UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

GUILHERME MELO E MARANHÃO

# Interpretação e disseminação de contexto: filtragem semântica, projeto arquitetural e estudo de caso em Saúde

Goiânia
2015

**TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR AS TESES E DISSERTAÇÕES ELETRÔNICAS (TEDE) NA BIBLIOTECA DIGITAL DA UFG**

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

**1. Identificação do material bibliográfico**: **[X] Dissertação** **[ ] Tese**

**2. Identificação da Tese ou Dissertação**

| | | | | | |
|---|---|---|---|---|---|
| Autor (a): | Guilherme Melo e Maranhão | | | | |
| E-mail: | guilherme82@gmail.com | | | | |
| Seu e-mail pode ser disponibilizado na página? [ **X** ]Sim [ ] Não | | | | | |
| Vínculo empregatício do autor | nenhum | | | | |
| Agência de fomento: | | | | Sigla: | |
| País: | | UF: | | CNPJ: | |
| Título: | Interpretação e disseminação de contexto: filtragem semântica, projeto arquitetural e estudo de caso em Saúde | | | | |
| Palavras-chave: | Interpretação de contexto, filtro de eventos, web semântica, projeto arquitetural, avaliação | | | | |
| Título em outra língua: | Context reasoning and dissemination: semantic filtering, architectural design and case study in Health | | | | |
| Palavras-chave em outra língua: | Context reasoning, event filter, semantic web, architectural design, evaluation | | | | |
| Área de concentração: | Ciência da Computação | | | | |
| Data defesa: (dd/mm/aaaa) | 06/05/2015 | | | | |
| Programa de Pós-Graduação: | Mestrado em Ciência da Computação | | | | |
| Orientador (a): | Renato de Freitas Bulcão Neto | | | | |
| E-mail: | renato@inf.ufg.br | | | | |
| Co-orientador (a): | | | | | |
| E-mail: | | | | | |

**3. Informações de acesso ao documento:**
Liberação para disponibilização?[1] [**X**] total [ ] parcial

Em caso de disponibilização parcial, assinale as permissões:
[ ] Capítulos. Especifique: _____

[ ] Outras restrições: _____

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF ou DOC da tese ou dissertação.
O Sistema da Biblioteca Digital de Teses e Dissertações garante aos autores, que os arquivos contendo eletronicamente as teses e ou dissertações, antes de sua disponibilização, receberão procedimentos de segurança, criptografia (para não permitir cópia e extração de conteúdo, permitindo apenas impressão fraca) usando o padrão do Acrobat.

_____ Data: / / 2015.
Assinatura do (a) autor (a)

[1] Em caso de restrição, esta poderá ser mantida por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Todo resumo e metadados ficarão sempre disponibilizados.

GUILHERME MELO E MARANHÃO

# Interpretação e disseminação de contexto: filtragem semântica, projeto arquitetural e estudo de caso em Saúde

Dissertação de Mestrado apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Programa De Pós-Graduação Em Ciência Da Computação.

**Área de concentração:** Ciência da Computação.

**Orientador:** Prof. Dr. Renato de Freitas Bulcão Neto

Goiânia
2015

Ficha catalográfica elaborada automaticamente
com os dados fornecidos pelo(a) autor(a), sob orientação do Sibi/UFG.

Melo e Maranhao, Guilherme
Interpretação e disseminação de contexto: filtragem semântica, projeto arquitetural e estudo de caso em Saúde [manuscrito] / Guilherme Melo e Maranhao. - 2015.
154, cliv f.

Orientador: Prof. Dr. Renato de Freitas Bulcão Neto.
Dissertação (Mestrado) - Universidade Federal de Goiás, Instituto de Informática (INF) , Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2015.
Bibliografia. Apêndice.
Inclui siglas, lista de figuras, lista de tabelas.

1. Interpretação de contexto. 2. filtro de eventos. 3. web semântica. 4. projeto arquitetural. 5. avaliação. I. de Freitas Bulcão Neto, Renato, orient. II. Título.

Guilherme Melo e Maranhão

**Interpretação e disseminação de contexto: filtragem semântica, projeto arquitetural e estudo de caso em Saúde**

Dissertação defendida no Programa de Pós-Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Ciência da Computação, aprovada em 06 de maio de 2015, pela Banca Examinadora constituída pelos professores:

Renato Bulcão Neto

**Prof. Dr. Renato de Freitas Bulcão Neto**
Instituto de Informática - UFG
Presidente da Banca

**Prof. Dr. Fábio Moreira Costa**
Instituto de Informática - UFG

**Prof. Dr. Renan Gonçalves Cattelan**
Faculdade de Computação - UFU

**Guilherme Melo e Maranhão**

Graduou-se em Ciência da Computação pela Universidade Federal de Goiás. Possui especialização pela UnB em "Objetos, Sistemas Distribuídos e Internet". Atualmente, é analista de sistemas do Ministério Público do Estado de Goiás.

Dedico este trabalho a Deus, ao qual devo toda honra e glória, à minha amada esposa Bianca, cúmplice neste projeto, e à minha querida filha Lara, a quem quero sempre servir de exemplo.

# Agradecimentos

Agradeço a Deus, refúgio e consolo em todos os momentos.

À minha esposa Bianca, pelas palavras de incentivo e pelos gestos de amor, cuidado e companheirismo, se sacrificando, mesmo em meio ao seu mestrado, para que, juntos, concluíssemos esse projeto.

À minha filha Lara, sempre carinhosa e alegre, que soube compreender, mesmo pequenina, os vários afazeres do pai neste período.

Aos meus pais, irmã e cunhado, pelas orações, favores e palavras de apoio neste momento tão atarefado. Em especial, à minha querida mãe Kenize, que, prontamente, sempre nos auxiliou nos cuidados com a Lara e em outras necessidades cotidianas.

À família da minha esposa, pelas orações e palavras de incentivo.

A todos os meus amigos e demais familiares.

Ao meu orientador, Prof. Dr. Renato de Freitas Bulcão Neto, que sempre me incentivou, orientou e capacitou para a conclusão deste trabalho. A sua experiência e companheirismo foram fundamentais em cada etapa desse curso.

Aos amigos Cláudio e Ernesto, pelos momentos de estudo, aprendizado e descontração.

Aos profissionais de enfermagem do HC/UFG, pelo auxílio e pelas informações relevantes para a construção desta pesquisa.

Aos gestores do MP-GO, que concederam as liberações necessárias para comparecimento às aulas, reuniões e demais obrigações do mestrado.

# Resumo

A computação sensível a contexto é a área da Ciência da Computação que estuda os meios para possibilitar a interação entre dispositivos computacionais e o ambiente, ou contexto, em que estão inseridos. Para promover essa interação, a literatura define quatro etapas para o ciclo de vida da informação de contexto: aquisição, modelagem, interpretação e disseminação. Dentre essas etapas, as de interpretação e disseminação têm sido fortemente impactadas pela heterogeneidade dos dados publicados, resultante da crescente implantação de sensores. Criar mecanismos para manipular e interpretar essas informações e depois notificá-las de maneira eficiente e com relevância para as aplicações sensíveis a contexto tem sido um desafio reportado pela ciência. Diante do problema apresentado, esse trabalho objetiva alinhar os princípios de projeto recomendados pela literatura com uma proposta de abordagem para interpretação e filtragem de contexto fundamentados em modelagem contextual semântica. Portanto, as principais contribuições deste trabalho incluem um mecanismo extensível e manutenível de filtragem semântica de contexto; o modelo arquitetural de um componente interpretador de contexto, denominado *Hermes Interpreter*, desenvolvido a partir de princípios de projeto defendidos da literatura; o componente *Hermes Base*, que expõe API de acesso para um *middleware* de comunicação, que opera sob o paradigma *publish/subscribe*; e por último, validações funcionais e de desempenho do *Hermes Interpreter* em cenário de simulação de monitoramento de sinais vitais de pacientes em UTIs e enfermarias. Por meio delas, comprovou-se a eficiência do mecanismo de filtragem para notificar de maneira precisa e eficiente os dados relevantes para os usuários. Além disso, observou-se que, devido a características de implementação, por maior que seja a quantidade de assinantes, o tempo de resposta foi aceitável em praticamente todos os experimentos. Apesar disso, constatou-se o alto custo do processamento de filtros em comparação às atividades puramente de interpretação de contexto. Por último, o projeto arquitetural do componente constitui um artefato reutilizável para pesquisas na área de intepretação de contexto.

**Palavras–chave**

Interpretação de contexto, filtro de eventos, Web Semântica, projeto arquitetural, avaliação

# Abstract

Context-aware computing investigates how to seamlessly enable the interaction among computing devices and the environment, or context, in which they are located in. The literature has defined the life cycle of context data as composed of four phases: acquisition, modelling, reasoning and dissemination. These latter two phases have been strongly influenced by the heterogeneity of published data, as a consequence of the increasing deployment of sensors. A great challenge reported in the literature has been the development of mechanisms for handling and reasoning about sensor data, and also for notifying context-aware applications in an efficient and relevant manner. Aiming to solve this problem, this research focuses on a new approach for semantic context reasoning and filtering in accordance with design principles recommended by the literature. As a result of this work, the main contributions include an extensible and maintainable mechanism for semantic context filtering; the architectural model of a context reasoning component, called Hermes Interpreter (HI); the Hermes Base component, which exposes an API for accessing a publish-subscribe-based communication middleware; and HI's functional validation and performance evaluation in a simulated scenario of vital signs monitoring in Intensive Care Units and wards. This research demonstrated the efficiency of the behaviour of the semantic filtering mechanism for end users applications. Besides, by increasing the number of subscribers, it was observed that the response time was acceptable in almost all experiments. Despite this, it was also verified the high cost of the semantic filtering processing in comparison with pure context reasoning activities. Regarding the HI component's architectural design, this work recommends it as a reusable artifact for researches on the subject of context reasoning.

**Keywords**

Context reasoning, event filter, Semantic Web, architectural design, evaluation

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Códigos de Programas

# Lista de Siglas

API - *Application Programming Interface*
CSV - *Comma-separated values*
DDS - *Data Distribution Service*
EPL - *Event Processing Language*
IEC - *International Electrotechnical Commission*
IoT - *Internet of Things*
IRI - *Internationalized Resource Identifier*
ISO - *International Organization for Standardization*
JSON - *JavaScript Object Notation*
HB - *Hermes Base*
HI - *Hermes Interpreter*
HTTP - *Hypertext Transfer Protocol*
HW - *Hermes Widget*
LAN - *Local Area Network*
MQTT - *MQ Telemetry Transport*
MSVH - Monitoramento de Sinais Vitais Humanos
OMG - *Object Management Group*
OSSIM - *Open Source Security Information Management*
OWL - *Web Ontology Language*
POJO - *Plain Old Java Object*
QoS - Qualidade de Serviço
RDF - *Resource Description Framework*
RDFS - *Resource Description Framework Schema*
SeVocab - *Software and Systems Engineering Vocabulary*
SIG - *Software Improvement Group*
SPARQL - *Simple Protocol and RDF Query Language*
SQL - *Structured Query Language*
SUS - *System Usability Scale*
SWRL - *Semantic Web Rule Language*
TAM - *Technology Acceptance Model*
Turtle - *Terse RDF Triple Language*
UTI - Unidade de Terapia Intensiva
W3C - *World Wide Web Consortium*
WAN - *Wide Area Network*
XML - *Extensible Markup Language*

CAPÍTULO 1

# Introdução

A Computação Ubíqua, vislumbrada pelo cientista Mark Weiser [48] em 1991, foi assim definida em seu clássico trabalho: *"Elementos especializados de hardware e software, conectados por fios, ondas de rádio e de infravermelho, serão tão ubíquos que ninguém perceberá suas presenças"*. Dentre as linhas de estudo da Computação Ubíqua, destaca-se a Computação Sensível a Contexto [34], que foi descrita por Dey et al. [10] como a área que estuda os mecanismos para fornecimento e utilização de informações de contexto, a fim de oferecer serviços e informações relevantes a usuários e a outras aplicações na realização de alguma tarefa.

Conforme a literatura [10], contexto[1] é qualquer informação - por exemplo, identidade e localização - que possa ser utilizada para caracterizar a situação de uma entidade. Uma entidade pode ser uma pessoa, lugar, objeto físico ou computacional considerado relevante em uma interação usuário-aplicação.

Como exemplo de solução sensível a contexto, pode-se citar a arquitetura EXEHDA-TG [44] cuja finalidade é automatizar a verificação do processo de terapia medicamentosa em hospitais, reduzindo assim os erros dessa administração, pois permite aos médicos observarem se o efeito desejado dos medicamentos está sendo atendido. Para isso, EXEHDA-TG acompanha a evolução dos seus sinais vitais em função dos medicamentos, emitindo alertas conforme um sinal vital indique uma resposta inesperada ao medicamento.

Outro exemplo de aplicação é a descrita em Boufleuer et al. [8], que utiliza informações de umidade, pluviometria e outros dados de contexto com a finalidade de fornecer informações eficientes sobre o nível de irrigação que deve ocorrer para determinado contexto identificado, além de possuir automonitoramento sobre o consumo de energia dos diversos sensores implantados no meio agrícola monitorado.

Para que as informações de contexto sejam entregues às aplicações, os projetos, em geral, adotam o ciclo de vida de contexto representado na Figura 1.1:

[1] Ao longo do texto, os termos contexto e informação de contexto serão utilizados indiscriminadamente.

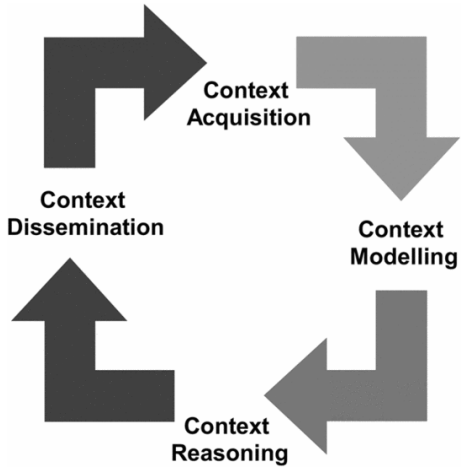


**Figura 1.1:** *Ciclo de vida de contexto tradicional [34]*

Resumidamente, o dado de contexto é adquirido de alguma fonte de dados que podem ser sensores, preferências de usuários, ou serviços web. Após a aquisição, essa informação é representada conforme o tipo de modelagem corrente do sistema que podem ser linguagens de marcação, ontologias [15] ou esquemas relacionais.

Na próxima etapa, o contexto modelado é submetido a um ou mais processos de inferência[2] para que novos fatos sejam produzidos. Os dados de contexto, ao iniciarem essa fase, são considerados de baixo nível e, em determinados cenários, de pouca validade para os usuários, como por exemplo: em um sistema de localização de pontos turísticos, as coordenadas geográficas enviadas por um GPS podem não ter sentido algum para um turista. Mas essa informação, juntamente com os dados do local inseridos em uma base de dados ou servidor web, podem resultar em uma infinidade de opções turísticas para o usuário. Em outro exemplo, no cenário de monitoramento de sinais vitais, uma medida aferida isoladamente pode ser simplesmente um valor numérico. Quando submetido à etapa de interpretação, esse valor juntamente com parâmetros pré-configurados e perfil do paciente podem significar uma grave anormalidade.

Esses fatos, sendo de relevância para as aplicações, são a elas notificados ou transmitidos por meio de consultas. Para que esses dados sejam enviados às aplicações, inicia-se a etapa de disseminação[3] de contexto, que é viabilizada por soluções de *middleware*, com suporte ao paradigma *publish/subscribe*, por exemplo, ou por meio de serviços web. A partir do momento que esse dado é acessado pelas aplicações, inicia-se um novo ciclo de vida da informação de contexto.

[2] Ao longo do texto, os termos inferência e interpretação serão utilizados indiscriminadamete.
[3] Ao longo do texto, os termos disseminação e distribuição serão utilizados indiscriminadamente.

## 1.1 Motivação

Devido à grande implantação de sensores para as mais diversas finalidades, tais como saúde[52], localização[28], segurança[51], trânsito[45] e entretenimento [34], aliada com a profusão de dispositivos móveis na sociedade, tornam a computação sensível a contexto um campo propício para a pesquisa e o desenvolvimento de soluções.

Duas consequências diretas desse cenário, de acordo com Sehic et al. [42] e Perera et al. [34], são o grande volume de dados gerados e publicados e a grande heterogeneidade dessas informações, cujas distinções vão desde os seus formatos até os seus significados propriamente ditos: coordenadas geográficas, perfis pessoais, medidas de sinais vitais, umidade ou tráfego urbano.

Essas consequências têm impactado cada etapa do ciclo de vida de contexto apresentado na Figura 1.1. Da fase de aquisição, um dos principais impactos é a geração de dados conflituosos ou imprecisos, resultantes de leituras equivocadas de sensores.

Para a fase de modelagem, esse cenário produz domínios contextuais cada vez mais complexos, devido à grande distinção semântica entre as informações. Em cenário de monitoramento de sinais vitais humanos, por exemplo, essa modelagem deve incorporar informações de medidas de sinais vitais, anormalidades associadas, perfis de pacientes, perfis de profissionais de saúde, localização, dentre outros [4]. Para isso, a literatura tem proposto a evolução quanto à expressividade, robustez, formalismo e uso de padrões na representação de informação contextual [42], com a finalidade de abranger a totalidade dos conceitos, entidades, relacionamentos e atributos do domínio ao qual o sistema está inserido. *Combinar e representar esse volume de informações de maneira que esses dados possam ser futuramente inferidos, consultados e disseminados com qualidade, precisão e relevância* estão entre os desafios dessa etapa.

As consequências para a etapa de interpretação de contexto se assemelham às apresentadas para a fase de modelagem. Domínios cada vez mais complexos demandam por maior quantidade de situações novas, implícitas e relevantes que necessitam ser inferidas. O desafio para essa etapa, portanto, é utilizar e combinar as técnicas de inferência apropriadas para o contexto corrente. Várias são as técnicas existentes para contemplar as diversas situações de contexto, as quais estão resumidamente descritas na sequência:

- Aprendizado supervisionado: ideal para reconhecimento de atividades humanas [11]
- Aprendizado não supervisionado: ideal para detecção de comportamentos não usuais [24]
- Regras: Apropriado para identificação de eventos, em que situções de baixo nível devem ser convertidas em situações de alto nível [46]

- Lógica *Fuzzy*: Inferir situações imprecisas e lidar com incertezas [37]
- Ontologia: é uma especificação explícita de um conceito [15], que utiliza um vocabulário pré-definido de termos para definir entidades e seus relacionamentos dentro de um domínio específico. Aplicada em cenários em que, para se inferir situações e produzir fatos novos, é preciso o conhecimento do domínio do contexto [46]
- Lógica probabilística: Para situações em que se conhecem as probabilidades de fatos ocorrerem é preciso combinar as evidências para determinar um evento [33]

*Combinar esses mecanismos, aproveitando as vantagens de cada um para o cenário que se deseja inferir*, tem sido um desafio para a construção de interpretadores de contexto [34].

Finalizando o ciclo de vida do contexto, a fase de disseminação é impactada ao ter que incorporar mecanismos para que o contexto correto seja entregue para as aplicações corretas. A complexidade crescente dos domínios que resultam em grandes quantidades de novas situações produzidas por meio da etapa de inferência leva ao desafio de *determinar o que e para quem devem ser disseminadas, ou distribuídas, todas essas informações*.

Essa etapa, portanto, demanda por *soluções que possibilitem às aplicações sensíveis a contexto especificarem, com alto grau de detalhamento, os eventos de que desejam ser notificadas*. Retomando o exemplo do monitoramento de sinais vitais, qual a necessidade do enfermeiro assistente da paciente Maria ser notificado dos sinais vitais do paciente João? Ou, mais especificamente, qual a necessidade de um enfermeiro ser notificado de todas as aferições de todos sinais vitais de um paciente, se o que é relevante é ser informado apenas da ocorrência de *hipotensão* resultante de uma cirurgia cardíaca? A essa capacidade de parametrização dos dados que devem ser disseminados, dá-se o nome de *filtragem de contexto*.

Se considerar a modelagem ontológica, recomendada para domínios complexos [34], essa alta especificidade, por vezes, não é atendida pelo paradigma convencional *publish/subscribe*. Esse paradigma, via de regra, oferece apenas combinação textual ou operações lógicas e matemáticas simples para que as aplicações assinantes especifiquem os seus eventos de contexto de interesse, limitando assim, o potencial que a modelagem semântica proporciona.

Faz-se necessária, portanto, para sistemas sensíveis a contexto que utilizam a modelagem ontológica, uma camada que alinhe a ontologia do contexto aos tópicos oferecidos pelo sistema. Essa camada, portanto, deve oferecer às aplicações, de forma transparente, *a capacidade de filtrarem, por meio da semântica do contexto, os eventos de contexto que desejam ser notificados*. Para prover dinamicidade às aplicações com respeito à capacidade de filtragem, esse mecanismo deve ainda ser *extensível e manutenível* para

se adequar às alterações da modelagem ontológica do contexto, bem como *flexível* quanto ao mecanismo de inferência que deve ser executado para que seja possível identificar os parâmetros de filtro solicitados pelos assinantes.

Outras vantagens de se implementar a filtragem são a redução do tráfego de dados, pois somente os dados relevantes são comunicados; e a redução do consumo de recurso computacional nos dispositivos das aplicações, pois a filtragem retira dessas aplicações a necessidade de terem que selecionar os eventos e dados de contexto relevantes para suas atividades.

As exigências apontadas para cada uma das etapas apresentadas orientam para a necessidade de infraestruturas reutilizáveis que forneçam, de forma transparente, todos os serviços e mecanismos anteriormente identificados para aplicações sensíveis a contexto, com as quais os usuários interagirão por meio dos diversos dispositivos computacionais. A principal vantagem para as aplicações, ao se beneficiarem de tais serviços, é poderem se concentrar nas funcionalidades específicas de suas atuações finais e nas respectivas interações com os usuários.

Para complementar o requisito de *filtragem semântica de contexto*, Perera et al.[34] e Bettini et al.[6] enumeram outros princípios de projeto para nortear o desenvolvimento de infraestruturas sensíveis a contexto, dos quais destacam-se:

- Arquitetura em camadas com componentes reutilizáveis
- Extensibilidade
- Independência entre modelagem de contexto e código executável
- Suporte a múltiplas técnicas de interpretação de contexto
- Monitoramento e detecção de eventos
- Verificação de consistência entre fatos e modelo de contexto subjacente

Retomando a Figura 1.1, atesta-se a importância de haver um componente de software exclusivo para interpretação de contexto com capacidade para filtragem semântica. Esse componente ainda deve operar de forma desacoplada do restante da infraestrutura para prover o reúso, extensibilidade e flexibilidade [34] [10]. Componentes para essa finalidade, normalmente, são apresentados na literatura como caixas-pretas, em que as mensagens de contexto fluem sem que o leitor tenha conhecimento de suas camadas internas, os padrões e tecnologias escolhidos e como eles são utilizados para efetuar a interpretação, limitando assim, o compartilhamento do conhecimento e a possibilidade de reúso para desenvolvedores e pesquisadores da área.

## 1.2 Objetivos

O objetivo geral deste trabalho consiste em elaborar uma nova abordagem de filtragem para apoio à disseminação de contexto, associada à interpretação e modelagem semânticas. Como objetivos específicos, têm-se:

- aderir os resultados desta pesquisa quanto à interpretação e disseminação de contexto segundo princípios de projeto recomendados na literatura, de maneira a verificar a manutenibilidade e extensibilidade do mecanismo quanto ao modelo de contexto ontológico e a flexibilidade com relação às técnicas de inferência suportadas
- validar os resultados desta pesquisa por meio de um estudo de caso de saúde que permita explicitar as contribuições para as fases de interpretação e disseminação de contexto

## 1.3 Materiais e Métodos

Esta seção descreve o método de pesquisa adotado em conjunto com as tecnologias empregadas para que os objetivos fossem atendidos:

1. Os requisitos integrantes do arcabouço conceitual de Dey et al. [10] foram o ponto de partida para se delimitar o conjunto de capacidades que uma infraestrutura de apoio ao ciclo de vida de aplicações sensíveis a contexto deveria ter. A arquitetura de componentes do Context Toolkit [10], do mesmo autor, serviu também como guia para o projeto arquitetural dessa infraestrutura, tendo como foco o componente interpretador de contexto
2. Em seguida, identificou-se a necessidade de estudos sobre outras infraestruturas, a fim de identificar pontos em comum, pontos em aberto e as capacidades que a literatura apontava como essenciais e desejáveis para uma infraestrutura sensível a contexto como também para um componente interpretador
3. Como premissa do trabalho e comprovado pela revisão bibliográfica, foi escolhida a modelagem ontológica como a padrão para especificar as informações de contexto na infraestrutura. Por meio dessa forma de modelagem, o domínio das informações pode ser representado com um formalismo tal que favorece a inferência de situações novas a partir de fatos instanciados. Além disso, é uma solução manutenível e de fácil compartilhamento entre sistemas. Para processar a semântica do contexto, descrita em ontologia e regras, adotou-se o suporte a padrões da Web Semântica[4],

[4] http://www.w3.org/standards/semanticweb/

como RDF [50], RDFS [17], OWL [49], SWRL [22], SPARQL [19], bem como suporte a máquinas de inferência com capacidade para tal. A escolha dessas tecnologias foi para padronização de informação no nível estrutural, semântico e lógico, como também prover facilidades de interoperabilidade

4. Devido às características identificadas de aquisição e disseminação das informações de contexto, a forma de transmissão escolhida para ser a padrão da infraestrutura foi a via *middleware* de comunicação com suporte ao paradigma *publish/subscribe*. A especificação de *middleware* DDS [41], elaborada pela OMG[5], foi a escolhida pois atendeu às demandas do projeto, além de se tratar de um padrão aberto da indústria para comunicação distribuída. Para desacoplar o *middleware* DDS do restante da infraestrutura, foi implementado um componente de software que se comunica diretamente com o *middleware* e expõe uma API independente desse para que os demais componentes e aplicações se comuniquem na infraestrutura
5. O projeto e a implementação do componente interpretador de contexto objetivaram desenvolver uma solução desacoplada do restante da infraestrutura, extensível e flexível para trabalhar com múltiplas técnicas de interpretação de contexto, independente de modelo ontológico de contexto, flexível para manipular ontologias de contexto distintas simultaneamente e suportassem filtragem semântica de contexto. Para atender a essas exigências, a arquitetura foi concebida sob padrões de projeto clássicos da Engenharia de Software, apoiada em tecnologias e especificações, dentre as quais se destacam JSON e SPARQL
6. Realizada avaliação do componente quanto à manutenibilidade conforme norma ISO/IEC 25000:2011[6] por meio de software *SonarQube*[7] e tendo o *Modelo de Manutenibilidade SIG* [21] como referência para pontuação das respectivas subcaracterísticas de manutenibilidade prescritas na norma
7. Os componentes foram validados em cenário que simula monitoramento de sinais vitais humanos por meio de casos de teste que descrevem cenários para validar as características exigidas para os componentes implementados. Essa etapa também validou o componente *Hermes Interpreter* com relação ao seu desempenho frente à inclusão de assinantes na infraestrutura, visando identificar a sua capacidade para atender as exigências da literatura médica quanto ao tempo de resposta para notificação de alarmes. Essa validação teve a participação de enfermeiros que orientaram os procedimentos a serem observados para que a simulação fosse a mais próxima possível da realidade dos plantões de UTIs e enfermarias, considerando a infra-

[5] http://portals.omg.org/dds/

[6] http://www.iso.org/iso/home/store/catalogue$_$tc/catalogue$_$detail.htm?csnumber=64764

[7] http://www.sonarqube.org/

estrutura do projeto disponível. Essa validação também abordou um comparativo entre o tempo dispendido para executar as atividades de inferência e as atividades de filtragem

## 1.4 Contribuições

Considera-se como contribuições deste trabalho de pesquisa:

1. Um mecanismo de filtragem semântica de contexto para notificação de eventos de interesse das aplicações; essa abordagem é relevante para apoiar a fase de disseminação de informações de contexto em cenários nos quais os assinantes possuem necessidades de contexto distintas entre si. Conforme a Figura 1.2, esse mecanismo se comporta como uma camada semântica para notificação de eventos de contexto

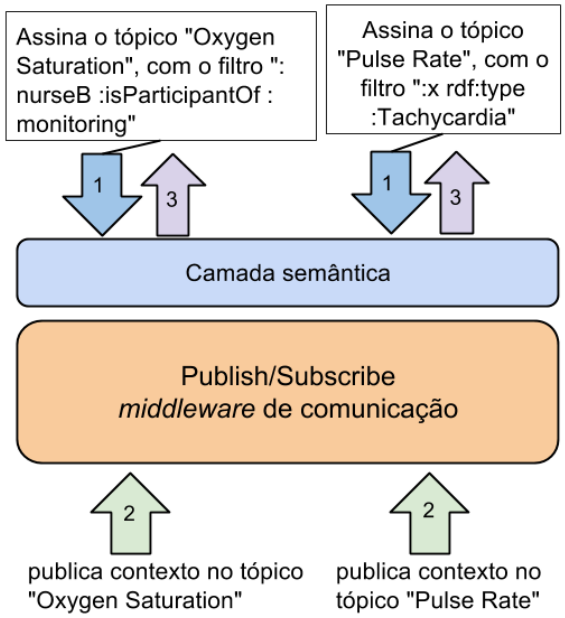


**Figura 1.2:** *Mecanismo semântico de filtragem para disseminação de contexto.*

2. O modelo arquitetural do componente *Hermes Interpreter*, que do ponto de vista dos desenvolvedores de sistemas sensíveis a contexto, produz dois benefícios diretos: o reúso da arquitetura proposta para desenvolvimento de interpretadores e o reúso de um componente, em prol de aplicações, o qual interpreta e filtra a semântica de contexto e gerencia a distribuição de eventos [27]
3. O componente *Hermes Base* que viabiliza o desacoplamento entre o *middleware* de comunicação e o restante da infraestrutura *Hermes*, por meio da exposição de API para efetuar a comunicação via paradigma *publish/subscribe* [27]

4. Teste e validação do componente *Hermes Intepreter* em um cenário de monitoramento de sinais vitais humanos, com a participação de profissionais da saúde
5. Avaliação de desempenho do componente *Hermes Interpreter* no cenário de monitoramento de sinais vitais

## 1.5 Estrutura da dissertação

Esta dissertação de mestrado está assim organizada: o Capítulo 2 discorre sobre a fundamentação teórica na qual se apoia a proposta do trabalho; o Capítulo 3 apresenta a infraestrutura *Hermes*; o Capítulo 4 detalha o *Hermes Interpreter*; o Capítulo 5 apresenta o estudo de caso de aplicação da arquitetura; o Capítulo 6 aborda trabalhos relacionados e, o Capítulo 7, as conclusões e limitações deste trabalho de pesquisa, bem como propostas de trabalhos futuros.

