

**MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO E
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE DEFESA**

EQUIPE IME PROJETO S2C2

**ESPECIFICAÇÃO DE PROCEDIMENTOS SISTEMÁTICOS PARA APOIAR A
INTEROPERABILIDADE PARA SOS DE C^2 (NÍVEL LÓGICO)**

**RIO DE JANEIRO
JULHO/2025**

SUMÁRIO

1	INTRODUÇÃO	2
2	PROCEDIMENTO SISTEMÁTICO PARA OPERACIONALIZAÇÃO DE ONTOLOGIAS	3
2.1	DIRETRIZES PARA A OPERACIONALIZAÇÃO DE ONTOLOGIAS EM OWL E SWRL	5
2.1.1	PONTOS DE ATENÇÃO NA TRANSFORMAÇÃO DE ONTOUML PARA OWL	6
2.1.2	CONFIGURAÇÃO DE RESTRIÇÕES LÓGICAS E SEMÂNTICAS	9
3	CONCLUSÃO	14
	REFERÊNCIAS	16

1 INTRODUÇÃO

O presente relatório complementa o relatório *Especificação de Procedimentos Sistemáticos para apoiar a Interoperabilidade para SoS de C2 (nível conceitual)*, e corresponde a um indicador físico de execução da atividade de *Especificação da Sistemática¹ para o nível lógico* (atividade 2), prevista no Plano de Trabalho do projeto S2C2, para atingir a Meta física 5: *Interop - Conceber linhas de ação para a interoperabilidade de dados com relação à modelagem nos níveis conceitual e lógico.*

Além da introdução, o presente relatório é estruturado como se segue. O Capítulo 2 apresenta o procedimento sistemático adotado para a operacionalização das ontologias concebidas na etapa de captura e formalização de conceitos, conforme descrito no relatório anterior. Nesse capítulo apresentamos a descrição da abordagem SABIO e acrescentamos algumas diretrizes para a operacionalização de ontologias em OWL utilizando a OntoUML como linguagem de concepção de ontologias de referência. O capítulo 3 apresenta uma breve conclusão e a infraestrutura necessária para a operacionalização das ontologias em OWL com base nas ontologias de referência expressas em OntoUML.

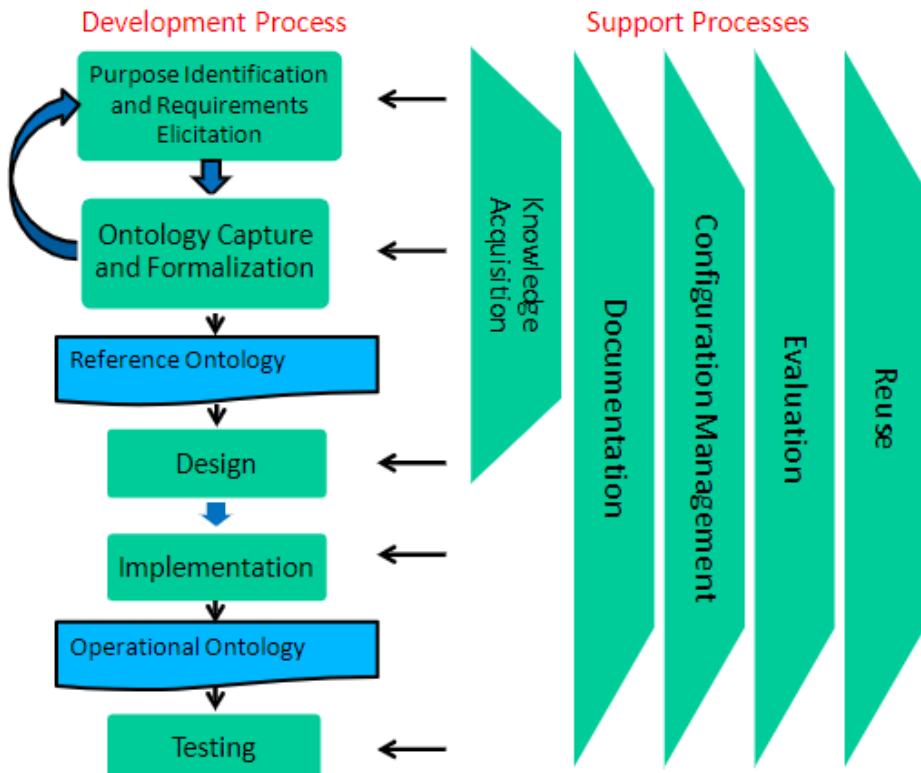
¹ Termo originalmente usado para o planejamento de atividades do projeto, mas que foi ajustado para refletir melhor o significado pretendido. Assim, pelo restante do documento passaremos a nos referir a *Procedimento Sistemático*.

