UNIVERSIDADE FEDERAL DE SANTA CATARINA

Uso de uma ontologia de requisitos não funcionais no suporte ao desenvolvimento de sistemas embarcados

Bruno Goulart Andrade1

Florianópolis – SC

2019/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Uso de uma ontologia de requisitos não funcionais no suporte ao desenvolvimento de sistemas embarcados

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação

Bruno Goulart Andrade1

Florianópolis – SC

2019/2

Bruno Goulart Andrade

Uso de uma ontologia de requisitos não funcionais no suporte ao desenvolvimento de sistemas embarcados

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação

Orientador: Cristian Koliver

Banca examinadora

Jean Carlo Rossa Hauck

Roberto Willrich

RESUMO

No processo de desenvolvimento de sistemas embarcados, a fase de análise de requisitos não funcionais é, no mínimo, tão importante quanto à de análise dos requisitos funcionais: enquanto uma falha no cumprimento de um requisito funcional pode comprometer parte do sistema, uma falha em cumprir um requisito não funcional pode tornar todo o sistema inútil. Por outro lado, há uma dificuldade adicional no levantamento desse tipo de requisito: é um processo intimamente ligado à experiência e conhecimento do projetista em relação ao escopo da aplicação, tornando-o processo propenso a omissões ou erros. Com o intuito de fornecer um melhor suporte à fase de análise de requisitos não funcionais quando do projeto de sistemas embarcados, este trabalho descreve uma proposta de uma ontologia de requisitos não funcionais. Essa ontologia representa os requisitos não funcionais como conceitos e mostra seus atributos, propriedades e regras, bem como seus inter-relacionamentos.

Palavras-chave: requisitos não funcionais, sistemas embarcados, ontologia, engenharia de software.

ABSTRACT

In the embedded system development process, the non-functional requirements analysis phase is at least as important as the functional requirements analysis phase: while a failure to meet a functional requirement can compromise part of the system, a failure to meet a non-functional requirement can render the whole system useless. On the other hand, there is an additional difficulty on defining this type of requirement if compared to functional requirements: such a process is closely linked to the designer's experience and knowledge of the scope of the application, making it prone to omission or error. In order to better support the non-functional requirements analysis phase when designing embedded systems, this work describes a proposal of non-functional requirements ontology. This ontology represents non-functional requirements as concepts and shows their attributes, properties and rules, as well as their interrelationships.

Keywords: non-functional requirements, embedded systems, ontology, software engineering.

SUMÁRIO

1. INTRODUÇÃO

Segundo Luigi Carro e Flávio Rech Wagner (2007), os sistemas computacionais embarcados estão presentes em praticamente todas as atividades humanas e tendem a aumentar a presença no cotidiano das pessoas. Desde sistemas de controle de veículos, máquinas de lavar roupa e micro-ondas, até os telefones celulares, todos possuem os sistemas computacionais embarcados.

O espaço de projeto arquitetural de um sistema embarcado é bastante extenso, tanto em termos de software quanto em termos de hardware. Esses sistemas podem possuir um ou diversos processadores, pouca ou muita quantidade de memória, bateria interna ou ligações elétricas, interfaces dos mais variados tipos, ligações internas entre os componentes desde um único barramento a uma rede complexa (De Micheli e Benini, 2002). Ademais, muitos deles – por exemplo, um sistema embarcado em um equipamento de manutenção cardíaco-respiratória hospitalar – são sistemas de tempo real crítico (*hard real time systems*) cuja execução exige um sistema operacional de tempo real.

Conforme Wehrmeister et al. (2007), sistemas de tempo real embarcados distribuídos (*Distributed Embedded Real-Time Systems* ou DERTS) tornaram-se muito complexos, exigindo a distribuição de funcionalidades em diferentes unidades de processamento que aproximam os serviços ao local onde eles são exigidos – por exemplo, implantar tarefas de sensoriamento e controle de cada asa rotativa de um helicóptero em pequenos processadores em vez de agrupá-los em um processador mais potente – sem violar as restrições do sistema. Essa crescente complexidade dos DERTS requer novas técnicas de desenvolvimento, a fim de melhor apoiar a evolução do sistema e reutilização de artefatos previamente desenvolvidos.

No processo de desenvolvimento de DERTS, a análise de requisitos engloba, além do levantamento dos requisitos funcionais, requisitos não funcionais (*Non-Functional Requirements* ou NFR).

Os NFRs são requisitos que não dizem respeito diretamente às funcionalidades fornecidas pelo sistema. Eles podem estar relacionados a propriedades do sistema como confiabilidade, tempo de resposta, espaço em disco, desempenho e outros atributos de qualidade do produto (PAULA FILHO, 2000). Exemplos óbvios de NFRs relacionados a DERTS incluem: métricas como o melhor tempo de execução, tempo médio e pior tempo, prazo e períodos de execução de tarefas; atrasos e variação de atraso (*jitter*); consumo de energia; alocação de memória; peso.

Em DERTS, muitas vezes os NFRs podem dizer respeito ao sistema como um todo, tornando-os tão ou mais importantes do que requisitos funcionais individuais. Se uma falha em cumprir um requisito funcional pode comprometer parte do sistema, uma falha em cumprir um NFR pode tornar todo o sistema inútil (SOMMERVILLE, 2008).

Enquanto o levantamento dos requisitos funcionais depende, majoritariamente, de informações coletadas junto ao usuário, no caso dos NFRs esse processo também depende da análise e conhecimento do projetista em relação ao escopo da aplicação e das técnicas e ferramentas que serão utilizadas no desenvolvimento do projeto, tornando-o altamente propenso a omissões ou erros devido ao desconhecimento ou desatenção dos inter-relacionamentos existentes entre os NFRs. A dificuldade é potencializada devido à inexistência de bases que subsidiem o projetista quando da especificação dos NFRs.

Vale ainda ressaltar aqui a norma ISO/IEC 9126, a qual trata da qualidade de produto de software. Ela define parâmetros para a padronização na avaliação da qualidade de um software e nela estão contempladas métricas relacionadas a requisitos não funcionais como confiabilidade, modificabilidade, estabilidade, tolerância a falhas, recuperabilidade, entre diversos outros.

Dentro desse contexto, o objetivo deste trabalho é definir uma base de consulta de NFRs com seus atributos, propriedades e inter-relacionamentos na forma de uma ontologia de aplicação.

Este documento está estruturado da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica utilizada na consecução deste trabalho. No Capítulo 3 são apresentadas a proposta e a metodologia de pesquisa usadas. O Capítulo 4 descreve o processo de criação da ontologia. O Capítulo 5 discute alguns resultados e dificuldades encontradas e apresenta sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 ONTOLOGIAS

O termo *ontologia* possui diferentes definições conforme o campo de pesquisa. Na Filosofia, esse termo é usado para designar um dos ramos da Metafísica, oriundo das combinações das palavras latinas *onto* (ser; o que é) e *logia* (discurso lógico). Nesse contexto, ontologia é o estudo filosófico do ser em geral, ou do que se aplica de maneira neutra a tudo que é real. O termo foi cunhado pelo filósofo alemão Jacob Lorhard (*Lorhardus*), que o utilizou na sua obra *Ogdoas Scholastica*, de 1606.

Apesar do termo *ontologia* ter surgido apenas no século XVII, o conceito associado foi introduzido por Aristóteles em seu Livro IV da Metafísica, mais de 300 anos a.C., na qual ele referenciava essa área de estudo como “filosofia primeira”. Mais tarde, o conceito foi popularizado pelo filósofo racionalista alemão Christian Wolff em seus escritos latinos, especialmente *Philosophia Prima Sive Ontologia* (1730; “Filosofia Primeira ou Ontologia”). Wolff contrastava a ontologia, ou metafísica geral, que se aplicava a todas as coisas, com teorias metafísicas especiais, como as da alma, dos corpos ou de Deus. Wolff afirmou que a ontologia era uma disciplina *a priori* que poderia revelar as essências das coisas, uma visão fortemente criticada no final do século XVIII por David Hume e Immanuel Kant.

No início do século 20, o termo foi adotado pelo fundador alemão da fenomenologia, Edmund Husserl, que chamou a metafísica geral de Wolff de

“ontologia formal” e a contrastou com “ontologias regionais” especiais, como as ontologias da natureza, matemática, mente, cultura e religião (SIMONS, 2015).

Segundo Devedzic (2002), as ontologias representam conceitos e suas relações, onde os conceitos são definidos e interpretados de forma declarativa, determinando os vocábulos de um domínio e as restrições de como eles podem ser combinados para modelar esse domínio. No entanto para Hepp et al. (2007), as ontologias vão além disso: elas formam contratos e são formadas a partir de processos sociais entre os participantes, os quais podem modificar, descartar ou incluir novos elementos. As ontologias permitem que o conhecimento seja explicitado socialmente (DACONTA ET AL., 2003).

A definição geralmente utilizada para o termo ontologia no âmbito da Ciência da Informação é aquela dada por T. R. Gruber (1993): “Uma ontologia é uma especificação formal e explícita de uma conceituação compartilhada”. Nessa definição, a palavra *forma*l refere-se ao fato de que a ontologia deve ser “entendível” por máquinas; *explícita* significa que o tipo de conceitos usados e as restrições em seu uso são explicitamente definidos; *conceituação* refere-se ao fato de que uma ontologia é um modelo abstrato de algum fenômeno do mundo que identifica os conceitos relevantes desse fenômeno.

Ontologias tornaram-se um tema popular a partir da década de 1990, investigado por diversas comunidades da área de Inteligência Artificial, incluindo engenharia do conhecimento, processamento de linguagem natural e representação de conhecimento, principalmente para facilitar o compartilhamento e a reutilização do conhecimento. Posteriormente, seu uso difundiu-se em áreas como integração inteligente de informações, sistemas de informações cooperativas, recuperação de informações, comércio eletrônico e gerenciamento de conhecimento.

Vale notar que os conceitos anteriores de ontologia estão intimamente relacionados ao seu uso nas diferentes áreas de pesquisa. Por exemplo, enquanto na Filosofia ontologias são usadas em problemas relativos à existência e pressupostos existenciais que surgem na lógica, na Ciência da Informação elas são usadas como um mecanismo para o entendimento comum

e compartilhado de alguns domínios entre pessoas e sistemas de Tecnologia da Informação (TI).

Neste trabalho, utilizaremos o conceito de ontologia definido por T. R. Gruber (1993).

2.2 ELEMENTOS

Conforme Kiryakov (2006), formalmente uma ontologia pode ser representada pelo conjunto $O = \{C, R, I, A\}$, onde:

• ***C*:** é o conjunto de classes que representam os conceitos em um dado domínio de interesse.

• ***I*:** é o conjunto de instâncias derivadas das classes, ou ainda, os exemplos de conceitos representados em uma ontologia;

• ***R*:** é o conjunto de relacionamentos ou associações entre os conceitos do domínio;

• ***A*:** é o conjunto de axiomas do domínio, que servem para modelar restrições e regras inerentes às instâncias.

Os elementos que constituem uma ontologia são fundamentais para a criação de uma estrutura que representa um domínio de conhecimento.