CAPÍTULO 2

# Fundamentação Teórica

Este capítulo descreve conceitos e tecnologias que servem de fundamentação para o desenvolvimento da infraestrutura proposta: a arquitetura de referência do *Context Toolkit* [10], padrões para tratamento de semântica de informação, padrões de projeto e componentização de software, padrões para intercâmbio e descrição de dados e, por fim, uma especificação de *middleware* para aquisição e distribuição de contexto com suporte ao paradigma *publish/subscribe*.

## 2.1 *Context Toolkit*

Em seu trabalho, Dey et al. [10], dentre os pioneiros na pesquisa em computação sensível a contexto, elencam os requisitos para construção desse tipo de software. São eles:

1. Especificação de informações de contexto
2. Separação entre aquisição e utilização de informações de contexto
3. Interpretação de informações de contexto
4. Comunicação distribuída e transparente
5. Disponibilidade contínua de componentes de captura de informações de contexto
6. Armazenamento de informações de contexto
7. Descoberta de recursos

O *Context Toolkit* consiste em um arcabouço de componentes de software que implementam os requisitos descritos, fornecendo serviços para a descoberta, obtenção, agregação e interpretação de contexto. Tendo como base de seu projeto um *framework* conceitual, também apresentado por Dey et al. [10], o *Context Toolkit* visa fornecer não apenas os serviços básicos necessários, mas principalmente o suporte arquitetural para apoio ao desenvolvimento de aplicações sensíveis a contexto.

Conforme a Figura 2.1, os principais elementos do *framework* são:

- *Widget*: componente para aquisição de contexto, que oferece uma interface uniforme para as aplicações que utilizam contexto, tornando transparente a obtenção

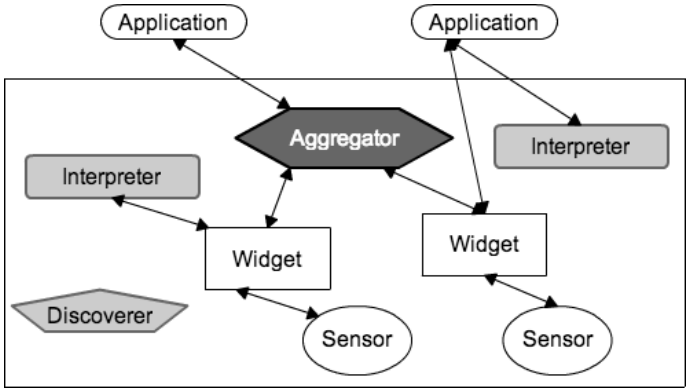


**Figura 2.1:** *Arquitetura do Context Toolkit [10]*

dessas informações a partir de diferentes tipos de sensores e outros provedores de contexto

- *Aggregator*: componente que pode adquirir e agregar contexto de um conjunto de *widgets*
- *Interpreter*: componente responsável pelo mapeamento de representações de contexto e pela realização de interpretações complexas com base em um conjunto de informações de contexto
- *Discoverer*: componente que permite que as aplicações localizem componentes de seu interesse, mesmo que não possuam informações prévias sobre eles. Também facilita a adaptação a mudanças na camada de infraestrutura de sensibilidade ao contexto, como por exemplo a ativação ou desativação de componentes

O trabalho de Dey et al. [10] ressalta a necessidade de identificar, implementar e apoiar as abstrações e serviços adequados para a manipulação do contexto, no sentido de minimizar e/ou extinguir problemas como:

1. Dificuldades na aquisição de contexto, uma vez que existe a necessidade de utilizar uma diversidade de sensores, de acordo com a aplicação
2. Abstrair o contexto para que faça sentido para a aplicação
3. Distribuição e heterogeneidade das fontes de aquisição de contexto
4. Dinamicidade do contexto

A identificação dos problemas básicos quanto à especificação, aquisição e ação em relação ao contexto, juntamente com uma análise minunciosa do processo de projeto de aplicações, permitiram a especificação do *framework* conceitual *Context Toolkit*, para suporte de aplicações sensíveis ao contexto. A infraestrutura criada por Dey [10] trouxe facilidades em relação ao projeto, desenvolvimento e evolução de aplicações sensíveis ao contexto, pois a partir de então, não mais seria responsabilidade dessas, lidar com os detalhes intrínsecos à comunicação, descoberta de recursos e outros serviços pertinentes às infraestruturas sensíveis a contexto.

## 2.2 Web Semântica

Hebeler et al. [20] definem Web Semântica como uma rede de dados descrita e relacionada de tal forma a estabelecer contexto e semântica associados à gramática e construtores de linguagem pré-definidos. Como exemplo, pode-se compará-la à web tradicional, também chamada web sintática. No conceito sintático da web, o termo *livro*, descrito isoladamente dentro de uma página web, só tem significado para seres humanos que compreendem o idioma português. Para que esse dado tenha sentido para as máquinas que processarão aquela página web, ele deveria estar relacionado a outros termos de seu contexto, que da mesma forma, sucessivamente, estariam relacionados a outros termos que os descreveriam, formando uma grande rede semântica. Nesse exemplo, *livro* agora não seria um elemento isolado, mas estaria associado a elementos como *autor*, *editora*, *assunto* e assim por diante. Essa rede de relacionamentos, portanto, revelaria a semântica do termo *livro*. Além dos termos que agregam significado ao termo central *livro*, a semântica dos dados só é completa através da especificação dos relacionamentos entre os termos, conforme uma gramática que descreve aqueles termos. Por exemplo, o relacionamento entre *livro* e *autor* seria declarado como *foiEscritoPor*, e esse tipo de relacionamento estaria definido na gramática *literatura*. Pela Figura 2.2 pode-se comparar a representação isolada de *Livro* com uma representação na Web Semântica.

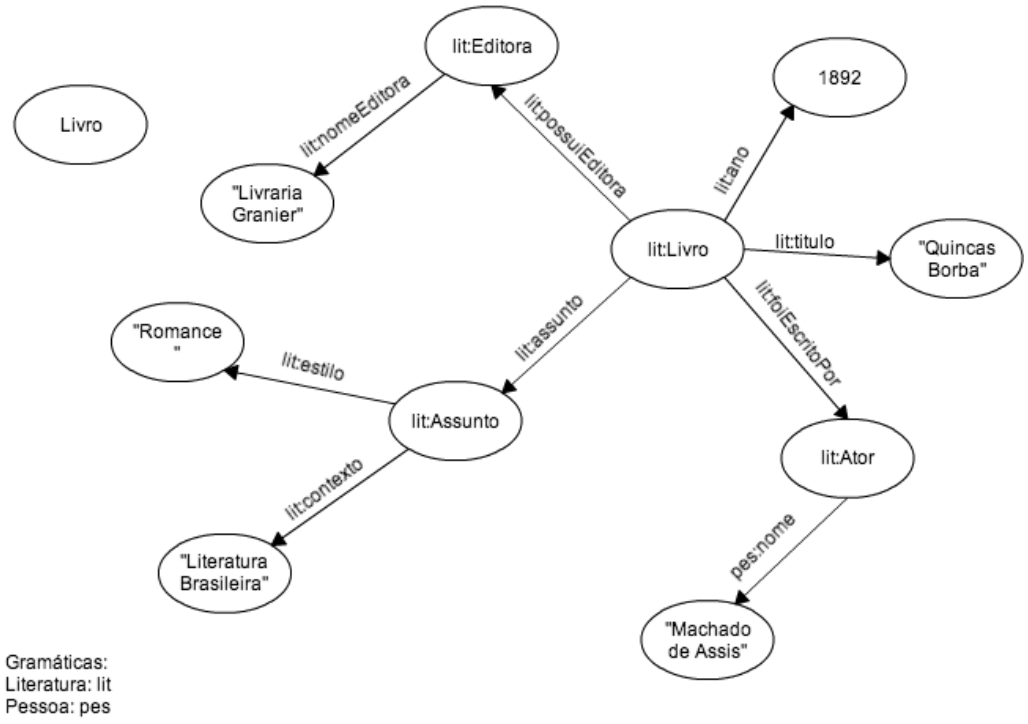


**Figura 2.2:** *Livro descrito isoladamente x Livro na Web Semântica*

Conforme Hebeler et al. [20], uma das formas de se descrever os vocabulários (ou gramáticas) dentro da Web Semântica é por meio de ontologias. A ontologia modela

o conhecimento de um domínio através de entidades e seus relacionamentos. O relacionamento entre duas entidades é denominado declaração ou axioma ontológico. Na Figura 2.2, por exemplo, as gramáticas *Literatura* e *Pessoa* são ontologias dos domínios os quais elas modelam: *Literatura* e *Pessoa*, respectivamente.

Duas das principais características da ontologia são a capacidade para alta *expressividade* e *estruturação*. A expressividade se refere à extensão e facilidade com as quais uma ontologia pode descrever a semântica de um domínio, ou seja, quão numerosos e detalhados são os relacionamentos que podem ser estabelecidos entre os seus elementos. A estruturação diz respeito ao grau de extensão organizacional ou hierárquica de uma ontologia, ou seja, quão específico é o nível de representação de um determinado conceito. A combinação dessas duas características determina o *formalismo* de uma ontologia. Quanto maior o formalismo, maior será o potencial para validação dos fatos de contexto ou instâncias registrados em uma ontologia frente ao seu respectivo esquema ontológico. Também, quanto maior o formalismo, maior será a *capacidade de inferência* da ontologia, conceito esse que será explicado adiante nesta seção.

Além da capacidade de inferência, o formalismo semântico presente nas ontologias possibilita que os sistemas se comuniquem com pouco esforço, pois o conhecimento descrito nos relacionamentos e declarações é compartilhado entre as aplicações, permitindo que as instâncias de dados tenham o mesmo significado e sejam compreendidas entre todos os sistemas comunicantes.

Outra vantagem da utilização de ontologias nas aplicações é o considerável *desacoplamento* que ela provoca entre as regras de negócio que estão modeladas em sua estrutura ou esquema ontológico e o código-fonte, pois a ontologia constitui uma estrutura à parte do código do sistema. A manutenção das regras ontológicas é muito fracamente acoplada ao sistema, sendo até imperceptível em alguns casos. Isso confere à modelagem ontológica a possibilidade de extensão de forma dinâmica e transparente para as aplicações que a utilizam.

Para concretização e difusão da Web Semântica, a W3C tem elaborado diversos padrões e especificações para as mais diversas finalidades e com o objetivo de auxiliar desenvolvedores de software. Algumas dessas recomendações serão brevemente explanadas nas subseções seguintes.

### 2.2.1 RDF

RDF é o padrão da Web Semântica para descrição de dados, os quais, na web, são identificados como recursos. Um recurso na web é referenciado por meio de uma identificação IRI, como por exemplo a IRI `<http://xmlns.com/foaf/0.1>` [20]. Uma IRI é única na web e dessa forma, descreve somente um recurso que pode ser um sujeito,

predicado ou objeto. Essa tripla de informações é a estrutura básica de uma declaração na Web Semântica.

Cada parte dessa tripla tem uma finalidade na representação da informação: o sujeito é a entidade que está sendo descrita, o predicado é o tipo de relacionamento com o objeto, e o objeto é a caracterização do sujeito para o predicado que pode ser um outro recurso ou um literal, por exemplo: a tripla (*pessoa*, *possuiNome*, *José da Silva*) representa o sujeito *pessoa* com predicado *possuiNome* e o objeto literal "José da Silva". O conjunto de triplas e seus relacionamentos constituem um grafo direcionado em que os sujeitos e objetos são os nós e os predicados são as arestas que ligam esses nós, conforme a Figura 2.2.

Devido à utilização de IRIs para descrever os dados aliada à estruturação de dados mediante a abordagem de triplas que permite que quaisquer informações possam ser representadas e interpretadas, o RDF se tornou o *padrão para compartilhamento* de informações na Web Semântica.

RDF possui em sua estrutura uma coleção básica de predicados para serem reutilizados pelas modelagens ontológicas. Um dos mais comuns é o *rdf:type* que conceitualiza o tipo de um sujeito em um domínio específico, por exemplo: (:José, rdf:type, foaf:Person).

### 2.2.2 RDFS

RDFS foi projetado para expressar a semântica das triplas declaradas em RDF por meio de construtores para definição de classes, propriedades, associações e instanciação. Como se referem a especificações básicas de quaisquer ontologias, compreendem uma biblioteca essencial pra interpretação de dados na Web Semântica.

Dentre os seus construtores, destacam-se *rdfs:Class*, que agrupa todas as classes descritas por RDF, ou seja, todas as classes em RDF são instâncias de *rdfs:Class*; *rdfs:subClassOf* que atribui o conceito de especialização de classes em um domínio, por exemplo *:escola rdfs:subClassOf :construção*, e *rdfs:subPropertyOf*, para especializar propriedades ou predicados, por exemplo, *:éFilhoDe rdfs:subPropertyOf :éParenteDe*.

Apesar do formalismo que RDFS atribui aos modelos ontológicos, sua aplicação se restringe à construção de ontologias simples, em que não são necessários maiores detalhamentos para especificar os relacionamentos entre as entidades, como os que serão apresentados na Subseção 2.2.3.

### 2.2.3 OWL

A linguagem OWL, assim como RDFS, tem a finalidade de descrever uma ontologia, porém com maior capacidade de expressão. Para isso, estende RDFS e incor-

pora construtores para especificação mais detalhada de relacionamentos e entidade. Para exemplificar o potencial da linguagem, expandir-se-á a semântica das declarações RDFS citadas na subseção anterior aplicadas a alguns construtores de OWL:

- Simetria: considerando que em determinada ontologia esteja declarado que *:filho :éParenteDe :pai* e que a propriedade *:éParenteDe* seja simétrica, então a declaração *:pai :éParenteDe :filho* é válida
- Funcional: Declara que um indivíduo só pode estar associado a um único objeto ou literal de uma propriedade *funcional*, porém outros indivíduos podem compartilhar aquele mesmo objeto ou literal para a mesma propriedade, como por exemplo: a propriedade *:possuiPaiBiológico* é *funcional* porque um *filho* só tem um *pai biológico*, porém seus irmãos biológicos, caso tenha, também possuirão o mesmo *pai biológico*
- Transitividade: *:escola :contém :sala de aula* e *:sala de aula :contém :computador*, se a propriedade *:contém* for declarada como transitiva, então *:escola :contém :computador*
- Cardinalidade máxima nos relacionamentos: a propriedade *:temAlunos* para *sala de aula*, teria cardinalidade 40
- Disjunção entre classes e propriedades: possibilita especificar que um indivíduo de uma classe *A* nunca será de outra classe disjunta de *A* ou que o objeto ou literal de uma propriedade *P* nunca será o objeto ou literal de outra propriedade disjunta de *P*, como por exemplo: *:mãe owl:disjointWith :pai* ou *:hasPai owl:disjointWith :hasMae*

Esses são alguns exemplos que demonstram a maior expressividade de OWL frente a RDFS, o que permite que a primeira seja utilizada para representar domínios ontológicos mais complexos do que a segunda.

Na próxima subseção, será apresentada a especificação SWRL, a qual complementa os conceitos descritos por OWL.

### 2.2.4 SWRL

A linguagem SWRL é a especificação W3C para agregar regras à descrição ontológica via OWL. O acréscimo de regras expande ainda mais a capacidade de mapeamento do conhecimento [7] , pois SWRL contém descritores ausentes nos padrões citados. SWRL possui uma linguagem própria para expressar regras no formato *body* $\longrightarrow$ *head*, em que a parte *body* contém os antecedentes da regra, ou seja, as cláusulas condicionais e a parte *head*, as cláusulas consequentes. Uma regra trivial para se observar esses conceitos é apresentado no Código 2.2, em que se verifica se o elemento *x1* tem algum

tio. Os antecedentes, parte *body* da regra, são *hasParent (x1, x2) e hasBrother (x2, x3)* e o consequente, parte *head*, é *hasUncle(x1, x3)*.

**Código 2.1** Regra SWRL para verificar se *x1* tem tio

```
<ruleml:imp>
  <ruleml:_rlab ruleml:href="#example1"/>
  <ruleml:_body>
    <swrlx:individualPropertyAtom  swrlx:property="hasParent">
      <ruleml:var>x1</ruleml:var>
      <ruleml:var>x2</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom  swrlx:property="hasBrother">
      <ruleml:var>x2</ruleml:var>
      <ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:individualPropertyAtom  swrlx:property="hasUncle">
      <ruleml:var>x1</ruleml:var>
      <ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

Complementando os exemplos de relações familiares citados com RDFS e OWL nas subseções anteriores, a relação *hasUncle(x1, x3)* não é possível de ser descrita em nenhuma dessas linguagens. A causa disso é que em OWL puramente não há construtores lógicos que permitam relacionar as propriedades *hasParent* com *hasBrother* para se inferir *hasUncle*.

A solução, portanto, para representar conceitos e relacionamentos mais complexos é fazê-la por meio de regras no formato SWRL que, além da vantagem anteriormente exposta, SWRL é uma linguagem baseada em OWL, o que possibilitará aos sistemas manterem o mesmo formalismo, padronização e interoperabilidade de que se beneficiam quando utilizam somente OWL.

### 2.2.5 SPARQL

SPARQL é a linguagem padrão da W3C [1] para consulta na Web Semântica. Pode ser utilizada para consultar em fontes de dados distintas, desde que os dados estejam em

[1] http://www.w3.org

formato RDF. Sua sintaxe é bem semelhante à SQL, pois é estruturada em cláusulas: (*SELECT*, *ASK*, *DESCRIBE* ou *CONSTRUCT*), *FROM* (opcional) e *WHERE*, além de outras opcionais como *FILTER* e *ORDER*. Uma consulta SPARQL pode ser convertida em um grafo *G*. A execução da consulta sobre um conjunto de dados RDF, ao qual chamar-se-á de *Gr*, consiste na combinação desses dois grafos. *S* é uma solução para *G* sobre *Gr*, se *G* for um subgrafo de *Gr*, com as devidas substituições das variáveis da consulta pelas declarações do grafo *Gr*.

As consultas com *SELECT* retornam os dados estruturados em tabela com os respectivos atributos que estão especificados na cláusula *SELECT*. No exemplo a seguir, para a fonte de dados listada na Tabela 2.1, a consulta em 2.2 produz o resultado descrito na Tabela 2.2: Na consulta 2.2, são solicitados os valores das variáveis *:nome* e *:mbox* das instâncias presentes na fonte de dados 2.1 e o resultado, por consequência, são as ocorrências listadas na Tabela 2.2.

**Tabela 2.1:** *Fonte de dados*

| Sujeito | Predicado | Objeto |
|---|---|---|
| a | foaf:name | "José da Silva" |
| a | foaf:mbox | <mailto:josesilva@exemplo.com.br> |
| b | foaf:name | "Maria Rocha" |
| b | foaf:mbox | <mailto:mariarocha@exemplo.com.br> |
| c | foaf:mbox | <mailto:pedroalves@exemplo.com.br> |

**Código 2.2** Consulta básica de SPARQL

```
1 PREFIX foaf: <http://xmlns.com/foaf/0.1>
2 SELECT ?name ?mbox
3 WHERE
4         { ?x foaf:name ?name .
5           ?x foaf:mbox ?mbox }
```

**Tabela 2.2:** *Resultado da consulta*

| name | mbox |
|---|---|
| "José da Silva" | <mailto:josesilva@exemplo.com.br> |
| "Maria Rocha" | <mailto:mariarocha@exemplo.com.br> |

A Figura 2.3 ilustra como é realizada a consulta através da combinação entre o grafo *Gr* da fonte de dados e o grafo *G* da consulta SPARQL.

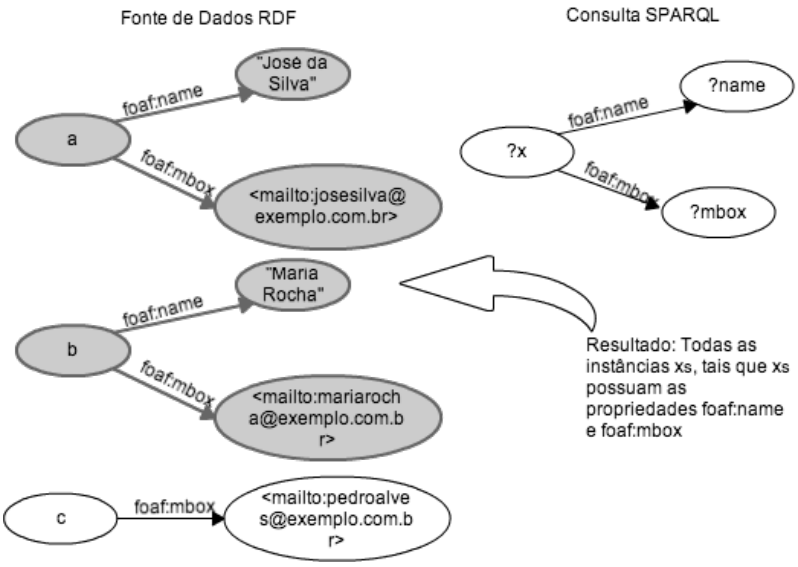


**Figura 2.3:** *Combinação entre grafos na consulta SPARQL*

Outra forma para se criar consultas SPARQL é por meio da cláusula *CONSTRUCT*, que retorna um conjunto de triplas, conforme especificado nessa cláusula da consulta. Essas triplas resultantes podem ser originadas da própria fonte de dados ou serem triplas novas, inferidas a partir de instâncias existentes na origem de dados. O resultado é um novo grafo RDF contendo todas essas triplas.

Como exemplo, considere a consulta descrita no Código 2.3. A realização dessa consulta sobre a fonte de dados apresentada no Código 2.4, no formato Turtle (esse formato será apresentado na Subseção 2.2.6), produz como resultado o grafo RDF ilustrado no Código 2.5::

**Código 2.3** Consulta SPARQL com CONSTRUCT

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>
PREFIX d: <http://learningsparql.com/ns/data#>

CONSTRUCT
{ ?person ?p ?o . }
WHERE {
?person ab:firstName "Craig" ; ab:lastName "Ellis" ;
?p ?o . }
```

**Código 2.4** Fonte de dados

```
@prefix ab: <http://learningsparql.com/ns/addressbook#> .
@prefix d: <http://learningsparql.com/ns/data#> .

d:i0432 ab:firstName            "Richard" .
d:i0432 ab:lastName             "Mutt" .
d:i0432 ab:homeTel              "(229) 276-5135" .
d:i0432 ab:email                "richard49@hotmail.com" .

d:i9771 ab:firstName            "Cindy" .
d:i9771 ab:lastName             "Marshall" .
d:i9771 ab:homeTel              "(245) 646-5488" .
d:i9771 ab:email                "cindym@gmail.com" .

d:i8301 ab:firstName            "Craig" .
d:i8301 ab:lastName             "Ellis" .
d:i8301 ab:email                "craigellis@yahoo.com" .
d:i8301 ab:email                "c.ellis@usairwaysgroup.com" .
```

**Código 2.5** Grafo RDF resultante

```
@prefix d:              <http://learningsparql.com/ns/data#> .
@prefix ab:             <http://learningsparql.com/ns/addressbook#> .

d:i8301
        ab:email                "c.ellis@usairwaysgroup.com" ;
        ab:email                "craigellis@yahoo.com" ;
        ab:firstName            "Craig" ;
        ab:lastName             "Ellis" .
```

### 2.2.6 Turtle

Refere-se a um dos formatos recomendados pela W3C para descrever um documento RDF [9]. É uma serialização intuitiva para o ser humano, com sintaxe bastante amigável. Comparado à XML, é um formato mais conciso, pois não precisa representar um grafo RDF como uma árvore, assim como XML o faz. A descrição das triplas é direta e simples, com *sujeito*, *predicado* e *objeto* em sequência, como no exemplo: *:Pesquisador rdfs:subClassOf :Professor .*, separados apenas por espaço em branco e a declaração é encerrada com um ponto final.

Para várias declarações sobre um mesmo sujeito, Turtle possibilita que os dados

sejam descritos em linhas diferentes, omitindo o sujeito e, com exceção da última declaração, finaliza a linha com ponto e vírgula (;). Para melhor visualização, os predicados e objetos, nesse caso, são identados com um espaçamento à esquerda, conforme exemplo abaixo, em que são descritos os atributos *sobrenome* e *tipo* do indivíduo *Maria*. Esses recursos da linguagem tornam o documento mais sucinto, o que favorece diretamente a legibilidade do mesmo.

*:Maria*
    *:sobrenome "Silva"* ;
    *rdf:type :FuncionariaPublica* .

Os exemplos apresentados nessa seção apontam para a leveza e simplicidade da tecnologia que tornam *Turtle* um dos padrões mais utilizados para a serialização da informação na web semântica.

### 2.2.7 Inferência Ontológica

Nesta subseção, serão apresentados os fundamentos para inferência ontológica e sua contribuição em aplicações que se apoiam na modelagem ontológica.

Como citado anteriormente, o formalismo ontológico possibilita a inferência de fatos novos associados aos indivíduos declarados em uma ontologia. As Subseções 2.2.2 e 2.2.3 descreveram como os domínios ontológicos podem se beneficiar a partir da utilização de RDFS e OWL, respectivamente.

O processo de inferência consiste em gerar novos conhecimentos a partir de fatos associados a axiomas ontológicos. As aplicações que executam esse procedimento são chamadas *Motores de Inferência* ou simplesmente *Reasoners* que podem estar acoplados à base de conhecimento da aplicação ou serem invocados separadamente das aplicações, por questões de desempenho [20]. Para sistemas modelados por ontologias, Bikakis et al. [7] recomendam a utilização de inferência ontológica, pois se integram muito bem ao modelo de domínio, afinal, os motores de inferência utilizarão a própria declaração da ontologia para instanciar fatos novos a partir do contexto modelado. Os exemplos a seguir mostrarão esse procedimento.

Os *frameworks* para *Web* Semântica em geral dão suporte à inferência para as linguagens *RDFS* e *OWL* [20]. Por possuir menos construtores, *RDFS* tem menor capacidade de geração de novos fatos.

Um exemplo de inferência RDFS é apresentado a seguir. Cada propriedade é formada por um conjunto domínio e um conjunto imagem que formalizam quais os tipos de indivíduos que devem compor a relação.

Para o exemplo, considere como conjuntos domínio as classes ontológicas *:Homem* e *:Marido*, como conjuntos imagem, as classes *:Mulher* e *:Esposa* e os seguintes axiomas descritos no formato Turtle:

*:Homem :éAmigo :Mulher .*
*:Marido :éCasado :Esposa .*
*:éCasado rdfs:subPropertyOf :éAmigo .*

Juntamente, considere os seguintes indivíduos:
*:Márcia rdf:type :Esposa .*
*:Roberto :éCasado :Márcia .*

Dos axiomas e indivíduos listados, os seguintes fatos podem ser inferidos de acordo com a semântica RDFS:
*:Roberto rdf:type :Homem .*
*:Roberto rdf:type :Marido .*
*:Roberto :éAmigo :Márcia .*

Para a primeira situação contemplada, como *Roberto* é casado com *Márcia*, e sendo a propriedade *:éCasado* uma subpropriedade de *:éAmigo*, que tem *:Homem* como conjunto domínio, foi inferido que o indíviduo *Roberto* é uma instância da classe ontológica *Homem*.

Para a segunda situação, sendo a classe *:Marido* o conjunto domínio da propriedade *:éCasado*, foi inferido que o indivíduo *Roberto* também é uma instância da classe ontológica *Marido*.

Por fim, como a propriedade *:éCasado* é subpropriedade de *:éAmigo*, então *Roberto* e *Márcia*, além de *casados*, também são *amigos*.

Além da linguagem RDFS, foi apresentada nessa seção a linguagem OWL para formalização de conceitos na web semântica. Dos construtores de OWL, também é possível inferir situações novas a partir de axiomas e indivíduos pré-declarados.

A seguir, são apresentados axiomas que descrevem como conjunto domínio, a classe ontológica *:Filho* e como conjunto imagem a classe *:Mãe*. Foi definida também que a propriedade *:éFilho* é inversa da propriedade *:éMãe*.

*:Filho :éFilho :Mãe .*
*:éFilho owl:inverseOf :éMãe .*

Além do axioma, foi definida explicitamente a situação abaixo:
*:João :éFilho :Leila .*

Por fim, foram inferidos os seguintes fatos conforme a semântica OWL:
*:João rdf:type :Filho .*
*:João rdf:type owl:Thing .*
*:Leila :éMãe :João .*

A primeira inferência refere-se à semântica RDFS que é estendida pela linguagem OWL. Pelo conjunto domínio da propriedade *:éFilho*, deduz-se que o indivíduo *João* é uma instância da classe ontológica *:Filho*.

A segunda inferência oriunda do fato de que, pela especificação OWL, todas as classes são subclasses de *owl:Thing*.

A última situação inferida é resultante do construtor *owl:inverseOf* que inverte os conjuntos domínio e imagem de propriedades que são declaradas como inversas.

### 2.2.8 Inferência por Regras

Como mencionado anteriormente, a utilização de regras amplia a semântica das ontologias. Assim, fatos implícitos são determinados também a partir desse conjunto de regras lógicas. Serão contempladas as regras no formato *Horn* que contêm as cláusulas *antecedente* $\longrightarrow$ *consequente*. Os motores de inferência baseados em regras, na maioria dos casos, as processam pelo método *Forward Chaining* que será detalhado na sequência.

***Forward Chaining***: Nesse método, a cada novo fato explícito adicionado à base de conhecimento, é desencadeado o processo de inferência e novos fatos implícitos são adicionados. A vantagem dessa abordagem é que consultas nessa base terão relativamente boa performance, pois não haverá nenhum processo de inferências, considerando que todos os encadeamentos já foram realizados.

A maior parte dos motores de inferência prefere bases de conhecimento formadas por *forward chaining*, porque são mais simples de serem implementadas e os custos decorrentes das operações de inclusão ou remoção de fatos são aceitáveis, além do que consultas à base de conhecimento são largamente executadas, o que exige que essas operações sejam realizadas com bom desempenho.

A regra abaixo descreve as condições para se inferir a doença Taquicardia [4]. Dentre elas, existem classes ontológicas que fazem parte da semântica da ontologia, como 'pulse rate' e operadores como *nonNegativeInteger* e *equal*. Além desses, as variáveis *rp, pr, t e rv*.