2 PROCEDIMENTO SISTEMÁTICO PARA OPERACIONALIZAÇÃO DE ONTOLOGIAS

Como já apresentado no relatório anterior (Espec. Proc. Sistem. - Nível Conceitual) para construir as ontologias, optou-se por seguir um processo de desenvolvimento conhecido como Abordagem Sistemática para Construção de Ontologias (Systematic Approach for Building Ontologies, em inglês)(SABiO) (1). Esta abordagem materializa os conceitos da Modelagem Conceitual Dirigida por Ontologias (ODCM) ao distinguir claramente dois artefatos: a ontologia de referência (nível conceitual), apoiada por uma ontologia de fundamentação, e uma ontologia operacional (nível lógico) que as máquinas podem interpretar.

A SABiO constitui-se de cinco fases: *Identificação de Propósito e Elicitação de Requisitos*, *Captura e Formalização*, *Projeto*, *Implementação* e *Teste*, como apresentado na Figura 1. Os procedimentos envolvidos em cada fase já foram descritos no relatório anterior. Neste relatório, o foco é nas fases de Projeto e Implementação.

Figura 1 – SABiO's processes



Em alguns casos, a entrega da ontologia de referência é suficiente e a abordagem não deve mais ser seguida. No entanto, no caso deste projeto, a ideia foi seguir e implementar ontologias operacionais com base nas respectivas ontologias de referência concebidas.

Assim, seguindo a abordagem SABIO, o desenvolvedor inicia o desenvolvimento da ontologia operacional na fase de *Projeto*. Ao contrário da ontologia de referência, a ontologia operacional é concebida e implementada em uma linguagem de máquina, concentrando-se em garantir propriedades computacionais desejáveis. Inicialmente, o desenvolvedor tem de complementar a lista de requisitos tecnológicos não funcionais para a ontologia operacional e depois, definir o ambiente em que esta será implementada. Uma vez definido o ambiente, é necessário rever a modularização da ontologia elaborada no início do projeto e definir a arquitetura da ontologia com base nos requisitos tecnológicos não funcionais e no ambiente construído.

Nas últimas fases (*Implementação* e *Teste*), a ontologia operacional desenvolvida é implementada na linguagem operacional escolhida e testada em estudos de caso. Mais uma vez, as questões de competência desempenham um papel importante na criação global da ontologia. Embora sejam essenciais para a coleta de conceitos e relações do domínio na fase *Captura e Formalização*, as questões de competência são a espinha dorsal dos testes da ontologia. De acordo com Falbo, um caso de teste compreende um conjunto de consultas no ambiente de implementação, uma instanciação de dados de acordo com a ontologia sendo testada (entrada) e um resultado esperado (saída). Neste cenário, uma consulta é a materialização de uma questão de competência em uma linguagem de consulta que seja compatível com a linguagem escolhida pelo projetista para implementar a ontologia. O desenvolvedor da ontologia deve executar os casos de teste em diferentes fragmentos, tais como sub-ontologias e partes integradas, até chegar ao teste da ontologia completa.

A implementação da ontologia de referência, em sua versão operacional, legível por máquina, pode ser realizada na linguagem OWL¹ (Web Ontology Language). Essa linguagem apresenta o diferencial de ser uma linguagem padronizada pela W3C para a Web semântica, permitindo não só a descrição de dados enriquecidos semanticamente, através da integração de metadados e vocabulários em um grafo de conceitos interconectados, como também a aplicação de regras de inferência, que permitem a expansão automática do grafo. Essas características tornam a ontologia operacional expressa em OWL mais interoperável em comparação à sua operacionalização em linguagens de programação tradicionais (e.g. Java ou Python). Nesse caso, a ontologia é incorporada diretamente nos códigos das aplicações para representar as estruturas de dados. Essa alternativa é também válida e praticada, em especial quando há requisitos que envolvem desempenho ou compatibilidade com outras linguagens de desenvolvimento. No entanto, como já falado, ao optar por linguagens como RDF/OWL, obtemos uma representação independente e interoperável.

A escolha da ontologia de fundamentação feita na fase de Captura e Formalização da ontologia de referência é também uma escolha estratégica. A escolha da UFO como ontologia

¹ <https://www.w3.org/OWL/>

de fundamentação é vantajosa uma vez que tem-se à disposição a linguagem OntoUML² que facilita a concepção de ontologias de referência fundamentadas na UFO. Além disso, pode-se contar com o ferramental já existente para a concepção e operacionalização das ontologias, como a ferramenta de diagramação UML Visual Paradigm³ e o plugin⁴ para a linguagem OntoUML. Essas ferramentas possibilitam que a versão operacional das ontologias seja gerada semi-automaticamente, utilizando uma versão leve da UFO, denominada gUFO (2) Através desse plugin é possível verificar a corretude sintática da ontologia e gerar sua versão operacional no formato turtle (ttl)(*Terse RDF Triple Language*). A partir desse código e utilizando a ferramenta Protégé⁵ foi possível especializar e instanciar os elementos da ontologia para então gerar o arquivo final em formato OWL/XML.