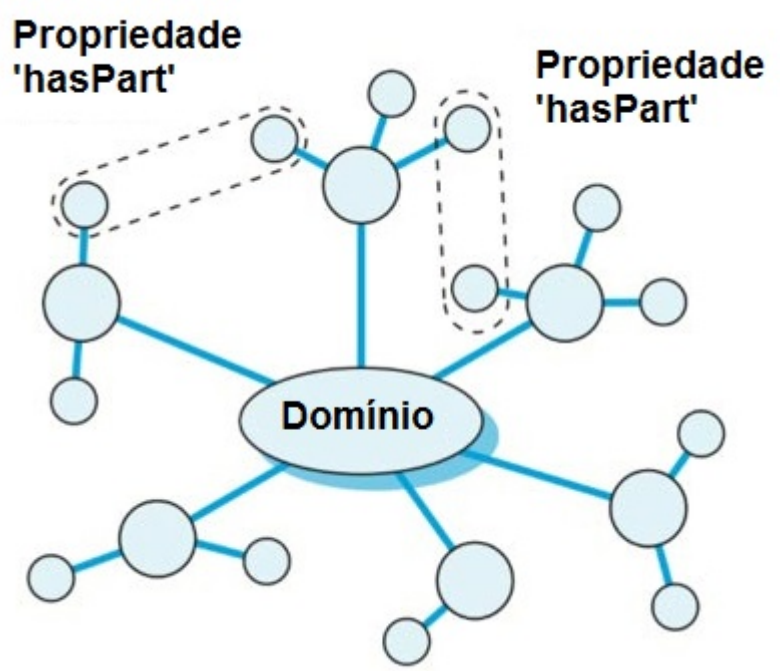

Figura 1 Ontologia simples

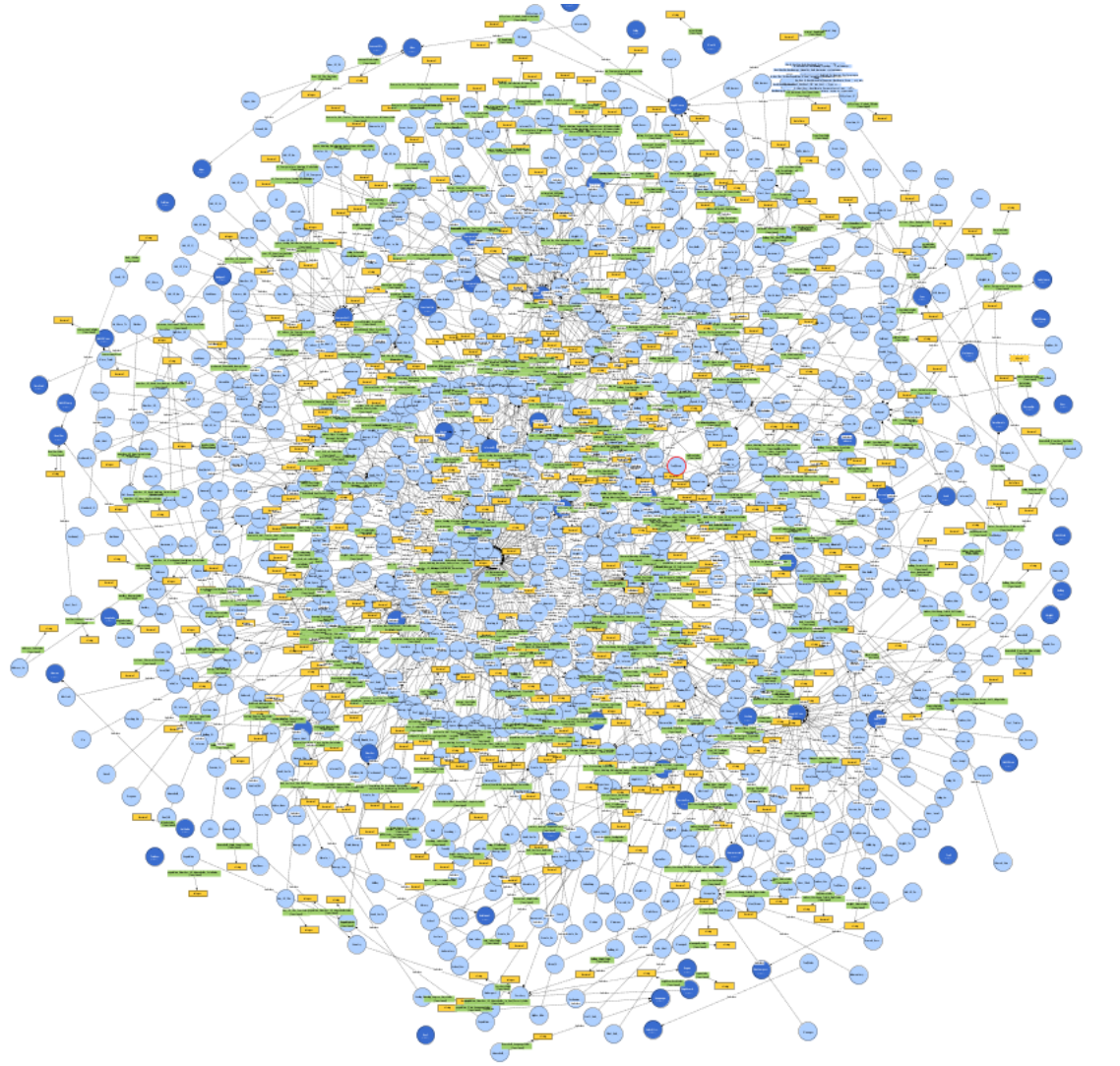

Figura 2 Ontologia complexa

As *instâncias* (também chamadas de *indivíduos* ou *objeto*s, no jargão da orientação a objetos) são concretizações das classes. Por exemplo, o indivíduo *João da Silva* pode ser uma concretização da classe *Pessoa*; o cão Lessie pode ser uma concretização da classe Cachorro. Uma ontologia não precisa, necessariamente, incluir indivíduos. Porém, um dos propósitos gerais de uma ontologia é apresentar um meio de classificação de indivíduos, mesmo que estes não façam, explicitamente, parte da ontologia.

Os *atributos* ou *propriedade*s servem para descrever características de um indivíduo. Por exemplo, na classe *Pessoa*, um atributo razoável é o nome, mas outros podem existir, como etnia, endereço, CPF, etc.

Atributos podem estabelecer *relacionamentos* entre classes quando o valor de um atributo é um objeto de outras classes da ontologia. Muito do poder das ontologias vem da habilidade de descrever estas relações. O conjunto de todas as relações descreve a semântica do domínio.

Um tipo muito comum de relacionamento é o relacionamento *is* (é) que estabelece relações hierárquicas entre classes. Por exemplo, as classes *Pai* e *Mãe* podem ter um relacionamento *is* com a classe *Pessoa*. Assim, elas são *subclasses* de *Pessoa* (que é a *superclasse* delas). Uma superclasse é uma *generalização*; uma subclasse é uma *especialização*.

Outro tipo comum de relacionamento é a do tipo *is-part* (é parte) ou *part-whole* (parte do todo) que representa uma *agregação*, ou seja, objetos se combinam para formar objetos compostos. Por exemplo, *Cabeça*, *Braço*, *Tronco* e *P*erna podem ter um relacionamento *part-whole* com *Pessoa*.

Os relacionamentos têm uma cardinalidade (por *default*, 1). No exemplo anterior, os relacionamentos *Braço* e *Perna* com *Pessoa* teriam cardinalidade dois.

Além das relações comuns como *is* e *part-whole*, as ontologias geralmente incluem outros tipos de relacionamentos que refinam ainda mais a semântica do modelo. Estas relações geralmente são específicas do domínio e são utilizadas para responder tipos particulares de questões.

A Figura 1[1] mostra uma ontologia simples representada por meio de um grafo. Nessa figura, os vértices são classes e as arestas relacionamentos do tipo *is*; os pontilhados representam outros tipos de relacionamentos. Poderíamos pensar que o domínio seria uma classe pessoa, que por sua vez agrega dois braços e eles por sua vez agregam uma mão que por sua vez agregam 5 dedos, por exemplo. A Figura 2[2] é um exemplo de uma ontologia complexa contendo centenas de classes e relacionamentos.

1 https://www.opengeospatial.org/projects/initiatives/ogctestbed12
2 https://www.opengeospatial.org/projects/initiatives/ogctestbed12

2.7 SISTEMAS DE TEMPO REAL

São considerados sistemas de tempo real aqueles sistemas que devem realizar as tarefas tendo um tempo limite para que elas aconteçam (TSAI ET AL., 1996). Estes tipos de sistemas podem possuir restrições relativas à autenticidade dos dados após um prazo definido ao realizar a tarefa (FARINES; FRAGA; OLIVEIRA, 2000).

Conforme Tsai et al. (1996), os sistemas de tempo real podem ser classificados em sistemas de tempo real *hard* e sistemas de tempo real *soft*. Os sistemas de tempo real *hard* (ou *críticos*) são altamente sensíveis aos prazos (ou *deadlines*) para a execução de suas tarefas. Se esses prazos não forem cumpridos, uma grande perda ou desastre pode acontecer. Exemplos de sistemas de tempo real do tipo *hard* são componentes de aeronaves, carros ou componentes utilizados na área médica, entre outros. Os sistemas de tempo real *soft* (ou *flexíveis*) são aqueles nos quais uma distribuição estatística de atrasos na execução das tarefas é aceitável, podendo até ocorrer perdas e danos devido a esse atraso, porém sem um impacto drástico. Exemplos de sistemas de tempo real do tipo *soft* são aplicações multimídia, como o YouTube ou a TV digital.

A previsibilidade é outro fator importante quando se trata de sistemas de tempo real. Ela refere-se à possibilidade de previsão, segundo uma série de fatores, do início e término de uma tarefa. Uma das métricas mais importantes nessa abordagem é conhecer o pior caso do tempo de execução (*worst-case time*), que permite controlar melhor esse tempo.

2.2 SISTEMAS EMBARCADOS

Um sistema é classificado como embarcado quando ele é dedicado a uma única tarefa e interage continuamente com o ambiente a sua volta por meio de sensores e atuadores (CHASE, 2007). Por exigir uma interação contínua com o ambiente, esse tipo de sistema requer do projetista um conhecimento em programação, sistemas digitais, noções de controle de processos, sistemas de tempo real, tecnologias de aquisição de dados

(conversão analógico/digital e sensores) e de atuadores (conversão digital/analógico, acionamento eletromecânico e modulação de largura de pulso - PWM), e cuidados especiais na eficiência de estruturação do projeto e do código produzido.

Todo sistema embarcado é composto por uma unidade de processamento, que é um circuito integrado fixado a uma placa de circuito impresso (CHASE, 2007). Ele possui uma capacidade de processamento de informações vindas de um software que está sendo processado internamente nessa unidade, logo o software está embarcado na unidade de processamento. Todo software embarcado é do tipo *firmware* (CUNHA, 2007).

O projeto de um sistema embarcado passa por uma sequência de níveis de abstração e é orientado a um determinado domínio de aplicação. A definição dos níveis depende de metodologias e ferramentas particulares de projeto. Cada nível permite a validação de determinadas propriedades de projeto e serve de partida para o processo de síntese para o nível inferior (HAVERINEN, 2002). Para a representação de cada domínio de aplicação, existe um modelo de computação mais adequado (EDWARDS, 1997). Diferentes linguagens de especificação e de projeto têm sido adotadas para tratamento destes níveis de abstração e modelos de computação.

