto:** um mesmo conceito é representado por mais que um construto;
- **Excesso de construto:** um construto não representa algum conceito;
- **Interoperabilidade:** um mesmo elemento é definido de formas diferentes de uma linguagem para outra, impedindo que usuários utilizando linguagens de modelagem diferentes possam dialogar.

A Figura 1 exemplifica os problemas descritos por Moody (2009). Estes problemas são identificados por meio dos Princípios de Clareza Semiótica, propostos com base na teoria dos símbolos (GOODMAN, 1976), a qual estabelece que uma linguagem bem descrita deve definir uma correspondência de um pra um entre conceitos e construtos de uma linguagem.

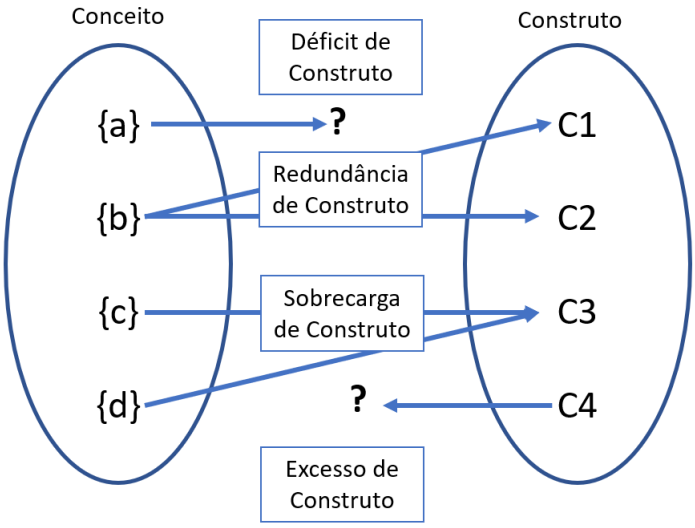


Figura 1 – Exemplos de problemas sintáticos de linguagens de modelagem.

1.2 Objetivos

O objetivo deste trabalho é construir uma ontologia que sirva como um artefato de comunicação eficiente para interessados em GORE, podendo assim ser utilizada para análise e evolução das linguagens atuais, permitindo que os problemas identificados sejam corrigidos com base em conceituação bem fundamentada.

Nomeada GORO (do inglês, *Goal-Oriented Requirements Ontology*), a ontologia também poderá ser usada para delimitar claramente o escopo de GORE; verificar e validar construções de linguagens de modelagem de objetivos; permitir que linguagens de modelagem especifiquem claramente a semântica de seus construtos, fundamentando-os em uma ontologia de referência formal; e ser usada como um interlíngua entre diferentes linguagens de modelagem, ou seja, como ferramenta de interoperabilidade entre modelos em diferentes linguagens GORE.

De forma mais detalhada, os objetivos deste trabalho são:

- Estudo de linguagens de modelagem de objetivos existentes e identificação de conceitos relevantes;
- Construção de uma ontologia sobre o domínio de GORE;
- Validação, verificação e avaliação da ontologia utilizando métodos pré-estabelecidos;
- Uso da ontologia na prática como forma de prova de conceito;
- Avaliação de linguagens existentes com suporte da ontologia.

1.3 Etapas de Realização da Dissertação

As seguintes atividades foram conduzidas para satisfação do objetivo deste trabalho:

Entendimento do domínio. Com base em literatura disponível, foi realizado estudo de diferentes abordagens GORE a fim de entender o domínio a ser modelado. As abordagens foram escolhidas de acordo com artigos relevantes dentro da área, principalmente revisões de literatura. A partir deste estudo, foram identificados e delimitados o escopo de GORE.

Construção e Formalização da Ontologia. Após o estudo do domínio, a ontologia foi desenvolvida com suporte de um método de Engenharia de Ontologias. É importante utilizar métodos maduros para guiar a construção de ontologias pois estes possuem etapas que verificam e validam o resultado obtido. GORO foi desenvolvida com base nas diretivas propostas por SABiO (FALBO, 2014), escolhido por ter sido utilizado com sucesso na construção de diversas ontologias no domínio de Engenharia de Software (RUY et al., 2016), incluindo algumas ontologias reutilizadas neste trabalho.

A ontologia foi fundamentada em UFO (*Unified Foundational Ontology*) (GUIZZARDI, 2005), que define conceitos gerais sobre o mundo real. O uso de ontologias de fundamentação traz diversas vantagens para processos de construção de ontologias, pois elas: definem conceitos que são utilizados para definir outros conceitos mais específicos, não sendo necessário ter que "reinventar a roda"; aumentam a qualidade de ontologias de domínio ao usar conceitos bem definidos e validados; guiam a construção de outras ontologias; e facilitam a interoperabilidade entre ontologias alinhadas à mesma ontologia de fundamentação.

Verificação e Validação da Ontologia. Tendo a primeira versão da ontologia construída, ela foi submetida aos processos de verificação e validação propostos por SABiO, bem como a seu uso em prática através de uma versão operacional da ontologia, que foi utilizada em uma ferramenta de conversão de modelos, artefato para modelagem GORE. Processos de validação e verificação verificam se a ontologia está correta e representa o domínio de forma adequada. Visto que ontologias buscam representar um domínio de forma consensual, processos que verificam sua completude e corretude são cruciais. Avaliar uma ontologia em casos práticos permite que sejam verificados e identificados problemas que passam despercebidos por outros métodos de verificação e validação. Além disso, o uso em prática permite aferir a cobertura de domínio da ontologia, assim como sua capacidade de interoperar modelos, no caso de GORE. Estes processos ocorreram de forma iterativa com o processo de construção e formalização da ontologia, já que a verificação e validação ajudam a identificar pontos que precisam ser reajustados.

Escrita de artigos e da dissertação. Com os resultados obtidos, foi escrita esta dissertação de mestrado. Este trabalho deu origem a duas publicações: em (BERNABÉ et al., 2019b), apresentamos a GORO como ontologia sobre o domínio GORE. Já em (BERNABÉ et al., 2019a), utilizamos GORO para fazer análises de construtos de uma linguagem de modelagem GORE (iStar).

1.4 Organização da Dissertação

O restante desta dissertação está organizado da seguinte maneira:

- **Capítulo 2 – Referencial Teórico.** Neste capítulo é apresentado o embasamento teórico utilizado como referência neste trabalho. A Engenharia de Requisitos Orientada a Objetivos é introduzida de forma geral e, então, são descritas abordagens GORE propostas dentro deste contexto. Também neste capítulo são discutidas a UFO, ontologia de fundamentação utilizada como base para a GORO, e outras três ontologias reutilizadas pela GORO. Por fim, são descritas as atividades propostas por SABiO para processos de Engenharia de Ontologias;

- **Capítulo 3 – Proposta.** Neste capítulo, é detalhado o processo de construção da GORO. A ontologia propriamente dita é apresentada e descrita em detalhes;
- **Capítulo 4 – Avaliação.** Neste capítulo são descritos os processos de verificação, validação e prova de conceito aos quais a GORO foi submetida. Neste capítulo também é descrito um experimento de modelagem realizado no contexto deste trabalho para verificar a cobertura da ontologia. Por fim, o processo de avaliação de GORO permitiu levantar alguns aspectos relevantes em relação aos construtos dessas linguagens também são discutidos;
- **Capítulo 5 – Conclusão.** Este capítulo apresenta as conclusões obtidas neste trabalho, compara trabalhos relacionados e, por fim, lista perspectivas futuras.

2 Referencial Teórico

A Seção 2.1 descreve o contexto de Engenharia de Requisitos Orientada a Objetivos (*Goal-Oriented Requirements Engineering* ou GORE), apresentando, em suas subseções, diferentes abordagens GORE. A Seção 2.2 descreve a *Unified Foundational Ontology*, ontologia de fundamentação utilizada para construção da GORO. A Seção 2.3 descreve as ontologias que foram reutilizadas na elaboração da GORO. Ao final, a Seção 2.4 apresenta o SABiO. Alguns trabalhos relacionados foram identificados no contexto desta pesquisa, e são apresentados e comparados na Seção 2.5.

2.1 Engenharia de Requisitos Orientada a Objetivos

A Engenharia de Requisitos Orientada a Objetivos (*Goal-Oriented Requirements Engineering* ou GORE) é uma subárea da Engenharia de Requisitos. GORE introduz um novo paradigma aos processos de levantamento e análise de requisitos ao se basear em dois conceitos principais: objetivo e contexto. Objetivos são definidos como um estado da realidade que se deseja atingir e são importantes pois justificam a necessidade de cada funcionalidade de um determinado sistema (LAMSWEERDE, 2001). Diferentemente da ER tradicional, GORE passa a considerar também o contexto no qual o software está inserido, analisando objetivos organizacionais e atores necessários para cumprir estes objetivos (LAPOUCHNIAN, 2005).

Objetivos podem ser modelados em diferentes níveis de especificidade, onde objetivos de nível mais alto abordam preocupações estratégicas, enquanto os de nível mais baixo lidam com preocupações operacionais. Ademais, objetivos cobrem requisitos funcionais, geralmente focados em serviços a serem implementados, e não funcionais, geralmente ligados à qualidade destes serviços (LAMSWEERDE, 2001). O processo de levantamento de objetivos consiste na análise dos ambientes organizacional (o contexto geral em que o software está inserido), operacional (os processos organizacionais que serão apoiados pelo sistema a ser desenvolvido) e técnico (o ambiente tecnológico no qual o software será instalado). Os objetivos levantados são então refinados até que sejam detalhados o suficiente para serem operacionalizados. Esse processo permite que sejam identificados casos de conflitos entre objetivos, obstáculos (situações que podem impedir a realização de um objetivo) e soluções alternativas (LAPOUCHNIAN, 2005).

Requisitos orientados a objetivos geralmente são representados em modelos gráficos. Outras formas de representação podem ser textual, em tabelas ou em lógica formal. De forma geral, modelos de objetivos apresentam diversos elementos além dos objetivos em si:

- **Atores:** componente de um sistema ou do ambiente de execução do sistema, capaz de executar ações com a finalidade de atender seus objetivos (YU, 1996);
- **Refinamentos:** relações entre elementos que descrevem diferentes graus de detalhamento de objetivos, denotando que um objetivo refinado é realizado por seus refinamentos (YU, 1996);
- **Softgoals:** objetivos sem critérios claros de satisfação, geralmente atrelados a questões de qualidade e úteis para comparar diferentes alternativas de solução (LAMSWEERDE, 2001);
- **Restrições de Qualidade:** definem valores mensuráveis a características não mensuráveis do sistema, como qualidades não definidas quantitativamente (ex.: "Interface amigável", "Carregamento rápido", etc.) (BORGIDA et al., 2010);
- **Tarefas:** funcionalidades que, ao serem executadas com sucesso, produzem um estado da realidade em que os objetivos são atingidos (LAMSWEERDE, 2001);
- **Recursos:** entidades físicas ou informacionais, produzidas ou utilizadas por uma tarefa (YU, 1996);
- **Pressuposições de Domínio:** afirmações sobre a realidade, geralmente definida por normas organizacionais, leis da física, etc. (LAMSWEERDE, 2001);
- **Contribuições:** denotam o grau e a intensidade em que um determinado objetivo contribui para a satisfação de uma qualidade (YU, 1996);
- **Conflitos:** relações entre objetivos conflitantes, ou seja, o sistema não pode alcançar um estado em que seja possível satisfazer os dois objetivos simultaneamente (RESPECT-IT, 2007);
- **Obstáculos:** estados indesejados da realidade que podem prejudicar a realização de um determinado objetivo (SALES et al., 2018).

Dadas as vantagens percebidas dentro do campo de GORE, diversas abordagens foram propostas. Em 1992, o NFR Framework (MYLOPOULOS; CHUNG; NIXON, 1992) foi a primeira abordagem conhecida a definir-se como orientada a objetivos. Um ano após, KAOS (DARDENNE; LAMSWEERDE; FICKAS, 1993) foi proposto, seguido por *i** (YU, 1996) em 1995, GBRAM (ANTON, 1996) em 1996, GSN (KELLY; WEAVER, 2004) e Tropos (BRESCIANI et al., 2004) em 2004, Techne em 2009, GLR (AMYOT et al., 2010) em 2010 e, finalmente, uma segunda versão do *i**, renomeada para iStar (DALPIAZ; FRANCH; HORKOFF, 2016), foi publicada em 2016. As próximas subseções discutem estas abordagens em detalhe. Estas linguagens foram identificadas com base em dois trabalhos

conhecidos na literatura: a revisão sistemática em abordagens GORE (HORKOFF et al., 2016) e o trabalho de *Van Lamsweerde* (LAMSWEERDE, 2001).

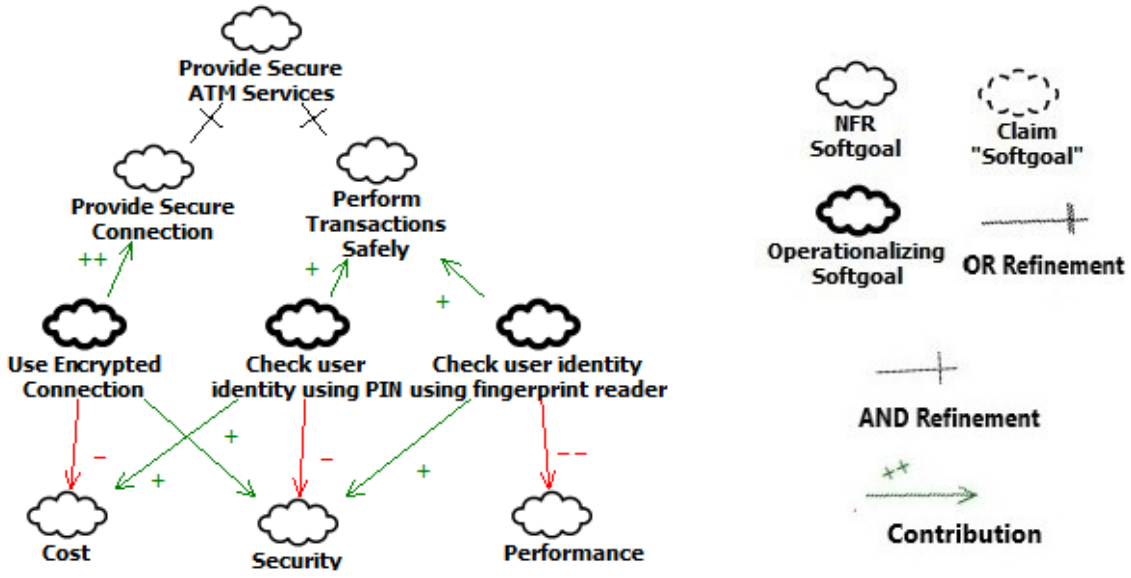
Para ilustrar cada linguagem, este texto utiliza como exemplo, fragmentos de modelos obtidos a partir de um experimento de modelagem realizado no contexto deste trabalho (vide Capítulo 4). Os modelos de exemplo descrevem um sistema de caixas eletrônicos, onde são executadas operações bancárias básicas como saque, depósito, transferência e extrato. A descrição completa do domínio e as versões completas dos modelos podem ser encontradas em <https://nemo.inf.ufes.br/projects/rose/>. É importante salientar que os modelos produzidos no experimento estão em inglês, porém o texto a seguir refere-se aos nomes dos elementos dos modelos em português, apresentando o nome original dos elementos entre parenteses. Na página do projeto, também é possível encontrar uma lista completa de todos os elementos das linguagens apresentadas a seguir, bem como suas definições e o mapeamento de cada conceito para conceitos da GORO. Este mapeamento é discutido na Seção 4.2.

2.1.1 NFR Framework

O NFR Framework (MYLOPOULOS; CHUNG; NIXON, 1992) (NFRF) representa Requisitos Não-funcionais (Softgoals) como um grafo de objetivos. Em NFRF, Softgoals também podem ser do tipo Operationalizing Softgoal, quando modelados como operações que atendem a um determinado objetivo e Claim Softgoal, quando modelados de forma a guardar o processo de raciocínio ou para justificar um determinado refinamento. Portanto, percebe-se que foco principal de NFRF é lidar com *trade-offs* e prioridades entre Requisitos Não-funcionais, já que este *framework* busca encontrar a melhor solução para um determinado domínio por meio de um algoritmo de propagação, no qual decisões tomadas em um nível mais refinado de detalhamento são propagadas até o nível superior.

Elementos de NFRF são refinados por meio de relações de decomposição, que podem assumir semânticas diferentes: *AND Decomposition*, significando que o Softgoal de nível mais alto é atendido quando todos os subgoals o são; e *OR Decomposition*, quando é necessário que apenas um Softgoal seja atendido para que o de nível mais alto também seja.

Na Figura 2 é apresentado um exemplo de modelo NFRF, com a legenda dos elementos mostrada na Figura 2b. No modelo, é mostrado o Softgoal *Prover Serviços de Caixa Eletrônico com Segurança* (*Provide Secure ATM Services*), que é *AND Decomposed* em outros dois: *Prover Conexão Segura* (*Provide Secure Connection*) e *Executar Transações com Segurança* (*Perform Transactions Safely*). Os Operationalizing Softgoals apresentados no modelo contribuem para vários Softgoals. As relações de contribuição (*Contribution Relations*) são identificadas por simbolos + e −, dependendo do tipo e intensidade, sendo que contribuições parciais são denotadas por + ou − e totais por ++ e −−. Por exemplo,



(a) NFR softgoals. (b) Elementos de NFR (sintaxe).

Figura 2 – Exemplo de modelo em NFR.

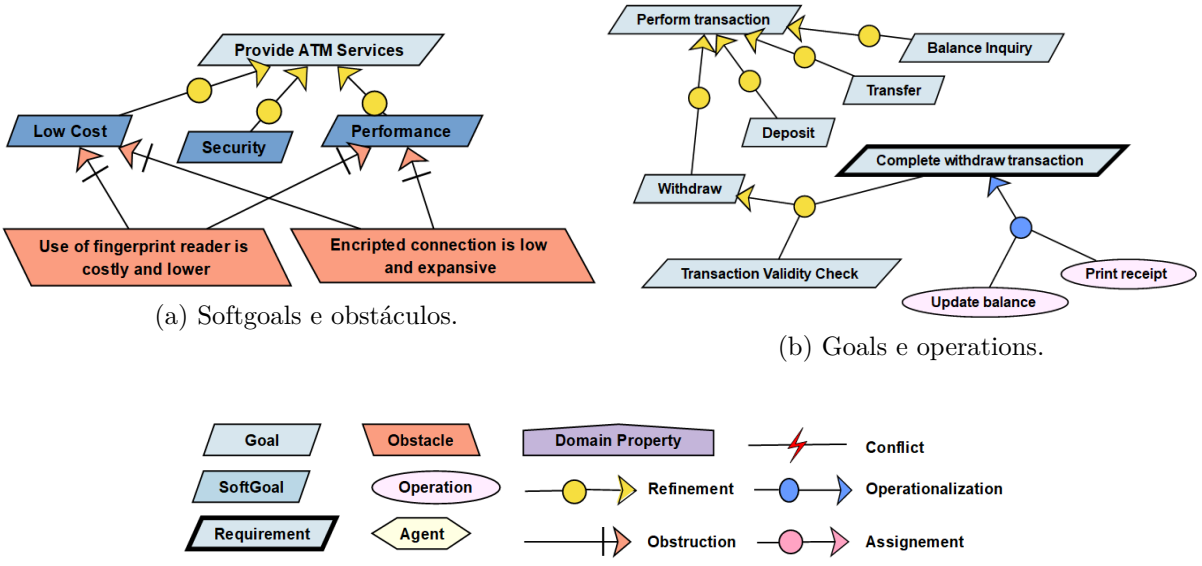
o Operationalizing Softgoal *Checar identidade do cliente usando leitor biométrico* (*Check user identity using fingerprint reader*) contribui completa e negativamente (−−) para o Softgoal *Desempenho* (*Performance*).

2.1.2 KAOS

KAOS (DARDENNE; LAMSWEERDE; FICKAS, 1993) (*Keep All Objects Satisfied* ou *Knowledge Acquisition in autOmated Specification*) enxerga objetivos de uma forma diferente do NFRF. Em KAOS, objetivos (Goals) são estados desejados das coisas, com critérios claros de satisfação e geralmente relacionados a Requisitos Funcionais. O *framework* trata tanto agentes humanos como agentes de software como Agents, atores presentes no contexto do sistema. Quando um objetivo é associado a um agente humano, ele é classificado como Expectation. Caso um objetivo seja associado a um agente de software, ele passa a ser classificado como um Requirement. Ambos, Expectation e Requirement, são subtipos de Goal e podem ser operacionalizados por Operations, que são tarefas realizadas para atender um objetivo. Além disso, KAOS traz os conceitos de Domain Property (uma pressuposição de domínio, verdadeira em um determinado conjunto de situações) e Obstacle (comportamentos indesejados do sistema em um determinado contexto). Apesar de os artigos que propõe a linguagem KAOS não definirem Softgoal explicitamente, este elemento acabou sendo utilizado pela comunidade, ao ponto de ser incluído na ferramenta de modelagem oficial da linguagem. Neste caso, um Softgoal é visto como um requisito de qualidade sem critério de satisfação preciso.

Os elementos de KAOS relacionam-se através de *Refinement links*, que obedecem à

mesma lógica AND/OR proposta por NFRF; *Assignment links*, que relacionam Agents e Requirements/Expectations; *Operationalization links*, que relacionam uma tarefa a um Requirement/Expectation; *Obstacle links* que relacionam um obstáculo a um Goal; e *Conflict links*, que relaciona dois Goals conflitantes (RESPECT-IT, 2007).



(a) Softgoals e obstáculos.

(b) Goals e operations.

(c) Elementos KAOS (sintaxe).

Figura 3 – Examplo de Modelo KAOS.

A Figura 3 mostra dois exemplos de modelos KAOS. Na Figura 3a, o Goal *Prover Serviços de Caixa Eletrônico* (*Provide ATM Services*) é refinado em três Softgoals. Este refinamento é um refinamento do tipo OU (OR), já que os círculos amarelos entre as linhas estão desconectados. Se todas as linhas se concentrassem no mesmo círculo, teríamos um refinamento do tipo E (AND). Os *Softgoals Baixo Custo* (*Low Cost*), *Segurança* (*Security*) e *Desempenho* (*Performance*) são limitados por obstáculos. Já a Figura 3b traz um outro trecho do mesmo modelo, onde é possível ver refinamentos do tipo E (AND) entre o Goal *Saque* (*Withdraw*) e os Subgoals *Checar Veracidade da Transação* (*Transaction Validity Check*) e *Completar Transação de Saque* (*Complete withdraw transaction*). Esta última operacionalizada por duas atividades: *Atualizar Saldo* (*Update Balance*) e *Imprimir Recibo* (*Print Receipt*).

2.1.3 *i**

A linguagem *i** (YU, 1996) foca na representação dos interesses de atores e *stakeholders* dentro do contexto organizacional do sistema. Portanto, o principal elemento do *i** é o Ator (Actor). Objetivos (Goals), Softgoals, Tarefas (Tasks) e Recursos (Resources) são representados dentro da fronteira de um Actor, que por sua vez pode ser modelado por elementos mais específicos: Agent, Position e Role.