Embora expressiva, a linguagem OWL não é suficiente para expressar algumas regras de inferência. Assim, complementarmente, podem ser definidas regras de inferência na linguagem SWRL⁶, outra linguagem padrão da W3c, de modo a permitir maior poder de raciocínio. Ao instanciar as classes e relacionamentos da ontologia operacional, o raciocinador consegue inferir novas informações. Parte dessas inferências é feita com base somente nas classes e nos relacionamentos expressos em OWL, enquanto outras inferências são possíveis somente com base em regras expressas em SWRL.

O procedimento de geração semi-automática provido pelo plugin OntoUML para a ferramenta de diagramação Visual Paradigm deixa algumas lacunas que precisam ser tratadas. A seção seguinte apresenta algumas diretrizes para esse tratamento, exemplificando no contexto da operacionalização das ontologias HINT e MISCON.

2.1 Diretrizes para a operacionalização de ontologias em OWL e SWRL

Esta seção apresenta um conjunto de diretrizes ou boas práticas identificadas durante a operacionalização das ontologias HINT e MISCON. As ontologias operacionais em formato OWL/TTL podem ser encontradas no repositório do projeto⁷. O detalhamento da transformação das ontologias de referência em ontologias operacionais e os trechos de código das ontologias MISCON e HINT podem ser encontrados nos trabalhos de Demori (3) e Tesolin (4), respectivamente.

Através dessa implementação e das inferências que foram realizadas, as questões de competência identificadas na fase de levantamento de requisitos da abordagem SABIO

² <https://ontouml.org/>

³ <https://www.visual-paradigm.com/>

⁴ <https://github.com/OntoUML/ontouml-vp-plugin>

⁵ <https://protege.stanford.edu/software.php>

⁶ <https://www.w3.org/submissions/SWRL/>

⁷ <https://github.com/comp-ime-eb-br/S2C2-IME/tree/main/onto>

podem ser respondidas. Essas questões podem ser traduzidas em consultas (*queries*) SPARQL, que é a linguagem de consulta usada para recuperação de informação em bases de conhecimento representadas nas linguagens RDF ou OWL.

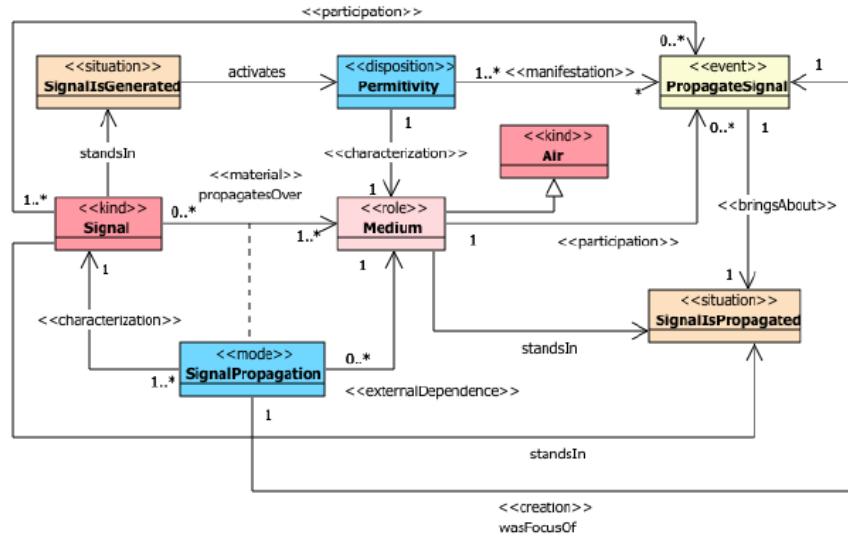
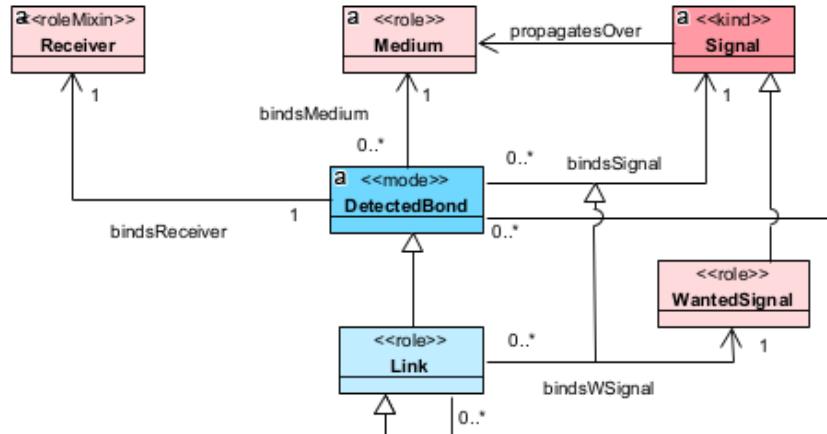
Nas subseções seguintes foram descritas possíveis otimizações, como a transformação de relações materiais e a supressão de papéis únicos. Em seguida, apresentamos como as ontologias operacionais podem ser enriquecidas com as restrições definidas na ontologia de referência, usando as linguagens de regras semânticas como a SWRL. Por fim, indicamos como os artefatos podem ser documentados seguindo os princípios FAIR.

