Bruno Borlini Duarte

RRO - Uma ontologia sobre o uso de Requisitos de Software em Tempo de Execução

Vitória, ES

2016

Bruno Borlini Duarte

RRO - Uma ontologia sobre o uso de Requisitos de Software em Tempo de Execução

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. Silva Souza

Coorientador: Prof. Dr. André Luiz de Castro Leal

Vitória, ES

2016

Bruno Borlini Duarte
RRO - Uma ontologia sobre o uso de Requisitos de Software em Tempo de Execução/ Bruno Borlini Duarte. – Vitória, ES, 2016-
97 p. : il.; 30 cm.

Orientador: Prof. Dr. Vítor E. Silva Souza

Dissertação de Mestrado – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Programa de Pós-Graduação em Informática, 2016.

1. Requisitos em tempo de execução. 2. Ontologias. I. Souza, Vítor Estêvão Silva. II. Universidade Federal do Espírito Santo. IV. RRO - Uma ontologia sobre o uso de Requisitos de Software em Tempo de Execução

CDU 02:141:005.7

Bruno Borlini Duarte

RRO - Uma ontologia sobre o uso de Requisitos de Software em Tempo de Execução

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, 04 de agosto de 2016:

Prof. Dr. Vítor E. Silva Souza
Orientador

André Luiz de Castro Leal, D.Sc.
Universidade Federal Rural do Rio de Janeiro
Co-Orientador

Renata Silva Souza Guizzardi, Ph.D.
Universidade Federal do Espírito Santo

Fernanda Araújo Baião, D.Sc.
Universidade Federal do Estado do Rio de Janeiro

Vitória, ES
2016

A minha Mãe, que sempre viveu por mim, me dando todo suporte possível. Devo essa e qualquer outra conquista que eu venha a alcançar nessa vida a ela.

Agradecimentos

Agradeço primeiramente a minha mãe Celina e a minha tia Lourdes por todo amor incondicional e por todo suporte que me deram durante toda minha vida. Sem as duas eu simplesmente não estaria aqui hoje.

Agradeço a minha irmã Karina e a minha namorada Aline por sempre estarem ao meu lado me dando apoio e me ajudando.

Agradeço a meu Orientador Vitor e a meu Coorientador por todo conhecimento que me proporcionaram durante essa fase da minha vida.

Agradeço aos professores do mestrado por tudo que me ensinaram nesses dois anos.

Agradeço a minha amiga e colega de mestrado Beatriz por sempre ter me ajudado e tirado minhas dúvidas quando precisei. Eu não teria conseguido sem ela.

Resumo

Nos últimos anos, temos assistido um constante aumento de interesse em sistemas de software que são capazes de monitorar seu meio de execução e, se necessário, adaptar seus requisitos para continuar cumprindo seu propósito. Esse tipo de software normalmente consiste em um software base, comumente chamado na literatura de *target system*, que é responsável por executar a função principal para qual foi desenvolvido; além de um software, interno ou externo, responsável por monitorar o software base, realizar uma análise dos dados coletados para verificar se os requisitos que foram originalmente propostos estão sendo cumpridos e, se necessário, reagir sugerindo adaptações para garantir que o sistema base continue executando suas funções principais de forma satisfatória.

Existem na literatura diversos trabalhos que propõem o uso de requisitos em tempo de execução, como sistemas adaptativos ou autônomos. No entanto, não há na literatura uma representação formal e explicita do que são requisitos em tempo de execução e quais são os objetivos por trás de sua utilização. Ainda dentro do contexto de requisitos em tempo de execução, a maioria das propostas e métodos que propõem sua utilização usam linguagens de modelagem e métodos próprios para representar o que são os requisitos de sistema e como utilizá-los em tempo de execução. Não há, assim, um senso comum no domínio de uso de requisitos em tempo de execução, resultando numa sobrecarga excessiva de termos e construtos.

Essa falta de formalização e de consenso dentro do domínio e os problemas de comunicação gerados pela falta de um vocabulário formal e bem fundamentado foram as motivações principais para que fosse realizado um estudo sistemático sobre as diversas metodologias existentes na literatura de requisitos em tempo de execução e através do conhecimento adquirido fosse construída a RRO (*Runtime Requirements Ontology*), uma ontologia de referência de domínio sobre o uso de requisitos em tempo de execução. RRO foi construída através da metodologia de construção de ontologias SABiO e é fundamentada em UFO com objetivo atuar como uma representação formal do conhecimento dentro do domínio de requisitos em tempo de execução, fornecendo, desta maneira, uma descrição precisa de todas as entidades principais que compõem o domínio e estabelecendo um vocabulário comum para ser utilizado por engenheiros de software e *stakeholders*.

Palavras-chaves: Requisitos de Software, Tempo de Execução, Ontologias, UFO, RRO

Abstract

In the last years, we have witnessed a growing interest in software systems that can monitor their environment and, if necessary, change their requirements in order to continue to fulfill their purpose. This particular kind of software usually consists of the main system responsible for the main functionality, along with a component that monitors the main system, analyzes the data collected and then,if necessary, reacts properly to make sure that the system continues to fulfill its requirements and executing its main functions in a proper way.

There are many works in the literature that propose different solutions to this issue, such as adaptive or autonomic systems. However, there is not, in the scientific literature, a formal and explicit representation of what requirements at runtime are and what are the primary goals of their use. Still, in this context, most of the existing frameworks and methods that proposes their use have their own modeling languages and ways to represent, specify and make use of requirements at runtime. So, there is no common ground or a common sense about the use of requirements at runtime, thus resulting in a domain with overloaded concepts.

This lack of consensus inside the presented domain and the problems caused by the lack of a formal and well-foundamented vocabulary about the domain were the main motivations for the execution of a systematic mapping (SysMap) about the methodologies that exists in the literature about requirements at runtime (RRT) and through the acquired, RRO (Runtime Requirements Ontology) was built.RRO is a domain reference ontology, grounded in UFO and built throuth SABiO framework to be a formal representation about the domain of RRT, providing a precise description about the entities that are part of the domain and establishing a commom vocabulary to be used for software engineers and stakeholders.

Keywords: Requirements, Runtime, Ontology, UFO, RRO

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

UFO	Unified Foundational Ontology
RRO	Runtime Requirements Ontology
SABiO	Systematic Approach for building Ontologies
CQ	Competency Question
SRO	Software Requirements Ontology
RF	Requisitos Funcionais
RNF	Requisitos Não-Funcionais
AP	Anti-Pattern
AssCyc	Association Cycle
ImpAbs	Imprecise Abstraction
RelSpec	Relation Specialization
RepRel	Repeatable Relator Instances
RelRig	Relator Mediating Rigid Types
XML	eXtensible Markup Language
OCL	Object Constraint Language

Sumário

1 Introdução

Este capítulo apresenta o contexto, as motivações, o método e a organização estrutural adotada no desenvolvimento desse trabalho.

1.1 Contexto

Nos últimos anos, temos assistido um constante aumento de interesse em sistemas de software que são capazes de monitorar seu meio de execução e, se necessário, adaptar seus requisitos para continuar cumprindo seu propósito (DALPIAZ et al., 2013; SOUZA et al., 2013). Esse tipo de software normalmente consiste em um software base, comumente chamado na literatura de *target* ou *base system*, que é responsável por executar a função principal para qual o software foi desenvolvido, além de um software, interno ou externo, responsável por monitorar o software base, realizar uma análise dos dados coletados para verificar se os requisitos que foram originalmente propostos estão sendo cumpridos e, se necessário, reagir disparando adaptações para garantir que o sistema base continue executando suas funções principais de forma satisfatória.

Existem na literatura, diversos trabalhos que propõem o uso de requisitos em tempo de execução, como sistemas adaptativos, autônomos ou sistemas sensíveis ao contexto (SOUZA, 2012; HUEBSCHER; MCCANN, 2008; CHENG et al., 2009; LEMOS et al., 2013). Dentro desse contexto, Bencomo et al. (2013) enfatiza que o uso de modelos de requisitos em tempo de execução podem fornecer informações valiosas sobre o estado e o contexto do sistema durante a operação do sistema, ajudando muito na tomada de decisões além de permitir o monitoramento de potenciais inconsistências. Tais benefícios de utilização resultaram num rápido crescimento de pesquisas sobre esse assunto na última década. Contudo, a maioria das propostas encontradas na literatura usam suas próprias linguagens de modelagem e metodologias para representar o que são os requisitos de sistema e como utilizá-los em tempo de execução. Não há, assim, um senso comum no domínio de uso de requisitos em tempo de execução, resultando numa sobrecarga excessiva de termos e construtos, ou seja, entidades semanticamente diferentes sendo representadas pelo mesmo nome ou objeto (em um possível modelo representativo), o que pode causar falhas de comunicação e de entendimento entre os atores que trabalham com esse domínio. Tais falhas, por sua vez, podem gerar problemas maiores.

1.2 Motivação

Embora já amplamente discutida na literatura científica e aplicada em diversos tipos de pesquisas envolvendo sistemas adaptativos e de monitoramento, a área de requisitos em tempo de execução é relativamente recente, devido a isso é possível encontrar alguns problemas:

1. Não existe na literatura uma formalização para a área de requisitos em tempo de execução que possa servir como um modelo de referência para o domínio.
2. Não existe na literatura uma representação ontológica sobre os tipos de requisitos em tempo de execução.
3. Existem diversas abordagens que propõem o uso de requisitos em tempo de execução dentro da literatura que utilizam-se de um vocabulário similar, o que acaba causando uma sobrecarga de construtos dentro do domínio.

Nesse contexto, foi realizado um mapeamento sistemático de estudos científicos sobre o uso de requisitos em tempo de execução e, a partir do conhecimento adquirido, uma ontologia de referência de domínio foi proposta para servir como um vocabulário formal para o domínio, auxiliando na resolução de problemas de comunicação entre os diversos atores envolvidos com a utilização de requisitos em tempo de execução.

1.3 Objetivos

Esse trabalho tem como objetivo geral propor uma representação formal do uso de requisitos em tempo de execução, fornecendo uma descrição precisa e bem fundamentada das entidades que compõem o domínio para auxiliar na comunicação e solução de problemas dentro da área de engenharia de requisitos. Esse objetivo geral pode ser decomposto nos seguintes objetivos específicos:

- Estudar a literatura existente sobre o domínio escolhido a fim de definir uma conceituação compartilhada sobre o tema;
- Definir uma ontologia de referência sobre requisitos em tempo de execução em um alto nível ontológico de abstração;
- Avaliar a ontologia por meio de processos de verificação e validação de ontologias existentes na literatura.

1.4 Método

Esse trabalho foi conduzido da seguinte forma:

Revisão da Literatura: foi realizado um extenso mapeamento sistemático da literatura, para conhecer o estado da arte sobre a utilização de requisitos de software durante o tempo de execução de um programa. Primeiramente foram feitas buscas e leituras de forma não sistemáticas sobre tema, com o objetivo de adquirir mais conhecimento sobre o assunto a ser tratado durante a pesquisa. Após ter atingido um nível de conhecimento satisfatório sobre o uso de requisitos em tempo de execução, o mapeamento sistemático foi realizado de acordo com as metodologias propostas em (KITCHENHAM; CHARTERS, 2007). O mapeamento sistemático foi escolhido como forma principal de aquisição de conhecimento para a construção da ontologia, pois o mesmo fornece um método já amplamente aceito dentro da literatura para o levantamento de um conhecimento compartilhado e consensual dentro de um determinado domínio, insumo que é essencial para a construção de uma ontologia. Durante o processo de mapeamento sistemático foram realizadas pesquisas em diversos mecanismos de buscas de trabalhos científicos. entretanto foram utilizadas regras e critérios bem definidos para incluir ou excluir um trabalho dentro do mapeamento.

Construção e Formalização da Ontologia: após todo o processo do mapeamento sistemático ter sido concluído e já de posse de seus resultados, a ontologia, denominada RRO—*Runtime Requirements Ontology*—começou a ser desenvolvida. RRO é uma ontologia de referência de domínio fundamentada em UFO (GUIZZARDI, 2005), construída segundo o método SABiO (FALBO, 2014) e reutilizando ontologias relevantes existentes. UFO foi escolhida como a ontologia de fundamentação base para a construção de RRO pois, além de ser a ontologia de fundamentação sugerida por SaBiO, fornece uma grande quantidade categorias de entidades e relacionamentos que permitem uma representação fidedigna do domínio. O processo de construção e formalização de RRO foi altamente iterativo, com reuniões semanais que contaram com a participação de diversos pesquisadores seniores, atuando diretamente como especialistas de domínio e engenheiros de ontologia. SABiO foi escolhido como método para construção de RRO pois é um método focado na construção de ontologias de domínio.

Avaliação da Ontologia: conforme sugerido pelo método SABiO, uma ontologia de referência pode ser avaliada através de processos de verificação e validação. Para realizar a verificação de RRO foi construída uma tabela com os elementos da ontologia que respondem as questões de competência (CQs) (GRÜNINGER; FOX, 1995) que foram levantadas. Como segunda forma de verificação, RRO foi remodelada utilizando OntoUML pra que fosse possível utilizar as ferramentas de verificação existentes na linguagem (verificação semântica e anti-padrões), que não são diretamente aplicáveis em ontologias que tem parte de sua fundamentação baseadas em UFO-B e UFO-C. Para validação, foi realizada a instanciação de RRO com 2 projetos propostos na literatura que utilizam

requisitos em tempo de execução.

Escrita da Dissertação: os resultados obtidos nas etapas anteriores foram descritos nessa dissertação de mestrado. O trabalho realizado para essa dissertação também deu origem à publicação (DUARTE et al., 2016), onde RRO é apresentada em forma de artigo científico à comunidade internacional de ontologias.

1.5 Organização da Dissertação

Nesse primeiro capítulo da dissertação, foram apresentadas as ideias gerais dessa dissertação, foram descritos o contexto de aplicação, os objetivos e a metodologia adotada. Além dessa introdução a dissertação organiza-se da seguinte forma:

Capítulo 2 - Revisão da Literatura

Na primeira parte do Capítulo 2, são apresentadas e discutidas as principais metodologias e o estado-da-arte sobre requisitos de software em tempo de execução, as metodologias e conceitos apresentados aqui são fruto do mapeamento sistemático citado anteriormente. Na segunda parte, são descritos principais conceitos relacionados às ontologias relevantes a essa dissertação, UFO, a ontologia de fundamentação utilizada como base para construção desse trabalho é apresentada em conjunto com as ontologias que foram reusadas nas construção de RRO.

Capítulo 3 - RRO - Runtime Requirements Ontology

No Capítulo 3, é detalhado todo o processo de construção de RRO, desde o mapeamento sistemático que foi realizado como processo de aquisição de conhecimento consensual sobre o domínio, o levantamento dos requisitos da ontologia, que são apresentados na forma de questões de competência e a construção e formalização até a obtenção do produto final que é a ontologia de referência de domínio devidamente representada em um modelo conceitual.

Capítulo 4 - Avaliação da Ontologia

O Capítulo 4 descreve todo o processo de avaliação que foi realizado sobre RRO para garantir que a mesma foi construída corretamente e atende a seus requisitos. Essa avaliação foi dividida em verificação e validação e realizada através de processos e métodos existentes na engenharia de ontologias e na literatura especifica sobre metodologias verificação e validação de ontologias.

Capítulo 5 - Conclusões

O Capítulo 5 descreve as conclusões obtidas na realização desse trabalho. As principais contribuições listas e possíveis trabalhos futuros e pontos para a continuação dessa pesquisa são apresentados.

Apêndice A - Mapeamento Sistemático

Apresenta o método aplicado no mapeamento sistemático realizado sobre uso de requisitos em tempo de execução que serviu como base teórica para a realização desse trabalho.

2 Fundamentação Teórica

Nesse capítulo serão apresentados os conceitos referentes ao estado da arte do uso de requisitos em tempo de execução e que dão o embasamento teórico para a criação e desenvolvimento deste trabalho. Também serão apresentadas propostas que fazem uso de requisitos em tempo de execução que serão utilizadas posteriormente para validar a ontologia proposta.

O capítulo é divido da seguinte forma: A Seção 2.1 apresenta o conceito de requisitos em tempo de execução, a Seção 2.2 apresenta dois *frameworks* que fazem uso de requisitos em tempo de execução que serão utilizados para instanciar RRO no capítulo referente à avaliação da ontologia, a Seção 2.3 contém fundamentação ontológica utilizada como base para a construção de RRO, apresentando UFO, a ontologia de fundamentação na qual RRO foi fundamentada e as duas ontologias que foram reutilizadas na construção de RRO. Na Seção 2.4 é apresentado o método SABiO, que foi utilizado como a metodologia principal para a construção de RRO. Por fim, a Seção 2.5 apresenta trabalhos relacionados ao dessa dissertação.

2.1 Requisitos de Software

Engenharia de Requisitos (ER) é o campo da Engenharia de Software que se dedica a entender, modelar, analisar, negociar, validar e documentar requisitos para sistemas de software (CHENG; ATLEE, 2007). Na literatura sobre Engenharia de Requisitos existe uma forte sobreposição do termo "requisito" e, dessa forma, diferentes trabalhos referem-se a requisitos como diferentes tipos de entidades como por exemplo: um artefato, um objetivo, um desejo ou uma propriedade de um sistema de software.

Em uma das mais antigas definições sobre requisitos, Ross e Jr (1977) afirmam que requisitos são a definição das funcionalidades que um sistema deve prover e especificam como o sistema será construído. Essa definição aproxima a natureza do requisito à de um artefato que documenta e especifica o que um sistema deve fazer.

Em seu trabalho, Zave e Jackson (1997) definem *requisito* como uma propriedade desejada de um ambiente que compreende o sistema de software e seu meio e que pretende expressar as intenções e os desejos dos *stakeholders* de um determinado projeto de desenvolvimento de software. Esta visão claramente aproxima o requisito de um desejo ou objetivo que um *stakeholder* tem para com um sistema. Em um trabalho mais recente, Wang et al. (2014a) utilizam o termo *high-level requirement* (requisito de alto nível) para descrever um conceito de requisito que representa um desejo ou uma intenção de uma ou

mais partes interessadas no projeto de software, aproximando sua definição com a proposta por Zave e Jackson (1997).

Pelas definições apresentadas acima podemos perceber que o requisito é uma entidade complexa e consequentemente pode ser visto e representado de diversas outras formas dentro de um processo de modelagem conceitual. Partimos da tese que requisitos nascem originalmente como objetivos dos *stakeholders* do software a ser construído e ao longo do ciclo de vida do software são descritos de formas variadas, tornando-se artefatos.

No entanto, é importante perceber que essa definição de requisito como um artefato que descreve um objetivo pode ter vários níveis de abstração, por exemplo: se o processo está no começo da fase de análise de requisitos o requisito será documentado em linguagem natural para que possa ser entendido pelos *stakeholders* do projeto. Caso o requisito precise ser processado e consumido por uma máquina, durante a execução de um software, o mesmo deve ser descrito em uma linguagem que permita um raciocínio da máquina sobre ele.

Tais características que demonstram a complexidade do requisito e a inexistência de uma definição formalizada e consensual sobre o uso de requisitos em tempo de execução (*requirements at runtime*), motivando a pesquisa sobre o assunto.

O uso de requisitos em tempo de execução é uma área de interesse relativamente nova dentro da Engenharia de Requisitos e, consequentemente, da Engenharia de Software. No entanto, é possível encontrar diversos métodos que propõem a utilização de requisitos durante o tempo de execução de um software para os mais variados motivos como, por exemplo, para monitoramento de sistemas de software baseado em requisitos ou para o uso em sistemas adaptativos, que são monitorados e podem receber mudanças em seus requisitos durante sua operação.

Dentre as várias propostas existentes na literatura, algumas propõem definições diretas para a entidade "requisito em tempo de execução":

- Bencomo et al. (2010a) propõem que requisitos não só sejam materializados como entidades presentes durante a execução do software mas também que seja mantida uma rastreabilidade do requisito em tempo de execução para com os modelos de requisitos de alto nível.
- Qureshi e Perini (2010) definem que requisitos em tempo de execução são expressados por requisições do usuário que são capturadas pela interface gráfica do software e monitoradas para avaliar se essas estão de acordo com as funcionalidades que foram definidas na especificação original do software.
- Souza et al. (2013) têm uma visão similar, com a qual definem um requisito em tempo de execução como classes ou *scripts* que representam requisitos sendo instanciados

durante a execução de um software.

- Mais recentemente, Dalpiaz et al. (2013) propõem, no contexto de GORE (*Goal Oriented Requirements Engineering*), a distinção entre modelos de objetivos em tempo de design — que são utilizados durante o processo de construção do sistema — e modelos de objetivos em tempo de execução — utilizados para analisar o comportamento do sistema durante sua execução em relação a seus requisitos.

Esses e outros trabalhos ilustram que requisitos são entidades de natureza complexa que pode existir durante todo ciclo de vida de um software, tanto em tempo de desenho quando em tempo de execução. No entanto, seu uso em tempo de execução é relativamente recente e ainda não há uma representação formal ou uma descrição consensual e compartilhada de sua definição. Esta ausência de uma conceituação formalizada motivaram uma pesquisa mais profunda sobre o domínio e o desenvolvimento de uma ontologia de referência de domínio com o objetivo de fornecer uma interpretação bem fundamenta sobre requisitos e suas contrapartes utilizadas durante a execução de um programa.

2.2 Métodos sobre Requisitos em Tempo de Execução

Nessa seção serão apresentadas abordagens que propõem o uso de requisitos em tempo de execução baseados em modelos de requisitos. Os experimentos práticos dos métodos apresentados aqui serão utilizadas posteriormente para a realização de uma instanciação de RRO como uma forma primária de validação da ontologia.

2.2.1 Zanshin Framework

Zanshin (SOUZA, 2012; SOUZA et al., 2013; SOUZA et al., 2013) é um *framework* para o desenvolvimento de sistemas adaptativos baseado na engenharia de requisitos de software. A proposta principal do método é fazer com que os elementos do *feedback loop* que operacionaliza a adaptação se tornem entidades importantes dos modelos de requisitos utilizados (ANGELOPOULOS; SOUZA; PIMENTEL, 2013). Para isso, Zanshin utiliza-se de modelos de requisitos baseados em GORE (*Goal Oriented Requirements Engineering*) para representar os requisitos do sistema. Os modelos utilizados pelo Zanshin *framework* possuem adições de novos elementos que, quando combinados com elementos clássicos de GORE (*Goals*, *Softgoals*, *Tasks* e *AND-OR refinements*)[1], são capazes de representar

[1] Objetivos (*goals*) representam estados do mundo que deseja-se alcançar, *softgoals* são objetivos que não possuem critério claro de satisfação (ex.: "deseja-se minimizar custos", porém não se especifica qual é o valor máximo aceitável), tarefas (*tasks*) representam sequências de atividades que são conduzidas geralmente para satisfazer parcial ou totalmente um objetivo; refinamentos (*refinements*) permitem decompor elementos em partes menores ou estabelecer critérios de satisfação entre tarefas e objetivos. Para uma visão geral de GORE e seus elementos, vide (LAMSWEERDE, 2001).

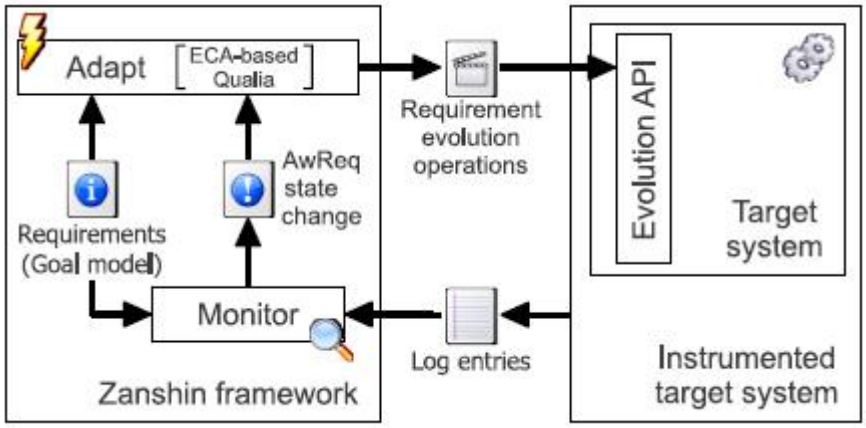

Figura 1 – Representação da Arquitetura do Zanshin Framework (SOUZA, 2012)

estratégias de monitoramento e adaptação durante a execução de um sistema de software que podem ser disparadas pelo sistema para reagir a situações críticas. Os novos elementos de GORE propostos são chamados de *Awareness Requirements* e *Evolution Requirements*.

A Figura 1 apresenta a arquitetura básica do Zanshin. *Awareness Requirements* (AwReqs) são requisitos que se referem ao estado assumido por outros requisitos durante o tempo de execução de um sistema de software (SOUZA et al., 2011), em outras palavras, AwReqs representam situações nas quais os *stakeholders* do sistema de software desejam que este último seja capaz de adaptar-se (ANGELOPOULOS; SOUZA; PIMENTEL, 2013).

Evolution Requirements (EvoReqs) são aqueles que descrevem como outros requisitos no modelo devem mudar/evoluir em resposta à falha de um AwReq (ANGELOPOULOS; SOUZA; PIMENTEL, 2013). EvoReqs atuam diretamente nos requisitos de um sistema por meio de estratégias de adaptação. Durante o tempo de execução, eles são responsáveis por garantir que esse sistema continue a cumprir o que foi especificado pelos *stakeholders*.

A Figura 2 apresenta o modelo de requisitos orientados a objetivos de um sistema de agendamento de reuniões dentro de uma empresa chamado *Meeting Scheduler*, criado como ferramenta de demonstração do Zanshin Framework. O modelo do *Meeting Scheduler* será utilizado como exemplo para demonstrar os conceitos propostos pelo Zanshin e, posteriormente, será também utilizado para uma instanciação direta de RRO. Na figura, AwReqs aparecem na forma de um pequeno circulo preto anexado a uma seta, representando os requisitos que devem ter seu estado monitorado.

Já os EvoReqs não são representados graficamente no modelo de requisitos proposto no Zanshin entretanto, eles são representados na forma de um artefato de código a ser utilizado pelo Zanshin quando um AwReq falha. A Figura 3 apresenta o algoritmo de um EvoReq disparado quando o AR1 (*Never Fail*) representado na Figura 2 falha. Se, por algum motivo, uma reunião não consegue ser marcada no sistema, o AR1 dispara o

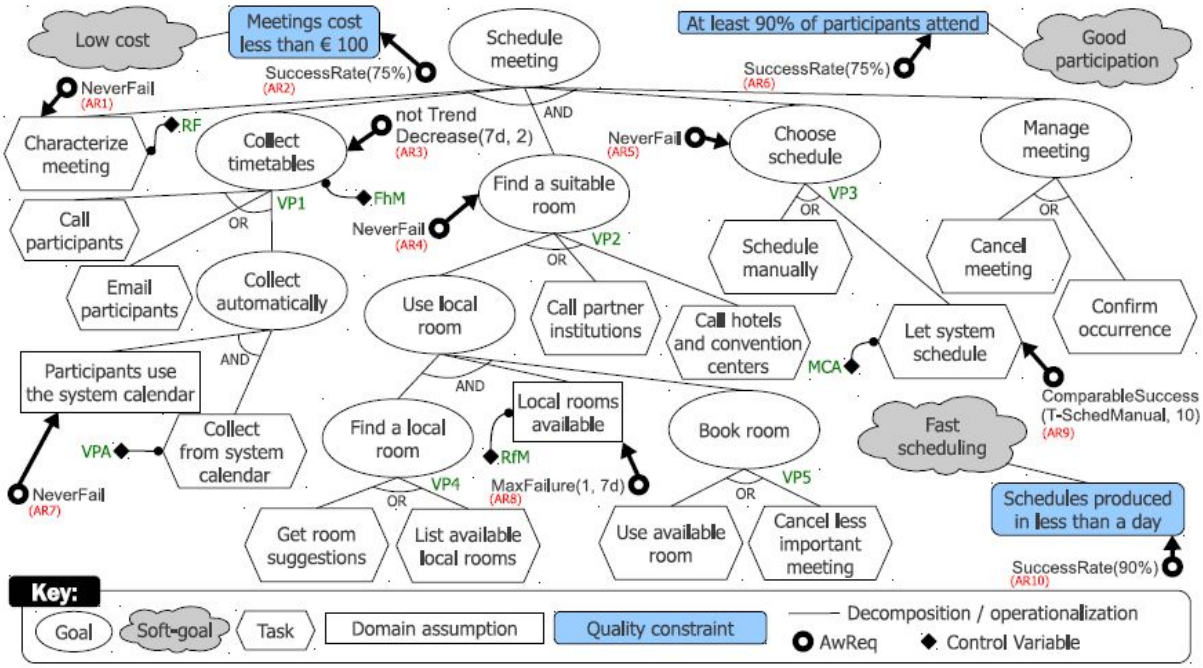

Figura 2 – Exemplo de Modelo de Requisitos do *Meeting Scheduler* utilizando o Zanshin Framework (SOUZA, 2012)

```
t' = new-instance(T_CharactMeet);
copy-data(t, t');
terminate(t);
rollback(t);
wait(5s);
initiate(t');
```

Figura 3 – EvoReq Retry Characterize Meeting (SOUZA, 2012)

EvoReq *Retry Characterize Meeting* que faz com que o sistema de *rollback*, espere cinco segundos e tente marcar a reunião novamente.