Um Goal em *i** é definido como conteúdo intencional de um ator, algo que ele deseja

que se torne verdadeiro na realidade. Um Softgoal é um Goal sem critério claro de satisfação e, portanto, geralmente refere-se a requisitos de qualidade. Tasks são descrições de tarefas que devem ser executadas de maneira específica para atingir um objetivo. Resources são entidade físicas ou informacionais utilizadas durante a execução de uma tarefa. Em *i**, recursos são apenas consumidos por tarefas, não podendo ser produzidos por elas (YU, 1996).

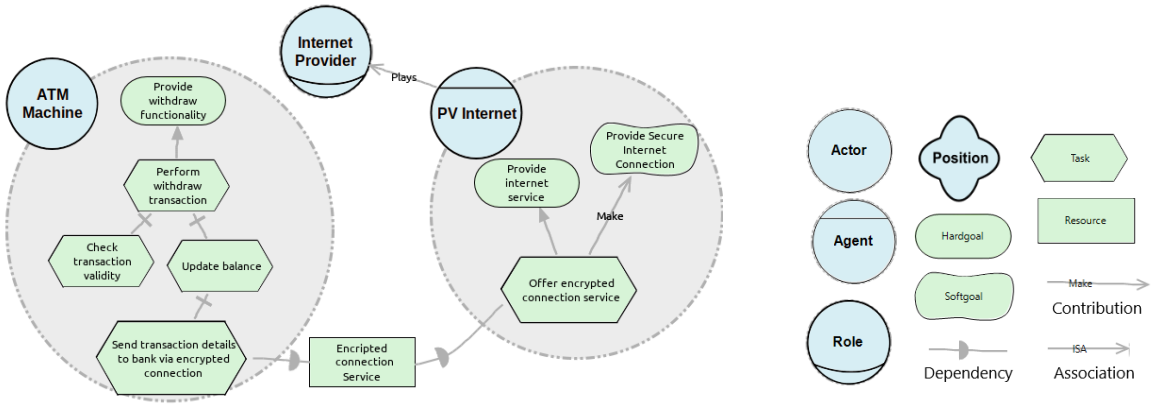
*i** relaciona os elementos apresentados através de links de decomposição (*Decomposition links*), links meio-fim (*Means-end links*), links de contribuição (*Contribution links*) e links de dependência (*Dependency links*). *Decomposition links* conectam objetivos de nível mais alto (mais genéricos) a objetivos de nível mais específico, obedecendo a lógica AND/OR tradicional. O *Means-end link* conecta uma Task (o meio) a um Goal (o fim), representando que aquela tarefa é uma forma de atender ao objetivo. *Contribution links* conectam Goals, Tasks e Softgoals à Softgoals, representando que a origem contribui total ou parcialmente, positiva ou negativamente para a realização do destino, assumindo as seguintes classificações: *Break* (contribuição negativa completa), *Hurt* (contribuição negativa parcial), *Help* (contribuição positiva parcial) e *Make* (contribuição positiva completa).

Dependências são o diferencial de *i** e indicam que um ator depende de outro para atingir um objetivo, realizar uma tarefa ou prover um recurso. Um relação de dependência possui três elementos: o Depender, o Dependee e o Dependum. O Depender é o Actor que depende de outro Actor (o Dependee) para fornecer o Dependum. Um Dependum pode ser um Goal, um Softgoal, uma Task ou um Resource e é o elemento que justifica a existência daquela dependência.

A Figura 4 mostra um exemplo de um modelo de *i**. No modelo, é apresentado o objetivo principal (um Hardgoal) *Prover funcionalidade de saque* (*Provide withdraw functionality*), que é atendido pela Task *Executar transação de saque* (*Perform withdraw transaction*), decomposta em duas outras tarefas, *Checar validade da transação* (*Check transaction validity*) e *Atualizar saldo* (*Update balance*), por sua vez decomposta na tarefa *Enviar dados da transação ao banco via conexão criptografada* (*Send transaction detils to bank via encrypted connection*). Esta última é atendida por outro ator via relação de dependência. O ator *PV Internet*, que desempenha o papel de *Provedor de Internet* (*Internet Provider*), atende a dependência sobre ele através da tarefa *Oferecer serviço de conexão criptografada* (*Offer encrypted connection service*).

2.1.4 GBRAM

GBRAM (Goal-Based Requirements Acquisition Method) (ANTON, 1996) é um método proposto em 1996, sendo uma das poucas linguagens a não utilizar modelagem gráfica. “Modelos” GBRAM são descritos através de tabelas nas quais são listados os elementos e suas dependências. O método define duas atividades para levantamento de



(a) Fragmento de modelo *i**. (b) Elementos *i** (sintaxe).

Figura 4 – Exemplo de modelo *i**.

requisitos: Análise de Objetivos, onde diferentes fontes de informação são analisadas, e Refinamento de Objetivos, onde os objetivos levantados na primeira atividade são refinados e possíveis obstáculos são identificados. Os conceitos usados em GBRAM são:

- Agent: entidades que realizam tarefas para atingir objetivos;
- Achievement Goals: objetivos que são atingidos em um ponto específico do tempo;
- Constraints: requisitos que impõem condições à satisfação de um objetivo;
- Maintenance Goals: objetivos considerados satisfeitos enquanto alguma proposição sobre a realidade for verdadeira;
- Obstacles: impedem (parcial ou totalmente) a satisfação de um objetivo;
- Scenario: descrições de um domínio utilizadas para identificar possíveis obstáculos;
- Stakeholder: entidades que possuem objetivos e podem realizar atividades para atingi-los, ou atribui-los a agentes.

Conforme mencionado, GBRAM não utiliza nenhum tipo de representação gráfica, mas utiliza tabelas para atribuir Achievement e Maintenance Goals a Agents e Stakeholders e identificar Obstacles e Scenarios. A Tabela 1 mostra um pequeno exemplo da documentação de um Maintenance Goal. Para exemplificar, o Maintenance Goal *Prover serviços de caixa eletrônico* é atribuído ao agente *Sistema de caixa eletrônico* e foi elicitado do Stakeholder *Banco Dinero*. A satisfação do objetivo mencionado pode ser obstruída pelo obstáculo *Caixa eletrônico não funcionando devido à falta de eletricidade*.

Tabela 1 – Exemplo de documentação de objetivo em GBRAM.

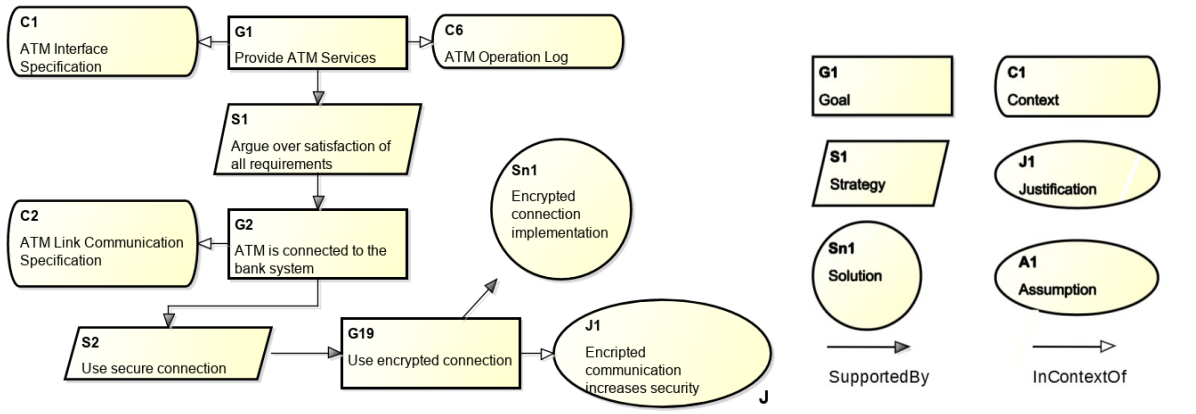
Maintenance Goals	**Agent**	**Stakeholder**	**Obstacles**
G1: Prover serviços de caixa eletrônico	Sistema de caixa eletrônico	Banco Dinero	Caixa eletrônico não funcionando devido a falta de eletricidade
G2: Prover conexão criptografada	Provedor de Internet	Banco Dinero	Conexão Indisponível

2.1.5 GSN

GSN (KELLY; WEAVER, 2004) (*Goal Structure Notation*) é uma linguagem surgida em 2004. Devido ao foco em sistemas de segurança, tornou-se famosa na indústria, onde foi utilizada por alguns órgãos internacionais como o Controle de Tráfego Aéreo Inglês e a NASA (KELLY; WEAVER, 2004). Dado seu foco particular, a linguagem apresenta alguns elementos específicos, que não estão presentes em outras linguagens, mas também apresenta elementos comuns à GORE:

- Assumption: pressuposição sobre a realidade.
- Context: delimita o escopo no qual um objetivo pode ser satisfeito.
- Goal: objetivos que devem ser atingidos no sistema ou contexto.
- Justification: usado para descrever a necessidade (justificativa) de um objetivo.
- Solution: apresenta evidência para a satisfação de um objetivo, ou seja, prova que um objetivo pode ser satisfeito.
- Strategy: descreve a lógica adotada ao se realizar processos de inferência (refinamento) entre objetivos.

A Figura 5a mostra um exemplo de modelo em GSN. O objetivo *Prover Serviços de caixa eletrônico* (*Provide ATM Services*) é atendido assim que todos os seus subgoals são satisfeitos. Em geral, refinamentos mais complexos são conectados por Strategies, como no caso do Goal *O caixa eletrônico está conectado ao sistema do banco* (*ATM is connected to the bank system*), ligado ao Strategy *Usar conexão segura* (*Use secure connection*), então suportado pelo Goal *Usar conexão criptografada* (*Use encrypted connection*), por sua vez documentada pela Justification *Criptografia aumenta segurança* (*Encrypted communication increases security*), finalmente suportada pela solução *Implementação de conexão criptografada* (*Encrypted connection implementation*).



(a) Fragmento de modelo GSN. (b) Elementos de GSN (sintaxe).

Figura 5 – Exemplo de modelo GSN.

2.1.6 Tropos, GRL e demais variantes de *i**

Devido à popularidade de *i**, diversas variantes desta linguagem surgiram desde que ela foi proposta, sendo Tropos (BRESCIANI et al., 2004) e GRL (AMYOT et al., 2010) as mais populares (GONÇALVES et al., 2018a).

Tropos trouxe um *framework* para análise de requisitos que vai desde a elicitação até a implementação. O *framework* trouxe também o conceito de Capability aos modelos de objetivos baseados em *i**. Capability é descrito como a habilidade de um ator em executar uma atividade para atender um objetivo. A linguagem também utiliza o nome Hardgoal para falar de objetivos e Plan para falar de tarefas.

GRL traz duas diferenciações entre a relação de decomposição do tipo OR (*OR-Decomposition*: inclusiva, que aceita que **pelo menos** um objetivo seja atingido para atender o objetivo maior, e exclusiva, que só aceita que **um único** objetivo seja satisfeito para que o super-objetivo também seja. A linguagem também traz o conceito de *Correlation*, uma contribuição sem tipo e intensidade definidos, que representa efeitos colaterais em vez de impactos desejados. Um exemplo de *Correlation*, representada por uma linha tracejada sem símbolos próximos, pode ser visto na Figura 6. *Usar leitor de impressão digital* (*Use fingerprint reader*) contribui negativamente para *Custo* (*Cost*), enquanto *Usar senha* (*Use PIN*) contribui positivamente, representando assim alternativas de solução. Em outras palavras, a contribuição de *Usar senha* é positiva pois é contrastada com a sua alternativa (*Usar leitor de impressão digital*). Enquanto isso, *Ler cartão* (*Read card*) afeta *Custo* (*Cost*), mas não faz parte de uma alternativa específica de solução e, por isso, não é do tipo positivo ou negativo.

Gonçalves et al. (2018a) apresenta uma revisão sistemática de literatura onde são catalogadas 96 propostas de extensões diretas de *i*/iStar* ou de extensões de *i*/iStar* através de GRL, Tropos, Secure Tropos (MOURATIDIS; GIORGINI, 2007) (uma extensão de Tropos voltada para Segurança) ou Nòmos (SIENA et al., 2012) (uma extensão de Secure

Tropos focada em normas e leis). O estudo analisa as extensões em relação à proposta de novos conceitos e construtos e também identifica possíveis problemas e conflitos entre as propostas levantadas.

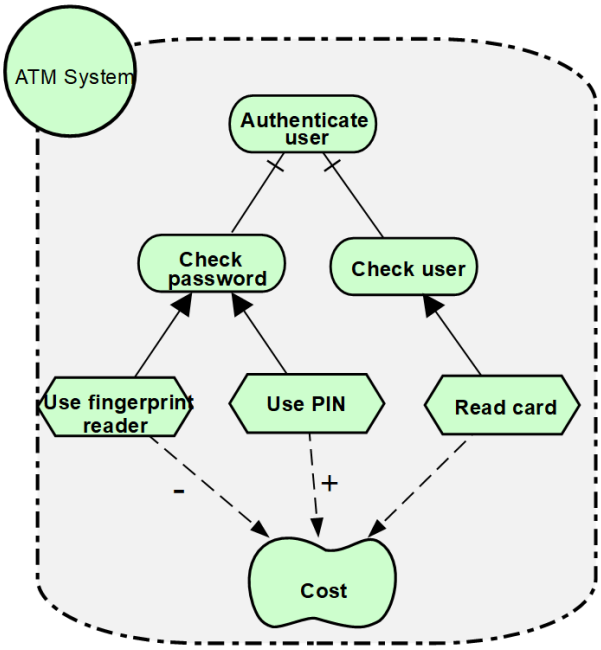


Figura 6 – Exemplo de modelo GRL.

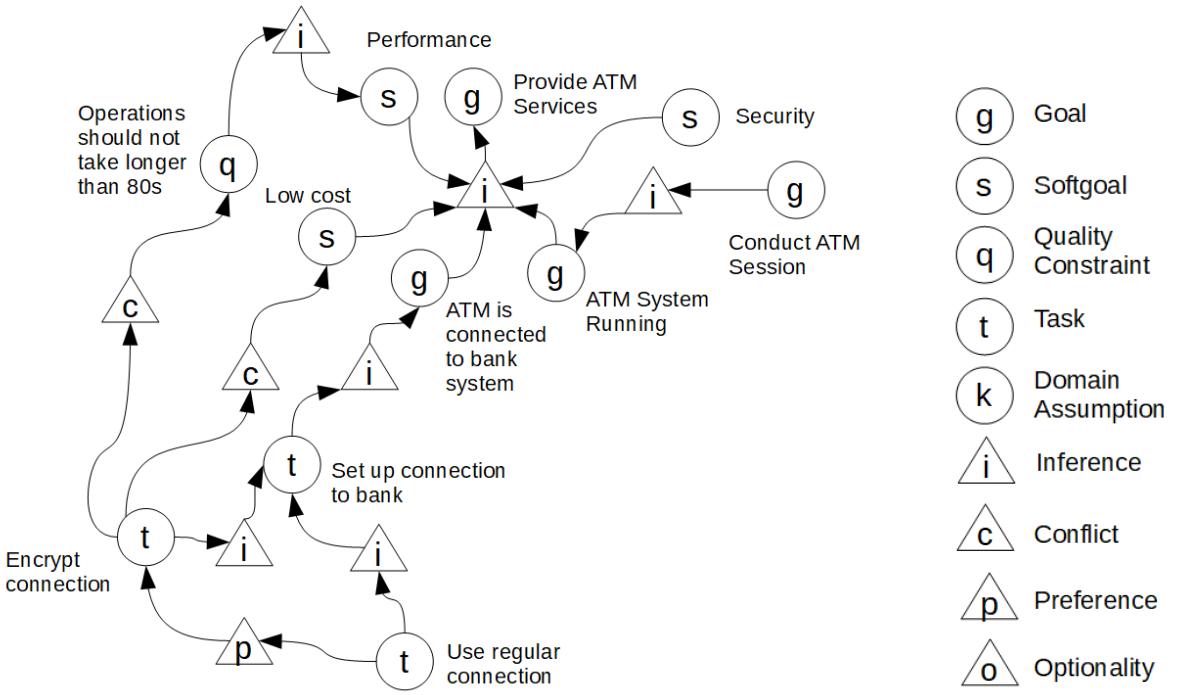
2.1.7 Techne

Techne (BORGIDA et al., 2010) é uma linguagem fundamentada na ontologia DOLCE (MASOLO et al., 2003). A linguagem diferencia objetivos em dois tipos: Goals, que são objetivos com critérios claros de satisfação, e Softgoals, que não possuem critérios precisos para serem considerados satisfeitos. Entretanto, o grau em que um Softgoal pode ser considerado satisfeito pode ser quantificado por uma Quality Constraint. Uma pressuposição de domínio (Domain Assumption) captura uma propriedade do estado de coisas que as partes interessadas acreditam ser verdade (no ambiente de interesse).

Em Techne, elementos são relacionados através de outros elementos (nós), que podem ser do tipo *Inference* (o destino é considerado satisfeito assim que a origem o é), *Conflict* (conecta dois elementos que não podem estar no mesmo conjunto solução), *Preference* (a satisfação da origem tem maior prioridade perante a satisfação do destino) e *Optionality* (marca um nó cuja satisfação é opcional).

A Figura 7 mostra um exemplo de modelo em Techne. No modelo, nós "elementos" são representados por círculos, enquanto nós "relações" são representados por triângulos. O objetivo principal *Prover serviços de caixa eletrônico* (*Provide ATM Services*) é inferido (através do nó de inferência (*Inference*) denotado por um triangulo com a letra "i") por outros Goals e Softgoals. O Softgoal *Desempenho* (*Performance*) é quantificado pela Quality

Constraint *Operações não devem levar mais que 80 segundos* (*Operations should not take longer than 80s*). Na Figura 7a, a Task *Criptografar conexão* (*Encrypt connection*) conflita (*Conflicts*) com o Softgoal *Baixo Custo* (*Low cost*). O triângulo com a letra "p" representa relações de preferência (*Preference Relationship*). No exemplo, a Tarefa *Usar conexão comum* (*Use regular connection*) possui preferência sobre *Criptografar conexão* (*Encrypt connection*).



(a) Fragmento de modelo Techne.

(b) Elementos de Techne (sintaxe).

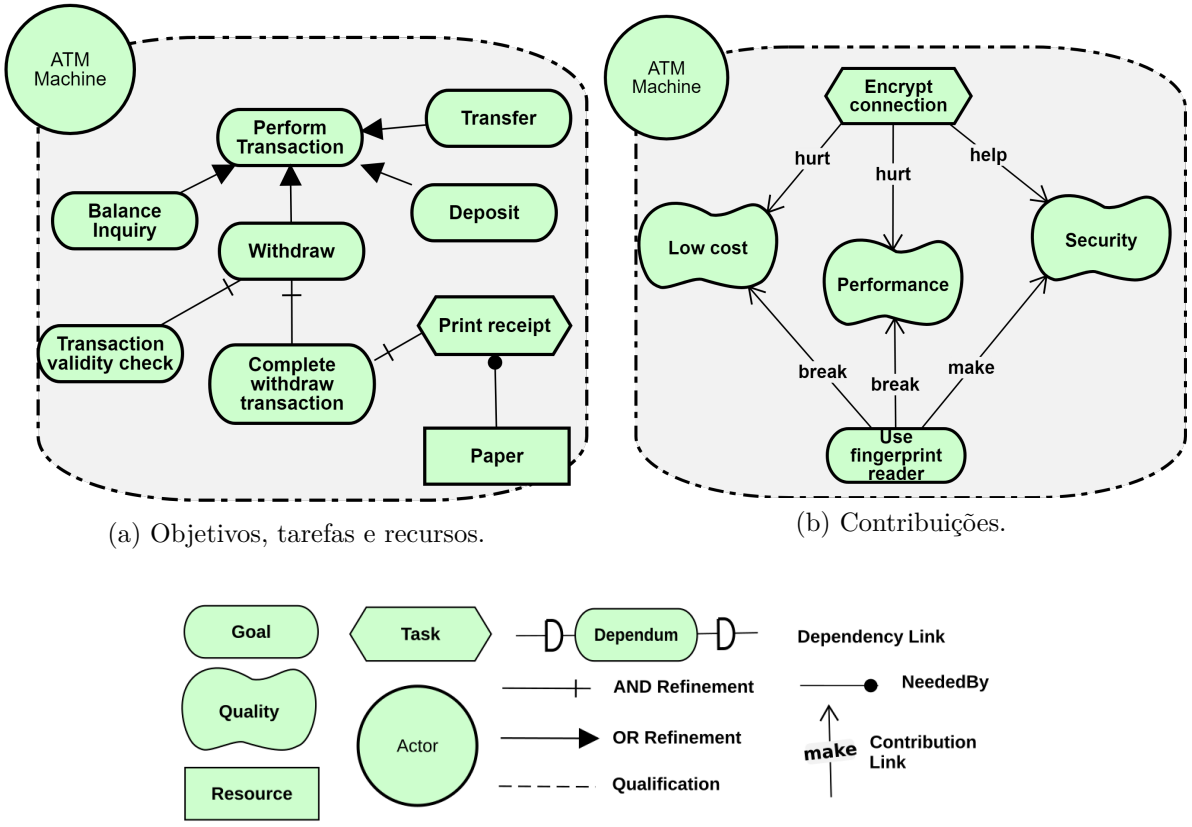
Figura 7 – Exemplo de modelo Techne.

2.1.8 iStar

Uma nova versão do *i** foi proposta em 2016, com o nome *iStar* (DALPIAZ; FRANCH; HORKOFF, 2016). O principal foco desta versão foi reduzir a dificuldade de aprendizado da linguagem e padronizar o uso da mesma. Assim, alguns elementos foram renomeados, outros removidos ou adicionados, ou agrupados em elementos mais gerais. O construto Softgoal passou a se chamar Quality. O construto Position foi removido. As relações de *Means-end* e *Task Decomposition* foram agrupadas na relação *Refinement Link*, que também considera *Goal Decomposition*. Foram criadas as relações *neededBy*, que conecta um recurso a uma tarefa que o consome/utiliza, e *Qualification*, que liga uma Quality a um Goal.

A Figura 8 mostra dois trechos de um modelo iStar para o Sistema de Caixa Eletrônico. Na Figura 8a, o Goal *Executar Transação* (*Perform transaction*) é refinado com tipo OR (*OR-Refined*) em quatro subobjetivos. O Goal *Saque* (*Withdraw*), por sua

vez, é *AND-Refined* em outros dois sub-objetivos. O Goal *Completar Transação de Saque* (*Complete withdraw transaction*) é *AND-Refined* pela Task *Imprimir recibo* (*Print receipt*), que usa o Resource *Papel* (*Paper*). A Figura 8b descreve a contribuição (*Contribution*) da Tarefa *Criptografar Conexão* (*Encrypt connection*) e do Goal *Usar leitor biométrico* (*Use fingerprint reader*) para as Qualities *Baixo Custo* (*Low cost*), *Desempenho* (*Performance*) e *Segurança* (*Security*).



(a) Objetivos, tarefas e recursos.