2.1.1 Pontos de atenção na transformação de OntoUML para OWL

Como já foi dito, a OntoUML permite a criação de modelos conceituais sólidos baseados na ontologia fundamental UFO. Através do plugin OntoUML para a ferramenta de diagramação Visual Paradigm, os desenvolvedores de ontologias podem projetar um modelo de referência com validação sintática automática de acordo com as teorias UFO e exportar diagramas de classes baseados em OntoUML como ontologias operacionais escritas em OWL, economizando tempo de desenvolvimento ao entregar uma ontologia operacional pronta para ser testada. No entanto, é necessário revisar alguns aspectos do projeto antes de transformar as ontologias de referência em ontologias operacionais. A revisão se concentrou na modularização da ontologia, na reificação das relações materiais e na implementação de entidades que desempenham papéis.

As ontologias de referência propostas por Tesolin (4) procuraram apresentar os elementos necessários para apoiar a gestão da mobilidade em redes de comunicação móvel através do raciocínio semântico. Assim, as ontologias de referência desenvolvidas não apenas descrevem os elementos necessários de uma rede de comunicação móvel, mas também o processo de atribuição de valores aos atributos que caracterizam esses elementos ao longo do tempo. Todos os conceitos e relações derivadas fazem parte da mesma ontologia operacional. No entanto, foi verificado que a ontologia de referência proposta para a atribuição de valores de qualidade poderia ser usada em outros domínios. Consequentemente, duas ontologias operacionais foram criadas: uma chamada “Heterogeneous Telecommunication Network Ontology” (**HINT**) para a descrição dos elementos da rede móvel e outra chamada “Quality Value Assignment Ontology” (**QVAS**) para as descrições das atribuições de valor de qualidade. Por outro lado, a MISCON, desenvolvida por Demori (3) deteve-se apenas na representação de cenários militares, não sendo verificada nenhuma necessidade de segmentação da ontologia de referência em sub-ontologias operacionais.

Uma vantagem de usar a UFO como a ontologia de fundamentação é sua capacidade inerente de representar a reificação das relações materiais. Na fase de formalização da ontologia usando a OntoUML, representamos simultaneamente as relações materiais e sua reificação (ou *truthmaker*), melhorando a conceituação da ontologia, especialmente no

Figura 2 – SignalPropagation e *propagatesOver* na ontologia de referênciaFigura 3 – Apenas *propagatesOver* na ontologia operacional

que diz respeito à cardinalidade e à análise das relações n-árias. No entanto, na fase de operacionalização, é possível escolher o que faz sentido manter. Assim, foram revisadas todas as relações materiais propostas para identificar quais poderiam ser substituídas por seus *truthmakers*. Por exemplo, as Figuras 2 e 3 mostram a opção por remover o EDM reificado **SignalPropagation** e manter a relação *propagatesOver*(*Signal*, *Medium*), uma vez que era usada apenas para descrever um conceito.

Por outro lado, todas as outras relações materiais propostas nos modelos de referência foram substituídas por seus *truthmakers*, uma vez que há a necessidade de caracterizar tais relações com indicadores de desempenho (por exemplo, taxa de transmissão e latência), entre outros. Assim, é especialmente necessário representar a relação ternária **Signal-Medium-Receiver**. As Figuras 4 e 5 mostram um exemplo dessa transforma-