2.2.2 ReqMon

ReqMon (ROBINSON, 2006) é um *framework* de monitoramento de requisitos de sistemas de software durante tempo de execução desenvolvido com o objetivo de aumentar a visibilidade das políticas de conformidade de sistemas de software. ReqMon foi criado com o intuito de ser utilizado por todos os usuários de um sistema software, desde analistas e profissionais da área de tecnologia da informação a usuários com nível de permissão que lhes permita um acesso, em tempo de execução, a um *feedback* sobre a satisfação dos requisitos de um sistema de software.

O framework apresenta, então, três contribuições: (1) uma linguagem de definição de requisitos baseada em KAOS (LAMSWEERDE et al., 1991); (2) uma metodologia sistemática para análise dos requisitos em tempo de design, identificação de possíveis

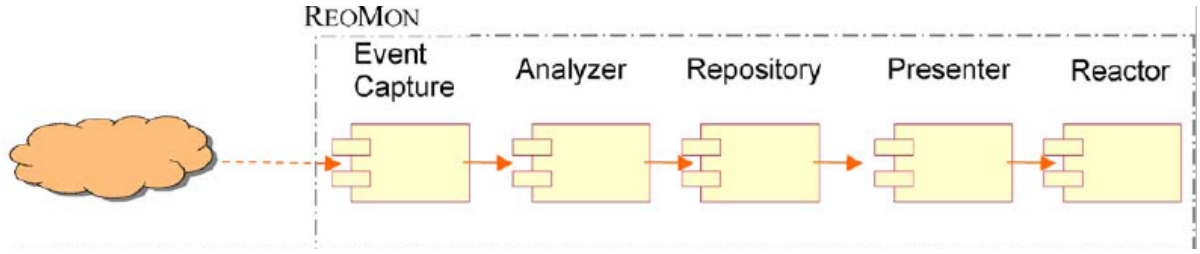

Figura 4 – Representação da Arquitetura do ReqMon Framework (ROBINSON, 2006)

obstáculos para a satisfação dos requisitos e análise do *feedback* provido pelo monitor em tempo de execução; (3) uma ferramenta para monitoramento de requisitos construída a partir de padrões aceitos pela comunidade científica e que visa prover um serviço completo para o monitoramento de sistemas de software.

A Figura 4 apresenta a arquitetura básica do ReqMon com seus principais componentes. *Event Capturer* monitora e captura, em tempo de execução, qualquer desvio nos requisitos do sistema sendo monitorado. O *Analyzer* é responsável por atualizar os status dos monitores do sistema principal, enquanto o *Repository* atua como um banco de dados de eventos e histórico de indicadores monitorados.

ReqMon propõe ainda a criação de dois outros componentes para compor o monitorador principal: uma ferramenta gráfica chamada *Presenter* para que o usuário final possa acompanhar em tempo real o monitoramento dos requisitos e uma ferramenta de ação chamada *Reactor* que é responsável por disparar alterações (adaptações) em resposta às mudanças nos estados dos requisitos monitorados.

Da mesma forma que no Zanshin Framework, ReqMon caracteriza requisitos como objetivos (goals) e os entende como entidades complexas, que existem tanto em tempo de desenho, na forma de artefatos documentados quanto seus análogos em tempo de execução, organizados em modelos de monitoramento de requisitos que visam permitir a avaliação e rastreabilidade dos requisitos definidos previamente.

A Figura 5 apresenta o requisito *Retailer Purchase* de uma página de comércio eletrônico utilizado como exemplo da aplicação do ReqMon Framework. A árvore de objetivos define o escopo e pontos de interesse para o sistema de monitoramento. Objetivos e sub-objetivos são monitorados e, consequentemente, possíveis eventos relacionados a eles são interpretados como satisfação ou violação de um requisito e propagados para cima na estrutura da árvore de forma a prover aos analistas responsáveis uma capacidade maior de interpretação dos eventos internos do sistema monitorado. Caso um sub-objetivo falhe, ele é propagado até a raiz da árvore e permite a um analista responsável rastrear a ocorrência da falha através da ferramenta de monitoramento proposta no ReqMon.

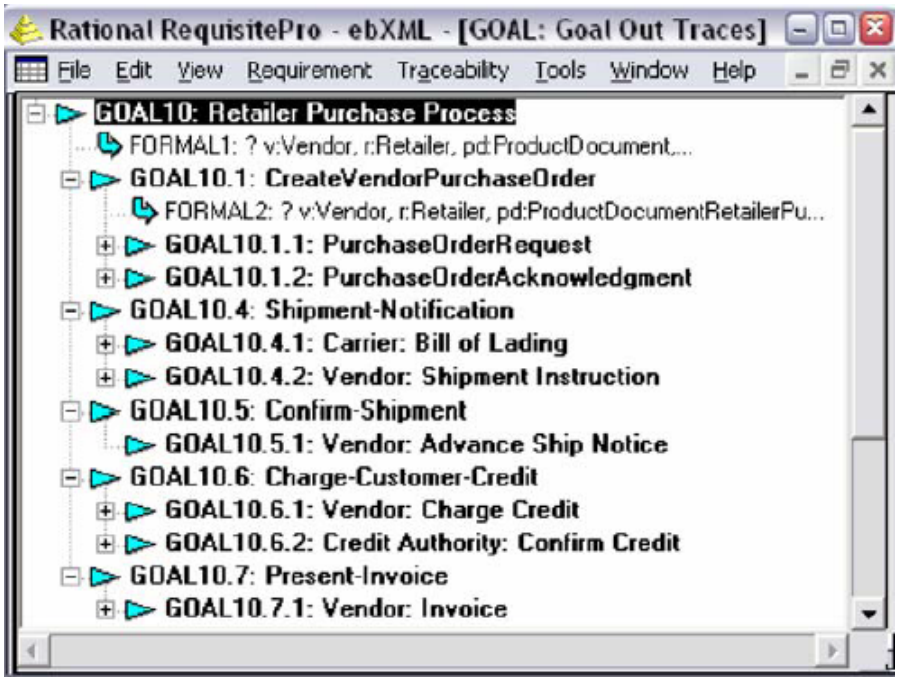

Figura 5 – Representação do Requisito *Retailer Purchase* escrito na forma de arvore de objetivos (ROBINSON, 2006)

2.3 Ontologias

Uma das definições mais aceitas para o significado de ontologia é a de Borst (1997), que define o termo como "uma especificação formal de uma conceituação compartilhada". No entanto, o termo *ontologia* possui vários significados e é utilizado por diferentes comunidades. Na Filosofia, mais precisamente no ramo que lida com a natureza e a estrutura da realidade, Aristóteles define ontologia como "a ciência do ser como um ser", ou seja, uma ciência que foca em estudar a estrutura e as características das coisas independentemente de quaisquer outras considerações (GUARINO; OBERLE; STAAB, 2009).

Nas sub-áreas da Ciência da Computação, como Engenharia de Software e Web Semântica, ontologias são um tipo de artefato computacional utilizado para modelar formalmente a estrutura de um sistema de informação. Mais precisamente na área da Engenharia de Software, área foco desse trabalho, ontologias são utilizadas para remover ambiguidades conceituais e facilitar o compartilhamento de informações e interoperabilidade de sistemas (JASPER; USCHOLD et al., 1999). Conceitos, relações e axiomas previamente definidos e fundamentados são utilizados para descrever um modelo de domínio uniforme e não ambíguo de entidades e suas relações, fornecendo assim uma conceituação sobre o domínio modelado (FALBO; GUIZZARDI; DUARTE, 2002).

Com relação às classificações existentes para ontologias, Guarino (1998) define uma das mais referenciadas classificações de ontologias, de acordo com sua especificidade. Tal classificação é a adotada nesse trabalho e demonstrada na Figura 6:

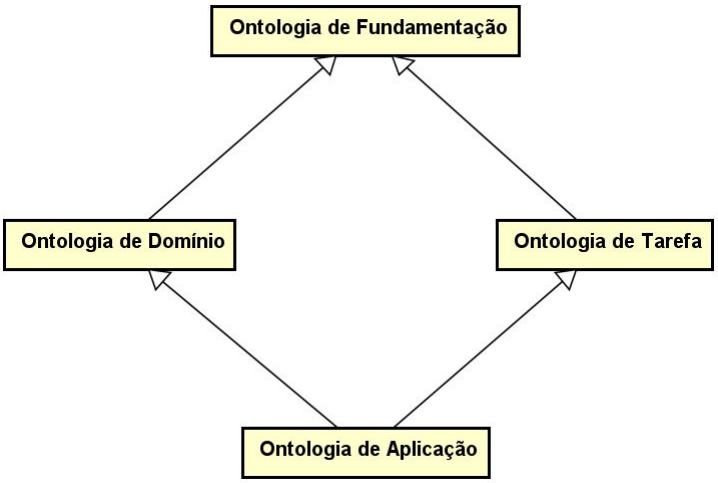

Figura 6 – Tipos de Ontologias de acordo com seu nível de especificidade e dependências entre esses níveis (GUARINO, 1998).

- Ontologias de Fundamentação ou de alto nível descrevem conceitos gerais como espaço, tempo, objetos, eventos, etc. Dessa forma, se tornam independentes de um determinado domínio ou problema.
- Ontologias de Domínio descrevem um vocabulário relativo a um determinado domínio, por exemplo, Engenharia de Software ou Medicina. Ontologias de Domínio utilizam-se de conceitos especializados de conceitos presentes em ontologias de fundamentação.
- Ontologias de Tarefa descrevem tarefas ou atividades genéricas, como por exemplo, compra e venda. Assim como em ontologias de domínio, seus conceitos são especializados de entidades existentes em ontologias de fundamentação.
- Ontologias de Aplicação descrevem conceitos que dependem de domínios e tarefas específicas, que são comumente especializados de duas ontologias relacionadas.

Outra classificação largamente utilizada (ortogonal à classificação acima) é a que separa ontologias em dois grupos distintos. **Ontologias de Referência**, que são um tipo especial de modelo conceitual utilizado para representar um determinado domínio formalmente, por meio da especialização de conceitos oriundos de uma ontologia de fundamentação, além de servirem como um vocabulário não ambíguo do domínio em questão (FALBO; BERTOLLO, 2009). Guizzardi (2007) tem uma visão análoga e afirma que ontologias de referência devem ser construídas de forma que façam a melhor descrição possível de fragmentos da realidade e sejam capazes de ajudar os humanos em tarefas que necessitem do estabelecimento de um consenso. Já as **Ontologias Operacionais** são versões implementadas de ontologias de referência capazes de serem interpretadas e utilizadas por máquinas. Diferentemente das ontologias de referência, ontologias operacionais não são

focadas na adequação da representação mas sim em garantir propriedades computacionais desejadas (FALBO, 2014).

Por fim, é importante acrescentar que esse trabalho é focado na utilização de ontologias de fundamentação para a criação de uma ontologia de referência de domínio e que os demais tipos e classificações de ontologias mencionadas não serão mais abordados.

2.3.1 UFO

UFO (GUIZZARDI, 2005) é uma ontologia de fundamentação que foi desenvolvida baseada em várias teorias das áreas de Lógica Filosófica, Ontologias Formais, Linguística e Psicologia Cognitiva, que por muitos anos vem sendo utilizada com sucesso para prover semântica para ontologias de domínio através de um sistema completo de categorias de entidades que permitem um mapeamento sólido de entidades e relações existentes no mundo real. Nessa subseção serão apresentados todos os conceitos de UFO relativos ao desenvolvimento desse trabalho através da explicação de seus três fragmentos.

- **UFO-A** lida com objetos, seus aspectos estruturais, seus tipos, suas partes, os papéis que são capazes de desempenhar suas propriedades intrínsecas e relacionais. UFO-A é comumente chamada de "uma ontologia de objetos".
- **UFO-B** lida com eventos, suas partes, os relacionamentos entre eventos, a participação de objetos em eventos e as propriedades temporais das entidades. UFO-B é comumente chamada de "uma ontologia de eventos".
- **UFO-C** lida com aspectos sociais como objetivos, agentes, intenções e compromissos. UFO-C especializa boa parte de seus conceitos de entidades ja definidas em UFO-A e UFO-B e por isso é a mais específica das 3 partes de UFO.

2.3.2 UFO-A

UFO-A, como a primeira das três partes de UFO, tem seus conceitos utilizados e especializados pelos demais fragmentos. O primeiro e mais genérico conceito de UFO é o conceito de Entidade (*Entity*). *Entity* representa algo que possa ser concebido ou percebido. *Entity* é especializado em duas categorias básicas que distinguem as entidades existentes em UFO: *Individuals* e *Universals*, onde *Individuals* são entidades que existem no mundo real, como por exemplo uma pessoa ou a Lua, e que possuem uma identidade única que as define. *Universals* são padrões de características, que são instanciadas por *Individuals*. A Figura 7 apresenta a distinção básica realizada em UFO.

Individuals, também sofrem especialização em dois conceitos disjuntos: *Concrete Individuals* e *Abstract Individuals*. *Endurants* são *Concrete Individuals* que existem no tempo enquanto mantêm sua identidade. Eles são divididos em três classes:

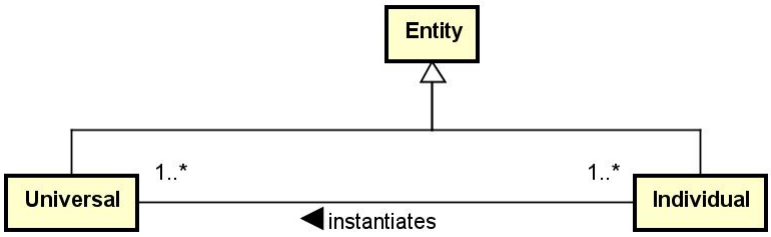

Figura 7 – Fragmento de UFO-A exibindo a distinção básica entre *Individuals* e *Universals*.

- *Substantials* são existencialmente independentes, ou seja, que não dependem necessariamente de outros indivíduos para existir. Entidades inanimadas (*Objects*) como uma cadeira ou a Lua são um tipo de *Substantial*.

- *Moments* são *Concrete Individuals* que denotam uma propriedade particular instanciada ou uma propriedade ou característica de um indivíduo (ex: a altura de um objeto) e são existencialmente dependentes de outros *Individuals*. *Intrinsic Moments* e *Relators* são especializações de *Moment*, o primeiro é dependente de um único *Individual* (ex: a cor de uma camisa), já *Relators* são existencialmente dependentes de vários *Individuals* (ex: um casamento, que depende da esposa e do marido).

- *Situations* são um tipo complexo de *Endurant*, eles são constituídos por muitos *Endurants*, incluindo outras *Situations* e representam uma porção da realidade que pode ser compreendida como um todo. Um exemplo de *Situation* é **Aline está feliz e sorridente** onde o *Substantial* **Aline** e seus *Moments* **feliz** e **sorridente** estão presentes (relação *is present in*) na *Situation* **Aline está feliz e sorridente**. *Situations* satisfazem (relação *satisfies*) *Propositions*, que são uma representação abstrata (especializam *Abstract Individual*) de uma classe de *Situations* referenciadas por um *Intentional Moment*. *Relators* (também chamados de *Relational Moments*) são *Individuals* que conectam outros *Individuals*. Em outras palavras, um *Relator* pode ser entendido como um contrato que media a relação entre duas entidades: uma matricula, um casamento ou mesmo um tratamento médico podem ser entendidos como *Relators* já que são responsáveis por mediar a relação entre dois outros indivíduos.

A Figura 8 apresenta os tipos *Inviduals* discutidos até então, as entidades que aparecem em cinza já foram explicadas anteriormente e aparecem apenas para demonstrar a continuidade do modelo.

Voltando ao conceito de *Universal*, trataremos agora das especializações que essa entidade sofre, no entanto, é importante acrescentar que alguns dos conceitos já explicados na parte de *Individuals* possuem uma contraparte do lado dos *Universals* em UFO, isso se deve ao fato de *Individuals* serem instanciações de *Universals* e, por esse motivo, esses conceitos não serão explicados novamente.

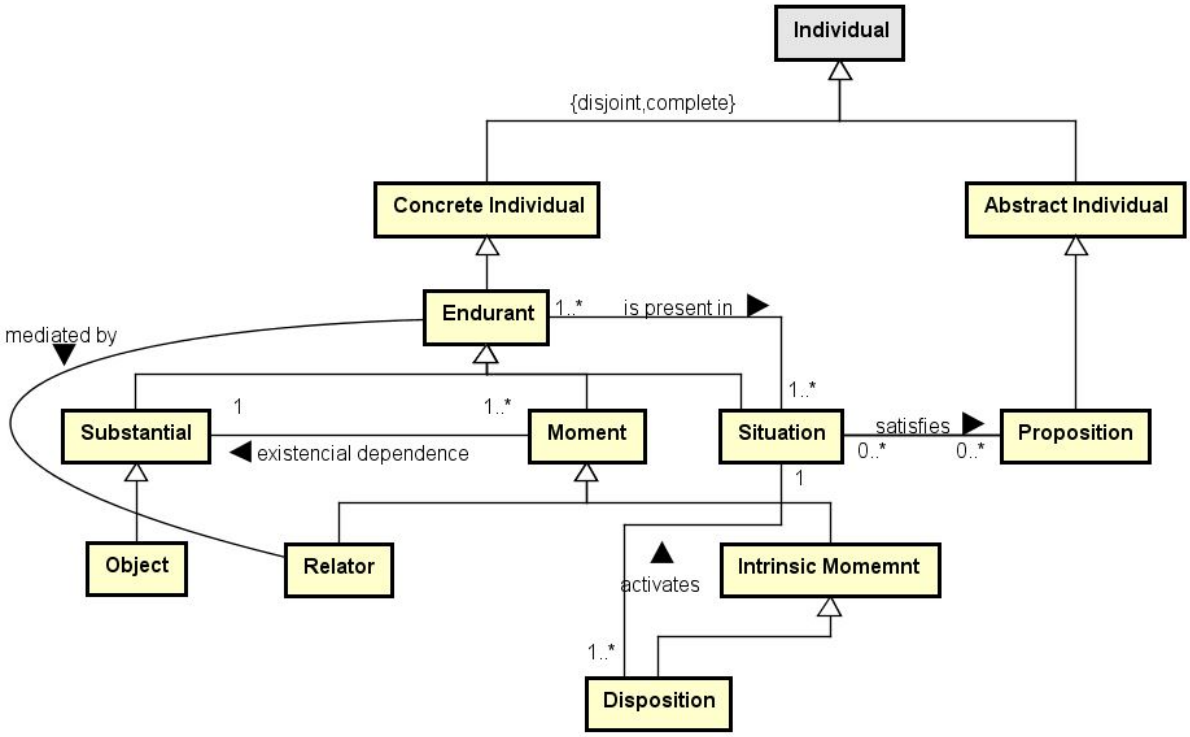

Figura 8 – Fragmento de UFO-A exibindo as especializações de *Individuals*.

Monadic Universal e *Relation Universal* são as especializações diretas de *Universal*. *Relations* são entidades que unem outras entidades. Levando-se em consideração a fundamentação filosófica de UFO, *Relation Universal* possui duas especializações: *Material Relations* possuem uma estrutura material própria e são derivadas de *Relators*, já previamente explicados. Exemplos de *Material Relations* incluem **estar casado com** e **trabalhar em**. O outro sub-tipo de *Relation Universal* é denominado *Formal Relation*, que representa as relações que acontecem entre duas ou mais entidades sem que outro indivíduo intervenha e que podem ser definidas a partir propriedades intrínsecas dos dois indivíduos. A relação "mais alto que" é um exemplo de relação formal pois o que torna a relação verdadeira é uma propriedade intrínseca dos relata, nesse caso a altura. A Figura 9 apresenta os tipos de relações detalhadas nesse parágrafo.

Monadic Universals representam *Universals* que não são responsáveis por unir outras entidades. Em UFO-A, Monadic Universal possui uma especialização que é *Endurant Universal* e essa por sua vez possui duas especializações, *Substantial Universal* e *Moment Universal*. *Substantial Universal* é especializada por *Sortal Universals* e *Non-Sortal Universals*. A Figura 10 apresenta a parte da divisão dos *Monadic Universals*.

Sortal Universals são *Monadic Universals* que possuem ou herdam um princípio de identidade de outros *Sortals*. *Kinds* e *Subkinds* são *Rigid Sortals*, ou seja, são *Sortal Universals* cujos padrões de características são aplicados a todas as suas instâncias. Por sua vez, *Phases* e *Roles* são *AntiRigid Sortals*, um tipo de *Sortal Universal* cujos padrões

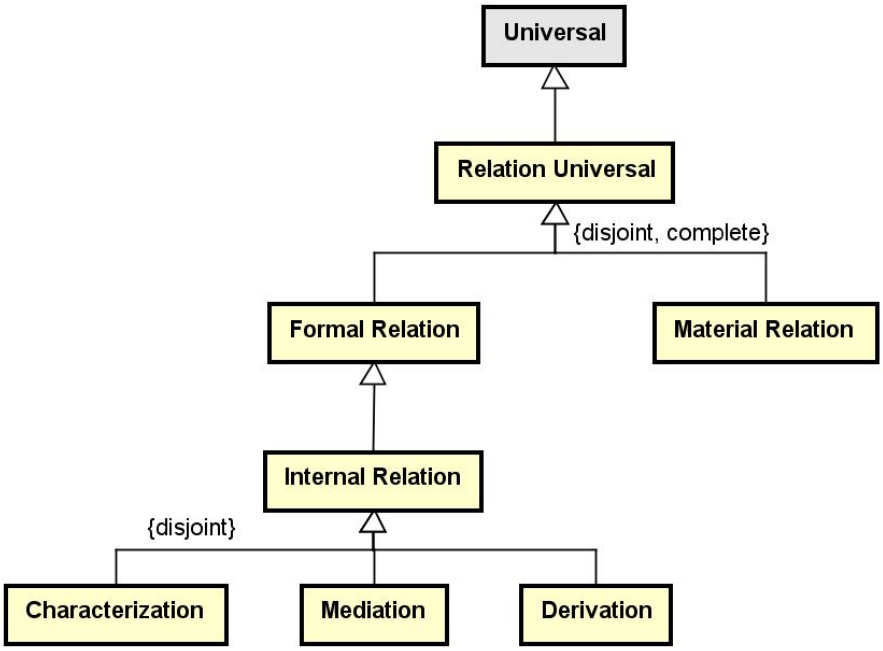

Figura 9 – Fragmento de UFO-A exibindo as especializações de *Relation Universal*.

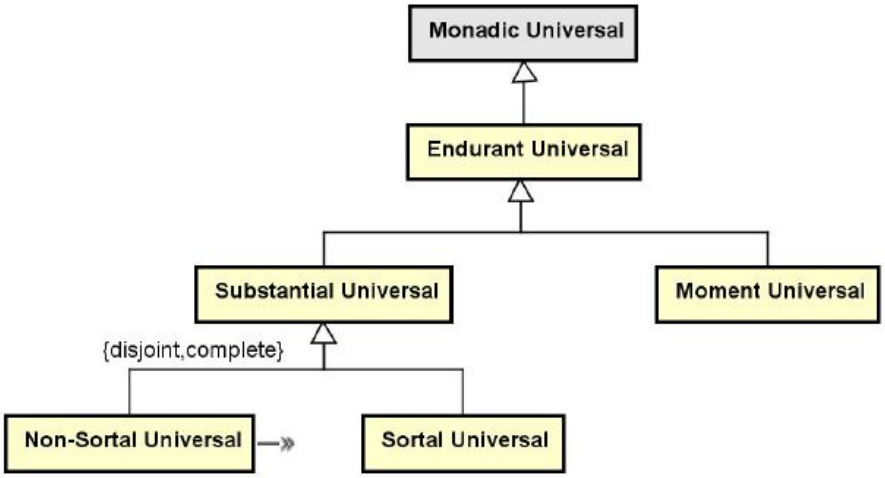

Figura 10 – Fragmento de UFO-A exibindo as especializações de Monadic Universal.

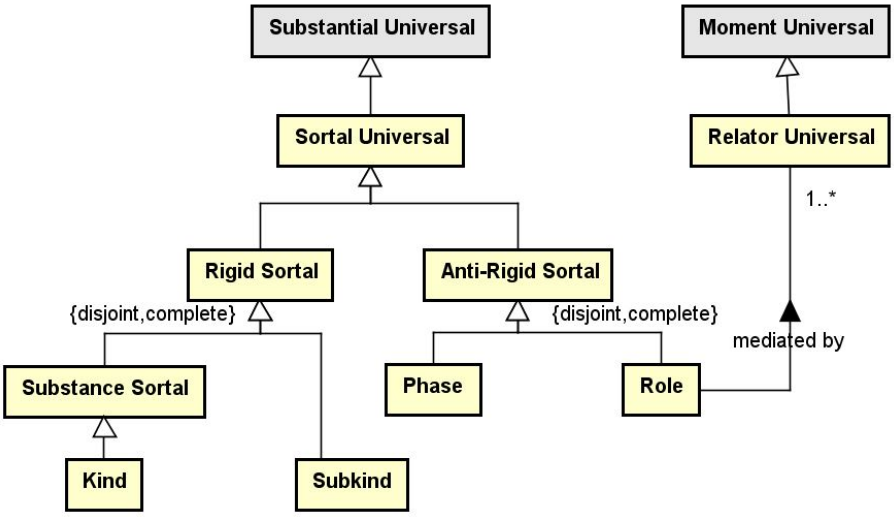

Figura 11 – Fragmento de UFO-A exibindo as especializações de Sortal Universal.

de características não se aplicam a todas as suas instâncias. Para facilitar o entendimento, podemos apresentar o seguinte exemplo: um homem é um *Subkind* de Pessoa (*Kind*), esse homem encontra-se na fase (*Phase*) adulta de sua vida e desempenha o papel (*Role*) de gerente geral em uma corporação, esse *Role* é mediado por um *Relator Universal* emprego entre homem e a corporação onde ele trabalha. A Figura 11 apresenta a estrutura dos *Sortal Universals*.

Mixin Universals são um tipo de *Non-Sortal Universal*, ou seja, não são capazes de prover uma identidade a suas instâncias. São divididos em *Rigid Mixins* e *Non-Rigid Mixin*. *Rigid Mixins* (*Category*) agregam propriedades essenciais e comuns a diferentes *Sortal Universals*, da mesma forma que os *Rigid Sortals*, seus padrões e características são aplicadas a todas as entidades com essa natureza. *Anti-Rigid Mixins* (*RoleMixin*) representam papéis que podem ser desempenhados por tipos diferentes e disjuntos entre si. Um exemplo comum de *Category* é **Mamífero**, categoria essa que é composta por diversos tipos (*Kinds*) diferentes de animais como pessoas, gatos e cachorros. Já com relação ao *RoleMixin*, um exemplo também bastante comum é o da entidade **Cliente**, que pode ser tanto um Cliente Pessoa-Fisica (que especializa do *Kind* Pessoa) quanto um Cliente Pessoa Jurídica (que especializa de um *Kind* Empresa). A Figura 12 apresenta a estrutura hierárquica dos *Non-Sortal Universals* mencionados acima.

2.3.3 UFO-B

UFO-B, é o fragmento de UFO que trata de *Events*, também chamados de *Perdurants*. Enquanto *Endurants* são *Concrete Individuals* que **existem no tempo**, no sentido de que eles existem através do tempo e são capazes de mudar sem perder sua identidade, *Events* são *Concrete Individuals* imutáveis, compostos de partes temporais e *acontecem no tempo*. *Events* podem ser *Atomic Events*, que não possuem partes, ou *Complex Events*,

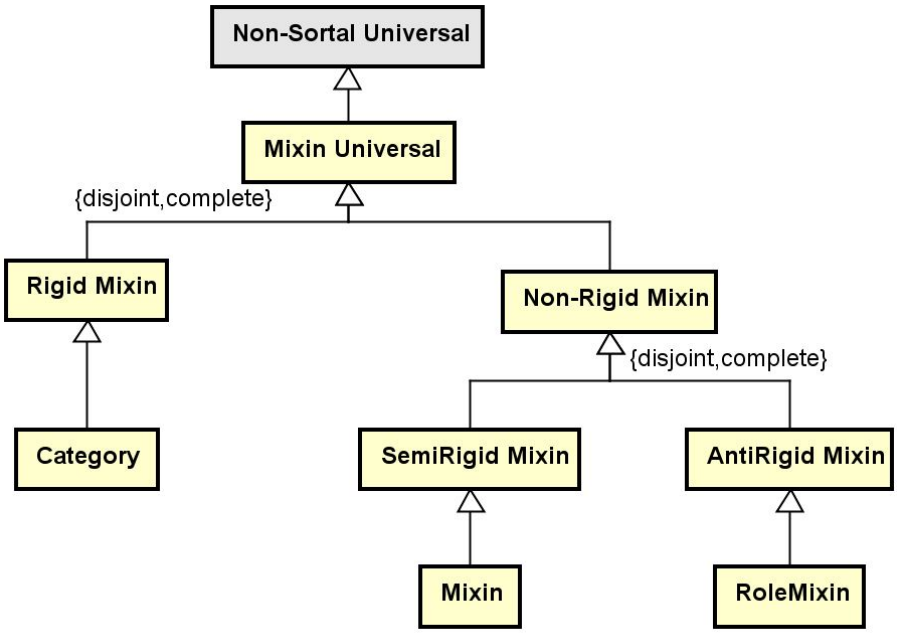

Figura 12 – Fragmento de UFO-A exibindo as especializações de Non-Sortal Universal.

compostos por dois ou mais *Events* (GUIZZARDI; FALBO; GUIZZARDI, 2008a). Como exemplos de *Events* podemos citar um jogo de futebol, uma festa de casamento ou a execução de um programa de computador.