(b) Contribuições.

(c) Elementos de iStar (sintaxe).

Figura 8 – Exemplo de modelo iStar.

2.2 UFO

Esta seção oferece uma breve introdução sobre ontologias de fundamentação, mais precisamente a UFO (*Unified Foundational Ontology*) (GUIZZARDI, 2005). Em geral, uma ontologia de fundamentação define um sistema de categorias formais, independentes de domínio e bem fundamentadas filosoficamente. Ontologias de fundamentação são utilizadas para articular modelos específicos de domínio, já que descrevem conceitos como espaço, tempo, objeto, eventos e ações (GUARINO, 1998).

A UFO surgiu a partir da unificação da GFO (*Generalized Formalized Ontology*) (HELLER; HERRE, 2004) e da *OntoClean* (GUARINO; WELTY, 2002). A partir desta unificação, UFO passou a ser desenvolvida com base em várias teorias das áreas de

Ontologias Formais, Lógica Filosófica, Filosofia de Linguagem, Linguística e Psicologia Cognitiva (GUIZZARDI et al., 2015).

Atualmente a UFO está estruturada em três partes:

- **UFO-A**: parte central da UFO, lida com entidades estruturais. Trata de objetos e suas partes e propriedades;
- **UFO-B**: focada em eventos e partes temporais, baseia-se em entidades de UFO-A;
- **UFO-C**: composta por entidades intencionais e sociais, como agentes, crenças e ações. Utiliza as outras duas partes de UFO.

2.2.1 UFO-A

O conceito base de UFO é o **Entity**, que representa algo que pode ser concebido ou percebido. Uma Entity pode ser classificada como Individual ou Universal. Um **Individual** refere-se a entidades que existem na realidade e possuem identidade única, enquanto um **Universal** descreve padrões de características comuns a vários Indivíduals. *Pabllo Vittar* é um exemplo de Individual, enquanto *pessoa* ou *cantora* são exemplos de Universals. Individuals são instâncias (*instance of*) de Universals, conforme pode ser visto na Figura 9.

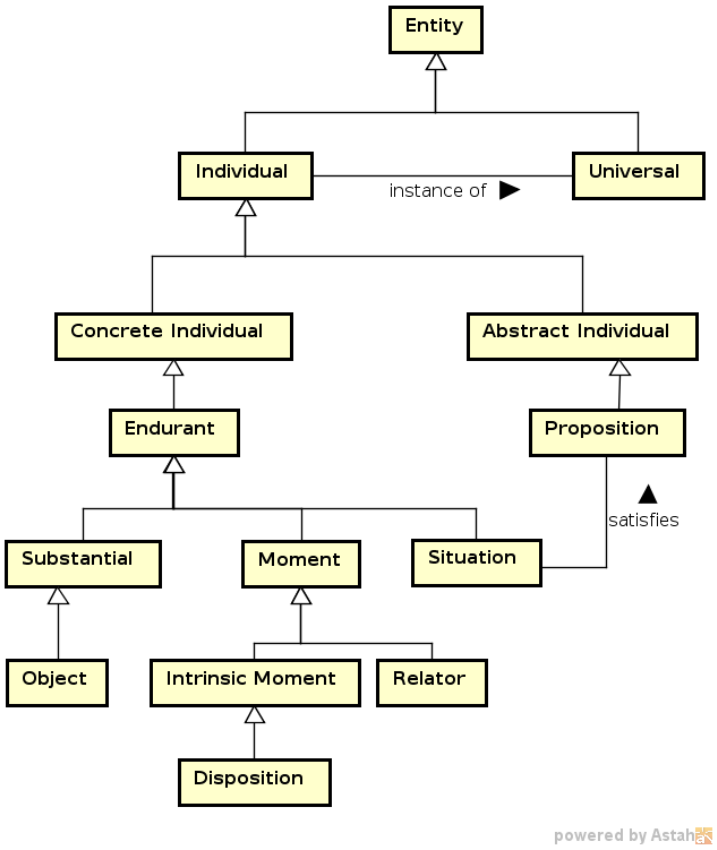


Figura 9 – Fragmento de UFO-A (NEGRI, 2017).

Um Individual, por sua vez, pode ser especializado em Concrete Individual e Abstract Individual. **Propositions** são subtipos de Abstract Individuals que representam o conteúdo de um estado mental de um agente e fazem referência a uma situação. Assim, uma situação (**Situation**) satisfaz uma proposição. **Endurants** são tipos de Concrete Individuals que mantêm sua identidade ao longo do tempo e podem ser classificados em Substantial, Moment e Situation.

Substantials são entidades existencialmente independentes, não inerentes a outro Individual. Um **Object** é um tipo de Substantial com identidade bem definida e que não depende de todas as suas partes.

Moments, por outro lado, são existencialmente dependentes em um Individual, representando uma propriedade deste (cor, altura, peso, etc). Um Moment pode ser intrínseco (**Intrinsic Moment**) quando depender de um único Individual para existir. A vontade de *Pabllo Vittar* em gravar uma nova música é um exemplo de Intrinsic Moment. **Disposition** é um tipo de Intrinsic Moment que se manifesta somente em determinadas situações, como a habilidade de um *ímã* em *atrair metal*, só manifestada na presença de um metal. Diferentemente de um Intrinsic Moment, um **Relator** é um tipo de Moment que depende de mais de um Individual, conectando-os. Um *abraço* ou *contrato de casamento* são exemplos de Relators.

Situations são tipos complexos, ou seja, podem ser compostos por outros tipos de Endurants (até mesmo outras Situations), representando porções da realidade que pode ser entendida como um todo. Assim, uma Situation pode satisfazer (*satisfies*) uma Proposition.

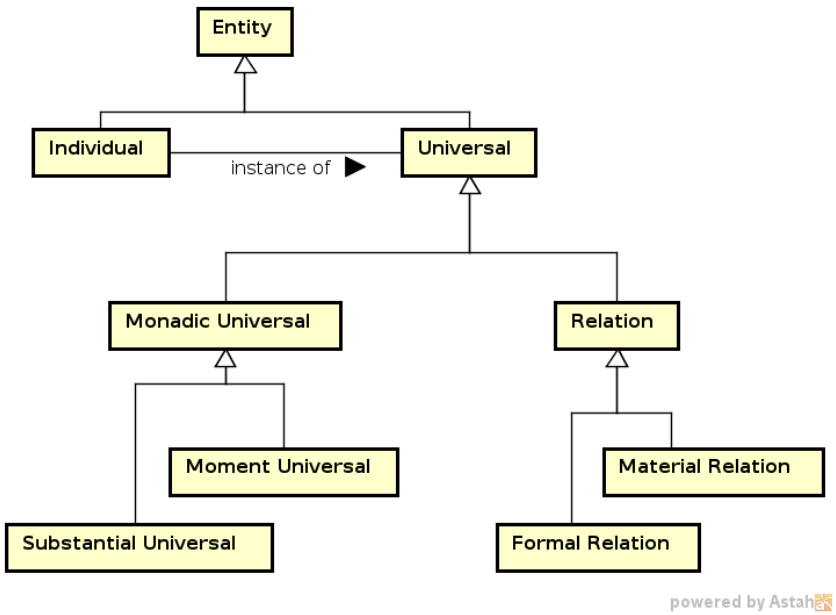


Figura 10 – Fragmento de UFO-A (NEGRI, 2017).

Na Figura 10, é explorado o conceito Universal, que pode ser classificado em Relation Universal e Monadic Universal.

Um **Relation Universal** é uma entidade que une outras entidades, podendo ser especializado em **Formal Relation**, quando a relação for estabelecida com base em propriedades intrínsecas aos indivíduos, ou em **Material Relation**, quando possuem estrutura material própria. *Mais pesado que* é um exemplo de Formal Relation, enquanto *trabalhar em* é um exemplo de Material Relation.

Um **Monadic Universal** é um tipo de Univesal que não une outras entidades, podendo ser especializado em **Moment Universal** (um Universal instanciado por um Moment Individual) ou **Substantial Universal** (um Universal instanciado por um Substantial Individual).

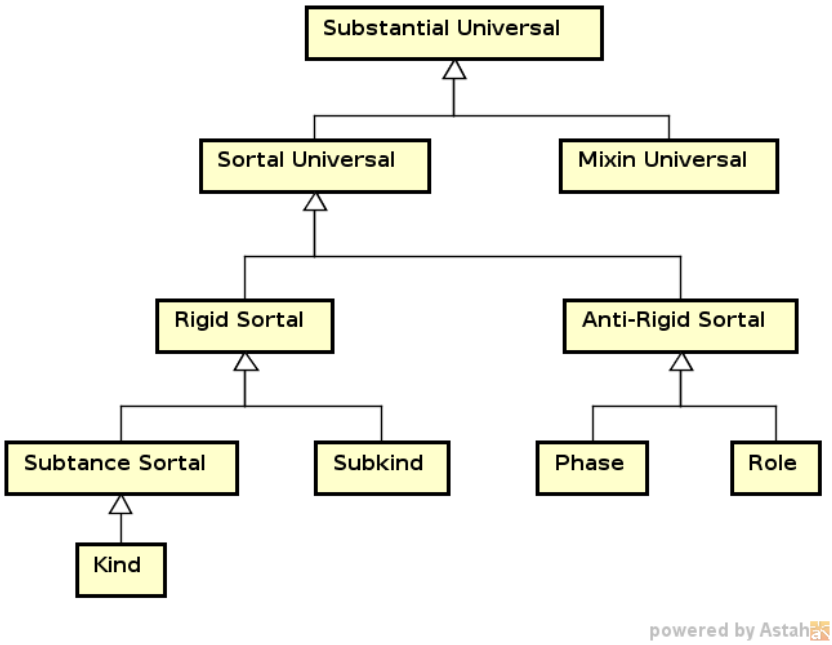


Figura 11 – Fragmento de UFO-A (NEGRI, 2017).

2.2.2 UFO-B

A Figura 13 apresenta UFO-B, a segunda parte de UFO, com foco em eventos. Um **Event (Perdurant)** é uma especialização de Individual (UFO-A) que acontece no tempo (Time Interval), acumulando partes temporais. Uma *palestra* ou um *show musical* são exemplos de Events. Um Event é uma instância (*instance of*) de um **Event Universal**. Eventos podem ser do tipo **Complex Events**, quando compostos por dois ou mais eventos, ou **Atomic Events**, caso contrário.

Eventos são existencialmente dependentes, visto que precisam de participantes (**Participation**) para existir. Por exemplo, em um *show musical* tem-se a participação da *cantora*, do *público*, da *banda*, das *caixas de som* e dos *instrumentos musicais*.

Eventos impactam o mundo real, transformando a realidade. Assim, eventos levam uma Situation X (*pre-state*) a uma Situation Y (*pos-state*).

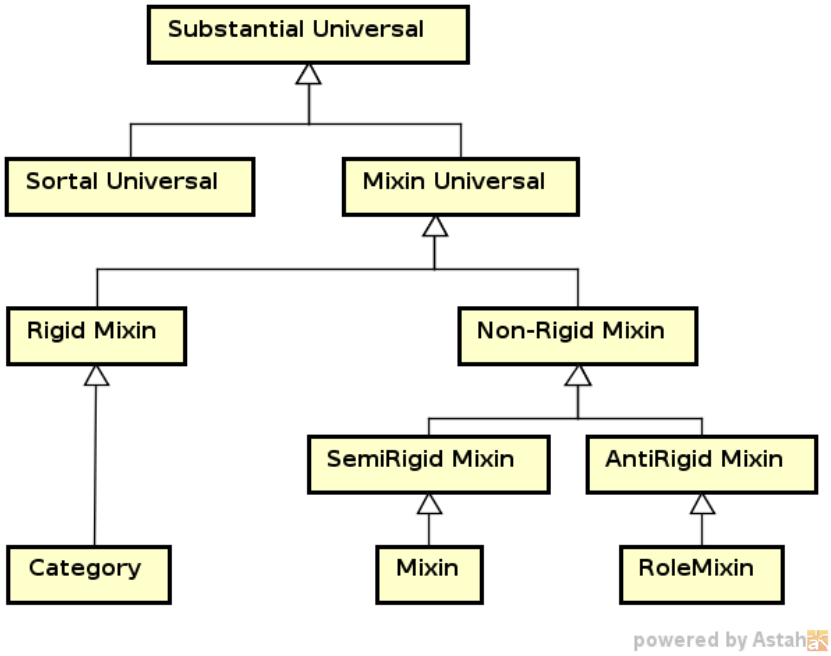


Figura 12 – Fragmento de UFO-A (NEGRI, 2017).

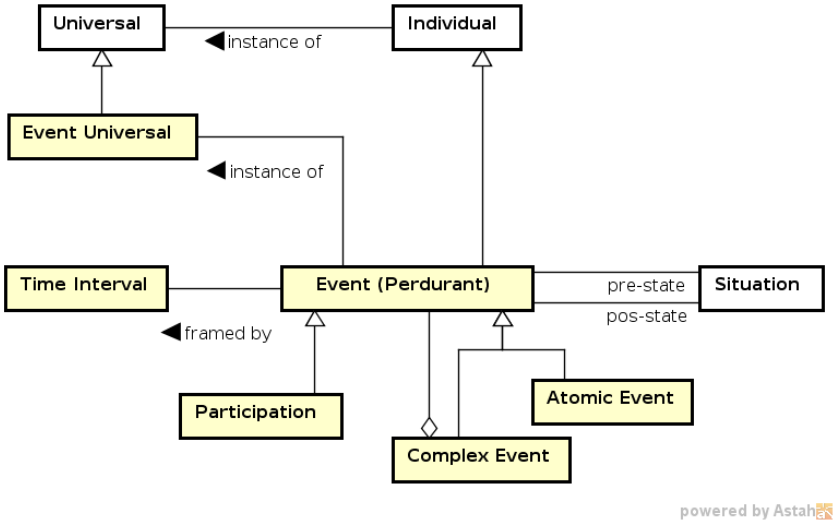


Figura 13 – Fragmento de UFO-B (NEGRI, 2017).

2.2.3 UFO-C

UFO-C é a terceira parte de UFO, com foco em entidades sociais. A Figura 14 mostra que o conceito Substantial de UFO-A é especializado em dois tipos em UFO-C: agentes (Agents) e objetos (Objects).

Um **Agent** é capaz de desempenhar ações e perceber eventos, podendo ser classificado em **Physical Agent**, como uma *pessoa*, ou **Social Agent**, como uma *organização*, uma *banda*, etc.

Um **Object** é um Substantial incapaz de desempenhar ações ou perceber eventos.

Da mesma forma, podem ser classificados como **Physical Object** (livro, mesa, carro, etc.) e **Social Object** (dinheiro, a linguagem brasileira de sinais, etc.).

Um Agent pode ter um **Intentional Moment**, tipo de Intrinsic Moment inerente a um agente. Um Intentional Moment pode ser do tipo Mental Moment ou Social Moment.

Um **Mental Moment** pode ser especializado em Intention, Desire ou Belief, como pode ser visto na Figura 15. Um Belief é a crença de que uma determinada situação é verdadeira na realidade (a crença de que a Terra é redonda e orbita em torno do Sol, por exemplo). Já a diferença entre Desire e Intention é dada pelo comprometimento (Commitment). Enquanto um **Desire** descreve apenas uma vontade em agir no estado das coisas, uma **Intention** descreve um compromisso interno em agir no estado das coisas. Por exemplo, posso ter o desejo (Desire) em viajar para a Grécia um dia, e posso ter o compromisso interno de juntar dinheiro todo mês para viajar para a Grécia no próximo ano (Intention).

Intentions fazem agentes tomar uma ação (**Action**) para mudar o estado das coisas. Uma Action pode ser complexa (**Complex Action**) quando composta por partes menores, ou atômica (**Atomic Action**) caso contrário. Uma Action pode ser regulamentada ou normalizada por **Normative Descriptions**, descrições normativas descritas por objetos sociais e reconhecidas por agentes. A *Constituição Brasileira* é um exemplo de Normative Description.

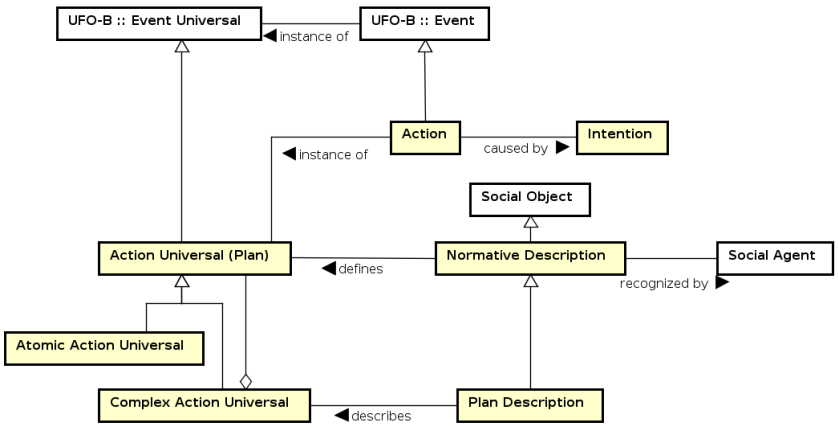


Figura 14 – Fragmento de UFO-C (NEGRI, 2017).

2.3 Ontologias Reutilizadas

Reúso é uma das importantes fases presentes em diversos métodos de Engenharia de Ontologias (REGINATO et al., 2019). Neste trabalho, foram reutilizadas cinco ontologias: a *Assumptions Ontology* (ASMP) (WANG et al., 2016), a *Non-functional Requirements Ontology* (NFRO) (GUIZZARDI et al., 2014), a *Common Ontology for Value and Risk* (COVR) (SALES et al., 2018), a *Software Process Ontology* (SPO) (RUY et al., 2016) e a

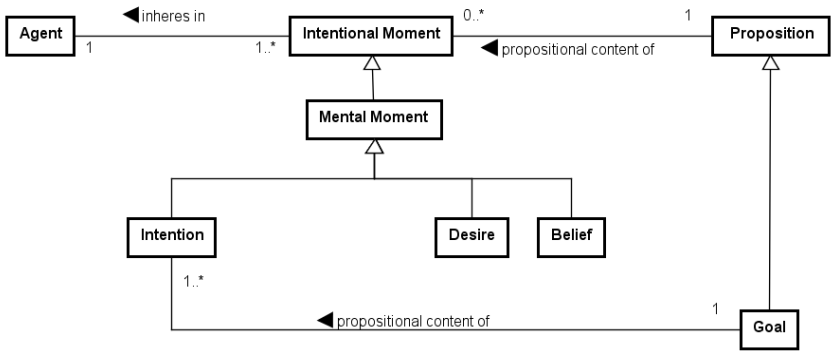


Figura 15 – Fragmento de UFO-C (NEGRI, 2017).

Reference Software Requirements Ontology (RSRO) (DUARTE et al., 2018). Todas elas são fundamentadas em UFO e quatro delas (exceto a COVR) fazem parte da *Software Engineering Ontology Network* (SEON) (RUY et al., 2016), uma rede de ontologias para o domínio de Engenharia de Software. As ontologias reutilizadas neste trabalho são discutidas nas próximas subseções.

2.3.1 Assumptions Ontology

A Assumptions Ontology (ASMP) é uma ontologia focada na definição de pressuposições de domínio, e classifica o conceito de **Assumption** ortogonalmente em dois grupos: (i) Assumption Used e Assumption Needed e (ii) Machine Assumption, World Assumption, World Dependence Assumption e Machine Dependence Assumption.

Assumptions-Used são classificadas pelo fato de serem usadas para construir um argumento, ou seja, pressuposições usadas para falar de propriedades do domínio da solução. **Assumption-Needed** são classificadas pelo fato de darem suporte a um alguma conclusão anterior, ou seja, explicam situações nas quais a solução dada se comportaria conforme o esperado. As *leis da física* são exemplos de Assumption-Used, enquanto o fato de que o *condensador do ar condicionado deve ser instalado do lado de fora de uma sala* é um Assumption-Needed.

Assumptions são classificadas em outro grupo, ortogonal ao primeiro, onde são definidas as fronteiras entre fenômenos da maquina e do mundo, ou seja, como a máquina e o mundo interagem:

- **World Assumption**: descreve fenômenos do mundo/ambiente externos à máquina, invisíveis a ela;
- **Machine Assumption**: descreve fenômenos internos à máquina, visíveis somente a ela mesma;
- **Machine Dependence Assumption**: fenômenos do mundo externo dependentes

da máquina;

- **World Dependence Assumption**: fenômeno da máquina dependente do mundo externo.

Para ilustrar as descrições acima, um agendador de reuniões pode ser usado como exemplo. Um *World Assumption* para esse sistema pode ser a afirmação de que existem salas suficientes para todos os pedidos. Um *Machine Assumption* para este sistema pode ser a premissa de que, ao se registrar um novo pedido de reunião, existe espaço disponível no banco de dados para salvar tal informação. Um exemplo de *Machine Dependence Assumption* seria assumir que uma reunião está agendada no mundo real pois foi adicionada uma entrada no banco de dados do sistema agendador de reuniões. Por fim, uma ilustração de *World Dependence Assumption* seria assumir que uma sala está de fato livre no mundo real se ela aparecer como disponível no sistema.

A ASMP foi totalmente incorporada em GORO com a finalidade de prover suporte a análises futuras sobre o conceito de pressuposição de domínio, presente em algumas linguagens GORE.

2.3.2 Non-functional Requirements Ontology

A *Non-functional Requirements Ontology* (NFRO) é derivada a partir da interpretação ontológica dada a requisitos não-funcionais, à luz de UFO. Um dos focos dessa ontologia é traçar fronteiras bem definidas entre requisitos funcionais (**Functional Requirement – FR**) e requisitos não-funcionais (**Non-funcitional Requirement – NFR**). Assim, a NFRO propõe que ambos **FR** e **NFR** sejam especializações de objetivo (nesse caso do conceito **goal** de UFO-C). Os conceitos de **Functional Requirement**, **Non-functional Requirement**, **Hardgoal** e **Softgoal** foram reutilizados em GORO.

Conforme a Figura 16, um **FR** é aquele que refere-se a funções (*refers to* **UFO-C::Function**), que em UFO são definidas como uma capacidade inerente a uma entidade. Essa capacidade, em determinadas situações, pode se manifestar através de um evento. Por outro lado, um **NFR** refere-se a qualidades (*refers to* **UFO::Quality Universal**). Nesse caso, qualidades são definidas como propriedades de um indivíduo, e que se manifestam sempre que o indivíduo existe. Uma qualidade pode ter valores em uma região particular (**Quality Region**).