Figura 4 – Relação ternária na ontologia de referência

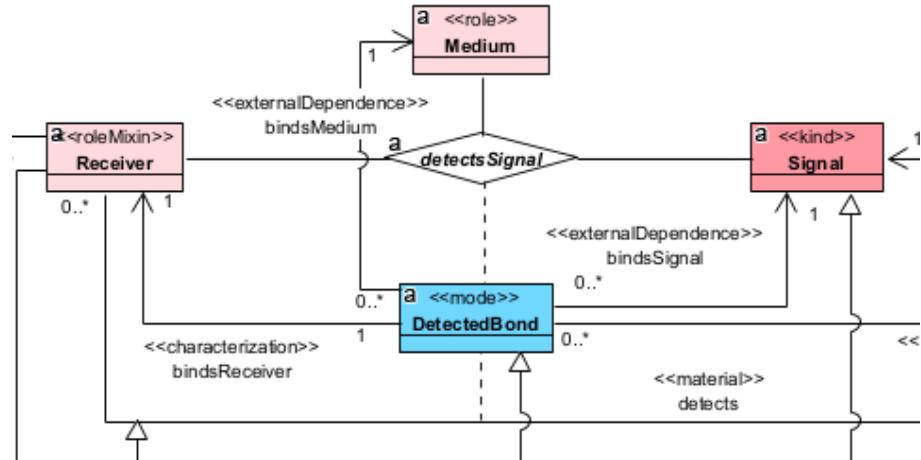
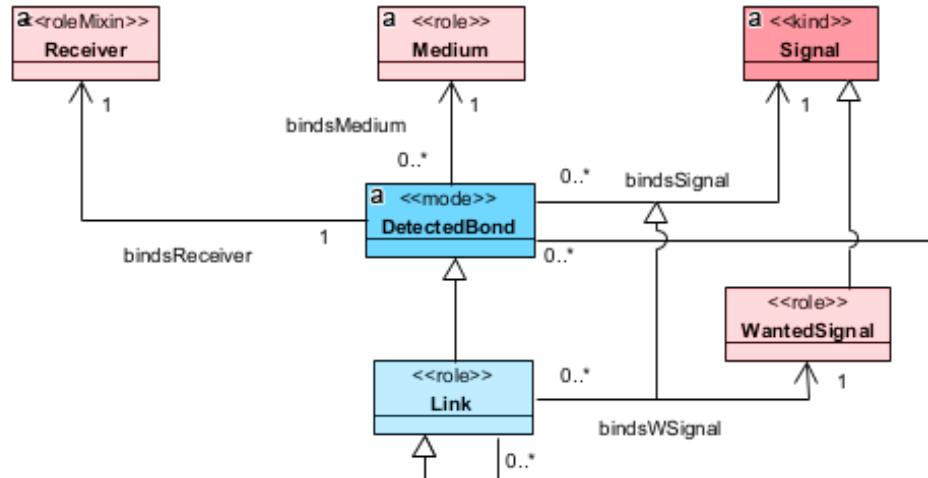


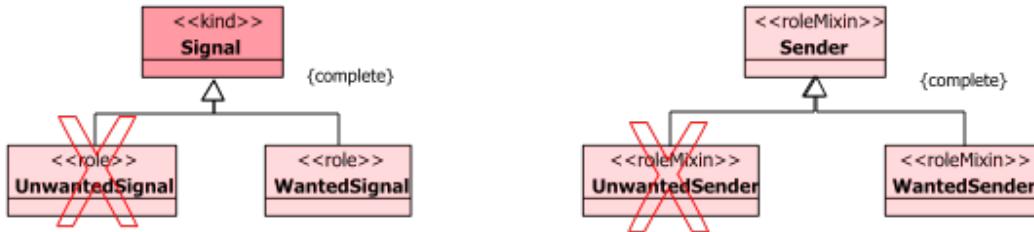
Figura 5 – Relação ternária na ontologia operacional



ção, em que a relação ternária **detectsSignal** no modelo de referência é reificada para (**DetectedBond**) na ontologia operacional.

Finalmente, todas as entidades meta categorizadas como papel (**Role**) foram revisadas. Embora representar funções em ontologias de referência melhore a conceituação e a comunicação do domínio, isso pode levar a descrições desnecessárias em ontologias operacionais. O modelo de referência ilustrado na Figura 6 é um exemplo. Embora seja importante comunicar as possíveis funções de **Signal** e **Sender** na ontologia de referência, não é útil representar **UnWantedSignal** e **UnWantedSender** na ontologia operacional. Portanto, estas entidades foram suprimidas nas ontologias operacionais. Outros papéis foram analisados como candidatos à supressão, como **ULNode** e **LLNode**. Eles são o único conjunto de generalização de **ServiceNode**, relacionado-se apenas entre si. No entanto, neste caso, os papéis fornecem os meios para criar restrições e não devem ser suprimidos (por exemplo, **Commboard** só pode desempenhar a função de **LLNode**).

Figura 6 – Candidatos para supressão de papéis



Assim, após revisões e ajustes, a consistência sintática foi novamente verificada. Em seguida, as ontologias operacionais (**hint**, **qvas** e **MISCON**) foram criadas como artefatos acionáveis por máquina, escritos em OWL usando o recurso de exportação do plug-in OntoUML no Visual Paradigm.