Events são entidades existencialmente dependentes de seus *Participations*. No evento onde Robberto Baggio perde o ultimo pênalti na final da Copa do Mundo de 1994 temos como participantes, o próprio Baggio, Taffarel, o gol e a bola. O evento principal é composto pelas participações individuais de cada uma dessas entidades sendo que cada um desses participantes pode ser um *Event* por si só (complexo ou atômico) mas existencialmente dependente de um único *Substantial*.

Events também são capazes de alterar o estado das coisas, alterando uma situação da realidade de um *pre-state* para um *post-state*. A Figura 13 apresenta a entidade *Event* em UFO-B e suas características. As entidades que apresentam o *namespace* UFO-A pertencem originalmente à UFO-A e foram inseridas no modelo para melhorar o entendimento do mesmo, as demais entidades, sem *namespace*, são originais de UFO-B.

2.3.4 UFO-C

O primeiro conceito a ser explorado em UFO-C é o de *Agent*. *Agent* é um *Substantial* que pode ser especializado como um *Physical Agent*, uma pessoa, ou um *Social Agent*, uma organização. Inerente ao *Agent*, existe o *Intentional Moment*, um tipo de *Moment* que possui um tipo de intencionalidade (*Belief*, *Desire*, *Intention*) e uma *Proposition*, ou seja, um conteúdo proposicional. *Intentions* são, então, *Intentional Moments* que caracterizam alguma situação desejada pelo agente e que podem ser frustradas. Como um exemplo de

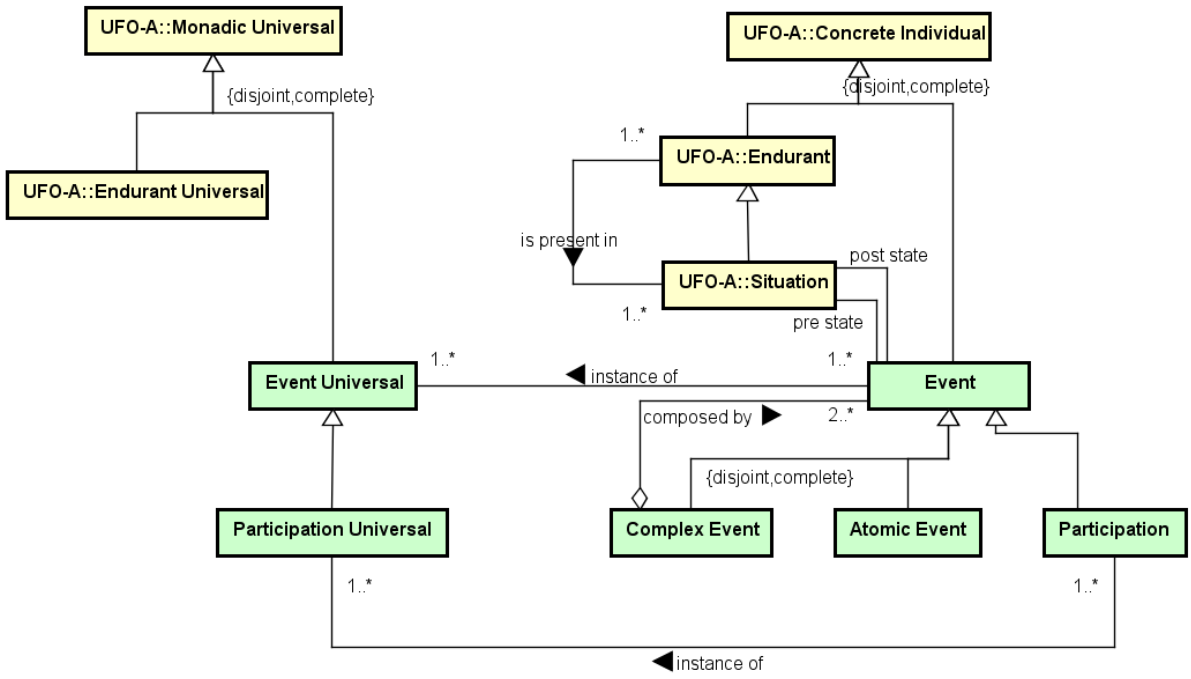

Figura 13 – Fragmento de UFO-B exibindo a entidade *Event* e seus relacionamentos.

Intention podemos citar: Gustavo tem a intenção de se formar na faculdade.

O conteúdo proposicional de uma intenção é, em UFO-C, um *Goal*. Poderíamos dizer então que o conteúdo proposicional do exemplo dado acima seria ser aprovado em todas as matérias do curso de graduação. *Desires* também podem ser realizados ou frustrados e expressam a vontade de um agente para com um estado das coisas na realidade, por exemplo, o desejo que uma pessoa tem de ficar rica. *Beliefs* por outro lado podem ser justificados por *Situations* na realidade. *Functions* são um tipo de *Dispositions*, que por sua vez, são *Intrinsic Moments* que representam capacidades exibidas por um *Individual*. *Functions* (*Dispositions* de forma geral) são instâncias de propriedades potencialmente realizáveis que se manifestam através da ocorrência de um *Event*, em contrapartida, a ocorrência do evento acaba por produzir uma certa *Situation* no mundo. A Figura 14 apresenta os conceitos de UFO-C explicados até então, ela e a figura 14 as classes que pertencem às outras partições de UFO apresentam os *namespaces* UFO-A e UFO-B quando demonstradas nos modelos.

Intentions também são responsáveis por fazer com que um *Agent* execute ações. *Actions* são eventos intencionais, ou seja, um tipo de *Event* que instanciam um *Plan* e são causados por uma *Intention* inerente a um *Agent*. Assim como *Events*, *Actions* também podem ser atômicos (*Atomic Action*) ou complexos (*Complex Action*). Nem toda participação de um agente é considerada uma ação, apenas as participações que possuem intencionalidade, dessa forma, dentre todas as entidades, apenas agentes são capazes de realizar ações. *Objects* participam de ações como *Resources*. Diferentemente de *Intentions*, que possuem um comprometimento por parte do agente, *Desires* não possuem

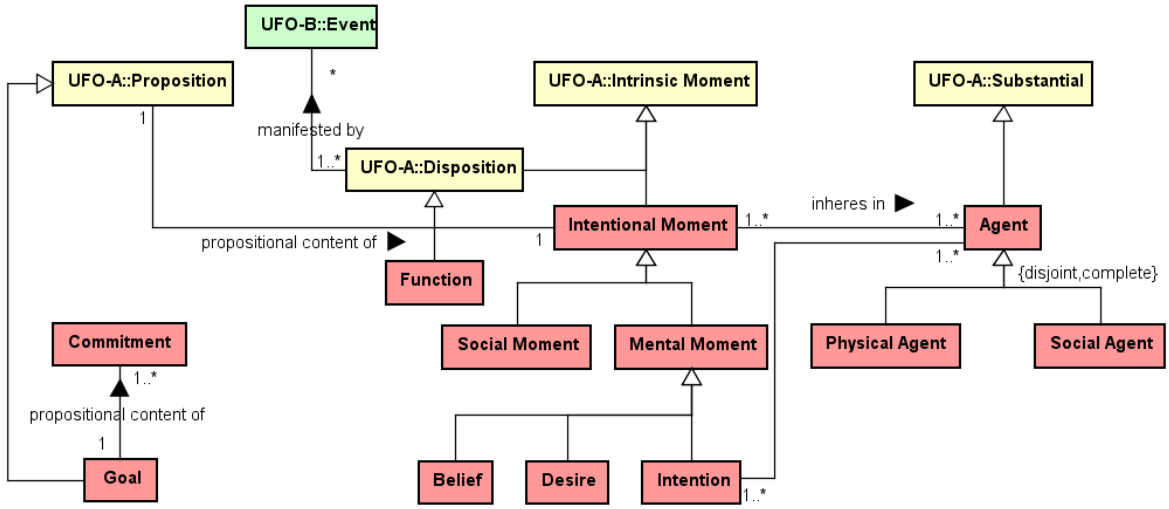

Figura 14 – Fragmento de UFO-C exibindo a entidade *Agent* e seus relacionamentos.

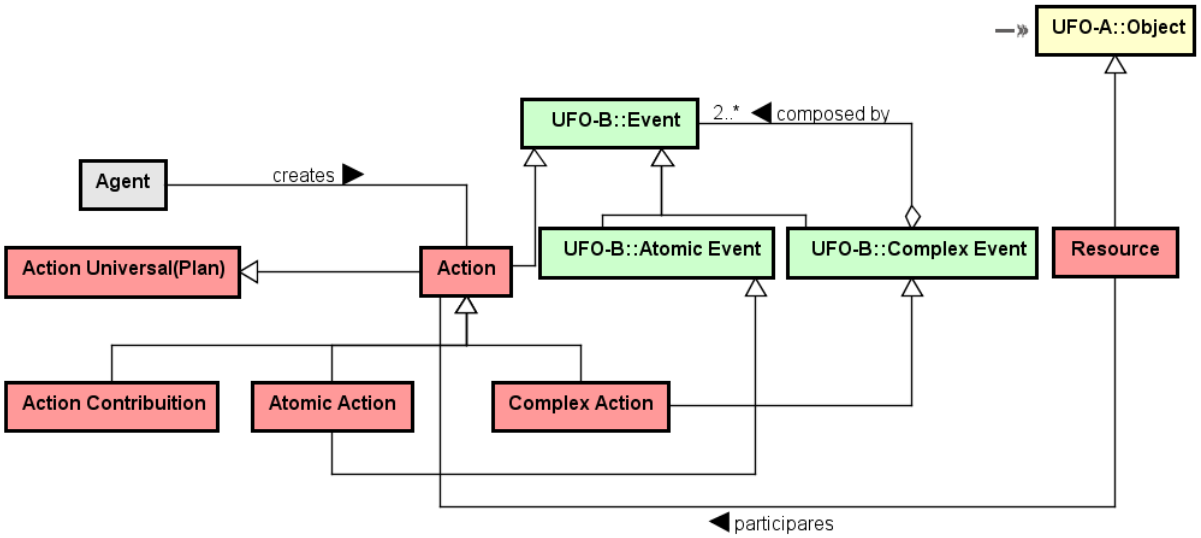

Figura 15 – Fragmento de UFO-C exibindo a entidade *Action* e seus relacionamentos.

um comprometimento, representando um desejo que um agente tem mas que sobre o qual ele não possui um comprometimento direto, por exemplo, quando um torcedor deseja que a seleção de seu país ganhe a próxima copa do mundo, ele tem o desejo de que tal situação se torne verdade mas não pode se comprometer a fazer com que tal situação ocorra de fato. A Figura 15 apresenta as entidades *Action*, *Intentions* e *Desires* e suas relações.

2.3.5 Ontologia de Artefatos de Software

Embora RRO tenha sido fundamentada em UFO, foram reutilizados conceitos da ontologia de artefatos de software proposta em (WANG et al., 2014b) e posteriormente estendida em (WANG et al., 2014a). Wang et al. propõem que software é um tipo especial de entidade capaz de sofrer mudanças enquanto mantém sua identidade. As mudanças que

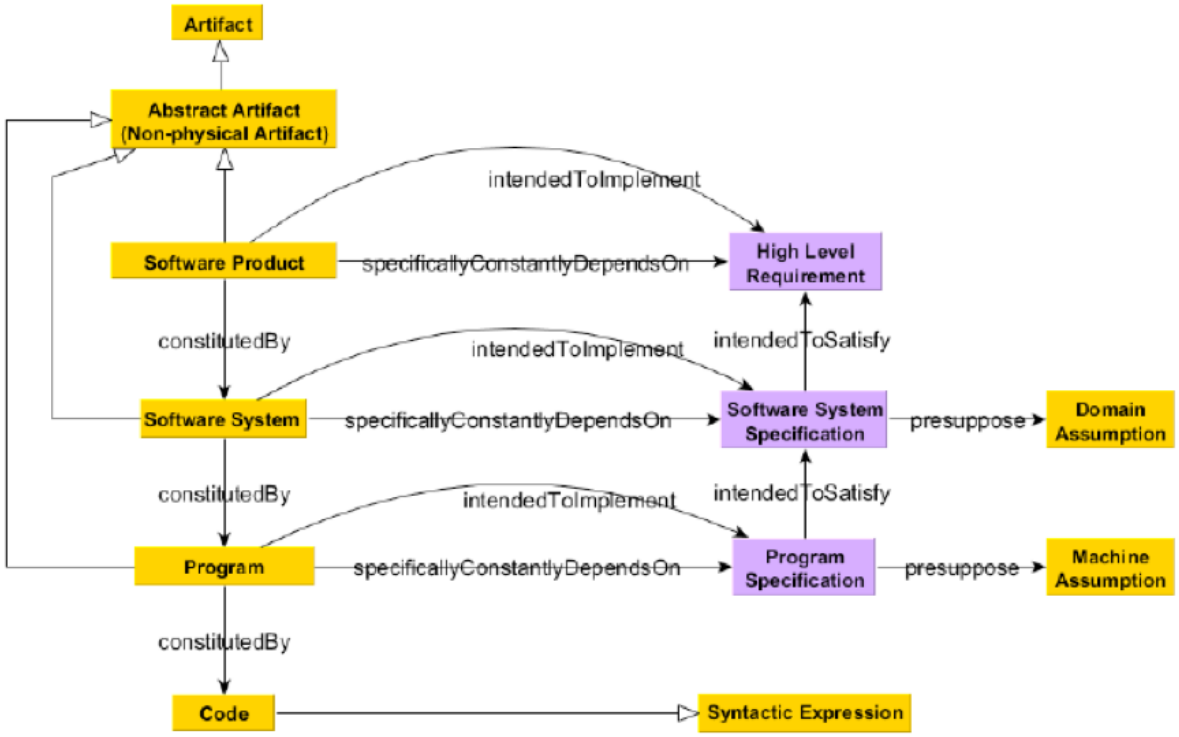

Figura 16 – Ontologia de Artefatos de Software proposta em (WANG et al., 2014b)

um software é capaz de sofrer, como a correção de um bug ou mesmo o lançamento de uma nova versão, são necessárias para a evolução natural do software. Essa capacidade de constante evolução é um dos motores que move a indústria de software como um todo. Quando esse pensamento é instanciado em um exemplo prático, o processo se torna mais transparente: por exemplo, o editor de planilhas Microsoft Excel possui vários lançamentos e ainda mais versões em seus 30 anos de existência, mas sempre mantendo sua identidade de Microsoft Excel.

Para tentar responder questões como "O que significa uma mudança em um Software?", "Qual a diferença entre um novo lançamento e uma nova versão" e para demonstrar as razões ontológicas pelas quais um software é capaz de mudar mantendo sua identidade, Wang et al. apresentam uma ontologia de software que é inspirada pela literatura da Engenharia de Requisitos. A Ontologia de Artefatos de Software, como é denominada pelos autores, faz distinção entre três artefatos: *Software Product* (produto de software), *Software System* (sistema de software) e *Program* (programa). A ontologia também lida com a diferença entre um programa visto como um artefato constituído por um código-fonte e um programa como um processo, rodando em uma máquina. A Figura 16 apresenta a ontologia de artefatos de software.

Code (código fonte) é uma sequência de instruções lógicas escritas em uma linguagem de programação. Essas instruções são expressões sintáticas e sequências de símbolos de forma que o principio de identidade de um código-fonte é definido pela ordem dos símbolos

na qual está escrito, ou seja, um código é idêntico a outro se e somente se os dois possuírem a mesma estrutura sintática e, portanto, qualquer mudança em um dos códigos, como por exemplo adicionar um comentário ou renomear uma variável resultará em um código diferente.

Já um programa, é um *Artifact* (artefato), constituído por um código-fonte e criado com o propósito imediato de produzir um comportamento específico dentro de uma máquina. Esse comportamento é especificado em uma *Program Specification*, que é única e estritamente necessária, ou seja, para que um *Program* possa existir, é necessário a existência de uma *Program Specification*, mesmo que ela exista somente na cabeça do programador. Como é um tipo de artefato, um programa tem sua identidade conectada diretamente com sua função principal, ou seja, dois programas P_1 e P_2 são idênticos se ambos executarem o mesmo comportamento em uma máquina computadorizada. O programa é ainda considerado uma entidade complexa devido à sua natureza: ele se comporta como um *Universal* já que suas características próprias são replicadas em todas as suas cópias, mas ele não é capaz de existir fora do espaço-tempo como números e outros artefatos abstratos. Na verdade, um programa não é capaz de preceder seu criador e é dependente historicamente de um ato intencional de batismo e, portanto, dependente de seu criador.

Diferentemente de um programa, o qual está relacionado com uma função secundária de um software, que é executar um comportamento específico dentro de uma máquina, *Software System* e *Software Product* são artefatos que estão relacionados com a função principal de um software, que é produzir um conjunto de efeitos no mundo real (ambiente externo à máquina onde o programa foi originalmente executado). Essa função é realizada em duas partes. A primeira acontece com o comportamento do computador que é obtido por meio da execução do programa e gera efeitos físicos na interface com a máquina (ex.: uma mensagem aparecendo na tela). A segunda, dados pressupostos adequados, acontece sobre o mundo real (ex.: uma pessoa lê a mensagem e executa uma ação).

Baseando-se na existência desses dois passos para a realização da função principal de um software, Wang et al. (2014b) definem um *Software System* como um artefato constituído por um programa e projetado para determinar um comportamento externo da máquina. Esse comportamento externo é especificado em um *Software System Specification*. Um *Software Product* é constituído por um *Software System* e projetado para determinar os efeitos específicos do comportamento gerado pela máquina no ambiente externo. Tais efeitos são especificados pelos requisitos de alto nível (*High Level Requirements*).

Com relação aos relacionamentos presentes na ontologia, eles são definidos da seguinte forma pelos autores:

- **constitutedBy**: é uma relação de dependência assimétrica, não-reflexiva que utiliza-

se da axiomatização de DOLCE.

- **specificallyConstantlyDependsOn**: se x depende especificamente e constantemente de y, então, necessariamente, toda vez que x estiver presente y deverá também estar presente. Essa relação também mantém sua fundamentação na axiomatização presente em DOLCE.
- **intendedToImplement**: essa relação é responsável por conectar um artefato à sua especificação como resultado de um ato de intencionalidade. No entanto, é preciso entender que a simples intenção de implementar não necessariamente implica que a implementação será a mais adequada ou mesmo correta.
- **intentedToSatisfy** e **presuppose**: essas relações são propostas para descrever a fórmula de Engenharia de Requisitos de Software proposta por Zave e Jackson (1997). Pressuposição é uma forma de dependência histórica de certos estados de conhecimento.

Por fim, é importante acrescentar que assim como RRO, a Ontologia de Artefatos de Software também é fundamentada em UFO e devido a isso o reuso dessa ontologia foi facilitado.

2.3.6 Uma Interpretação Ontológica de Requisitos

Guizzardi et al. (2014) propõem em seu trabalho uma interpretação ontológica de requisitos à luz de UFO. Nesta proposta, requisitos funcionais (RFs) e requisitos não-funcionais (RNFs) são vistos como objetivos (*Goal* em UFO), com a diferença primária de que RFs referem-se a funções (*Function* em UFO) e RNFs a qualidades (*Quality* em UFO). A ontologia também define uma classificação de Requisitos de acordo com sua especificidade. Essa classificação apresenta uma visão ortogonal sobre requisitos não-funcionais e funcionais e suas respectivas especificações, de forma que ambos podem ser vagos ou específicos dependendo de como forem formalizados e afastando a ideia de que um requisito funcional será sempre mais específico do que um requisito não-funcional.

Qualidades e funções são definidas em UFO como *Intrisic Moments*, entretanto qualidades são propriedades que se manifestam sempre que elas existem, já funções são um tipo de *Dispositions* que se manifestam somente por meio da execução de um evento. No trabalho, os autores avançam sobre o proposto pela ontologia CORE (vide Seção 2.5.1) e motivam a utilização de UFO para tratar do domínio de requisitos. A interpretação de requisitos como objetivos foi uma das bases para a criação de RRO e, consequentemente, dessa dissertação. Em RRO, nós importamos a distinção entre RFs e RNFs, mas não exploramos extensivamente o conteúdo da interpretação ontológica de requisitos.

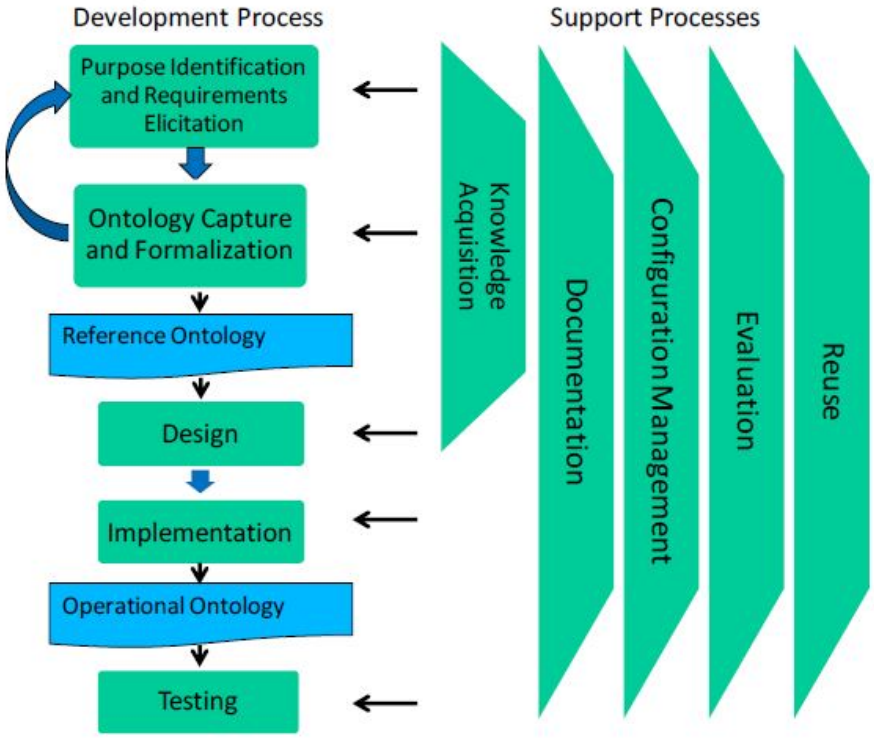

Figura 17 – Processo de Desenvolvimento do Método SABiO (FALBO, 2014)

Por fim, é possível dizer que RRO e a interpretação ontológica de requisitos funcionais e não-funcionais são visões complementares, de forma que representam diferentes aspectos da entidade requisito, enquanto a primeira está mais preocupada em definir ontologicamente o que é um requisito em *runtime* e sua relação com os conceitos que existem no domínio, a segunda preocupa-se em explicar, ontologicamente, o que são requisitos funcionais e não-funcionais, além de prover a visão ortogonal sobre a especificidade dos requisitos, previamente mencionada. Entendemos que RRO e a interpretação ontológica de requisitos funcionais e não-funcionais são ontologias perfeitamente compatíveis e poderiam ser reutilizadas em conjunto, por outras ontologias, caso fosse necessário.

2.4 SABiO - Uma Metodologia Sistemática para construção de Ontologias

Para construir RRO, foi utilizado o método SABiO (Systematic Approach to Build Ontologies) (FALBO, 2004; FALBO, 2014) pois o mesmo além de ser baseado em métodos e processos da engenharia de software clássica, já amplamente aceitos pela comunidade, é focado na construção de ontologias de domínio, encaixando-se perfeitamente para o que foi inicialmente planejado para a construção de RRO. SABiO é composto por cinco fases e cinco processos de apoio, como mostra a Figura 17:

- **Identificação de Proposta e Levantamento de Requisitos**: a primeira fase da metodologia é destinada a identificar o propósito e a utilização esperada para a ontologia a ser construída, nessa fase são definidas as questões de competência (GRÜNINGER; FOX, 1995) que a ontologia deverá responder.
- **Captura e Formalização da Ontologia**: na segunda fase do método, conceitos, relações, restrições e axiomas considerados importantes para a conceituação da ontologia devem ser identificados e organizados. Nessa fase também é caracterizada uma representação (através de uma linguagem formal de representação) da conceituação capturada pela ontologia. SABiO ainda sugere que modelos que utilizem linguagem gráfica sejam utilizados para facilitar a comunicação entre especialistas de domínio e o engenheiro de ontologias. Ao final dessa fase a ontologia de referência de domínio é obtida.
- **Design**: nessa fase, a especificação conceitual da ontologia de referência deve ser transformada em uma especificação de design para uma ontologia operacional a ser implementada. Essa especificação deve levar em consideração questões arquiteturais e tecnológicas da implementação.
- **Implementação**: assim como na Engenharia de Software tradicional, a fase de implementação de ontologias trata da codificação da ontologia para uma linguagem operacional escolhida durante a fase de design que seja capaz de ser interpretada por uma máquina.
- **Teste**: a fase de testes é a ultima fase proposta pela metodologia e consiste em verificar e validar de forma dinâmica e o objetiva o comportamento da ontologia operacional através de conjunto de casos de teste e comparar os resultados com o comportamento esperado baseando-se nas questões de competência. O método sugere que os testes sejam feitos inicialmente partindo das sub-ontologias que compõem a ontologia de domínio e subindo à medida que as sub-ontologias vão sendo integradas até que a ontologia final esteja devidamente integrada e possa ser testada funcionando como um todo.

A Figura 17 apresenta ainda cinco processos de suporte que são desenvolvidos em paralelo ao processo de desenvolvimento do método: Aquisição de Conhecimento, Documentação, Gerência de Configuração, Avaliação e Reuso.

2.5 Trabalhos Relacionados

Nessa seção serão apresentados alguns trabalhos sobre ontologias em engenharia de software que foram relevantes para a realização dessa pesquisa e consequentemente para a construção de RRO.

2.5.1 Core Ontology of Requirements

Em (JURETA; MYLOPOULOS; FAULKNER, 2008; JURETA; MYLOPOULOS; FAULKNER, 2009), Jureta et al. propõem uma ontologia de núcleo para requisitos de software chamada *Core Ontology of Requirements* (ou simplesmente CORE). Por definição, uma ontologia de núcleo é uma ontologia que se encontra entre uma ontologia de fundamentação e uma ontologia de domínio, não sendo um conjunto de categorias e tipos genéricos capazes de representar diversos tipos de domínios distintos como uma ontologia de fundamentação mas, também não limitada a um domínio específico como acontece nas ontologias de domínio. Para definir e exemplificar a diferença entre uma ontologia de núcleo de uma ontologia de domínio, podemos supor a existência de duas ontologias distintas: uma ontologia de testes de forma geral e uma outra ontologia que trata especificamente de teste de software. Enquanto a primeira trata de todos os tipos de testes de forma geral (inclusive teste de software) mas de maneira não específica, a segunda é focada diretamente no domínio de teste de software e trata de forma profunda suas especificidades, como por exemplo, a existência de conceitos específicos como teste de módulo e teste de unidade. A CORE é uma ontologia que baseia-se em estender o trabalho de Zave e Jackson (1997) no que diz respeito ao *requirements problem*, de forma que tem como objetivo principal estabelecer um novo critério para determinar se o processo de engenharia de requisitos foi completado de forma bem-sucedida.

A ontologia é fundamentada em DOLCE (MASOLO et al., 2003), devido ao fato de DOLCE ser uma ontologia de fundamentação que descreve ontologicamente as categorias de discurso presentes na linguagem natural humana. A CORE relaciona seus conceitos principais (*goal*, *plan*, *domain assumption* e *evaluation*) com os tipos básicos de discursos existentes na *speech act theory* e que são utilizados pelos *stakeholders* para se comunicar com os engenheiros de requisitos. Ao entender o tipo de discurso do *stakeholder* e a semântica por trás dele, o engenheiro de requisitos consegue associar esse discurso ao que o *stakeholder* realmente deseja. Dessa forma a *core ontology* estabelece uma metodologia baseada na semântica da comunicação entre *stakeholders* e engenheiros de requisitos, para o desenvolvimento do processo de engenharia de requisitos. A Figura 18 apresenta uma taxonomia dos conceitos que compõem a core ontology.