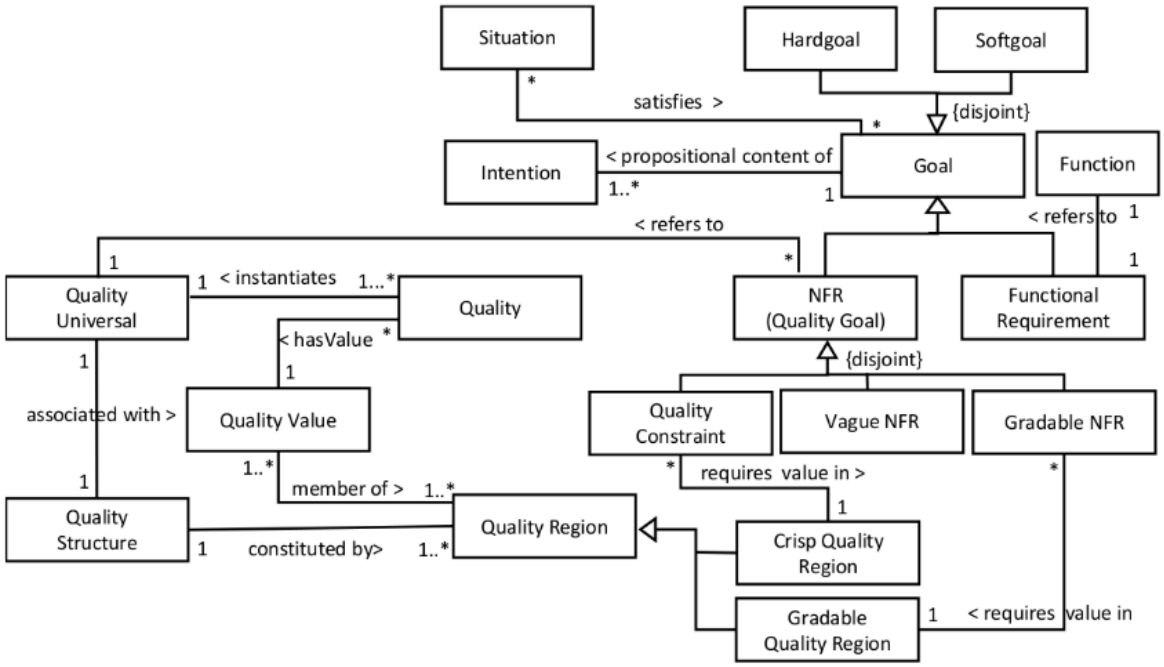


Figura 16 – Ontologia de Requisitos Não-funcionais (*Non-functional Requirements Ontology*) (GUIZZARDI et al., 2014).

A NFRO descreve os conceitos de **hardgoal** e **softgoal** de acordo com o grau de especificidade de um requisito. Nesse caso, utiliza-se a noção de valores de qualidade para diferenciar um do outro. Um **hardgoal** pode ser objetivamente satisfeito pois é possível limitar a região em que se darão seus valores. Por outro lado, um **softgoal** não possui critérios precisos de satisfabilidade, e portanto não se sabem os limites da região de seus valores de qualidade. Esses conceitos são ortogonais aos conceitos de **Functional** e **Non-functional Requirement**.

2.3.3 The Common Ontology of Value and Risk

A *Common Ontology for Value and Risk* (COVR) é uma ontologia criada para formalizar conceitos de risco e valor. A ontologia descreve estes conceitos com base em três perspectivas diferentes:

- **Perspectiva Empírica:** descreve valor e risco com base em eventos e suas causas;
- **Perspectiva Relacional:** identifica a natureza subjetiva de risco e valor;
- **Perspectiva Quantitativa:** projeta valor e risco em escalas mensuráveis.

Este trabalho reusa apenas a perspectiva empírica da COVR, mostrada na Figura 17. Nesta ontologia, evento é visto também como algo possível de acontecer no futuro (UFO originalmente descreve eventos somente no passado). Assim, risco é descrito como um

evento esperado. Em COVR, um **Risk Event** é especializado em **Threat Event**, um evento que impacta intenções negativamente, ou **Loss Event**, um evento com potencial em causar perdas.

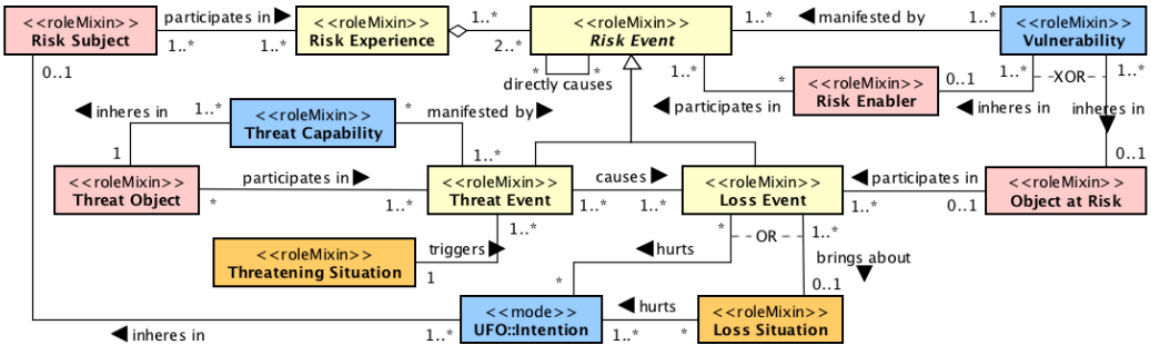


Figura 17 – Fragmento da COVR (SALES et al., 2018).

Uma **Threat Capability** descreve a capacidade em representar risco, é inerente a um **Threat Object**, e manifesta-se em um Threat Event, que também conta com a participação do **Threat Object**. Uma **Threatening Situation** é uma situação que dispara um **Threat Event**. GORO reutiliza o conceito de **Threatening Situation** para definir conceitos relacionados a obstáculos.

2.3.4 Software Process Ontology

A *Software Process Ontology* (SPO) foca no domínio do Processo de Software, considerando processos, atividades, recursos, pessoas, artefatos e procedimentos envolvidos em alguma parte do processo de criação de Software. A SPO é modularizada em três pacotes e suas relações.

O primeiro módulo de SPO, denominado *Standard Software Process Definition*, refere-se a processos estabelecidos pela organização, definindo requisitos básicos para a execução de atividades de projetos, como por exemplo o Processo de Desenvolvimento. Esses processos e seus requisitos são definidos pelo documento de definição de processo padrão (*Standard Process Definition Document*).

O segundo módulo, *Intended Software Process Definition*, descreve a aplicação de conceitos do primeiro módulo, adaptando processos padrões ou atividades ao contexto de um projeto ou área organizacional específicos, considerando assim as peculiaridades desses. Nesses casos, processos podem ser agendados para serem executados em um determinado período de tempo. Este módulo também descreve a alocação de Stakeholders para a realização de atividades pretendidas, desempenhando um certo papel organizacional.

O terceiro módulo (*Software Process Execution*) compreende processos já executados, definindo conceitos como Processo Executado, Participação de Stakeholders,

Participação de Artefatos, Participação de Recursos e Participação de Procedimentos. Assim, este módulo define ações realizadas no passado, por algum agente que aplicou um processo ou procedimento, e pode ter utilizado artefatos e recursos.

A Figura 18 mostra um fragmento da SPO que define conceitos de **Stakeholder**, que pode assumir três subtipos diferentes: **Organization Stakeholder**, **Team Stakeholder** e **Person Stakeholder**. Estas classificações são reutilizadas por GORO e portanto serão discutidas na Seção 3.2.5.

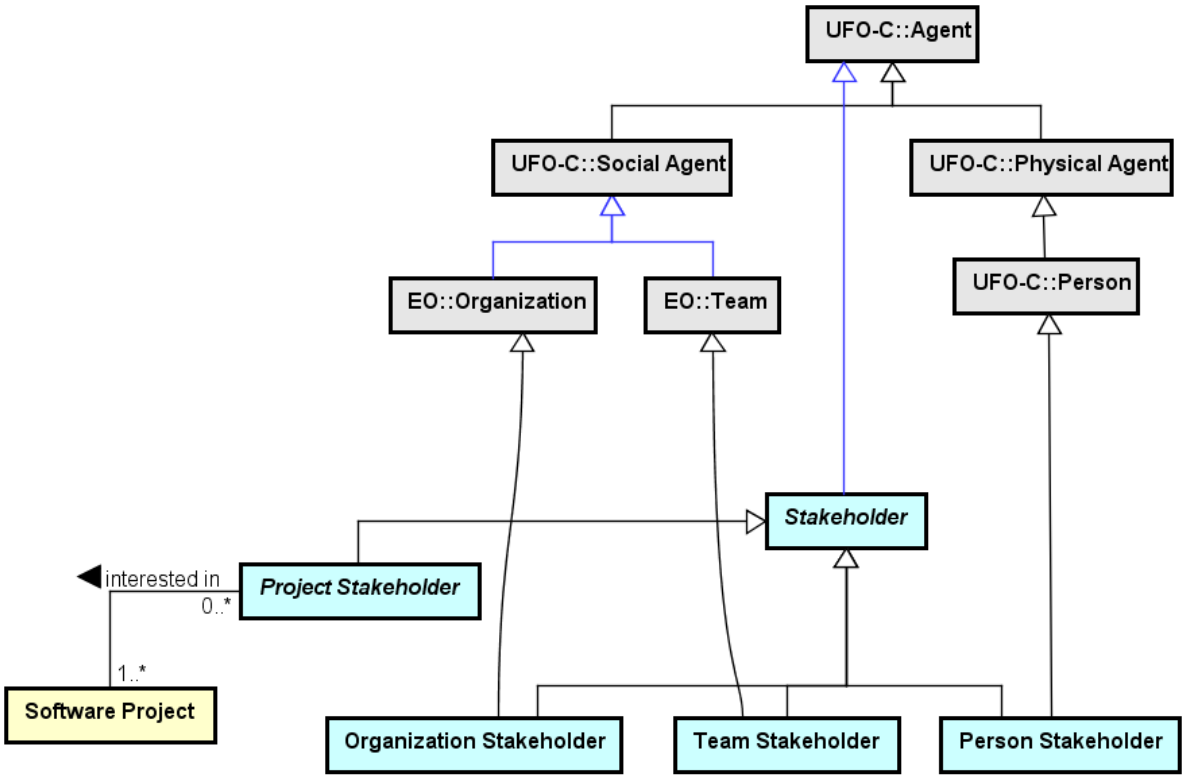


Figura 18 – Fragmento da SPO (RUY et al., 2016).

A Figura 19 mostra um trecho da SPO que define conceitos relacionados a participações de recursos, como o **Software Resource**, ou seja, o software que participa como recurso em uma atividade de Engenharia de Requisitos.

2.3.5 Reference Software Requirements Ontology

A *Reference Software Requirements Ontology* (DUARTE et al., 2018) (RSRO) tem foco na conceitualização de requisitos de software. A ontologia descreve requisitos como um objetivo a ser atingido, diferenciado-os entre requisitos funcionais e não funcionais. A ontologia também diferencia requisitos em duas visões: (i) o requisito representado como a propriedade mental de um agente e (ii) o requisito documentado como um artefato. Assim como a SPO, a RSRO também é parte da SEON.

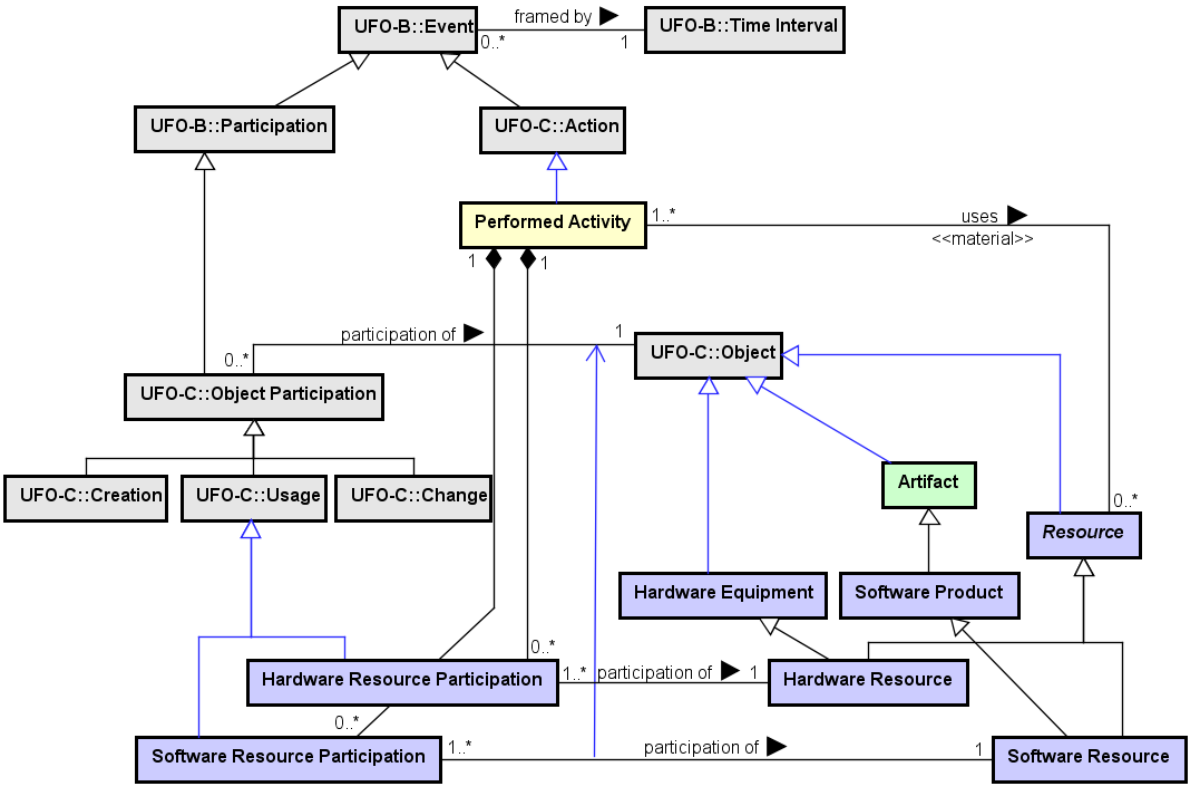


Figura 19 – Fragmento da SPO (RUY et al., 2016).

O conceito principal da RSRO é o **Requirement**, requisito descrito como um objetivo que representa as necessidades e expectativas dos agentes, descritas pelo **Stakeholder Intention**. Requisitos podem ser Funcionais (Functional Requirements), quando descrevem funções a serem implementadas no sistema, e Não-Funcionais (Non-functional Requirements), quando descrevem critérios e capacidades do sistema.

Na RSRO, requisitos são documentados em **Requirement Artifacts**, itens informacionais que descrevem requisitos, por sua vez agrupados pelo Documento de Requisitos (Requirements Document), artefato sob responsabilidade do Engenheiro de Requisitos (**Requirements Engineer**). Requisitos são elicitados a partir dos **Requirements Stakeholder**.

A Figura 20 mostra um fragmento da RSRO, onde é descrita a relação de intenções de uma **Stakeholder Intention** com um **Requirement**, e a descrição formal deste **Requirement** através de um **Requirement Artifact**.

2.4 SABiO

O SABiO (FALBO, 2014) (*Systematic Approach for Building Ontologies*) é um método para Engenharia de Ontologias composto por cinco fases, a saber:

Identificação de Propósito e Elicitação de Requisitos. Fase composta por

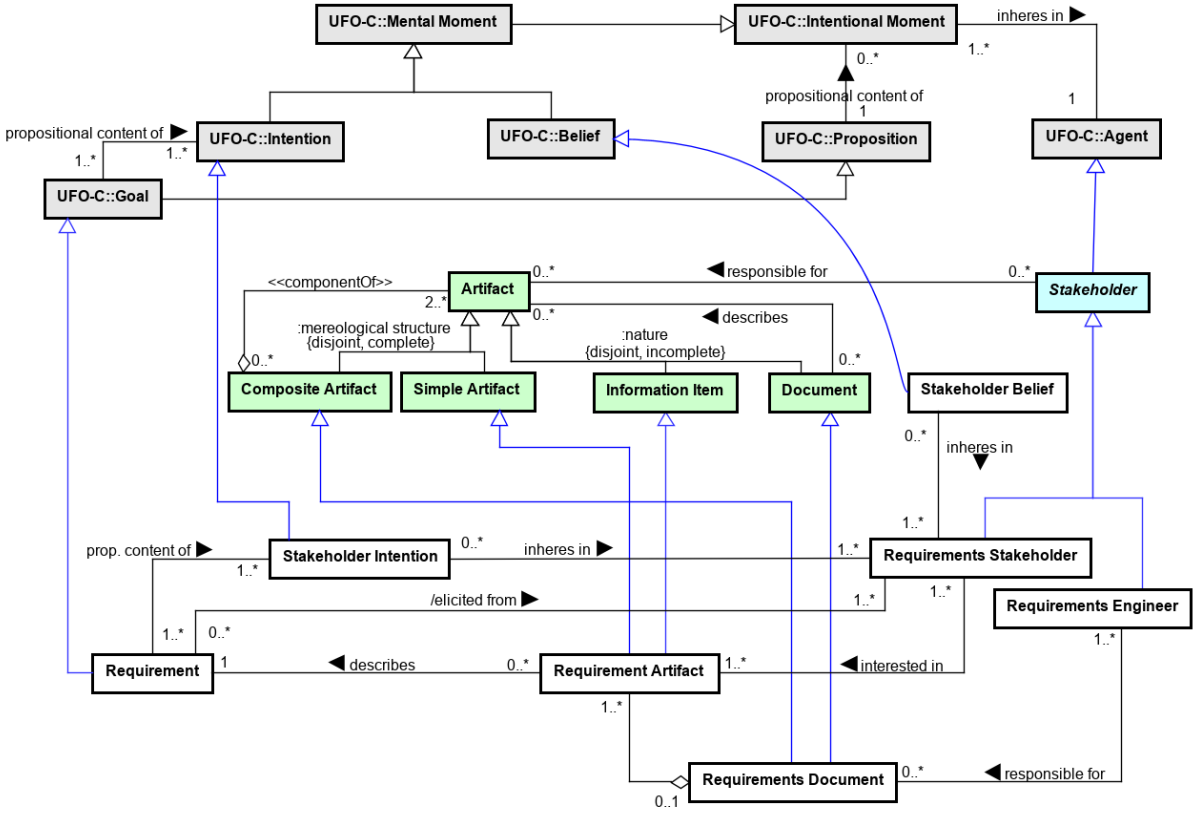


Figura 20 – Fragmento da RSRO (DUARTE et al., 2018).

quatro sub-fases. A primeira fase é a de Identificação do Propósito e Usos Pretendidos, na qual são identificados os casos em que a ontologia será aplicada. A segunda fase é a de Elicitação de Requisitos, na qual os requisitos não-funcionais desejados para a ontologia são elicitados. Requisitos não-funcionais referem-se a características, qualidades e aspectos gerais da ontologia. A terceira fase é a de Identificação das Questões de Competência (requisitos funcionais), na qual as questões de competência são levantadas utilizando diferentes estratégias (*top-down*, *bottom-up* ou *middle out*). Finalmente, a quarta fase é a de Modularização da Ontologia, realizada caso o domínio a ser modelado seja muito complexo, necessitando ser modularizado em partes menores.

Captura e Formalização da Ontologia. Guiada pelas QCs, o principal foco desta fase é a captura de conhecimento. Nesta fase, conceitos e relações relevantes à ontologia são identificados e organizados com suporte de uma ontologia de fundamentação. Esta fase inclui sub-fases de Modelagem Conceitual, na qual modelos são construídos para apoiar a comunicação e negociação com especialistas de domínio; Definição do Dicionário de Termos, na qual os termos utilizados pela ontologia são propriamente definidos; Definição Formal e Informal de Axiomas, caso necessário.

Design. Fase realizada caso seja necessária a implementação da ontologia operacional (para ser utilizada por máquinas). Nesta fase são definidas a linguagem de

implementação e requisitos funcionais e não-funcionais para a implementação.

Implementação. Neste fase a ontologia é implementada com base no que foi definido na fase anterior.

Teste. A ontologia é verificada e validada com base nos objetivos e questões de competência previamente levantados.

O método SABiO ainda descreve cinco processos de suporte, executados iterativamente e em paralelo às fases de desenvolvimento. São eles: Aquisição de Conhecimento (*Knowledge Acquisition*), Documentação (*Documentation*), Gerência de Configuração (*Configuration Management*), Avaliação (*Evaluation*) e Reuso (*Reuse*).

2.5 Trabalhos Relacionados

O conjunto inicial de trabalhos relacionados foi criado com base em referências relevantes da área, tais como (HORKOFF et al., 2016) e (GUIZZARDI et al., 2013a). Consideram-se trabalhos relacionados aqueles que utilizam ontologias como base para análise ou construção de linguagens de modelagem de objetivos ou aqueles que propuseram metamodelos para abordagens GORE, a fim de unificar conceitos ou diferentes linguagens dessa área.

Em relação ao uso de ontologias: a *Core Ontology for Requirements Engineering* (CORE) (JURETA; MYLOPOULOS; FAULKNER, 2009) tem como objetivo principal revisar a conceitualização de vários elementos de Engenharia de Requisitos. No entanto, CORE é baseada em uma ontologia de fundamentação na qual aspectos essenciais da modelagem conceitual (por exemplo, propriedades relacionais e a diferenciação entre relações formais e materiais) não receberam atenção suficientemente detalhada (NEGRI et al., 2017). A CORE é utilizada como base da linguagem Techne. Guizzardi et al. (2013a), por sua vez, usa o UFO como um modelo de referência semântica para analisar *i** e suas diversas variantes, buscando promover a interoperabilidade entre estas. Almendra et al. (2019) usa OWL-DL — uma versão operacional de OWL utilizada para descrever conceitos em lógica descritiva — para implementar a iStar2.0-OWL, criada com o objetivo principal de avaliar a consistência de modelos. Os dois últimos trabalhos, entretanto, estão restritos à família de linguagens *i**.

Negri et al. (2017) propõe uma ontologia para GORE, também denominada GORO, que foi proposta com o objetivo de fornecer semântica formal aos principais conceitos de GORE, e considerou um subconjunto de conceitos de três linguagens de modelagem de objetivos: *i**, KAOS e Techne. Assim, são identificadas algumas limitações em (NEGRI et al., 2017). Por ser baseada em um conjunto menor de linguagens e considerar apenas um subconjunto de elementos destas linguagens, a ontologia proposta carece de conceitos,

falhando também em relação à cobertura do domínio. A ontologia proposta também não foi submetida a uma prova de conceito. As deficiências da proposta de Negri et al. (2017) foram a principal motivação deste trabalho.

Quanto ao uso de metamodelos: no trabalho de Fayoumi, Kavakli e Loucopoulos (2015), os conceitos de oito linguagens de modelagem são levantados e organizados em um metamodelo, cujo principal objetivo é interoperabilidade entre os modelos GORE. O trabalho de Lucena et al. (2008) apresenta um metamodelo criado para unificar a versão original de *i** e Tropos, considerando as semelhanças e diferenças entre eles. O trabalho de Cares e Franch (2011) define um *super-metamodelo* criado com base em GRL e Tropos, que é validado por um algoritmo de tradução que usa o formato iStarML, baseado em XML. Patricio et al. (2011) propõe uma linguagem GORE unificada chamada UGL (Unified Goal-oriented Language), que incorpora conceitos de *i**, GRL e KAOS. Para propor a UGL, um metamodelo baseado nos metamodelos existentes de *i** e KAOS foi elaborado.