O projeto de um sistema embarcado deve levar em conta, além dos fatores cruciais como o núcleo e o processador a ser utilizado e o firmware controlador, também as dimensões físicas para interação com o usuário (quando houver), o consumo de energia e o ambiente no qual será utilizado ou para qual deve ser resistente.

Os sistemas embarcados estão presentes em nosso cotidiano, dentro de milhares de dispositivos utilizados constantemente, como celulares, eletrodomésticos, carros, televisões, dentre muitos outros. A aquisição de dados é o exemplo de aplicação mais utilizada em sistemas embarcados. Para tal, sensores (temperatura, umidade, pH e outros) capturam variáveis de ambiente a serem analisadas que são gravadas em memória para consultas posteriores. Esses sistemas, além de monitorar o ambiente com adição de

atuadores ao projeto, podem ter a capacidade de controlar as variáveis de ambiente com base em um critério estabelecido pelo projetista (CHASE, 2007). No setor automobilístico, por exemplo, os veículos considerados “top de linha” possuem um complexo sistema embarcado, o qual atua em diversos pontos do carro. Centenas de sensores fornecem informações sobre todo o funcionamento do veículo. Várias unidades de processamento independentes atuam em regiões diferentes e se comunicam entre si, captando os sinais desses sensores e fazendo com que as ações referentes a cada caso sejam tomadas.

Diversos outros sistemas embarcados mais simples também estão presentes no nosso cotidiano, como os sistemas de controles remotos, micro-ondas, interfones e telefones.

2.3 VEÍCULO AÉREO NÃO TRIPULADO

Um veículo aéreo não tripulado (*unmanned aerial vehicle* - UAV) é uma aeronave sem piloto a bordo. Os UAVs podem ser aeronaves de controle remoto (por exemplo, pilotadas por um piloto em uma estação de controle de solo) ou podem voar autonomamente com base em planos de voo pré-programados ou em sistemas de automação dinâmica mais complexos. Atualmente, os UAVs são usados para várias missões, incluindo funções de reconhecimento e ataque. O acrônimo UAV foi expandido em alguns casos para UAVS (*unmanned aircraft vehicle system*) e UAS (*unmanned aircraft system*) para refletir o fato de que esses sistemas complexos incluem estações terrestres e outros elementos além dos veículos aéreos reais[3].

A Figura 3[4] mostra um exemplo de um UAV.

3https://www.theuav.com/
4https://topskytech.en.made-in-china.com/product/vXREfpzrVYUK/China-Multi-Rotor-Unmanned-Aerial-Vehicles.html

Figura 3 Exemplo de UAV do tipo sexrotor ou sexcóptero

Wehrmeister et al. (2000) observam que o conjunto de NFRs de um UAV inclui aqueles comumente identificados em DERTSs: tempo, desempenho, distribuição e incorporado. Os requisitos de tempo estão relacionados a restrições temporais das atividades do sistema embarcado do UAV (por exemplo, prazos das tarefas, períodos, custo, tempo de liberação, latência de ativação, início e fim). Requisitos de desempenho estão intimamente relacionados ao tempo e à distribuição. Eles expressam uma necessidade global de desempenho, como tempo de resposta fim-a-fim para uma determinada atividade, bem como a taxa de transferência necessária. Requisitos de distribuição estão relacionados à distribuição das atividades do sistema em diferentes nós (alocação de tarefas, hosts, comunicação, sincronização, etc.). Finalmente, preocupações relacionadas a sistemas embarcados geralmente apresentam requisitos/restrições relacionadas ao uso de memória, consumo de energia, e à área de hardware necessária.

3. PROPOSTA

O objetivo do presente trabalho é descrever o projeto e a implementação de uma ontologia de NFRs para DERTSs. Essa ontologia tem por meta auxiliar projetistas de DERTSs não só na especificação dos NFRs, mas também na identificação da inter-relação entre eles modelada na ontologia por meio de propriedades, atributos e axiomas. Essa ontologia será disponibilizada em um repositório público para que ela possa ser compartilhada e estendida.

3.1. METODOLOGIA

O levantamento dos NFRs que compõem a ontologia foi realizado por meio de pesquisa bibliográfica nos principais veículos (livros, anais de congressos e periódicos) de divulgação das áreas de sistemas embarcados e engenharia de software. A Tabela 1 mostra os artigos coletados e que fundamentaram a construção da ontologia proposta. Eles foram buscados por meio do Google Acadêmico entre 05/2019 e 10/2019. Os critérios de busca estão mostrados na coluna “palavras-chave”.

REFERÊNCIA	PALAVRAS-CHAVE	DESCRIÇÃO
Sotelo et al. (2018)	Non-Functional Requirements	Propõe uma taxonomia simples de NFRs.
Galster; Bucherer (2008)	Non-Functional Requirements	Discorre sobre a problemática da identificação de NFRs para sistemas distribuídos na fase de projeto, na qual muitas estruturas e equipamentos ainda não estão definidos. Propõe uma taxonomia para

		identificação dos mesmos e uma tabela que ajuda a identificá-los.
Ryan (2000)	Approach To Non-Functional Requirements In Software Development	Propõe uma abordagem hierárquica para facilitar a identificação dos NFRs.
Wehrmeister; Freitas, Pereira; Wagner (2007)	Non-Functional Requirements distributed embedded system	Traz uma taxonomia, bem como um diagrama de relacionamento entre os NFRs utilizados como tema de estudo.
Jingbai, T.; Keqing, H.; Chong, W.; Wei (2008)	create ontology for Non-Functional Requirements with protege	Trata da problemática da identificação e da crescente importância que vêm tomando os NFRs. Propõe uma meta-ontologia de domínio para NFRs como facilitador para encontrar esses requisitos.
Ontology Development 101: A Guide to Creating Your First Ontology	Guide to Creating Ontology	Explica, de forma pragmática, o que é uma ontologia, bem como provê um tutorial passo a passo de como construir uma ontologia

		com a ferramenta escolhida para este trabalho.
Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools	Protege pizza tutorial	Tutorial utilizado para a criação da ontologia.

Tabela 1 – Levantamento dos artigos que fundamentaram a construção da ontologia.

Dos trabalhos acima relacionados, vale ressaltar a importância de alguns, como “Ontology Development 101: A Guide to Creating Your First Ontology”, essencial para a compreensão dos elementos de uma ontologia. Também “Non-Functional Requirements distributed embedded system”, de onde foi retirado uma taxonomia na forma de uma árvore de requisitos não funcionais para DERTS. Através da pesquisa, compreendeu-se o valor que podemos agregar a uma taxonomia através dos elementos que uma ontologia oferece, os relacionamentos, restrições e regras. As taxonomias são úteis no momento de organizar um domínio de informação, com relações hierárquicas entre os conceitos. Já as ontologias, são capazes de estabelecer relações semânticas entre conceitos, chegando mais próxima da estrutura que trabalha a mente humana (VITAL, 2011).

A ontologia foi modelada utilizando-se o software Protégé (NOY ET AL., 2001). O uso dessa ferramenta deveu-se ao fato de ela já possuir uma razoável documentação, ser amplamente aceita, ser gratuita e *open source*. Além disso, o Protégé oferece uma interface gráfica e amigável para criação e edição de ontologias e possui um validador dos modelos baseados nas ontologias criadas. A ferramenta foi desenvolvida na linguagem Java, na Universidade de Stanford, sendo considerada a ferramenta líder em engenharia de ontologias. Existem muitos outros softwares que fornecem *plug-ins* para Protégé. A Figura 4 mostra a interface do Protégé; a Figura 5 mostra uma tela de um *plug-in* desenvolvido para a ferramenta para visualização gráfica da ontologia.

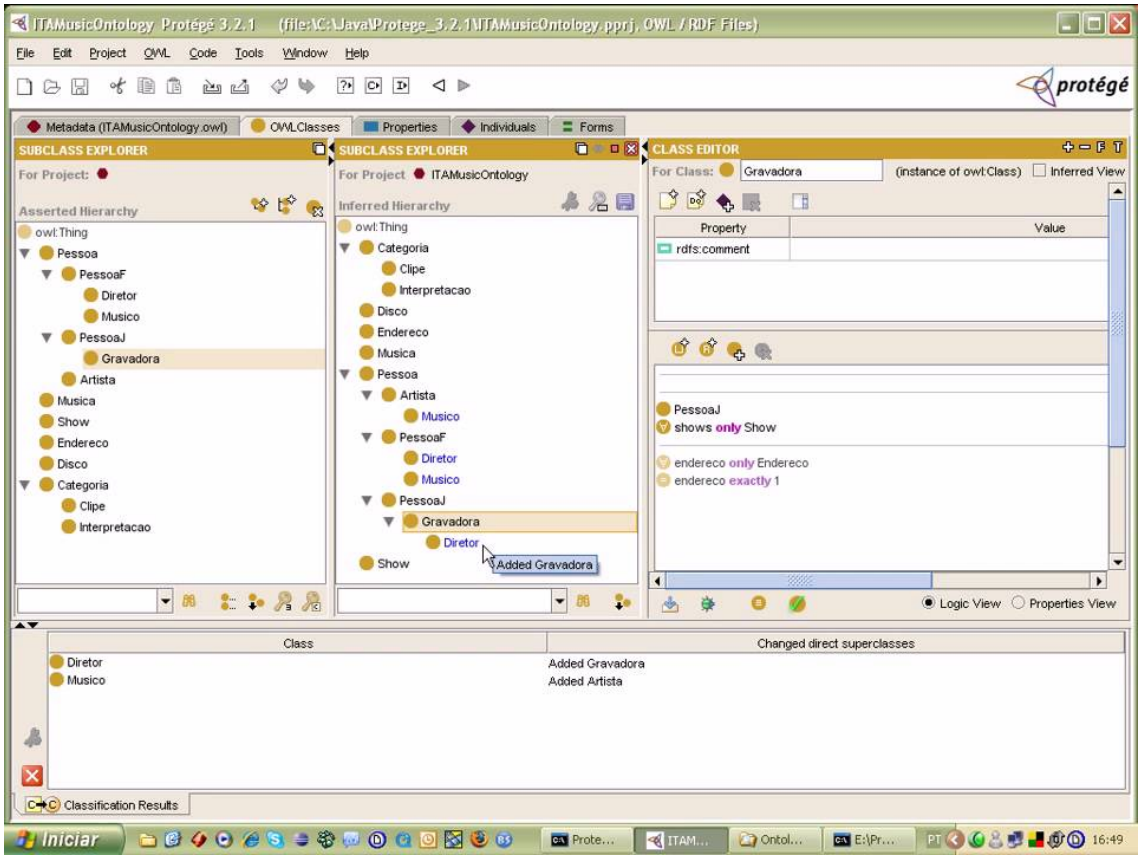

Figura 4: Software Protégé – classes e atributos.