A CORE foi o trabalho com maior número de referências encontrado no mapeamento sistemático que foi realizado como base para a criação da ontologia proposta nesse trabalho (vide Seção 3.1), entretanto a ontologia não lida de forma direta com conceitos específicos do domínio de requisitos em tempo de execução

2.5.2 SRO - Software Requirements Ontology

A *Software Requirements Ontology* (SRO) é um ontologia de referência de domínio, proposta originalmente por Nardi & Falbo em (NARDI; FALBO, 2006) com o objetivo de

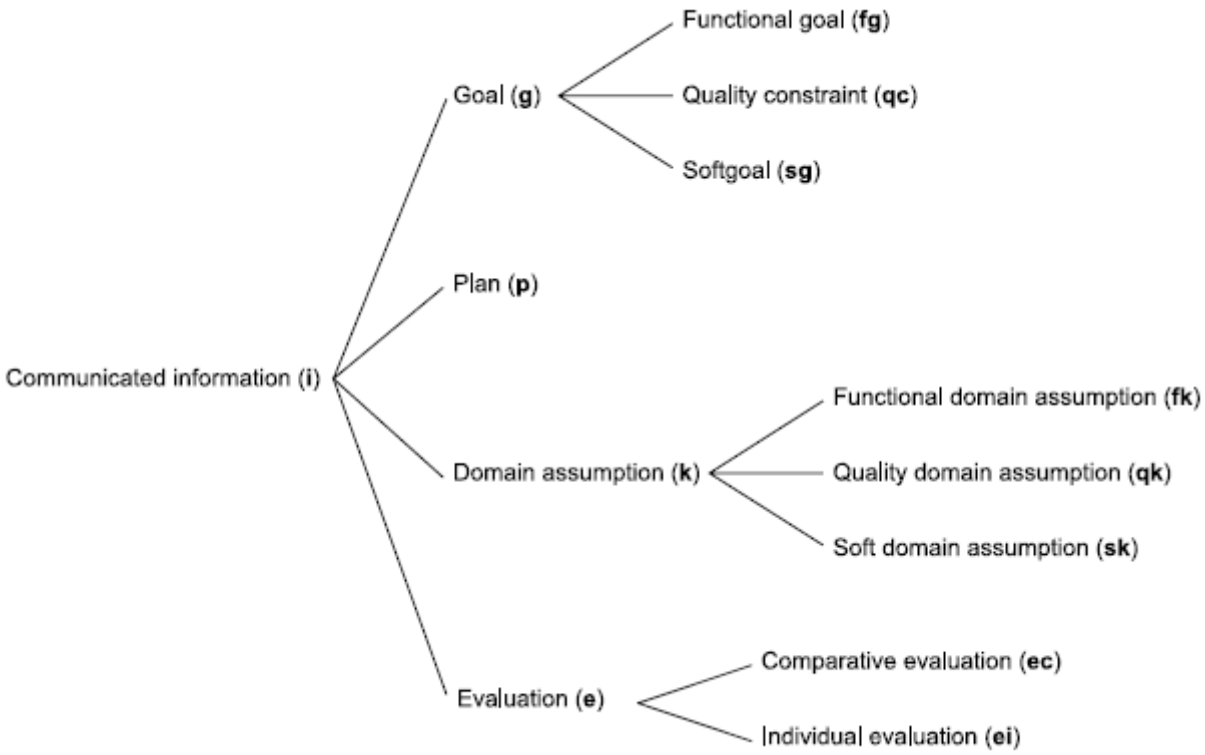

Figura 18 – Taxonomia dos Conceitos presentes na CORE (JURETA; MYLOPOULOS; FAULKNER, 2009)

formalizar o conhecimento no domínio de Engenharia de Requisitos e de prover um suporte semântico para a integração e o desenvolvimento de ferramentas. Em (NARDI; FALBO, 2008), SRO foi reformulada de forma a ser fundamentada em UFO. SRO foi construída utilizando o método SABiO e reutiliza conceitos de outras três ontologias: a *Software Process Ontology* (GUIZZARDI; FALBO; GUIZZARDI, 2008b), a *Software Configuration Management Ontology* (ARANTES; FALBO; GUIZZARDI, 2007) e a *Software Organization Ontology* [2]. A Figura 19 apresenta um fragmento da SRO que lida com a definição e taxonomia de um Requisito.

Comparada com RRO, SRO trata requisitos como artefatos do processo de Engenharia de Software, entretanto, em RRO foi feita uma distinção na qual Requisitos foram definidos como objetivos e suas descrições como os artefatos que estão presentes em SRO. Nós realizamos esta distinção para separar o requisito visto como objetivo (conteúdo proposicional de um *Moment* que é inerente a um *Agent* em UFO, nesse caso, o *stakeholder*) dos artefatos que o representam tanto em tempo de design como em tempo de execução. Essa separação é importante pois em tempo de execução, o requisito tem a natureza de um artefato que deve ser interpretado por uma máquina para ser utilizado e não de um objetivo de um agente (no caso, um *stakeholder*) que posteriormente será documentado e também e se tornará um artefato em tempo de design. Apesar das diferenças naturais

[2] Disponível em http://nemo.inf.ufes.br/projects/seon/

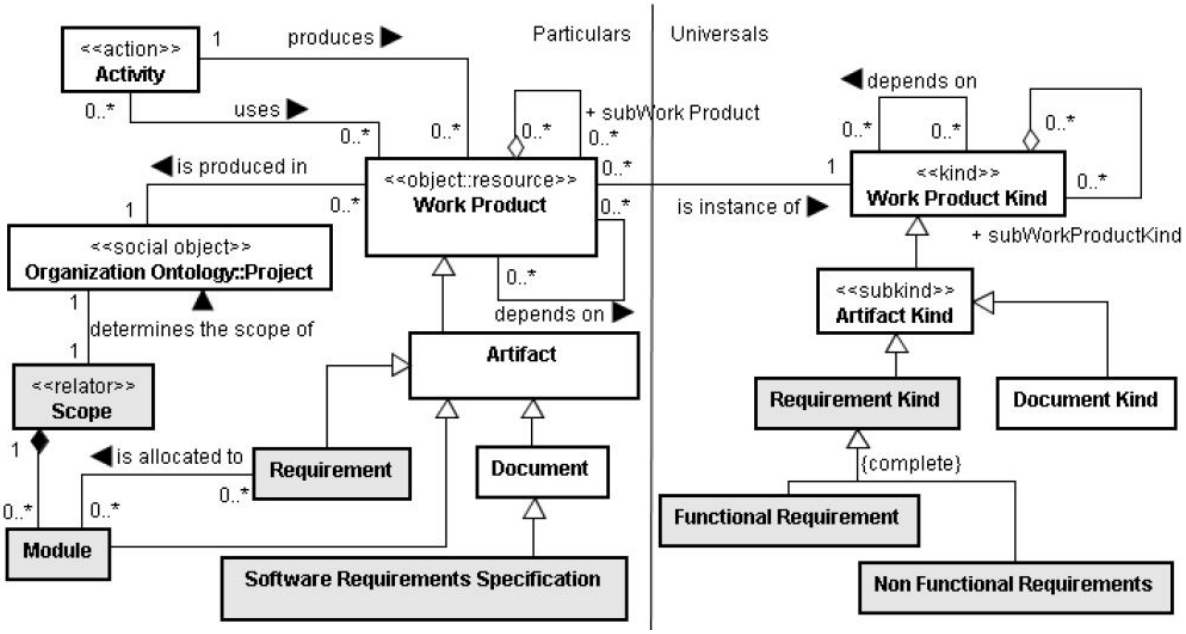

Figura 19 – Fragmento da Ontologia Software Requirements Ontology que lida com a definição de requisito e sua taxonomia (NARDI; FALBO, 2008)

relativas ao domínio e as distinções realizadas com relação a natureza de um requisito, RRO e SRO são fundamentadas em UFO e devido a isso compartilham conceituações a nível de fundamentação e poderiam ser reusadas em conjunto caso fosse necessário.

2.5.3 SEON - Software Engineering Ontology Network

No contexto da Engenharia de Software, vista como um macro domínio, ontologias são comumente utilizadas como ferramentas para tratar problemas relacionados com a gerência de conhecimento. Porém, quando diversas ontologias de sub-domínios da Engenharia de Software são criadas de forma isolada, problemas relacionados à integração de conhecimento podem permanecer. Entretanto, tentar representar todas as especificidades desta área em uma única, grande e complexa ontologia acabaria por tornar inviável seu processo de uso, manipulação e manutenção.

Devido a tais dificuldades, a *Software Engineering Ontology Network* (SEON) (RUI et al., 2016) tem como objetivos: prover uma rede bem fundamentada de ontologias de referência no contexto da Engenharia de Software, oferecer mecanismos para facilitar a construção e a integração de novas ontologias dentro deste domínio à rede, além de promover a integração de ontologias, mantendo uma semântica consistente entre os conceitos e relações que compõem a rede. Uma rede de ontologias é definida como uma coleção de ontologias que se relacionam através de uma variedade de relacionamentos, modularizações e dependências, enquanto uma ontologia em rede é um nó incluído na rede que compartilha conceitos e relações, através de reuso, com outras ontologias.

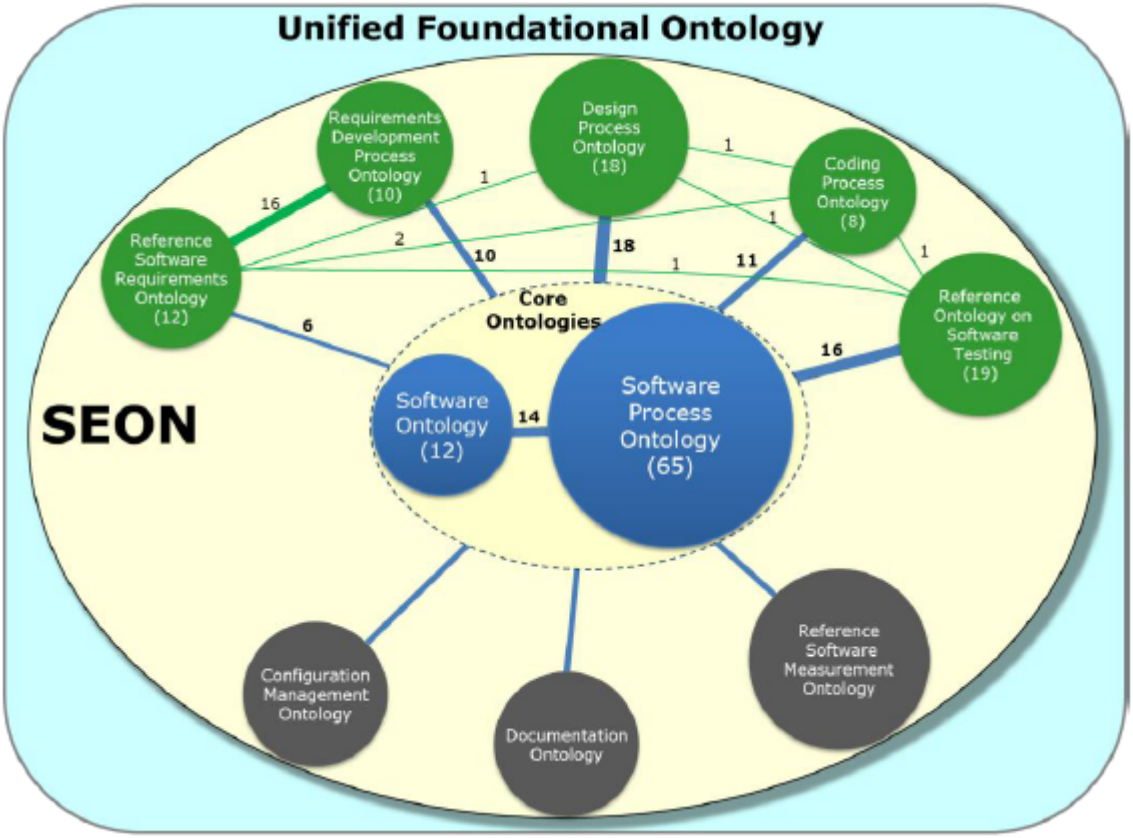

Figura 20 – Visão da SEON e das ontologias que a compõem (RUI et al., 2016)

.

A Figura 20 apresenta a visão de rede da SEON, onde cada um dos círculos representa uma ontologia componente e as linhas denotam os links entre as ontologias integradas de forma que, quanto maior o círculo, maior a quantidade de conceitos presentes na ontologia e quanto mais espessa a linha, maior a quantidade de links entre as ontologias. Os círculos azuis, presentes na parte central da SEON, representam ontologias de núcleo, enquanto círculos verdes, nas periferias, representam ontologias de domínio. Os círculos em cinza representam ontologias de domínio desenvolvidas através de UFO porém ainda não totalmente integradas à SEON.

Em comparação com as ontologias que compõem a SEON, RRO fornece uma visão complementar, visto que trata de um sub-domínio da engenharia de software, requisitos em tempo de execução, não abordado pela SEON. Outra característica que aproxima de RRO das ontologias presentes na SEON, é o fato de também ser fundamentada em UFO. Devido a esses fatores, é possível que em uma versão futura, RRO seja integrada à SEON.

3 Runtime Requirements Ontology

Neste capítulo,será apresentada a proposta principal desenvolvida nesse trabalho, a *Runtime Requirements Ontology* (RRO), uma ontologia de domínio sobre o uso de requisitos em tempo de execução. O capítulo está estruturado da seguinte forma: na Seção 3.1 a fase de captura de requisitos de RRO é apresentada. A Seção 3.2 apresenta a formalização e os modelos que compõem a RRO.

3.1 Levantamento de Requisitos

Da mesma forma que na Engenharia de Software convencional, na Engenharia de Ontologias os requisitos podem ser divididos em funcionais e não-funcionais (FALBO, 2004). SABiO sugere que os requisitos funcionais de uma ontologia devam ser escritos na forma de questões de competência (QCs) que a ontologia deverá ser capaz de responder (GRÜNINGER; FOX, 1995). QCs são responsáveis por delimitar o escopo da ontologia a ser construída (FALBO, 2014) e devem ser utilizadas como uma forma de avaliação da ontologia.

As QCs de RRO foram levantadas por meio de diversas reuniões semanais realizadas com pesquisadores especialistas nas áreas de requisitos em tempo de execução e ontologias. As reuniões tomaram os resultados encontrados no mapeamento sistemático sobre o uso de requisitos em tempo de execução, apresentado no Apêndice A, como base de informação consensual e definiram o escopo e os requisitos da ontologia que, por sua vez, foram traduzidos para as questões de competência apresentadas aqui utilizando uma estratégia *bottom-up*, ou seja, as questões mais simples foram modeladas primeiro e compostas para dar origem às questões mais complexas. Por fim, é importante destacar que o levantamento de requisitos e consequentemente das QCs de RRO foi iterativo, tendo sendo discutido ao longo das reuniões realizadas pelo grupo de pesquisadores, de forma que as QCs foram sendo refinadas até chegarmos nas seis questões listadas abaixo.

- **QC1:** O que é um programa em execução?
- **QC2:** Onde um programa é executado?
- **QC3:** O que pode ser observado na execução de um programa?
- **QC4:** Qual a relação entre um programa em execução e seus requisitos?
- **QC5:** O que é um requisito em tempo de execução?
- **QC6:** Quais os tipos de requisito em tempo de execução encontrados na literatura?

Como requisitos não-funcionais de RRO foi definido que:

- **NFR1:** RRO deve ser formalizada com base em uma ontologia de fundamentação.
- **NFR2:** RRO deve representar o conhecimento consensual presente na literatura referente ao domínio da ontologia.
- **NFR3:** RRO deve ser capaz de ser reutilizada em partes ou totalmente por qualquer outra ontologia de domínio fundamentada em UFO.

3.2 Formalização da Ontologia

De posse dos resultados do mapeamento sistemático e das QCs (requisitos) da ontologia, foi iniciada a segunda fase proposta por SABiO. Nessa fase, as principais entidades ontológicas e relações que compõem RRO foram definidas e representadas em modelos UML, de forma a compor a ontologia de referência de domínio. Como um dos objetivos principais de RRO é prover uma caracterização formal das principais entidades que compõem o domínio e suas restrições, algumas entidades de UFO foram apresentadas (prefixo `UFO` e cor amarela) e especializadas no modelo de RRO, de forma que a semântica dos elementos relativos ao domínio fica restrita pela semântica da entidade de UFO que foi especializada.

RRO é então apresentada dividida nas figuras 21 e 22 para diminuir a complexidade do modelo que representa a ontologia. Da mesma forma que os elementos de UFO que foram especializados, os conceitos importados de outras ontologias aparecem identificados em RRO por um prefixo que é o acrônimo do nome de sua ontologia original, dessa forma, o prefixo `NFR` (entidades com a cor vermelha) representa os conceitos referentes a interpretação ontológica de requisitos não-funcionais de Guizzardi et al. (2014) e o prefixo `OSA` (entidades com a cor roxa) representa os conceitos reutilizadas da ontologia de artefatos de software de Wang et al. (2014b). A figura 23 apresenta um eixo contínuo que demonstra a generalidade de RRO, quando comparada com outras ontologias existentes. Pela figura pode-se perceber de forma clara que RRO é considerada uma ontologia de domínio por ser mais específica que ontologias como UFO-C e a SPO, apresentadas no capitulo 2.

A Figura 21 explica a natureza de um requisito em tempo de execução e sua relação com requisitos em tempo de design. Assim como em (GUIZZARDI et al., 2014), um requisito é definido como um *Goal* em UFO, ou seja, o conteúdo proposicional de uma intenção inerente a um agente (no domínio de engenharia de requisitos, um *stakeholder*). Quando esse requisito, que anteriormente era apenas um objetivo (relacionado a uma intenção), é documentado em uma especificação de requisitos (como resultado de um processo de Engenharia de Requisitos) é criado então um *Requirement Artifact* que descreve

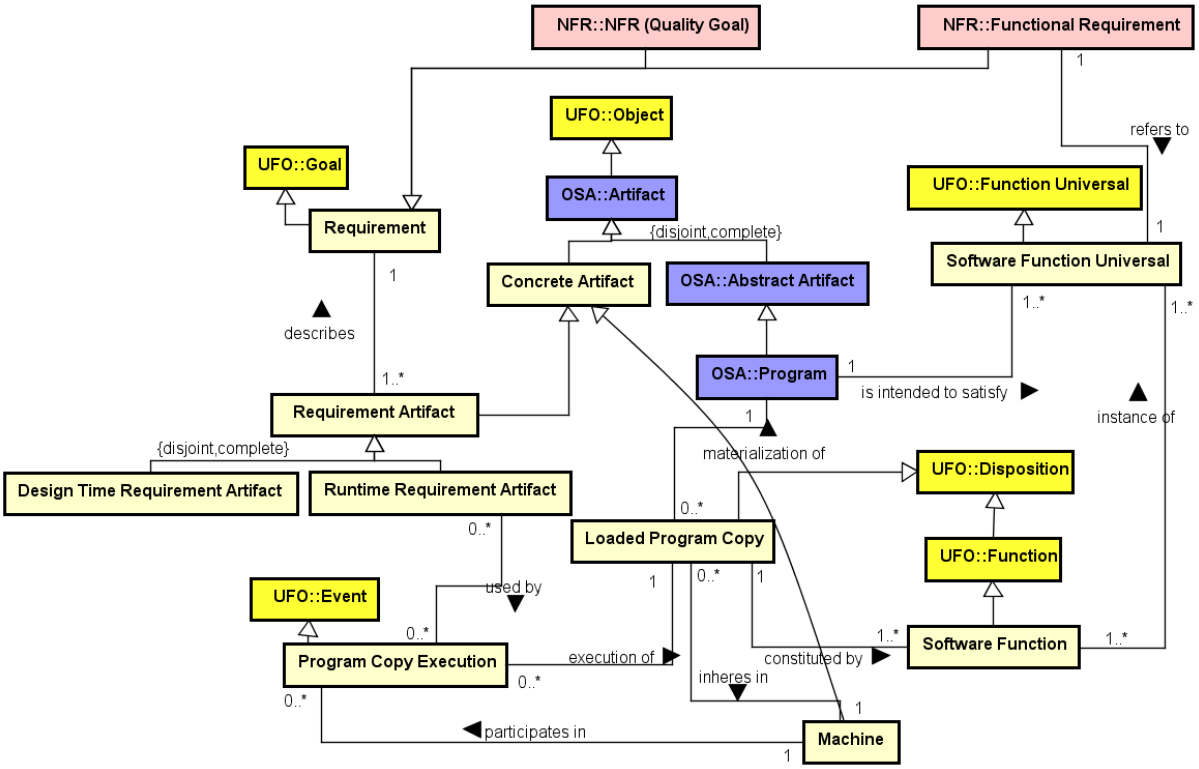

Figura 21 – Fragmento de RRO descrevendo a natureza de requisitos em tempo de execução.

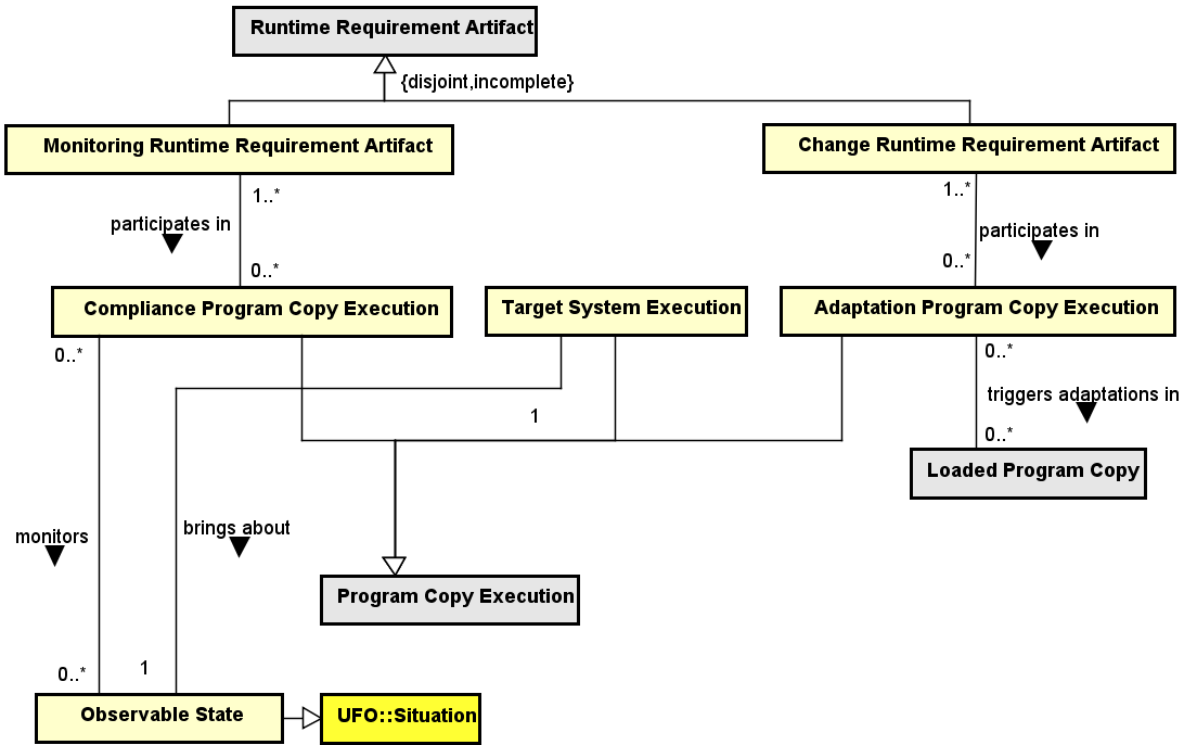

Figura 22 – Fragmento de RRO que demonstra a relação entre requisitos e programas durante o tempo de execução.

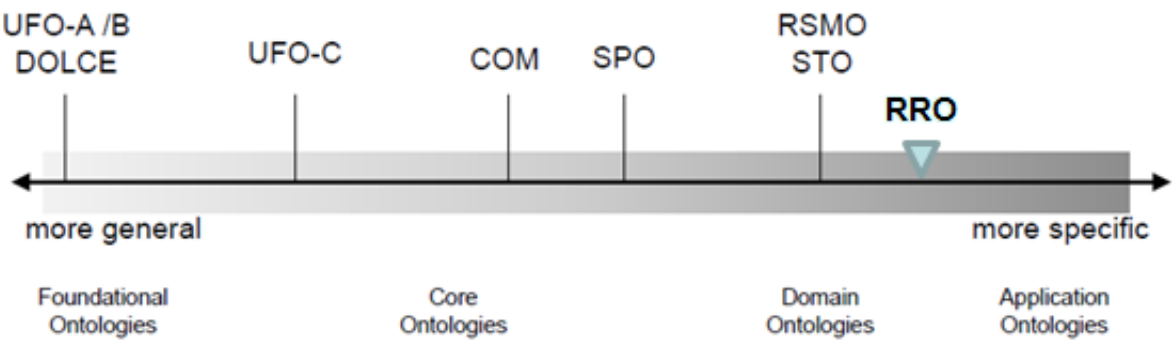

Figura 23 – Comparação de RRO, segundo sua generalização, com outras ontologias.

o *Requirement*. Um *Requirement Artifact* é um *Concrete Artifact*, um objeto físico criado por seres humanos e, dessa forma, classificado como um objeto (*Object* em UFO) com um determinado propósito de existência, que possui dois sub-tipos distintos: *Design Time Requirement Artifact*, que representa os artefatos de requisitos existentes durante o processo de desenvolvimento de software e *Runtime Requirement Artifiact*, que representa os artefatos que descrevem requisitos durante o tempo de execução de um software.

A descrição de um requisito em um ou vários artefatos pode ocorrer nas mais variadas formas, como um texto em linguagem natural escrito na especificação de requisitos de um sistema, que é criado com o intuito de ser utilizado por humanos e que sobrevive por todo o projeto de software, desde a fase de levantamento de requisitos até a fase testes, representado em RRO como *Design Time Artifact*. Pode ocorrer também em um arquivo lógico, escrito em uma linguagem computacional (ex.: um arquivo XML) e capaz de ser processado por um software. Esse último, se concebido com o intuito de ser utilizado durante o tempo de execução de um programa se torna um *Runtime Requirement Artifact*. O *Runtime Requirement Artifact* é responsável por descrever de alguma forma passível de ser entendida por máquina, durante o tempo de execução de um programa, um *Requirement* que representa o desejo de um *stakeholder*.

Com a diferença primária entre os dois tipos de artefatos de requisitos definida, é importante mencionar que embora grande parte da literatura tradicional sobre engenharia de requisitos trata do requisito em tempo de desenvolvimento do software e não faça uma distinção explicita entre tempo de execução e tempo de desenvolvimento, tal distinção existe e é amplamente aceita dentro da literatura de requisitos em tempo de execução, como pode ser visto em (DALPIAZ et al., 2013), (SOUZA et al., 2011), (WHITTLE et al., 2010), (BORGIDA et al., 2013), (BENCOMO et al., 2010b) e portanto, deve ser refletida na ontologia. A distinção entre requisito em tempo de execução e em tempo de desenvolvimento também é importante porque demonstra que um objetivo do *stakeholder* pode dar origem a dois artefatos distintos, que existem e são utilizados em duas fases, também distintas, do ciclo de vida do software.

Com relação ao *stakeholder*, é importante destacar que o mesmo existe e que o requisito é de fato o conteúdo proposicional de sua intenção, contudo, foi convencionado não representar sua existência em RRO pois o mesmo não é vital para o entendimento do domínio.

Voltando ao conceito de requisito como um tipo de *Goal*, é amplamente aceito na literatura que requisitos são classificados em requisitos não-funcionais e requisitos funcionais, sendo que o primeiro refere-se normalmente a uma qualidade e o último a uma funcionalidade. Tais definições são ontologicamente apresentadas e explicadas na ontologia NFR (*Non-Functional Requirements*) criada por Guizzardi et al. (2014) e, dessa forma, reusados em RRO, trazendo consigo toda a carga conceitual de suas definições que foram apresentadas em sua ontologia original (vide Seção 2.3.6). Requisitos funcionais referem-se então a um *Software Function Universal*, uma entidade que descreve características que são comuns a todos as *Software Fuction* (*Individual*) que são suas instâncias. No contexto de RRO, a classificação de requisitos como *design-time* ou *runtime* é considerado ortogonal à classificação que os define como funcionais ou não funcionais, de forma que é possível a existência de um *Runtime Requirement Artifact* que representa um requisito não-funcional.

Program, segundo Wang et al. (2014b), é um *Abstract Artifact* constituído por um código escrito especificamente para um meio em uma máquina (ex.: Microsoft Excel para MacOS em um Macbook). Para que esse programa exista em tempo de execução um individuo deve possuir uma cópia e executá-la. Irmak (2013) define tal cópia como *Dispositions* de componentes do computador (ex.: o disco rígido do Macbook). Ele então descreve a execução de um programa como um *Event* que representa a manifestação física de tais *Dispositions*. Um *Program* destina-se a satisfazer um conjunto de *Software Function Universal* que podem ser considerados como especificações de requisitos abstratas para esse *Program*.

No domínio de requisitos em tempo de execução (vide Seção 2.1), requisitos são utilizados tanto para monitorar (o que pode ser feito por meio da observação de eventos como menciona Irmak) tanto como para adaptar (acarretar mudança em uma situação em um estado futuro) o programa. Nesse caso particular, nem os eventos (que por definição são imutáveis) ou a cópia do programa no disco rígido (que não está executando) são interessantes. Sendo assim, em RRO um *Loaded Pogram Copy* é uma materialização de um *Program* inerente a uma *Machine*, ou seja, uma cópia do Microsoft Excel carregada na memória principal do MacBook pelo MacOS.

Loaded Program Copy é uma *Disposition* complexa, constituída por uma ou mais *Software Functions* que são, por sua vez, instâncias de *Software Function Universals* (de uma forma mais simplista, tipos de funcionalidades). Quando o processo de desenvolvimento de software é executado corretamente, as funções que compõem o programa carregado em memória são instâncias exatas das funcionalidades que o programa é pressuposto a

realizar. Além disso, um *Loaded Program Copy*, como um tipo de *Disposition*, é um tipo de *Endurant* capaz de mudar enquanto mantém sua identidade.