Diferentemente das ontologias, metamodelos não fornecem uma base semântica suficiente para explicar conceitos complexos do domínio. Metamodelos não são eficientes o suficiente para promover a interoperabilidade entre linguagens GORE pois, embora sejam estruturas poderosas para definir a sintaxe de uma linguagem, sofrem várias limitações em relação a definições semânticas (GUIZZARDI et al., 2013a), como por exemplo a falta de uma conceituação sobre aspectos gerais da realidade, que fundamente a definição de elementos do modelo de domínio específico.

No próximo capítulo, a proposta deste trabalho, assim como seu processo de construção, será apresentada e discutida em detalhes.

3 Proposta

Este capítulo aborda, na Seção 3.1, o processo de construção da GORO. Na Seção 3.2, a ontologia propriamente dita é descrita e detalhada.

3.1 Método

GORO foi construída seguindo o método SABiO, que descreve, em uma de suas fases, processos de elicitação de requisitos e questões de competência (QCs). Ao longo da construção de GORO, foram levantadas QCs relacionadas a três aspectos diferentes. O primeiro grupo de QCs são relativas aos aspectos que envolvem *stakeholders*/usuários e suas relações com requisitos de forma geral, e estão descritas a seguir:

- QC01: Como um *stakeholder* está envolvido no levantamento de requisitos?
- QC02: Quais os tipos de *stakeholder* que podem estar envolvidos no processo de software?
- QC03: Como um usuário está relacionado a um Requisito?
- QC04: Como um software participa na realização de um objetivo?

O segundo grupo de QCs está relacionado a construtos de GORE:

- QC05: O que é um objetivo?
- QC06: O que é uma pressuposição de domínio?
- QC07: O que é uma tarefa?
- QC08: O que é um obstáculo?
- QC09: O que é um recurso?

O terceiro grupo de questões lida com as possíveis relações que podem ser definidas entre os construtos GORE endereçados pelo grupo de QCs anterior:

- QC10: Como objetivos se relacionam entre si?
- QC11: Como objetivos podem ser satisfeitos?
- QC12: Como uma tarefa pode ser decomposta?

- QC13: Como um obstáculo pode ser divido em obstáculos menores?
- QC14: Como um obstáculo pode ser resolvido?
- QC15: Como um objetivo pode influenciar indiretamente na satisfação de outro?
- QC16: Como uma tarefa pode influenciar indiretamente na satisfação de um objetivo?
- QC17: Com quais outros objetivos pode um objetivo conflitar?

As questões de competência também são usadas para validar o escopo da ontologia e, por isso, serão também discutidas no Capítulo 4. A seguir, são descritos os Requisitos Não-Funcionais levantados para a GORO:

- **NFR1:** ser baseada em uma ontologia de fundamentação consolidada;
- **NFR2:** ser integrada com uma rede de ontologias relacionada ao domínio;
- **NFR3:** usar ontologias existentes de tópicos relacionados ao domínio;
- **NFR4:** ser construída usando a caracterização de domínio, considerando diferentes abordagens GORE.

O **NFR1** foi definido pois, além das vantagens providas pelo uso de uma ontologia de fundamentação, diversos trabalhos de ontologias para a área de Engenharia de Software também fazem utilização de alguma ontologia deste tipo, o que facilitaria assim o reúso e interoperabilidade com outras ontologias. É possível verificar que GORO satisfaz o **NFR1** pois a ontologia é fundamentada em UFO, que foi escolhida por ter sido utilizada como ontologia de fundamentação de diversas ontologias que poderiam ser reutilizadas por GORO.

O **NFR2** foi elicitado com o objetivo de melhorar a qualidade da ontologia em aspectos relativos a interoperabilidade, cobertura de domínio e visibilidade da ontologia perante a comunidade. Para satisfazer o **NFR2**, GORO foi integrada à SEON. A SEON foi escolhida pois, além de ser relacionada ao domínio, já integra quatro das cinco ontologias que foram são reutilizadas por GORO.

O **NFR3** foi definido pois reúso é característica importante do método SABiO, que frisa diversas vantagens relativas ao reúso: evita redundância de conceitos, evita retrabalho e melhora a interoperabilidade da ontologia com outras, facilitando que a mesma seja também reutilizada em outros trabalhos. Por reutilizar cinco ontologias diferentes (ASMP, NFRO, COVR, SPO e RSRO), GORO satisfaz o **NFR3**.

Por fim, o **NFR4** foi definido visando a cobertura de domínio e utilidade da ontologia. Ao considerar linguagens consolidadas em GORE, a ontologia trata de conceitos

já comuns para a comunidade. Assim, durante o processo de caracterização de domínio, nove linguagens GORE foram analisadas (NFR Framework, KAOS, *i**, GBRAM, GSN, Tropos, GRL, Techne e iStar), satisfazendo o **NFR4**. É importante mencionar que as linguagens foram selecionadas com base em revisões de literatura sobre GORE (vide Seção 2.1). A escolha dessas linguagens foi validada com especialistas em domínio, um grupo de cinco profissionais acadêmicos com mais de dez anos de experiência em modelagem GORE, que nos aconselharam a considerar *i** e *iStar* como linguagens diferentes, dadas as diferenças entre elas. GORO, como o nome indica, é focada em requisitos e, portanto, não foram consideradas outras abordagens que usam construtos relacionados a objetivos, mas não se definem especificamente como linguagens GORE.

A análise das diferentes linguagens GORE mostrou uma variedade de construtos nas linguagens, alguns mais comuns que outros. Para definir o escopo de GORO, foram aplicados dois critérios para a inclusão (CI) desses construtos. O primeiro critério (CI_1) determina que construtos devem aparecer em pelo menos duas linguagens de modelagem GORE, enquanto o segundo critério (CI_2) estabelece que o construto deve ser considerado um conceito GORE pelo grupo de especialistas de domínio (previamente descrito). Esses critérios foram aplicados para excluir construtos que não eram GORE, mas na verdade recursos extras de uma determinada linguagem. Em seguida, para verificar se diferentes construtos compartilhavam o mesmo significado, o grupo de especialistas de domínio também foi consultado. A lista dos conceitos que foram considerados seguindo os critérios mencionados também está disponível no material suplementar.

Com base no método descrito, a GORO foi construída e modularizada de acordo com grupos de características GORE: objetivos; atores e suas relações com objetivos; tarefas, operacionalizações e relações entre objetivos; pressuposições de domínio; e obstáculos, conflitos e contribuições. A ontologia e seus respectivos módulos são descritas na seção a seguir.

3.2 Goal-Oriented Requirements Ontology

Conforme mencionado, a GORO foi construída seguindo as diretrizes do método SABiO, que propõe a organização de ontologias em módulos, apresentados na Figura 21. A GORO é composta por cinco módulos: *Mental Moments* e Objetivos (*Mental Moments and Goals*; Usuários e Stakeholders (*Users and Stakeholders*); Pressuposições de Domínio (*Assumptions*); Obstáculos, Conflitos e Contribuições (*Obstacles, Conflicts and Contributions*); e Tarefas, Refinamentos e Recursos (*Tasks, Refinements and Resources*). A Figura 21 também faz referência às ontologias reutilizadas e à ontologia de fundamentação utilizada por elas. Todos os módulos serão explicados em mais detalhes nas subseções à seguir. Por representarem construtos de linguagens propostas, os conceitos da ontologia aqui

representados estão em inglês.

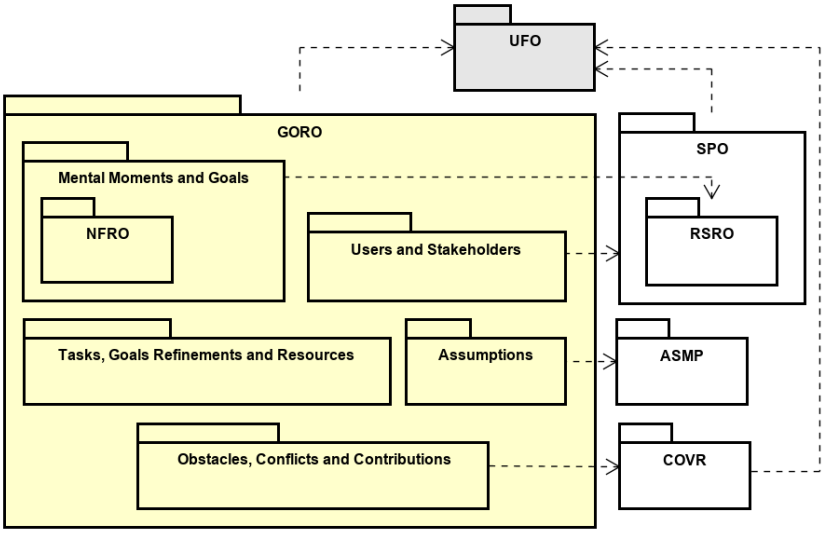


Figura 21 – Módulos de GORO.

3.2.1 *Mental Moments* e Objetivos

A Figura 22 apresenta o primeiro módulo da ontologia, com foco em objetivos representados como propriedades mentais existencialmente dependentes em agentes. Um Mental Moment pode ser especializado em três classificações: Belief, Desire ou Intention. Um Belief é uma crença de um agente dada como verdadeira em um determinado conjunto de situações. A diferenciação entre Desire ou Intention é dada pelo compromisso. Enquanto um Desire é apenas uma vontade, Intention é um compromisso interno, conforme discutido na Seção 2.2.3.

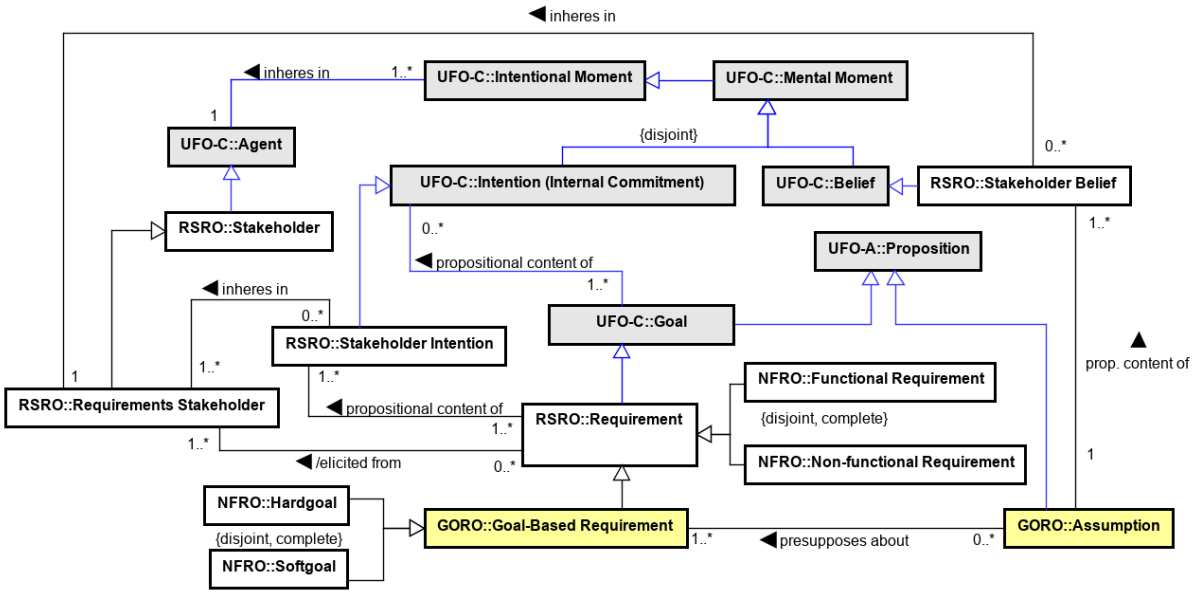


Figura 22 – Módulo de GORO sobre Objetivos.

Um Goal é visto como o conteúdo proposicional (*propositional content of*) de uma Intention, que é inerente a um Agent, supertipo (*supertype*) de Stakeholder. Assim, um Requirement é um objetivo elicitado (*elicited from*) a partir da intenção de um Requirement Stakeholder. Este último é definido como um *Stakeholder* que provê necessidades e expectativas para um projeto de engenharia de requisitos.

Em GORO, requisitos podem ser classificados em Requisitos Funcionais (Functional Requirement) e Requisitos Não-funcionais (Non-Functional Requirement). Ao se aplicar uma abordagem GORE a um processo de Engenharia de Requisito, tem-se um Goal-Based Requirement (GBR). Em outras palavras, um GBR é um requisito elicitado em um processo de Engenharia de Software que especificamente aplica orientação a objetivos na etapa de Engenharia de Requisitos, enquanto um Requirement é um requisito elicitado em um processo de Engenharia de Software de forma geral, ainda que, em última análise, ambos sejam objetivos (Goal). Um GBR pode ser do tipo Hardgoal, quando provê critérios claros de satisfação, e Softgoal caso contrário. Essas quatro últimas classificações foram incorporadas a partir da NFRO (GUIZZARDI et al., 2014) e, por isso, são representadas com esse prefixo.

Ao se combinar ortogonalmente as quatro classificações de NFRO, temos um Functional Requirement & Hardgoal, um Functional Requirement & Softgoal, um Non-functional Requirement & Hardgoal e um Non-functional Requirement & Softgoal. Estas combinações estão implicitamente representadas na Figura 22. Essas quatro combinações ortogonais permitem classificar objetivos quando ao seu critério de satisfação (*hard/soft*) e quanto ao fato de se referirem a funcionalidades ou qualidades de softawre (*functional/non-functional*).

3.2.2 Tarefas, Refinamentos e Recursos

A Figura 23 exibe o módulo de GORO focado na descrição de Tarefas, Refinamentos e Recursos. Uma tarefa (Task) é uma ação (Action Universal (Plan)) que pretende operacionalizar (*intends to operationalize*) um objetivo (GBR). Tarefas e objetivos podem ser do tipo complexo ou atômico. Uma Complex Task, por exemplo, é uma tarefa composta por duas ou mais tarefas. No caso de um Complex Goal-Based Requirement (Complex GBR), este pode ser do tipo And Complex GBR ou Or Complex GBR, obedecendo à tradicional lógica de refinamento comum a diversas linguagens GORE. Tasks podem consumir (*requires*) ou produzir (*produces*) recursos.

3.2.3 Pressuposições de Domínio

Este módulo reutiliza a ontologia ASMP em sua totalidade. A mesma foi descrita em detalhes na Seção 2.3.1 e é mostrada na Figura 24.

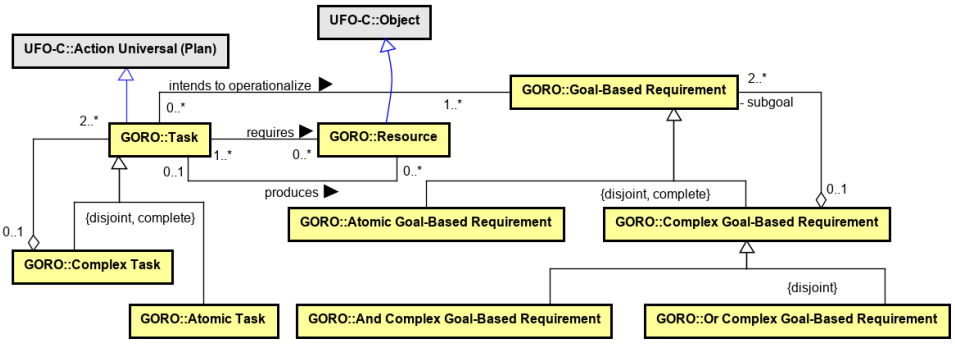


Figura 23 – Módulo de GORO sobre Tarefas, Refinamentos e Recursos.

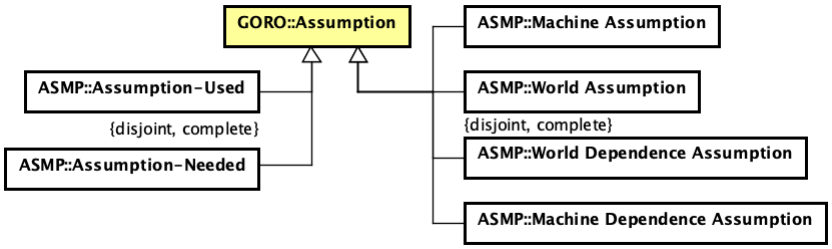


Figura 24 – Módulo de GORO sobre Pressuposições de Domínio.

3.2.4 Obstáculos, Conflitos e Contribuições

A Figura 25 apresenta o módulo de GORO que descreve conceitos relacionados a obstáculos, conflitos e contribuições. Lamsweerde e Letier (2000) descrevem obstáculos como a versão oposta a objetivos: "enquanto objetivos capturam condições desejadas, obstáculos capturam condições indesejadas". Essa definição é suficientemente similar à definição de Threat Event descrita pela COVR (uma situação (indesejada) possível no futuro em que algo de valor é posto em risco). GORO define um Obstacle como uma Threatening Proposition, visto que objetivos também são tratados como proposições. Uma Threatening Proposition satisfaz (*satisfies*) uma Threatening Situation da mesma forma que uma Proposition satisfaz (*satisfies*) uma Situation, portanto, a primeira relação deriva da última (a derivação é denotada pelo símbolo /). Um Obstacle obstrui (obstructs) a satisfação de um Goal-Based Requirement, visto que um este último também é uma proposição. Assim, tem-se uma relação entre proposições em que uma representa risco para outra, ou seja, a situação que satisfaz uma implica em uma situação de risco para a outra.

Lamsweerde e Letier (2000) argumentam que um Obstáculo pode ser mitigado por um objetivo. Em GORO, considera-se que tarefas (ações) na verdade mitigam obstáculos (um objetivo que mitiga um obstáculo pode estar implicitamente representando a tarefa que atende a este objetivo) e, portanto, propõe-se dois tipos de tarefa de mitigação: Mitigation Task, que são tarefas realizadas para reduzir ou prevenir a probabilidade do acontecimento de um Obstacle; e Contingency Task, tarefas realizadas para reduzir os danos causados caso uma situação de risco Threatening Situation se torne verdadeira.

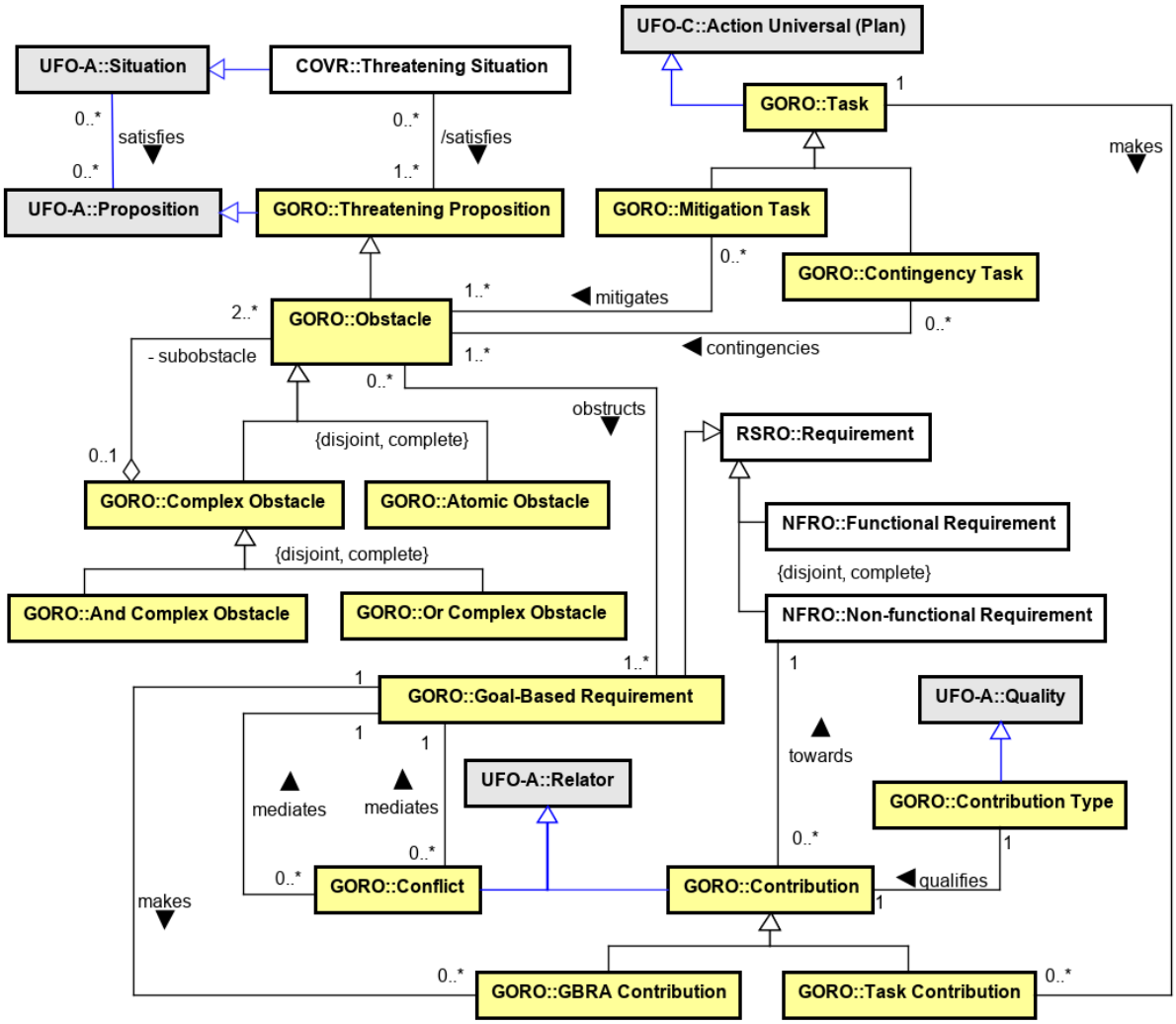


Figura 25 – Módulos de GORO sobre Obstáculos, Conflitos e Contribuições.

Similarmente a tarefas e objetivos, obstáculos podem ser decompostos em complexos e atômicos (Complex Obstacles e Atomic Obstacles). Um obstáculo complexo pode ser do tipo *And* ou *Or*. Um Or Complex Obstacle descreve situações alternativas em que o obstáculo descrito torna-se verdadeiro, enquanto um And Complex Obstacle descreve um conjunto de situações em que todas elas necessitam tornar-se verdadeiras para que um obstáculo também se torne.

Um conflito é um tipo de relação que conecta objetivos que não podem ser atingidos conjuntamente. Em outras palavras, dados dois objetivos conflitantes G_1 e G_2, pertencentes a um modelo M, não existe um conjunto solução S de M que contenha ambos G_1 e G_2. GORO modela conflito (Conflict) como um Relator que relaciona dois GBRs.

É importante salientar a diferença entre conflitos e obstáculos. Enquanto o primeiro descreve uma relação entre duas situações desejadas que não podem ser atingidas conjuntamente, o segundo descreve situações que são indesejadas na realidade.