Por fim, como artefatos de pesquisa, todas as ontologias estão em conformidade com os princípios FAIR. Nesse sentido, as ontologias operacionais foram aprimoradas da seguinte forma:

Findability: todas as ontologias foram registradas na organização Internet Archive to para receber URLs como identificadores únicos. As ontologias **hint**, **qvas**, **MISCON** podem ser encontradas em <<https://purl.org/s2c2/hint>>, <<https://purl.org/s2c2/qvas>> e <<https://purl.org/s2c2/miscon>>

Accessibility: todas as ontologias estão disponíveis sob a licença CC BY 4.0 ⁸

Interoperability and Reusability: além de estarem expressos em OWL, todos os conceitos e relacionamentos estão descritos utilizando vocabulários-padrão como a Dublin Core e o RDFS.

2.1.2 Configuração de Restrições Lógicas e Semânticas

Até o momento da escrita deste relatório, o plugin OntoUML não suportava a descrição de restrições e a sua transformação numa linguagem de regras da web semântica. Para superar tal limitação, usamos o Protege Ontology Editor⁹⁽⁵⁾ juntamente com o seu plugin SWRL para complementar as ontologias operacionais com as restrições necessárias.

Cada regra lógico-semântica descrita em Demori (3) e Tesolin (4) é transformada em regras da web semântica usando axiomas OWL ou SWRL. As Tabelas 1, 2 e 3 mostram exemplos de algumas transformações para a ontologia **HINT**, enquanto a Tabela 4 mostra um exemplo de transformação para a ontologia **QVAS**.

⁸ available at <<https://creativecommons.org/licenses/by/4.0/legalcode>>

⁹ disponível em <<https://protege.stanford.edu/>>

Tabela 1 – Transformação das restrições da taxonomia de CommBoards em regras da web semantica

Op. Ontology	hint
Ref. Ontology Eq.	$\forall a(Tx(a) \rightarrow \neg Receiver(a))$
Rule Transf. Language	OWL Axiom
Rule Statement	<code>:Receiver owl:disjointWith :Tx</code> (a) Restrição: Um Tx não pode estar no papel de Receiver
Op. Ontology	hint
Ref. Ontology Eq.	$\forall a(Rx(a) \rightarrow \neg Sender(a))$
Rule Transf. Language	OWL Axiom
Rule Statement	<code>:Sender owl:disjointWith :Rx</code> (b) Restrição: Um Rx não pode estar no papel de Sender
Op. Ontology	hint
Ref. Ontology Eq.	$\forall a, b, c(TRx(a) \wedge Tx(b) \wedge Rx(c) \wedge (hasTx(a, b) \vee hasRx(a, c) \rightarrow \neg(Sender(a) \wedge Receiver(a)))$
Rule Transf. Language	SWRL
Rule Statement	$TRx(?a) \wedge Tx(?b) \wedge hasTx(?a, ?b) \wedge Sender(?a) \wedge Sender(?b) \rightarrow sameAs(?a, ?b)$ $TRx(?a) \wedge Rx(?b) \wedge hasRx(?a, ?b) \wedge Receiver(?a) \wedge Receiver(?b) \rightarrow sameAs(?a, ?b)$
(c) Restrição: Se Tx(b) e Rx(c) compõem TRx(a), então TRx(a) não pode exercer os papéis de Sender, tampouco de Receiver	

Outras regras em SWRL foram geradas durante a operacionalização da ontologia MISCON. Estas regras estão descritas em (3) e no *Relatório de Construção da Prova de Conceito*.

Tabela 2 – Transformação das restrições de HandShakeBond em regras da web semântica

Op. Ontology	hint
Ref. Ontology Eq.	$\forall a, b(handshakes(a, b) \rightarrow (TRx(a) \wedge TRx(b) \wedge (a \sqsubseteq \neg b)))$
Rule Transf. Language	SWRL
Rule Statement	$TRx(?a) \wedge TRx(?b) \wedge HandShakeBond(?c) \wedge bindsTRx(?c, ?a) \wedge bindsTRx(?c, ?b) \rightarrow differentFrom(?a, ?b)$
(a) Se existe a relação <i>handshakes</i> entre a e b, então necessariamente a e b são do tipo <i>TRx</i> e não são a mesma entidade	
Op. Ontology	hint
Ref. Ontology Eq.	$\forall a, b, c(TRx(a) \wedge TRx(b) \wedge HandShakeBond(c) \wedge bindsTRx(c, a) \wedge bindsTRx(c, b) \rightarrow \exists d, e SelectedBond(d) \wedge SelectedBond(e) \wedge bindsReceiver(d, b) \wedge bindsSender(d, a) \wedge bindsReceiver(e, a) \wedge bindsSender(e, b) \wedge hasSelectedBond(c, d) \wedge hasSelectedBond(c, e))$
Rule Transf. Language	SWRL
Rule Statement	$TRx(?a) \wedge TRx(?b) \wedge HandShakeBond(?c) \wedge bindsTRx(?c, ?a) \wedge bindsTRx(?c, ?b) \wedge SelectedBond(?d) \wedge SelectedBond(?e) \rightarrow bindsReceiver(?d, ?b) \wedge bindsSender(?d, ?a) \wedge bindsReceiver(?e, ?a) \wedge bindsSender(?e, ?b) \wedge hasSelectedBond(?c, ?d) \wedge hasSelectedBond(?c, ?e)$
(b) Se <i>TRx(a)</i> e <i>TRx(b)</i> participam de um processo de handshake representado por <i>HandShakeBond(c)</i> , e o mesmo está efetivamente vinculado a ambos, então esse handshake deve originar dois vínculos selecionados distintos. Um desses vínculos selecionados conecta <i>TRx(a)</i> como <i>Sender</i> e <i>TRx(b)</i> como <i>Receiver</i> , enquanto o outro conecta <i>TRx(b)</i> como <i>Sender</i> a <i>TRx(a)</i> como <i>Receiver</i> . Ambos os vínculos resultantes são explicitamente associados ao <i>HandShakeBond(c)</i> .	