*'pulse rate'(?rp), 'pulse rate measurement datum'(?pr), nonNegativeInteger[> 100](?t), isMeasurementPulseRate(?pr, ?rp), isParameterizedValue(?rp, ?rv), valuePulseRate(?pr, ?t), equal(?rv, false)* $\longrightarrow$ *Tachycardia(?pr), TachycardiaAlarm(?pr)*

A execução dessa regra no método *forward chaining* compreende verificar se todos os fatos ou condições já foram inferidos em um primeiro momento. Para esse caso,

se o indivíduo *rp* for uma instância da classe *pulse rate*, ou *frequência de pulso*, se o indivíduo *pr* pertencer à classe *pulse rate measurement datum*, se a medida *pr* for relativa a uma medida de pulso para o indivíduo *rp*, se *rp* não for um valor parametrizado e se o valor da medida *pr* for *t*, sendo *t* um valor maior do que 100, então é inferido que o indivíduo *pr* é uma instância das classes ontológicas *Tachycardia* e *TachycardiaAlarm*. Esse novo fato, a partir de então, irá compor a base de conhecimento da aplicação.

## 2.3 JSON

Nesta seção será brevemente apresentado o padrão JSON, utilizado para intercâmbio e descrição de dados.

De acordo com a RFC 4627, disponível em http://www.ietf.org/rfc/rfc4627.txt, JSON é um formato leve para intercâmbio e registro de dados, autodescritivo, sendo, portanto, legível para seres humanos, baseado em texto, eficiente para navegação e obtenção de dados e independente de linguagem. Os tipos de dados suportados são: *strings, numerais, booleans, null, objetos e arrays*. Os dados em JSON são organizados em estruturas aninhadas, no formato *chave-valor*, em que chave é uma string, escrita entre aspas, que representa o domínio da informação, e o valor, a descrição da chave associada. A sintaxe de JSON também inclui *vírgulas* para separar os pares *chave-valor*; *dois-pontos*, para separar chave de valor; *chaves*, para inserir objetos e finalmente *colchetes*, para inserir arrays. O exemplo a seguir descreve um documento JSON padrão:

```
{
    "Nome":"Universidade Federal de Goiás",
    "Ano de fundação":1960,
    "Possui Biblioteca":true,
    "Reitoria":{"Reitor":"Prof. Dr. Orlando Afonso Valle do Amaral",
"Vice-Reitor":"Prof. Dr. Manoel Rodrigues Chaves"},
    "Unidades Acadêmicas":[
        {"Nome":"Câmpus Colemar Natal e Silva", "Cidade":"Goiânia"},
        {"Nome":"Câmpus Samambaia", "Cidade":"Goiânia"},
        {"Nome":"Câmpus Aparecida de Goiânia", "Cidade":"Aparecidade de
Goiânia"},
        {"Nome":"Regional Catalão", "Cidade":"Catalão"},
        {"Nome":"Regional Goiás", "Cidade":"Cidade de Goiás"},
        {"Nome":"Regional Jataí", "Cidade":"Jataí"}
    ]
}
```

As características apresentadas acima tornam JSON muito semelhante à XML[2]. Além delas, ambas podem ser encapsuladas em uma requisição HTTP, o que tem tornado JSON um padrão na comunicação de dados na web.

Em compensação, JSON e XML apresentam diferenças, a saber [3]:

- Sintaxe mais simples, pois não precisa de *tags* de fechamento para declarações simples do tipo *chave-valor*
- Um documento JSON é menor do que um documento XML
- A leitura e escrita de um arquivo JSON são mais rápidas
- JSON suporta arrays
- XML precisa de um *parser* XML para ser manipulado, enquanto que JSON precisa apenas de uma função JavaScript padrão

Tanto XML quanto JSON são largamente utilizados para os propósitos citados nessa seção, devendo os desenvolvedores e cientistas optarem pelo que melhor atende às necessidades dos seus projetos.

## 2.4 Padrões de projeto de software

Esta seção expõe duas boas práticas da Engenharia de Software para desenvolvimento de sistemas: aplicação de padrões de projeto e componentização de software.

Mudança é uma característica intrínseca ao desenvolvimento de software e para lidarem com isso, desenvolvedores possuem uma ferramenta poderosa que pode amenizar suas consequências: os Padrões de Projeto. Conforme Shuai et al. [43], um padrão de projeto é uma solução genérica e reutilizável para um problema frequente no projeto de software. Seis dos principais padrões de projeto da Engenharia de Software serão descritos a seguir [13] [53] [1]:

- *Facade*: Padrão estrutural cuja finalidade é prover uma interface de alta granularidade para um conjunto de funcionalidades complexas [13]. Com essa estratégia, os detalhes de acesso a componentes e camadas especialistas para execução de uma atividade crítica, por exemplo, são desacoplados dos clientes requisitantes, ficando apenas a cargo do *Facade*. Esse desacoplamento possibilita que alterações dos componentes que realizam alguma tarefa requisitada pelo *Facade* não interfiram na implementação do cliente

[2] http://www.w3.org/XML/
[3] http://www.json.org/xml.html

- *Factory*: Desacopla do cliente a forma de instanciar um conjunto de classes [13]. O conhecimento do cliente se limita à interface ou classe abstrata da qual esse conjunto de classes implementa ou herda. Em tempo de execução, a partir de parâmetros fornecidos pelo cliente ou informações de configuração do sistema, a classe *Factory* consegue instanciar a classe concreta da necessidade do cliente
- *Strategy*: Refere-se ao encapsulamento de uma coleção de implementações de uma mesma funcionalidade [13] [43]. Com isso, ele oferece uma interface única para clientes acessarem determinada funcionalidade cuja implementação será determinada pelo contexto corrente do cliente que não conhece quem de fato implementará sua requisição. A vantagem é a extensibilidade que provê ao código, pois novas classes que implementam a respectiva funcionalidade podem ser adicionadas de forma que somente o contexto saberá da sua existência
- *Singleton*: Objetiva garantir a existência de somente uma instância de uma classe com único ponto global de acesso [13]. Para isso, o construtor da classe *Singleton* deve ser *privado* e ela própria ou alguma outra classe, como *Factory*, por exemplo, deve manter uma referência *estática* ao objeto da classe *Singleton*. A justificativa para utilização desse padrão é que classes *Singleton* encapsulam atributos e objetos que não devem ser replicados no código [53]
- *Domain Object*: Padrão definido por Alur et al. [1] para separar o processamento das funcionalidades da lógica e estrutura do domínio da aplicação. Esses objetos encapsulam atributos, comportamento, validação e persistência dos dados. Assim, é possível centralizar a lógica e o estado de negócios em uma aplicação, aumentando a capacidade de reutilização dessa lógica entre os casos de uso
- *Transfer Object*: Outro padrão definido por Alur et al. [1] que recomenda a utilização de um mesmo objeto para perpassar ao longo das camadas de software durante a execução de um caso de uso. Para isso, esse objeto deve encapsular os dados que serão acessados pelos métodos ao longo das camadas. Essa medida traz simplicidade ao código e melhora a sua manutenibilidade e extensibilidade, pois os dados de entrada para os métodos, quando alterados ou incluídos novos, não afetam as suas respectivas assinaturas, mas simplesmente a classe que implementa o *Transfer Object*

Complementando as motivações para a aplicação de padrões de projeto, Riva et al. [38] e Rossi et al. [40] fazem análises das soluções para computação sensível a contexto da época e, a partir de aplicação de técnicas de Engenharia Reversa, extraem os padrões de projetos utilizados pelos sistemas e como eles favorecem para o êxito dos seus respectivos projetos, reforçando a relevância do estudo de padrões de projeto em aplicações sensíveis a contexto.

Além da aplicação de padrões de projeto, outra prática que a literatura recomenda para o desenvolvimento de software é a *componentização*. Ela consiste na produção de uma unidade de software que pode ser reutilizada por outras unidades para formar um sistema. Os componentes devem ser desenvolvidos a fim de serem facilmente acoplados ao código-fonte do sistema, oferecendo uma interface bem definida, ou seja, expor classes e métodos de maneira funcional e clara para as demais unidades. Dentre suas vantagens, destacam-se:

1. Possibilitar reutilização em sistemas diferentes ou em locais diversos de um mesmo sistema
2. Abstrair as complexidades da tecnologia componentizada por meio de uma API simples
3. Promover a extensibilidade da arquitetura facilitando a inclusão de novos componentes
4. Promover a flexibilidade para alteração e manutenção da tecnologia componentizada, pois detalhes de acesso a ela estão centralizados no componente
5. Atribuir manutenibilidade e evolução, pois as funcionalidades estão desacopladas entre si
6. Facilitar a integração de informações entre vários canais de acesso

## 2.5 Especificação de *middleware DDS*

Para ambientes pervasivos e distribuídos, em que as aplicações necessitam ser notificadas de situações de contexto de seu interesse, a literatura recomenda a utilização de *middlewares* de comunicação que ofereçam suporte ao paradigma *publish / subscribe* [34]. Por atender a esse requisito, será detalhada a especificação DDS (*Data Distribution Service*) da OMG. DDS é o primeiro padrão internacional aberto de *middleware* direcionado para comunicação, segundo o paradigma *publish/subscribe* para sistemas embarcados.

Dentre as características desse padrão destacam-se o paradigma *publish/subscribe*, a forte tipagem de dados e a localização dinâmica de publicadores, assinantes e tópicos declarados.

Com DDS, as aplicações publicadoras e assinantes de um determinado tópico podem especificar os tipos de dados que utilizarão para comunicação. Por esse motivo, é considerada uma tecnologia com forte tipagem de dados. Esses tipos de dados são declarados em um formato específico denominado *DDL - Data Definition Language*[4], o qual é transmitido entre as aplicações dentro da rede DDS.

[4] http://www.omg.org/spec/

Em um domínio *DDS*, cada tópico está associado a um *DDL*, porém um mesmo *DDL* pode estar associado a vários tópicos. Para que as aplicações publicadoras e assinantes de um tópico tenham conhecimento dos tipos de dados encapsulados no respectivo *DDL*, a especificação DDS possibilita que as assinaturas dos métodos de suas classes e interfaces relativas ao *DDL* em questão já contenham em suas assinaturas os próprios tipos de dados definidos naquele *DDL*. Isso é possível porque as implementações da especificação DDS fornecem uma ferramenta para a geração de classes de acesso ao *middleware* específicas para algum *DDL* declarado, conforme a linguagem de programação da aplicação. Essa forte tipagem de dados inibe a ocorrência de erros em tempo de execução, pois ao invés de interagirem com o formato *DDL*, as aplicações publicadoras e assinantes interagem com o *middleware* através de métodos que contêm assinaturas com tipos de dados básicos e previamente conhecidos, como *string*, *long*, *float* e *array*.

A Figura 2.4 esboça os principais elementos da arquitetura DDS[5]:

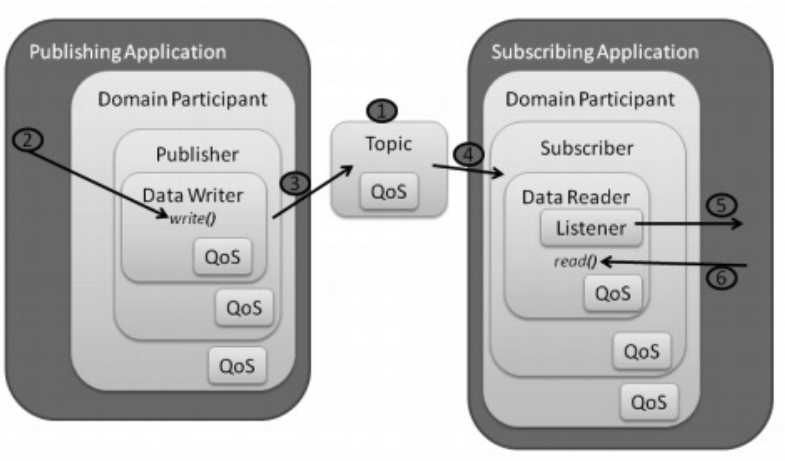


**Figura 2.4:** *Arquitetura DDS*

- *Publishing Application*: aplicações que publicam dados em tópico(s)
- *Subscribing Application*: aplicações que assinam tópico(s) dos quais desejam ser notificadas
- *Domain Participant*: representa a participação de uma aplicação numa rede virtual, conectando todas as aplicações que compartilham um mesmo identificador de domínio. Dessa forma, várias redes DDS podem coexistir numa mesma rede física, sem interferirem umas nas outras
- *Publisher*: elemento responsável pela publicação dos dados. Pode estar associado a diferentes tipos de dados (*DDLs*), sendo que cada um possui um *DataWriter* específico. Em virtude disso, trata-se de um *container* de *DataWriters* (explicado

[5] http://www.coredx.com/documents/CoreDX_DDS_Programmers_Guide_v3.6.pdf

adiante). Tanto o *Publisher* quanto o *DataWriter* possuem políticas de QoS para definir seus comportamentos quanto à publicação de dados

- *DataWriter*: objeto que efetua a escrita nos tópicos. Cada *DataWriter* está associado a somente um tópico, que por sua vez também possui um tipo de dado específico, através do qual o *DataWriter* realiza sua escrita
- *QoS*: corresponde às políticas de QoS estabelecidas pelo DDS. Os objetos identificados na figura para os quais estão associados o elemento QoS têm suas atividades influenciadas por essas políticas. De forma resumida, a comunicação entre uma aplicação publicadora e uma aplicação assinante através de um tópico só será bem sucedida se possuirem QoS compatíveis, ou seja, se as políticas dos assinantes forem iguais ou menos exigentes às políticas dos publicadores e do respectivo tópico assinado
- *Topic*: elemento central na rede DDS por meio do qual publicadores e assinantes se comunicam. É identificado por um nome, tipo de dado e políticas de QoS
- *Subscriber*: análogo ao *Publisher*, porém referente à aplicação assinante. É responsável por receber os dados publicados e disponibilizá-los para a aplicação, através do objeto *DataReader* específico do tópico assinado. É, portanto, um *container* de *DataReaders*. Ambos estão associados a políticas de QoS que determinam as suas formas de atuação
- *DataReader*: análogo ao *DataWriter*, porém referente à aplicação publicadora. Está associado a um tópico e tipo de dado (*DDL*) somente, por meio do qual faz a leitura dos dados comunicados
- *Listener*: elemento associado a um *DataReader*, cuja finalidade é realizar o recebimento assíncrono dos dados publicados

Resumidamente, as ações enumeradas na Figura 2.4 podem assim ser descritas:

1. *DataReaders* e *DataWriters* são associados a um *Topic*
2. A aplicação publicadora invoca o método *DataWriter::write()* para escrita
3. O respectivo *Publisher* publica o dado
4. O respectivo *Subscriber* recebe o dado
5. O *Listener* notifica a aplicação assinante sobre o recebimento de dados novos
6. A aplicação assinante invoca o método *DataReader::read()* para acessar os dados

## 2.6 Considerações Finais

Esse capítulo apresentou as bases teóricas para a implementação da proposta deste trabalho: um interpretador semântico de informações de contexto apoiado na filtragem semântica de eventos.

Para a concepção arquitetural, foi apresentada a arquitetura *Context Toolkit* que integra componentes com funções bem definidas para prover serviços de contexto às aplicações. Este trabalho também contempla um *arcabouço conceitual* que fornece subsídios para que desenvolvedores possam implementar aplicações sensíveis a contexto de forma ágil e com escopo limitado às funcionalidades do domínio de sua atuação, pois os serviços básicos de contexto são gerenciados e fornecidos pela infraestrutura.

A Seção 2.2 abordou os conceitos da Web Semântica e suas vantagens comparadas à web tradicional, como a inclusão de significado nos termos compartilhados na web. Após essa introdução, definiu-se ontologia e foram expostos os motivos para esse paradigma ser o adequado para representação da informação na Web Semântica e na modelagem do contexto em sistemas sensíveis a contexto. Os motivos elencados foram: alta capacidade de inferência, desacoplamento entre regras de negócio do contexto e aplicações, suporte à validação e interoperabilidade semântica entre sistemas.

Na continuidade da seção, foram apresentados alguns padrões e especificações para manipulação de dados na web semântica, tais como RDF, RDFS, OWL, SWRL, SPARQL e Turtle. Do padrão SPARQL, foi destacada a forma de consulta CONSTRUCT, que pela sua capacidade de gerar modelos RDF a partir das consultas, foi a escolhida para ser utilizada neste trabalho. O outro formato mencionado nessa seção foi a serialização Turtle para modelos de dados RDF. Devido às características de simplicidade e tamanho dos arquivos gerados, escolheu-se Turtle para serializar os documentos RDF do trabalho. Para concluir a seção, foram descritas duas técnicas de inferência: baseada em ontologia e baseada em regras.

Na sequência, na Seção 2.3, foi abordado o formato JSON, utilizado para intercâmbio e descrição de dados. Pelas vantagens descritas, essa tecnologia foi escolhida para representação dos dados de configuração e dos templates de filtros semânticos na infraestrutura Hermes.

A Seção 2.4 apresenta duas boas práticas recomendadas pela Engenharia de Software: o uso de padrões de projeto e a componentização de software. Para isso, foram contemplados alguns *patterns* consolidados da literatura com suas respectivas vantagens. Os seguintes padrões foram brevemente detalhados: *facade*, *factory*, *strategy*, *singleton*, *domain object* e *transfer object*. As principais razões pela escolha desses padrões foram a manutenibilidade e extensibilidade que eles conferem ao código das aplicações, estruturando as camadas de forma desacoplada, porém com funcionalidades bem definidas.

Para finalizar o capítulo, foi apresentado o *middleware* DDS da OMG, e como suas características de possibilitar comunicação distribuída, transparente e mediante o paradigma *publish / subscribe* pode contribuir para a arquitetura de sistemas sensíveis a contexto.

Diante do exposto nesse capítulo, a Tabela 2.3 relaciona as características de projeto apresentadas no Capítulo 1 aos fundamentos teóricos.

**Tabela 2.3:** *Relação entre princípios de projeto e fundamentos*

| **Princípio de Projeto** | **Fundamentos Teóricos** |
|---|---|
| Arquitetura em camadas com componentes reutilizáveis | *Context Toolkit*, padrões de projeto e componentização |
| Extensibilidade | Modelagem ontológica, JSON, padrões de projeto e componentização |
| Modelo de contexto independente | Modelagem ontológica |
| Suporte a múltiplas técnicas de interpretação | Inferências ontológica e por regras |
| Monitoramento e Detecção de Eventos | *Context Toolkit*, modelagem ontológica, DDS |
| Verificação de consistência entre fatos e modelo de contexto | Modelagem ontológica |
| Filtragem de eventos | Modelagem ontológica, JSON |

CAPÍTULO 3

# Infraestrutura Hermes

Após levantamento sobre o estado da arte de Computação Sensível a Contexto, foi iniciada a etapa de análise de requisitos a fim de implementar uma infraestrutura que contivesse as seguintes características:

- Serviços de apoio ao gerenciamento do ciclo de vida das informações de contexto (aquisição, modelagem, interpretação e disseminação) [10] [34]
- Componentização da infraestrutura [34]
- Fornecimento de serviços apoiados na semântica das informações de contexto, como modelagem, inferência e filtragem [34] [6]

A próxima etapa consistiu em definir os fundamentos arquiteturais da infraestrutura. Dentre os modelos consolidados da literatura para suporte a aplicações sensíveis a contexto, escolheu-se como inspiração o arcabouço conceitual de *Context Toolkit*. Conforme apresentado na Seção 2.1, o *Context Toolkit* constitui uma referência sólida para desenvolvedores de infraestruturas sensíveis a contexto. A sua organização interna foi motivadora para que fosse escolhido para apoiar o desenvolvimento da infraestrutura proposta, dentre os quais destacam-se: o desacoplamento dos serviços básicos para gerenciamento do ciclo de vida das informações de contexto, a especificação de componentes para cada etapa desse ciclo e sua simplicidade e capacidade de reúso.

## 3.1 Arquitetura do *Hermes*

A infraestrutura foi então projetada e denominada *Hermes*[1]. Os seus componentes internos são análogos aos propostos por Dey et al. [10], o que reforça que este trabalho não objetiva refazer o *Context Toolkit*, mas apenas acrescentar serviços para manipular a semântica das informações de contexto. A Figura 3.1 retrata uma visão geral da infraestrutura projetada com seus principais componentes [47].

[1] A nomenclatura é proveniente da mitologia grega, em que o deus *Hermes* era o mensageiro dos deuses e que sempre apelava para a inteligência antes de realizar suas ações - ref. Wikipedia

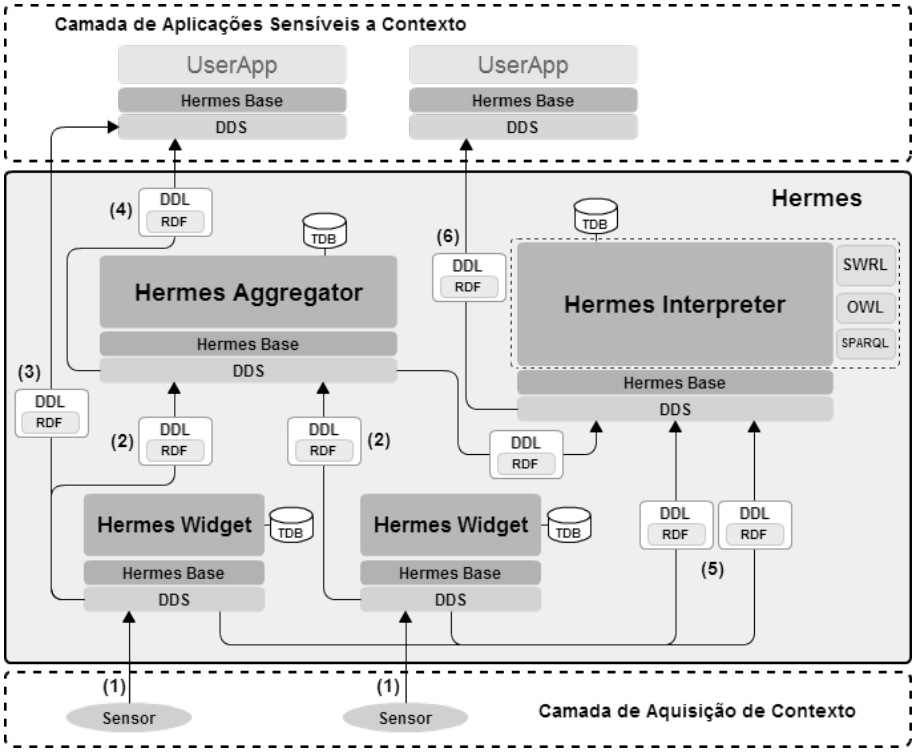


**Figura 3.1:** *Visão geral da arquitetura de Hermes [27]*

*Hermes Base*: Componente que encapsula os detalhes de acesso ao *middleware* DDS, fornecendo uma API para aplicações e componentes de *Hermes* se comunicarem. Dentro do gerenciamento do ciclo de vida de contexto, juntamente com o *middleware* DDS são responsáveis pelas etapas de aquisição e distribuição do contexto por meio do paradigma *publish/subscribe*. Essa abstração permite o desenvolvimento mais simplificado de aplicações publicadoras e assinantes como também aumenta a manutenibilidade do sistema quanto a possíveis alterações no *middleware* DDS. A próxima seção deste capítulo discorrerá com mais profundidade sobre o seu funcionamento

*Hermes Widget*: Componente que abstrai das aplicações e do restante da infraestrutura o serviço de aquisição de informações de contexto. As fontes desses dados podem ser sensores físicos, lógicos ou virtuais. Sensores físicos são aqueles que coletam dados fisicamente observáveis como sinais vitais, temperatura ambiente e luz. Sensores lógicos são aqueles que fornecem informações que não são provenientes de um só sensor, sendo necessária a fusão de vários deles, como por exemplo: informação de clima, reconhecimento de atividades e identificação de localização. Os sensores virtuais coletam informações que não podem ser medidas fisicamente, tais como contatos telefônicos, email, preferências de usuários e informações de dispositivos computacionais

Adquiridos os dados, *Hermes Widget* os traduz para um formato mais significativo,

para o caso de sensores físicos, modela-os na ontologia corrente do contexto e solicita suas publicações ao *Hermes Base*. O componente *Hermes Widget* não será melhor detalhado por ser escopo de outro trabalho de pesquisa [47]

*Hermes Aggregator*: Encapsula as informações de vários *Hermes Widgets* com a finalidade de fornecer contexto de mais alto nível e centralizar o acesso a determinadas entidades relevantes do modelo, por exemplo: em um cenário de monitoramento de sinais vitais humanos, vários *Hermes Widgets* coletariam os dados de determinado paciente. Seria recomendado que houvesse um componente *Hermes Aggregator* específico para a entidade *paciente*, que agregaria as informações provenientes de todos os *Hermes Widgets* daquele paciente. Assemelha-se a um sensor virtual, explicado no item anterior, todavia com a diferença de fornecer dados de mais alto nível para o contexto monitorado. Esse componente não será melhor detalhado por ser escopo de outro trabalho de pesquisa [47]

*Hermes Interpreter*: A partir de notificações do contexto, interpreta as informações conforme um ou mais mecanismos de inferência e produz informações de mais alto nível do que as coletadas pelos *Hermes Widgets* e *Hermes Aggregators*, ou seja, informações novas resultantes da inferência baseada na ontologia do contexto ou eventos relevantes identificados por meio de regras pré-configuradas. Essas novas informações irão compor a base de conhecimento da infraestrutura e serão filtradas pelo componente para que sejam identificados eventos de interesse das aplicações assinantes, que foram previamente especificados por elas. Se verificados tais eventos, *Hermes Interpreter* notificará as respectivas aplicações por meio do componente *Hermes Base*.

As tarefas de intepretação e filtragem são desempenhadas com o objetivo de extrair a semântica contida nos modelos de contexto transmitidos, utilizando para isso padrões e linguagens apropriados. O componente foi concebido de forma bastante desacoplada de modelo ontológico, o que o capacita a manipular diferentes ontologias simultaneamente e ser manutenível quanto às alterações nas ontologias acessadas. Demais características e funcionamento desses componentes serão contemplados no Capítulo 4

Outros elementos que compõem a arquitetura, visíveis na Figura 3.1, são os formatos de dados *RDF* e *DDL*. Como mencionado na seção 2.5, o tipo de dado trafegado na rede *DDS* é o *DDL*, que encapsula as estruturas de dados que armazenam as informações publicadas nos tópicos. Como a maior parte dos tópicos de *Hermes* são referentes a notificação de contexto modelado semanticamente, a Figura 3.1 ilustra que as informações de contexto são trafegadas no formato de dados padrão *RDF*.

Em função da infraestrutura *Hermes* ser objeto de validação em cenários de monitoramento de sinais vitais humanos, os exemplos de funcionamento dos componentes

contemplarão este estudo de caso. A Figura 3.2 contém uma visão geral do fluxo básico da infraestrutura *Hermes* nesse ambiente, cujos passos podem ser resumidamente[2] descritos da seguinte maneira:

1. *Hermes Widget* publica, por meio do componente *Hermes Base*, uma informação de contexto na rede DDS;
2. O componente assinante *Hermes Interpreter* é notificado dessa alteração do contexto
3. *Hermes Interpreter* interpreta a semântica das informações de contexto e produz novos fatos e situações de alto nível de contexto
4. *Hermes Intepreter* acessa os filtros referentes ao tópico de contexto em questão
5. *Hermes Intepreter* filtra essas novas informações geradas conforme os filtros previamente solicitados pelas aplicações assinantes
6. *Hermes Interpreter* publica se algum evento de contexto de interesse tiver sido identificado
7. As aplicações assinantes são notificadas da ocorrência dos eventos

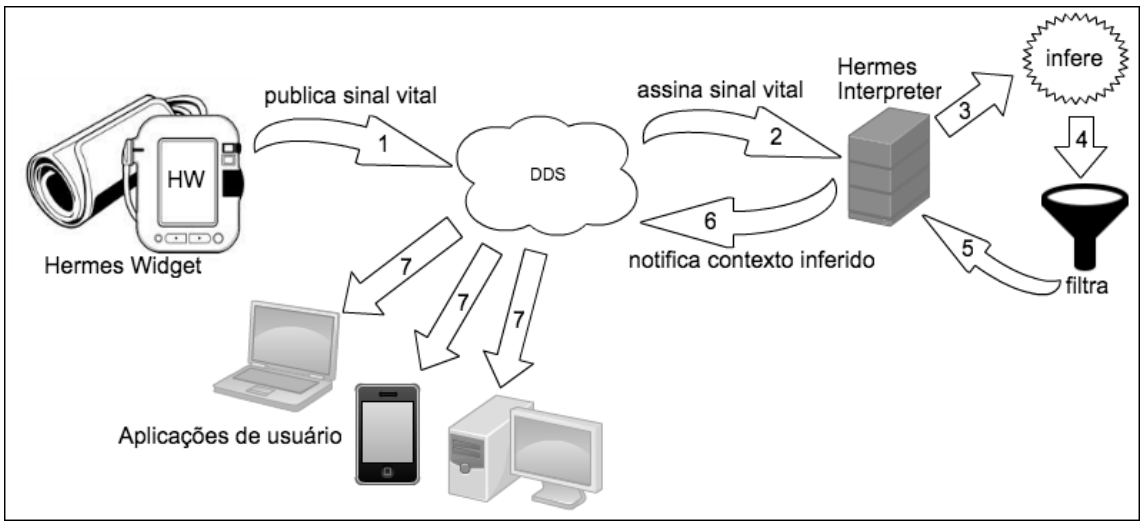


**Figura 3.2:** *Visão geral do funcionamento da infraestrutura Hermes*

## 3.2 Hermes Base

Elemento central na distribuição e aquisição de contexto, o componente *Hermes Base* fornece uma interface de comunicação para os componentes supracitados e para as aplicações sensíveis a contexto acessarem o *middleware* de comunicação DDS. Para isso,

[2]Esses passos serão melhor detalhados no Capítulo 4.