Program Copy Execution é um *Event* no qual uma máquina participa. RRO define então a característica principal que distingue um *Runtime Requirement Artifact* de sua contra-parte em *design-time*: eles podem ser utilizados pelo *Program Copy Execution* em tempo de execução.

O segundo fragmento de RRO mostrado na Figura 22 descreve a relação existente entre requisitos e programas durante o tempo de execução. *Runtime Requirement Artifact* é especializado em dois subtipos: *Monitoring Runtime Requirement Artifact* define os critérios para verificar se um requisito está sendo satisfeito ou não durante o tempo de execução do programa; analogamente, *Change Runtime Requirement Artifact* é responsável por especificar adaptações no comportamento do sistema, de forma que este continue a realizar suas funções.

Esse *generalization set* é definido como disjunto (*disjoint*), pois de acordo com os resultados do mapeamento sistemático realizado as duas entidades são sempre distintas, não tendo sido encontrado um conceito específico que representasse requisitos de monitoramento e adaptação ao mesmo tempo. No caso da anotação *incomplete*, ela define a possibilidade da existência de outros tipos de requisitos em tempo de execução não tratados em RRO. As propostas de RRT encontradas no mapeamento sistemático e classificadas como *Monitoring Requirements* apresentaram apenas *Monitoring Runtime Requirements Artifacts* em suas metodologias e modelos, já as classificadas como *Change Requirements* apresentaram os dois tipos de *Runtime Requirement Artifacts* demonstrados aqui. A presença dos dois tipos de *Runtime Requirement Artifacts* nas propostas relacionadas a adaptação de software faz todo sentido, pois para que haja a mudança/adaptação em um software em execução, é necessário que suas características, indicadores e variáveis principais sejam constantemente monitorados de forma a detectar possíveis desvios de comportamento (por parte do software) que devam ser reconfigurados.

Os dois tipos de *Runtime Requirement Artifact* mencionados anteriormente participam como recursos, no sentido de *Resource* existente em UFO, em dois eventos importantes: *Compliance Program Copy Execution* e *Adaptation Program Copy Execution*. O *generalization set* composto por essas duas entidades e pelo *Target System Execution*, entidade que representa o programa principal, responsável executar as funções que estão sendo monitoradas, é definido em RRO como não-disjunto, isso significa que um único programa pode executar as três funções ao mesmo tempo, por exemplo, um software complexo, que executa várias funções, pode ser responsável por executar uma funcionalidade específica, tida como principal ou prioritária, enquanto monitora e dispara adaptações que serão aplicadas em si próprio.

Uma *Program Copy Execution* (a execução de um programa carregado em memória,

seja ele o software responsável pela funcionalidade principal descrita nos *Requirements Artifacts* e representado no modelo de RRO pelo *Target System Execution*, ou softwares adjacentes utilizados para monitoramento/disparo de adaptações) traz à tona um tipo particular de situação (*Situation*, no sentido de UFO e descrita mais detalhadamente em (GUIZZARDI et al., 2013)) denominada em RRO como um *Observable State*. Conforme explicado na Seção 2.3.1, uma *Situation* é uma porção particular da realidade que pode ser entendida completa, que pode ser caracterizada pela presença de objetos, seus *Moments* intrínsecos e relacionais e pelos valores que as qualidades (*Quality* em UFO) desse objetos assumem.

Um *Observable State* é, então, uma *Situation* que envolve as qualidades da máquina (*Machine*) à qual o *Loaded Program Copy* é inerente durante a execução de um programa. Todo o processo de execução que acontece na máquina, seus estados anteriores e posteriores ao evento de execução, os recursos utilizados e até mesmo as interações de um usuário que afetam a execução do programa compõem esse **estado observável**. Podemos então descrever o processo acontece em tempo de execução dentro da máquina (memória principal + processador + memória secundária) com o seguinte exemplo: o *Compliance Program Copy Execution* monitora o *Observable State* para verificar se o estado do software está de acordo com o que está especificado nos *Monitoring Runtime Requirement Artifacts*; caso isso não seja verdade e os requisitos de monitoramento em tempo de execução não estejam sendo satisfeitos, o *Adaptation Program Copy Execution* dispara mudanças, por exemplo, o relaxamento de valores de indicadores propostos nos requisitos, no *Loaded Program Copy*, seguindo o que está escrito na especificação do *Change Runtime Requirement Artifact*.

Finalmente, considerando os modelos propostos nas figuras 21 e 22 é importante observar que:

- As especializações de *Program Copy Execution* (*Compliance Program Copy Execution* e *Adaptation Program Copy Execution*) são obviamente execuções de algum tipo de especialização de *Loaded Program Copy* (ex.: *Compliance Loaded Program Copy*) que, por sua vez, são materializações de um tipo de programa (ex.: *Compliance Program*). Nesse sentido, as especializações de *Program* e *Loaded Program Copy* não são explicitamente demonstradas no modelo de RRO por razões de simplificação do mesmo. Um usuário da ontologia pode derivar e perceber a existência de tais entidades de forma direta.

- *Compliance Program Copy Execution* e *Adaptation Program Copy Execution* são explicitamente apresentados no modelo de RRO como entidades externas ao *Program Copy Execution* que representa a execução do programa principal. No entanto, sua existência pode ocorrer na forma de subprogramas do programa principal. Esse tipo de distinção foi considerada como fora do escopo de RRO, pois tal diferenciação não

agregaria nenhum benefício ao modelo.

- *Compliance Program Copy Execution* e *Adaptation Program Copy Execution*, quando em execução, também são passíveis de serem monitorados/modificados por outras instâncias de *Compliance/Adaptation Program Copy Execution* formando, assim, hierarquias de controladores de monitoramento/adaptação de requisitos, por exemplo um *Adaptation Program Copy Execution* que atue como *Target System* para um *Compliance Program Copy Execution Externo* é um cenário possível e levado em consideração por RRO.

Para um melhor entendimento dos conceitos e relações propostos na ontologia, RRO foi instanciada utilizando dois exemplos presentes na literatura sobre requisitos em tempo de execução. As Tabelas 3 e 4 presentes no Capítulo 4 ilustram os resultados das instanciações.

Por fim, é importante destacar que apenas a primeira e a segunda fase de SABiO foram realizadas pois o objetivo principal sempre foi o de obter a ontologia de referência de domínio (produto final dessas duas fases segundo) demonstrada aqui. As fases subsequentes não foram realizadas visto que elas tem como objetivo a criação de uma ontologia operacional, o que fugia do escopo desse trabalho.

4 Avaliação da Ontologia

Avaliação de ontologias é a atividade que analisa a qualidade técnica de uma ontologia baseado em parâmetros de referência. Esse capítulo apresenta os métodos que foram utilizados para avaliar RRO e demonstrar que a mesma atende a seus requisitos funcionais e não-funcionais sendo, assim, uma ontologia de referência de domínio funcional.

SABiO propõe que o processo de avaliação de uma ontologia de referência de domínio compreenda duas perspectivas principais: (i) verificação da ontologia, que visa garantir que a mesma seja construída de forma que os artefatos produzidos atinjam o que foi especificado; (ii) validação da ontologia, que visa garantir que a ontologia certa seja construída, em outras palavras, que ela sirva ao seu propósito principal.

Assim, este capítulo é então dividido conforme as duas perspectivas principais citadas. A Seção 4.1 apresenta a verificação das questões de competência, além de uma verificação por análise sintática realizada por meio de uma ferramenta para construção de ontologias. A Seção 4.2 fecha o capítulo, apresentando duas formas de validação realizadas em RRO: validação por instanciação e validação por simulação.

4.1 Verificação da Ontologia

Verificar uma ontologia significa avaliar se a mesma foi construída corretamente (FALBO, 2014). Para verificar RRO foram adotadas duas estratégias: a primeira consiste em responder as questões de competência que foram propostas durante a fase de levantamento de requisitos da ontologia. A segunda consiste em verificar a ontologia através de uma ferramenta de modelagem

4.1.1 Verificação por Questões de Competência

SABiO sugere que o processo de verificação seja baseado nos requisitos da ontologia, ou seja, a ontologia deve ser capaz de responder as Questões de Competência (QCs) que foram propostas. O método sugere que seja construída uma tabela na qual as QCs são respondidas.

A Tabela 1 ilustra os resultados da verificação por QCs que foi realizado em RRO de forma a verificar se a ontologia criada corresponde à que foi especificada em seus requisitos. A tabela de verificação pode ainda ser utilizada como uma ferramenta de rastreabilidade da ontologia, dando suporte à gerencia de configuração.

Tabela 1 – Verificação de RRO através de suas QCs.

QC	**Concepts e *Relations***
QC1 - O que é um programa em execução?	Loaded Program Copy é um *subtype of* Disposition e um *materialization of* Program. Program Copy Execution é *subtype of* Event e um *execution of* Loaded Program Copy.
QC2 - Onde um programa é executado?	Loaded Program Copy *inheres in* Machine, que é *subtype of* Artifact. Machine *participates in* Program Copy Execution.
QC3 - O que pode ser observado na execução de um programa?	Observable State é *subtype of* Situation. Program Copy Execution *brings about* Observable State. Compliance Program Copy Execution *monitors* Observable State.
QC4 - Qual a relação entre um programa em execução e seus requisitos?	Loaded Program Copy é *constituted by* Software Functions, o qual são *instances of* Software Function Universals. Loaded Program Copy é uma *materialization of* Program, que *intends to fulfill* Software Function Universals. Functional Requirement é um *subtype of* Requirement e *refers to* Software Function Universals.
QC5 - O que é um requisito em tempo de execução?	Runtime Requirement Artifact é *subtype of* Requirement Artifact, o qual é *subtype of* Artifact, o qual é *subtype of* Object. Runtime Requirement Artifact é *used by* uma Program Copy Execution.
QC6 - Quais os tipos de requisito em tempo de execução encontrados na literatura?	Monitoring Runtime Requirement Artifact é *subtype of* Runtime Requirement Artifact e é *used by* a Compliance Program Copy Execution para *monitor* Observable States. Change Runtime Requirement Artifact é *subtype of* Runtime Requirement Artifact e é *used by* uma Adaptation Program Copy Execution para *change* o Loaded Program Copy.

4.1.2 Análise Sintática e Anti-Patterns

Para realizar uma segunda forma de verificação, RRO foi reestruturada em OntoUML (GUIZZARDI, 2005), um perfil do diagrama de classes de UML 2.0 ontologicamente fundamentado em UFO em que as entidades ontológicas são refletidas como esteriótipos UML. Essa reestruturação foi realizada pois OntoUML fornece um conjunto de ferramentas que possibilitam a realização de diversas formas de avaliação da ontologia.

No entanto, é importante destacar que, até o momento da escrita desse trabalho, não foi encontrada na literatura uma metodologia sistemática para a conversão de uma ontologia construída e fundamentada a partir de entidades de UFO-B e UFO-C para OntoUML, que tem sua teoria de tipos originalmente baseada em entidades ontológicas presentes em UFO-A. Além disso, tal processo de conversão pode gerar perdas na expressividade de RRO. Devido a isso, as seguintes heurísticas foram adotadas com o objetivo de assegurar que a versão alternativa de RRO, construída em OntoUML, representasse o domínio de requisitos em tempo de execução de forma fidedigna, assim como a versão original construída a partir de UFO-B e UFO-C:

- Ontologias já existentes criadas em OntoUML e de domínios relacionados ao de RRO foram consultadas de modo a encontrar entidades comuns às duas ontologias e mapeá-las em OntoUML de acordo com a ontologia pesquisada.
- Várias entidades de UFO-B e UFO-C são generalizações diretas de entidades de UFO-A. Essas generalizações foram levadas em consideração para reescrever o modelo de RRO em ontoUML.
- Foram levados em consideração a própria estrutura de OntoUML e seus conceitos principais (ex: as diferenças entre rigidez e anti-rigidez, sortais e não-sortais) além de conhecimento sobre o domínio de requisitos em tempo de execução para evitar discrepâncias na semântica representada pelo dois modelos, sempre tentando reduzir ao mínimo a perda de representatividade.

Além das diretrizes já mencionadas acima, é importante destacar que estrutura da linguagem OntoUML obriga, através dos relacionamentos e entidades da linguagem, a existência uma completude no modelo para que esse esteja sinteticamente correto.

A Figura 24 apresenta RRO reconstruída em OntoUML, denominada RRO-OntoUML de forma que se torna possível realizar a análise sintática do modelo utilizando a ferramenta de modelagem OLED (OntoUML Lightweigth Editor) (GUERSON et al., 2015).

Com o modelo pronto, o próximo passo consiste em analisar a presença de anti-padrões (APs) existentes no modelo. Anti-padrões são padrões de modelagem descobertos e

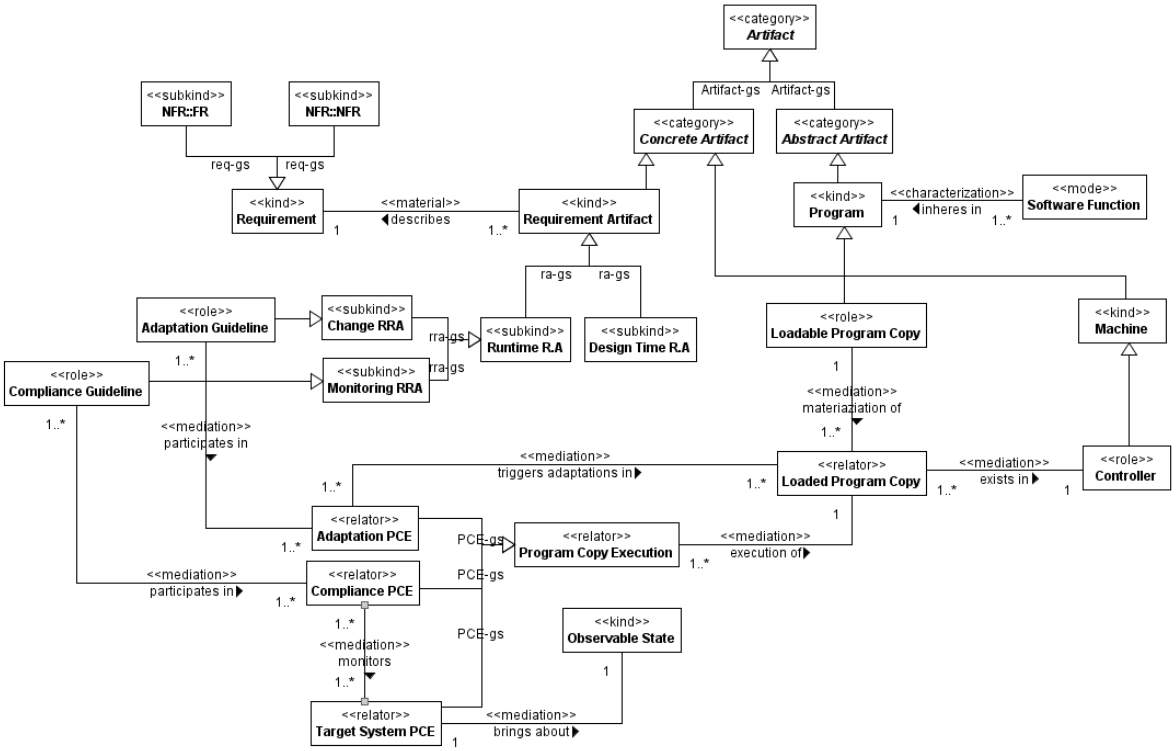

Figura 24 – RRO reescrita em OntoUML para verificação do modelo

definidos de maneira empírica que aparecem em modelos conceituais e que, embora estejam sintaticamente corretos, podem produzir resultados indesejados (muitas vezes de difícil visualização) na hora de representar um determinado domínio (SALES; GUIZZARDI, 2015; GUIZZARDI; SALES, 2014). Anti-padrões devem ser analisados e tratados com cautela, pois mesmo que não representem um erro diretamente, podem causar instâncias indesejadas no modelo conceitual, acabando por afetar negativamente a capacidade do modelo conceitual de representar o domínio.

Para analisar os APs existentes em RRO-OntoUML, foi utilizada a ferramenta existente dentro do OLED que analisa as entidades, relacionamentos e cardinalidades em busca da existência de APs. A Figura 25 apresenta o resultado da análise inicial de APs realizada em RRO-OntoUML.

Dos 21 anti-padrões analisados no modelo de RRO-OntoUML foram encontradas 11 ocorrências de 6 APs conforme demonstrado na Figura 25. A Figura 26 apresenta a tela de análise dos APs encontrados nessa ontologia. É a partir dessa tela que cada AP é verificado individualmente e a estratégia de tratamento do AP é escolhida, sendo que cada AP pode possuir uma ou várias estratégias de tratamento e cabe ao engenheiro de ontologias decidir qual a estratégia mais adequada para seu modelo.

O primeiro anti-padrão a ser avaliado é o de *Association Cycle* (AssCyc), um anti-padrão genérico, ou seja, não depende necessariamente do aparecimento de uma entidade específica para ocorrer, podendo acontecer com quaisquer classes que cumprem

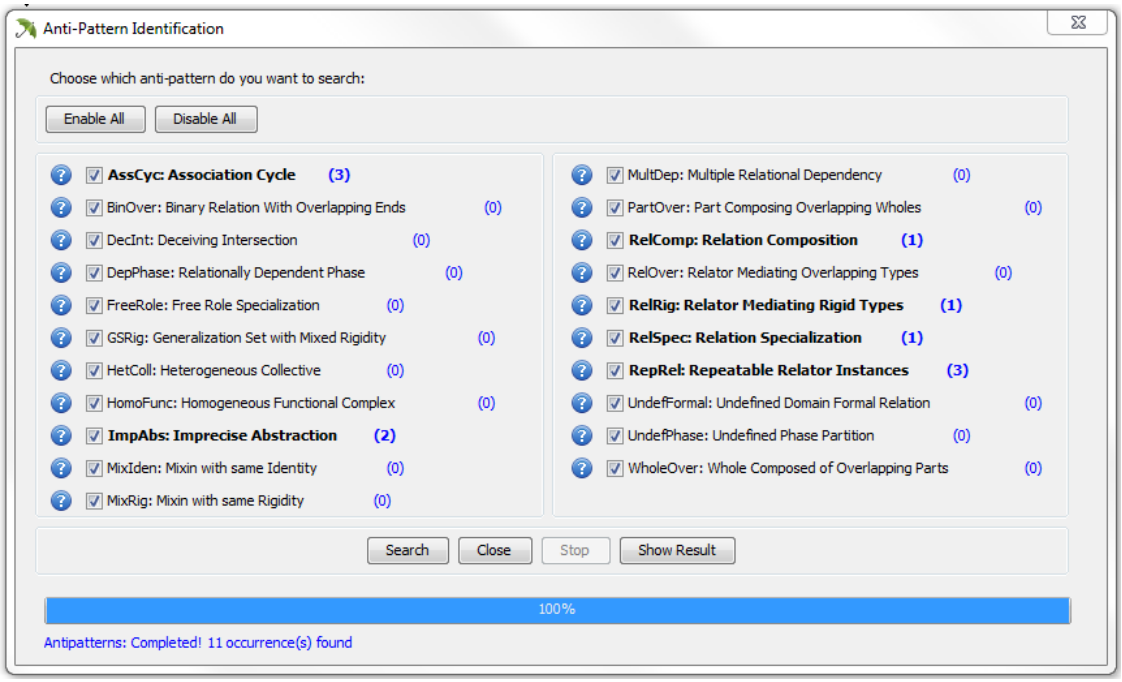

Figura 25 – Análise de anti-padrões realizada em RRO-OntoUML utilizando o editor OLED

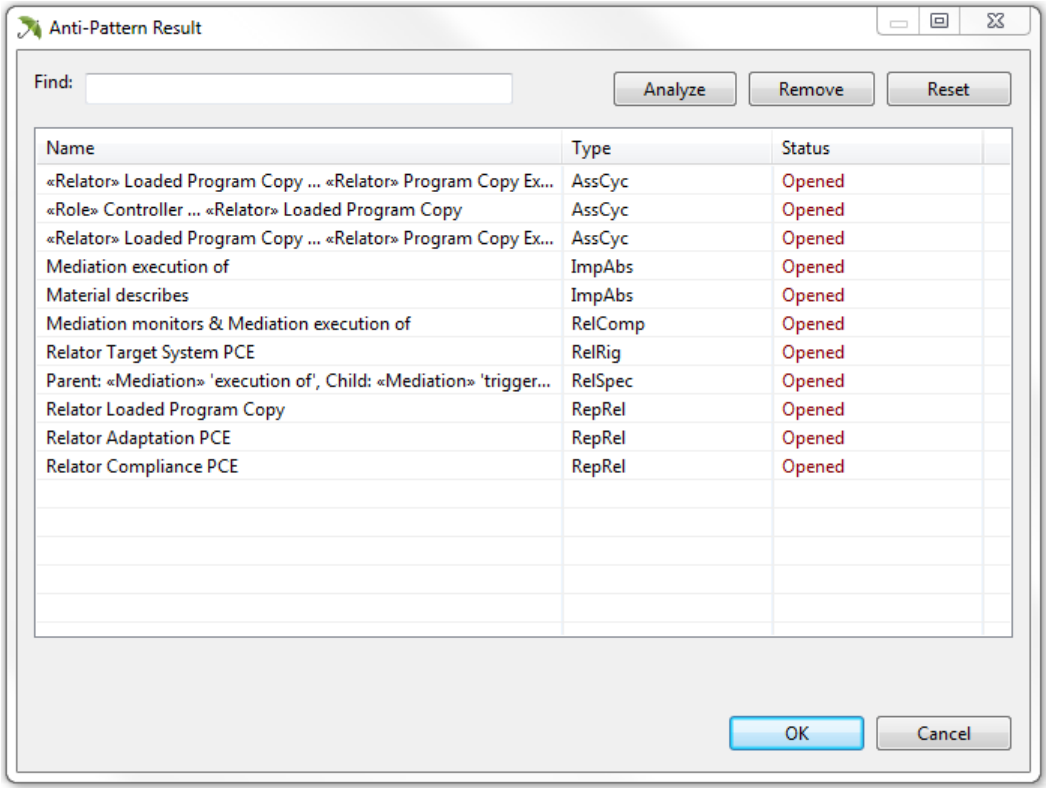

Figura 26 – APs exibidos separadamente pelo OLED

suas condições de existência. AssCyc se caracteriza quando um número arbitrário de classes do modelo estão conectadas através do mesmo número de relações (sejam essas relações formais, materiais ou mesmo generalizações) de forma a compor um ciclo (SALES; GUIZZARDI, 2015). Dessa forma, o anti-padrão se preocupa justamente com a existência de ciclos no nível de instanciação do modelo, visto que ciclos no modelo de referência não necessariamente acontecem em nível de instanciação.

Para tratar esse AP, existem três estratégias possíveis: reforçar a existência do ciclo em possíveis instâncias do modelo, através da definição de uma restrição escrita em linguagem OCL; definir uma outra restrição para proibir a existência do ciclo em todas as instâncias da ontologia; ou definir uma das relações existentes no modelo como derivada e especificar a regra OCL correspondente a essa derivação. No caso de RRO-OntoUML, a primeira estratégia foi adotada pois o AssCyc detectado existe de fato no domínio de RRO e devido a isso, deve ser reforçada com a regra OCL correspondente. A Listagem 4.1 apresenta o código OCL definido no modelo que reforça o ciclo existente entre as classes em nível de instância do modelo, e assim, corrigindo as ocorrências do anti-padrão.

Listagem 4.1 – Código OCL que corrige o AP AssCyc reforçando os ciclos existentes na ontologia .

```
1 context _'Compliance PCE'
2 inv cyclic : self._'compliance guideline'.oclAsType(_'Controller')._'loaded
    program copy'._'program copy execution'.oclAsType(_'Compliance PCE')->includes
    (self)
3
4 context _'Adaptation PCE'
5 inv cyclic : self._'loaded program copy'._'program copy execution'.oclAsType(_'
    Compliance PCE')._'compliance guideline'.oclAsType(_'Adaptation Guideline')._'
    adaptation pce'->includes(self)
6
7 context _'Loaded Program Copy'
8 inv cyclic : self._'loadable program copy'.oclAsType(_'Compliance Guideline')._'
    compliance pce'.oclAsType(_'Program Copy Execution')._'loaded program copy'->
    includes(self)
```

O segundo anti-padrão a ser analisado é o denominado *Imprecise Abstraction* (ImpAbs) que se caracteriza quando uma relação R possui uma de suas multiplicidades (fonte ou alvo) igual ou maior a dois e a classe C conectada possui dois ou mais subtipos C1 e C2. ImpAbs é considerado um AP por tornar o modelo muito permissivo, admitindo a existência de instâncias indesejadas do modelo. Para corrigir o AP podem ser utilizadas três estratégias: (i) definir restrições de cardinalidade da relação R para todos os subtipos C1, C2, ..., Cn de C através de uma regra OCL; (ii) definir novas relações que sejam subsets da relação original R para com os subtipos de C; (iii) especificar novas relações para com os subtipos de C, porém definindo valores para meta propriedades específicas de cada relação. Para corrigir os dois casos de ImpAbs encontrados em RRO, a estratégia de definição de restrições de cardinalidade através de regra OCL foi adotada.

A Figura 27 apresenta as duas ocorrências AP *Imprecise Abstraction* que acontecem

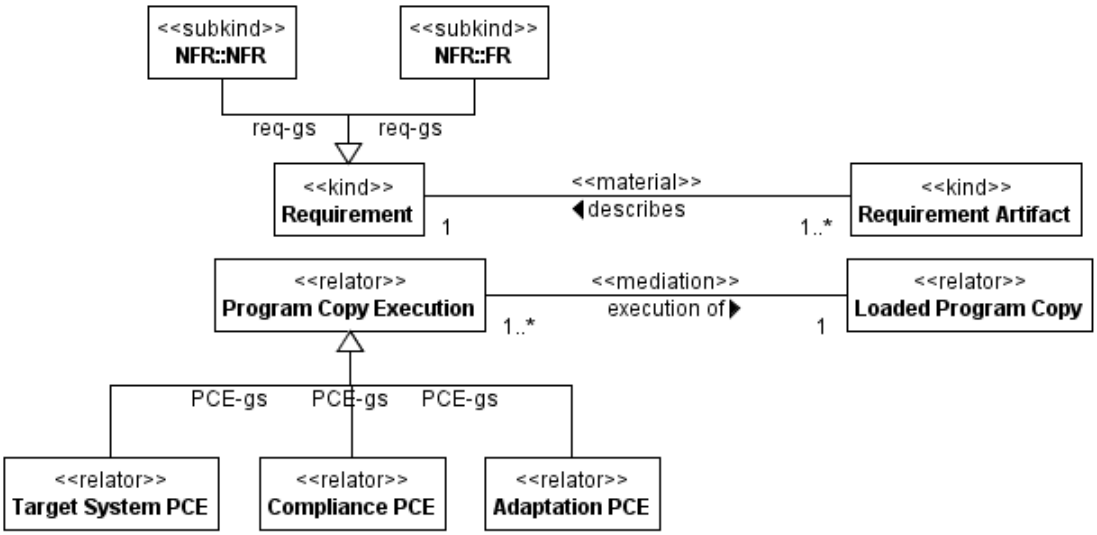

Figura 27 – Ocorrência do Anti-Padrão *Imprecise Abstraction* em RRO

em RRO: a primeira ocorre entre as classes *Requirement Artifact* e *Requirement*, a segunda ocorre entre as classes *Program Copy Execution* e *Loaded Program Copy*. A Listagem 4.2 apresenta os códigos em OCL que adicionam restrições de multiplicidade às classes envolvidas, dessa forma, tratando as ocorrências do AP.

Listagem 4.2 – Código OCL que trata as ocorrências do AP ImpAbs.

```
1 context _'RequirementArtifact'
2 inv: let sub1Size =
3 self.requirement->select( x |x.oclIsTypeOf(_'NFR::FR'))->size()
4 in sub1Size >= min1 and sub1Size <= max1
5
6 context _'Loaded Program Copy'
7 inv: let sub1Size =
8 self.programcopyexecution->select( x |x.oclIsTypeOf(_'AdaptationPCE'))->size()
9 in sub1Size >= min1 and sub1Size <= max1
```

O terceiro AP encontrado em RRO é o de *Relation Specialization* (RelSpec). Esse anti-padrão acontece quando duas relações A e B que conectam respectivamente A-fonte a A-alvo, B-fonte a B-alvo existem de tal forma que A-fonte é super-tipo de B-fonte e A-alvo é super-tipo de B-alvo. Esse padrão deve ser tratado pois sua estrutura sugere a existência de uma especialização entre as relações, portanto, é preciso especificar qual o tipo de dependência existente entre as duas relações. No caso de RRO, uma relação não é derivada e nem sub-tipo da outra, dessa forma, foi adotada a estratégia de definir uma regra OCL que define as relações *execution of* e *triggers adaptations in* como disjuntas.

A Figura 28 apresenta a ocorrência do AP RelSpec que acontece entre as relações *execution of* e *triggers adaptations in*. A Listagem 4.3 apresenta o código OCL utilizado para definir a disjunção entre as relações {*monitors*,*brings about*} e {*execution of*,*changes*} em RRO.