Tabela 3 – Transformação das restrições de PeerNodes em regras da web semântica

Op. Ontology	hint
Ref. Ontology Eq.	$\forall a, b, c, d(CommDevice(a) \wedge CommDevice(b) \wedge LayerNode(c) \wedge LayerNode(d) \wedge hasLN(a, c) \wedge hasLN(b, d) \wedge detects(c, d) \rightarrow (a \sqsubseteq \neg b))$
Rule Transf. Language	SWRL
Rule Statement	$CommDevice(?a) \wedge CommDevice(?b) \wedge LayerNode(?c) \wedge LayerNode(?d) \wedge Link(?e) \wedge hasLN(?a, ?c) \wedge hasLN(?b, ?d) \wedge bindsReceiver(?e, ?c) \wedge bindsSender(?e, ?d) \rightarrow differentFrom(?a, ?b)$
(a) Para quaisquer <code>CommDevice(a)</code> e <code>CommDevice(b)</code> , e quaisquer <code>LayerNode(c)</code> e <code>LayerNode(d)</code> , se <code>CommDevice(a)</code> está associado a <code>LayerNode(c)</code> , <code>CommDevice(b)</code> está associado a <code>LayerNode(d)</code> , e <code>LayerNode(c)</code> detecta <code>LayerNode(d)</code> , então <code>CommDevice(a)</code> e <code>CommDevice(b)</code> pertencem a classes mutuamente exclusivas.	
Op. Ontology	hint
Ref. Ontology Eq.	$\forall a, b, c, d(PeerNode(a) \wedge PeerNode(b) \wedge Protocol(c) \wedge Protocol(d) \wedge hasProtocol(a, c) \wedge hasProtocol(b, d) \wedge decodes(a, b) \rightarrow (c \equiv d))$
Rule Transf. Language	SWRL
Rule Statement	$PeerNode(?a) \wedge PeerNode(?b) \wedge Protocol(?c) \wedge Protocol(?d) \wedge Link(?e) \wedge isProtocolOf(?c, ?a) \wedge isProtocolOf(?d, ?b) \wedge bindsReceiver(?e, ?a) \wedge bindsSender(?e, ?b) \rightarrow sameAs(?c, ?d)$
(b) Para quaisquer <code>PeerNode(a)</code> e <code>PeerNode(b)</code> e quaisquer <code>Protocol(c)</code> e <code>Protocol(d)</code> , se <code>PeerNode(a)</code> utiliza o <code>Protocol(c)</code> , <code>PeerNode(b)</code> utiliza <code>Protocol(d)</code> , e <code>PeerNode(a)</code> é capaz de decodificar mensagens provenientes de <code>PeerNode(b)</code> , então os protocolos <code>Protocol(c)</code> e <code>Protocol(d)</code> devem ser equivalentes.	

Tabela 4 – Transformação das restrições de **ValueAsgmt** em regras da wewb semântica

Op. Ontology	qvas
Ref. Ontology Eq.	$\forall a, b(AsgnEvent(a) \wedge ValueAsgmt(b) \wedge isResultOf(b, a) \wedge hasEndTime(a, \{t\}) \rightarrow hasResultime(b, \{t\})$
Rule Transf. Language	SWRL
Rule Statement	$AsgnEvent(?a) \wedge ValueAsgmt(?b) \wedge isResultOf(?b, ?a) \wedge endTime(?a, ?value) \rightarrow resultTime(?b, ?value)$
(a)	Para quaisquer AsgnEvent(a) e ValueAsgmt(b) , ValueAsgmt(b) resulta de AsgnEvent(a) , e o evento AsgnEvent(a) possui instante final t, então o valor atribuído ValueAsgmt(b) deve ter o mesmo instante t registrado como seu tempo de designação de resultado.