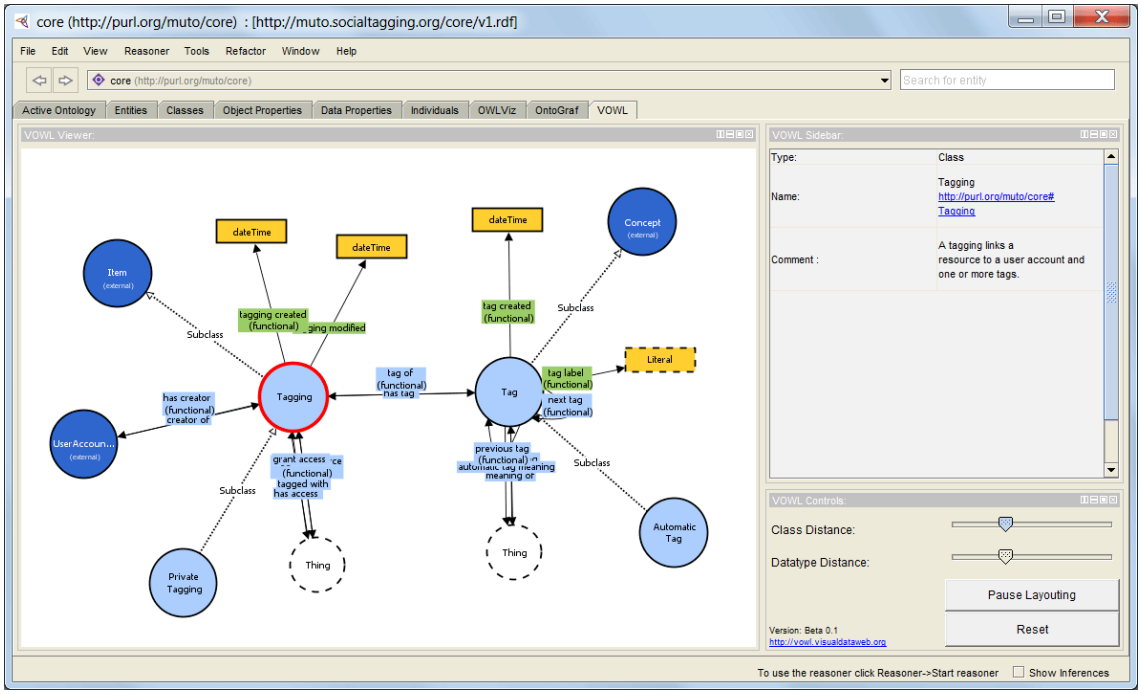

Figura 5: VOWL – Plugin para o software Protégé.

Conforme sugerem N. F. Noy e D. L. McGuinness (2001), antes da modelagem da ontologia, o projetista deve ter claro os seguintes pontos básicos:

- Qual o domínio que a ontologia se propõe a cobrir;
- Qual será o uso da ontologia;
- Para que tipo de questões a informação contida na ontologia pretende ter respostas;
- Quem usará e quem manterá a ontologia.

O domínio da ontologia proposta são NFRs para sistemas embarcados de tempo real. A ontologia cobrirá, inicialmente, NFRs de um tipo específico de sistema embarcado: UAVs. Ela será utilizada por projetistas desse tipo de sistema embarcado para a identificação dos NFRs na fase de levantamento de requisitos quando do desenvolvimento de um sistema UAV. A ontologia

resultante será disponibilizada para a comunidade em um repositório público e será mantida pelo autor deste trabalho e seu orientador.

Definidos os pontos básicos, parte-se para o levantamento de conceitos relacionados ao domínio. Nem todos os conceitos serão representados como classes: alguns podem ser representados como atributos, se sua importância no domínio é apenas marginal. Este é um processo iterativo: o levantamento dos conceitos e definição das classes e das propriedades são revistos e repensados durante a construção da ontologia.

Por fim, o trabalho se propôs a validar a ontologia com um especialista da área, para finalmente ser posta em um repositório para futura extensão.

4. DESENVOLVIMENTO DA ONTOLOGIA

Para o levantamento dos conceitos (principalmente NFRs) usados como base da ontologia criada neste trabalho, utilizamos algumas taxonomias e levantamentos de NFRs para sistemas embarcados encontrados na literatura, particularmente os trabalhos de: Sotelo et al. (2018); Galster e Bucherer (2008); Ryan (2000); Wehrmeister, Freitas, Pereira e Wagner (2007a); Wehrmeister, Freitas, Pereira e Wagner (2007b); Jingbai, Keqing, Chong e Wei (2008); Leite 2014). Aqui cabe ressaltar a diferença entre uma ontologia e uma taxonomia: enquanto uma taxonomia apenas identifica os relacionamentos entre itens e categorias, uma ontologia, além de poder envolver várias taxonomias que podem ser usadas para descrever um domínio de conhecimento, descreve os relacionamentos intra e intertaxonomias. Assim, uma ontologia é especializada em relacionamentos entre itens e categorias e nas complexidades existentes entre eles. A taxonomia identifica os relacionamentos, mas não possui a complexidade que a ontologia fornece para que se possa identificar relacionamentos e restrições mais explícitas e formais, podendo até serem mapeadas para a geração de código de uma linguagem de programação utilizada em desenvolvimento de DERTS, como o AADL.

OWL (Ontology Web Language) é a linguagem utilizada para definir e instanciar ontologias na World Wide Web. Esta linguagem foi criada para que o conteúdo da informação contida na ontologia possa ser processado e não apenas apresentado a humanos. Nossa ontologia trabalha com este formato.

A seguir, partimos para o agrupamento desses conceitos em categorias para, posteriormente, refinarmos em subcategorias. A maioria das categorias foi mapeada para classes; algumas poucas, por decisão de projeto, foram mapeadas para atributos de classes. Paralelamente à definição das classes, foram definidos os relacionamentos existentes entre elas e também os axiomas.

A construção da ontologia utilizando o Protégé foi subsidiada, principalmente, pelos seguintes trabalhos: Horridge et al. (2004); Horrocks et al.

(2004); Jingbai et al. (2008); Noy e McGuinness (2001); Noy et al. (2000); França (2009); Sachs (2006).

4.1. ESTRUTURA DE CLASSES E SUBCLASSES

Esta seção descreve as principais superclasses da ontologia proposta, hierarquicamente organizadas conforme a Figura 6.

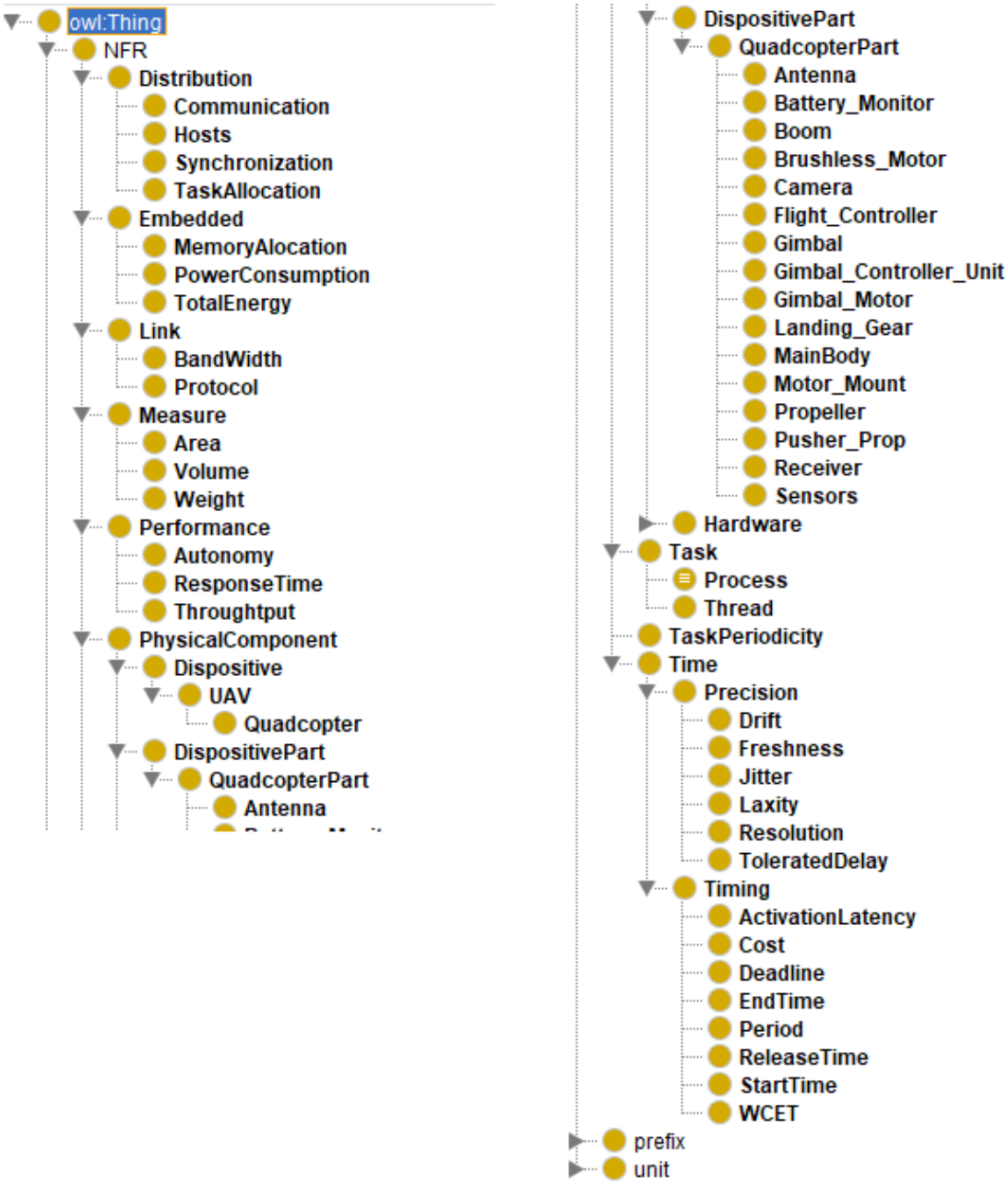

Figura 6 Hierarquia de classes da ontologia

4.1.1 Distribuição (*Distribution*)

São NFRs relacionados à disposição dos componentes lógicos e físicos do UAV no sistema. São eles:

- **Comunicação (*Communication*):** esta classe engloba as tecnologias utilizadas para comunicação, como tipos de protocolos, vazão de troca de mensagens, largura de banda, garantia ou não de entrega de pacotes, etc.

- **Estações Participantes (*Hosts*):** se relaciona com a checagem e verificações do estado em um determinado momento das estações do sistema, como o transmissor, a central de controle e o veículo em si. Também auxilia na geração das restrições na alocação de tarefas.

- **Sincronização (*Synchronization*):** responsável pelas políticas de acesso aos recursos do sistema.

- **Alocação de Tarefas (*TaskAllocation*):** é o requisito responsável pela distribuição e escalabilidade das tarefas para as diferentes estações participantes. Por exemplo, tratando-se de segurança, pode definir em qual estação um processamento deve ser executado preferencialmente.