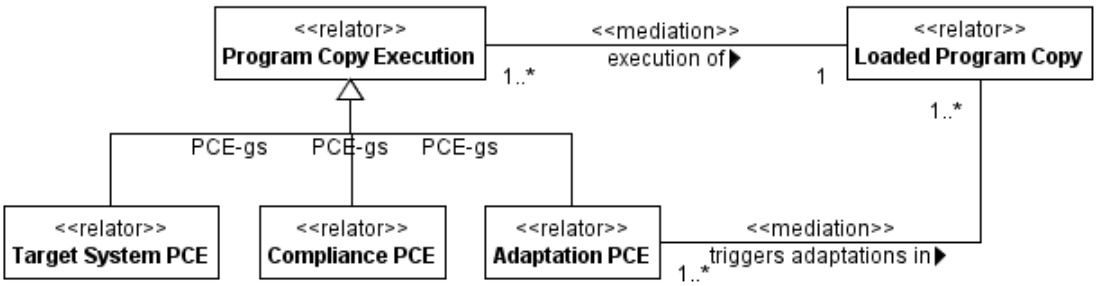

Figura 28 – Ocorrência do Anti-Padrão *Relation Specialization* em RRO

Listagem 4.3 – Código OCL que trata o AP RelSPec defindo os dois relacionamentos como disjuntos.

```
1 context _'Adaptation PCE'
2 inv triggers adaptations in_disjointWith_execution of : self._'loaded program
    copy'->asSet()->excludesAll(self.oclAsType(_'Program Copy Execution')._'loaded
     program copy'->asSet())
```

O quarto AP que aparece em RRO-OntoUML é o denominado *Repeatable Relator Instances* (RepRel). RepRel acontece quando um *Relator* está conectado a duas ou mais relações de *Mediation* com cardinalidade superior a 1 do lado do *Relator*. RepRel visa ajudar o modelador a especificar o número de diferentes instâncias de *Relators* que podem mediar o mesmo grupo de indivíduos.

A Figura 29 apresenta duas das ocorrências do AP RepRel em RRO. Na primeira, o *Relator Compliance PCE* se conecta às relações de mediação *monitors* e *participates in* com cardinalidade 1..*. Na segunda, o *Relator Adaptation PCE* está conectado as mediações *participates in* e *triggers adaptations in* com cardinalidade 1..*. Para corrigir uma ocorrência do AP foi definida uma regra OCL de forma que essa defina a unicidade do relator. A terceira ocorrência, não exibida na Figura 29, se dá com as mediações conectadas ao *Relator Loaded Program Copy*. A Listagem 4.4 apresenta a restrição OCL utilizada para definir a restrição de unicidade do *Relator Program Copy Execution*.

Listagem 4.4 – Código OCL que trata o AP RepRel.

```
1 context _'Program Copy Execution'
2 inv: _'Program Copy Execution'.allInstances()->select( r | r<>self and r._'
    Runtime Requirement Artifact'=self._'Runtime Requirement Artifact' and
3 r._'Loaded Program Copy'=self._'Loaded Program Copy' )->size()=<n-1>
```

O quinto AP encontrado é *Relation Composition* (RelComp). Esse AP é estritamente lógico e ocorre quando duas relações A e B, conectam A-fonte para A-alvo e B-fonte para B-alvo, respectivamente. Para que o AP ocorra, uma das condições a seguir deve ser verdadeira: B-fonte e B-Alvo são iguais ou são sub-tipos de A-Alvo ou B-fonte e B-Alvo são iguais ou são sub-tipos de A-fonte. Esse AP deve ser tratado pois a nível de instanciação

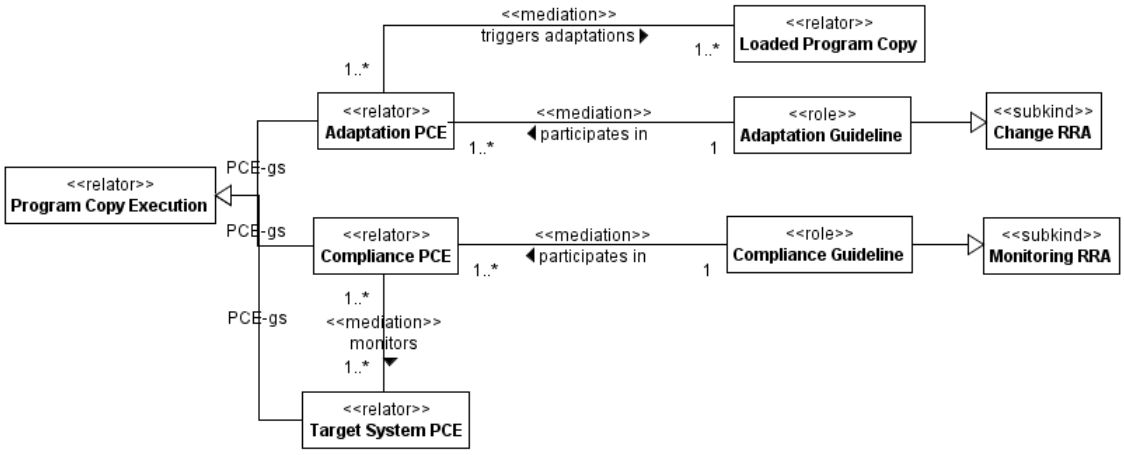

Figura 29 – Ocorrência do Anti-Padrão *Repeatable Relator Instances* em RRO

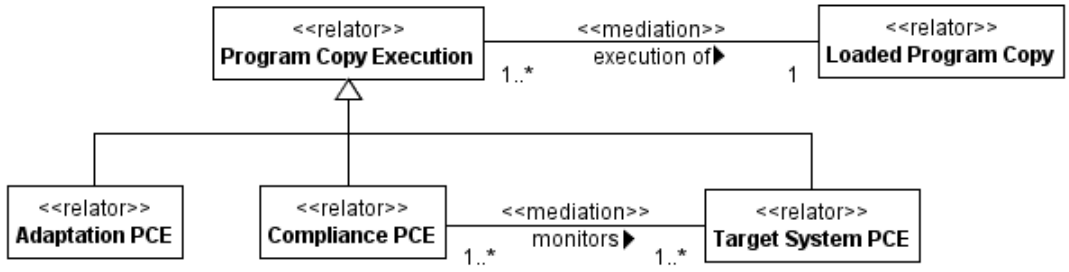

Figura 30 – Ocorrência do Anti-Padrão *Relation Composition* em RRO

do modelo, as duas relações identificadas podem restringir uma a outra e devido a isso acabar gerando instâncias não desejadas.

A Figura **??** apresenta a ocorrência desse AP em RRO, que se dá entre as classes *Program Copy Execution*, *Compliance PCE* e *Target System PCE* e *Loaded Program Copy*.

Para corrigir esse AP, foi criada uma restrição OCL, apresentada na listagem **??** que define a relação de composição existencial entre uma possíveis instâncias de *Compliance PCE* e *Target System PCE* e *Program Copy Execution*.

Listagem 4.5 – Código OCL que trata o AP RelComp.

```
1 context _'Program Copy Execution'
2 inv  : self._'loaded program copy'->asSet()->forAll( x: _'Loaded Program Copy' |
    self.oclAsType(_'Compliance PCE')._'target system pce'->asSet()->intersection(
    x.oclAsType(_'Compliance PCE')._'target system pce'->asSet())->size()>=1)
```

O ultimo AP encontrado em RRO foi *Relator Mediating Rigid Types* (RelRig). Esse AP ocorre quando um *Relator* está conectado, por meio de uma relação de mediação, a um dos tipos rígidos de OntoUML (*Kind*, *SubKind*, *Category*). Embora RelRig não seja um erro propriamente dito, ele deve ser tratado pois torna o modelo muito permissivo,

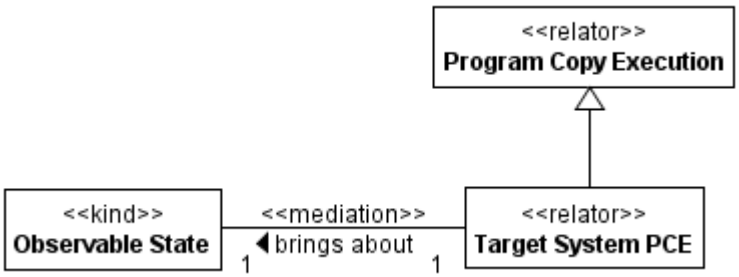

Figura 31 – Ocorrência do Anti-Padrão *Relator Mediating Rigid Types* em RRO

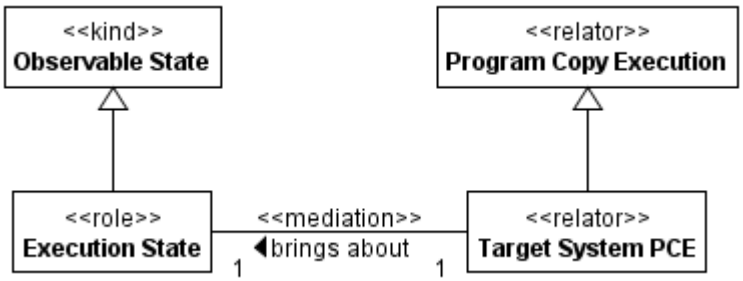

Figura 32 – Criação do *Role Execution State* para a correção de ocorrência do AP RelRig

no sentido de que ao ligar um tipo rígido a um *relator* com uma relação de mediação, o engenheiro de ontologias está afirmando que todas as possíveis papéis, fases ou instâncias que o tipo rígido possa ter durante sua existência serão mediadas por aquele *relator*, o que nem sempre é verdade e, assim, acaba por tornar o modelo impreciso.

Diferentemente dos demais APs apresentados aqui, RelRig não pode ser corrigido através da definição de uma restrição OCL. A estratégia adotada para corrigir a ocorrência de RelRig em RRO foi a de definir um *Role* (*Execution State*) para o tipo rígido *Observable State* de forma que esse *Role* fosse conectado ao *Relator Target System PCE* através da relação de mediação *brings about*. A Figura 31 apresenta a correção do AP demonstrado na Figura 30.

Por fim, a Figura 32 apresenta a tela do OLED com os APs encontrados em RRO devidamente tratados, a Tabela 2 apresenta os *anti-patters* encontrados em RRO-OntoUML e as regras OCL utilizadas em seus tratamentos e a Figura **??** apresenta a versão final de RRO-OntoUML como os anti-padrões devidamente corrigidos.

4.2 Validação da Ontologia

Validar uma ontologia significa garantir que a ontologia correta tenha sido construída e que essa cumpra o papel para qual foi projetada. Para validar RRO foram adotadas duas formas de validação presentes na literatura sobre ontologias: a primeira proposta em SABiO consiste em instanciar a ontologia através de exemplos do mundo real, a segunda

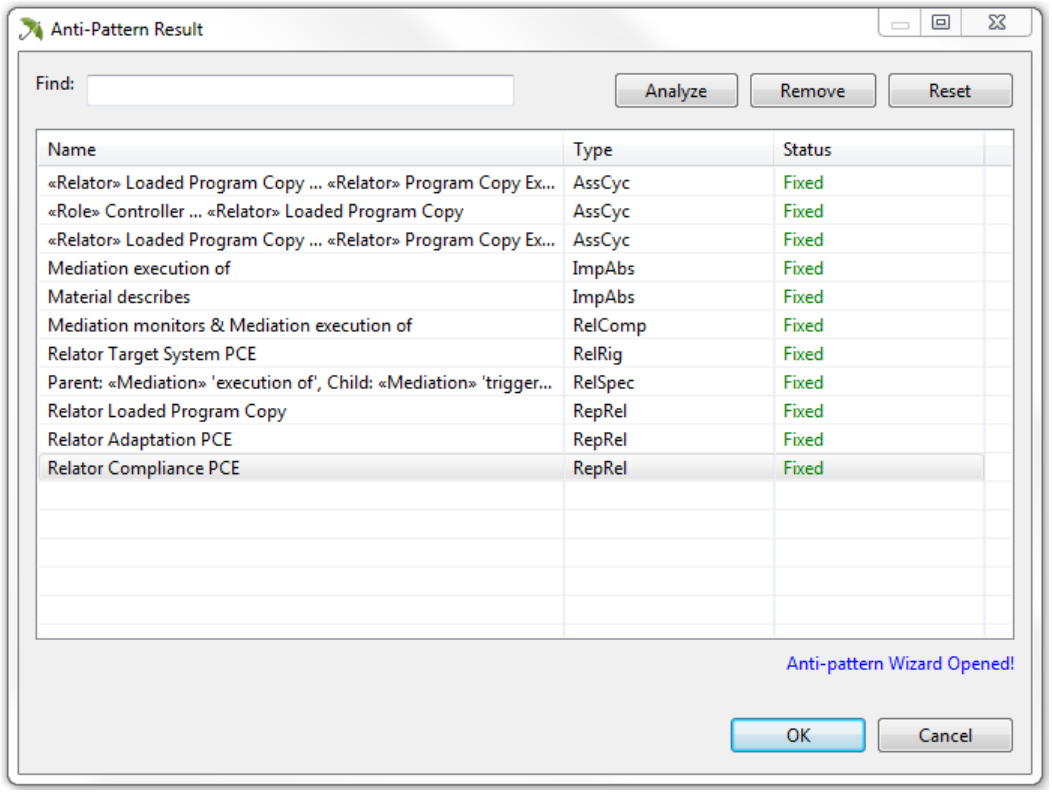

Figura 33 – Anti-Padrões Corrigidos no editor OLED

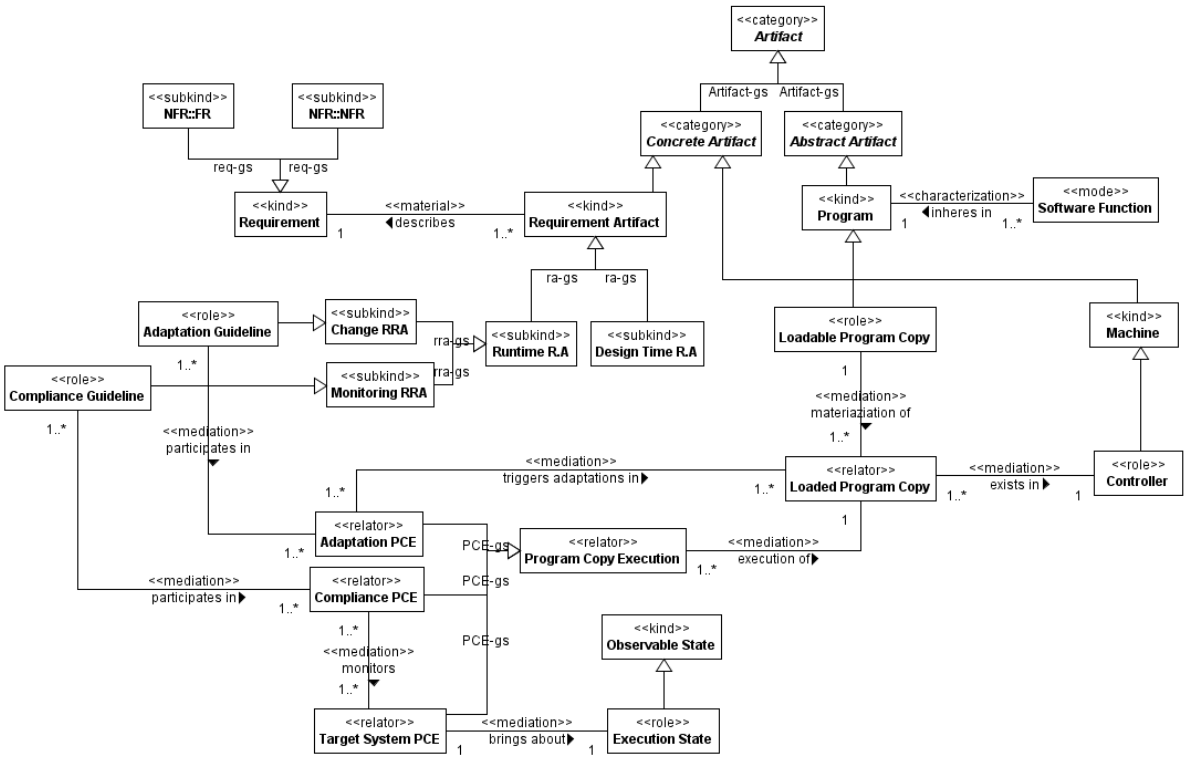

Figura 34 – Modelo de RRO-OntoUML com as modificações realizadas após a análise de APs.

Tabela 2 – Regras OCL utilizadas no modelo de RRO-OntoUML.

Anti-Padrão	**OCLs Utilizadas**
AssCyc	context _'**Compliance PCE**' inv cyclic : self._'compliance guideline'.oclAsType(_'**Controller**')._'loaded program copy'._'program copy execution'.oclAsType(_'**Compliance PCE**')->includes(self) context _'**Adaptation PCE**' inv cyclic : self._'loaded program copy'._'program copy execution'.oclAsType(_'**Compliance PCE**')._'compliance guideline'.oclAsType(_'**Adaptation Guideline**')._'adaptation pce'->includes(self) context _'**Loaded Program Copy**' inv cyclic : self._'loadable program copy'.oclAsType(_'**Compliance Guideline**')._'compliance pce'.oclAsType(_'**Program Copy Execution**')._'loaded program copy'->includes(self)
ImpAbs	context _'**Requirement Artifact**' inv: let sub1Size = self.requirement->select(x |x.oclIsTypeOf(_'**NFR::FR**'))->size() in sub1Size >= min1 and sub1Size <= max1 context _'**Loaded Program Copy**' inv: let sub1Size = self.programcopyexecution->select(x |x.oclIsTypeOf(_'**Adaptation PCE**'))->size() in sub1Size >= min1 and sub1Size <= max1
RelSpec	context _'**Adaptation PCE**' inv triggers adaptations in_disjointWith_execution of : self._'loaded program copy'->asSet()->excludesAll(self.oclAsType(_'**Program Copy Execution**')._'loaded program copy'->asSet())
RelComp :	context _'**Program Copy Execution**' inv self._'loaded program copy'->asSet()->forAll(x: _'**Loaded Program Copy**' | self.oclAsType(_'**Compliance PCE**')._'target system pce'->asSet()->intersection(x.oclAsType(_'**Compliance PCE**')._'target system pce'->asSet())->size()>=1)
RepRel	context _'**Program Copy Execution**' inv: _'**Program Copy Execution**'.allInstances()->select(r | r<>self and r._'**Runtime Requirement Artifact**'=self._'**Runtime Requirement Artifact**' and r._'**Loaded Program Copy**'=self._'**Loaded Program Copy**')->size()=<n-1>

Tabela 3 – Instanciação da RRO utilizando o exemplo *Meeting Scheduler* apresentado em (SOUZA, 2012).

Conceito	**Instanciação**
Requirement	Um *stakeholder* deseja um sistema de software capaz de permitir aos usuários a criação e o agendamento de reuniões (dentre outras funcionalidades).
Program	Uma implementação do *Meeting Scheduler* para uma determinada máquina (ex.: um sistema operacional específico).
Software Function	Uma *Disposition* de uma implementação específica do *Meeting Scheduler* que representa a caracterização de uma reunião dado um conjunto de parâmetros for fornecido pelo usuário.
Loaded Program Copy	Materialização de uma implementação do *Meeting Scheduler* carregada em memória.
Program Copy Execution	O evento da cópia carregada em memória do *Meeting Scheduler* ganhando a posse da CPU e sendo executada em uma máquina.
Observable State	Uma instância de uma execução completa do *Meeting Scheduler* feita por um usuário onde as características de uma reunião são fornecidas e uma reunião é então criada (ex: tem seu registro armazenado em um banco de dados).
Requirement Artifact	A tarefa *Characterize Meeting* do modelo de requisitos do *Meeting Scheduler*.
Runtime Requirement Artifact	Tarefa *Characterize Meeting* representada no formato XML de forma a ser consumida por um componente do *Zashin* durante a execução do programa.
Compliance Program Copy Execution	O Componente de monitoramento do *Zanshin* rodando em uma máquina.
Monitoring Runtime Requirement Artifact	O *AwReq* (SOUZA et al., 2013) *Characterize meeting should never fail*, representado em um arquivo XML.
Adaptation Program Copy Execution	O componente de adaptação do *Zanshin* rodando em uma máquina.
Change Runtime Requirement Artifact	O *EvoReq* (SOUZA et al., 2013) *Retry Characterize Meeting after 5 seconds*, representado em um arquivo XML.

consiste gerar instâncias da ontologia e utilizar os cenários gerados para validar o modelo.

4.2.1 Validação por Instanciação

SABiO sugere que a ontologia de referência seja instanciada e que tabelas de instanciação sejam criadas de forma a representar situações do mundo real por meio das entidades da ontologia. A Tabela 3 apresenta a instanciação de RRO feita com o *Meeting Scheduler* utilizado como prova de conceito do Zanshin *framework*, previamente descrito no Capítulo 2. Os conceitos de RRO são apresentados na tabela e comparados aos elementos do *Meeting Scheduler* apresentado em (SOUZA, 2012).

Da mesma forma, a Tabela 4 apresenta a instanciação de RRO utilizando um exemplo de uma página de comércio eletrônico utilizada por Robinson (2006) para ilustrar

Tabela 4 – Instanciação da RRO utilizando o exemplo de *e-commerce* baseado no ReqMon e em apresentado em (ROBINSON, 2006).

Conceito	**Instanciação**
Requirement	Um consumidor tem o objetivo realizar uma compra on-line.
Program	Uma implementação de um site de comércio eletrônico para uma determinada máquina (ex.: um sistema operacional específico).
Software Function	Uma *Disposition* de uma implementação específica de página de *E-commerce* que representa a compra de um determinado produto
Loaded Program Copy	Materialização de uma implementação do *E-commerce* carregada na memória principal.
Program Copy Execution	O evento que representa a cópia carregada em memória do *E-commerce* ganhando a posse da CPU e sendo executada em uma máquina (ex.: um servidor de aplicação).
Observable State	Uma instância de uma execução completa do *E-commerce* feita por um usuário
Requirement Artifact	O objetivo *CreateVendorPurchaseOrder* do modelo de requisitos da página de *e-commerce*
Runtime Requirement Artifact	O objetivo *CreateVendorPurchaseOrder* representado em formato XML para ser utilizado por um monitor do ReqMon.
Compliance Program Copy Execution	*ReqMon Monitoring Tools* sendo executado em uma máquina.
Monitoring Runtime Requirement Artifact	O monitor *IMonPurchaseOrderDenialOfService* derivado do requisito de mesmo nome.
Adaptation Program Copy Execution	ReqMon considera adaptação de requisitos mas não implementa o componente (*Reactor*) responsável no exemplo elaborado.
Change Runtime Requirement Artifact	A entidade não é implementada pelo ReqMon visto que o *framework* é focado apenas na parte de monitoramento e não de adaptação

o *framework* ReqMon, previamente apresentado na Seção 2.2.

Através do estudo realizado em cima dos dois *frameworks* citados e da instanciação dos mesmos, representadas nessa seção na forma das Tabela 3 e 4 pode-se concluir que RRO representa bem dois *frameworks* com características distintas, onde um trata da criação de sistemas adaptativos (*Zanshin Framework*), capazes de mudar seus requisitos caso seja necessário, enquanto o outro (*ReqMon Framework*) trata apenas do monitoramento de seus requisitos durante o tempo de execução .

4.2.2 Validação por Simulação Visual de Instâncias do Modelo

Para realizar uma simulação visual de RRO (BENEVIDES et al., 2010) a versão OntoUML de RRO foi submetida ao analisador Alloy (JACKSON, 2012) existente na

ferramenta OLED. Tendo como entradas um modelo OntoUML e um conjunto de regras e restrições do modelo escritas em OCL (OMG, 2014), o analisador é capaz de gerar automaticamente diversas instâncias do modelo de forma a simular o conjunto de propriedades desejadas/indesejadas que o modelo é capaz de refletir. Caso alguma instância não desejada aconteça, uma modificação no modelo deve ser feita ou uma restrição deve ser inserida para garantir que tal instância indesejada não ocorra novamente, no caso oposto, se alguma instância desejada não apareça, o engenheiro de ontologias deve verificar se não há alguma restrição impedindo a criação da instância. As instâncias permitem, ao engenheiro de ontologias, uma melhor visualização do que o modelo verdadeiramente representa.

A Figura 33 apresenta duas instanciações distintas da parte do modelo de RRO-OntoUML, que tratam do relacionamento entre os tipos de requisitos em tempo de execução e a nomenclatura clássica de requisitos, que os define como funcionais ou não-funcionais. A primeira parte da figura apresenta dois requisitos de monitoramento e um requisito de mudança que representam (em tempo de execução) três requisitos não funcionais distintos. A parte inferior da figura apresenta um requisito de mudança e um de monitoramento sendo descritos por dois requisitos funcionais, representados pelos objetos 4 e 2. Os mundos apresentados na Figura 33 demonstram que a classificação de requisitos como tempo de execução/tempo de desenho é ortogonal a classificação que os separa em funcional/não-funcional.

A Figura 34 apresenta outra possível instância do modelo de RRO-OntoUML criada pelo *Alloy Analyzer* do OLED. Na situação mostrada na figura temos a execução de 4 programas distintos representados pelas propriedades 1, 4, 9 e 3, sendo que os dois primeiros referem-se a execuções de programas principais e os dois últimos à execuções de programas que são responsáveis por disparar adaptações. Os programas representados pelas propriedades são execuções de programas carregados na memória de uma máquina representados pelas propriedades 7 e 5. As execuções representadas pelas propriedades 9 e 3 fazem uso de requisitos de mudança (artefatos concretos), que são representados pelos objetos 1 e 8 e que de descrevem respectivamente os requisitos que são representados pelos objetos 9 e 0. Por fim, todas as execuções 1, 4, 9 e 3 geram estados observáveis representados pelos objetos 3, 4, 5 e 6.

A Figura 35 apresenta um caso mais simples de monitoramento onde uma cópia de programa carregado em memória (property5) — que existe em uma maquina (object4) e é a representação de um programa (object6) — é executado (property7) e faz uso de dois requisitos de monitoramento (object8 e object5) para monitorar aspectos gerados por sua própria execução (object1).

As instâncias apresentadas nessa seção foram escolhidas pois representarem instâncias casos que RRO deve ser capaz de cobrir, quando trata-se do uso de requisitos

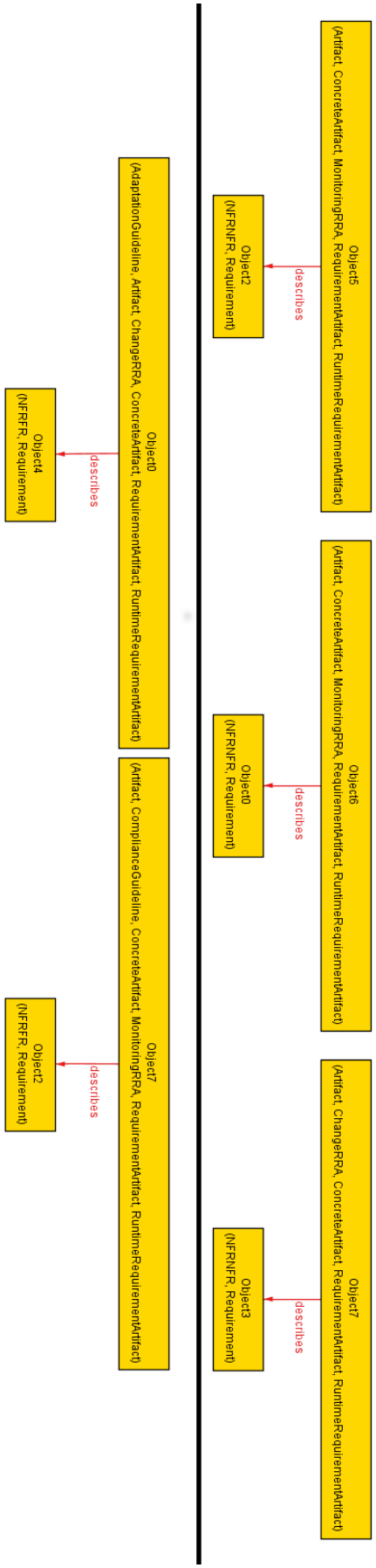

Figura 35 – Instanciações do Modelo de RRO-OntoUML que demonstram a relação entre os subtipos de requisitos em tempo de execução definidos em RRO com a nomenclatura clássica de requisitos

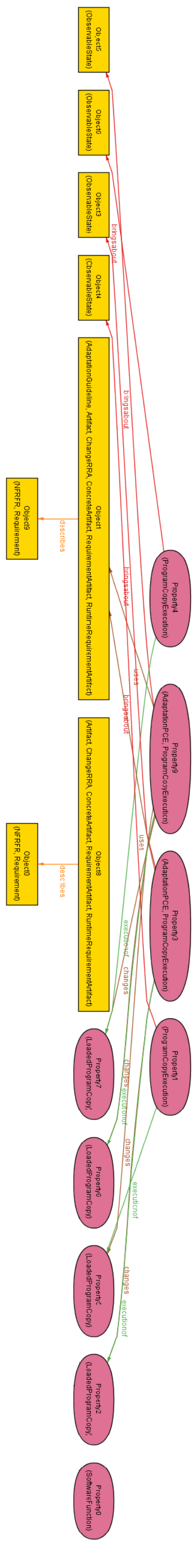

Figura 36 – Possível instanciação do Modelo de RRO-OntoUML criada a partir do *Alloy Analyzer* mostrando execuções de diversos programas

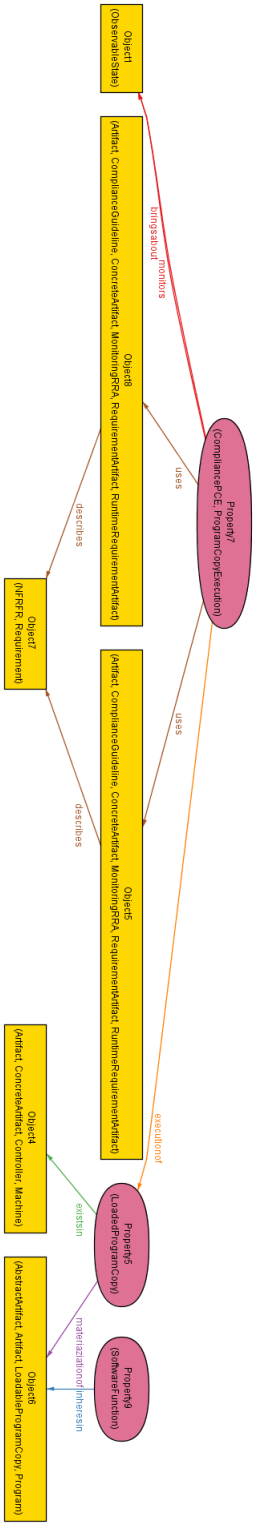

Figura 37 – Possível instanciação do Modelo de RRO-OntoUML criada a partir do *Alloy Analyzer* mostrando a execução de um programa de monitoramento

de monitoramento e de mudança (adaptação) e por serem de fácil visualização, sendo o ultimo fator considerado relevante pois como as instâncias são geradas automaticamente pelo analisador essas podem possuir uma quantidade muito grande de relacionamentos (representados pelas setas) e classes (representados pelas caixas e elipses) no modelo gerado dificultando muito a visualização da instância gerada e acabando por deixá-la inutilizável. Ao analisar as instâncias apresentadas nessa seção é possível perceber que RRO é capaz de gerar instâncias válidas para o domínio para o qual foi construída de forma satisfatória.