*Hermes Base* centraliza as complexidades de acesso ao DDS que, na verdade, é um fornecedor de *middleware* da especificação DDS e expõe classes e métodos simplificados independentes desse fornecedor para o restante da infraestrutura e aplicações, facilitando assim o desenvolvimento de aplicações e outros componentes, além de tornar a infraestrutura manutenível quanto ao *middleware* adotado. Pela Figura 3.3, é possível verificar quais são essas interfaces que a API do *Hermes Base* expõe:

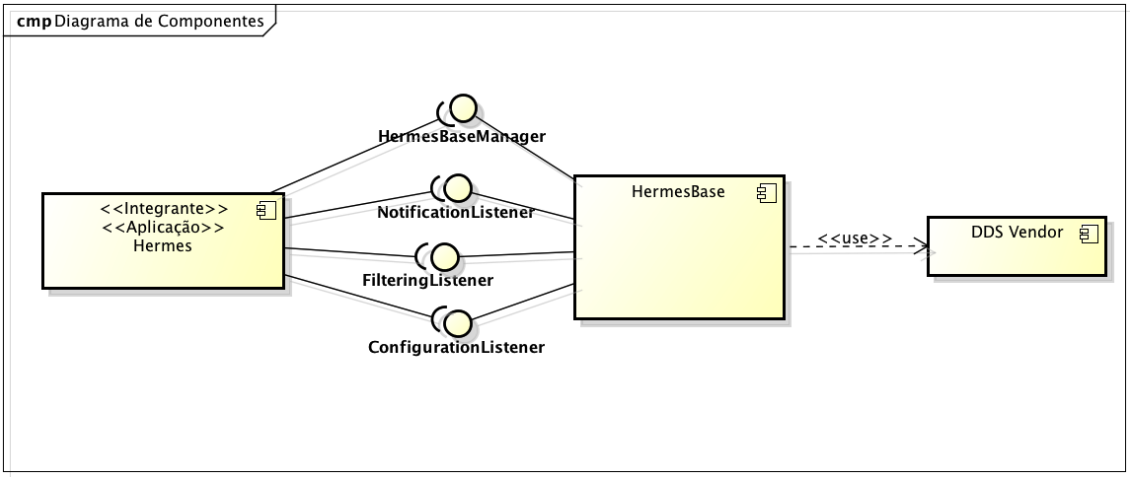


**Figura 3.3:** *Diagrama de Componentes do Hermes Base*

Como explicado na Seção 2.5, publicadores e assinantes de um conjunto de tópicos compartilham um mesmo *DDL*. Cada *listener* está associado a um *DataReaderListener* que, consequentemente, está associado a um DDL específico. O elemento central do *Hermes Base* é a interface *Hermes Base Manager*, que expõe métodos para registro, publicação e assinatura de tópicos. Eles são:

*createConfigurationTopic()*: aplicações e componentes podem solicitar a criação dos *tópicos de configuração* que desejarem acessar para publicação ou assinatura. Até o momento só existe um tipo de configuração que possibilita que publicadores configurem as técnicas de inferência por tópicos. Para acessá-lo, deve-se registrar o tópico *config*, com complemento (política QoS *partition*) com valor *tipo_inferencia*. A assinatura do método exige apenas o *nome do tópico* e o objeto *tipo dado configuracao* é o tipo DDL relativo ao tópico criado. O Diagrama de Sequência[3] na Figura 3.4 ilustra seu funcionamento.

*createFilteringTopic()*: uma das responsabilidades do componente *Hermes Interpreter* é efetuar a filtragem semântica dos dados de contexto, assunto que será abordado no Capítulo 4. Para desempenhar essa funcionalidade, esse componente deve primeiramente solicitar a criação do tópico de filtragem *createFilter*. Por esse tópico, *Hermes*

[3]Para simplificação dos diagramas, foram descritos somente os passos principais, alguns nomes de classes estão abreviados e as assinaturas dos métodos contêm, apenas, os parâmetros mais relevantes para compreensão do exemplo.

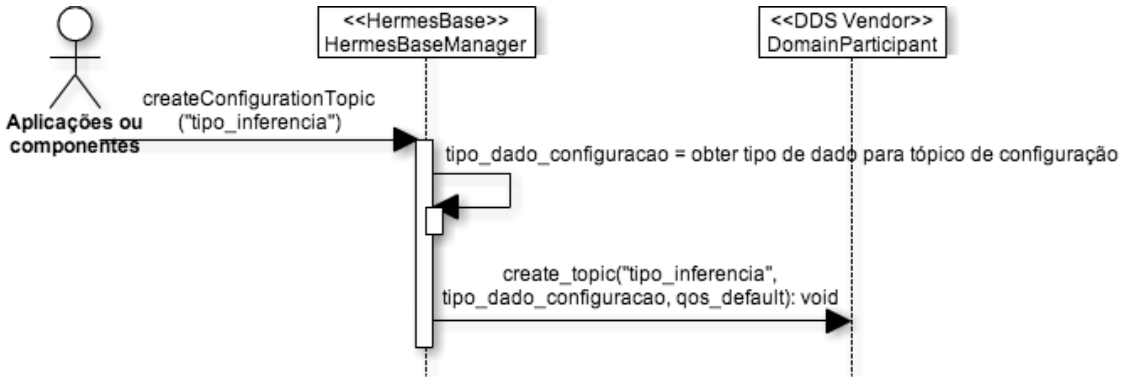


**Figura 3.4:** *Criação de tópico de configuração*

*Base* notifica *Hermes Interpreter* para criação de filtro para algum assinante para um tópico específico. A assinatura do método exige apenas o *nome do tópico*. O objeto *tipo dado filtro* é o tipo DDL relativo ao tópico criado. O diagrama de sequência na Figura 3.5 ilustra seu funcionamento.

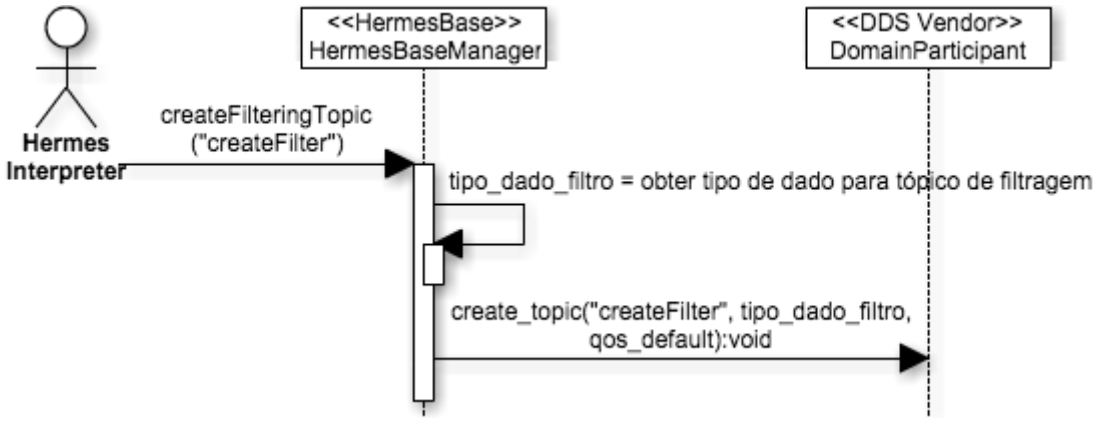


**Figura 3.5:** *Criação de tópico de filtragem*

*createNotificationTopic()*: por esse método, aplicações e componentes solicitam a criação dos *tópicos de notificação de contexto*. Para padronizar a nomenclatura, esses tópicos são nomeados conforme as classes ontológicas a que estão associados. Assim como nos métodos de criação de tópicos anteriores, a assinatura do método contém apenas o "nome do tópico". O objeto *tipo dado notificacao* é o tipo DDL relativo ao tópico criado. O diagrama de sequência na Figura 3.6 ilustra seu funcionamento que contém como um dos argumentos do método *createNotificationTopic*, o tópico *bloodPressure*, ou *pressão sanguínea*, relativo ao cenário de monitoramento de sinais vitais humanos [4] que será o estudo de caso realizado neste trabalho.

*subscribeConfigurationTopic()*: por esse método, aplicações e componentes solicitam a *assinatura de tópicos de configuração*. A assinatura do método exige os dados *nome do tópico*, definido por *config*, o *complemento do tópico* que se refere ao nome do item de configuração que se deseja parametrizar e por último, o objeto *listener de confi-*

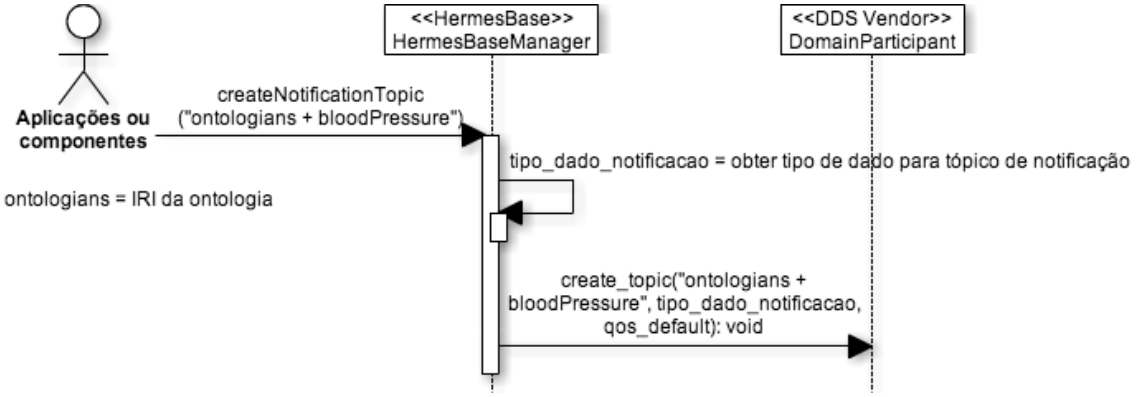


**Figura 3.6:** *Criação de tópico de notificação*

*guração do assinante* o qual será notificado por *Hermes Base* quando houver publicação no referido tópico. A infraestrutura suporta a configuração de vários parâmetros para seu funcionamento que serão definidos no Capítulo 4, para o contexto do componente *Hermes Interpreter*. Porém, o único deles que é configurável por meio de publicação e assinatura de tópicos é o parâmetro *tipo de inferência*.

*subscribeFilteringTopic()*: por esse método, os componentes e aplicações solicitam a assinatura do tópico de filtragem, chamado *createFilter*. A assinatura do método contém os argumentos *nome do tópico*, *complemento do tópico* e o objeto *listener de filtragem do assinante*.

*subscribeNotificationTopic()*: por esse método, componentes e aplicações solicitam a *assinatura de tópicos de notificação de contexto*. A assinatura do método contém os argumentos *nome do tópico*, *complemento do tópico*, o objeto *listener de notificação* do assinante e *filtro*, de acordo com a necessidade do assinante.

*publishConfiguration()*: por esse método, aplicações e componentes publicam no tópico de *configuração*. Como até o momento só é suportada uma opção de configuração, alteração de técnica de inferência para tópicos, a assinatura do método contém os seguintes parâmetros: *tipo de inferência* e *lista de tópicos*.

*publishNotification()*: por esse método, as aplicações e componentes publicam no tópico de notificação de contexto. A assinatura do método exige os dados *id da entidade notificada*, *nome do tópico*, *complemento do tópico*, *localização da ontologia*, *rdf do contexto serializado* e *tipo de serialização do rdf*.

Além desses métodos oferecidos por *HermesBaseManager*, a API ainda possui as seguintes interfaces: *NotificationListener, FilteringListener e ConfigurationListener*: Interfaces que têm a responsabilidade de aquisição de contexto dentro da infraestrutura *Hermes*. As três interfaces, portanto, são expostas às aplicações e componentes para que esses possam ser notificados sobre as publicações nos tópicos assinados.

Para permitir essa comunicação, cada interface de *listener* expõe o método *handleContext()* que deve ser implementado pelo componente ou aplicação, através de

suas próprias classes *Listeners* que implementam uma dessas três interfaces, conforme o tipo de tópico ao qual desejam assinar.

Cada método, por sua vez, possui assinaturas específicas de cada tipo de tópico, ou seja, idênticas aos atributos encapsulados pelo respectivo DDL associado ao tópico, como por exemplo, na interface *NotificationListener*, cujo método *handleContext()* contém os atributos transmitidos pelo *DDL* de *publicação de notificação*, a saber:

- *id_entidade*: elemento principal no contexto notificado que objetiva identificar a instância do objeto DDL publicado. Para o cenário de monitoramento de sinais vitais, é a concatenação da identificação do paciente juntamente com seu sinal vital, como por exemplo: "135679_frequenciaPulso"
- *nome_topico*: nome do tópico em que contexto foi publicado. Informação útil para os assinantes, considerando que um mesmo *listener* pode ser atribuído para vários tópicos. No cenário de exemplo, esse parâmetro pode conter o tópico "Frequência de Pulso"
- *complemento_topico*: Esse parâmetro é utilizado para publicar os eventos de contexto filtrados. Por meio dele, o publicador informa o código do assinante na infraestrutura *Hermes*, o que permite que somente o respectivo assinante seja notificado
- *caminho_ontologia*: Localização do *schema* ontológico que descreve a semântica do contexto publicado
- *rdfSerializado*: Informação mais relevante dentre os argumentos desse método, pois se refere ao modelo RDF do contexto publicado
- *tipo_serializacao*: Tipo de serialização do modelo RDF, como por exemplo: Turtle

O método *handleContext()* do listener *FilteringListener* possui os seguintes parâmetros, associados ao objeto DDL do tópico *filtro*:

- *idListener*: O argumento *idListener* se refere ao identificador do assinante na infraestrutura
- *nomeTopico*: O nome do tópico para o qual o filtro é destinado. No cenário de monitoramento de sinais vitais, um exemplo desse parâmetro é o tópico "Saturação de Oxigênio"
- *filterHash*: Filtro solicitado pela aplicação assinante que será notificado ao componente criador do filtro, como por exemplo: ("id_paciente = "134299"e anormalidade de saturação = "hipoxemia")

Por último, o método *handleContext()* do listener *ConfigurationListener*, que contém em sua assinatura, os atributos do objeto DDL de configuração, a saber:

- *tipo_inferencia*: Tipo de inferência que se deseja ser aplicada, como por exemplo: "inferência ontológica"

- *topicos*: Lista dos tópicos que serão reconfigurados para o novo tipo de inferência citado, como por exemplo: "pressão sanguínea e temperatura corpórea"

O Apêndice A contém os diagramas de sequência relativos aos cenários apresentados de cada método da interface *HermesBaseManager*, o diagrama de classe dos *listeners* e seus relacionamentos com a API de *Hermes Base* e também os diagramas de sequência referentes à notificação dos assinantes dos tópicos de *Configuração*, *Filtragem* e *Notificação*.

## 3.3 Considerações Finais

Esse capítulo apresentou a infraestrutura *Hermes*, juntamente com os detalhes de projeto que seguem a proposta de Dey et al. [10], com a incorporação de serviços semânticos e como sua organização atende aos princípios identificados por Perera et al. [34] e Bettini et al. [6]. A relação entre esses princípios e como eles são atendidos por *Hermes* está resumida na Tabela 3.1.

**Tabela 3.1:** *Relação entre princípios de projeto e Hermes*

| Princípio de Projeto | Hermes |
|---|---|
| Arquitetura em camadas com componentes reutilizáveis | Infraestrutura *Hermes* |
| Extensibilidade | *Hermes Interpreter*, quanto aos mecanismos de inferência e filtragem de contexto |
| Modelo de contexto independente | Toda infraestrutura *Hermes*, que é executada de forma desacoplada de modelo de contexto. |
| Suporte a múltiplas técnicas de interpretação | *Hermes Interpreter* |
| Monitoramento e Detecção de Eventos | *Hermes Base* e *Hermes Interpreter* |
| Verificação de consistência entre fatos e modelo de contexto | Todos os componentes efetuam tal verificação ao manipularem os modelos ontológicos de contexto |

Dentre os componentes de *Hermes*, esse capítulo destacou o componente *Hermes Base* que oferece uma API para os demais componentes e aplicações se comunicarem com a infraestrutura. Pelas interfaces expostas e seus respectivos métodos, os sistemas que interagem com *Hermes Base* não necessitam conhecer nenhuma especificidade do *middleware* DDS, pois os tipos de dados exigidos são apenas dados primitivos e coleções de *strings*, além do que favorece a manutenibilidade de *Hermes* quanto a modificações na solução de *middleware* DDS escolhida.

CAPÍTULO 4

# Hermes Interpreter

Este capítulo descreve o desenvolvimento do componente de interpretação de contexto da infraestrutura *Hermes*, denominado *Hermes Interpreter*[1]. Serão abordados os seus principais requisitos, sua habilidade de filtragem semântica de contexto, sua anatomia e principais camadas e, por fim, uma avaliação da sua capacidade de manutenção.

## 4.1 Requisitos

Por ser um componente da infraestrutura *Hermes*, *Hermes Interpreter* deve atender aos requisitos gerais que foram concebidos para essa infraestrutura, os quais foram expostos no Capítulo 3 e resumidos na Tabela 3.1.

Juntamente com esses requisitos, a *interpretação de contexto* se destaca como o requisito funcional mais importante do componente *Hermes Interpreter*. A importância desse requisito é comprovada pela dedicação de uma etapa exclusiva dentro do ciclo de vida de contexto, cuja finalidade é produzir informações de contexto novas e de relevância para as aplicações assinantes a partir de dados adquiridos pelos provedores de contexto [31].

A esses requisitos, foram acrescentadas outras necessidades identificadas na literatura, em trabalhos como de Bikakis et al. [7], Bettini et al. [6] e Perera et al. [34], tais como o requisito funcional *filtragem de contexto* e os requisitos não funcionais *manutenibilidade e flexibilidade*.

A *filtragem de contexto* complementa o requisito de *monitoramento e detecção de eventos*, pois é por meio desse requisito que o componente *Hermes Interpreter* poderá notificar as aplicações assinantes de *Hermes* com a precisão necessária para atender as suas respectivas necessidades, dentre os diversos dados de contexto publicados.

Os outros requisitos são os não funcionais *manutenibilidade* e *flexibilidade* que serão explicados na sequência de acordo com a norma internacional SEVOCAB[2] que

[1] Ao longo deste capítulo também será referenciado como HI.
[2] http://pascal.computer.org/sev$_$display/

define vocabulário consensual para a área de Engenharia de Software. O requisito não funcional *extensibilidade*, um dos requisitos apresentados na Tabela 3.1, também será descrito conforme essa norma.

*Manutenibilidade*: Facilidade com que um software ou componente pode ser modificado para alterar ou adicionar funcionalidades, corrigir defeitos, melhorar desempenho ou outras características. Conforme os requisitos da Tabela 3.1, HI deve ser *independente de modelo de contexto*. Para isso, ele deve ser manutenível quanto às alterações desse modelo, de forma que elas ocorram com a menor interferência possível no código e no funcionamento do componente.

Outro ponto propício para mudanças é na lógica de processamento da interpretação de contexto, quando novos mecanismos forem sendo adicionados ao componente, consequência da inclusão de novos requisitos como implantação de novos sensores e sensibilidade a novas situações de contexto.

*Flexibilidade*: Capacidade que um sistema ou componente tem para ser modificado para uso em aplicações ou ambientes diferentes para os quais eles foram incialmente projetados.

As soluções sensíveis a contexto devem ser flexíveis para suportarem diversos mecanismos de inferência, diferentes inclusive dos quais inicialmente foram especificados. E ainda, possibilitar que as técnicas apropriadas sejam aplicadas para as situações corretas [6]. Em um ambiente pervasivo, em que diversos sensores monitoram o contexto, produzindo notificações a todo instante, é uma atividade complexa prever a técnica de inferência adequada para alguma situação específica em tempo de desenvolvimento. Para solucionar essa complexidade, recomenda-se que as aplicações sejam configuráveis quanto à(s) técnica(s) de inferência(s) por situação de contexto ou que possuam mecanismos para que se adaptem dinamicamente aos variados contextos possíveis [34].

*Extensibilidade*: Capacidade que o software tem para incorporar novas funcionalidades. Novamente, considerando a *independência de modelo de contexto* a qual o HI deve suportar, ele deve ser extensível para que novas ontologias de contexto sejam a ele acrescentadas e ainda tenha capacidade de executar todos os modelos ontológicos simultaneamente.

Outro motivo para que HI seja extensível é decorrente das alterações no modelo ontológico de contexto, como a inclusão de novas entidades ou atributos e do acréscimo de novas ontologias na infraestrutura que demandam a incorporação de novos parâmetros para os filtros de contexto disponibilizados pelo componente.

Por último, HI deve ser extensível para suportar novos mecanismos de interpretação de contexto, conforme a literatura recomenda em Bettini et al. [6] e Perera el at. [34].

Os requisitos apresentados nessa seção, os quais compreendem os apontados na Tabela 3.1 e os novos que foram introduzidos neste capítulo são enumerados na seguinte listagem:

1. Arquitetura reutilizável
2. Independência de modelo de contexto
3. Suporte a múltiplas técnicas de interpretação de contexto
4. Extensibilidade
5. Monitoramento e detecção de eventos
6. Verificação de consistência entre fatos e modelo de contexto
7. Filtragem semântica de contexto
8. Manutenibilidade
9. Flexibilidade

## 4.2 Filtros semânticos

Juntamente com a capacidade de interpretar informações de contexto, o componente *Hermes Interpreter* possibilita que as aplicações sensíveis a contexto solicitem à infraestrutura *Hermes*, as situações de contexto ou eventos específicos das quais elas desejam ser notificadas. Para isso, *Hermes Interpreter* implementa a funcionalidade de *filtragem semântica de eventos*.

Conforme Perera et al. [34], a filtragem de contexto pode ocorrer em diferentes cenários: fontes de contexto, dados ou eventos. No primeiro cenário, a camada de aquisição seleciona o melhor provedor de dados para o contexto do usuário, cenário esse relevante considerando que a alta escalabilidade de sensores implantados em determinados ambientes exige que quanto melhor os sistemas conseguirem selecionar apenas as fontes de contexto adequadas para determinada situação, mais eficiente será o processamento das etapas subsequentes do ciclo de vida do contexto.

Na filtragem de dados adquiridos, o sistema realiza tratamento das informações devido à leitura de dados conflitantes, imprecisos ou com erros durante a etapa de aquisição, não permitindo que informações nessas condições sejam processadas nas etapas seguintes do ciclo de vida.

Por fim, a filtragem de eventos possibilita que aplicações especifiquem as situações de contexto as quais desejam ser notificadas. A filtragem de eventos, por exemplo, proporciona robustez e dinamicidade em sistemas sensíveis a contexto que se comunicam através do paradigma *publish/subscribe*. Nesse paradigma, os tópicos representam situações de contexto estáticas, pré-estabelecidas e, portanto, de alta granularidade, como por exemplo: em um cenário de monitoramento de sinais vitais, cada sinal vital aferido está

associado a um tópico, tais como pressão sanguínea, temperatura corpórea, frequência de pulso, frequência respiratória e saturação de oxigênio.

Entretanto, para cada sinal vital, diversas outras situações de contexto podem ser consideradas, como o seguinte evento para o tópico *Frequência de Pulso*: monitoramento dos pacientes com *idade entre 50 e 60 anos e que apresentem Taquicardia*.

Por dois motivos principalmente, esse evento não poderia ser contemplado em tópicos: é de baixa granularidade, o que exigiria que os sistemas pré-definissem uma grande quantidade de tópicos; e representam situações dinâmicas, dificilmente previstas, que correspondem às necessidades particulares das aplicações assinantes e que perduram por momentos específicos, apenas.

Considerando a implantação da infraestrutura *Hermes* em um ambiente hospitalar, por exemplo, o monitoramento dos pacientes ocorreria de forma móvel pelos profissionais de saúde, ou seja, eles estariam desempenhando suas atividades em locais diferentes ao do leito do paciente. Nesse cenário, é de grande utilidade para os profissionais de saúde a possibilidade de parametrizarem os perfis de monitoramento que desejam ser notificados.

Com respeito a esses três cenários de filtragem apresentados, fontes de contexto, dados e eventos, apenas o último será contemplado por *Hermes Interpreter*, do qual foram extraídos os seguintes requisitos de projeto:

1. Ter a capacidade de filtrar modelos ontológicos
2. Separar código-fonte de modelo ontológico
3. Priorizar execução de filtros cujos parâmetros já são conhecidos pelo HI

O objetivo do primeiro requisito é tornar o componente *Hermes Interpreter* apto para extrair dos modelos RDF de contexto, as situações semânticas das quais as aplicações assinantes desejam ser notificadas.

A finalidade do segundo requisito, separar código-fonte de modelo ontológico, é possibiltar que a arquitetura de *Hermes Interpreter* seja a mais desacoplada possível do modelo ontológico de contexto, de tal forma que o componente possa usufruir das vantagens oferecidas por esse tipo de modelagem, tais como a manutenibilidade, a extensibilidade e a reusabilidade.

O último requisito de projeto, priorizar execução de filtros cujos parâmetros já são conhecidos pelo HI, está relacionado às diferenças de complexidade entre os possíveis filtros assinados para cada tópico. Alguns deles não necessitam ser executados após a fase de inferência, pois o conhecimento requerido já está descrito no contexto publicado, como por exemplo, um assinante pode, simplesmente, requerer ser notificado caso a medida de algum sinal vital seja maior do que uma constante. Nesse exemplo, a medida do sinal vital é uma informação já publicada no contexto pelos provedores de contexto, os componentes *Hermes Widgets*.

Contudo, outros assinantes podem requerer por filtros muito mais complexos, em que suas execuções devam ocorrer após a fase de inferência, como por exemplo, um assinante necessita ser notificado se um indíviduo do contexto é de algum tipo específico ou se determinada propriedade inversa ocorre no dado de contexto publicado. Para esses dois casos, ambas informações podem ser desconhecidas, *a priori*, pelo HI.

Devido a essas diferenças, definiu-se que os filtros que não demandassem pela etapa de inferência deveriam ser executados anteriormente aos mais complexos, para que os seus assinantes sejam notificados primeiramente. Considerando a inferência baseada em ontologia em modelos ontológicos complexos, a quantidade de dados inferidos após a fase de inferência aumenta consideravelmente o tempo de filtragem, sendo, portanto, desnecessário para os primeiros assinantes aguardarem para ter os seus resultados notificados.

Os requisitos funcionais e não funcionais do *Hermes Interpreter* apresentados nessa seção, juntamente com os gerais da infraestrutura *Hermes*, constituem algumas necessidades que devem ser atendidas por um componente interpretador de contexto que deve oferecer serviços semânticos de forma transparente, flexível, manutenível, extensível e reutilizável para as aplicações sensíveis a contexto.

As decisões de projeto para atender a esses requisitos, assim como os detalhes de funcionamento do *Hermes Interpreter* e alguns cenários aos quais o componente é submetido serão apresentados na sequência desse capítulo.

## 4.3 Arquitetura

A arquitetura de *Hermes Interpreter* foi projetada em camadas de forma a atender aos requisitos não funcionais de *manutenibilidade*, *flexibilidade* e *extensibilidade*, prevendo as alterações pelas quais o componente passará e a execução dos diferentes fluxos que o componente fará, quanto à interpretação de contexto, manipulação de modelo ontológico e filtragem semântica.

Para isso, a organização das camadas seguiu o princípio do baixo acoplamento e alta coesão, constituindo camadas bem definidas, como ilusta a Figura 4.1, e que serão descritas na próxima subseção.

Na Subseção 4.3.9, será apresentado um cenário de uso da arquitetura para melhor compreensão dessas camadas.

As subseções seguintes descrevem detalhadamente as camadas do componente *Hermes Interpreter*.

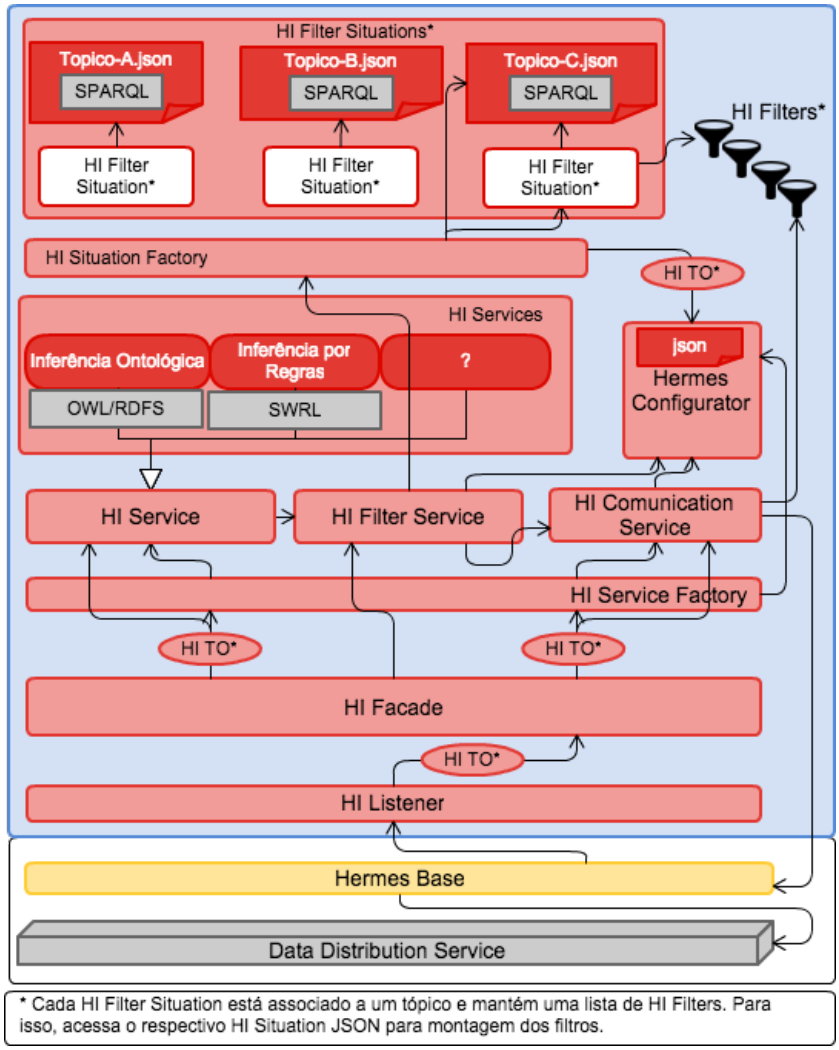


**Figura 4.1:** *Anatomia do Hermes Interpreter [27]*

### 4.3.1 HI Listener

Camada para notificação dos tópicos assinados pelo *Hermes Interpreter*. São três os tipos de tópicos assinados: solicitação de criação de filtro, notificação de contexto e configuração. Para cada tipo, *Hermes Interpreter* implementa, respectivamente, três interfaces da API de *Hermes Base*: *FilteringListenerComponent*, *NotificationListenerComponent* e *ConfigurationListenerComponent*.

Na solicitação de criação de filtro, as aplicações inicialmente solicitam à infraestrutura *Hermes* a assinatura de um tópico qualquer. Essa assinatura pode conter uma restrição de situação de contexto, em que será necessária a criação de um filtro, ou sem restrição, o que significa que o assinante deseja ser notificado de **todas** as alterações de contexto que forem publicadas pelos *Hermes Widgets* para aquele tópico.

Se a assinatura contiver restrição de contexto, o componente *Hermes Base*, então, notifica o componente *Hermes Interpreter* sobre a necessidade de criação de um filtro para o respectivo assinante no tópico recém-assinado. Essa notificação é feita através do

tópico *createFilter*, cujo objeto DDL contém os seguintes atributos: nome do tópico, o identificador do assinante e o *hashmap* do filtro do assinante. O identificador do assinante é gerado pelo *Hermes Base* e será utilizado pelo *Hermes Interpreter* para notificar aquele assinante específico posteriormente, caso a situação de contexto solicitada seja filtrada.