4.1.2 Embarcados (*Embedded*)

Engloba restrições devidas ao fato do sistema ser um sistema embarcado. Os NFRs associados a essa categoria são:

- **Alocação de Memória (*MemoryAllocation*)**: restringe o uso e alocação de memória para as diferentes tarefas do sistema.
- **Consumo de Potência (*PowerConsumption*)**: rege o consumo de potência que as tarefas podem obter em suas atividades.
- **Energia Total (*TotalEnergy*)**: este conceito define o quanto de energia deve ser disponibilizada para o sistema, quanto deve ser a duração dessa energia para o fim para o qual o sistema foi

desenvolvido. Pode ser uma medida temporal, distância que se deseje alcançar ou até mesmo por número de tarefas a realizar.

4.1.3 Ligação (*Link*)

Esta classe define o tipo de protocolo e a largura de banda para a comunicação do UAV com a estação base. Inicialmente, possui apenas dois NFRs:

- **Largura de Banda (*BandWidth*)**: este conceito define a taxa de transferência a ser utilizada em cada operação.
- **Protocolo (*Protocol*)**: relaciona os tipos de protocolos a serem utilizados nas comunicações.

4.1.4 Medida (*Measure*)

Refere-se a NFRs relacionados com medidas. Engloba os seguintes NFRs:

- **Área (*Area*)**: restringe a máxima área que pode ser ocupada por um componente físico do sistema, como uma placa de hardware ou uma entrada de carregamento;
- **Peso (*Weight*)**: restringe o peso de um componente de hardware ou que componha o corpo, controle ou diferentes partes do UAV e dos componentes físicos do sistema;
- **Volume (*Volume*)**: restringe o volume de um componente de hardware ou que componha o corpo, controle ou diferentes partes do UAV e dos componentes físicos do sistema.

4.1.5 Desempenho (*Performance*)

Envolve NFRs de desempenho de recursos do sistema. São eles:

- **Vazão (*Throughtput*)**: é a taxa computacional na qual um recurso deverá executar sua atividade ou a quantidade de requisições que

um recurso deve conseguir atender de chamadas por unidade de tempo;

- **Tempo de Resposta (*ResponseTime*)**: representa o tempo necessário para que, executando atividades locais ou remotas, o recurso retorne uma resposta final.

4.1.6 Componente Físico (*Physical Component*)

Descreve NFRs relacionados a componentes como processador, memória e ao próprio dispositivo. Neste último caso, contém também as partes físicas que compõem o dispositivo, relacionando-se a ele por meio de uma agregação (relacionamento *part-whole*). São subclasses:

- **Dispositivo (*Dispositive*)**: um dispositivo qualquer com um sistema embarcado. Uma especialização é um UAV, que, por sua vez, pode ser especializado em um tipo de UAV específico, como um quadcóptero (helicóptero de quatro hélices);
- **Partes do Dispositivo (*DispositivePart*)**: é refinado em partes específicas de um dispositivo específico. Por exemplo, a subclasse *QuadcopterPart* define as diferentes partes da estrutura física do quadcóptero descritas em https://www.dronezon.com, sendo elas: *antenna, batery monitor, boom, brushless motor, camera, flight controller, gimbal, gimbal controller unit, gimbal motor, landing gear, main body, motor mount, propeller, pusher prop, receiver, sensors*.

- ***Hardware***: subdivide-se em hardwares específicos. Por exemplo, *Memory* e *Processor*, que também podem ser refinados. Por exemplo, *VolatileMemory* e *NonVolatileMemory* (*SSD e HD*).

4.1.7 Tarefa (*Task*)

Descreve as tarefas do tipo processo e thread do sistema, com atributos (por exemplo, periodicidade – periódica, aperiódica e esporádica).

4.1.8 Tempo (*Time*)

Esta classe abrange os NFRs de natureza temporal das tarefas, bem como define métricas de precisão que cada tarefa deve obedecer para que haja um bom funcionamento operacional. Suas duas subclasses são descritas a seguir.

4.1.8.1 Temporização (*Timing*)

Os NFRs de temporização possuem relativa facilidade de compreensão e são diretamente mapeados em propriedades para a execução das atividades. Eles geralmente são específicos por tarefa, não permeando diversas atividades conjuntamente. Os requisitos de temporização são:

- **Latência de ativação (*ActivationLatency*)**: é o tempo entre a liberação de um comando e o momento em que ele realmente começa a ser executado.
- **Custo ou Tempo de Execução (*Cost*)**: é a média do tempo para executar uma atividade específica no sistema.
- **Prazo (*Deadline*)**: é o limite máximo que uma atividade pode levar para executar.
- **Período (*Period*)**: corresponde ao intervalo entre duas ativações sucessivas de uma tarefa periódica.
- **Instante de Liberação (*ReleaseTime*)**: corresponde ao momento em que o sistema deve estar pronto para executar uma atividade específica no sistema.
- **Inicio e Fim (*StartAndEnd*)**: momento de início e fim de uma atividade.
- **Pior tempo de execução (*Worst Case Execution Time* – WCET)**: tempo de execução de uma tarefa no pior caso.

4.1.8.2 Precisão

Esses NFRs são mais abrangentes do que os requisitos de temporização, podendo permear diversas propriedades, por exemplo, o *Retardo Admitido* pode se tratar de um processo em específico, bem como pode se tratar do atraso de uma resposta final do sistema. Os requisitos de precisão são:

- **Exatidão (*Drift*)**: corresponde à diferença máxima entre o tempo físico e o tempo lógico utilizado pelo sistema.
- **Utilidade (*Freshness*) ou Prazo de Validade**: diz respeito à validade de um dado. Se um dado é enviado em um momento x, esse dado pode perder sua validade depois de decorrido certo tempo.
- **Variação do atraso (*Jitter*)**: relaciona-se com a variação e atraso na entrega de dados. O sistema se torna menos preciso quanto maior for o *jitter*.
- **Rigidez (*Laxity*)**: define uma precisão para outras classes. Certos dados gerados tem uma utilidade que se dá por certo prazo. À medida que o prazo é esgotado, o dado vai perdendo a utilidade. Esse prazo pode ter uma rigidez *hard* ou *soft*, de forma análoga aos *deadlines* das tarefas.
- **Resolução (*Resolution*)**: é a granularidade mais fina que um elemento de temporização pode trabalhar. Por exemplo, podemos definir o *deadline* de uma tarefa *x* com período de granularidade milissegundos ou nano segundos.
- **Retardo Admitido (*ToleratedDelay*)**: tempo máximo admitido para iniciar a execução de uma atividade.

4.2 RELACIONAMENTOS

Os principais relacionamentos (chamados de *Object Properties* na nomenclatura do Protégé) são mostrados na Figura 7. O relacionamento *has* é refinado utilizando uma nomenclatura na qual o *string* após *has* indica o *range*

do relacionamento. À exceção do relacionamento *hasPart*, usado para representar agregação (vide adiante), todos os demais têm como domínio *Task*. Assim, por exemplo, *Task hasPeriod Period*.

Os relacionamentos *isMeasuredBy* associam classes da nossa ontologia a classes de uma ontologia de quantidades e unidades de medida descrita por Rijgersberg, Van Assem e Top (2013) e disponibilizada em https://github.com/HajoRijgersberg/OM. Este trabalho reusa essa ontologia para criar relacionamentos de unidades de medida. A Figura 8 mostra um fragmento dessa ontologia. Alguns relacionamentos são:

- *Period isMeasuredByTimeUnit time unit*;
- *Area isMeasuredByAreaUnit area unit*;
- *PowerConsumption isMeasuredByPowerUnit power unit*, etc.

4.2.1 Agregação

Conforme dito na Seção 4.1.6, utilizamos relacionamentos para representar agregação, associando um dispositivo específico aos componentes que o compõem. Diferentemente do Diagrama de Classes UML, o Protégé (e ontologias, de um modo geral) não possui uma forma padrão para representação de agregação[5]. Neste trabalho, utilizamos a recomendação do World Wide Web Consortium (W3C), a principal organização de padronização da Web e disponível em

https://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/

Nessa recomendação, a agregação é representada por dois relacionamentos inversos, *partOf* e *hasPart*. Por exemplo:

Quadcopter hasPart only

(*Antenna* or *Battery_Monitor* or *Boom* or ... or *Receiver* or *Sensors*)

Antenna partOf some *Quadcopter*

5 Cabe ressalvar que existem diferentes tipos de agregação. Neste trabalho, usamos este termo para designar aquele tipo referenciado como *part-whole aggregation*.

Battery_Monitor partOf some *Quadcopter*

Boom partOf some *Quadcopter*

...

Receiver partOf some *Quadcopter*

Sensors partOf some *Quadcopter*

As expressões acima representam restrições. Restrições mais complexas podem ser definidas por meio de axiomas, conforme veremos a seguir.