3 CONCLUSÃO

O presente relatório apresentou a *Especificação do Procedimento Sistemático para apoiar a interoperabilidade para SoS de C2 (nível lógico)*. Diferente do procedimento de nível conceitual, com base no qual foram desenvolvidas cinco (5) ontologias de C2, o procedimento de nível lógico busca operacionalizar as ontologias, isto é, transformá-las em uma linguagem que seja processável por máquina. Para que tal ontologia seja interoperável, é recomendável utilizar uma linguagem rica em semântica, como a OWL.

Nesse sentido, foram operacionalizadas 3 das ontologias mencionadas, HINT/QVAS e MISCON. Durante a operacionalização dessas ontologias, algumas representações que são feitas no nível conceitual precisam ser simplificadas ou adaptadas no nível lógico. Para efetuar essas simplificações foram realizados alguns procedimentos que foram identificados e apresentados como diretrizes na seção 2.1.

Os resultados alcançados com a aplicação dos procedimentos deste e do relatório anterior estão relatados no *Relatório de Construção da Prova de Conceito* do projeto S2C2. Os artefatos e publicações gerados nesse contexto estão disponíveis através do site¹ e do repositório². Adicionalmente, as ontologias operacionais possuem URLs permanentes, cadastradas na Internet Archive³, para permitir o acesso à comunidade científica, seguindo as recomendações dos princípios FAIR.

Como infraestrutura necessária para conceber e operacionalizar as ontologias foram necessários os seguintes softwares:

- VISUAL PARADIGM (VP) - versão 17 ou superior
<https://www.visual-paradigm.com/download/>
- Plug-in ONTOUML para o VISUAL PARAGIGM - versão compatível com o VP
<https://github.com/OntoUML/ontouml-vp-plugin>
- PROTÉGÉ - versão 4 ou superior com raciocinadores
<https://protege.stanford.edu/software.php>
- Plug-in SWRLTAB para o PROTÉGÉ - versão compatível
<https://github.com/protegeproject/swrltab-plugin>
- PYLODE - versão 3.x ou superior
<https://github.com/RDFLib/pyLODE>

¹ <https://comp-ime-eb-br.github.io/S2C2-IME/>

² <https://github.com/comp-ime-eb-br/S2C2-IME>

³ <https://purl.archive.org/>

- Plataforma de Hospedagem de Software GitHub <<https://github.com/comp-ime-eb-br/S2C2-IME>>

Hardware

Para instalar os softwares listados e realizar o desenvolvimento e/ou experimentos com as ontologias geradas, um computador com a seguinte configuração de hardware ou superior deve ser suficiente:

- Processador: 8 cores / 16 threads
- RAM: 64 GB DDR5 6000 MHz
- GPU: RTX 3060 12 GB ou equivalente para tarefas de Machine learning
- Armazenamento: NVMe 2 TB PCIe 4.0 (sistema + projetos + Dockers)
HDD 8 TB (base de dados 1 TB+, arquivos, backups)

REFERÊNCIAS

- 1 FALBO, R. d. A. Sabio: Systematic approach for building ontologies. In: GUIZZARDI, G.; PASTOR, O.; WAND, Y.; CESARE, S. de; GAILLY, F.; LYCETT, M.; PARTRIDGE, C. (Ed.). *Proceedings of the 1st Joint Workshop ONTO.COM / ODISE on Ontologies in Conceptual Modeling and Information Systems Engineering 2014*. CEUR-WS.org, 2014. (CEUR, 1). March, 2019. Disponível em: <<http://ceur-ws.org/Vol-1301/ontocomodise2014\2.pdf>>.
- 2 ALMEIDA, J.; GUIZZARDI, G.; SALES, T.; FALBO, R. *gUFO: A Lightweight Implementation of the Unified Foundational Ontology (UFO)*. 2019. Disponível em: <<http://purl.org/nemo/doc/gufo>>.
- 3 DEMORI, A. M. *UMA ABORDAGEM BASEADA EM ONTOLOGIA PARA REPRODUÇÃO DE CENÁRIOS DE OPERAÇÕES MILITARES*. Dissertação (Mestrado) — Instituto Militar de Engenharia (IME), 12 2023. Disponível em: <https://github.com/comp-ime-eb-br/S2C2-IME/blob/main/publi/Disserta%C3%A7%C3%A3oAndreMunizDemori_IME_versao_final.pdf?raw=true>.
- 4 TESOLIN, J. C. C. *Towards A Mobile Wireless Network Ontology For Radio Access Points Selection Supported By Semantic Reasoning*. Tese (Doutorado) — Instituto Militar de Engenharia, 07 2024. Disponível em: <https://github.com/comp-ime-eb-br/S2C2-IME/blob/main/publi/20240704_Tese_JTesolin.pdf?raw=true>.
- 5 MUSEN, M. A. The protégé project: a look back and a look forward. *AI Matters*, v. 1, n. 4, p. 4–12, 2015. Disponível em: <<https://doi.org/10.1145/2757001.2757003>>.