Por fim, é importante acrescentar que todas as formas de avaliação apresentadas aqui visam comprovar que RRO é uma ontologia bem construída, que atende a seus requisitos e é capaz de cumprir bem sem papel como uma ontologia de referência de domínio.

5 Conclusões

O uso de requisitos em tempo de execução é uma área relativamente nova dentro da Engenharia de Requisitos de software e que vem crescendo ao longo dos anos, como pode ser percebido nos resultados do mapeamento sistemático realizado. Contudo, sendo uma área nova, ainda não existe um consenso da comunidade sobre as principais entidades e conceitos que compõem o domínio. Neste contexto, RRO foi pensada e construída como uma ontologia de referência de domínio com o objetivo de prover um vocabulário formal sobre o domínio de requisitos em tempo de execução.

RRO foi originalmente projetada e construída para ser uma ontologia de referência de domínio fundamentada em UFO com o objetivo representar formalmente o conhecimento relevante ao uso de requisitos de software em tempo de execução, além de prover um vocabulário formal consistente sobre domínio que possa ser utilizado como base para a comunicação entre Engenheiros de Requisitos, *Stakeholders* e os diversos atores que participam do processo de Engenharia de Requisitos de sistemas que farão uso de requisitos em tempo de execução. RRO foi criada segundo o método SABiO e, devido a isso, optamos por seguir as sugestões do método e reusar ontologias já existentes e a realizar um forte processo de aquisição de conhecimento, executado em RRO sob a forma de um mapeamento sistemático da literatura. Duas ontologias foram reusadas em RRO, a ontologia de artefatos de software (representada no modelo de RRO com o prefixo OSA) e a ontologia de requisitos não-funcionais (representada no modelo de RRO com o prefixo RNF). O mapeamento sistemático foi realizado segundo a metodologia apresentada em (KITCHENHAM; CHARTERS, 2007) e serviu como base de conhecimento para todas as decisões de projeto tomadas durante o processo de construção de RRO. É importante destacar também que todo o processo de realização do mapeamento sistemático e da construção de RRO foi altamente iterativo, contando com a participação de diversos especialistas de domínio e engenheiros de ontologias em reuniões semanais foram realizadas para que os resultados parciais do trabalho estivessem sempre sendo discutidos. O resultado completo do mapeamento sistemático e todo o *reasoning* sobre ele devem ser publicados em um futuro próximo.

RRO, como modelo de referência, fornece então uma descrição ontologicamente fundamentada de requisitos em tempo de execução, seus subtipos, suas relações e relacionamentos com as demais entidades que compõem o domínio e do propósito de utilização dos tipos de requisitos em tempo de execução originalmente encontrados na literatura e representados na ontologia. RRO também apresenta uma análise ontológica sobre a natureza da entidade requisito, que inicialmente existe como o objetivo de um agente e que posteriormente, conforme o ciclo de vida do software avança, passa a ser representado

como um artefato para uso humano ou para uso por máquinas.

Para que RRO pudesse ser avaliada de forma consistente, a mesma teve de ser reescrita em linguagem OntoUML, visto que ainda não existe na literatura um método sistemático para validar ontologias fundamentadas em UFO-B/UFO-C. Em contrapartida, um modelo de ontologia de referência escrito em OntoUML pode beneficiar-se de diversas formas de verificação e validação encontradas na literatura. De posse desse novo artefato foram utilizadas diversas formas de verificação e validação de ontologias presentes na literatura.

Com relação à validação, foram adotadas duas estratégias para validar RRO: a primeira consistiu em comparar os conceitos presentes em RRO com os conceitos propostos por dois *frameworks* que fazem uso de requisitos em tempo de execução. Essa comparação serviu para demonstrar que RRO é capaz de demonstrar o conhecimento sobre requisitos em tempo de execução presente na literatura e para demonstrar que a mesma pode representar exemplos reais existentes na literatura sobre o domínio. A segunda forma de validação consistiu em utilizar um editor de desenvolvimento de ontologias denominado OLED para reescrever o modelo original de RRO em linguagem OntoUML e através desse novo modelo utilizar uma ferramenta do OLED que permite a geração automática de diversas instâncias gráficas possíveis da ontologia para que possam ser analisadas individualmente e validadas.

Para verificar RRO também foram adotadas duas formas distintas de verificação: a primeira consistiu em responder as questões de competência de RRO para determinar se a ontologia foi construída corretamente e consequentemente atende a seus requisitos. A segunda forma de verificação consistiu em verificar a existência de *anti-patterns* no modelo de RRO. Assim como a validação através de linguagem Alloy, essa verificação é feita de forma automática por uma das ferramentas presentes no OLED, sendo capaz de apontar possíveis pontos de atenção na construção da ontologia, que devem ser verificados para evitar o aparecimento de erros de representatividade do domínio. Os *anti-patterns* encontrados em RRO foram devidamente tratados utilizando os métodos apontados na literatura sobre o assunto de forma que no final do processo uma ontologia mais robusta (embora mais pesada e axiomatizada) é obtida.

5.1 Contribuições

As principais contribuições desse trabalho foram:

- Uma ontologia de referência de domínio sobre o uso de requisitos em tempo de execução fundamentada em UFO que fornece uma descrição da natureza dos tipos de requisitos em tempo de execução e das demais entidades que compõem o domínio;
- Uma versão alternativa da ontologia de referência de domínio original, escrita em

linguagem OntoUML e assim sendo passível de ser facilmente convertida para uma ontologia operacional;

- RRO fornece uma base para um trabalho que permita a interoperabilidade de modelos de requisitos em tempo de execução baseados em abordagens diferentes;
- Através da descrição de suas entidades, RRO fornece uma base para a construção de um vocabulário formal sobre o uso de requisitos em tempo de execução.
- Um mapeamento sistemático da literatura existente sobre requisitos em tempo de execução que contou com a participação de diversos especialistas no assunto e foi utilizado como base de conhecimento para a construção de RRO.

5.2 Dificuldades e Limitações

Dentre as principais dificuldades encontradas na realização desse trabalho podemos destacar a dificuldade em realizar o mapeamento sistemático, que resultou em um número de artigos bem maior do que o esperado e consequentemente acabou por causar um aumento exponencial na carga de trabalho a ser realizada. A outra principal dificuldade encontrada foi para avaliar a ontologia proposta. Essa dificuldade se deu principalmente pela falta de metodologias sistemáticas para a validação de ontologias baseadas em UFO-B e UFO-C e pela falta de maturidade das ferramentas para avaliação de ontologias escritas em modelo OntoUML. As ferramentas existentes atualmente ainda encontram-se em versão beta ou pré-lançamento e, devido a isso, estão sujeitas a *bugs* e inconsistências que levaram ao corrompimento do arquivo do modelo de RRO em mais de um momento durante a realização desse trabalho.

Com relação as limitações, é importante considerar o fato de que o mapeamento sistemático que fora realizado como base de adquisição de conhecimento para a construção de RRO ainda não foi publicado e portanto, ainda não foi criticado e aprovado pela comunidade. Outra limitação que deve ser mencionada é que RRO, embora fundamentada em UFO, o que garante um nível alto de formalização, não possui axiomas de consolidação definidos, conforme é sugerido por SABiO, a definição de tais axiomas poderia tornar a formalização de RRO mais robusta. A terceira limitação de RRO é que a mesma não fora analisada por especialistas das áreas de ontologias e requisitos em tempo de execução externos e que consequentemente não participaram de seu processo de construção, contudo, é importante destacar que RRO foi apresentada e devidamente aceita pela comunidade científica através do artigo (DUARTE et al., 2016).

5.3 Trabalhos Futuros

Considerando o estágio atual do trabalho descrito nessa dissertação, algumas das perspectivas e possíveis trabalhos futuros são apresentados abaixo.

- Evoluir RRO por meio de uma exploração mais extensa das ontologias reusadas durante essa dissertação e, se possível, adicionar novos pontos de vista para o domínio de requisitos em tempo de execução através do reuso/combinação de novas ontologias como, por exemplo, a SRO, mencionada na Seção 2.5.

- Realizar uma formalização mais robusta em RRO através da definição de axiomas de consolidação e de derivação em lógica de primeira ordem de forma a criar restrições que devem ser satisfeitas pelas relações existentes no modelo, aumentando assim sua representatividade.

- Analisar a possibilidade/necessidade da criação de uma OPL (*Ontology Pattern Language*) para o domínio de requisitos em tempo de execução. Tal OPL seria baseada em RRO e definiria a existência de padrões a serem utilizados por engenheiros de ontologia e especialistas de domínio na criação de ontologias que levassem em consideração a utilização de conceitos relacionados ao uso de requisitos em tempo de execução.

- Utilizar RRO, como ontologia de referência e ontologia operacional, na implementação de metodologias para o desenvolvimento de sistemas adaptativos ou sistemas de monitoramento que utilizem requisitos em tempo de execução.

- Atualizar e expandir o mapeamento sistemático realizado. O mapeamento sistemático realizado como base de conhecimento para a construção de RRO foi realizado entre os anos de 2015 e 2016, devido a isso o mesmo não contempla todas as publicações sobre o assunto que ocorreram no ano de 2016 e que poderiam ser relevantes para seu resultado final. Portanto, o mapeamento pode ser expandido em um futuro próximo com o objetivo adicionar trabalhos mais recentes ao resultado final. Bibliotecas digitais diferentes das já utilizadas também podem ser incluídas para tornar o estudo ainda mais robusto.

A Mapeamento Sistemático da Literatura

Para a captura do conhecimento consensual sobre o como requisitos são usados em tempo de execução a ser utilizado como base para a construção de RRO, foi realizado um Mapeamento Sistemático da Literatura vigente sobre requisitos em tempo de execução. Um Mapeamento Sistemático é definido como um estudo científico que visa encontrar e classificar os principais estudos em um domínio específico (KITCHENHAM et al., 2009).

Kitchenham e Charters (2007) definem um mapeamento sistemático como um estudo extensivo sobre um determinado tópico com o intuito de identificar e registrar grande parte conhecimento científico disponível sobre o tema. Mapeamentos Sistemáticos geralmente utilizam as mesmas técnicas de busca e extração de dados que revisões sistemáticas de literatura porém eles estão mais comumente relacionados a pesquisas que buscam entender "como" um tópico específico existe dentro de um domínio do que "o que" existe sobre aquele domínio específico (ex: no mapeamento sistemático realizado para a construção de RRO o objetivo principal foi descobrir como requisitos eram utilizados em tempo de execução segundo a literatura vigente).

O mapeamento sistemático realizado aqui possuiu dois objetivos principais: (i) identificar e classificar propostas na literatura que fazem uso de requisitos em tempo de execução, formando assim uma base do conhecimento compartilhado sobre o domínio que seria representado na ontologia de referência de domínio; e (ii) auxiliar nas atividades de definição de escopo, de propósito e de possíveis usos da ontologia conforme é indicado pelo método SABiO.

Para a realização desse mapeamento sistemático foram adotadas as diretrizes definidas em (KITCHENHAM; CHARTERS, 2007). O processo apresentado é dividido em três fases: planejamento do estudo, condução do estudo e apresentação do estudo, sucintamente descritas a seguir.

A.1 Planejamento do Estudo

A primeira tarefa realizada nesse mapeamento sistemático foi a definição das questões de pesquisa (QPs) do mapeamento. QPs são responsáveis por conduzir todo o processo e por meio da avaliação de suas repostas serão definidos os resultados do processo. Para cada um dos artigos selecionados para o estudo, as questões são devidamente respondidas e os dados levantados são analisados qualitativa e quantitativamente ao final do estudo. A Tabela 5 apresenta as QPs que foram utilizadas no mapeamento sistemático.

A segunda tarefa da fase de planejamento do estudo foi a definição da *string* de

Tabela 5 – Questões de Pesquisa.

ID	Questão de Pesquisa	Motivação
QP1	Quando e onde os trabalhos foram publicados?	Identificar as comunidades e história da pesquisa sobre requisitos em tempo execução.
QP2	Em que tipo de Venue os trabalhos foram publicados?	Identificar os tipos de publicação encontradas sobre requisitos em tempo de execução
QP3	Quais os propósitos,encontrados na literatura, para a utilização de requisitos em tempo de execução	Identificar quais as diferentes propostas de utilização para requisitos em tempo de execução
QP4	Quais os tipos de modelos apresentado na proposta (i*, KAOS, Techne, etc.)?	Classificar as propostas segundo tipo de modelo utilizado, provendo assim uma visão de popularidade com relação a requisitos em tempo de execução.
QP5	Qual tipo de formalismo é adotado para explicar os modelos propostos?	Identificar e classificar as propostas segundo o formalismo adotado para explicar a semântica do modelo adotado. Essa questão tem como objetivo secundário identificar a utilização de ontologias como forma de formalização semântica dentro do domínio estudado.
QP6	Qual tipo de pesquisa foi feita?	Identificar o tipo de pesquisa que foi feito no estudo analisado (validação de proposta, experiência prática, avaliação de proposta, etc.). Essa questão é considerada importante pois pode ser utilizada para avaliar a maturidade do domínio estudado.

busca utilizada no mapeamento. A *string* de busca é a sentença de pesquisa que será utilizada nos bancos de de trabalhos científicos com o objetivo de obter uma lista dos trabalhos relevantes sobre o tema pesquisado.

A *string* de busca completa é composta por três partes (separadas pelos elementos AND e OR em negrito), ela garante que caso o termo "*requirements at runtime*" não apareça nos meta-dados da publicação (titulo, palavras-chave e resumo), pelo menos um termo de cada uma das outras duas partes esteja presente. Ela também leva em consideração possíveis variações na nomenclatura de termos relativos ao domínio pesquisado (Ex: *runtime* OR *run time* OR *run-time*). A *string* foi construída dessa forma para que retornasse o maior número de resultados possível mas tentando manter o escopo da pesquisa dentro das áreas de engenharia de requisitos e tempo de execução (*runtime*):

```
("requirements at runtime") OR (("requirements model" OR "requirements engineering" OR "requirements analysis" OR "requirements reasoning" OR "goal model" OR "gore" OR "goal analysis" OR "goal reasoning") AND ("runtime" OR "run time" OR "run-time")).
```

Para garantir a eficiência da *string* de busca adotada, foram selecionados artigos de controle (ACs) de forma que a *string* de busca deve ser capaz de retornar todos os artigos de controle. A escolha dos artigos de controle é feita de forma não-sistemática pelos pesquisadores que realizaram o mapeamento, os trabalhos escolhidos assim o foram por serem considerados importantes para a área a ser analisada. Os ACs escolhidos são apresentados abaixo.

- **AC1:** *Dealing with softgoals at runtime: A fuzzy logic approach*
- **AC2:** *From awareness requirements to adaptive systems: A control-theoretic approach*
- **AC3:** *(Requirement) evolution requirements for adaptive systems*
- **AC4:** *Requirements reflection: requirements as runtime entities*
- **AC5:** *Fuzzy goals for requirements-driven adaptation*

Com a *string* de busca definida, foram selecionados os bancos de artigos onde as buscas seriam realizadas. Os bancos utilizados no mapeamento sistemático foram escolhidos pois apresentaram mecanismos de pesquisa que facilitaram a utilização da *string* de busca criada e fornecerem formas mais fáceis para o *download* e agrupamento dos resultados da pesquisa. Obviamente, bancos de trabalhos científicos que permitem a realização da pesquisa mas que não permitem o *download* do artigo completo não foram utilizados. Por fim, é importante frisar que a *string* apresentada teve de ser modificada sintaticamente para atender as diferenças de cada um dos mecanismos de pesquisa utilizados, no entanto, a semântica da mesma foi mantida em todo os casos. Os bancos utilizados no mapeamento sistemático estão listados abaixo.

- ACM Digital Library (http://portal.acm.org);
- Engineering Village (http://www.engineeringvillage.com/);
- IEEE Xplore (http://ieeexplore.ieee.org/);
- Science Direct – Elsevier (http://www.elsevier.com);
- Scopus (http://www.scopus.com);
- Web of Science (http://www.webofknowledge.com).

Tabela 6 – Critérios de Inclusão e Exclusão utilizados no mapeamento sistemático

Identificador	**Critério**
IC1	A publicação deve apresentar algum tipo de informação sobre requisitos sendo utilizados em tempo de execução.
EC1	A publicação não apresenta um *abstract*
EC2	A publicação é apenas um *abstract*
EC3	A publicação não está em escrita em inglês
EC4	A publicação é uma cópia ou versão mais antiga de um trabalho já considerado
EC5	A publicação não é um estudo primário
EC6	Não foi possível obter o trabalho completo

A.2 Condução do Estudo

Primeiramente, para determinar quais artigos seriam incluídos no estudo foram criados critérios de inclusão e exclusão para analisar os resultados obtidos com a execução da *string* de busca nos mecanismos de pesquisa. A Tabela 6 apresenta os critérios de inclusão (identificador IC) e exclusão (identificador EC) adotados no estudo.

Com as etapas iniciais já resolvidas, o estudo começou a ser desenvolvido da seguinte forma: na primeira fase os artigos obtidos pelas buscas foram filtrados para a remoção de possíveis repetidos, os trabalhos selecionados (distintos) foram avaliados segundo os critérios exclusão definidos de forma que caso o trabalho se enquadrasse em um ou mais critérios de exclusão (ECs) ele era desqualificado do estudo. Após essa filtragem inicial, os artigos selecionados tiveram seus *abstracts* analisados para separar os que se encaixavam no critério de inclusão (IC1), os trabalhos que não se encaixavam eram removidos. Na terceira fase, os trabalhos que foram aprovados na analise dos *abstracts* (segunda fase) foram reavaliados segundo o IC1 dessa vez, tomando como base o texto completo do artigo, as questões de pesquisam foram então respondidas para cada trabalho lido e os aprovados formaram o resultado final do mapeamento sistemático.

A.3 Apresentação do Estudo

A Figura 36 apresenta um resumo dos resultados com números obtidos em cada etapa do mapeamento sistemático. Com um total inicial de 903 encontrados foram removidos 483 trabalhos duplicados totalizando 420 trabalhos que foram analisados através dos critérios de inclusão e exclusão até chegar no resultado final de 117 trabalhos selecionados, distribuídos no período de 1993 à 2016.

Outro resultado importante do mapeamento é o que classifica os estudos encontrados pelo ano de publicação. Tal resultado é responsável por responder a primeira questão de pesquisa além de demonstrar de forma simples a evolução da pesquisa sobre requisitos em

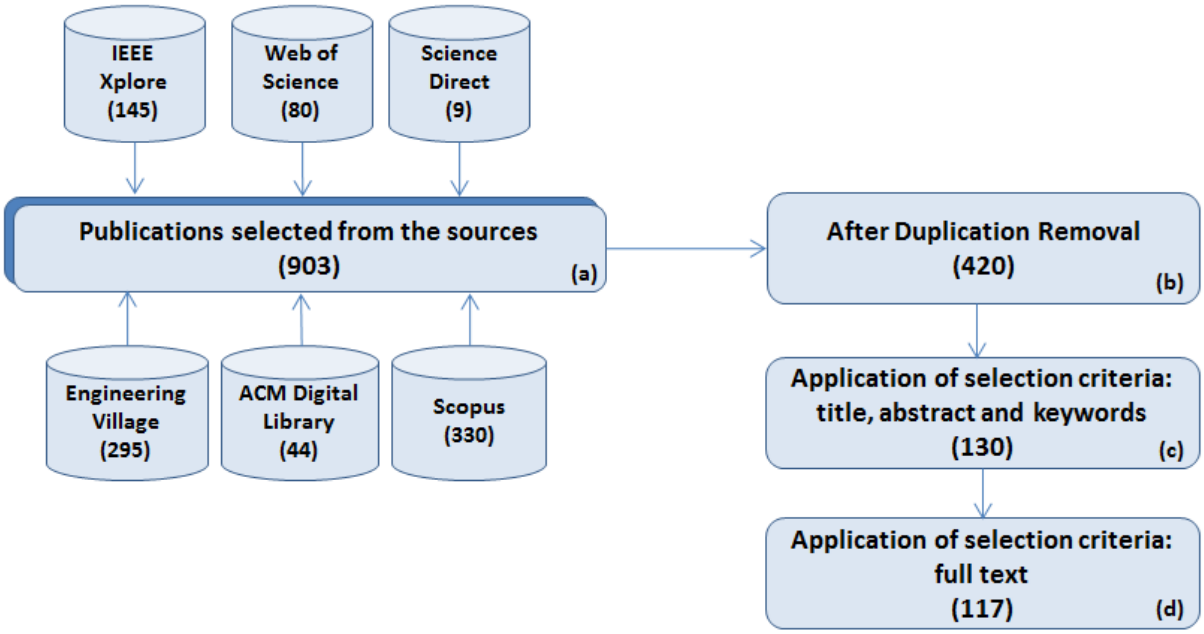

Figura 38 – Resultado da quantidade de artigos avaliados nas fases do Mapeamento Sistemático

tempo de execução ao longo dos anos. A Figura 37 apresenta a distribuição dos estudos sobre requisitos em tempo de execução ao longo dos anos. Nela podemos perceber que a maioria dos estudos sobre o uso de requisitos em tempo de execução está fortemente concentrada na última década, onde vemos um crescimento não uniforme a partir da segunda metade da primeira década dos anos 2000, com o ápice sendo atingido no ano de 2012, entretanto, também foram encontrados trabalhos no início e no fim da década de 90, mostrando que o interesse pelo tema já existia, embora fosse pouco explorado. A Figura 37 é responsável por responder a primeira questão de pesquisa (QP1), que busca verificar quando e como os trabalhos sobre requisitos em tempo de execução foram distribuídos ao longo dos anos.

Por sua vez, Figura 38 apresenta a resposta da segunda questão de pesquisa (QP2), que busca identificar em quais tipos de venue os trabalhos sobre requisitos em tempo de execução se concentram. 92 (78.63%) estudos foram publicados em algum tipo de evento (conferencias, simpósios ou *workshops*), enquanto 25 (21.37%) foram publicados em revistas.

Também é importante diferenciar os trabalhos encontrados no mapeamento sistemático pelo propósito de utilização de requisitos em tempo de execução encontrados na literatura. Este resultado em particular, foi importante pois serviu para definir diretamente os elementos que aparecem no modelo de referência de RRO. A Figura 39 apresenta os dois tipos principais de utilização de requisitos em tempo de execução percebidos durante o mapeamento sistemático: requisitos de monitoramento, que são utilizados apenas para o monitor sistemas de software e requisitos de adaptação, que são utilizados para provocar

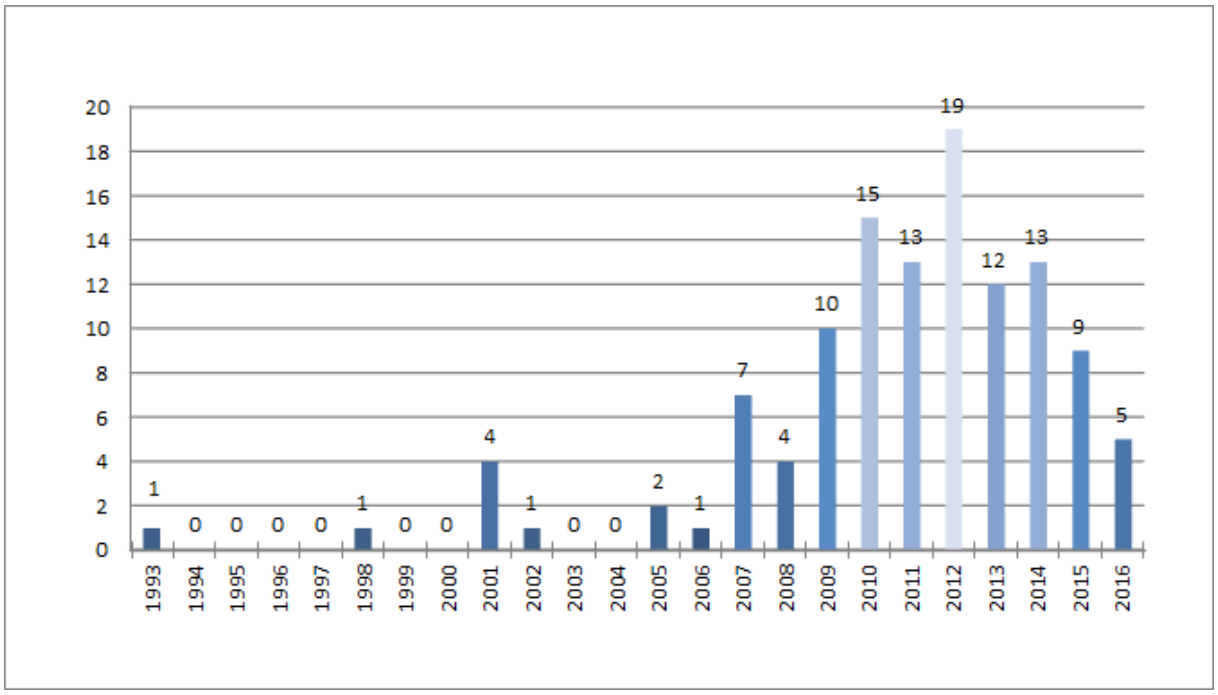

Figura 39 – Distribuição dos estudos sobre requisitos em tempo de execução ao longo dos anos

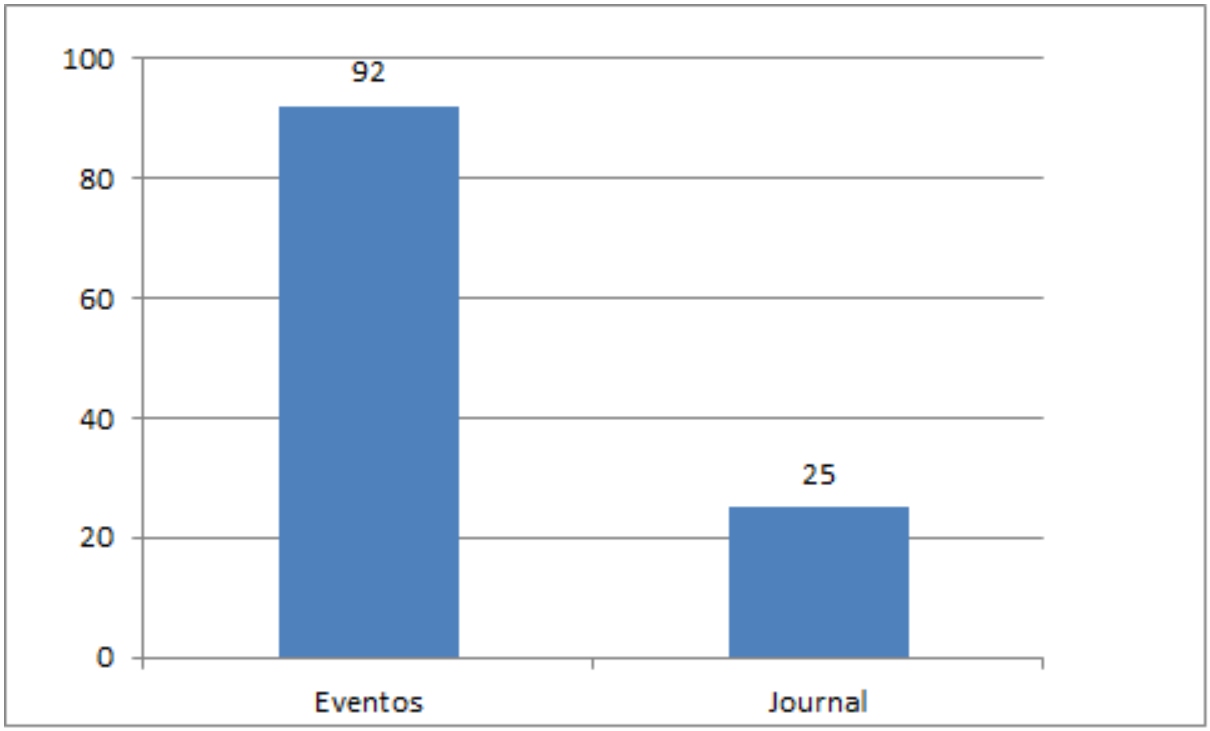

Figura 40 – Número de estudos sobre requisitos em tempo de execução por tipo de venue