GORO introduz o conceito de contribuição (Contribution), que acontece entre objetivos/tarefas (GBRs/Tasks) e Requisitos Não-funcionais (Non-functional Goal-Based Requirement). Contribuições devem ter somente Requisitos Não-funcionais como destino pois: (i) no caso de contribuições totais, uma contribuição negativa para um Functional Requirement seria semanticamente semelhante a um Conflito, enquanto uma contribuição positiva teria o mesmo significado que um Complex GBR ou operacionalização; (ii) no caso de contribuições parciais, não faz sentido satisfazer/negar parcialmente um GBR que, por sua vez, possui critérios de satisfação precisos. Várias linguagens GORE têm certos tipos de relações de contribuição: *i**, iStar e o NFR Framework, por exemplo, definem contribuições com base nos seguintes labels: *make*, *help*, *hurt* e *break* (YU, 1996; DALPIAZ; FRANCH; HORKOFF, 2016; MYLOPOULOS; CHUNG; NIXON, 1992). Uma Contribution pode ter propriedades relacionadas a intensidade (por exemplo, parcial ou total) e tipo (positivo ou negativo). Tais propriedades não são representadas em GORO, mas adicionadas na implementação da ontologia.

3.2.5 Stakeholders

GORO importa o conceito de Stakeholder de SPO, que é definido como um Agent interessado ou afetado pelo processo de software ou seus resultados. Um Stakeholder pode assumir três classificações diferentes: Organization Stakeholder, que representa uma organização envolvida no processo de requisitos de software, como por exemplo um cliente ou fornecedor; um Team Stakeholder, no caso em que deseja-se representar um grupo de pessoas com um mesmo propósito definido, como por exemplo um grupo de desenvolvedores; e Person Stakeholder, no caso em que deseja-se falar de um indivíduo específico, como um programador ou usuário.

GORO herda o conceito Requirements Stakeholder de RSRO para definir o Stakeholder diretamente envolvido no processo de levantamento de Requisitos. De um Requirements Stakeholder, elicitam-se Requirements. Conforme explicado na Subseção 3.2.1, ao aplicar essa atividade a um processo orientado a objetivos, tem-se um Goal-Based Requirement e, portanto, um Requirements Stakeholder diretamente envolvido no processo de levantamento de objetivos e, por consequência, Assumptions, Obstacles e Tasks.

GORO possui o conceito User para definir o usuário envolvido no processo de detalhamento de objetivos, mas que não teve seus requisitos diretamente elicitados, podendo mesmo assim ter algum objetivo ou ser associado à execução de uma tarefa. Por exemplo, o usuário de um editor de código tem seus próprios objetivos com a ferramenta (ex. *Ter código indentado automaticamente*), mas também deve executar a tarefa *Escrever código* para satisfazer o objetivo *Ter Software Y pronto para venda* que pertence ao Organization Stakeholder *Fábrica de Software LTDA*. Assim também, o editor de código pode oferecer a opção de trocar a cor da *interface*, pois a equipe que criou o editor de código pode ter

previsto que o usuário poderia ter o objetivo de adaptar a interface para seu conforto visual.

Por fim, GORO também traz o conceito de Software Actor, sendo definido como um Software Product que participa na realização de uma tarefa. Por exemplo, um dos objetivos de *Fábrica de Software LTDA* é ter o trabalho dos desenvolvedores salvo a cada 1 hora e a tarefa que operacionaliza este objetivo é realizada automaticamente pelo Software Actor *Editor de Código*.

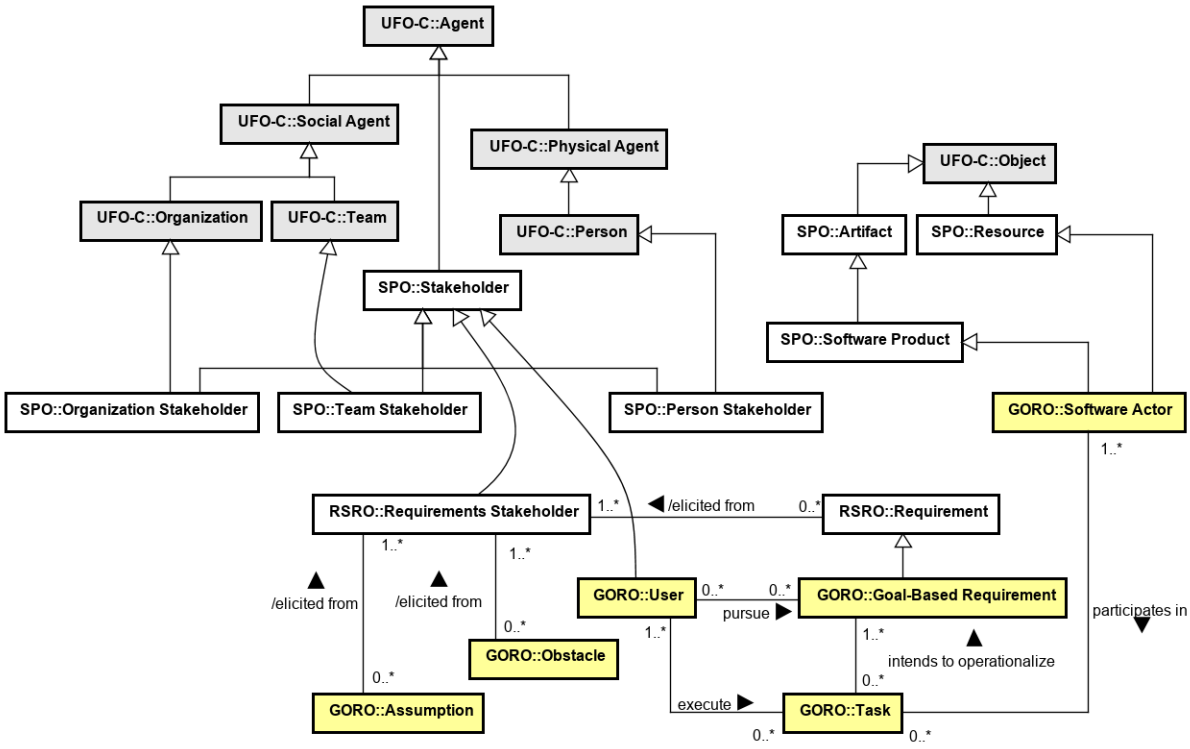


Figura 26 – Módulos de GORO sobre Usuários e Stakeholders.

Este capítulo descreveu os quatro módulos da GORO, explicando os elementos e relações definidos em cada um. No capítulo a seguir, os módulos apresentados são submetidos a processos de verificação, validação e avaliação. Estas fases tem por objetivo determinar se o escopo da ontologia está correto, ou seja, se os módulos apresentados cobrem o domínio adequadamente, bem como se a ontologia atende a todos os seus requisitos. Além disso, os processos que serão detalhados a seguir são úteis para averiguar a se os conceitos e relações foram definidos corretamente.

4 Avaliação do Trabalho

GORO foi validada e verificada conforme as diretivas de SABiO. Além disso, a ontologia foi avaliada em duas aplicações práticas: ao ser usada em um experimento de modelagem e também ao ser utilizada como base da construção de uma ferramenta de conversão de modelos. A seguir, a Seção 4.1 descreve como a ontologia foi verificada em termos de questões de competência, a Seção 4.2 apresenta como a ontologia foi validada em temos de mapeamento de conceitos das linguagens GORE para GORO, e por fim, a Seção 4.3 descreve os dois casos em que a ontologia foi avaliada em uso prático. Por fim, a Seção 4.4 descreve alguns problemas observados nas linguagens GORE durante o processo de validação da GORO.

4.1 Verificação

Conforme descrito em SABiO, o processo de verificação de um modelo conceitual verifica se a ontologia consegue responder as Questões de Competência levantadas no início do processo de engenharia da mesma. É importante frisar que as questões de competência são levantadas antes da construção da ontologia, e então verifica-se se a ontologia construída é capaz de respondê-las. A seguir, o resultado deste processo é apresentado e discutido.

Tabela 2 – Tabela de Questões de Competência - Grupo 1.

Questão	Resposta
QC01: Como um *stakeholder* está envolvido no levantamento de requisitos?	Requirement *elicited from* Requirement Stakeholder (Fig. 26)
QC02: Quais os tipos de *stakeholder* que podem estar envolvidos no processo de software?	Organization Stakeholder *subtype of* Stakeholder; Team Stakeholder *subtype of* Stakeholder; Person Stakeholder *subtype of* Stakeholder (Fig. 26)
QC03: Como um usuário está relacionado a um requisito?	User *pursue* Goal-Based Requirement *subtype of* Requirement; User *execute* Task *intends to operationalize* Goal-Based Requirement *subtype of* Requirement (Fig. 26)
QC04: Como um software participa na realização de um objetivo?	Software Actor *participates in* Task *intends to operationalize* Goal-Based Requirement *subtype of* Requirement (Fig. 26)

A Tabela 2 traz o grupo de questões de competência que lida com as definições e relações de Atores, Stakeholders e outros. Esse grupo de questões de competência foi levantado ao se analisar linguagens que trazem o conceito de Agentes (entidades que realizam objetivos ou são impactadas por ele), como as linguagens KAOS, *i**/iStar e GBRAM.

Na análise, foi possível perceber que as distinções realizadas por cada linguagem eram diferentes. Enquanto algumas consideravam a divisão entre humano e máquina (KAOS), outras dividiam atores em entidades físicas e papéis (*i**/iStar) ou partes interessadas e partes afetadas (GBRAM). As questões levantadas tentam abranger e organizar esses conceitos, definindo papéis para partes interessadas (QC02), dividindo as relações entre partes interessadas e afetadas (QC03) e considerando os casos de atores humanos e não humanos (QC04).

Tabela 3 – Tabela de Questões de Competência - Grupo 2.

Questão	Resposta
QC05: O que é um objetivo?	Goal-Based Requirement *subtype of* Requirement, *subtype of* Goal; Goal *propositional content of* Intention ou Desire; Requirement *propositional content of* Stakeholder Intention ou Stakeholder Desire (Fig. 22)
QC06: O que é uma tarefa?	Task *subtype of* Action Universal (Plan); Task *intends to operationalize* Goal-Based Requirement (Fig. 23)
QC07: O que é uma pressuposição de domínio?	Assumption *propositional content of* Stakeholder Belief; Stakeholder Belief *inheres in* Requirements Stakeholder (Fig. 24)
QC08: O que é um obstáculo?	Obstacle *obstructs* Goal-Based Requirement (Fig. 25)
QC09: O que é um recurso?	Resource *subtype of* Object; Task *requires* Resource; Task *produces* Resource; (Fig. 23)

A Tabela 3 traz questões de competência relacionadas aos elementos GORE comumente encontrados nas linguagens de modelagem objetivos. Todas as linguagens investigadas descrevem o conceitos de objetivo e tarefa (QC05, QC06), seja de forma mais geral ou detalhada. Muitas linguagens, como Techne e KAOS, trazem também o conceito de pressuposições de domínio e obstáculo (QC07, QC08). O conceito de recurso (QC09) é bem difundido entre as linguagens da família *i**.

A Tabela 4 mostra questões de competência criadas para endereçar as relações entre os elementos GORE. De forma geral, percebeu-se que alguns elementos relacionavam-se com elementos do mesmo tipo, aumentando o nível de detalhamento entre o elemento de origem e destino da relação. Esse tipo de refinamento acontece, em geral, em objetivos, tarefas e obstáculos (QC10, QC12 e QC13). Em todas as linguagens, objetivos são operacionalizados por tarefas (QC11). Além disso, elementos podem se relacionar com outros elementos através de relações de obstáculo, conflito e contribuições indiretas (QC15, QC16 e QC17). Algumas linguagens, como KAOS, também tratam formas de contingência e mitigação de obstáculos (QC14).

Tabela 4 – Tabela de Questões de Competência - Grupo 3.

Questão	Resposta
QC10: Como objetivos se relacionam entre si?	Complex Goal-Based Requirement *subtype of* Goal-Based Requirement; Atomic Goal-Based Requirement *subtype of* Goal-Based Requirement; Complex Goal-Based Requirement *composed of* Goal-Based Requirement; And Complex Goal-Based Requirement *subtype of* Complex Goal-Based Requirement; Or Complex Goal-Based Requirement *subtype of* Complex Goal-Based Requirement (Fig. 23)
QC11: Como objetivos podem ser satisfeitos?	Task *subtype of* Action Universal (Plan); Task *intends to operationalize* Goal-Based Requirement (Fig. 23)
QC12: Como uma tarefa pode ser decomposta?	Complex Task *subtype of* Task; Atomic Task *subtype of* Task; Complex Tasks *composed of* Task (Fig. 23)
QC13: Como um obstáculos pode ser divido em obstáculos menores?	Complex Obstacle *subtype of* Obstacle; Atomic Obstacle *subtype of* Obstacle; Complex Obstacle *composed of* Obstacle; And Complex Obstacle *subtype of* Complex Obstacle; Or Complex Obstacle *subtype of* Complex Obstacle (Fig. 25)
QC14: Como um obstáculo pode ser resolvido?	Mitigation Task *subtypes of* Task; Contingency Task *subtypes of* Task; Mitigation Task *mitigates* Obstacle; Contingency Task *contingencies* Obstacle (Fig. 25)
QC15: Como um objetivo pode influenciar indiretamente na satisfação de outro?	Goal-Based Requirement *makes* GBRA Contribution *towards* Non-functional Requirement *subtype of* Requirement *supertype of Goal-Based Requirement* (Fig. 25)
QC16: Como uma tarefa pode influenciar indiretamente na satisfação de um objetivo?	Task *makes* Task Contribution *towards* Non-functional Requirement *subtype of* Requirement *supertype of Goal-Based Requirement* (Fig. 25)
QC17: Com quais outros objetivos pode um objetivo conflitar?	Goal-Based Requirement *conflicts with* Goal-Based Requirement (Fig. 25)

4.2 Validação

Para validar GORO, verificou-se se os elementos das linguagens GORE levantadas podem ser devidamente mapeados para a ontologia. A Tabela 5 mostra o mapeamento realizado: na primeira coluna estão descritos os conceitos de GORO, enquanto na segunda são listados os conceitos equivalentes de cada linguagem, marcados com os nomes de cada uma. Esta atividade também foi realizada com suporte do grupo de especialistas de domínio, seguindo um processo pré-estabelecido: primeiramente a definição dos conceitos de cada linguagem foi extraída dos trabalhos que propunham cada uma, então os especialistas

de domínio discutiram sobre cada definição e selecionaram o mapeamento apropriado. Uma definição completa de cada conceito com a referência bibliográfica apropriada está disponível como material suplementar no site do projeto.[1]

Tabela 5 – Mapeamentos entre os conceitos das linguagens GORE para GORO.

GORO	**Conceitos de Linguagens GORE**
Assumption	i*::Belief, GRL::Belief, Tropos::Belief, KAOS::Domain Property, Techne::Domain Assumption, GSN::Assumption, GSN::Context
Task *componentOf* Complex Task	i*::Decomposition Link, GRL::Decomposition Link, Tropos::Decomposition Link, iStar::Refinement Link, NFR::Decomposition, Techne::Inference
conflicts with	KAOS::Conflict, Techne::Conflict, NFR::Conflict
Contribution	i*::Contribution Links, GRL::Contribution Links, Tropos::Contribution Links, iStar::Contribution Links, NFR::Contribution Decomposition
Functional Requirement & Hard-goal	i*::Goal, GRL::Goal, Tropos::Goal, iStar::Goal, Techne::Goal, KAOS::Goal, KAOS::Expectation, KAOS::Requirement, NFR::Functional Requirement, GSN::Goal, GBRAM::Achievement Goal, GBRAM::Requirement
GBR *componentOf* And Complex Goal-Based Requirement	i*::Decomposition Link (AND), GRL::Decomposition Link (AND), Tropos::Decomposition Link (AND), iStar::Refinement Link (AND), KAOS::Refinement (AND), NFR::Decomposition (AND), GSN::Supported-by, GBRAM::Decomposition
GBR *componentOf* Or Complex Goal-Based Requirement	i*::Decomposition Link (OR), GRL::Decomposition Link (OR), Tropos::Decomposition Link (OR), iStar::Refinement (OR), KAOS::Refinement (OR), Techne::Inference, NFR::Decomposition (OR)
intends to operationalize	i*::Means-end, GRL::Means-end, Tropos::Means-end, iStar::Refinement, KAOS::Operationalization, NFR::Operationalization, GBRAM::Operationalization, GSN::SupportedBy, Techne::Inference

[1] <https://nemo.inf.ufes.br/projects/rose/>

Tabela 5 – Mapeamentos entre os conceitos das linguagens GORE para GORO.

GORO	**Conceitos de Linguagens GORE**
Non-functional Requirement & Hardgoal	KAOS::Goal, KAOS::Expectation, KAOS::Requirement, Techne::Quality Constraint, GSN::Goal, GBRAM::Constraint Goal
Non-functional Requirement & Softgoal	i*::Softgoal, GRL::Softgoal, Tropos::Softgoal, iStar::Quality, KAOS::Goal, KAOS::Expectation, KAOS::Requirement, Techne::Softgoal, NFR::Softgoal, GSN::Goal, GBRAM::Maintenance Goal
Obstacle	KAOS::Obstacle, GBRAM::Obstacle
Requirements Stakeholder, Organization Stakeholder, Team Stakeholder or Person Stakeholder	i*::Actor, GRL::Actor, Tropos::Actor, iStar::Actor, KAOS::Agent, GBRAM::Agent
Resource	i*::Resource, GRL::Resource, Tropos::Resource, iStar::Resource, GSN::Context
Software Actor	i*::Actor, GRL::Actor, Tropos::Actor, iStar::Actor, KAOS::Agent
Task	i*::Task, GRL::Task, Tropos::Plan, iStar::Task, KAOS::Operation, Techne::Task, NFR::Operationalizing Softgoal, GSN::Solution
User	i*::Actor, GRL::Actor, Tropos::Actor, iStar::Actor, KAOS::Agent, GBRAM::Agent

4.3 Avaliação

GORO foi avaliada em duas formas para checar sua capacidade de servir como interlíngua entre diferentes modelos GORE. A Subseção 4.3.1 descreve o experimento de modelagem realizado no contexto deste trabalho, enquanto a Subseção 4.3.2 descreve a ferramenta desenvolvida utilizando uma versão operacional de GORO.

4.3.1 Experimento

Para validar o escopo de GORO, foi conduzido um experimento de modelagem no qual participantes tiveram de criar modelos em diferentes linguagens — iStar, KAOS, Techne e GSN — dada a descrição e domínio de um Sistema de Caixa Eletrônico (conforme descrito no Capítulo 2). As linguagens iStar, KAOS e Techne foram selecionadas devido ao fato de serem populares no meio acadêmico (HORKOFF et al., 2016), enquanto GSN

foi selecionada dada sua popularidade na indústria (KELLY; WEAVER, 2004). Outras linguagens não foram incluídas no experimento por diferentes motivos. O NFR Framework não foi utilizado pois foca somente em requisitos não-funcionais, assim seria difícil comparar os modelos produzidos. GBRAM não foi incluída por não apresentar um modelo gráfico. Por fim, outras versões ou dialetos de iStar (*i**, GRL, Tropos) foram desconsiderados dada sua semelhança com a mesma.

O principal objetivo do experimento foi levantar um grupo de diferentes modelos de objetivos em linguagens diferentes, porém referentes ao mesmo domínio. Esse grupo de modelos teria então duas finalidades: (i) ser utilizado para validar a GORO, colocando em teste sua capacidade de interoperabilidade e cobertura de domínio (questão de pesquisa principal); e (ii) servir como um artefato para a comunidade GORE, podendo ser utilizado para fins didáticos ou para aplicações de pesquisa (objetivo secundário).

Oito pessoas foram convidadas para participar do experimento. O perfil destes participantes é formado por pessoas com considerável experiência em modelagem GORE: três participantes tem mais de dois anos de experiência em modelagem, enquanto o restante tem entre 1 e 2 anos de experiência. Um dos participantes já usou pelo menos uma linguagem GORE em âmbito profissional, enquanto o restante usou apenas em meio acadêmico.

Depois que os participantes concordaram em participar do experimento, receberam uma descrição geral do sistema, extraída de <http://www.math-cs.gordon.edu/courses/cs211/ATMExample/>, e um exemplo de objetivos extraído de (WANG et al., 2007), que não segue a sintaxe de uma linguagem GORE específica. Os participantes tiveram duas semanas para produzir, individualmente, modelos de objetivos de duas linguagens GORE selecionadas previamente. Portanto, cada linguagem foi utilizada por quatro participantes diferentes.

Em seguida, os modelos produzidos foram por nós analisados e passaram pelo seguinte processo, representado na Figura 27:

1. Partes inconsistentes dos modelos, como por exemplo relações incorretas entre elementos, foram removidas;

2. Um modelo foi produzido para cada linguagem com base na maior interseção possível encontrada entre os modelos daquela linguagem recebidos. Esses modelos são chamados de modelo unificado. Por exemplo, os modelos de *iStar* recebidos dos participantes foram comparados e deles foi extraído o Modelo Unificado de *iStar*;

3. Para cada modelo unificado, três outros modelos foram produzidos utilizando GORO como interlíngua. Ou seja, elementos de cada modelo unificado foram mapeados para GORO e, a partir do mapeamento, os modelos destino foram extraídos nas

Tabela 6 – Comparação entre modelos do experimento.

Origem	**Destino**	**Orig. Original**		**Dest. Original**		**Dest. Traduz. (GORO)**	
		Elm.	**Rel.**	**Elm.**	**Rel.**	**Elm**	**Rel.**
iStar	KAOS	43	45	44	41	31 (70%)	27 (66%)
iStar	Techne	43	45	42	44	37 (88%)	32 (73%)
iStar	GSN	43	45	49	38	35 (71%)	34 (89%)
KAOS	Techne	44	41	42	44	41 (98%)	43 (98%)
KAOS	GSN	44	41	49	38	39 (80%)	37 (97%)
Techne	GSN	42	44	49	38	34 (69%)	33 (87%)

outras linguagens utilizadas no experimento. Por exemplo, o modelo unificado de *iStar* foi mapeado para a GORO e então os modelos de KAOS, Techne e GSN foram reconstruídos a partir desse mapeamento;

4. Para cada linguagem, os modelos produzidos a partir do mapeamento dos modelos unificados para a GORO foram comparados com os próprios modelos unificados de cada, considerando o número de elementos e relações equivalentes. Por exemplo, o modelo de *iStar* produzido a partir do mapeamento do modelo KAOS (unificado) para GORO foi comparado com o modelo *iStar* (unificado). O resultado desta comparação pode ser visto na Tabela 6.