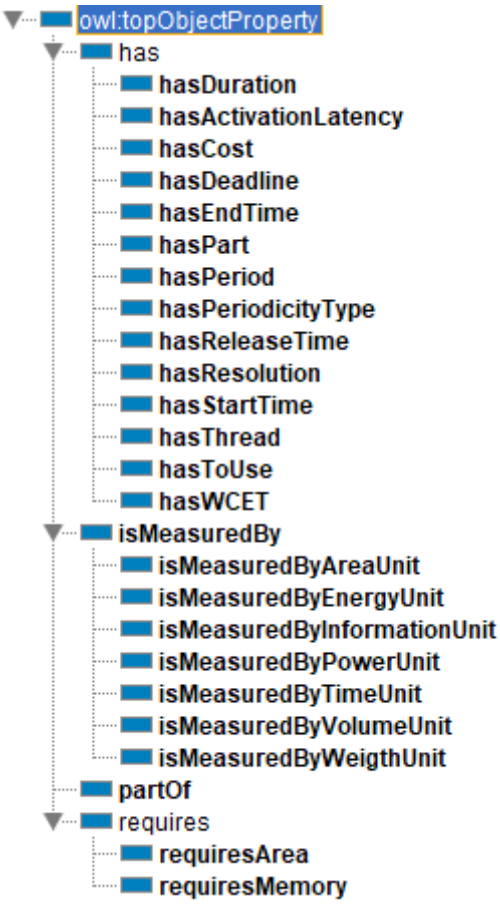

Figura 7 Relacionamentos

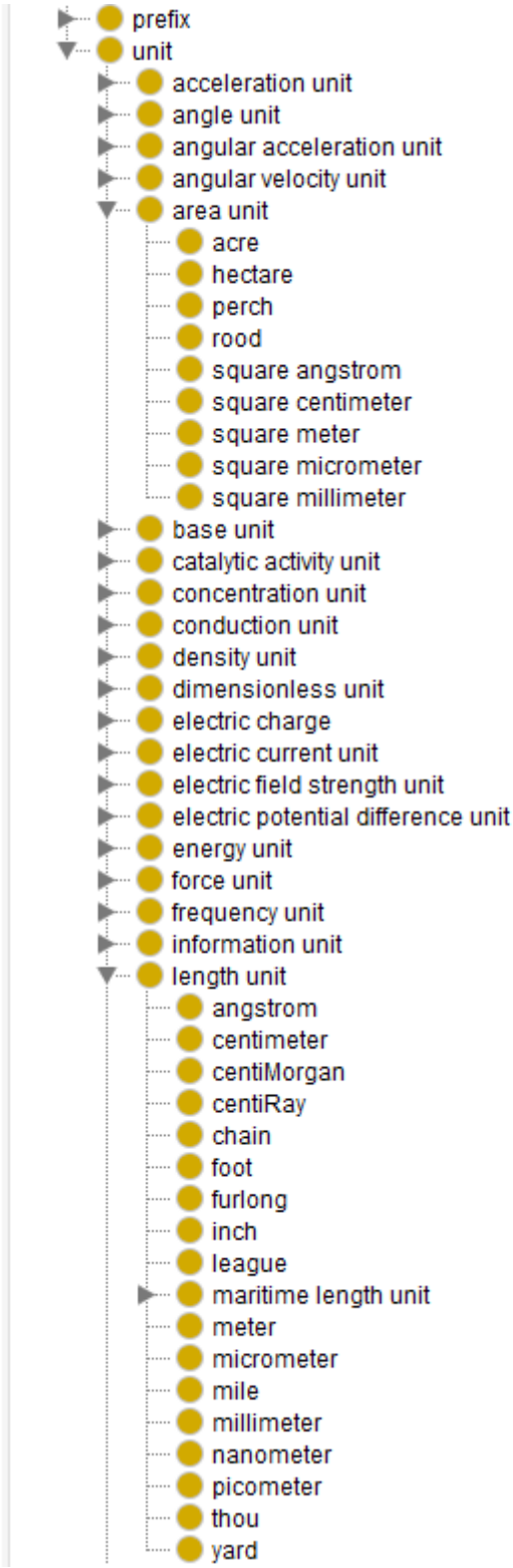

Figura 8 Ontologia de medidas

4.3 AXIOMAS

Os axiomas criados baseados nas classes e nas propriedades restringem e definem os relacionamentos entre os elementos do sistema. No Protégé existem algumas palavras reservadas para essas descrições. Por exemplo, dizendo que uma classe (lado esquerdo) se relaciona por meio de uma propriedade com outra classe (lado direito) por meio da palavra reservada *some*, significa uma restrição existencial, ou seja, a classe da esquerda tem, pelo menos, uma relação por meio dessa propriedade com a classe da direita. Na nossa ontologia, inserimos alguns axiomas utilizando a notação nativa do Protégé, que possui algumas palavras reservadas para a sua construção e posteriormente pode ser exportada em diferentes formatos, como OWL, e, no caso de axiomas mais complexos, por meio da linguagem de regras da Web semântica (*Semantic Web Rule Language* - SWRL), é uma linguagem proposta pela W3C que pode ser usada para expressar regras e lógica.

Alguns axiomas já foram mostrados na Seção 4.2. Não pretendemos aqui fazer uma demonstração exaustiva de todos os axiomas de nossa ontologia, mas apenas de um subconjunto que mostre o potencial de representatividade da ontologia e da linguagem utilizada. Estas regras servem para auxiliar o projetista a relacionar os diferentes conceitos no momento da construção do dispositivo. Por questões de legibilidade, exemplificaremos algumas dessas regras de maneira semiformal, conforme abaixo:

- Todo *Process*.p tem ao menos uma *Thread*.t;
- Se *Task*.t é periodic então *Task*,t tem *Period*.p;
- Toda *Task*.t tem *WCET*.w;
- *Period*.p ≥ *Deadline*.d;
- *Deadline*.d ≥ *WCET*.w
- Se *Battery*.b *partOf Dispositive*.d e *Battery*.b *hasDuration Time*.t e *Dispositive*.d *hasAutonomy Autonomy*.a então *Autonomy*.a ≤ *Time*.t.

4.4 ATRIBUTOS

Nossa ontologia contém poucos atributos (*Slots* ou *Data Properties* na nomenclatura do Protégé). Todos eles são utilizados para representar valores de tipos de dados básicos (inteiro, real, *string*) para indivíduos de algumas classes. Por exemplo, *área-value*, *bandwidth_value*, *deadline_value*, *period_time*, *deadline_time*, etc.

4.5 VALIDAÇÃO

Do ponto de vista formal, a ontologia foi validada utilizando o *plugin* HermiT do Protégé. O HermiT é um racionalizador (*reasoner*) semântico, um software capaz de inferir consequências lógicas de um conjunto de fatos ou axiomas declarados. De maneira mais específica, o *reasoner* permite[6]:

- Verificar a satisfação de um conceito: determinar se uma descrição do conceito não é contraditória, ou seja, se um indivíduo pode existir, ou seja, ser uma instância de uma classe da ontologia;
- Subsumir conceitos: determinar se o conceito *C* substitui o conceito *D*, isto é, se a descrição de *C* é mais geral que a descrição de *D*;
- Consistir *A* em relação a *T* – determine se o indivíduo *A* não viola as descrições e axiomas descritos por *T*;
- Verificar se um indivíduo é uma instância de um conceito;
- Recuperar indivíduos: encontrar todos os indivíduos que são instâncias de um conceito;
- Realizar um indivíduo: encontrar todos os conceitos aos quais o indivíduo pertence, especialmente os mais específicos.

6 https://www.obitko.com/tutorials/ontologies-semantic-web/reasoning.html

4.5.1 Parecer de um especialista da área

A validação da ontologia, do ponto de visita do domínio da aplicação, foi realizada pelo professor Leandro Buss Becker[7], do Departamento de Automação (DAS) e Sistemas da UFSC. O professor Becker possui mestrado e doutorado na área de Desenvolvimento de Aplicações Tempo Real. Atualmente ele participa de dois projetos de pesquisa nesta área: VANT3D - Inspeção Óptica Tridimensional por Veículo Aéreo Não Tripulado; e Desenvolvimento de um Veículo Aéreo Não Tripulado Convertível com Fontes de Energia Renovável.

Para realização da validação, apresentamos para ele a ontologia criada, mostrando seus elementos e explicando os papéis deles nessa ontologia. O professor Becker concordou com os conceitos, relacionamentos e axiomas, apontando, também, novos conceitos ou refinamentos de conceitos existentes relacionados a NFRs que também seriam relevantes no projeto de um UAV e que deveriam, portanto, ser acrescentados na ontologia proposta para este trabalho. As alterações sugeridas pelo professor Becker são detalhadas a seguir.

Na classe *Timing* foram refinados os tempos de execução das tarefas: em vez de apenas constar o tempo de execução no pior caso, foi criada uma subclasse *ExecutionTime* refinada em *WorstCaseExecutionTime*, *BestCase ExecutionTime* e *AverageCaseExecutionTime*.

O conceito de Sistema Operacional (*OperatingSystem*) também foi acrescentado, o qual define, além do sistema operacional a ser usado, se ele não é de tempo real (*NonRealTime*), ou se é de tempo real crítico (*RealTimeHard*) ou não crítico (*RealTimeSoft*). Esse sistema se liga a classe *ActivationLatency* por meio da propriedade *hasActivationLatency,* fazendo da latência de ativação uma propriedade do sistema operacional. Outro conceito acrescentado relacionado a esta nova classe, mas também à classe preexistente *Task,* foi o de política de escalonamento (*SchedulingPolicy*), já

7 Currículo lattes em http://lattes.cnpq.br/4970009027814810

que diferentes tarefas, conforme sua criticidade, são alocadas na CPU por diferentes escalonamentos. Mais um conceito relacionado às tarefas é o de prioridade (*Priority*), utilizado por escalonadores baseados em prioridades estáticas ou dinâmicas.

A classe *Processor* foi refinada em duas subclasses: *Capability* (Capacidade) e *Frequency* (Frequência), essa última, tendo uma unidade de medida de frequência, através da propriedade (*hasFrequency only 'frequencyUnit'*).

A classe *Brushless* (Hélices) (subclasse de *DispositivePart*) passou a ter um relacionamento de cardinalidade *um-para-um* com *Brushless_Motor* e *Esc* (nova classe criada para representar os conectores da hélice). Aos quadcópteros criou-se um axioma que liga ele às hélices com uma cardinalidade de 4 e aos dicópteros, classe identificada também nesta validação, com cardinalidade de 2.

Também a classe *Camera* foi subdividida em duas subclasses: *NavigationCamera* e *MovieCamera*, as quais representam, respectivamente, a câmera de navegação, usada pelo piloto, e uma possível câmera de filmagem.

A classe *Sensor* foi refinada em *Accelerometer, Gyroscope, Magnetometer*, para posterior extensão e utilização como restrição para outros conceitos.

A Figura 9 mostra a estrutura final de classes (à esquerda) e um exemplo de restrição por atributo (à direita).

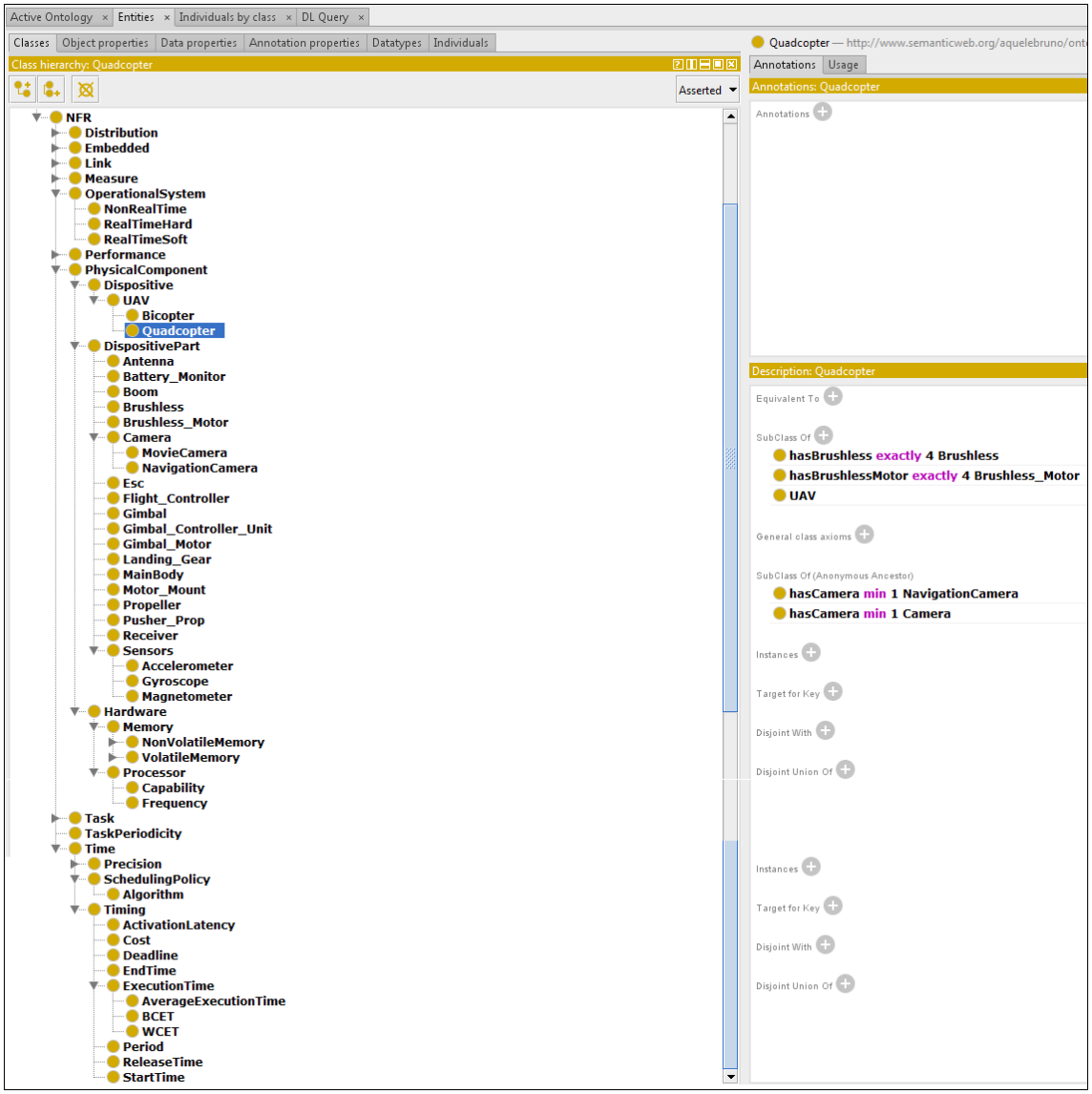

Figura 9 Estrutura final de classes

5. CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho descrevemos uma ontologia para representação de NFRs de sistemas embarcados. A despeito de algumas propostas de taxinomias de NFRs encontradas na literatura, desconhecemos a existência de uma ontologia de NFRs como a aqui proposta.