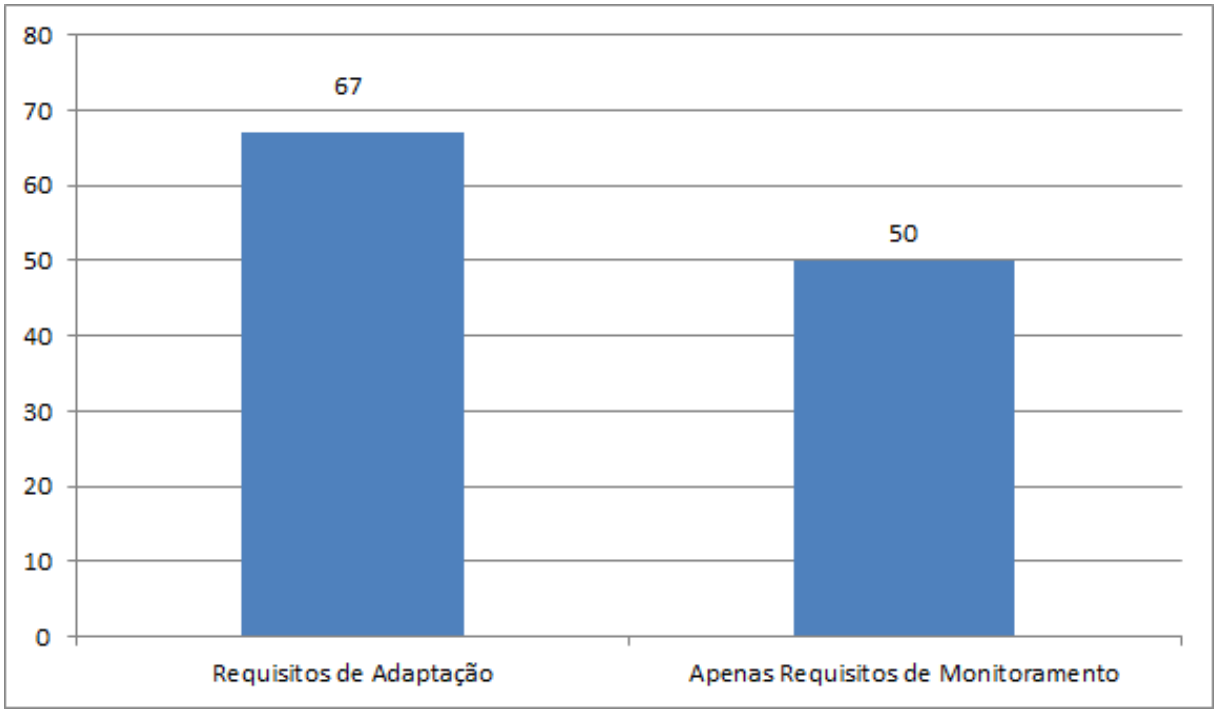

Figura 41 – Tipos de requisitos utilizados em tempo de execução encontrados no mapeamento sistemático

mudanças e adaptações nos softwares aonde são utilizados, dessa forma, respondendo a terceira questão de pesquisa (QP3), que tem como objetivo principal identificar os propósitos de utilização de requisitos em tempo de execução encontrados na literatura. A figura mostra ainda que trabalhos que tratam de algum tipo de mudança no software foram maioria no resultado, com 67 trabalhos (57.26% dos resultados), enquanto os trabalhos que utilizam requisitos em tempo de execução apenas para monitoramento totalizaram 50 artigos (42.37% dos resultados).

A Figura 40 apresenta os resultados relacionados com a quarta questão de pesquisa, que busca categorizar os trabalhos encontrados pelo tipo de modelo utilizado para representar os requisitos. Pela figura, podemos perceber uma predominância de modelos orientados a objetivos como forma de representação dos requisitos, com 48 (41.0%) dos estudos selecionados, enquanto trabalhos com modelos não-GORE aparece em segundo, com 36 (30.08%) estudos. Em terceiro, aparecem os trabalhos que não apresentaram nenhum tipo de modelo, ou que apresentaram apenas representações textuais, com 33 (28.22%) estudos.

A quinta questão de pesquisa (QP5) visa identificar as propostas da literatura conforme o tipo de formalismo utilizado para representar a semântica do modelo de representação de requisitos adotado. Dos 117 estudos selecionados na etapa final do mapeamento sistemático, 56 (47.86%) apresentaram algum tipo de formalismo para suportar seus modelos de requisitos. Essa questão possui ainda um segundo objetivo objetivo, que é identificar a quantidade de propostas que utilizam ontologias como formalismo para o

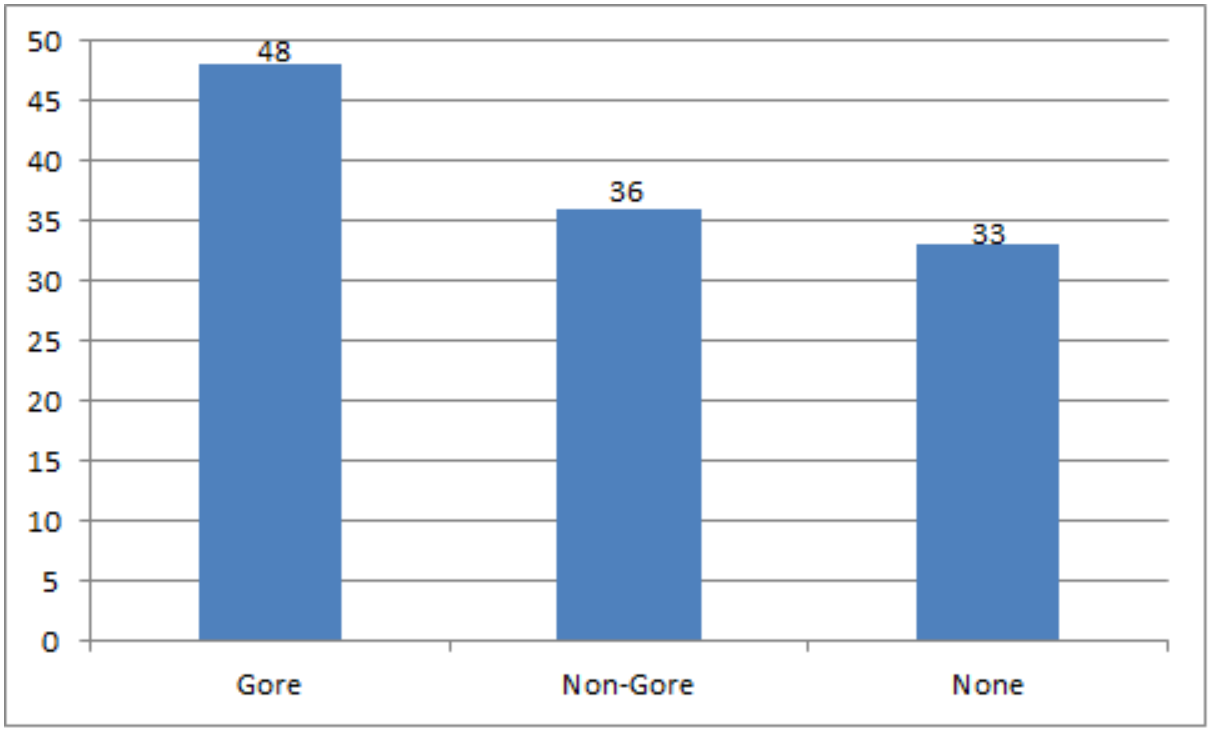

Figura 42 – Número de estudos sobre requisitos em tempo de por tipo de modelo de representação utilizado

modelo de requisitos. Tal objetivo é considerado importante, pois como a meta principal da realização do mapeamento sistemático era o levantamento do conhecimento consensual do domínio de requisitos em tempo de execução para a construção de uma ontologia de referência de domínio, conhecer as ontologias já existentes e possivelmente relacionadas com o tema se torna uma informação valiosa. Pela figura, pode-se perceber que 13 (11.11%) dos 117 estudos selecionados apresentaram ontologias como método de formalização do modelo de requisitos, resultado esse que comprova a existência de uma preocupação mais acentuada com a formalização dos modelos de requisitos em tempo de execução, ainda que não tão explorada. A distinção entre os tipos formalismos encontrados no mapeamento sistemático é apresentada na Figura 41

A Figura 42 apresenta os resultados da última questão de pesquisa (QP6) adotada no mapeamento sistemático, que trata do tipo dos estudos realizados. Dos 117 artigos analisados, a grande maioria (114 artigos, 97.4%) apresentou algum tipo de proposta para a utilização de requisitos em tempo de execução, contudo, apenas 2 deles (1,71%) apresentaram estudos com avaliações empíricas, realizadas em ambientes de escala industrial. Esse resultado tão expressivo demonstra que há um grande esforço da comunidade em propor e evoluir os métodos de utilização de requisitos em tempo de execução, porém, a pequena quantidade de propostas avaliadas empiricamente, demonstra que há uma dificuldade de utilização de requisitos em tempo de execução em cenários práticos e que os métodos e propostas existentes precisam ser mais extensivamente testados em ambientes reais. O

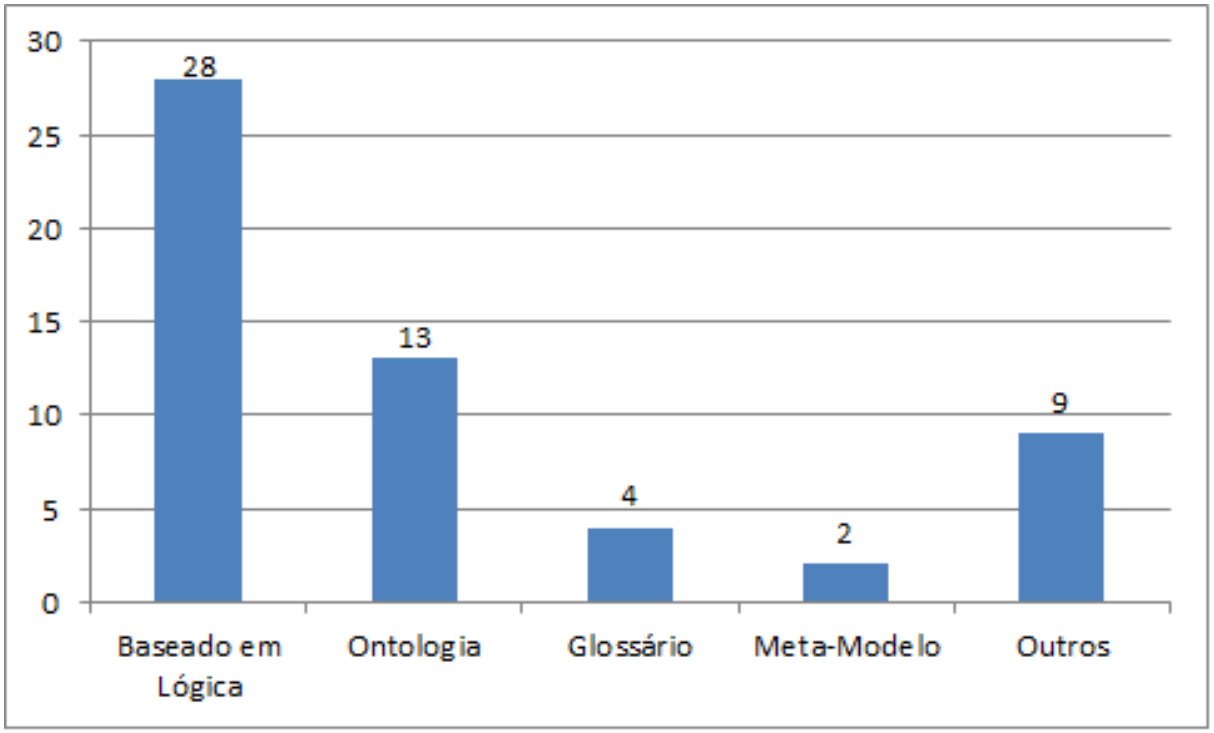

Figura 43 – Número de estudos por tipo de formalismo adotado

resultado também reafirma a importância da representação de um conhecimento compartilhado sobre o domínio, devido ao grande número de propostas distintas encontradas, demonstrada nessa dissertação na forma de uma ontologia de referência. Os 3 artigos restantes apresentaram opiniões específicas dos autores sobre o uso de requisitos em tempo de execução e devido a isso, foram classificados como artigos de opinião.

Por fim, é importante destacar que o resultado do mapeamento sistemático, com a discussão completa e detalhada dos resultados obtidos e a lista de todos os trabalhos analisados, foi submetido recentemente para o *Journal Information and Software Technology*, encontrando-se em fase de análise para aprovação.

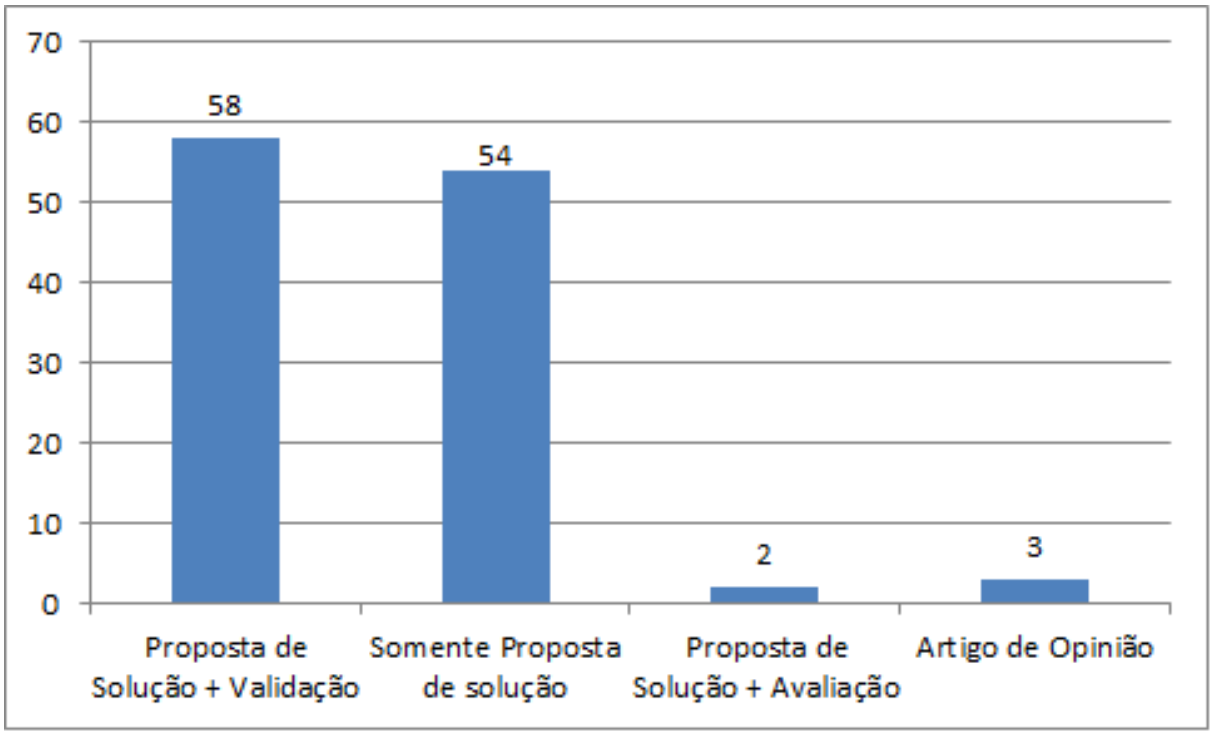

Figura 44 – Tipos de pesquisa realizados

Referências

ANGELOPOULOS, K.; SOUZA, V. E. S.; PIMENTEL, J. Requirements and Architectural Approaches to Adaptive Software Systems: A Comparative Study. In: *Proc. of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. San Francisco, CA, USA: IEEE, 2013. p. 23–32. Citado 2 vezes nas páginas 29 e 30.

ARANTES, L. D. O.; FALBO, R. D. A.; GUIZZARDI, G. Evolving a software configuration management ontology. Citeseer, 2007. Citado na página 49.

BENCOMO, N. et al. The role of models@ run. time in supporting on-the-fly interoperability. *Computing*, Springer, v. 95, n. 3, p. 167–190, 2013. Citado na página 21.

BENCOMO, N. et al. Requirements Reflection: Requirements as Runtime Entities. In: *Proc. of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE '10)*. Cape Town, South Africa: ACM, 2010. v. 2, p. 199–202. Citado na página 28.

BENCOMO, N. et al. Requirements reflection: requirements as runtime entities. In: ACM. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. [S.l.], 2010. p. 199–202. Citado na página 56.

BENEVIDES, A. B. et al. Validating modal aspects of ontouml conceptual models using automatically generated visual world structures. *J. UCS*, v. 16, n. 20, p. 2904–2933, 2010. Citado na página 71.

BORGIDA, A. et al. Requirements models for design-and runtime: A position paper. In: IEEE PRESS. *Proceedings of the 5th International Workshop on Modeling in Software Engineering*. [S.l.], 2013. p. 62–68. Citado na página 56.

BORST, W. N. *Construction of engineering ontologies for knowledge sharing and reuse*. [S.l.]: Universiteit Twente, 1997. Citado na página 33.

CHENG, B. H. C.; ATLEE, J. M. Research Directions in Requirements Engineering. In: *Future of Software Engineering (FOSE '07)*. [S.l.]: IEEE, 2007. p. 285–303. Citado na página 27.

CHENG, B. H. C. et al. (Ed.). *Software Engineering for Self-Adaptive Systems*. [S.l.]: Springer, 2009. v. 5525. (Lecture Notes in Computer Science, v. 5525). Citado na página 21.

DALPIAZ, F. et al. Runtime goal models. In: *Proc. of the IEEE 7th International Conference on Research Challenges in Information Science*. Paris, France: IEEE, 2013. p. 1–11. ISBN 978-1-4673-2914-9. Citado 3 vezes nas páginas 21, 29 e 56.

DUARTE, B. B. et al. Towards an ontology of requirements at runtime. In: IOS PRESS. *Formal Ontology in Information Systems: Proceedings of the 9th International Conference (FOIS 2016)*. [S.l.], 2016. v. 283, p. 255. Citado 2 vezes nas páginas 24 e 81.

FALBO, R. A. SABiO: Systematic Approach for Building Ontologies. In: GUIZZARDI, G. et al. (Ed.). *Proc. of the Proceedings of the 1st Joint Workshop ONTO.COM / ODISE on Ontologies in Conceptual Modeling and Information Systems Engineering*. Rio de Janeiro, RJ, Brasil: CEUR, 2014. Citado 6 vezes nas páginas 13, 23, 35, 46, 53 e 61.

FALBO, R. d. A. Experiences in using a method for building domain ontologies. In: CITESEER. *Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering, SEKE'2004. International Workshop on Ontology in Action, OIA'2004*. [S.l.], 2004. p. 474–477. Citado 2 vezes nas páginas 46 e 53.

FALBO, R. D. A.; BERTOLLO, G. A software process ontology as a common vocabulary about software processes. *International Journal of Business Process Integration and Management*, Inderscience Publishers, v. 4, n. 4, p. 239–250, 2009. Citado na página 34.

FALBO, R. d. A.; GUIZZARDI, G.; DUARTE, K. C. An ontological approach to domain engineering. In: ACM. *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. [S.l.], 2002. p. 351–358. Citado na página 33.

GRÜNINGER, M.; FOX, M. Methodology for the Design and Evaluation of Ontologies. In: *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing*. [S.l.: s.n.], 1995. Citado 3 vezes nas páginas 23, 47 e 53.

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 3 vezes nas páginas 13, 33 e 34.

GUARINO, N.; OBERLE, D.; STAAB, S. What is an ontology? In: *Handbook on ontologies*. [S.l.]: Springer, 2009. p. 1–17. Citado na página 33.

GUERSON, J. et al. Ontouml lightweight editor: A model-based environment to build, evaluate and implement reference ontologies. In: IEEE. *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*. [S.l.], 2015. p. 144–147. Citado na página 63.

GUIZZARDI, G. *Ontological Foundations for Structural Conceptual Models*. Tese (PhD Thesis) — University of Twente, The Netherlands, 2005. Citado 3 vezes nas páginas 23, 35 e 63.

GUIZZARDI, G. On ontology, ontologies, conceptualizations, modeling languages, and (meta) models. *Frontiers in artificial intelligence and applications*, IOS Press, v. 155, p. 18, 2007. Citado na página 34.

GUIZZARDI, G.; FALBO, R. de A.; GUIZZARDI, R. S. Grounding software domain ontologies in the unified foundational ontology (ufo): The case of the ode software process ontology. In: *CIbSE*. [S.l.: s.n.], 2008. p. 127–140. Citado na página 40.

GUIZZARDI, G.; FALBO, R. de A.; GUIZZARDI, R. S. Grounding software domain ontologies in the unified foundational ontology (ufo): The case of the ode software process ontology. In: *CIbSE*. [S.l.: s.n.], 2008. p. 127–140. Citado na página 49.

GUIZZARDI, G.; SALES, T. P. Detection, simulation and elimination of semantic anti-patterns in ontology-driven conceptual models. In: *Conceptual Modeling*. [S.l.]: Springer, 2014. p. 363–376. Citado na página 64.

GUIZZARDI, G. et al. Towards ontological foundations for the conceptual modeling of events. In: *Conceptual Modeling*. [S.l.]: Springer, 2013. p. 327–341. Citado na página 59.

GUIZZARDI, R. S. S. et al. An Ontological Interpretation of Non-Functional Requirements. In: GARBACZ, P.; KUTZ, O. (Ed.). *Proc. of the 8th International Conference on Formal Ontology in Information Systems*. Rio de Janeiro, RJ, Brasil: IOS Press, 2014. v. 267, p. 344–357. Citado 3 vezes nas páginas 45, 54 e 57.

HUEBSCHER, M. C.; MCCANN, J. A. A survey of Autonomic Computing—Degrees, Models, and Applications. *ACM Computing Surveys*, ACM, v. 40, n. 3, p. 1–28, 2008. Citado na página 21.

IRMAK, N. Software is an abstract artifact. *Grazer Philosophische Studien*, Rodopi, v. 86, n. 1, p. 55–72, 2013. Citado na página 57.

JACKSON, D. *Software Abstractions: logic, language, and analysis*. [S.l.]: MIT press, 2012. Citado na página 71.

JASPER, R.; USCHOLD, M. et al. A framework for understanding and classifying ontology applications. In: *Proceedings 12th Int. Workshop on Knowledge Acquisition, Modelling, and Management KAW*. [S.l.: s.n.], 1999. v. 99, p. 16–21. Citado na página 33.

JURETA, I.; MYLOPOULOS, J.; FAULKNER, S. Revisiting the Core Ontology and Problem in Requirements Engineering. In: *Proc. of the 16th IEEE International Requirements Engineering Conference*. [S.l.]: IEEE, 2008. p. 71–80. Citado na página 48.

JURETA, I. J.; MYLOPOULOS, J.; FAULKNER, S. A core ontology for requirements. *Applied Ontology*, IOS Press, v. 4, n. 3-4, p. 169–244, 2009. Citado 3 vezes nas páginas 13, 48 e 49.

KITCHENHAM, B. et al. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, Elsevier, v. 51, n. 1, p. 7–15, 2009. Citado na página 83.

KITCHENHAM, B. A.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. [S.l.], 2007. Citado 3 vezes nas páginas 23, 79 e 83.

LAMSWEERDE, A. V. et al. The kaos project: Knowledge acquisition in automated specification of software. In: *Proceedings of the AAAI Spring Symposium Series*. [S.l.: s.n.], 1991. Citado na página 31.

LAMSWEERDE, A. van. Goal-Oriented Requirements Engineering: A Guided Tour. In: *Proc. of the 5th IEEE International Symposium on Requirements Engineering*. [S.l.]: IEEE, 2001. p. 249–262. ISBN 0-7695-1125-2. Citado na página 29.

LEMOS, R. de et al. (Ed.). *Software Engineering for Self-Adaptive Systems II*. [S.l.]: Springer, 2013. v. 7475. (Lecture Notes in Computer Science, v. 7475). Citado na página 21.

MASOLO, C. et al. Dolce: a descriptive ontology for linguistic and cognitive engineering. *WonderWeb Project, Deliverable D17 v2*, v. 1, 2003. Citado na página 48.

NARDI, J. C.; FALBO, R. A. Evolving a Software Requirements Ontology. In: *Proc. of the 34th Conferencia Latinoamericana de Informatica (CLEI 08)*. Santa Fe, Argentina: [s.n.], 2008. Citado 3 vezes nas páginas 13, 49 e 50.

NARDI, J. C.; FALBO, R. de A. Uma ontologia de requisitos de software. In: *CIbSE*. [S.l.: s.n.], 2006. p. 111–124. Citado na página 48.

OMG. Ocl specification v2.4. 2014. Disponível em: <http://www.omg.org/spec/OCL/2.4>. Citado na página 71.

QURESHI, N. A.; PERINI, A. Requirements Engineering for Adaptive Service Based Applications. In: *Proc. of the 18th IEEE International Requirements Engineering Conference*. Sydney, Australia: IEEE, 2010. p. 108–111. Citado na página 28.

ROBINSON, W. N. A requirements monitoring framework for enterprise systems. *Requirements Engineering*, Springer, v. 11, n. 1, p. 17–41, 2006. Citado 7 vezes nas páginas 13, 15, 31, 32, 33, 70 e 73.

ROSS, D. T.; JR, K. E. S. Structured analysis for requirements definition. *Software Engineering, IEEE Transactions on*, IEEE, n. 1, p. 6–15, 1977. Citado na página 27.

RUI, F. B. et al. SEON: A Software Engineering Ontology Network. In: *Proceedings of the IEEE 20th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2016)*. Bologna, Italy: Elsevier, 2016. Citado 3 vezes nas páginas 13, 50 e 51.

SALES, T. P.; GUIZZARDI, G. Ontological anti-patterns: Empirically uncovered error-prone structures in ontology-driven conceptual models. *Data & Knowledge Engineering*, Elsevier, v. 99, p. 72–104, 2015. Citado 2 vezes nas páginas 64 e 66.

SOUZA, V. E. S. *Requirements-based Software System Adaptation*. Tese (PhD Thesis) — University of Trento, Italy, 2012. Citado 8 vezes nas páginas 13, 15, 21, 29, 30, 31, 70 e 72.

SOUZA, V. E. S. et al. Requirements-driven software evolution. *Computer Science - Research and Development*, Springer, v. 28, n. 4, p. 311–329, nov 2013. Citado 3 vezes nas páginas 28, 29 e 72.

SOUZA, V. E. S. et al. Awareness requirements for adaptive systems. In: ACM. *Proceedings of the 6th international symposium on Software engineering for adaptive and self-managing systems*. [S.l.], 2011. p. 60–69. Citado 2 vezes nas páginas 30 e 56.

SOUZA, V. E. S. et al. Awareness Requirements. In: LEMOS, R. et al. (Ed.). *Software Engineering for Self-Adaptive Systems II*. [S.l.]: Springer, 2013, (Lecture Notes in Computer Science, v. 7475). p. 133–161. ISBN 978-3-642-35812-8. Citado 3 vezes nas páginas 21, 29 e 72.

WANG, X. et al. Software as a social artifact: A management and evolution perspective. In: *Conceptual Modeling*. [S.l.]: Springer, 2014. p. 321–334. Citado 2 vezes nas páginas 27 e 42.

WANG, X. et al. Towards an Ontology of Software: a Requirements Engineering Perspective. In: GARBACZ, P.; KUTZ, O. (Ed.). *Proc. of the 8th International Conference on Formal Ontology in Information Systems*. Rio de Janeiro, RJ, Brasil: IOS Press, 2014. v. 267, p. 317–329. Citado 6 vezes nas páginas 13, 42, 43, 44, 54 e 57.

WHITTLE, J. et al. Relax: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering*, Springer, v. 15, n. 2, p. 177–196, 2010. Citado na página 56.

ZAVE, P.; JACKSON, M. Four Dark Corners of Requirements Engineering. *ACM Transactions on Software Engineering and Methodology*, ACM, New York, NY, USA, v. 6, n. 1, p. 1–30, jan 1997. Disponível em: <http://doi.acm.org/10.1145/237432.237434>. Citado 4 vezes nas páginas 27, 28, 45 e 48.