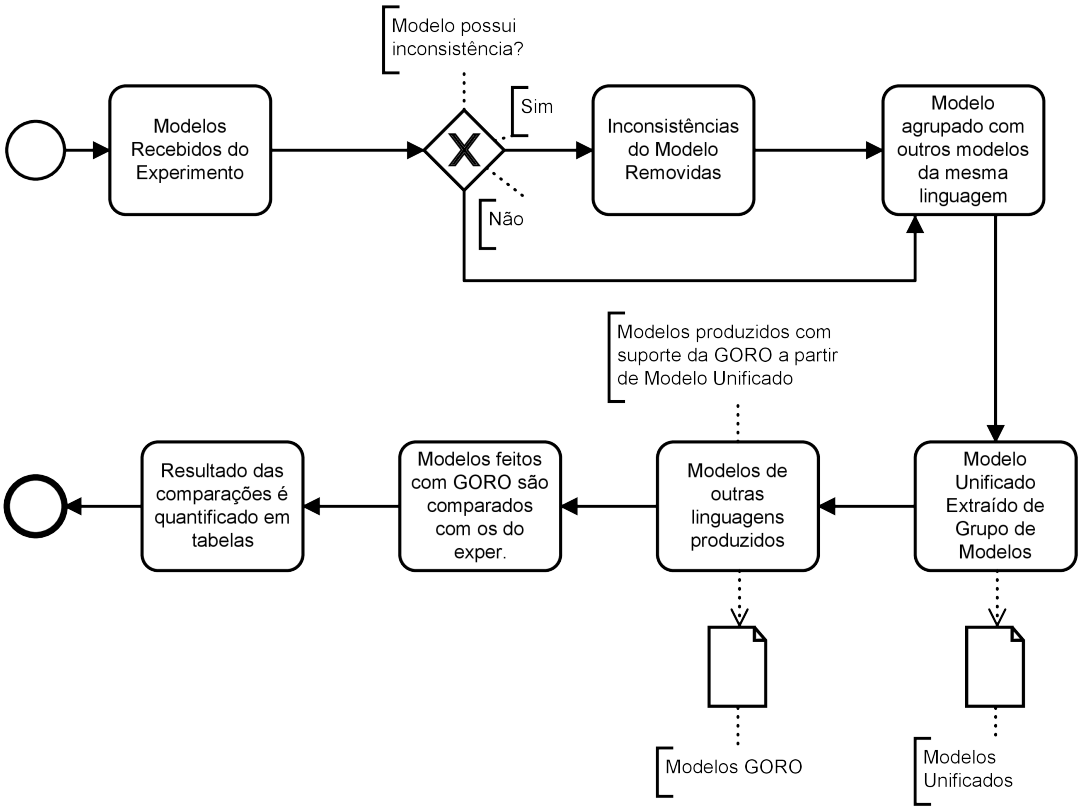


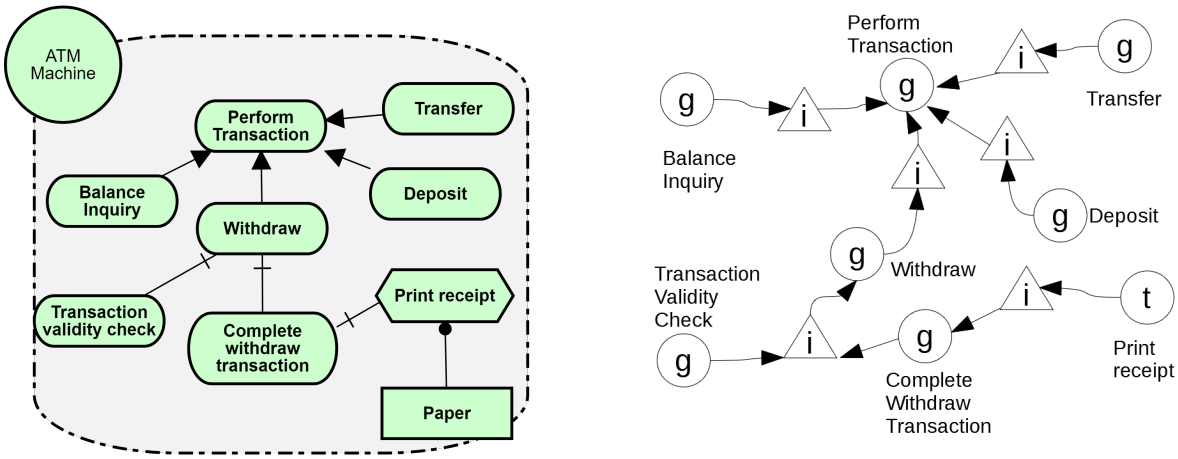
Figura 27 – Processo de extração de modelos para comparação.

A Tabela 6 mostra que GORO conseguiu cobrir pelo menos 77% do escopo dos modelos originais. É importante salientar que os resultados apresentados levam em consideração a semântica dos elementos. Em outras palavras, elementos foram analisados em relação ao contexto (ex.: considerando outros elementos conectados a eles). Assim, foi possível verificar elementos que, mesmo não tendo o texto exatamente igual, por exemplo, eram elementos equivalentes. Portanto, alguns aspectos do processo de transformação e análise dos modelos produzidos no contexto do experimento podem influenciar nos resultados. Estes aspectos são listados a seguir:

- iStar não modela conflitos (Conflicts) e obstáculos (Obstacles), elementos comuns a outras linguagens;
- KAOS, Techne e GSN descrevem nenhum tipo de relação de contribuição (Contribution) ou dependência (Dependency), muito comuns em *iStar*;
- Techne usa nós (Nodes) para representar a semântica de relações entre elementos. Esses nós não foram contabilizados como elementos durante a comparação com outras linguagens, já que, caso contrário, modelos Techne teriam um porcentagem maior de nós na comparação com outras linguagens. Essa porcentagem não representaria o número real de elementos em si;
- Techne e GSN não modelam nem usuários (User) nem Stakeholder, que são elementos fundamentais em modelos iStar e largamente utilizados em diagramas KAOS;
- Techne e GSN apresentam elementos muito particulares a suas sintaxes. Por exemplo, Techne tem os nós **Preference** e **Optionality**, enquanto GSN possui os elementos **Strategy** e **Justification**.

Para exemplificar, a Figura 28a mostra um fragmento do modelo iStar obtido a partir dos resultados do experimento (modelo unificado), enquanto a Figura 28b mostra o mesmo fragmento no modelo Techne obtido (com auxílio da GORO) a partir da conversão do modelo iStar da Figura 28a. O fragmento do modelo unificado de iStar (Fig. 28a) possui 10 elementos (1 Actor, 7 Goals, 1 Task, 1 Resource) e 8 relações, enquanto o modelo Techne equivalente (Fig. 28b) possui 8 elementos (7 Goals e 1 Task), já que Techne não possui os conceitos equivalentes a Actor e Resource. No modelo Techne, são contadas apenas as relações entre nós destino e nós de inferência (Inference), e os nós de inferência propriamente ditos não são contados. Com essa abordagem é possível normalizar o número de relações em Techne em comparação a outras linguagens. Ao final, é possível quantificar, no exemplo, que o modelo Techne possui 80% de elementos e 87,5% das relações em comparação ao modelo iStar.

Algumas ameaças à validade foram identificadas durante a elaboração deste experimento. Essas ameaças foram levantadas com base no trabalho de Wohlin et al. (2012), que



(a) Fragmento de modelo unificado de iStar.

(b) Fragmento de modelo Techne gerado a partir do modelo unificado de iStar

Figura 28 – Comparação entre iStar e Techne.

propõe uma lista de ameaças à validade para experimentos de Engenharia de Software. Essa lista é divida em quatro seções: Ameaças a Conclusões, Ameaças Internas, Ameaças à Construção e Ameaças Externas.

Ameaças a conclusões lidam com a relação entre o tratamento dos dados e conclusão sobre o resultado final. Neste experimento, se aplicam as seguintes ameaças:

- ***Fishing*:** ao se analisar modelos recebidos dos participantes, analistas poderiam procurar (pescar) por um resultado esperado. Por isso, nenhum modelo pode ser corrigido de modo a evitar influência do avaliador. Partes inconsistentes (erros no modelo de acordo com a definição da linguagem) foram apenas removidas.

- **Confiabilidade das medidas:** a análise do número de elementos de cada modelo, assim como a comparação entre modelos foram feitas manualmente. Assim, erros humanos podem influenciar os resultados das comparações. Para diminuir a probabilidade de erros, as contagens foram realizadas duas vezes (ou mais, caso os valores das duas contagens fossem discrepantes).

- **Confiabilidade da implementação do tratamento:** o processo de tratamento dos modelos recebidos (remoção de inconsistências) e unificação dos modelos pode sofrer influência do analista (e da experiência do mesmo). Buscou-se validar os resultados desse processo com o grupo de especialistas de domínio.

- **Heterogeneidade aleatória dos sujeitos:** esta ameaça lida com o fato de haver pessoas com diferentes níveis de experiência no grupo de participantes. Na verdade, a diferença no nível de familiaridade dos participantes com modelagem GORE foi um fator buscado durante a seleção dos participantes. Modelos com diferentes

complexidades poderiam garantir a cobertura de GORO em diversos níveis de detalhamento e especificidade.

Ameaças à validade interna são influências que podem afetar o resultado independente em relação à causalidade, sem o conhecimento do pesquisador:

- **Difusão ou imitação de tratamentos:** relacionadas ao comportamento dos participantes e do avaliador. Nesse caso, um participante poderia copiar o modelo de outro participante ou de outra fonte. Para reduzir as chances de plágio, participantes não foram informados sobre a identidade de outros participantes.

Ameaças à validade de construto dizem respeito à ameaça de generalização do resultado do experimento para o conceito ou teoria por trás do experimento:

- **Explicação pré-operacional inadequada dos construtos:** essa ameaça relaciona-se ao entendimento do processo do experimento por parte dos participantes. As instruções do experimento foram enviadas aos participantes por e-mail, processo que é passível de interpretações errôneas, visto que a linguagem natural pode ser ambígua. Portanto, participantes puderam tirar dúvidas sobre o processo do experimento na primeira semana do mesmo.

Ameaças à validade externa são condições que podem afetar os resultados do experimento. Essas ameaças são afetadas por três variáveis — pessoa, local e tempo:

- **Interação de seleção e tratamento:** esta ameaça relaciona-se aos participantes selecionados para o experimento. Nesse caso, um participante poderia ter grande experiência em uma determinada linguagem e pouca experiência em outra. Assim, esperou-se que o modelo genérico fornecido guiasse os participantes a produzirem modelos de qualidade similar. Além disso, procurou-se selecionar participantes com diferentes níveis de experiência em GORE.
- **Interação entre cenário e tratamento:** este relaciona-se às ferramentas utilizadas pelo processo. Durante a elaboração dos modelos, participantes ficaram livres para escolher a ferramenta de modelagem para criar os modelos. O uso de ferramentas diferentes pode ter um impacto na qualidade dos modelos (por exemplo, uma ferramenta pode ter correção automática de inconsistências enquanto a outra não tem). Por outro lado, forçar o participante a usar ferramentas específicas poderia poderia aumentar a chance de que o participante desistisse do experimento, não conseguisse criar o modelo ou criasse modelos inferiores por não se sentir confortável com determinado software. Assim, o experimento sugeriu ferramentas aos participantes, mas não obrigou o uso das mesmas.

- **Interação da história e tratamento:** esse é o efeito de que o experimento é realizado em um horário ou dia especial que afeta os resultados. Por exemplo, participantes tiveram duas semanas para criar os modelos solicitados. Caso os participantes estivessem sobrecarregados com outras atividades, isto poderia resultar em modelos de baixa qualidade. Como forma de mitigação, participantes puderam optar pela melhor data que gostariam de receber um experimento, dentro de um prazo adequado.

O objetivo de análise das ameças de um experimento é fazer dele o mais realista possível, trazendo resultados coesos. Buscou-se modelar o processo de experimentação e análise dos resultados de forma a reduzir os riscos apresentados. Todo o material produzido está disponível no site do projeto ROSE (<http://nemo.inf.ufes.br/projects/rose/>).

4.3.2 Ferramenta de Conversão

Como prova de conceito, foi desenvolvida a *GORO Conversion Tool* (GCT). Uma ferramenta de conversão de modelos baseada em Java™, que converte modelos iStar produzidos na ferramenta *piStar*[2] para modelos KAOS que podem ser carregados na ferramenta *Objectiver*.[3]

A GCT utiliza a GORO para identificar a relação entre os elementos das linguagens a serem convertidas, por exemplo, os elementos iStar::Task e KAOS::Operation são mapeados para GORO::Task. Portanto, ao realizar uma transformação de iStar para KAOS, é identificada a relação entre iStar::Task e KAOS::Operation. Assim, a GCT realiza a transformação da seguinte forma: (i) os elementos da linguagem de entrada são carregados e mapeados para seus correlatos em GORO; (ii) os elementos identificados em (i) são mapeados de GORO para seus correspondentes na linguagem destino; (iii) o modelo destino é gerado.

Conforme mencionado, a GCT utiliza uma versão operacional de GORO, implementada em OWL (*Web Ontology Language*). Durante o processo de conversão, a ferramenta também cria um arquivo OWL com os elementos do modelo de entrada instanciados. O arquivo OWL gerado pode ser carregado em ferramentas de edição de ontologias como o *Protègè*.[4] A linguagem OWL permite que versões operacionais de uma ontologia sejam implementadas sem perda expressiva de semântica. Assim, conceitos como *tipo* e *subtipo*, *relações parte-todo* e *restrições de multiplicidade* são preservadas e podem ser utilizadas durante o processamento de informações (MCGUINNESS; HARMELEN et al., 2004). A Figura 29 traz um exemplo de visualização do arquivo OWL criado pela ferramenta, nele podem ser vistas as hierarquias, restrições e instanciações da ontologia.

[2] <https://www.cin.ufpe.br/~jhcp/pistar/>
[3] <http://www.objectiver.com/>
[4] <https://protege.stanford.edu/>

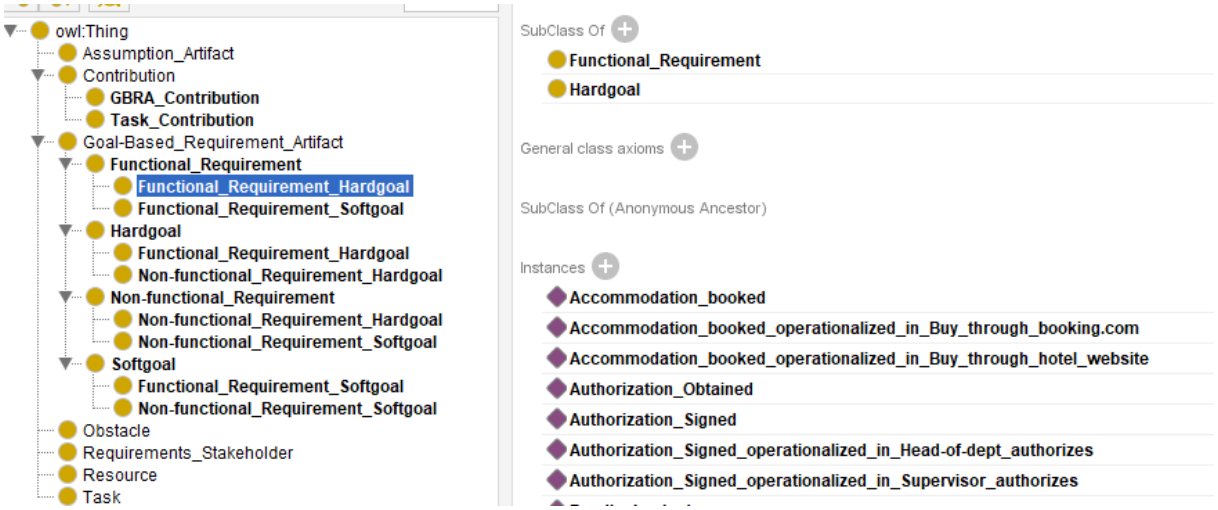


Figura 29 – Exemplo da versão operacional de GORO no Protègè.

É importante mencionar que a GCT é um protótipo, criada para validar a interoperabilidade de GORO. Ao utilizar a ontologia como ponto central no processo de conversão, diminui-se significativamente o trabalho necessário para se adicionar novas linguagens, já que só é necessário implementar a conversão da nova linguagem para a ontologia e vice-versa.

No caso em que elementos do modelo de origem não estão presentes na linguagem de destino (por exemplo, contribuições estão presentes em *iStar*, mas não em KAOS), eles são listados no *log* da ferramenta e não aparecem no modelo de saída. Além disso, são tomadas algumas decisões de *design* para ajustar o modelo *iStar* às restrições dos modelos KAOS. Por exemplo, KAOS permite apenas que Operations sejam associadas a Requirements. Portanto, quando um relacionamento de refinamento entre um objetivo e uma tarefa é identificado no modelo de origem, um relacionamento Goal-Requirement-Operation é criado no modelo KAOS, conforme exemplificado na Figura 30.

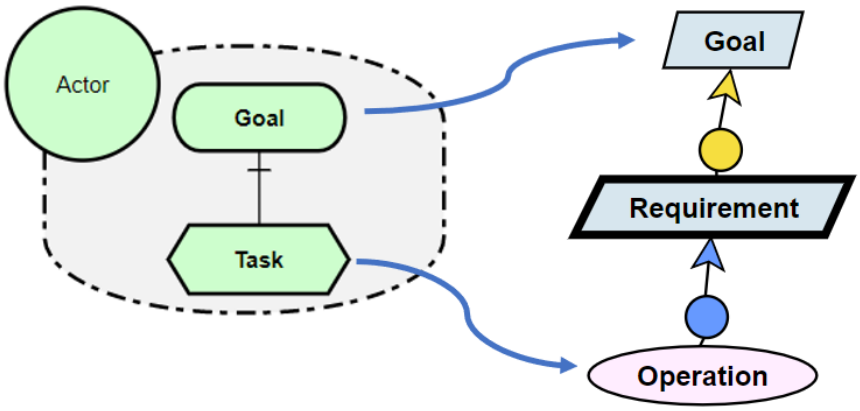


Figura 30 – Exemplo de decisão de *design* da GCT.

A GCT é uma ferramenta de código livre. Tanto o código fonte como a documentação da ferramenta estão disponíveis em <https://github.com/nemo-ufes/gorotool>.

4.4 Discussão

Esta seção descreve alguns aspectos observados durante o processo de validação discutidos na Seção 4.2. De acordo com Kelly e Tolvanen (2008), uma linguagem de modelagem bem formada oferece algumas vantagens, como prevenção precoce a erros, verificação de integridade e consistência das especificações. Além disso, Gonçalves et al. (2018b) afirmam que linguagens com conceitos pouco claros "podem dificultar a compreensão por parte de novatos que desejam usá-la". Argumentamos que ontologias podem apoiar a construção/evolução de linguagens de modelagem, servindo como base bem fundamentada e consensual para conceitos em um domínio específico.

Em relação ao mapeamento entre os elementos das linguagens e GORO, conseguimos encontrar uma correspondência entre as linguagens mapeadas e os elementos de GORO na maioria dos casos. No entanto, algumas linguagens, como Techne e GSN, possuem elementos que não foram incluídos nesse mapeamento por serem considerados não-GORE. Em outras palavras, eles não eram considerados elementos comuns no contexto do GORE, mas um construto extra fornecido especificamente pela linguagem de modelagem. O restante deste capítulo discute aspectos de cada linguagem e como eles foram mapeados para a ontologia. Usa-se sansserif e *itálico* para representar conceitos e relações de GORO, respectivamente, e **negrito** para os construtos de linguagens GORE. Elementos de linguagem não mencionados no texto abaixo foram mapeados diretamente para construtos de mesmo nome em GORO.

NFR Framework. Dado o fato de que o NFR Framework é focado em qualidades, ambos **Claim Softgoal** e **NFR Softgoal** foram mapeados para Non-functional Requirement & Softgoal. Um **Operationalizing Softgoal** é definido como um dado ou uma operação no sistema destino (MYLOPOULOS; CHUNG; NIXON, 1992) e, portanto, este elemento foi mapeado para Task em GORO. É possível perceber que esse *framework* por si só não consegue cobrir um processo de elicitação de requisitos por inteiro, dado que seu foco está somente na análise de qualidades. Ainda assim, é muito difícil fazer análise de requisitos não-funcionais/*softgoals* sem considerar suas relações com requisitos funcionais/*hardgoals*.

KAOS. Este *framework* não faz uma separação explícita em requisitos hard/soft e, portanto, todos os elementos relacionados a objetivos (**Goal**, **Expectation** e **Requirement**) foram mapeados tanto para Functional Requirement & Hardgoal quanto para Non-functional Requirement & Softgoal, também dependendo do contexto modelado. Ou seja, dependendo de sua especificação e dos elementos conectados. Por exemplo, o **Goal** *Comunicação Segura com o Banco*, seria mapeado para Non-functional Requirement &

Softgoal, enquanto o **Goal** *Prover funcionalidade de saque* seria mapeado para Functional Requirement & Hardgoal. Da mesma forma, **Agent** foi mapeado tanto para Requirements Stakeholder, User ou Software Actor em GORO, dado que este elemento pode assumir qualquer uma dessas classificações, dependendo do contexto. Por fim, apesar do elemento **Softgoal** não aparecer nos artigos que propõe KAOS (DARDENNE; LAMSWEERDE; FICKAS, 1993; RESPECT-IT, 2007), este elemento aparece na ferramenta de modelagem oficial da linguagem[5] e em alguns modelos da literatura (WERNECK; OLIVEIRA; LEITE, 2009; AHMAD; BELLOIR; BRUEL, 2015). Assim, **Softgoal** foi mapeado para Non-functional Requirement & Softgoal em GORO. Isso pode causar problemas relacionados a ambiguidade e sobrecarga de construto e de conceito, visto que o elemento **Goal** representa mais do que um conceito (a depender do contexto do modelo), e que o conceito de Non-functional Requirement & Softgoal é representado por **Goal** em alguns modelos, e por **Softgoal** em outros.

i*. Assim como KAOS, o construto **Agent** de i^* é mapeado para Requirements Stakeholder, User ou Software Actor, dependendo do contexto. Por outro lado, i^* faz uma separação explícita entre hard/softgoals. Assim, **Goal** foi mapeado para Functional Requirement & Hardgoal, enquanto **Softgoal** foi mapeado para Non-functional Requirement & Softgoal. O **Means-end link** conecta **Tasks** (o meio — *the means*) a **Goals** (o fim — *the end*) e, portanto, foi mapeado para a relação *intends to operationalize* de GORO. A relação **decomposition link** é mapeada de acordo com os elementos conectados por ela: se os elementos forem **Goals** ou **Softgoals**, é mapeada para Complex GBR composition; no entanto, se os elementos forem **Tasks**, será uma Complex Task composition. Os conceitos relacionados a **dependency links** não foram mapeados porque não são modelados em GORO, pois os relacionamentos de dependência, embora muito úteis, são conceitos exclusivos de i^*/iStar e seus dialetos. Essa exclusividade fere o critério de inclusão de elementos na ontologia, conforme explicado na Seção 3.1. Existem diversos trabalhos na literatura (GUIZZARDI et al., 2013b; GUIZZARDI; FRANCH; GUIZZARDI, 2012; GUTIÉRREZ et al., 2011) que utilizam UFO para tentar criar fronteiras mais claras entre as definições do **Means-end link** e de **Make Contribution links** quando estes conectam tarefas a objetivos, já que em ambos os casos entende-se que uma tarefa está realizando um objetivo, causando problemas relacionados a ambiguidade e sobrecarga de conceito.