A construção de uma ontologia é um processo iterativo e interativo. Iterativo porque os projetistas podem começar com substantivos e verbos do domínio de conhecimento para esboçar a estrutura geral e, em seguida, realizarem várias iterações para refinar e corrigir a estrutura; interativo porque a construção de ontologias de maneira colaborativa e cada vez mais voltada para a comunidade tornou-se um paradigma central da moderna engenharia de ontologias. Assim, esperamos que a ontologia de domínio de NFRs descrita neste trabalho possa servir como auxílio para projetistas de DERTS, bem como possa ser incrementada e melhorada, para ser, de fato, utilizada como suporte para a fase de análise de requisitos quando do projeto e desenvolvimento de um sistema embarcado.

A proposta inicial do trabalho contemplava não só a construção da ontologia, mas também o seu mapeamento, via *Model-Driven Engineering* (MDE), para propriedades AADL (*Architecture Analysis and Design Language*), uma linguagem amplamente usada para descrever a arquitetura de sistemas embarcados em tempo real em termos de software e hardware, e descrita por Feiler, Gluch, e Hudakmde (2006). Todavia, o *Software Engineering Institute* (SEI) da *Carnegie Mellon University*, responsável pelo desenvolvimento e manutenção da linguagem, removeu todos os links para download do metamodelo AADL, absolutamente necessário para realização do mapeamento OWL2AADL via MDE (provavelmente em virtude de sua parceria com o Departamento de Defesa dos Estados Unidos – DARPA – para uso militar da linguagem).

Assim, uma proposta de trabalho futuro envolve a criação de um metamodelo restrito do AADL englobando apenas a descrição das propriedades e o seu uso na criação de um tradutor OWL-AADL para realizar o

mapeamento dos NFRs da nossa ontologia para propriedades AADL. Além da possibilidade de estender a ontologia criada, tanto na forma de agregar mais conceitos, quanto em criar mais propriedades e restrições entre eles.

O arquivo contendo a ontologia está disponibilizado no GitHub e pode ser acessado por meio do link: https://github.com/BrunoAquelebruno/NFR_ontology.git.

6. REFERÊNCIAS

Bézivin, Jean *et al*. **First experiments with the ATL model transformation language: Transforming XSLT into XQuery**. 2003. Tese (Mestrado em sistemas de informação) - Atlas Group, INRIA and IRIN University of Nantes, França, 2003.

Chase, Otavio. Sistemas Embarcados. **Introdução a Sistemas Embarcados**, Curitibanos, p. 1-7, 1 dez. 2007. <http://www.maxpezzin.com.br>. Acesso em: 10 mai. 2019.

Cunha, Alessandro. **Sistemas Embarcados**, Revista Saber Eletrônica, 414, Editora: Saber, BRASIL, 2007.

DroneZon. **Quick Drone Parts Overview Along With Handy DIY Tips.** 2019. Disponível em: <https://www.dronezon.com/learn-about-drones-quadcopters/drone-components-parts-overview-with-tips/>. Acesso em: 20 out. 2019.

Feiler, Peter H ; Gluch, David P; Hudak, John J. **The Architecture Analysis & Design Language (AADL): An Introduction**. Software Engineering Institute, Carnegie Mellon University. Estados Unidos, 2006.

FRANÇA, Patrícia Cunha. **Conceitos, classes e/ou universais: com o que é que se constrói uma ontologia**. 2009. Disponível em: http://www.linguamatica.com/index.php/linguamatica/article/view/10/13. Acesso em: 03 dez. 2018.

Galster, M.; Bucherer, E. **A Taxonomy for Identifying and Specifying Non-Functional Requirements in Service-Oriented Development.** IEEE Congress on Services - Part I, Honolulu, HI, 2008, pp. 345-352.

Gruber, T *et al*. Encyclopedia of Database Systems. **Springer-Verlag**, [*S. l.*], p. 31-43, 29 out. 2019.

Horridge, M., H. Knublauch, A. Rector, R. Stevens, and C. Wroe. **A Practical Guide to Building OWL Ontologies with the Protege-OWL Plugin, 1 edition.** University of Manchester. Estados Unidos , 2004.

Horrocks, I., P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, **SWRL: a semantic Web rule language combining OWL and RuleM Technical report**, Estado Unidos, 2004.

Jingbai, T.; Keqing, H.; Chong, W.; Wei, L. **A Context Awareness Non-functional Requirements Metamodel Based on Domain Ontology**. IEEE International Workshop on Semantic Computing and Systems, Huangshan, 2008, pp. 1-7.

Jouault, F.; Allilaire, F.; Bézivin, J.; Kurtev, I. *ATL:* **A model transformation tool, Science of Computer Programming**, Volume 72, Issues 1–2, 2008.

Jouault, Frédéric *et al*. ATL: A model transformation tool. **Science of Computer Programming**, França, 1 jun. 2008. Periódico.

Leite, Marcela. **Tratamento De Requisitos Não funcionais Em Sistemas De Tempo-Real Embarcados Implementados Em Vhdl/Fpga**. 2014. Tese (Mestrado em Ciência da Computação) - Computação Aplicada, Joinville, 2014.

López, H., Veresi, F., Viñolo, M., y otros. **Estado del arte de lenguajes y herramientas de transformación de modelos**. Reportes Técnicos 09-19. UR. FI – INCO, 2009.

Nayala, F. Noy, S. Michael, D. Stefan, C. Monica, W. Ray. Fergerson, M. Mark, **Creating Semantic Web Contents with Protégé-2000.** Stanford University. Estados Unidos, 2001.

Noy, Natalya F., McGUIN Deborah L.. **Ontology Development 101: A Guide to Creating Your First Ontology**. Knowledge Systems Laboratory. 32, 2001.

Picinin, D.; Farines, J-M.; Santos, C.A.S.; Koliver, C.. **A design-oriented method to build correct hypermedia documents. Multimedia Tools and Applications**, Brasil. Volume 77, pp 21003–21032, 2017.

Rautenberg, Sandro; Todesco, José Leomar; Steial, Andrea Valéria. **Ontologias de domínio no mapeamento de instrumentos da gestão do conhecimento e de agentes computacionais da engenharia do conhecimento: o estado da arte**. Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis, p. 163-182, 2010.

Rijgersberg, Hajo; Van Assem, Mark; Top, Jan. Ontology of units of measure and related concepts. **Journal Semantic Web - Linked Data for science and education archive**. v. 4 n. 1, 2013, pp 3-13.

Ryan, Andrew. (2000). **An approach to quantitative non-functional requirements in software development**. Disponível em https://www.researchgate.net/publication/228609536_An_approach_to_quantitative_non-functional_requirements_in_software_development (Acesso em 29/10/2019).

S. Gerard, J.-P. Babau, and J. Champeau. **Model Driven Engineering for Distributed Real-Time Embedded Systems: Wiley-IEEE Press**, 2010.[25].

Sachs, E. **Getting Started with Protege-Frames**. (2006) Disponível em: http://protege.stanford.edu/doc/tutorial/get_started/get-started.html. Acesso em 09 out. 2019.

SIMONS, Peter M. **Ontology. Encyclopædia Britannica**., janeiro de 2015. Disponível em: https://www.britannica.com/topic/ontology-metaphysics. Acesso em 29/08/2019.

Sommerville, I. **Engenharia de software**. Addison Wesley, 2008.

Sotelo, Karla I. Gomez; Baron Claude; Esteban, Philippe; Estrada, Citlalih Y. A. Gutiérrez; Velázquez, Luis Laredo. **How to find non-functional requirements in system developments**. 16th IFAC Symposiumon Information Control Problems in Manufacturing (INCOM 2018), Jun. 2018, Bergamo, Itália.

Stanford University (Estados Unidos). **Protégé Wiki.** 2016. Disponível em: <https://protegewiki.stanford.edu/wiki/Main_Page>. Acesso em: 03 dez. 2018.

VITAL, Luciane Paula and CAFE, Ligia Maria Arruda. **Ontologias e taxonomias: diferenças. Perspect.** ciênc. inf. [online]. 2011, vol.16, n.2, pp.115-130. ISSN 1981-5344.

Wehrmeister, M. A.; Freitas, E. P.; Pereira; C. E.; Wagner F. R. **An Aspect-Oriented Approach for Dealing with Non-Functional Requirements in a Model-Driven Development of Distributed Embedded Real-Time Systems,** ***10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)***, Santorini Island, 2007, pp. 428-432.

Wehrmeister, M. A.; Freitas, E. P.; Pereira; C. E.; Wagner F. R., **Automatized Translation From A Functional Into An Architectural Model: An Application To The Quadrotor UAV.** Departamento de Automação e Sistemas. Universidade Federal de Santa Catarina, 2007.

Wikipedia (Comp.). **Ontologia (ciência da computação).** 2018. Disponível em: <https://en.wikipedia.org/wiki/Architecture_Analysis_%26_Design_Language>. Acesso em: 03 dez. 2018.

Wikipedia (Comp.). **Software Embarcado.** 2019. Disponível em: <https://pt.wikipedia.org/wiki/Software_Embarcado>. Acesso em: 20 out. 2019.

APÊNDICE A – ARTIGO SOBRE O TRABALHO DESENVOLVIDO

Pesquisa sobre o uso de ontologias para a definição de requisitos não funcionais em de Sistemas Embarcados Distribuídos de Tempo Real

Bruno Goulart Andrade[1]

[1]CTC – Universidade Federal de Santa Catarina
R. Eng. Agronômico Andrei Cristian Ferreira, s/n - Trindade, Florianópolis - SC, 88040-900

[2]Departamento de Ciências Tecnologicas
bruno.goulart.andrade@gmail.com

Abstract. *Understanding the problem of non-functional requirements gathering and the importance of these requirements in the development of Real-time Distributed Embedded Systems, a proposal to create an ontology for the definition of these requirements was investigated. After the creation of the ontology, using the Protégé tool, it was validated with a specialist in this area.*

Resumo. *Entendendo a problemática do levantamento de requisitos não funcionais e a importância desses requisitos no desenvolvimento de Sistemas Embarcados Distribuídos de Tempo Real, pesquisou-se uma proposta de criação de uma ontologia para a definição desses requisitos. Após a criação da ontologia, utilizando a ferrramenta Protégé, a mesma foi validada junto a um especialista da área.*

1. Introdução

Os sistemas embarcados, segundo Luigi Carro e Flávio Rech Wagner (2007), estão presentes em nosso cotidiano e utilizamos esses sistemas em diversas atividades. Celulares, máquinas de lavar, sistemas de controle automotivo são exemplos de uso destes sistemas.