Quando a interação com o *Hermes Interpreter* ocorre por meio do *NotificationListenerComponent*, o componente é notificado sobre alguma alteração de contexto. Ressalta-se que *Hermes Interpreter* é assinante de todos os tópicos em que os *Hermes Widgets* publicam seus dados. Nessa situação, *Hermes Interpreter* procederá com a interpretação e inferência de fatos de contexto e consequente execução dos filtros anteriormente criados para o tópico. Os detalhes desses procedimentos serão descritos nas camadas subsequentes.

Em ambas situações descritas acima, solicitação de criação de filtro e notificação de contexto, para que os dados de contexto sejam transmitidos até as camadas *HI Filter Service*, para criação dos filtros, e *HI Service*, para interpretação do contexto, *Hermes Interpreter* utiliza a classe *HermesInterpreterTO* que encapsula os seguintes dados:

- Nome do tópico
- Entidade do contexto: identificador de alguma instância relevante dentro do modelo RDF de contexto, tal como o identificador do indivíduo paciente, cujo sinal vital está sendo notificado
- Caminho da ontologia que contém o vocabulário do RDF transmitido
- O modelo RDF de contexto
- Tipo de serialização do modelo RDF, que pode ser: Turtle, RDF/XML, N-Triple, dentre outros
- *Hashmap* do filtro solicitado pelo assinante
- Identificador do assinante

Os objetos dessa classe perpassam todas as camadas do componente, implementando, portanto, o padrão de projeto *Transfer Object* [1]. Conforme observado na Seção 2.4, esse padrão contribui para *a manutenibilidade e a extensibilidade* do componente.

Por último, o *listener ConfigurationListenerComponent* é notificado por *Hermes Base* sobre as alterações de técnicas de inferência para os tópicos do sistema. Tais solicitações são realizadas pelas aplicações, quando desejam ser notificadas de alguma situação de contexto em que determinada técnica de inferência seja mais aconselhável. O *listener* então acessa a camada *HermesInterpreterConfigurator* para que a alteração seja efetuada. A camada *HermesInterpreterConfigurator* será descrita posteriormente nessa seção.

### 4.3.2 HI Facade

São clientes da camada *Facade*, a classe de inicialização que contém o método *main()* do próprio componente *Hermes Interpreter* e a camada *Listener*, explicada anteriormente. A finalidade dessa camada é remover desses clientes o acesso direto às camadas de negócio de *Hermes Interpreter*, por meio de uma interface com métodos para registro, publicação e assinatura de tópicos, criação de filtro semântico e interpretação de contexto.

Para os métodos de criação de filtro e interpretação de contexto, a entrada é o objeto *Transfer Object*, descrito anteriormente. Esse desacoplamento desobriga os clientes de conhecerem detalhes sobre o processamento para criação dos filtros e as técnicas de inferência suportadas pelo *Hermes Interpreter*.

*HI Facade* também contribui para a flexibilidade da camada de serviço (*HI Service*) quanto às possíveis variações de tipos de inferência por tópicos notificados. Além disso, desobriga o inicializador do componente de conhecer detalhes de como proceder para a criação e assinatura dos tópicos, pois o próprio *HI Facade* invoca o serviço de comunicação, *HI Comunication Service*, explicado posteriormente, para que esse realize essas tarefas.

Por essa causa, *HI Facade* é considerada uma camada de alta granulosidade, pois executa várias ações em um só método. Essa camada implementa, portanto, o padrão de projeto *Facade* [13] que contribui principalmente para a *manutenibilidade* de *Hermes Interpreter*.

### 4.3.3 HI Service Factory / HI Situation Factory

Abstraem detalhes de instanciação dos serviços de interpretação (*HI Services*) e de filtragem de situação de contexto (*HI Filter Situations*), respectivamente, da camada solicitante, conforme ilustra a Figura 4.2, relativa à criação do serviço de interpretação. A classe *HI Service Factory* acessa *Hermes Configurator* que conhece os detalhes de acesso das configurações de *Hermes Interpreter*, ou seja, qual serviço de interpretação deve ser instanciado para o tópico corrente. Baseado nessa informação, *HI Service Factory* instancia e retorna o objeto concernente ao tópico do parâmetro.

De forma análoga, a camada *HI Situation Factory* instancia o objeto *Filter Situation* apropriado ao tópico que se deseja realizar a filtragem.

Por essas características, ambas as classes, *HI Service Factory* e *HI Situation Factory* implementam o padrão *Factory* [13] que atribui *manutenibilidade, flexibilidade e extensibilidade* ao componente com respeito à instanciação desses serviços citados.

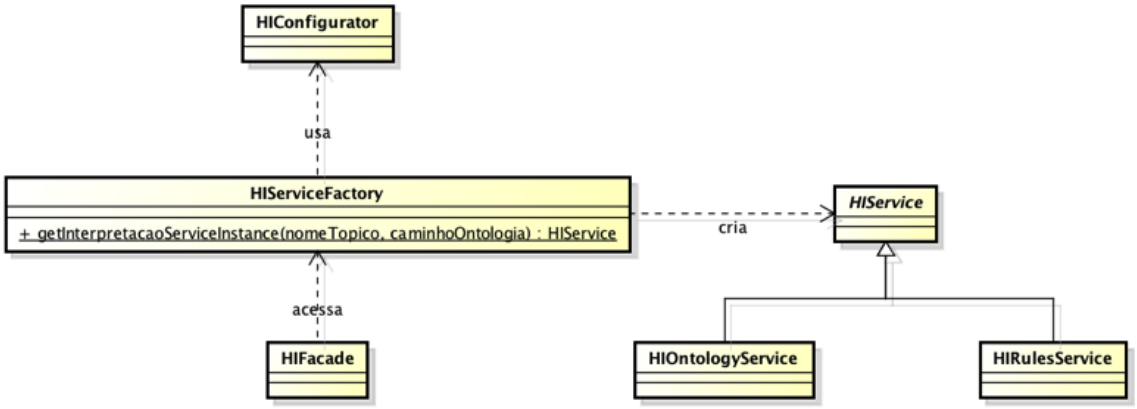


**Figura 4.2:** *Organização da camada HI Service Factory*

### 4.3.4 HI Communication Service

Invoca os serviços de comunicação do componente *Hermes Base*: registro, publicação e assinatura de tópicos, através do objeto *HermesBaseManager*, o qual centraliza o acesso ao *middleware* DDS, pois mantém as instâncias dos *Publishers*, *Subscribers* e demais objetos responsáveis pela comunicação na rede DDS.

Como *Hermes Interpreter* precisa manter somente uma instância de *HermesBaseManager*, esse serviço implementa o padrão de projeto *Singleton* [13].

Essa camada contribui para *manutenibilidade* do *Hermes Interpreter*, ao separar dos serviços de *comunicação* os demais serviços providos pelo componente. Se houver alterações na interface com *HermesBase*, somente essa classe será afetada.

### 4.3.5 HI Filter Service

Essa é uma das camadas responsáveis pela filtragem semântica, cujos requisitos foram apresentados na Subseção 4.2: ter a capacidade de filtrar modelos ontológicos, separar código-fonte de modelo ontológico e priorizar execução de filtros simples. A seguir, serão detalhadas as decisões de projeto para atender a cada um deles.

Para o primeiro requisito de projeto, **capacidade de filtrar modelos ontológicos**, foram consideradas duas possibilidades: utilizar o mecanismo de filtragem de tópicos da API do *DDS Vendor* ou a linguagem padrão da web semântica para consulta em modelos ontológicos, SPARQL.

A primeira opção se beneficia da implementação nativa do *middleware* já acessado pela infraestrutura. O *middleware* DDS possibilita que as aplicações assinem tópicos com filtros sobre os dados encapsulados no objeto *DDL* trafegado. O filtro suporta os operadores lógicos *AND, OR, NOT e IN* e os de comparação >, >=, <, <=, *LIKE e* =. Um possível filtro em DDS seria: *%sexo = F AND %data_nascimento <= '1980-05-20'*,

desde que houvesse os atributos *sexo* e *data_nascimento* no objeto *DDL* do respectivo tópico. Essa escolha apresenta duas desvantagens:

1. A necessidade de inclusão de novos parâmetros para filtragem exigiria a alteração do objeto *DDL* e de todas as assinaturas dos métodos de comunicação do *middleware* associados ao tópico do DDL alterado, pois, como descrito no Capítulo 3, DDS é uma tecnologia fortemente tipada
2. Os operadores lógicos são limitados se comparados à infinidade de propriedades ontológicas descritas em um modelo semântico. Para determinados contextos, o objeto *DDL* deveria conter vários atributos para filtrar determinada informação. Para exemplificar essa situação, considere a declaração ontológica (*ont:carro rdf:subClassOf ont:meio_transporte*) e o contexto publicado (*ont:palio rdf:type ont:meio_transporte*)
   Com apenas operadores lógicos, se alguma aplicação assinante desejasse filtrar por *meios de transporte do tipo carro*, o objeto *DDL* deveria encapsular todos os atributos necessários para definir dentre os *meios de transporte* quais são do tipo *carro* e ainda, dependendo da complexidade do domínio seria necessário que o próprio *middleware* efetuasse a análise de regras de negócio para concluir que *palio*, além de *meio de transporte*, também é um *carro*. Portanto, o acoplamento entre filtragem de contexto e o *middleware* DDS não é aconselhável em casos de domínios complexos.

A partir das desvantagens apresentadas acima, optou-se por uma solução que trabalhasse nativamente com modelagem ontológica. Escolheu-se ,então, a linguagem SPARQL, descrita na Subseção 2.2.5, solução essa desacoplada do *middleware* DDS: cabe ao objeto *DDL* somente encapsular as informações de contexto no formato RDF serializadas em um *array* de bytes e outras informações pertinentes para o componente *Hermes Base* desempenhar suas tarefas de comunicação, conforme explicado no Capítulo 3, sem esse ter a necessidade de executar regras de negócio complexas do sistema sensível a contexto ao qual Hermes está inserido.

Dentre as formas de consulta suportadas por SPARQL, escolheu-se a CONSTRUCT que, a partir das cláusulas especificadas em sua estrutura, retorna um grafo de triplas RDF como resultado (ver Subseção 2.2.5). Isso possibilita ao componente *Hermes Interpreter* notificar os assinantes com um grafo RDF, resultante da consulta, com exatamente os dados pré-definidos pelo filtro em questão, o que reduz consideravelmente o tamanho do RDF notificado.

Por exemplo, o HI está sendo aplicado em um cenário de monitoramento de sinais vitais de pacientes em UTIs, assunto que será aprofundado no Capítulo 5. Para este estudo, está sendo utilizada uma ontologia denominada MSVH [4] para descrever as

respectivas informações de contexto. A cada notificação de contexto, *Hermes Interpreter* realiza a inferência de novos fatos baseados na ontologia MSVH. Essa atividade produz um modelo de dados RDF com tamanho médio de 1,6Mb, enquanto que o tamanho do RDF gerado a partir da filtragem de dois parâmetros com a cláusula CONSTRUCT é de aproximadamente 4Kb.

A Figura 4.3 ilustra, de forma fictícia, o comparativo entre o grafo gerado após um processo de inferência e o produzido por uma consulta com CONSTRUCT. Essa forma de consulta produz um grafo com apenas as triplas referentes ao filtro semântico requerido.

Nos grafos da figura, os nós preenchidos e as arestas que se conectam a eles representam as novas informações de contexto produzidas, enquanto que o subgrafo composto pelos nós sem preenchimento com as arestas que partem deles representa o grafo de contexto original, publicado pelo provedor de contexto.

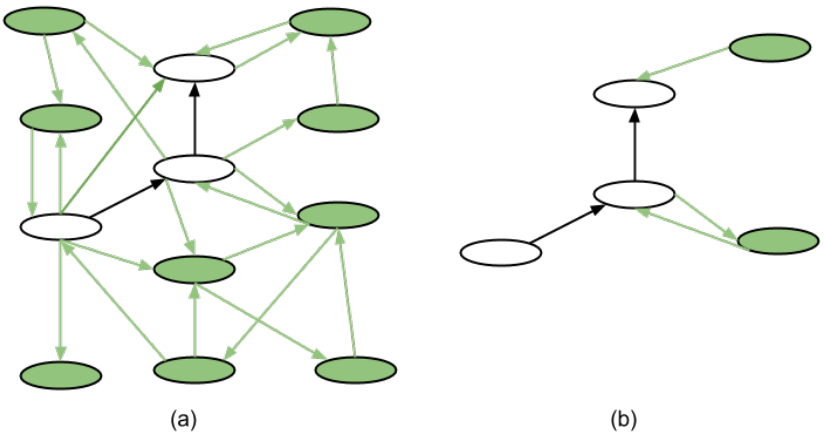


**Figura 4.3:** *(a) Grafo com todas as inferências. (b) Grafo resultante de consulta simples com CONSTRUCT.*

Para o segundo requisito, **separar código-fonte de modelo ontológico**, a solução definida foi desacoplar o código-fonte de *Hermes Interpreter* das descrições das cláusulas das consultas que se referem à ontologia do contexto. Essa separação visa tornar o componente o mais independente possível do modelo ontológico, pois, da mesma forma que as regras e modelagem do domínio se encontram no *schema* ontológico, separadas da aplicação, é recomendável que as cláusulas que irão compor as consultas SPARQL também o sejam, o que favorece a manutenibilidade e extensibilidade do modelo ontológico e da capacidade de filtragem do sistema.

O tipo de dado escolhido para armazenar as cláusulas semânticas foi o tipo *JSON*, descrito na Seção 2.3. O componente *Hermes Interpreter* mantém, para cada tópico, um arquivo JSON que descreve os templates SPARQL para cada filtro suportado pelo tópico. Além das cláusulas, o arquivo contém parâmetros que auxiliam a montagem da consulta SPARQL do filtro, conforme a Tabela 4.1.

**Tabela 4.1:** *Chaves dos arquivos JSON de filtro por tópico*

| **Nome** | **Finalidade** |
|---|---|
| construct | Cláusula *CONSTRUCT* básica, independente dos filtros escolhidos, com os respectivos prefixos das ontologias importadas. |
| where | Texto básico da cláusula *WHERE*. Contém a referida cláusula e abertura de chaves que indica o início da cláusula |
| clausula base | Declarações básicas que serão utilizadas independente dos filtros escolhidos. |
| conjuntos filtro | Tipo *array* que descreve os tipos de filtros organizados por um ID |
| id | Identifica um conjunto de filtros, no qual está relacionado a partes específicas do schema ontológico. Essa organização dos conjuntos por ID auxilia o componente no percurso, montagem e estruturação das cláusulas dos filtros. No caso de teste utilizado, existe um conjunto para os filtros relativos às informações do paciente e outro para os alarmes semânticos. |
| clausulas | Cada conjunto de filtro contém uma chave "clausulas", do tipo array, que possui como elementos os filtros disponíveis para o conjunto em questão. Cada elemento de "clausulas"é formado por uma combinação de diversos parâmetros associados aos filtros propriamente ditos. As chaves de cada elemento são: filtro, tipo dado, possui classe disjunta no filtro, possui parâmetro, construct, cláusula, cláusula filtro e tipo inferência, que serão detalhados na sequência. |
| filtro | Contém o nome de um filtro disponível para o tópico. Os nomes dos filtros escolhidos pelas aplicações assinantes e informado ao HI devem coincidir exatamente com o seu respectivo valor. O componente HI irá comparar a chave do filtro do assinante com a descrição de cada valor dessa chave para verificar se deve ou não inclui-lo na consulta SPARQL. |
| tipo dado | Tipo de dado do valor do respectivo filtro. Utilizado pelo HI para incluir no SPARQL a URI relativa ao tipo do dado. |
| possui classe disjunta no filtro | Boolean que sinaliza ao HI que a inclusão de mais de um filtro deste conjunto com valor *true* implica no acréscimo da cláusula UNION, pois se forem interligadas cláusulas de classes disjuntas com conectivo AND, a consulta retornará *vazio*. |
| possui parâmetro | Boolean que sinaliza ao HI a necessidade de inclusão da cláusula FILTER após a concatenação da cláusula principal do respectivo filtro. Considerando um filtro para *medida de sinal vital* por exemplo, deve-se incluir o valor primitivo da medida que o assinante deseja inserir como parâmetro na cláusula FILTER, como em *filter(?taxa_respiratoria > 40)*. Entretanto, existem outros tipos de filtros que não necessitam da cláusula FILTER, quando se deseja verificar, por exemplo, se uma determinada instância do RDF é de um determinado tipo T, ou se existe alguma propriedade P no modelo de dados consultado. |
| construct | Trecho relativo ao filtro corrente que deve ser concatenado à cláusula CONSTRUCT da consulta, para que tais triplas sejam acrescentadas no modelo semântico resultante. |
| cláusula | Principal chave do arquivo, contém a cláusula do filtro propriamente dita que deve ser concatenada à cláusula WHERE da consulta. |
| cláusula filtro | Se o respectivo filtro exigir a cláusula FILTER para comparação de algum parâmetro primitivo ou data, a cláusula deve ser descrita nessa chave. |
| tipo inferência | Indica qual tipo de inferência deve ser realizado sobre o modelo de contexto antes da execução da cláusula para que essa consulta tenha sucesso. Se a cláusula contiver as classes e relacionamentos ontológicos que já estão presentes no modelo de contexto original publicado pelos Hermes Widgets, então esse parâmetro é marcado com ”, o que significa que nenhum processo de inferência necessita ser realizado. Se a cláusula exigir inferência ontológica apenas, é atribuído o valor *ontologica*. Caso seja necessário o processamento de regras, o parâmetro é preenchido com valor *regras*. Para manter o valor dessa chave, o desenvolvedor do HI deve saber de antemão quais as triplas estão presentes no RDF notificado pelos Hermes Widgets e ter conhecimento do schema ontológico e das regras descritas. |

A lógica de acesso e *parsing* do arquivo *JSON* e criação da consulta SPARQL está no código do sistema, mas é totalmente desacoplada do tópico e, consequentemente, da ontologia em questão. A partir de um objeto *HashMap*, que possui como chave o nome do filtro escolhido pelo assinante e como valor, a referência para a respectiva chave, o componente constrói a consulta a ser utilizada para o filtro.

Para atender ao terceiro requisito, **priorizar execução de filtros cujos parâmetros já são conhecidos pelo HI**, estabeleceu-se como critério de complexidade a necessidade ou não de que seja executada a etapa de inferência antes da filtragem. Para isso, *Hermes Interpreter* mantém duas listas de filtros dos assinantes por tópico: uma que deve ser processada antes da inferência e outra após.

Os filtros da primeira lista são classificados como *filtros sem inferência ou FCI*. Os filtros da segunda lista são divididos entre *filtros com inferência sem classes disjuntas ou FCI* e *filtros com inferência com classes disjuntas ou FCID*. A causa para existência dos filtros FCID é permitir que sejam especificados múltiplos padrões de grafos com o objetivo de produzir um grafo que combine todas as informações associadas a quaisquer desses padrões. Para isso, esses filtros utilizam a cláusula UNION em sua estrutura para separar a ocorrência desses diversos padrões de grafos.

A indicação da necessidade de etapa de inferência para o parâmetro do filtro está registrada na chave "tipo inferência", do arquivo de configuração citado anteriormente. Se, ao final da montagem da consulta pelo *Hermes Interpreter*, for constatado que nenhum dos filtros solicitados pelo assinante exigem etapa de inferência, o filtro será adicionado na lista dos filtros que não exigem inferência e, portanto, terá prioridade na execução. Se for exigido, será adicionado na lista dos que exigem inferência e, assim, será executado posteriormente.

O diagrama de atividades da Figura 4.4 exibe o fluxo de ações resumido da montagem da consulta do filtro. O diagrama também ilustra as atividades pertinentes à análise sobre a necessidade de inferência para o respectivo parâmetro e ao final, a inclusão do filtro em uma das listas disponíveis: filtros sem inferência ou filtros com inferência.

As Tabelas 4.2 e 4.3 e os Códigos 4.1, 4.2 e 4.3 exemplificam, respectivamente, um filtro escolhido por um assinante para o tópico *Frequência Respiratória*, alguns parâmetros do arquivo JSON do referido tópico, a consulta SPARQL gerada e um modelo RDF serializado em Turtle resultante da filtragem de um contexto relativo a esse sinal vital.

O filtro escolhido, descrito na Tabela 4.2, é formado pelas chaves *idade limite inferior* e pelos alarmes associados às anormalidades do sinal vital *Frequência Respiratória*: *apneaalarm*, *bradypneaalarm* e *tachypneaalarm*. Como valores, para *idade limite inferior*, escolheu-se a idade de 50 anos como limite inferior, o que significa que o assinante deseja ser notificado dos sinais vitais de pacientes cujas idades sejam superiores a

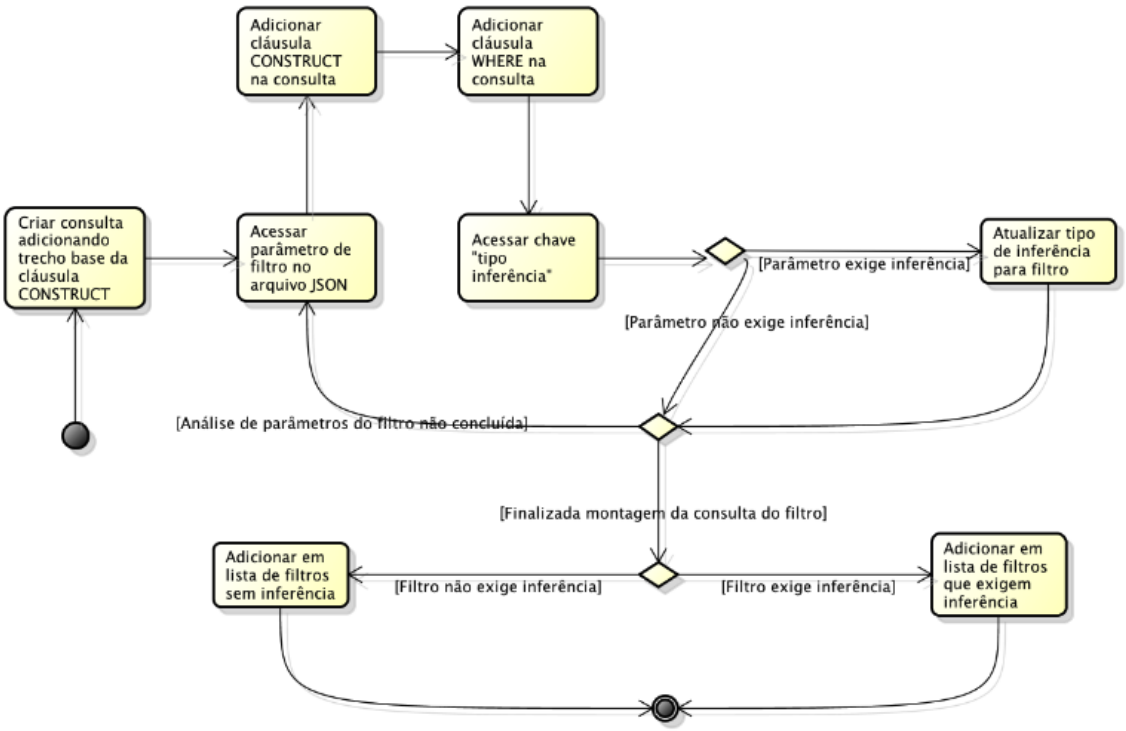


**Figura 4.4:** *Diagrama de atividades resumido da montagem de consulta para filtro semântico.*

50 anos. Para as chaves associadas aos alarmes, os valores são referentes aos respectivos nomes das classes ontológicas.

**Tabela 4.2:** *Filtro escolhido*

| **Chave** | **Valor** |
|---|---|
| idade limite inferior | 50 |
| apneaalarm | msvh:ApneaAlarm |
| bradypneaalarm | msvh:BradypneaAlarm |
| tachypneaalarm | msvh:TachypneaAlarm |

Para montagem do filtro semântico, *Hermes Interpreter* contém um arquivo JSON associado a cada tópico[3], do qual se extraíram os dados que compõem a Tabela 4.3, relativos ao tópico em questão, *Frequência Respiratória*. Foram apresentados somente os parâmetros do filtro de exemplo e as respectivas chaves, cujas explicações estão descritas na Tabela 4.1 que servem de apoio para construção do filtro.

Da análise da Tabela 4.3, o componente é capaz de criar a consulta SPARQL apresentada nos Códigos 4.1 e 4.2. A cláusula CONSTRUCT indica que o grafo resultante será formado somente pelas triplas associadas aos parâmetros do filtro, que são a instância do monitoramento de frequência respiratória, o paciente monitorado com sua *data de*

[3]O arquivo JSON pertence à camada *HI Filter Situation* que será descrita a seguir.

**Tabela 4.3:** *Parâmetros do arquivo json para o tópico Frequência Respiratória*

| **Filtro** | **Alguns parâmetros configurados para o filtro** |
|---|---|
| idade limite inferior | **possui classe disjunta no filtro**:false;<br>**possui parametro**:true;<br>**clausula**: ?x activity:hasParticipant ?y . ?y msvh:hasRole msvh:patient . ?y actor:hasBirthday ?data_nascimento .;<br>**clausula filtro**:filter(?data_nascimento <= ?idade_limite_inferior);<br>**tipo inferencia**:' ' |
| apneaalarm | **possui classe disjunta no filtro**:true;<br>**possui parametro**:false;<br>**clausula**: ?x msvh:hasMonitoringRespiratoryRate ?m . ?m msvh:hasMeasurementRespiratoryRate ?medida . ?medida rdf:type msvh:ApneaAlarm . ?sub_alarme rdfs:subClassOf msvh:RespiratoryRateAlarm . ?medida rdf:type ?sub_alarme .;<br>**tipo inferencia**:"regras" |
| bradypnea alarm | **possui classe disjunta no filtro**:true;<br>**possui parametro**:false;<br>**clausula**: ?x msvh:hasMonitoringRespiratoryRate ?m . ?m msvh:hasMeasurementRespiratoryRate ?medida . ?medida rdf:type msvh:BradypneaAlarm . ?sub_alarme rdfs:subClassOf msvh:RespiratoryRateAlarm . ?medida rdf:type ?sub_alarme . ;<br>**tipo inferencia**:"regras" |
| tachypnea alarm | **possui classe disjunta no filtro**:true;<br>**possui parametro**:false;<br>**clausula**: ?x msvh:hasMonitoringRespiratoryRate ?m . ?m msvh:hasMeasurementRespiratoryRate ?medida . ?medida rdf:type msvh:TachypneaAlarm . ?sub_alarme rdfs:subClassOf msvh:RespiratoryRateAlarm . ?medida rdf:type ?sub_alarme . ;<br>**tipo inferencia**:"regras" |

*nascimento* e a instância do *alarme* associado com a medida do sinal vital publicado, por meio da cláusula *?medida rdf:type ?sub_alarme*.

Devido à presença das classes disjuntas *msvh:ApneaAlarm*, *msvh:BradypneaAlarm* e *msvh:TachypneaAlarm* dentre os parâmetros do filtro, deve-se utilizar o operador UNION para separá-las dentro da estrutura da consulta. Esse operador deve ser utilizado para garantir que a ocorrência de alguma instância dessas classes, se existente na fonte de dados, seja retornada no grafo resultante, pois um indivíduo nunca pertencerá a duas ou três dessas classes simultaneamente, conforme explicado na Subseção 2.2.3. Por essas características, esse filtro é classificado como um FCID.

**Código 4.1** SPARQL para um filtro de Frequência Respiratória - parte 1 de 2

```
CONSTRUCT
  { ?x rdf:type msvh:MonitoringRespiratoryRate .
    ?x activity:hasParticipant ?y .
    ?y msvh:hasRole msvh:patient .
    ?y actor:hasBirthday ?data_nascimento .
    ?medida rdf:type ?sub_alarme .}
WHERE
  { ?x activity:hasParticipant ?y .
    ?y msvh:hasRole msvh:patient .
    ?y actor:hasBirthday ?data_nascimento
    FILTER ( ?data_nascimento <= "1964-12-27"^^xsd:date )
      { ?x rdf:type msvh:MonitoringRespiratoryRate .
        ?x msvh:hasMonitoringRespiratoryRate ?m .
        ?m msvh:hasMeasurementRespiratoryRate ?medida .
        ?medida rdf:type msvh:ApneaAlarm .
        ?sub_alarme rdfs:subClassOf msvh:RespiratoryRateAlarm .
        ?medida rdf:type ?sub_alarme
      }
```

**Código 4.2** SPARQL para um filtro de Frequência Respiratória - parte 2 de 2

```
    UNION
      { ?x rdf:type msvh:MonitoringRespiratoryRate .
        ?x activity:hasParticipant ?y .
        ?y msvh:hasRole msvh:patient .
        ?y actor:hasBirthday ?data_nascimento
        FILTER ( ?data_nascimento <= "1964-12-27"^^xsd:date )
        ?x msvh:hasMonitoringRespiratoryRate ?m .
        ?m msvh:hasMeasurementRespiratoryRate ?medida .
        ?medida rdf:type msvh:BradypneaAlarm .
        ?sub_alarme rdfs:subClassOf msvh:RespiratoryRateAlarm .
        ?medida rdf:type ?sub_alarme
      }
    UNION
      { ?x rdf:type msvh:MonitoringRespiratoryRate .
        ?x activity:hasParticipant ?y .
        ?y msvh:hasRole msvh:patient .
        ?y actor:hasBirthday ?data_nascimento
        FILTER ( ?data_nascimento <= "1964-12-27"^^xsd:date )
        ?x msvh:hasMonitoringRespiratoryRate ?m .
        ?m msvh:hasMeasurementRespiratoryRate ?medida .
        ?medida rdf:type msvh:TachypneaAlarm .
        ?sub_alarme rdfs:subClassOf msvh:RespiratoryRateAlarm .
        ?medida rdf:type ?sub_alarme
      }
  }
```

Para finalizar o exemplo da filtragem, é exibido no Código 4.3 o modelo RDF em Turtle resultante da consulta que contém o indivíduo *:mRespiratoryRate01*, pertencente a classe de monitoramento *:MonitoringRespiratoryRate* que, por sua vez, tem um relacionamento com o indivíduo paciente *:person132539*. Além dessas instâncias, contém a medida obtida *:FreqResp0* do tipo *:TachypneaAlarm* e a instância do paciente com a *data de nascimento*.

**Código 4.3** Modelo gerado após filtro com CONSTRUCT

```
Ontology1361391792831:mRespiratoryRate01
      a       Ontology1361391792831:MonitoringRespiratoryRate ;
      activity:hasParticipant
              Ontology1361391792831:person132539 .

Ontology1361391792831:FreqResp0
      a       Ontology1361391792831:TachypneaAlarm .

Ontology1361391792831:person132539
      actor:hasBirthday "1960-12-27"^^xsd:date ;
      Ontology1361391792831:hasRole
              Ontology1361391792831:patient .
```

Ao final da consulta, para cada filtro atendido, o novo modelo gerado é adicionado ao modelo de contexto original, e é solicitado à camada *HI Communication Service* a publicação do contexto para o assinante específico.