GBRAM. Assim como GORO, GBRAM faz uma diferenciação clara entre o stakeholder que teve seus requisitos diretamente elicitados e o stakeholder que comporta-se apenas como usuário do sistema. Portanto, **Agent** é mapeado para User e **Stakeholder** para Requirements Stakeholder. Nos artigos que propõe a linguagem e na literatura disponível, não foi possível encontrar alguma menção para agentes de software. O método

[5] <http://www.objectiver.com/>

GBRAM também define diferentes conceitos de objetivos, separando os objetivos que são alcançados em um determinado momento (**Achievement Goals**) daqueles que são mantidos verdadeiros em um determinado intervalo de tempo (**Maintenance Goals**). Assim, todos os elementos relacionados a objetivos foram classificados em Functional Requirement & Hardgoal ou Non-functional Requirement & Softgoal, novamente, dependendo do domínio modelado. Aqui novamente é possível identificar problemas relacionados a ambiguidade e sobrecarga de conceito, já que os elementos **Achievement Goals** e **Maintenance Goals** representam mais de um conceito.

GSN. Assim como em KAOS, o conceito de **Goal** em GSN pode ser mapeado tanto para Functional Requirement & Hardgoal como para Non-functional Requirement & Softgoal. A linguagem possui os conceitos de **Context**, **Assumption**, **Justification**, e **Strategy**. Assim, **Context**, dada sua definição, pode ser mapeado como uma Assumption ou Resource, dependendo do contexto. **Assumption** é mapeado para o elemento de mesmo nome em GORO. Essa sobreposição de conceitos pode indicar uma sobrecarga de construto, que pode resultar em modelos confusos ou ambíguos. **Justification** e **Strategy**, usados para explicar por que um objetivo existe e por que foi refinado de alguma maneira específica, respectivamente, não são mapeados. Da mesma forma que o mapeamento de **decomposition links** de *i**, a relação **Supported by** de GSN é mapeada dependendo do tipo de elemento que ela conecta, ou seja, ao ligar dois **Goals**, é um Complex GBR, e ao conectar um **Goal** com uma **Solution**, é mapeada para *intends to operationalize*.

Tropos e GRL. Por serem dialetos de *i**, Tropos e GRL tiveram a maioria de seus elementos mapeados da mesma maneira que os elementos de sua língua mãe. Os construtos extras dessas duas linguagens, ou seja, os contrutos que não entram no critério de inclusão de conceitos da GORO, acabaram não sendo mapeados para GORO. Portanto, o elemento **Capability** de Tropos, bem como o relacionamento **correlation** de GRL, não foram considerados. Por herdarem quase todos os conceitos de *i**, estas linguagens acabam também herdando seus problemas, como por exemplo problemas de ambiguidade identificados entre as relações de **Means-end** e **Contribution**.

Techne. Techne é uma linguagem que foi construída com base em DOLCE, uma ontologia de fundamentação. A maioria dos conceitos de Techne puderam ser diretamente mapeados em GORO. Entretanto, é necessário analisar o elemento **Inference** dessa linguagem, definido como um elemento que relaciona "um conjunto de instâncias de qualquer conceito ou relação com outra instância de um conceito ou relacionamento, para transmitir que a satisfação da conjunção do primeiro satisfaz o segundo" (BORGIDA et al., 2010), ficando claro que esse elemento só é um refinamento do tipo AND, sendo mapeado, portanto, para And Complex GBR. O refinamento do tipo OR (Or Complex GBR) em Techne é obtido ao se combinar os elementos **Inference** e **Optionality**.

iStar. Em sua segunda versão, iStar teve algumas modificações realizadas para

aumentar a simplicidade da linguagem. O construto **Softgoal** foi renomeado para *Quality*, enquanto a relação **Means-end link** foi agrupada com a **Decomposition Link**, formando a **Refinement Link**. Apesar desse agrupamento ter facilitado a compreensão da nova versão da linguagem, ele acabou sendo sobrecarregado, já que passou a representar decomposição entre objetivos (Complex GBR), operacionalização entre tarefas e objetivos (*intends to operationalize*) e decomposição de tarefas (Complex Task).

O **Refinement Link** também permite as relações entre tarefas alternativas (decomposição do tipo **OR** para tarefas) e decomposição de tarefas em objetivos, que não são aceitas pela GORO. Argumenta-se que tarefas refinadas utilizando a relação **OR** representam uma maneira específica de executar uma tarefa mãe, enquanto a relação **AND** representa uma sequência de passos para serem executados e, assim, atender a tarefa mãe, trazendo tipos de interpretação diferentes para o conceito de refinamento. Por fim, em relação ao refinamento de tarefas em objetivos, uma possível interpretação seria de que esses objetivos são identificados após a execução de uma determinada tarefa. Argumenta-se que objetivos surgidos após a execução de uma tarefa possuem um nível de especifidade maior do que os objetivos que justificam a necessidade daquela tarefa (os pais desta tarefa) e, portanto, os objetivos filhos desta seriam mais bem acomodados em modelos diferentes (adequados ao seu grau de especificidade). Assim, destaca-se que o processo de simplificação de *i**/iStar, apesar de buscar ajudar usuários (principalmente novos usuários) no processo de aprendizagem da mesma, pode causar ainda mais confusão. Este é um exemplo claro de como a análise ontológica pode ser útil, pois provê clara fundamentação ao processo de reformulação e simplificação de linguagens, sem prejudicar a semântica dos seus construtos. No caso de iStar, mesmo sem propor modificações na versão atual, é possível utilizar ontologias como a GORO para explicar conceitos, dando assim suporte a usuários iniciantes e experientes.

Neste processo, foi possível perceber alguns problemas semânticos em relação aos construtos de algumas linguagens. Por fim, as observações descritas neste capítulo também servem de base para trabalhos futuros, nos quais deseja-se analisar com mais rigor e critério cada uma das linguagens descritas ao longo desta dissertação. No próximo capítulo são apresentadas as conclusões do trabalho, incluindo as contribuições, limitações e trabalhos futuros.

5 Considerações Finais

A Engenharia de Requisitos Orientada a Objetivos consolida-se cada vez mais como uma área de grande importância dentro da Engenharia de Requisitos. Nos últimos 25 anos, diversas abordagens GORE foram propostas. Grande parte dessas abordagens, entretanto, foram definidas sem uma conceitualização bem fundamentada de domínio, oferecendo na verdade definições fracas e passíveis de múltiplas interpretações de seus construtos. Assim, surge a GORO, com a proposta de servir como base para conceitualização compartilhada e bem fundamentada no domínio de GORE.

Construída utilizando o método SABiO para Engenharia de Ontologias, GORO é fundamentada em UFO e faz reúso de cinco outras ontologias (COVR, NFRO, SPO, RSRO e ASMP). A GORO foi construída com base em conhecimento extraído de nove linguagens GORE diferentes (i*, iStar, KAOS, GBRAM, GSN, Techne, NFR Framework, Tropos e GRL) e com a ajuda de especialistas de domínio.

Parte integrante da SEON, GORO foi submetida a processos de verificação, onde foram verificadas as questões de competência; validação, onde foram mapeados os conceitos das diferentes linguagens analisadas para a ontologia; prova de conceito, onde a ontologia foi utilizada para apoiar a construção de uma ferramenta de transformação de modelos; e avaliada em um experimento de modelagem. Assim, conclui-se que a ontologia apresenta eficácia tanto para uso automatizado, pois foi utilizada em experimentos de mapeamento, como para comunicação humana, já que é capaz de, por exemplo, responder todas as suas questões de competência, além de ter sido verificada por especialistas de domínio.

No texto que segue, a Seção 5.1 lista as contribuições e artigos resultantes desta pesquisa, a Seção 5.2 apresenta as limitações deste trabalho, e a Seção 5.3 conclui esta dissertação apresentando trabalhos futuros que podem ser realizados a partir dos resultados alcançados nessa pesquisa.

5.1 Contribuições

Este trabalho apresenta as seguintes contribuições:

- Uma ontologia de referência para a área de Engenharia de Requisitos Orientada a Objetivos;
- Uma referência para construção e análise de linguagens GORE;
- Um artefato computacional que pode ser utilizado para interoperabilidade de modelos GORE construídos em linguagens diferentes;

- Uma ferramenta de transformação de modelos iStar para KAOS;
- Um conjunto de modelos em linguagens diferentes, de um mesmo exemplo, que podem ser utilizados para fins didáticos e de pesquisa.

Esta pesquisa resultou ainda na publicação de dois artigos (BERNABÉ et al., 2019a; BERNABÉ et al., 2019b).

5.2 Dificuldades e Limitações

Dentre as maiores dificuldades deste trabalho, destaca-se primeiramente a curva de aprendizado necessária para o entendimento do domínio GORE. Conforme já discutido, muitas linguagens apresentam conceitos mal definidos e confusos, dificultando o entendimento. Apesar disso, a dificuldade em realizar este trabalho é, ao mesmo tempo, uma de suas motivações.

Mapear os conceitos das linguagens para a ontologia também foi um ponto crítico pois alguns conceitos, por serem sobrecarregados, acabam sendo mapeados em uma interseção de elementos da GORO, dependendo de como são representados no modelo, dificultando, por exemplo, a implementação de transformações automáticas.

Uma das limitações deste trabalho está no fato de a ontologia cobrir apenas conceitos tradicionais de GORE, não considerando conceitos que começam a aparecer e despontar com diversas vantagens dentro deste campo, como por exemplo objetivos probabilísticos (GUIZZARDI et al., 2014). Dessa forma, algumas das limitações deste trabalho apresentam-se como perspectivas de trabalhos futuros, discutidos na seção a seguir. Por fim, outra limitação está relacionada ao pequeno número de integrantes do grupo de especialistas de domínio (apenas cinco pessoas) que apoiaram este trabalho. Pretende-se aumentar esse número no futuro.

5.3 Trabalhos Futuros

O campo da Engenharia de Requisitos Orientada a Objetivos está em evolução e, portanto, é necessário que a ontologia também evolua para acompanhar seu domínio. A seguir, são listados trabalhos futuros pretendidos para a GORO:

- Extensão da ferramenta de conversão (GCT) para que ela suporte mais linguagens de entrada e saída;
- A realização de novos experimentos com cenários diferentes;
- Uso da GORO para análise semântica de linguagens GORE, com o objetivo de propor

melhoria nessas linguagens e, se necessário, a proposta de padrões de modelagem para garantir consistência em modelos;

- Uso da GORO para propor uma linguagem de objetivos bem fundamentada ontologicamente, com o objetivo de reduzir problemas relacionados a ambiguidade de modelos, sobrecarga de construtos e/ou conceitos, etc;
- Análise estatística do experimento de modelagem executando, considerando aspectos como precisão e revocação (*recall*) (EUZENAT, 2007);
- Análise dos modelos dos participantes do experimento para identificar variações, vícios e padrões de modelagem.

Referências

AHMAD, M.; BELLOIR, N.; BRUEL, J.-M. Modeling and verification of functional and non-functional requirements of ambient self-adaptive systems. *Journal of Systems and Software*, Elsevier, v. 107, p. 50–70, 2015. Citado na página 67.

ALMENDRA, C. et al. istar2. 0-owl: an operational ontology for istar. 2019. Citado na página 43.

AMYOT, D. et al. Evaluating goal models within the goal-oriented requirement language. *International Journal of Intelligent Systems*, v. 25, n. 8, p. 841–877, 2010. Citado 2 vezes nas páginas 20 e 27.

ANTON, A. Goal-based requirements analysis. In: *Proc. of the 2nd Intl. Conference on Requirements Engineering (RE)*. [S.l.]: IEEE Comput. Soc. Press, 1996. p. 136–144. Citado 2 vezes nas páginas 20 e 24.

BERNABÉ, C. H. et al. Using goro to provide ontological interpretations of istar constructs. 2019. Citado 2 vezes nas páginas 17 e 71.

BERNABÉ, C. H. et al. Goro 2.0: Evolving an ontology for goal-oriented requirements engineering. In: SPRINGER. *International Conference on Conceptual Modeling*. [S.l.], 2019. p. 169–179. Citado 2 vezes nas páginas 17 e 71.

BORGIDA, A. et al. *Techne : A(nother) Requirements Modeling Language*. [S.l.], 2010. Disponível em: <ftp://www.cs.toronto.edu/dist/reports/csri/593/techne-techrep-v1.pdf>. Citado 3 vezes nas páginas 20, 28 e 68.

BRESCIANI, P. et al. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, v. 8, n. 3, p. 203–236, may 2004. Citado 2 vezes nas páginas 20 e 27.

CARES, C.; FRANCH, X. A Metamodelling Approach for I* Model Translations. In: *Proceedings of the 23rd International Conference on Advanced Information Systems Engineering*. [S.l.]: Springer, 2011. p. 337–351. Citado na página 44.

DALPIAZ, F.; FRANCH, X.; HORKOFF, J. iStar 2.0 Language Guide. *CoRR*, abs/1605.07767, 2016. Citado 3 vezes nas páginas 20, 29 e 52.

DARDENNE, A.; LAMSWEERDE, A. van; FICKAS, S. Goal-directed requirements acquisition. *Science of Computer Programming*, v. 20, n. 1-2, p. 3–50, apr 1993. Citado 4 vezes nas páginas 14, 20, 22 e 67.

DUARTE, B. B. et al. Ontological foundations for software requirements with a focus on requirements at runtime. *Applied Ontology*, IOS Press, v. 13, n. 2, p. 73–105, 2018. Citado 4 vezes nas páginas 9, 36, 40 e 42.

EUZENAT, J. Semantic precision and recall for ontology alignment evaluation. In: *IJCAI*. [S.l.: s.n.], 2007. v. 7, p. 348–353. Citado na página 72.

FALBO, R. A. SABiO: Systematic approach for building ontologies. In: *Proc. of the 1st Joint Ws. on Ontologies in Conceptual Modeling and Inf. Systems Engineering*. [S.l.]: CEUR, 2014. v. 1201. Citado 2 vezes nas páginas 16 e 41.

FAYOUMI, A.; KAVAKLI, E.; LOUCOPOULOS, P. Towards a Unified Meta-Model for Goal Oriented Modelling. In: *Proc. of the 12th European, Mediterranean & Middle Eastern Conf. on Information Systems (EMCIS)*. [S.l.: s.n.], 2015. p. 1–10. Citado na página 44.

GONÇALVES, E. et al. A Systematic Literature Review of iStar extensions. *Journal of Systems and Software*, 2018. ISSN 01641212. Citado na página 27.

GONÇALVES, E. et al. Understanding what is important in iStar extension proposals: the viewpoint of researchers. *Requirements Engineering.*, p. 1–28, 2018. ISSN 1432-010X. Citado na página 66.

GOODMAN, N. *Languages of art: An approach to a theory of symbols*. [S.l.]: Hackett publishing, 1976. Citado na página 15.

GUARINO, N. *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*. [S.l.]: IOS press, 1998. v. 46. Citado na página 30.

GUARINO, N.; OBERLE, D.; STAAB, S. What is an Ontology? In: STAAB, S.; STUDER, R. (Ed.). *Handbook on Ontologies*. 2. ed. [S.l.]: Springer, 2009, (International Handbooks on Information Systems). p. 1–17. Citado na página 15.

GUARINO, N.; WELTY, C. Evaluating ontological decisions with ontoclean. Citeseer, 2002. Citado na página 30.

GUIZZARDI, G. *Ontological foundations for structural conceptual models*. Tese (Doutorado) — University of Twente, 2005. Citado 2 vezes nas páginas 17 e 30.

GUIZZARDI, G. On Ontology, ontologies, Conceptualizations, Modeling Languages, and (Meta)Models. In: *Proc. of the 2007 Conference on Databases and Information Systems*. [S.l.]: IOS Press, 2007. Citado na página 15.

GUIZZARDI, G. et al. Towards ontological foundations for conceptual modeling: The unified foundational ontology (ufo) story. *Applied ontology*, IOS Press, v. 10, p. 259–271, 2015. Citado na página 31.

GUIZZARDI, R.; FRANCH, X.; GUIZZARDI, G. Applying a foundational ontology to analyze means-end links in the i* framework. In: *Proc. of the 6th Intl. Conf. on Research Challenges in Information Science (RCIS)*. [S.l.]: IEEE, 2012. p. 1–11. Citado na página 67.

GUIZZARDI, R. et al. Using a foundational ontology to investigate the semantics behind the concepts of the i* language. In: *Proc. of the 6th Intl. i* Workshop (iStar)*. [S.l.]: CEUR, 2013. v. 978, p. 13–18. Citado 2 vezes nas páginas 43 e 44.

GUIZZARDI, R. S. et al. Ontological distinctions between means-end and contribution links in the i* framework. In: SPRINGER. *International Conference on Conceptual Modeling*. [S.l.], 2013. p. 463–470. Citado na página 67.

GUIZZARDI, R. S. et al. An ontological interpretation of non-functional requirements. In: *Proc. of the 8th Intl. Conf. on Formal Ontology in Inf. Systems (FOIS)*. [S.l.: s.n.], 2014. p. 344–357. Citado 5 vezes nas páginas 9, 35, 38, 49 e 71.

GUTIÉRREZ, J. F. et al. Ontological analysis of means-end links. In: CEUR WORKSHOP PROCEEDINGS. *iStar 2011: proceedings of the 5th International i* workshop: 29-30th August, 2011, Trento, Italy*. [S.l.], 2011. p. 37–42. Citado na página 67.

HELLER, B.; HERRE, H. Ontological categories in gol. *Axiomathes*, Springer, v. 14, n. 1, p. 57–76, 2004. Citado na página 30.

HORKOFF, J. et al. Goal-oriented requirements engineering: an extended systematic mapping study. *Requirements Engineering*, Springer, v. 24, n. 2, p. 133–160, 2016. Citado 4 vezes nas páginas 14, 21, 43 e 58.

JURETA, I. J.; MYLOPOULOS, J.; FAULKNER, S. A core ontology for requirements. *Applied Ontology*, v. 4, n. 3-4, p. 169–244, 2009. Citado na página 43.

KELLY, S.; TOLVANEN, J.-P. *Domain-specific modeling: enabling full code generation*. [S.l.]: John Wiley & Sons, 2008. Citado na página 66.

KELLY, T.; WEAVER, R. The goal structuring notation—a safety argument notation. In: *Proc. of Dependable Systems and Networks 2004 Ws on Assurance Cases*. [S.l.: s.n.], 2004. Citado 3 vezes nas páginas 20, 26 e 59.

LAMSWEERDE, A. van. Goal-oriented requirements engineering: a guided tour. In: *Proc. of the 5th IEEE Intl. Symposium on Requirements Engineering*. [S.l.]: IEEE Comput. Soc, 2001. p. 249–262. Citado 4 vezes nas páginas 14, 19, 20 e 21.

LAMSWEERDE, A. van; LETIER, E. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, v. 26, n. 10, p. 978–1005, 2000. Citado na página 50.

LAPOUCHNIAN, A. Goal-Oriented Requirements Engineering : An Overview of the Current Research. *Requirements Engineering*, IEEE, v. 8, p. 32, 2005. Citado na página 19.

LUCENA, M. et al. Towards a Unified Metamodel for i*. In: *2008 Second International Conference on Research Challenges in Information Science*. [S.l.: s.n.], 2008. p. 237–246. Citado na página 44.

MASOLO, C. et al. *WonderWeb Deliverable D17 - The WonderWeb Library of Foundational Ontologies Preliminary Report*. [S.l.], 2003. Disponível em: <http://www.loa.istc.cnr.it/old/Papers/DOLCE2.1-FOL.pdf>. Citado na página 28.

MCGUINNESS, D. L.; HARMELEN, F. V. et al. Owl web ontology language overview. *W3C recommendation*, v. 10, n. 10, p. 2004, 2004. Citado na página 64.

MOODY, D. The physics of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, v. 35, p. 756–779, 2009. Citado na página 15.

MOURATIDIS, H.; GIORGINI, P. Secure tropos: a security-oriented extension of the

tropos methodology. *International Journal of Software Engineering and Knowledge Engineering*, World Scientific, v. 17, n. 02, p. 285–309, 2007. Citado na página 27.

MYLOPOULOS, J.; CHUNG, L.; NIXON, B. Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. *IEEE Transactions on Software Engineering*, v. 18, n. 6, p. 483–497, jun 1992. Citado 4 vezes nas páginas 20, 21, 52 e 66.

NASA, A.; SPACE, J. Error Cost Escalation Through the Project Life Cycle. 2018. Disponível em: <https://ntrs.nasa.gov/search.jsp?R=20100036670>. Citado na página 14.

NEGRI, P. et al. Towards an ontology of goal-oriented requirements. In: *Proc. of the 20th Ibero-American Conf. on Software Engineering (CIbSE)*. [S.l.: s.n.], 2017. Citado 2 vezes nas páginas 43 e 44.

NEGRI, P. P. *GORO - Uma ontologia sobre requisitos baseados em objetivos*. 83 p. Tese (Doutorado) — Universidade Federal do Espírito Santo, 2017. Citado 7 vezes nas páginas 9, 31, 32, 33, 34, 35 e 36.

PATRICIO, P. et al. Towards a Unified Goal-Oriented Language. In: *Proc. of the 35th Annual Computer Software and Applications Conf.* [S.l.]: IEEE, 2011. p. 596–601. Citado na página 44.

REGINATO, C. et al. Go-for: A goal-oriented framework for ontology reuse. In: IEEE. *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*. [S.l.], 2019. p. 99–106. Citado na página 35.

RESPECT-IT. *A KAOS Tutorial*. [S.l.], 2007. Disponível em: <http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf>. Citado 3 vezes nas páginas 20, 23 e 67.

RUY, F. B. et al. SEON: A Software Engineering Ontology Network. In: *Knowledge Engineering and Knowledge Management. LNCS*. [S.l.]: Springer, 2016. v. 10024, p. 527–542. Citado 6 vezes nas páginas 9, 16, 35, 36, 40 e 41.

SALES, T. P. et al. The Common Ontology of Value and Risk. In: *Proc. of the 37th International Conference on Conceptual Modeling (ER). LNCS*. [S.l.: s.n.], 2018. v. 11157, p. 121–135. Citado 4 vezes nas páginas 9, 20, 35 e 39.

SIENA, A. et al. Capturing variability of law with nómos 2. In: SPRINGER. *International Conference on Conceptual Modeling*. [S.l.], 2012. p. 383–396. Citado na página 27.

WANG, X. et al. How software changes the world: The role of assumptions. In: *Proc. of the 10th Intl. Conf. on Research Challenges in Information Science (RCIS)*. [S.l.]: IEEE, 2016. p. 1–12. Citado na página 35.

WANG, Y. et al. An automated approach to monitoring and diagnosing requirements. In: *ASE '07: 22nd IEEE/ACM international conference on Automated software engineering*. Atlanta, USA: ACM Press, 2007. p. 293–302. Citado na página 59.

WERNECK, V. M. B.; OLIVEIRA, A. d. P. A.; LEITE, J. C. S. do P. Comparing gore frameworks: i-star and kaos. In: *WER*. [S.l.: s.n.], 2009. Citado na página 67.

WOHLIN, C. et al. *Experimentation in Software Engineering*. [S.l.]: Springer, 2012. Citado na página 61.

YU, E. S. K. *Modelling strategic relationships for process reengineering*. Tese (Doutorado) — PhD thesis, University of Toronto, 1996. Citado 5 vezes nas páginas 14, 20, 23, 24 e 52.