Sistemas embarcados de tempo real exigem muita atenção do projetista para que as tarefas sejam executadas no momento definido, caso contrário, pode ocasionar uma catástrofe.

A coesão quando do levantamento de requisitos não funcionais (NFR) em Sistemas Embarcados Distribuídos de Tempo Real é tão ou mais importante do que o levantamento de requisitos funcionais. Se uma falha em cumprir um requisito funcional pode comprometer parte do sistema, uma falha em cumprir um NFR pode tornar todo o sistema inútil (SOMMERVILLE, 2008).

Além disso, enquanto que o levantamento de requisitos funcionais é feito através de entrevistas com o úsuário, o levantamento de NFRs depende muito da experiência e conhecimento do projetista da aplicação.

Visto isso, fez-se esse estudo sobre a criação de uma ontologia para o suporte no levantamento de NFRs.

2. Pesquisa

2.1. Ontologias

Adotou-se, para fins deste estudo a definição de ontologia dada por T. R. Gruber (1993): “Uma ontologia é uma especificação formal e explícita de uma conceituação compartilhada”. Nessa definição, a palavra formal refere-se ao fato de que a ontologia deve ser “entendível” por máquinas; explícita significa que o tipo de conceitos usados e as restrições em seu uso são explicitamente definidos; conceituação refere-se ao fato de que uma ontologia é um modelo abstrato de algum fenômeno do mundo que identifica os conceitos relevantes desse fenômeno.

As ontologias possuem 4 elementos principais: as classes ou conceitos; as propriedades, os indivíduos e os axiomas. As classes representam o domínio de interesse, as propriedades definem o relacionamento entre classes, os axiomas criam restrições e regras e os indivíduos são exemplos e representações dos conceitos.

2.2. Sistemas de tempo real

São considerados sistemas de tempo real aqueles sistemas que devem realizar as tarefas tendo um tempo limite para que elas aconteçam (TSAI ET AL., 1996).

Eles podem ser classificados como *hard* ou *soft*, dependendo da rigidez pretendida. Sistemas do tipo *hard* são, por exemplo, sistemas médico-hospitalares ou de controle de aeronaves, onde uma falha pode ocasionar uma catástrofe. Sistemas do tipo *soft* não exigem tamanha rigidez, como uma TV digital ou o *streaming* do YouTube.

2.3. Sistemas embarcados

Um sistema pode ser classificado como embarcado quando ele é dedicado a uma única tarefa, interagindo continuamente com o meio ambiente por meio de sensores e atuadores. Sua estrutura é um núcleo de processamento acoplado a uma placa de circuito impresso, onde roda um software que, no caso de sistemas embarcados, é chamado de *firmware*.

3. Requisitos não funcionais

Os requisitos não funcionais levantados no trabalho intitulado “Uso de uma ontologia de requisitos não funcionais no suporte ao desenvolvimento de sistemas embarcados”, realizado por Bruno Andrade (2019), são descritos a seguir. A edentação de tabulações define as classes e subclasses:

NFR

	Distribution

		Communication

		Hosts

- Synchronization
- TaskAllocation

Embedded
- MemoryAlocation
- PowerConsumption
- TotalEnergy

Fault
- Electrical
  - SlowADCStatus
  - SupplyVoltageStatus
- Logical
  - DataLoss
  - DelayedData
  - IncerrectData
  - InvalidFirmware
  - ProcessorError
  - ROMError

Link
- BandWidth
- Protocol

Measure
- Area
- Volume
- Weight

Modes
- AutomaticOperation
- EmergencyMode
- IrreversibleFaliure
- ManualOperation
- PartialOperation

OperationalSystem
- NonRealTime
- RealTimeHard

- RealTimeSoft
- Performance
  - ResponseTime
  - Throughtput
- PhysicalComponent
  - Dispositive
    - UAV
      - Dicopter
      - Quadcopter
      - Vtol-rpa
  - DispositivePart
    - Antenna
    - Bar
    - Battery_Monitor
    - Boom
    - Brushless
    - Brushless_Motor
    - Camera
      - MovieCamera
      - NavigationCamera
    - Esc
    - Flight_Controller
    - GPS
    - Gimbal
    - Gimbal_Controller_Unit
    - Gimbal_Motor
    - HighLevelHardware
    - IMU
    - Landing_Gear
    - LowLevelHardware
    - MainBody
    - Motor_Mount
    - Pitot

- PowerManager
- Propeller
- Pusher_Prop
- Radio
- Receiver
- Sensors
  - Accelerometer
  - Gyroscope
  - Magnetometer
- Servo
- Sonar

Hardware
- Memory
  - NonVolatileMemory
    - HardDisk
    - Solid-State_Drive
  - VolatileMemory
    - Cache
    - ProcessorInternalMemory
      - Data
      - Program
- Processor
  - Capability
  - Frequency

Task
- Process
- Thread

TaskPeriodicity

Time
- Precision
  - Drift
  - Freshness
  - Jitter

Laxity

Resolution

ToleratedDelay

SchedulingPolicy

Algorithm

Timing

ActivationLatency

Cost

Deadline

EndTime

ExecutionTime

AverageExecutionTime

BCET

WCET

Period

ReleaseTime

StartTime

A ferramenta para a modelagem da ontologia utilizada foi o software Protégé, por ser gratuita e *open source*, além de ser a mais difundida para a construção de ontologias.

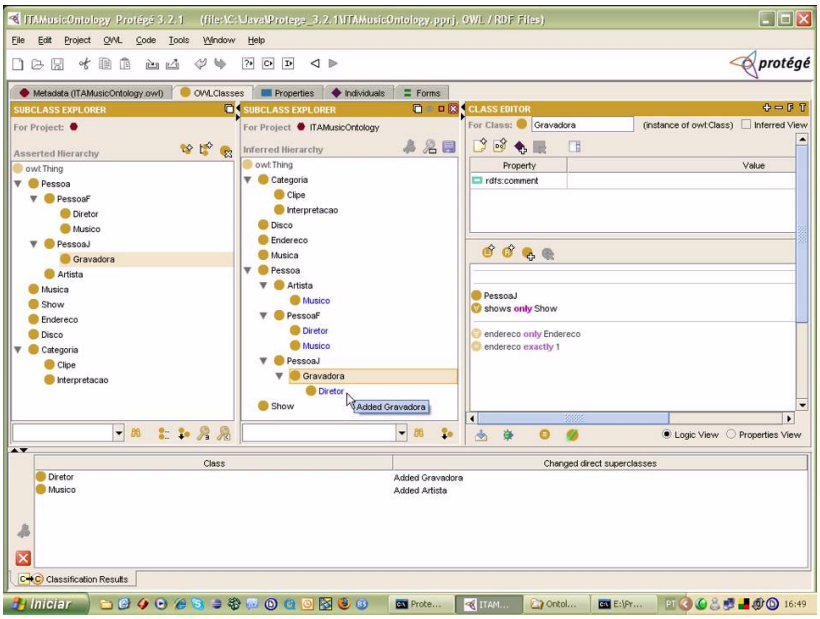

Figura 1. Protégé.

A figura a seguir mostra as propriedades criadas para relacionar as classes:

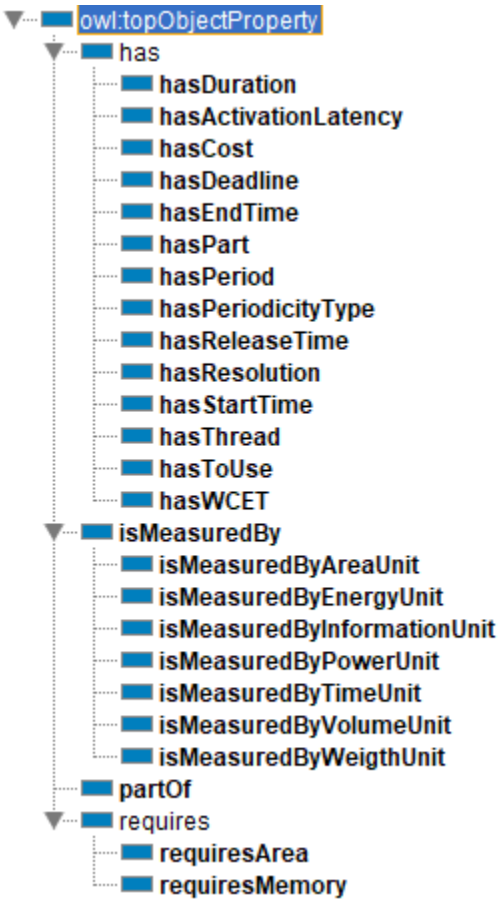

Figura 2. Propriedades.

Exemplos de relacionamento entre as classes através das propriedades acontecem por meio de palavras reservadas do Protégé, algumas delas são:

- *some*: indica existencialidade, elemento da esquerda possui *pelo menos* algo do elemento da direita;
- *min, max, exactly*: indica a cardinalidade;
- *only*: semantica de somente;

Os relacionamentos são escritos da seguinte maneira:

- Process hasThread min 1 Thread;
- Brushless has exactly 1 BrushlessMotor;
- PowerConsumption hasToUse some TotalEnergy;

4. Conclusão

O trabalho estudado se propôs a cobrir uma ontologia inicial, com algumas classes e propriedades. Também foi realizada a validação da ontologia com um especialista da área, além da aplicação prática, inserindo na ontologia um levantamento de requisitos não funcionais de uma projeto de construção de um VTOL-RPA (*Vertical Takeoff and Landing Remotely Piloted Air Vehicle*) pelo Departamento de Automação e Sistemas da UFSC. A ontologia estudada está disponível em: https://github.com/BrunoAquelebruno/NFR_ontology.git.

O presente artigo descreveu o levantamento dos requisitos não funcionais para sistemas embarcados distribuídos de tempo real através do uso de uma ontologia.

Ainda, sugere-se como trabalhos futuros uma investigação mais completa, que possa abarcar todos os requisitos não funcionais de algum dispositivo em específico e a agregação desses requisitos na ontologia.

5. Referencias

Andrade, Bruno. Uso de uma ontologia de requisitos não funcionais no suporte ao desenvolvimento de sistemas embarcados. Centro Tecnológico – Universidade Federal de Santa Catarina, 2019.

Sommerville, I. Engenharia de software. Addison Wesley, 2008.

Wehrmeister, M. A.; Freitas, E. P.; Pereira; C. E.; Wagner F. R. An Aspect-Oriented Approach for Dealing with Non-Functional Requirements in a Model-Driven Development of Distributed Embedded Real-Time Systems, 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07), Santorini Island, 2007, pp. 428-432.