### 4.3.6 HI Service

Oferece uma interface única para inferência de contexto, na qual a camada *Facade* solicitante faz sua requisição. *HI Service*, a nível de implementação, é uma classe abstrata que possui o método *inferirSituacao*, cuja assinatura é um objeto *HI Transfer Object*, explicado anteriormente. A cada nova técnica de inferência incorporada, é necessário que se estenda essa classe e implemente o respectivo método. No momento, duas técnicas de inferência são suportadas: ontológica e baseada em regras. A camada, dessa forma, implementa o padrão *Strategy* [13]. Esse padrão permite que o componente seja *extensível* quanto à quantidade de mecanismos de inferência suportados, atendendo, assim, o requisito *Suporte a múltiplas técnicas de interpretação de contexto* apontado por [34].

Cabe também a esse serviço, antes de realizar a inferência de contexto, verificar se os modelos ontológicos instanciados e as consultas dos filtros já criados estão sincronizados com a ontologia do contexto. Essa verificação é necessária, porque o modelo ontológico é mantido à parte do código do sistema. Assim, para garantir a consistência das etapas acima, o serviço consulta, através da camada *HI Configurator*, se houve alteração na ontologia. Esse indicador fica registrado no arquivo *topics.json* que mantém uma lista das ontologias suportadas por *Hermes*. Para cada ontologia, é definida uma chave *atualizada* do tipo *boolean* que é mantida manualmente por um usuário administrador. Se for *false*, todos os filtros dos assinantes são refeitos, conforme alterações nos respectivos

arquivos JSON de *HI Filter Situations* para se adequarem às mudanças na ontologia. O serviço de inferência também executa o passo de classificação dos modelos ontológicos para atualizá-los com respeito às mudanças ocorridas.

Outro princípio de projeto citado pela literatura é a *Verificação de consistência entre fatos e modelo de contexto* [6]. O alto grau de formalismo da modelagem ontológica possibilita identificar possíveis inconsistências que possam surgir da instanciação de fatos de contexto em um esquema ontológico. A principal vantagem dessa característica é não permitir que fatos inválidos sejam interpretados e persistidos na base de conhecimento do componente. Para garantir essa corretude das informações, *HI Service* realiza essa verificação antes do procedimento de interpretação de contexto.

Antes e após o procedimento de inferência, o serviço invoca a camada *HI Filter Service* para que essa proceda com a filtragem do contexto: antes da inferência, executando os filtros que não necessitam de inferência, e após, os filtros que exigem a inferência.

### 4.3.7 HI Filter Situations

As informações de contexto de alto nível produzidas pela interpretação e inferência de contexto de baixo nível são também referenciadas pela literatura como *Situações de contexto* [6]. Os elementos nessa camada do *Hermes Interpreter* encapsulam tais situações de contexto e têm as seguintes funcionalidades: (i) construir as consultas na linguagem SPARQL que atendam às demandas de situações de contexto das quais os usuários das aplicações desejam ser notificados e (ii) manter a lista de filtros desses assinantes para o tópico que estão associados. Para isso, essa camada é composta pela classe *HI Filter Situation* e pelo arquivo de filtro, no formato JSON, associado a algum tópico previamente definido em *Hermes*. O arquivo JSON, conforme a Subseção 4.2, descreve os templates SPARQL, de algum *schema* ontológico, relativos aos parâmetros dos seus respectivos filtros.

A camada *HI Filter Situations* implementa o padrão de projeto *Domain Object* [1], pois, por meio dos arquivos JSON, é a única no *Hermes Interpreter* que está acoplada ao modelo de domínio utilizado na infraestrutura, ou seja, o modelo ontológico de contexto.

O padrão de projeto *Domain Object* oferece *manutenibilidade e extensibilidade* ao componente, pois conforme novas classes vão sendo incluídas, alteradas ou excluídas da ontologia, somente essa camada sofre alterações. Caso o sistema necessite incorporar novos parâmetros de filtragem pelos usuários, bastaria inclui-los no arquivo JSON associado ao tópico.

**HI Filter**: Classe POJO[4] que encapsula as informações do filtro de um assinante, tais como os parâmetros do filtro, a consulta SPARQL, o identificador do assinante na infraestrutura *Hermes* e o tipo inferência exigido para o filtro.

Essa classe é importante pois constitui a estrutura elementar dos filtros criados por *Hermes Intepreter*. As instâncias dessas classes são mantidas nas listas de filtros de *HI Filter Situations*, para serem acessadas a cada notificação de contexto do respectivo tópico.

### 4.3.8 Hermes Configurator

Esse componente não é específico do *Hermes Interpreter*, pois cada integrante de *Hermes* possui o seu respectivo *Hermes Configurator* para acessar suas configurações de funcionamento que são registradas no arquivo ***topics.json***. No *Hermes Interpreter*, especificamente, os parâmetros configuráveis são de dois tipos: *Tópicos de acesso* e *Inferência por tópico*. No primeiro tipo, podem ser parametrizados:

- Os tópicos que devem ser registrados localmente, sobre os quais *Hermes Interpreter* interage, ou para publicação ou para assinatura
- Os tópicos em que o componente deve realizar assinatura

O segundo tipo de parâmetro, *Inferência por tópico*, inclui as seguintes chaves:

i) O caminho dos arquivos JSON *HI Filter Situations* por tópico: Possibilita a parametrização da localização dos arquivos de domínio para apoiar as tarefas de filtragem de contexto, conforme os tópicos e a ontologia corrente. Essa característica é útil para que a aplicação se adeque às mudanças do modelo ontológico, por exemplo: se for necessária a inclusão de novo tópico, referente à ontologia corrente ou a uma nova ontologia, e sobre esse tópico seja necessária a inclusão da funcionalidade de filtragem, bastaria criar o respectivo arquivo JSON *HI Filter Situation* e adicioná-lo no arquivo de configuração. A configuração desse parâmetro confere manutenibilidade e extensibilidade ao componente.

ii) *Mecanismo de inferência* por tópico: visa tornar o componente flexível para contemplar o melhor método de inferência para as diversas situações de contexto associadas aos tópicos. Para isso, a infraestrutura *Hermes* dispõe de um tópico específico relativo à *Configuração de Técnica de Inferência* em que os demais componentes ou aplicações podem publicar. A alteração desse parâmetro também pode ser feita automaticamente, conforme os filtros semânticos vão sendo criados, o HI solicita que a técnica de inferência seja atualizada para a respectiva técnica de inferência associada ao filtro.

A Tabela 4.4 resume as configurações supracitadas em que, para cada Tópico de Registro, deve-se parametrizar os itens Assinar, Tipo de Inferência e Situation.

[4] http://www.martinfowler.com/bliki/POJO.html

**Tabela 4.4:** *Parâmetros configuráveis do Hermes Interpreter*

| Nome | Finalidade | Mantenedor |
|---|---|---|
| Ontologias | Lista que descreve as ontologias acessadas por Hermes e está associada a uma chave boolean denominada *atualizada*, que indica ao componente se houver alterações no modelo ontológico | Usuário Administrador |
| Registro | Especificar tópico para registro no *Middleware*, no qual HI publica ou assina | Usuário Administrador |
| Ontologia | Para cada tópico, registra-se a ontologia associada a ele. | Usuário Administrador |
| Assinar | Especificar tópico do parâmetro *Registro* com os respectivos complementos, para os quais HI deve solicitar assinatura | Usuário Administrador |
| Tipo de Inferência | Especificar técnica de inferência para o tópico registrado | Demais componentes via tópico de configuração |
| Situation | Especificar a localização dos arquivos JSON que descrevem os eventos de contexto para filtragem | Usuário Administrador |

A coluna *Mantenedor* se refere ao responsável por manter aquele item de configuração. Os parâmetros do *Hermes Interpreter* são mantidos ou por um *Usuário Adminitrador*, usuário especializado conhecedor de detalhes técnicos do sistema, ou por componentes ou aplicações sensíveis a contexto que interagem com *Hermes Interpreter*.

Descritos os detalhes arquiteturais de *Hermes Interpreter*, serão apresentados na subseção seguinte, alguns dos principais cenários de execução do componente. Esses passos visam auxiliar o leitor sobre as camadas recém-apresentadas e prepará-lo para o estudo de caso contemplado no Capítulo 5.

### 4.3.9 Cenários de execução

Os cenários a seguir ilustrarão situações comuns da execução do *Hermes Intepreter*, no contexto de monitoramento de sinais vitais humanos. A formulação desse cenário teve participação da equipe de Enfermagem do Hospital das Clínicas da UFG.

O primeiro cenário é ilustrado na figura 4.5. No passo (1), a aplicação assinante solicita alteração de técnica de inferência para *ontológica* para os tópicos *Frequência de Pulso* e *Temperatura Corpórea*. Os motivos para esse tipo de solicitação podem ser as diferenças de desempenho ou quantidade de fatos novos produzidos entre as técnicas de inferência suportadas. A capacidade de alterar a técnica de inferência para um tópico de forma transparente para as aplicações, ou seja, somente por meio de publicação em tópico de configuração, é resultante da *flexibilidade* que a infraestrutura oferece para as aplicações.

No passo (2) *Hermes Interpreter* é notificado sobre publicação no tópico de configuração. Concluindo o cenário, no passo (3), *Hermes Interpreter* acessa camada *HI Configurator* e solicita que a alteração seja feita no arquivo *topics.json*, para os respectivos tópicos.

Para efetuar essa solicitação de alteração na configuração do *Hermes Interpreter*, é necessário que o usuário domine a ontologia do contexto em que *Hermes* está inserido, pois essa alteração poderá impactar na geração dos novos fatos de contexto e, consequentemente, na execução dos filtros já existentes.

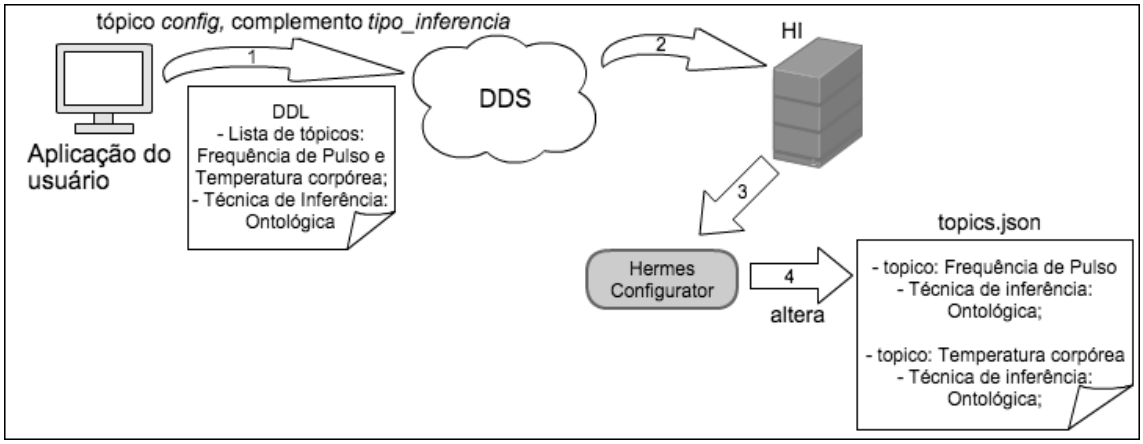


**Figura 4.5:** *Aplicação solicita alteração de técnica de inferência para alguns tópicos*

Os cenários seguintes abordam as quatro etapas de interpretação e filtragem de contexto, finalidade principal do componente. Na primeira etapa, Figura 4.6, é ilustrada a assinatura do tópico *Frequência de Pulso*.

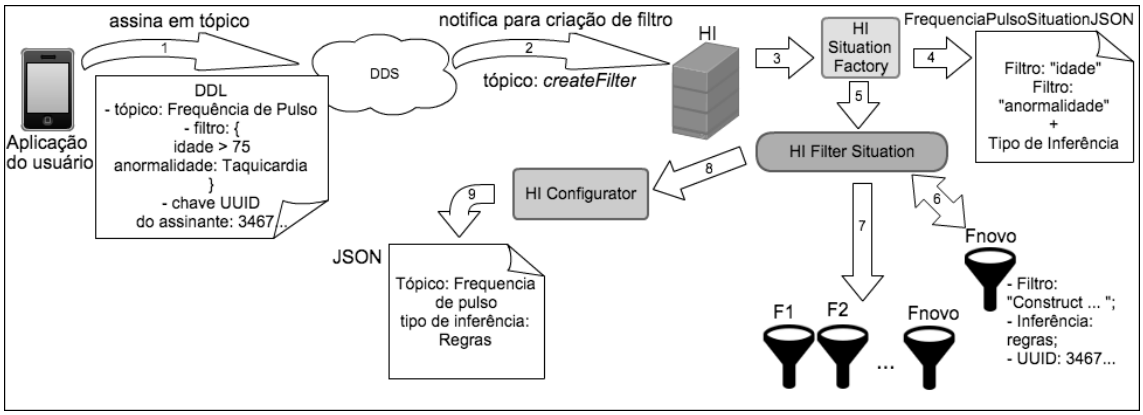


**Figura 4.6:** *Aplicação assina tópico com filtro*

No passo (1), dentre os atributos do objeto DDL, foram destacados o tópico que se deseja fazer assinatura, o filtro para *idade > 75 anos e anormalidade Taquicardia* e a chave de identificação do assinante, gerada pelo componente *Hermes Base*.

No passo (2), ao ser notificado através do tópico *createFilter*, *Hermes Interpreter* procede com a criação do filtro. No passo (3), a fábrica de *HI Situations* é acessada para se obter uma instância da classe *HI Filter Situation* responsável pelo tópico a ser assinado.

Nos passos (4) e (5), a fábrica então retorna uma instância da classe *HI Filter Situation* referente ao tópico juntamente com o arquivo JSON que descreve os *templates*

SPARQL para criação do filtro. Por meio desses dois elementos, objeto *filter situation* e arquivo de *template*, o componente procede com a criação da consulta para o assinante.

No passo (6), é instanciado um objeto *Filter*, ao qual são adicionados a consulta SPARQL recém-criada e demais atributos do filtro. No passo (7), esse objeto é incluído na fila apropriada de filtros do *HI Filter Situation*, observando o critério de necessidade de inferência para a execução da consulta. No exemplo, devido à complexidade do filtro, é necessária a etapa de inferência para execução desse filtro e, portanto, ele é adicionado na lista dos filtros que necessitam do passo de inferência antes de sua execução.

Para finalizar o fluxo, nos passos (8) e (9), como a técnica de inferência exigida para essa consulta SPARQL é *inferência por regras*, é então alterada a configuração do *Hermes Interpreter* quanto ao tipo de inferência para o tópico *Frequência de Pulso*. Conforme descrito anteriormente, essa substituição foi possível porque o tipo de inferência baseado em *regras* é mais abrangente do que o tipo *ontológica* que era o tipo de inferência registrado anteriormente para o tópico *Frequência de Pulso*.

Na segunda etapa do processo de interpretação de contexto, representada na Figura 4.7, são ilustrados os passos referentes à publicação no tópico *Frequência de Pulso*, cujo objeto DDL, no passo (1), contém o contexto notificado no formato RDF[5].

Nos passos (2) e (3), ao ser notificado, *Hermes Interpreter*, por meio da camada de configuração *Hermes Configurator*, acessa o arquivo *topics.json* que descreve a técnica de inferência que deve ser utilizada para o tópico publicado. Para o tópico *Frequência de Pulso*, a técnica de inferência mapeada é a *baseada em Regras*. Baseada nisso, no passo (4) então, *Hermes Interpreter* instancia o serviço *HI Service Regras*.

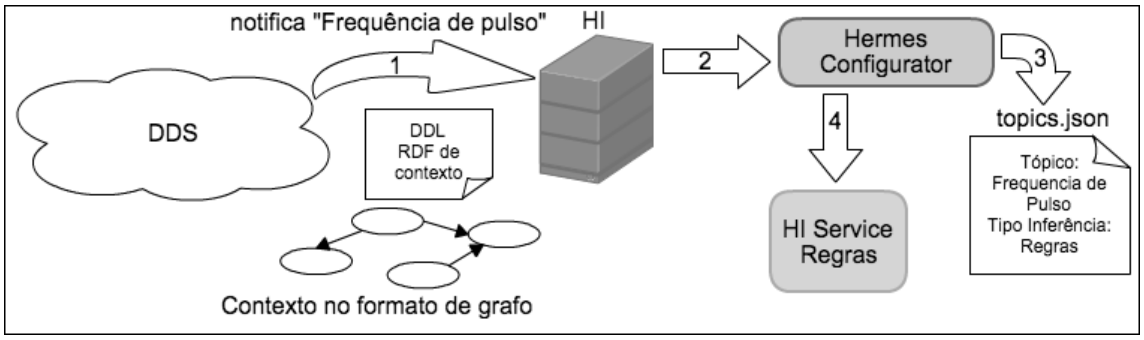


**Figura 4.7:** *Instanciação de serviço de inferência*

Na terceira etapa, ocorre a filtragem de contexto sem inferência, abordada na Figura 4.8. A camada *HI Service Regras*, nos passos (1) e (2), comunica o contexto **original** em RDF à camada *Filter Service*, para que ocorra a filtragem do contexto.

No passo (3), *Filter service* então, acessa o objeto *FrequenciaPulsoSituation* para obter as listas dos filtros que não exigem inferência. De posse da lista, *Filter Service*

[5]Devido ao pouco espaço na figura, o componente *Hermes Widget* foi omitido.

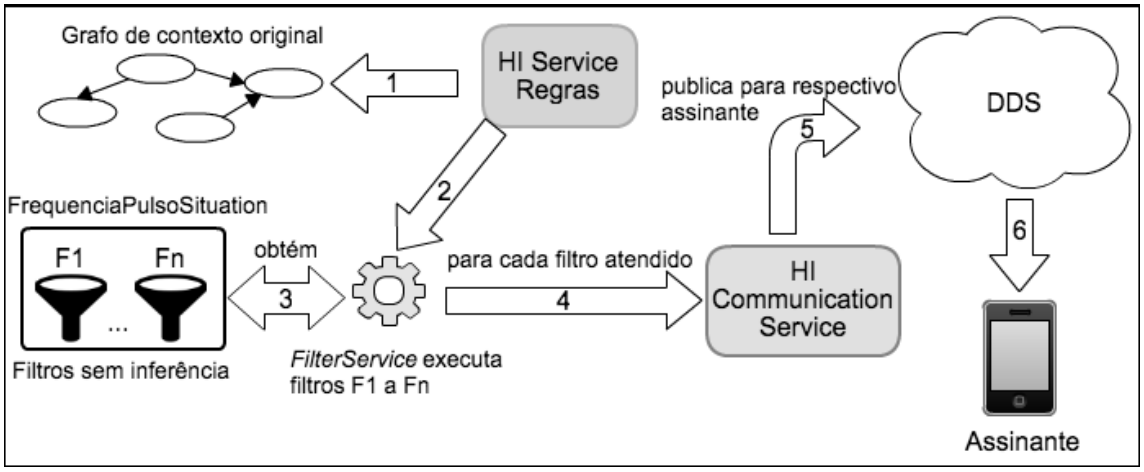


**Figura 4.8:** *Filtragem de contexto SEM inferência*

executa a filtragem de cada elemento e, no passo (4), para cada filtro atendido, solicita à camada *HI Communication Service* que publique o contexto para o respectivo assinante, conforme os passos (5) e (6).

A quarta etapa, que é referente à interpretação e inferência de novas situações de contexto, está ilustrada na Figura 4.9.

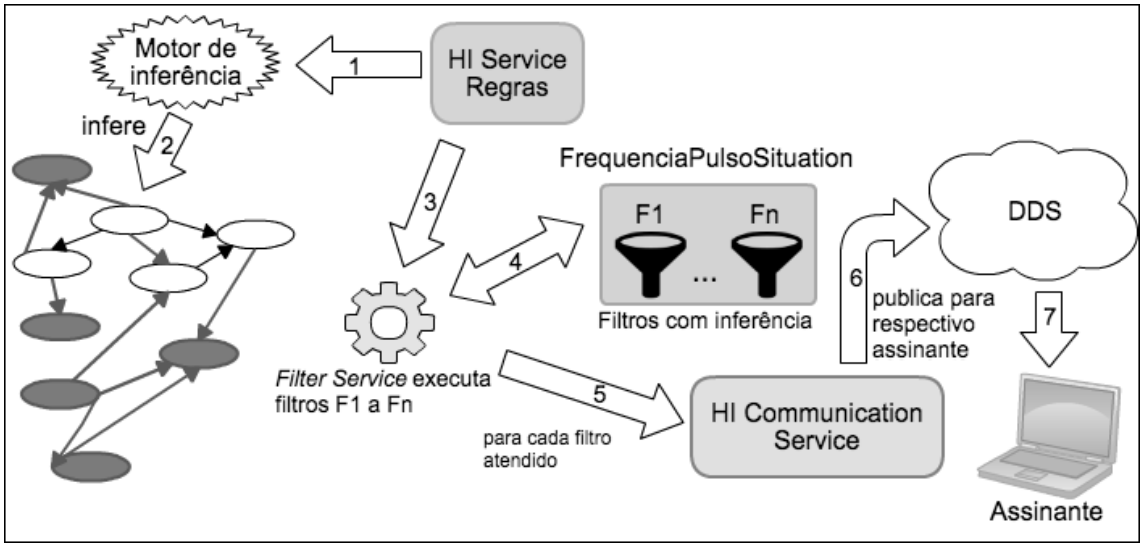


**Figura 4.9:** *Filtragem de contexto COM inferência*

No passo (1), *HI Service Regras* instancia o *motor de inferência* que efetua, no passo (2), a descoberta de novos fatos de contexto sobre o modelo RDF original notificado. Na Figura 4.9, esses novos fatos são representados pelos nós e arestas escuros do grafo.

Na sequência, passo (3), o serviço de regras acessa a camada *Filter Service* para proceder com a filtragem do contexto. Essa camada, de acordo com o passo (4), acessa o *HI Situation* específico para o tópico que é o objeto *FrequenciaPulsoSituation* que mantém a lista de filtros que demandam inferência.

Na sequência, *Filter Service* executa a filtragem de cada um e, para cada filtro atendido, conforme passo (5), solicita ao *HI Communication Service* que publique na rede DDS para o assinante correspondente, conforme passos (6) e (7).

## 4.4 Avaliação de manutenibilidade

*Hermes Interpreter* foi avaliado quanto à sua manutenibilidade segundo métricas consolidadas da indústria. Devido à ausência de normas para validar os demais requisitos não funcionais contemplados, tais como extensibilidade e flexibilidade, esses serão indiretamente validados por meio de alguns casos de teste descritos no Apêndice B e apresentados no Capítulo 5.

A manutenibilidade de software está entre os requisitos não funcionais mais relevantes dentre os critérios da qualidade de software. Sua importância é tal, que, as organizações ISO[6] e IEC[7] criaram a norma ISO/IEC 25000:2014[8], cujo propósito é normatizar o processo de avaliação da qualidade do produto de software, qualidade essa que tem a manutenibilidade como um dos seus critérios de avaliação.

Apesar da existência da versão 2014 da norma ISO/IEC 25000, a avaliação de *Hermes Intepreter* foi realizada com uma de suas versões anteriores, a ISO 9126[9], pois, como será explicado adiante, os critérios estabelecidos para se pontuar os itens de qualidade foram extraídos do *Modelo de Manutenibilidade SIG*[21] que foi desenvolvido baseado na versão ISO 9126. Para coleta dos indicadores da norma, foi utilizada a ferramenta open-source *SonarQube*.

Conforme a ISO, para avaliar a *manutenibilidade* de um software, devem ser consideradas as seguintes subcaracterísticas:

- Analisabilidade (*analysability*)
- Capacidade de alternância (*changeability*)
- Estabilidade (*stability*)
- Testabilidade (*testability*)

Para se determinar o indicador de cada subcaracterística, a norma estabelece um conjunto de propriedades que deve ser observado nos códigos-fonte dos sistemas:

- Volume: quantidade total de linhas de código-fonte

[6] http://www.iso.org/iso/home.html
[7] http://www.iec.ch/
[8] http://www.iso.org/iso/home/store/catalogue$_$tc/catalogue$_$detail.htm?csnumber=64764
[9] http://www.iso.org/iso/catalogue$_$detail.htm?csnumber=22749

- Complexidade ciclomática por unidade: determinada pela estrutura interna, como dependências e pontos de decisão, como condicionais ou laços do código-fonte. Em Java, por exemplo, uma unidade equivale a um método
- Duplicação: nível de repetição de código em várias partes do sistema
- Tamanho da unidade: tamanho médio dos métodos
- Teste de unidade: cobertura dos testes de unidade do sistema

A Tabela 4.5 relaciona como cada propriedade influencia nas subcaracterísticas de manutenibilidade:

**Tabela 4.5:** *Relação entre subcaracterísticas e propriedades do código-fonte*

| | Volume | Complexidade por unidade | Duplicação | Tamanho da unidade | Teste de unidade |
|---|---|---|---|---|---|
| Analisabilidade | x | | x | x | x |
| Capacidade de Alternância | | x | x | | |
| Estabilidade | | | | | x |
| Testabilidade | | x | | x | x |

Para pontuação das propriedades, decidiu-se pelo *Modelo de Manutenibilidade SIG*[21]. Nesse modelo, são adotadas as seguintes pontuações para cada propriedade que serão resumidamente descritas a seguir:

**Volume**: utiliza-se a convenção *Idade do ser humano proporcional à quantidade de linhas por linguagem* que correlaciona expressividade e produtividade das linguagens em um só indicador. Por exemplo, um sistema Java de 1400 KLOC é considerado *muito grande* (- -), enquanto um sistema Cobol de 1400 KLOC é considerado apenas *grande* (-). A Tabela 4.6 exemplifica com 3 linguagens de programação[21]:

**Tabela 4.6:** *Relação idade do ser humano (MY) x linhas de código-fonte (KLOC)[21]*

| ranking | MY | Java | Cobol | PL/SQL |
|---|---|---|---|---|
| ++ | 0 - 8 | 0-66 | 0-131 | 0-46 |
| + | 8 - 30 | 66-246 | 131-491 | 46-173 |
| o | 30 - 80 | 246-665 | 491-1310 | 173-461 |
| - | 80 - 160 | 665-1310 | 1310-2621 | 461-922 |
| - - | > 160 | > 1310 | > 2621 | > 922 |

Pela ferramenta *SonarQube*, verificou-se que *Hermes Interpreter* contém 1164 linhas de código, ou 1,164 KLOC, o que é classificado, pelo modelo SIG, um sistema *muito pequeno* (++).

**Complexidade ciclomática por unidade**: são utilizadas duas referências para obter a pontuação de *complexidade por unidade*. A primeira foi desenvolvida pela *Software Engineering Institute*[10] e classifica o nível de risco por unidade associado à sua complexidade. A Tabela 4.7 expõe tal relação.

**Tabela 4.7:** *Complexidade ciclomática por unidade*

| CC | Avaliação de risco |
|---|---|
| 1-10 | simples, sem muito risco |
| 11-20 | mais complexa, risco moderado |
| 21-50 | complexa, alto risco |
| >50 | difícil teste, altíssimo risco |

Da relação anterior, Heitlager et al.[21] propôs a agregação de complexidades por unidade relacionadas à quantidade total de linhas do código-fonte. Como exemplo dessa relação, se 40% das linhas de código de um sistema possuem complexidade *moderada* e 12%, complexidade *alta*, então esse software é classificado como *complexo* (-). A Tabela 4.8 detalha as possíveis classificações:

**Tabela 4.8:** *Complexidade por LOC[21]*

| ranking | moderada | alta | muito alta |
|---|---|---|---|
| ++ | 25% | 0% | 0% |
| + | 30% | 5% | 0% |
| o | 40% | 10% | 0% |
| - | 50% | 15% | 5% |
| - - | - | - | - |

Dos dados coletados de *SonarQube* para *Hermes Interpreter*, foi gerado o gráfico 4.10 de complexidade por linhas de código, indicando que o software é considerado *simples* (++), em que 964 linhas de código são consideradas simples e 200, moderadas.

**Duplicação**: quanto à duplicidade de código, o modelo SIG desenvolveu o esquema mostrado na Tabela 4.9, em que se percebe a relação indireta entre manutenibilidade e quantidade de duplicidades.

**Tabela 4.9:** *Classificação da duplicidade[21]*

| Ranking | Duplicação |
|---|---|
| ++ | 0-3% |
| + | 3-5% |
| o | 5-10% |
| - | 10-20% |
| - - | 20-100% |

Com o auxílio de *SonarQube*, todas as duplicidades de *Hermes Interpreter* foram corrigidas[11], fazendo com que o software fosse classificado como ++ nesse quesito.

[10] http://www.sei.cmu.edu/searchresults.cfm

[11] Na primeira vez que foi executado, o *SonarQube* acusou 3,3 % de duplicidade.

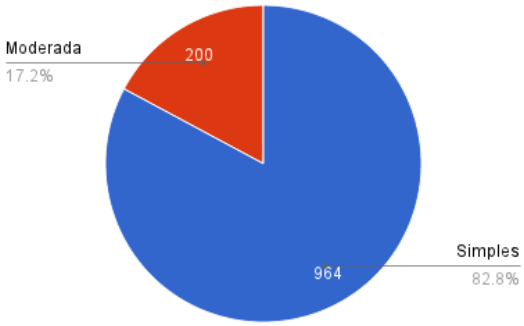


**Figura 4.10:** *Complexidade de Hermes Interpreter por linha de código*

**Tamanho por unidade**: a métrica é a mesma adotada para complexidade por unidade, ou seja, quanto maior o método, mais difícil é sua manutenibilidade, pois os seus indicadores de analisabilidade e testabilidade são baixos. Para medir o tamanho da unidade, utiliza-se a contagem por linha de código simples, em que a categoria de riscos e classificações são semelhantes à complexidade por unidade. O maior método identificado por *SonarQube* é o método responsável pela construção da cláusula *where* da consulta do filtro semântico. Sua complexidade decorre da análise que ele realiza do arquivo JSON de *template* SPARQL para inclusão dos trechos referentes aos parâmetros de filtros. Essa análise não é um processamento direto, pois são exigidos diversos fluxos condicionais para se verificar qual o tipo de dado do parâmetro que deve ser incluído na cláusula, a abertura e fechamento das chaves da cláusula, a presença ou ausência do operador UNION, dentre outras verificações necessárias.

Esse método, portanto, se enquadra na categoria de complexidade *moderada*. Como a maior parte dos métodos se enquadra na categoria *simples*, *Hermes Interpreter* é classificado como (++) nesse quesito.

**Teste de unidade**: quanto maior a cobertura de testes, mais manutenível o sistema. Heitlager et al. [21] definiram a classificação por teste de unidade de acordo com a Tabela 4.10:

**Tabela 4.10:** *Manutenibilidade por cobertura de testes de unidade[21]*

| Ranking | Cobertura de testes de unidade |
|---|---|
| ++ | 95-100% |
| + | 80-95% |
| o | 60-80% |
| - | 20-60% |
| - - | 0-20% |

Como não foram desenvolvidos testes automatizados para *Hermes Interpreter*, nessa propriedade, *Hermes Interpreter* está classificado como - -.

Coletadas as pontuações por propriedade do código-fonte, a visão geral da manutenibilidade de *Hermes Interpreter* pode ser observada na Tabela 4.11:

**Tabela 4.11:** *Manutenibilidade de Hermes Interpreter*

| | Volume | Complexidade por unidade | Duplicação | Tamanho da unidade | Teste de unidade | ***Hermes Interpreter*** |
|---|---|---|---|---|---|---|
| | ++ | ++ | ++ | ++ | - - | |
| Analisabilidade | x | | x | x | x | **+** |
| Capacidade de Alternância | | x | x | | | **++** |
| Estabilidade | | | | | x | **- -** |
| Testabilidade | | x | | x | x | **+** |

Ressalta-se que a análise do *SonarQube* e o modelo de Heitlager et al.[21] só consideram os arquivos fonte *.java*. Em *Hermes Interpreter*, os arquivos JSON desempenham funções críticas e indispensáveis, pois neles estão contidas as configurações de acesso aos tópicos (*topics.json*) e a interface com as ontologias de contexto (arquivos da camada *HI Filter Situation*). Ao analisar a Tabela 4.11, juntamente com as vantagens em se utilizar JSON, apresentadas na Subseção 2.3, tais como desacoplamento e legibilidade, pode-se considerar que *Hermes Interpreter* é um software manutenível.

## 4.5 Considerações Finais

Esse capítulo descreveu o componente *Hermes Interpreter* que incluiu os requisitos funcionais e não funcionais que ele deveria atender, o seu modelo arquitetural e os padrões tecnológicos escolhidos para implementar os requisitos, alguns cenários básicos de execução considerando o monitoramento de sinais vitais humanos e por último, a avaliação do componente quanto à manutenibilidade.

Para resumir como cada requisito foi implementado, será apresentada na Tabela 4.12 a relação entre o requisito e as soluções de projeto que o atendem. Para cada solução, se cabíveis, serão descritos os padrões de projeto implementados.

**Tabela 4.12:** *Relação entre requisitos e as soluções de projeto do Hermes Interpreter*

| Requisitos | Solução de projeto | Padrão de projeto |
|---|---|---|
| Reusabilidade | A arquitetura do componente é reutilizável quanto à modelagem e filtragem semânticas de contexto | Não se aplica |
| Independência de modelo de contexto | Os arquivos JSON da camada *HI Filter Situation* são as únicas interfaces do componente com o modelo de contexto | *Domain Object* |
| Suporte a múltiplas técnicas de interpretação de contexto | *HI Service* e subclasses | *Strategy* |
| Extensibilidade | Arquivos JSON da camada *HI Filter Situation* e camada *HI Service* | *Domain Object* e *Strategy*, respectivamente |
| Monitoramento e detecção de eventos | *HI Filter Service* | Não implementa padrão de projeto |
| Verificação de consistência entre fatos e modelo de contexto | *HI Service* | *Strategy* |
| Filtragem semântica de contexto | Camadas *HI Filter Service*, *HI Filter Situation* e *HI Filters* | *HI Filter Situation* implementa o padrão de projeto *Domain Object* e *HI Filters*, o padrão *POJO* |
| Manutenibilidade | *HI Facade*, *HI Transfer Object*, *HI Service Factory*, *HI Filter Situation Factory*, *HI Filter Situation*, *HI Communication Service* | *Facade*, *Transfer Object*, *Factory*, *Domain Object* e *Singleton* respectivamente |
| Flexibilidade | *HI Service Factory*, *HI Filter Situation Factory*, *Hermes Configurator*, arquivo *topics.json* | *HI Service Factory* e *HI Filter Situation Factory* implementam o padrão *Factory* |

CAPÍTULO 5

# Estudo de Caso

Este capítulo descreve o emprego do componente *Hermes Interpreter* em um cenário de monitoramento de sinais vitais humanos. A escolha desse cenário dá-se em função do mesmo contemplar situações de alto nível que precisam ser inferidas (ex: anormalidades de sinais vitais) e filtradas para informar assinantes (ex: enfermeiros) sobre eventos de seu interesse.

Nesse ínterim, destacam-se três objetivos principais com esse estudo de caso, segundo uma abordagem experimental:

- a validação funcional dos serviços de inferência e de filtragem oferecidos pelo HI
- uma avaliação do comportamento da filtragem semântica de eventos em função do tempo
- uma análise comparativa entre os tempos de filtragem e de inferência de contexto

Nas próximas seções, serão descritos os respectivos materiais e métodos utilizados e discussão dos resultados de cada experimento, para atender aos objetivos supracitados.

## 5.1 Configuração do Estudo de Caso

As informações de contexto do cenário de monitoramento de sinais vitais são modeladas em uma ontologia, representada na linguagem OWL, chamada MSVH [5]. Essa ontologia descreve as medidas dos sinais vitais com seus atributos e entidades relacionados, tais como parâmetros de normalidade, data e hora da coleta, perfis dos pacientes e dos profissionais que os assistem, dentre outras informações.

De forma complementar ao modelo ontológico citado, as anormalidades associadas a cada sinal vital (ex: taquicardia para o sinal vital de frequência de pulso) estão representadas na forma de regras na sintaxe SWRL.

Para efeitos de monitoramento de mudanças nesse estudo de caso, as principais questões a se considerar incluem: a atualização de medições de sinais vitais e a inferência

de anormalidades desses sinais. Ambas mudanças são acompanhadas do processo de filtragem semântica de eventos.

Os procedimentos adotados nesse estudo de caso estão fundamentados em entrevistas e reuniões presenciais com uma equipe de Enfermagem do Hospital das Clínicas da Universidade Federal de Goiás. Com o apoio desses profissionais, foi possível definir cenários de validação do HI, com diferentes relações enfermeiro-paciente, por exemplo.

O tipo de monitoramento sugerido pela equipe de Enfermagem foi o de assistência a pacientes de enfermaria e de UTIs por enfermeiros em períodos de 2 a 6 horas.

Como não se dispunha de sensores físicos para medição e, consequentemente, da integração desses com os componentes da infraestrutura *Hermes*, o autor deste trabalho realizou os experimentos na forma de simulação: Foram desenvolvidos programas que simulam os *Hermes Widgets*, um para cada tipo de sinal vital, que disparam notificações de leitura de sinais em intervalos de tempo relacionados à frequência de aferição do sinal vital, como será descrito adiante nessa seção. Para cada medida aferida de um sinal vital, é gerada uma *Thread* que é adicionada em um *pool de threads* para o respectivo paciente. Esse *pool de threads* executa cada *Thread* nos intervalos descritos no arquivo, simulando assim a frequência com que os sinais são aferidos e publicados na infraestrutura.

Para simular as *aplicações sensíveis a contexto*, também foram desenvolvidos programas que simulam o seu comportamento. Como cada instância do programa representa um profissional de saúde; esses programas efetuam a assinatura de sinais vitais e especificam os filtros de contexto que cada profissional deseja ser notificado.

Tanto os simuladores de *Hermes Widgets* quanto as *aplicações sensíveis a contexto* estão integradas ao componente *Hermes Interpreter* por meio do componente *Hermes Base* que provê a API para publicação e assinatura de tópicos na rede DDS.

## 5.2 Materiais do experimento

Essa simulação ocorreu através da leitura de sinais vitais reais coletados da base pública *PhysioBank* [14] [30], base essa que tem por finalidade fornecer informações reais para apoiar a pesquisa na área de informática médica. Essa base contém diversas outras fontes distintas de dados sobre particularidades de monitoramento, tais como sinais de arritmia ou medidas de pressão sanguínea durante o sono.

Para a simulação de *Hermes*, foram escolhidos os sinais vitais da fonte de dados MIMIC [1], uma das bases disponibilizadas pelo *PhysioBank*. Na base MIMIC, estão disponíveis dados gerais relativos à internação de pacientes em UTIs com a presença de todos os sinais vitais suportados pela ontologia MSVH. Para privacidade dos pacientes,

[1] http://physionet.org/mimic2/

esses são identificados por um valor alfa-numérico, como por exemplo, 243n. A opção escolhida para importação dos arquivos foi o formato CSV[2], em que cada arquivo se refere ao registro de um paciente.

Como nem todos os pacientes de MIMIC possuem registros de todos os sinais vitais de MSVH, foram escolhidos 15 pacientes que mais possuiam registros associados à ontologia. Os arquivos contêm registros de 12 horas de monitoramento com intervalo de um segundo entre as aferições. Para as avaliações, foram utilizados intervalos de até 8 horas com frequência de medição de 20 segundos. A escolha dessa frequência foi devido às capacidades de processamento das máquinas disponíveis para executar os *Hermes Widgets*.

Além da frequência de notificação, outras variáveis dessa simulação são a quantidade pacientes e o tempo de monitoramento. Portanto, para suportar os 15 pacientes escolhidos com testes de duração de até 8 horas, a frequência não poderia ser inferior à 20 segundos. Essa constatação se deu pela ocorrência de interrupção dos *Hermes Widgets* quando submetidos a parâmetros mais rígidos do que estes, como diminuição do tempo de frequência, aumento da quantidade pacientes ou aumento do tempo de monitoramento.

A Tabela 5.1 detalha os dados de dois pacientes. A amostra contém somente os dados relativos à frequência coletada por *Hermes*, que é de 20 segundos. A primeira coluna indica o tempo da aferição em segundos e as outras, os sinais vitais, a saber:

- *ABPMean*: Pressão sanguínea média
- *ABPsys*: Pressão sanguínea sistólica
- *ABPdias*: Pressão sanguínea diastólica
- *Pulse*: Frequência de pulso
- *Resp*: Frequência respiratória
- *SpO2*: Saturação de oxigênio

A segunda linha descreve a unidade de medida do respectivo sinal vital, e da terceira linha em diante, as medidas propriamente ditas.

Para fornecer os serviços de inferência e de filtragem semântica de eventos, o componente *Hermes Interpreter* utiliza as especificações da Web Semântica OWL, SWRL, RDF, RDFS e SPARQL. O *Hermes Interpreter* foi implementado na linguagem Java e fez uso do arcabouço Apache Jena para, dentre outros, manipular modelos de dados RDF e realizar inferências ontológicas, além de oferecer suporte à inferência por regras por meio da integração com o motor de inferência *Pellet*[3].

Os outros componentes de software de *Hermes* que foram executados na simulação foram *Hermes Base* e o *middleware CoreDX DDS*, desenvolvido pela empresa *Twin*

[2]Comma-separated values
[3]http://clarkparsia.com/pellet/

**Tabela 5.1:** *Amostras de registro da base MIMIC: pacientes 033n e 055n, respectivamente*

| ***Sample interval*** | **ABPMean** | **ABPsys** | **ABPdias** | **Pulse** | **Resp** | **SpO2** |
|---|---|---|---|---|---|---|
| sec | mmHg | mmHg | mmHg | bpm | bpm | % |
| 1 | 81 | 138 | 54 | 53 | 25 | 94 |
| 20 | 78 | 134 | 53 | 53 | 30 | 94 |
| 40 | 79 | 135 | 53 | 52 | 36 | 94 |
| 60 | 78 | 133 | 53 | 52 | 26 | 94 |
| ... | ... | ... | ... | ... | ... | ... |
| 1 | 89 | 114 | 78 | 101 | 15 | 100 |
| 20 | 89 | 114 | 78 | 100 | 15 | 99 |
| 40 | 89 | 114 | 78 | 101 | 15 | 99 |
| 60 | 89 | 114 | 78 | 100 | 15 | 99 |

*Oaks Computing*[4] que implementa a especificação de *middleware* DDS. Ambos foram exportados como arquivos no formato JAR e incluídos como bibliotecas em cada componente e aplicações anteriormente citados.

## 5.3 Validação funcional

Para a validação da funcionalidade dos serviços de inferência e de filtragem oferecidos pelo HI, foram construídos casos de teste, os quais estão listados no Apêndice B.

Em termos da configuração de hardware e software desse experimento, o HI foi executado em uma máquina Intel®Core(TM) i7-3537U com núcleos de velocidades 2GHz e 2,5GHz, memória RAM de 8GB, com SO Windows 8.1 de 64 bits. Os *Hermes Widgets* simulados executaram em um computador Intel®Core(TM) 2 Duo, de 2,2GHz cada, memória RAM de 4GB, com Windows 7 de 64 bits. Por fim, todas as aplicações simuladas foram executadas em uma máquina Intel Core i5 de velocidade 2,3 GHz com dois núcleos, memória RAM de 2GB, em máquina virtual Oracle VM VirtualBox6[5], que roda Windows XP de 32 bits.

Considerando as informações coletadas das reuniões com a equipe de Enfermagem do HC, a validação funcional consistiu em simular o monitoramento de 14 pacientes, em um intervalo de seis horas, assistidos por três enfermeiros, assim distribuídos: o "EnfermeiroA"e o "EnfermeiroB"assistem cinco pacientes cada, enquanto que o "EnfermeiroC", quatro pacientes.

Ainda segundo as reuniões com os profissionais de Saúde, estabeleceu-se que seriam utilizados três parâmetros para cada filtro criado no HI: (a) identificação do paciente ou do enfermeiro responsável (b) data e hora do monitoramento e (c) valor

[4] http://www.twinoakscomputing.com/coredx
[5] http://www.virtualbox.org/

ou anormalidade de sinal vital. A Tabela 5.2 descreve os tópicos assinados (e filtros associados) de cada instância do programa que simula os enfermeiros.

**Tabela 5.2:** *Filtros especificados na simulação*

| **Instância do programa** | **Tópico** | **Filtros** |
|---|---|---|
| 1 | Frequência de pulso | enfermeiro responsável: EnfermeiroA, anormalidades: bradicardia ou taquicardia e período entre 12h00 do dia 14/03/2015 e 17h do dia 14/03/2015 |
| 1 | Saturação de oxigênio | enfermeiro responsável: EnfermeiroA, valor de saturação de oxigênio < 90% e momento da coleta entre 12h do dia 14/03/2015 e 17h do dia 14/03/2015 |
| 1 | Pressão sanguínea | enfermeiro responsável: EnfermeiroA, anormalidades: hipotensão, hipertensão sistólica isolada, hipertensão estágio 1, hipertensão estágio 2 ou hipertensão estágio 3 e momento da coleta entre 12h do dia 14/03/2015 e 17h do dia 14/03/2015 |
| 2 | Frequência respiratória | enfermeiro responsável: EnfermeiroB, anormalidades: bradipneia, apneia ou taquipneia e momento da coleta entre 12h do dia 14/03/2015 e 17h do dia 14/03/2015 |
| 2 | Frequência de pulso | enfermeiro responsável: EnfermeiroB, anormalidades: bradicardia ou taquicardia e momento da coleta entre 12h do dia 14/03/2015 e 17h do dia 14/03/2015 |
| 2 | Saturação de oxigênio | enfermeiro responsável: EnfermeiroB, valor de saturação de oxigênio < 90% e momento da coleta entre 12h do dia 14/03/2015 e 17h do dia 14/03/2015 |
| 3 | Pressão sanguínea | enfermeiro responsável: EnfermeiroC, valor de pressão média < 70 mmHg e momento da coleta entre 12h do dia 14/03/2015 e 17h do dia 14/03/2015 |
| 3 | Pressão sanguínea | enfermeiro responsável: EnfermeiroC, valor de pressão média > 105 mmHg e momento da coleta entre 12h do dia 14/03/2015 e 17h do dia 14/03/2015 |
| 3 | Frequência de pulso | enfermeiro responsável: EnfermeiroC, valor de frequência de pulso < 60 bpm e momento da coleta entre 12h do dia 14/03/2015 e 17h do dia 14/03/2015 |
| 3 | Frequência de pulso | enfermeiro responsável: EnfermeiroC, valor de frequência de pulso > 100 bpm e momento da coleta entre 12h do dia 14/03/2015 e 17h do dia 14/03/2015 |
| 3 | Temperatura | enfermeiro responsável: EnfermeiroC, anormalidade: febre alta e momento da coleta entre 12h do dia 14/03/2015 e 17h do dia 14/03/2015 |

Ao relacionar os tipos de inferência aos tópicos assinados, apenas o tópico de saturação de oxigênio não exigia tipo de inferência baseada em regras, ou seja, requisitava o tipo de inferência baseada em ontologias. Após a assinatura dos tópicos de sinais vitais, a configuração do Hermes Interpreter ficou descrita conforme a Tabela 5.3:

Vale ressaltar que, embora todos os casos de teste descritos no Apêndice B tenham sido executados, por questões de relevância, serão detalhados a seguir apenas alguns cenários de dois assinantes enfermeiros. Os casos de teste escolhidos incluem a assinatura de tópico de notificação com filtro, caso de teste B.5; a publicação de contexto, caso de teste B.10 e a inferência de situações de contexto após alteração de modelo ontológico, caso de teste B.13.

**Tabela 5.3:** *Tópicos e tipos de inferência após assinatura das aplicações*

| Tópico | Tipo de Inferência |
|---|---|
| Frequência respiratória | Regras |
| Frequência de pulso | Regras |
| Pressão sanguínea | Regras |
| Saturação de oxigênio | Ontológica |
| Temperatura corpórea | Regras |

### 5.3.1 Assinatura de tópico de notificação com filtro

O objetivo desse caso de teste é validar a assinatura de um tópico com filtro, realizada pelo programa simulador do enfermeiro. A evidência é a geração de uma consulta SPARQL pelo componente HI. Como resultado, o assinante será notificado de situações de contexto que se adequem ao especificado em seu filtro, o que será comprovado na análise do caso de teste de publicação de contexto, na Subseção 5.3.2.

Segundo a Tabela 5.2, o enfermeiro 1 solicitou assinatura do tópico frequência de pulso; em seguida, o componente HI produz uma consulta SPARQL ilustrada nos Códigos 5.1, 5.2 e 5.3, com base nos parâmetros de filtro especificados[6].

**Código 5.1** Enfermeiro 1 - Consulta SPARQL, criada pelo HI, relativa ao filtro de frequência de pulso (parte 1 de 3)

```
construct {
        ?x rdf:type msvh:MonitoringPulseRate .
        ?x activity:hasParticipant ?n .
        ?n msvh:hasRole msvh:nurse .
        ?n actor:hasName "EnfermeiroA"^^string .
        ?x tEvent:startDateTime ?dateTimeEventMedida .
        ?dateTimeEventMedida rdf:type tEvent:InstantEvent .
        ?dateTimeEventMedida
        time:instantCalendarClockDataType ?instante .
        ?medida rdf:type ?sub_alarme .
        }
```

[6]Por questões de espaço, serão omitidas as importações dos prefixos ontológicos

**Código 5.2** Enfermeiro 1 - Consulta SPARQL, criada pelo HI, relativa ao filtro de frequência de pulso (parte 2 de 3)

```
where {
        ?x activity:hasParticipant ?n .
        ?n msvh:hasRole msvh:nurse .
        ?n actor:hasName "EnfermeiroA"^^string .
        ?x tEvent:startDateTime ?dateTimeEventMedida .
        ?dateTimeEventMedida rdf:type tEvent:InstantEvent .
        ?dateTimeEventMedida
        time:instantCalendarClockDataType ?instante .
        filter(?instante >=
        "2015-03-14T12:00:00"^^dateTime ) .
        ?x tEvent:startDateTime ?dateTimeEventMedida .
        ?dateTimeEventMedida rdf:type tEvent:InstantEvent .
        ?dateTimeEventMedida
        time:instantCalendarClockDataType ?instante .
        filter(?instante <=
        "2015-03-14T17:00:00"^^dateTime ) .
        {
                ?x rdf:type msvh:MonitoringPulseRate .
                ?x msvh:hasMonitoringPulseRate ?m .
                ?m msvh:hasMeasurementPulseRate ?medida .
                ?medida rdf:type msvh:TachycardiaAlarm .
                ?sub_alarme rdfs:subClassOf
                msvh:PulseRateAlarm .
                ?medida rdf:type ?sub_alarme .
        }
```

**Código 5.3** Enfermeiro 1 - Consulta SPARQL, criada pelo HI, relativa ao filtro de frequência de pulso (parte 3 de 3)

```
union  {
        ?x rdf:type msvh:MonitoringPulseRate .
        ?x activity:hasParticipant ?n .
        ?n msvh:hasRole msvh:nurse .
        ?n actor:hasName "EnfermeiroA"^^string .
        ?x tEvent:startDateTime ?dateTimeEventMedida .
        ?dateTimeEventMedida rdf:type tEvent:InstantEvent .
        ?dateTimeEventMedida
        time:instantCalendarClockDataType ?instante .
        filter(?instante >=
        "2015-03-14T12:00:00"^^dateTime ) .
        ?x tEvent:startDateTime ?dateTimeEventMedida .
        ?dateTimeEventMedida rdf:type tEvent:InstantEvent .
        ?dateTimeEventMedida
        time:instantCalendarClockDataType ?instante .
        filter(?instante <=
        "2015-03-14T17:00:00"^^dateTime ) .
        ?x msvh:hasMonitoringPulseRate ?m .
        ?m msvh:hasMeasurementPulseRate ?medida .
        ?medida rdf:type msvh:BradycardiaAlarm .
        ?sub_alarme rdfs:subClassOf msvh:PulseRateAlarm .
        ?medida rdf:type ?sub_alarme .
        }
}
```

Vale ressaltar que o HI associa um identificador único a cada filtro de cada programa assinante para que esse seja notificado posteriormente.

### 5.3.2 Publicar contexto

Esse caso de teste visa verificar o comportamento da infraestrutra com respeito aos sinais vitais publicados pelos *Hermes Widgets*. Para demonstração, será utilizada a mesma assinatura de tópico exibida no caso de teste anterior.

Em primeiro lugar, um *Hermes Widget* publica uma medida de frequência de pulso. Para o HI realizar as etapas de inferência e filtragem, todas as informações de perfil do paciente devem ser notificadas a cada sinal vital para que o HI tenha conhecimento do paciente cujo sinal vital foi publicado. Assim, um sinal vital de frequência de pulso do

paciente 033n publicado pelo *Hermes Widget* foi registrado da seguinte forma:[7]

```
HW.Sensor.Msg-> Vital Sign: VSO_0000030, Value: 53.000, for patient:
033n_VSO_0000030, broadcast on topic: VSO_0000030, at: Sat Mar 14
12:45:55 BRT 2015
```

O modelo RDF que está apresentado nos Códigos 5.4 e 5.5 refere-se à publicação de contexto descrita no trecho de log acima:

**Código 5.4** Modelo RDF publicado relativo à frequência de pulso do paciente 033n (parte 1 de 2)

```
msvh:radialPulseRate01
      a       owl:NamedIndividual ,
      <http://www.buffalo.edu/~ag33/VSO_0000032> ;
      OBO_REL:inheres_in msvh:radialArtery01 ;
      msvh:hasParameterizedValuePulseRate
              msvh:valuePulseRate01 ;
      msvh:isParameterizedValue
              "true"^^xsd:boolean .

msvh:mPulseRate01
      a       owl:NamedIndividual , msvh:MonitoringPulseRate ;
      activity:hasParticipant
              msvh:personEnfermeiroA , msvh:person033n ;
      activity:isPeriodicActivity
              "true"^^xsd:boolean ;
      tEvent:startDateTime
              msvh:dateTimeEventMedida ;
      msvh:hasMonitoringPulseRate
              msvh:radialPulseRate01 .
```

[7] Neste trecho de log do programa, VSO_0000030 refere-se ao nome do tópico, i.e. à classe frequência de pulso da ontologia MSVH.

**Código 5.5** Modelo RDF publicado relativo à frequência de pulso do paciente 033n (parte 2 de 2)

```
msvh:personEnfermeiroA
      a       msvh:Person ;
      actor:hasPersonName "EnfermeiroA"^^xsd:string ;
      msvh:hasRole
              msvh:nurse .

msvh:FreqPulso0
      a       msvh:VSO_0000030 ;
      msvh:isMeasurementPulseRate
              msvh:radialPulseRate01 ;
      msvh:unitPulseRate
              "bpm"^^xsd:string ;
      msvh:valuePulseRate
              "53"^^xsd:int .

msvh:dateTimeEventMedida
      a       tEvent:InstantEvent ;
      time:instantCalendarClockDataType
              "2015-03-14T12:46:55"^^xsd:dateTime .

msvh:person033n
      a       msvh:Person ;
      actor:hasPersonName "033n"^^xsd:string ;
      msvh:hasRole
              msvh:patient .

msvh:valuePulseRate01
      a       owl:NamedIndividual , msvh:PulseRate ;
      msvh:unitParameterizedValuePulseRate
              "bpm"^^xsd:string ;
      msvh:valueMaximumPulseRate
              "110"^^xsd:nonNegativeInteger ;
      msvh:valueMinimumPulseRate
              "70"^^xsd:nonNegativeInteger .
```

Em função do tópico "frequência de pulso"requerer inferência por regras (vide Tabela 5.3), após interpretar as regras relativas à medida de sinal vital (i.e. 53 bpm), o HI infere que o paciente encontra-se no estado de bradicardia.

Em seguida, o HI executa cada um dos filtros assinados para o respectivo tópico, por exemplo, o filtro apresentado nos Códigos 5.1, 5.2 e 5.3. Ao ser realizada uma consulta sobre o novo modelo de contexto inferido, o HI identifica um padrão de grafo que se adequa às necessidades parametrizadas pelo assinante, nesse caso, o enfermeiro responsável pelo paciente (EnfermeiroA), o tempo e a data de aferição da medida de frequência de pulso (entre 12h e 17h de 14/03/2015) e, por fim, que a medida de 53 bpm representa *bradicardia*.

O HI cria, segundo a consulta CONSTRUCT da SPARQL, um novo modelo de contexto contendo a especificação da subclasse *BradycardiaAlarm* para ser posteriormente notificado ao assinante. Dado que o modelo é semelhante ao apresentado nos Códigos 5.4 e 5.5, o Código 5.6 apresenta apenas as novas informações incorporadas ao modelo original.

**Código 5.6** Informação inferida para notificar assinante

```
msvh:FreqPulso0
      a       msvh:BradycardiaAlarm .
```

Para concluir a validação desse caso de teste, foi preciso verificar se a instância do programa do "EnfermeiroA"foi notificada da ocorrência do evento. O trecho de log desse programa é exibido a seguir, com destaque para a notificação de alarme de bradicardia para a medida de 53 bpm de frequência de pulso.

```
**** FILTRO ATENDIDO *** UUID -> 22063e12-8664-4cf1-a6d6-c7869f4f8563
Sinal Vital VSO_0000030 do paciente 033n_VSO_0000030 recebido às Sat
Mar 14 12:47:59 BRT 2015
Sinal Vital medido (valuePulseRate) -> 53 bpm
Tipo: VSO_0000030
Tipo: BradycardiaAlarm
-----------------------------------------------------------------
```

### 5.3.3 Inferência de contexto após alteração de modelo ontológico

Esse caso de teste visa validar a extensibilidade e a manutenibilidade do HI quanto a alterações no modelo de contexto. Para viabilizar essa validação, as alterações ontológicas devem possibilitar a criação de um novo parâmetro de filtro, ou mesmo impactar nos filtros já existentes.

A primeira alteração ontológica não foi motivada aleatoriamente, mas amparada por sugestão da equipe de enfermagem. A ontologia MSVH não suportava a medição *pressão arterial média*, útil para a tarefa diária de monitoramento de pressão sanguínea. Para atender à proposta, foi criada a propriedade *hasMeanBloodPressure* que teria como

conjunto domínio a classe *MonitoringBloodPressure* e como conjunto imagem, *valores do tipo inteiro*. Essa alteração poderia ser realizada antes da execução de HI, mas para validação do caso de teste, optou-se por incluí-la durante sua execução, para que não seja necessária a interrupção da execução do componente quando essas alterações ocorrerem.

Para efeitos dessa validação, o segundo tipo de alteração do modelo ontológico deveria impactar diretamente nos filtros já existentes. Optou-se então por alterar a descrição da propriedade *secom:hasName* da classe *secom:Actor* para *secom:hasPersonName*.

O procedimento de teste incluiu inicialmente a alteração da ontologia. Após a alteração, foram editados os arquivos JSON associados aos filtros impactados, tanto pela mudança da descrição da propriedade quanto pela inclusão do novo filtro para *pressão arterial média*.

Além da alteração nos arquivos de filtro, outra modificação no HI é informá-lo sobre a alteração da respectiva ontologia de contexto. Para isso, no arquivo *topics.json*, são especificadas as ontologias suportadas pelo componente e para cada uma, a informação *booleana atualizada*. O usuário administrador do HI, portanto, deve atribuir o valor *false* para essa propriedade de configuração para a ontologia recém-modificada.

A partir da alteração ontológica, *Hermes Widget* foi programado para publicar o contexto contendo a pressão arterial média, para os sinais vitais de pressão sanguínea, e a propriedade *secom:hasPersonName* para as instâncias da classe ontológica *secom:Person*.

Após a notificação de contexto pelos HWs, ao acessar o atributo *atualizada* para a ontologia corrente, o HI, antes de realizar a inferência das informações, refaz todos os filtros previamente criados, permitindo, de forma transparente, que todos os assinantes continuem a ser notificados sobre os dados de contexto, como mostra o trecho do filtro refeito do assinante "Enfermeiro C", no Código 5.7. Nessa consulta, além da filtragem por nome do enfermeiro, também parametriza-se pela informação de *pressão arterial média* para o tópico *pressão sanguínea*:

**Código 5.7** Filtro refeito para atender à alteração ontológica

```
construct { ...
        ?x msvh:hasMeanBloodPressure ?medida .
        ?n actor:hasPersonName "EnfermeiroC"^^string .
        ...
}
where {
        ...
        ?n actor:hasPersonName "EnfermeiroC"^^string .
        ?x msvh:hasMeanBloodPressure ?medida .
        filter(?medida >= "105"^^int ) .
}
```

O modelo de contexto produzido pela filtragem de HI, para uma notificação de *pressão sanguínea* posterior a essa alteração, é apresentado no Código 5.8:

**Código 5.8** Modelo de contexto para pressão sanguínea gerado por HI após alteração ontológica

```
msvh:mBloodPressure01
      a       msvh:MonitoringBloodPressure , owl:NamedIndividual ;
      activity:hasParticipant
              msvh:person226n , msvh:personEnfermeiroD ;
      tEvent:startDateTime
              msvh:dateTimeEventMedida ;
      msvh:hasMeanBloodPressure
              "106"^^xsd:int ;
      msvh:hasMonitoringBloodPressure
              msvh:bloodPressureSignal01 .

msvh:dateTimeEventMedida
      a       tEvent:InstantEvent ;
      time:instantCalendarClockDataType
              "2015-03-14T14:04:24"^^xsd:dateTime .

msvh:PresSang0
      a       msvh:VSO_0000005 ;
      msvh:isMeasurementBloodPressure
              msvh:bloodPressureSignal01 ;
      msvh:unitBloodPressure
              "mmHg"^^xsd:string ;
      msvh:valueDiastolicBloodPressure
              "84"^^xsd:int ;
      msvh:valueSystolicBloodPressure
              "155"^^xsd:int .
```

A validade da funcionalidade é comprovada pelas notificações de contexto que são realizadas aos assinantes após a alteração da ontologia. O trecho de log a seguir se refere à notificação do modelo apresentado no Código 5.8, em que, apesar de não exibir o dado de *pressão arterial média*, confirma tratar-se da mesma notificação pelos valores de pressão diastólica e sistólica.

```
**** FILTRO ATENDIDO *** UUID -> 8d6f98fe-4dbd-4946-8145-bef4e2dbacfc
Sinal Vital VSO_0000005 do paciente 226n_VSO_0000005 recebido às Sat
Mar 14 14:10:53 BRT 2015
Sinal Vital medido (valueSystolicBloodPressure) -> 155 mmHg
Sinal Vital medido (valueDiastolicBloodPressure) -> 84 mmHg
Tipo: VSO_0000005
-----------------------------------------------------------------
```

## 5.4 Avaliação de escalabilidade

Por meio desse teste, tem-se por objetivo analisar o comportamento do HI ao longo do tempo em função do acréscimo no número de assinantes por sinal vital, dentro de um tempo aceitável considerado pela literatura médica. Por assinantes, entende-se os filtros associados a cada tópico, i.e. uma aplicação sensível a contexto pode assinar filtros distintos para um mesmo sinal.

O referencial de tempo de resposta foi obtido do trabalho de Matthias et al. [29], que apontam soluções para diminiur a ocorrência de alarmes inefetivos ou ignorados, assim denominados, ou por serem falsos positivos, ou quando, após 5 minutos de sua ocorrência, nenhum profissional responsável pelo paciente ter feito um atendimento. A alta ocorrência desse tipo de alarme produz o que a literatura chama de fadiga de alarme. Esses mesmos pesquisadores classificaram em sua pesquisa 77% dos alarmes como inefetivos ou ignorados e concluíram que, se houver um atraso de até 19s antes da execução do alarme, 67% desses tipos de alarmes poderiam ser removidos.

*Hermes Interpreter* não objetiva solucionar o problema da *fadiga de alarme*, porém baseado nessa constatação, se conclui que um tempo de resposta de até 19s na notificação de um sinal vital é aceitável pela comunidade científica médica. Com base nesse referencial, foram realizados experimentos para explorar a capacidade de resposta de *Hermes Interpreter* quanto às etapas de inferência e filtragem de contexto.

### 5.4.1 Configuração dos experimentos

Estabeleceu-se que os filtros deveriam ter até três parâmetros, conforme apresentados na seção anterior, pois trata-se de uma quantidade condizente com as necessidades dos profissionais da saúde. O teste de escalabilidade pretende avaliar somente a capacidade do *Hermes Interpreter*, não considerando os tempos de processamento do contexto pelos *Hermes Widgets*, nem pelas aplicações, bem como o tempo de tráfego dos dados na rede DDS.

Conforme explicado na Subseção 4.3.5, os filtros suportados por HI são classificados em:

- FSI: Filtros sem inferência
- FCI: Filtros com inferência sem classes disjuntas
- FCID: Filtros com inferência com classes disjuntas

Determinou-se, assim, que o tópico (a) frequência de pulso receberia assinaturas com filtros do tipo FSI (b) frequência respiratória processaria filtros do tipo FCI e (c) pressão sanguínea receberia assinaturas de filtros do tipo FCID. A associação do tópico pressão sanguínea a filtros FCID dá-se porque, dentre os três sinais vitais mencionados, é o que possui maior quantidade de regras de anormalidades associadas, i.e. o que possui maior número de classes ontológicas disjuntas. Já para os filtros FSI e FCI, em função das similaridades da modelagem ontológica entre os sinais vitais frequência de pulso e frequência respiratória, não haveria diferença de complexidade entre os seus respectivos filtros e entre os modelos RDF de contexto sobre os quais esses filtros realizariam suas consultas.

Vale ressaltar também que nesse experimento não foram considerados os sinais vitais, temperatura corpórea e saturação de oxigênio em função da pequena quantidade de aferições desses sinais nos registros coletados da base MIMIC. A Tabela 5.4 apresenta um exemplo de filtro para cada tópico utilizado no experimento.

**Tabela 5.4:** *Exemplos de filtros por tópico*

| | **FSI** | **FCI** | **FCID** |
|---|---|---|---|
| Tópico | Frequência de pulso | Frequência respiratória | Pressão sanguínea |
| Parâmetro 1 | nome do paciente = "033n" | nome do paciente = "212n" | nome do paciente = "0408n" |
| Parâmetro 2 | período da coleta entre 01/03/2015 22h e 02/03/2015 6h | período da coleta entre 01/03/2015 22h e 02/03/2015 6h | período da coleta entre 01/03/2015 22h e 02/03/2015 6h |
| Parâmetro 3 | medida de pulso > 120 | medida seja referente à apnéia | medida seja referente à hipotensão ou hipertensão sistólica isolada ou hipertensão estágio 1 ou hipertensão estágio 2 ou hipertensão estágio 3 |

O passo seguinte foi o estabelecimento das quantidades de assinantes (ou filtros) para se avaliar o comportamento do HI no tratamento dos mesmos. Definiu-se por realizar três experimentos, em que o primeiro processaria 30 filtros por tópico, o segundo, 300 filtros por tópico e o terceiro, 3.000 filtros por tópico.

A escolha dessas quantidades foi definida de acordo com os cenários exemplificados pela equipe de enfermagem do HC/UFG durante as reuniões. Dessa forma, um

cenário típico em UTIs e enfermaria que contemple 30 filtros/sinal vital seria o monitoramento realizado por seis enfermeiros na proporção de um enfermeiro para cinco pacientes, em que cada enfermeiro assina, para cada paciente, um filtro para cada um dos sinais vitais abordados no experimento.

Conforme justificado na Seção 5.3, devido à capacidade das máquinas provedoras de contexto, a simulação foi realizada com 15 pacientes monitorados durante duas horas, com intervalos de 20 segundos entre as publicações dos sinais vitais. Assim, uma limitação do experimento é que parte dos filtros assinados pelos enfermeiros seria relativa a pacientes que não estavam sendo monitorados. Como o objetivo do experimento é avaliar o tempo dispensado para o processamento de inferência e filtragem do HI apenas, a ausência desses pacientes na simulação não interferiria no experimento.

Para a realização do experimento, foram utilizadas as seguintes máquinas por componentes:

- *Hermes Widgets*: Linux Ubuntu 12,4 LTS de 64 Bits, com processador Intel Core i5-3330 de 3GHz x 4 e memória RAM de 7,7 GB
- *Hermes Interpreter*: Windows 7 de 64 bits, com processador Intel(R) Core(TM) 2 Duo, de 2,2GHz cada e memória RAM de 4GB
- *Aplicações sensíveis a contexto*: Windows 7 de 32 bits, com processador Intel(R) Dual Core, de 1,73GHz e memória RAM de 1,5GB. Todas as instâncias de programas foram executadas somente nessa máquina

### 5.4.2 Análise dos resultados

Os resultados do experimento são visualizados nos gráficos seguintes. O primeiro, relativo ao tipo de filtro FSI, *filtros sem inferência*, foi efetuado sobre o tópico *frequência de pulso*. O resultado está exibido na Figura 5.1.

Os respectivos tempos médios e desvios padrão associados à inferência e filtragem, para cada quantidade de filtros foram: para 30 filtros, 123ms / 82ms, para 300 filtros, 953ms / 238ms e para 3.000 filtros, 9,6s / 2,36s.

Esses valores são, portanto, suficientes para atender à demanda da literatura médica de 19s, mesmo com os valores de desvio padrão. Pelo gráfico, se comprova o aumento linear de tempo de processamento com respeito ao aumento da quantidade de assinantes.

O crescimento linear também foi observado nos demais experimentos, relativos aos filtros FCI e FCID. A Figura 5.2 ilustra o resultado do experimento com o tópico frequência respiratória que contempla somente filtro de tipo FCI.

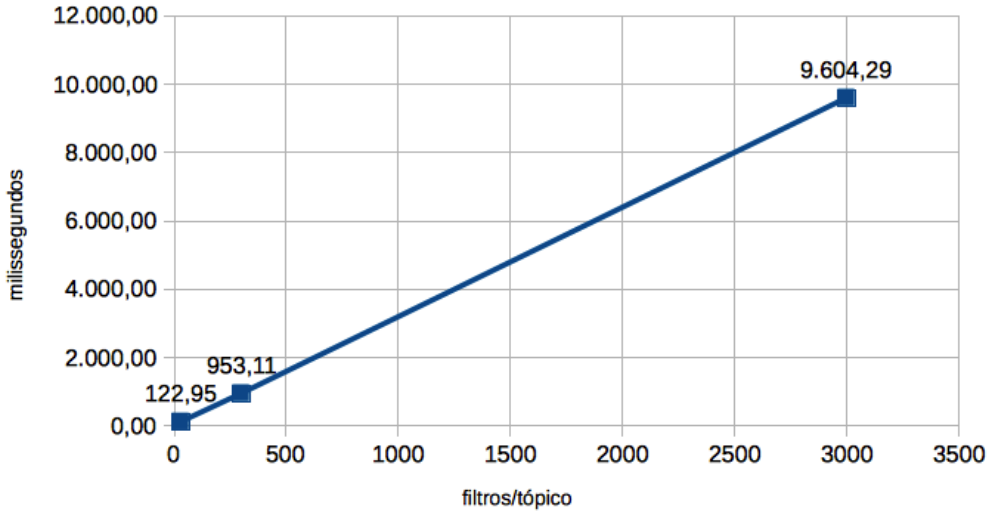


**Figura 5.1:** *Tempo médio de inferência com filtros FSI*

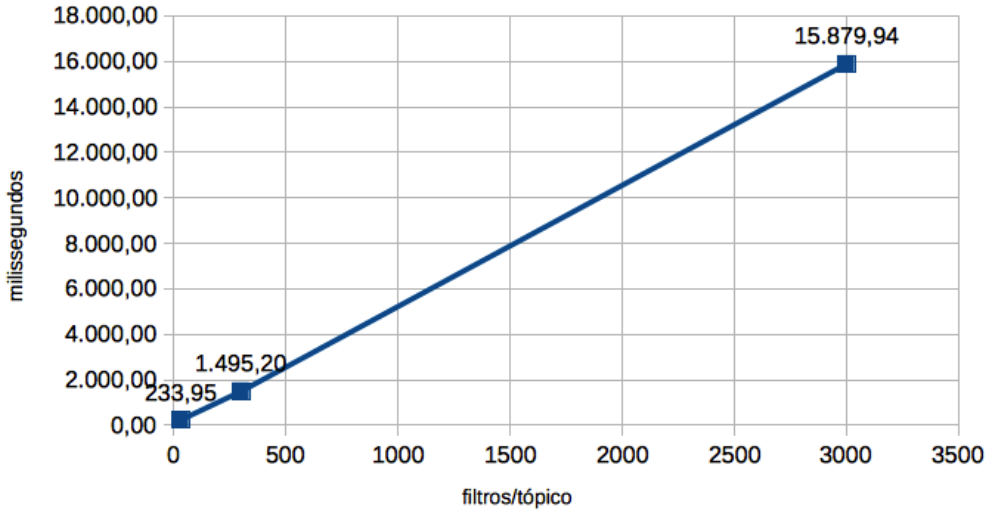


**Figura 5.2:** *Tempo médio de inferência com filtros FCI*

No caso da inferência com filtros FCI, os tempos médios e respectivos desvios padrão foram, respectivamente: para 30 filtros, 234ms / 266ms, para 300 filtros, 1,5s / 1,65s e para 3.000 filtros, 15,8s / 16,6s.

O melhor desempenho dos filtros do tipo FSI em comparação aos do tipo FCI é devido ao menor conjunto de dados sobre os quais a consulta é realizada. No momento em que os filtros FSI são executados, o HI ainda não realizou a interpretação do contexto e o modelo de contexto consultado ainda é o publicado originalmente pelos *Hermes Widgets*.

Após a etapa de interpretação, são acrescidos novos fatos de contexto ao modelo RDF original, aumentando-o consideravelmente. É sobre esse novo modelo que são

executados os filtros FCI e FCID, o que contribui para o aumento do tempo de pesquisa.

A provável causa do alto desvio padrão nos filtros FCI deve-se à variação de tempo para identificar o padrão de grafo requerido pelo filtro. Os filtros FCI foram construídos de tal forma que cada uma das três anormalidades de frequência respiratória fossem igualmente distribuídas entre as consultas. Assim, para cada aferição de frequência respiratória, ora a consulta SPARQL é rapidamente satisfeita, ora não o é, obrigando o processador da consulta percorrer todo o grafo.

O resultado da execução dos filtros FCID está ilustrado na Figura 5.3.

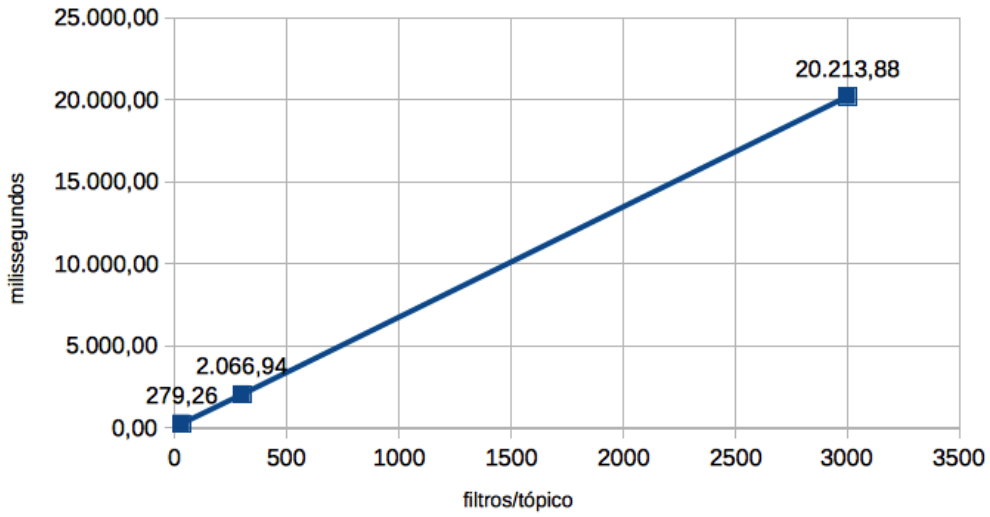


**Figura 5.3:** *Tempo médio de inferência com filtros FCID*

O comportamento linear também é observado para a execução dos filtros FCID. O tempo médio de processamento para 30 filtros foi de 279,26ms, com desvio padrão de 89,78ms. Para a execução de 300 filtros/tópico, o tempo médio observado foi de 2s, com desvio padrão de 564,5ms. Por fim, para 3.000 assinantes por tópico, foi de 20,2s, com desvio padrão de aproximadamente 5s.

O aumento médio do tempo de processamento, comparado ao experimento com o tipo de filtro FCI, é devido à presença das cláusulas UNION nos filtros FCID. Essa cláusula obriga o *processador SPARQL* a sempre percorrer todo o modelo RDF inferido até identificar todos os padrões de grafo compatíveis com cada cláusula UNION.

No caso de teste realizado, a consulta para o sinal vital pressão sanguínea contém cinco padrões de grafos, um referente a cada tipo de anormalidade que a medida pode ter. Como as classes de anormalidades são disjuntas entre si, o processador da consulta sempre percorrerá todo o grafo, pois, no máximo, somente um padrão de grafo será identificado. Em virtude disso, como a execução das consultas FCIDs obrigatoriamente

sempre percorrerão toda a fonte RDF de dados, o desvio padrão nos filtros FCID não é tão alto como nos filtros FCI, que ora percorre todo o grafo, ora não o faz.

Os experimentos demonstraram também que somente a execução do HI com 3.000 filtros para o tópico *pressão sanguínea* não atendeu ao requisito de tempo máximo aceitável para notificação de alarmes de pacientes em UTIs e enfermarias, pois essa configuração atingiu tempo médio de 20,21s, ultrapassando os 19s estabelecidos por Matthias et al. [29].

Sobre o aumento linear do tempo de processamento para os experimentos com os três tipos de filtros, a justificativa é que não houve aumento do tamanho da entrada para cada filtragem de contexto, ou seja, a fonte de dados RDF e a consulta SPARQL para cada tipo de filtro são sempre os mesmos, variando, apenas, os valores dos parâmetros filtrados. Essa característica não revela uma limitação do HI, mas representa um cenário factível, pois como comprovado na validação funcional, essa abordagem atendeu às necessidades de monitoramento para a equipe de enfermagem.

O componente *Hermes Interpreter*, a cada notificação de contexto, para os filtros FSI, realiza a consulta sobre o contexto publicado. Para os filtros FCI e FCID, como explicado no Capítulo 4, o componente descarta o contexto anterior do modelo RDF inferido e adiciona o novo modelo publicado. Assim, as consultas são sempre realizadas sobre o modelo RDF que contém o *schema* ontológico de MSVH juntamente com o modelo RDF recém-notificado.

Se a fonte de dados RDF fosse incrementada para cada notificação de contexto, de acordo com Pérez et al. [35] e Picalausa et al. [36], os filtros FSI e FCI continuariam sendo executados em tempo polinomial. Porém, os filtros FCID aumentariam seu tempo de execução exponencialmente, de acordo com o tamanho do modelo RDF que fosse sendo incrementado.

Pérez et al. [35] e Picalausa et al. [36] demonstram que consultas SPARQL, com operadores AND e FILTER somente, as quais são utilizadas nos filtros FSI e FCI, são polinomialmente executáveis. Por outro lado, as consultas que utilizam cláusulas UNION e operador AND, como as construídas para os filtros FCID, possuem complexidade exponencial relativas ao tamanho da entrada, sendo classificadas como problemas *NP-Completos*.

Por essas constatações, a quantidade máxima de assinantes para atender aos requisitos da enfermagem seria menor se o componente implementasse uma base de conhecimento. Nesse cenário, o volume de dados é crescente a cada publicação de contexto e o tempo de resposta consequentemente seria maior. Devido à complexidade exponencial dos filtros FCID, a quantidade desses seria mais influenciada do que a quantidade dos filtros FSI e FCI. Nesse novo cenário, seriam necessários, portanto, outros experimentos para avaliar a quantidade máxima de assinantes que *Hermes Interpreter*

suportaria para cada tipo de filtro.

## 5.5 Comparativo de variáveis de inferência e filtragem

Para medir e comparar as diversas etapas realizadas durante os processos de inferência e filtragem, utilizou-se como referência o trabalho de Bulcão Neto et al. [31]. Esse trabalho extrai as diversas etapas de processamento presentes em uma aplicação semântica e expõe o tempo percentual despendido de cada uma delas, baseado na complexidade dos modelos ontológicos acessados.

Dentre as variáveis contempladas na pesquisa de Bulcão Neto et al. [31], as seguintes foram relacionadas para esse comparativo:

- MLT (*Model loading time*): tempo de carregamento do modelo ontológico
- ST (*Satisfability time*): tempo para validação da consistência da ontologia, a fim de detectar fatos contraditórios
- CT (*Classification time*): tempo para identificar as relações entre as subclasses
- RT (*Reasoning time*): tempo para especificar as classes mais específicas das instâncias de contexto

Como o componente *Hermes Interpreter*, além de inferência, também realiza atividades de consulta a modelos RDF, este trabalho acrescenta as variáveis FSI (*filtros sem inferência*), FCI (*filtros com inferência sem classes disjuntas*) e FCID (*filtros com inferência com classes disjuntas*).

O experimento descrito nessa seção, portanto, objetiva identificar a relação entre as variáveis de inferência [31] e as associadas aos filtros. Os tempos totais médios de cada experimento não consideram outras atividades do HI, como escrita em arquivo de log ou comunicação com o componente *Hermes Base*, por exemplo.

A configuração do experimento contemplou 30 filtros por tópico, por ser o cenário mais próximo do real. As demais configurações foram as mesmas das avaliações de escalabilidade.

O primeiro gráfico, na Figura 5.4, revela o percentual dispensado para a execução dos 30 filtros FSI em comparação com as variáveis de inferência.[8]

Na Figura 5.5, observa-se o aumento desse percentual se comparado com os filtros FSI, confirmando a comparação realizada na seção anterior, entre os filtros FSI e FCI.

[8] As legendas contêm o tempo médio total para cada atividade e o respectivo desvio padrão.

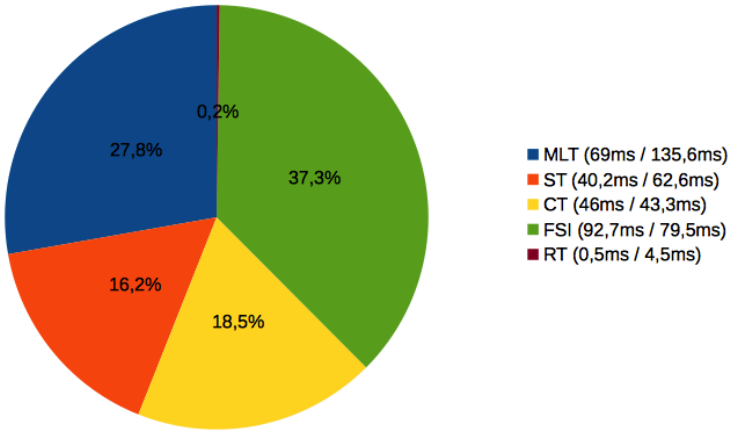


**Figura 5.4:** *Comparativo entre variáveis com filtros FSI para 30 filtros por tópico*

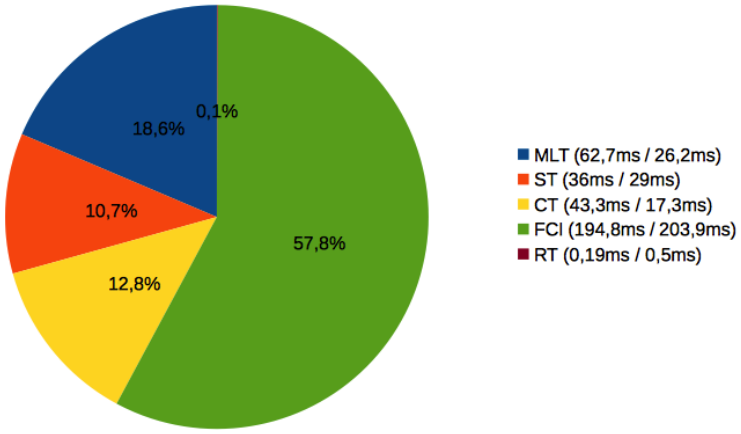


**Figura 5.5:** *Comparativo entre variáveis com filtros FCI para 30 filtros por tópico*

Por último, na Figura 5.6, verifica-se o maior percentual dispensado para o processamento dos filtros FCID, resultantes da presença das cláusulas UNION em sua estrutura.

Independente do tipo de filtro, foi possível perceber pela execução desse experimento o quanto a atividade de consulta em *Hermes Interpreter* tem maior custo do que as atividades de inferência, considerando o cenário real de 30 filtros por tópico. Conforme destacado na seção anterior, esses percentuais poderiam ser maiores se a fonte de dados RDF fosse sendo incrementada a cada notificação.

## 5.6 Considerações finais

Esse capítulo apresentou a aplicação do componente *Hermes Interpreter* em um cenário de monitoramento de sinais vitais de pacientes. Os experimentos, apesar de

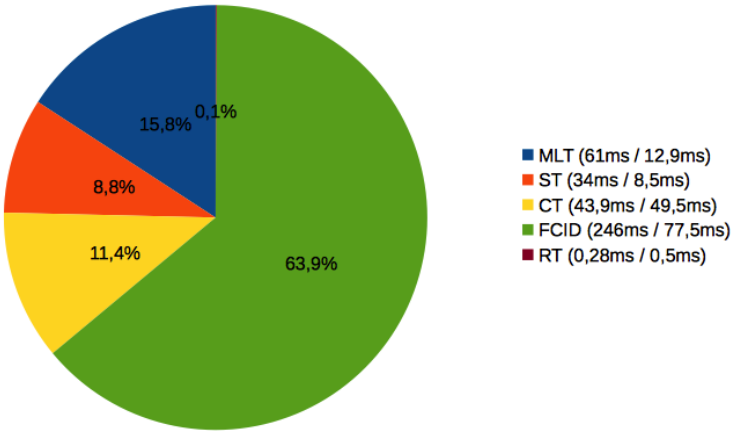


**Figura 5.6:** *Comparativo entre variáveis com filtros FCID para 30 filtros por tópico*

simulações, fundamentaram-se em critérios reais relativos às atividades assistenciais a pacientes realizadas por enfermeiros em UTIs e enfermarias.

Primeiramente, foi realizada validação funcional em que se comprovaram a corretude dos componentes envolvidos e a utilidade dos mesmos para auxiliar os profissionais da saúde em suas atividades diárias de monitoramento.

Na sequência, foi realizada avaliação de escalabilidade objetivando identificar o tempo de processamento do HI para três tipos de filtros: FSI, FCI e FCID, com diferentes quantidade de assinantes por tópicos. Os filtros FSI foram os que tiveram melhor desempenho, pois eram executados sobre o modelo RDF publicado originalmente. Os filtros FCI e FCID foram executados sobre o modelo RDF que contém os fatos inferidos. Dentre esses, os filtros do tipo FCID tiveram pior desempenho, devido à presença das cláusulas UNION em suas estruturas. Para todos os tipos de filtros, a cada acréscimo de assinantes, houve apenas aumento linear do tempo de processamento, resultado da não variação do tamanho da fonte de dados entre as consultas. De acordo com a literatura, para os filtros FCID, esse resultado teria sido exponencial, se houvesse incremento do tamanho da fonte de dados a cada consulta. Essa consideração, entretanto, não caracterizou uma limitação do componente, pois, funcionalmente, atendeu às necessidades a que estava proposto.

O último experimento visou comparar os tempos de inferência e filtragem pelo componente HI. Para isso, foi utilizada a configuração semelhante aos dos testes de escalabilidade com aplicação de 30 filtros por tópico, cenário mais próximo da realidade do HC/UFG. Foi observado um considerável aumento no percentual de tempo de execução dos filtros, em comparação às atividades de inferência. Por essa proporção, comprova-se o impacto da atividade de consulta a modelos RDF em sistemas sensíveis a contexto, o que aponta para um campo vasto de pesquisas para melhoria do desempenho dessa etapa.

CAPÍTULO 6

# Trabalhos Relacionados

Esse capítulo descreve os trabalhos relacionados ao *Hermes Interpreter*. Como na literatura de computação sensível a contexto, em geral, os trabalhos contemplam toda a infraestrutura e não somente o módulo de interpretação de contexto, elas serão primeiramente apresentadas e posteriormente será feita comparação dos respectivos módulos de interpretação com o componente *Hermes Interpreter*. A análise dos trabalhos relacionados está baseada nos requisitos atendidos por *Hermes Base* e *Hermes Interpreter*, apresentados nos Capítulos 3 e 4, respectivamente.

Para comparação com o presente trabalho, foram considerados softwares em geral, tais como, *middlewares*, arcabouços, arquiteturas e infraestruturas, que apoiam o desenvolvimento de aplicações sensíveis a contexto e o gerenciamento de contexto. Na sequência, os trabalhos relacionados serão brevemente apresentados e comparados com as principais características de *Hermes Interpreter*.

## 6.1 Teymourian et al.

Teymourian et al. [46] apresentam um modelo arquitetural para ser utilizado por sistemas orientados a eventos semânticos e um modelo ontológico para descrever tais eventos, associando-os com outras camadas ontológicas, tais como as de domínio, aplicações, atividades, tempo, situações, dentre outras que são necessárias para auxiliar no processo de detecção de eventos.

Conforme Teymourian et al. [46], os eventos não são somente estruturas simples e estáticas, mas também complexas, originadas da combinação de vários eventos simples, os quais podem ser representados ontologicamente por meio de hierarquias, como especializações e generalizações ou outras formas de relacionamento entre eventos que possam ser úteis para identificá-los. Outra maneira de detectar esses eventos complexos é através de regras previamente declaradas.

A visão arquitetural para processamento dos eventos complexos combina informações persistidas em base de conhecimento que incluem ontologias e regras, juntamente com eventos produzidos por provedores de contexto. O componente principal da arqui-

tetura é o *processador de eventos complexos*, cujas atividades incluem coletar os eventos simples e inferir novos eventos. Os novos eventos são gerados a partir dos relacionamentos semânticos que o evento simples original tem com outras instâncias de eventos e até mesmo com instâncias de conceitos que não são diretamente associados aos eventos, tais como indivíduos do domínio ou aplicações. Assim, proativamente, o sistema identifica eventos que são de relevância para aquele contexto.

Para ilustrar as capacidades desse modelo conceitual proposto, ele é brevemente exemplificado no cenário de monitoramento de pacientes. Nesse cenário, os eventos, tais como, temperatura corpórea, pressão sanguínea e movimentos musculares, são coletados por sensores conectados ao corpo do paciente. Esses eventos podem ser processados como eventos complexos nos respectivos contextos em que são utilizados, resultando em importantes fatores para tomada de decisões. Tais eventos complexos poderiam ser referentes a um paciente em situação de emergência ou necessidade de medicamentos. Outros eventos complexos podem ser identificados a partir desse, resultado do processamento de informações específicas do paciente que estão armazenadas na base de conhecimento, tais como alergias ou resistência a medicamentos. Notificada dessas informações adicionais, a equipe médica pode tomar decisões apropriadas para o tratamento do referido paciente.

Tanto Teymourian et al. [46] quanto *Hermes Interpreter* viabilizam a detecção de eventos por meio de inferências ontológica e baseada em regras. Ambos também realizam filtragem semântica de eventos de contexto para seleção do que é relevante para notificação.

Entretanto, se diferenciam quanto à origem dos eventos notificáveis: em Teymourian et al. [46], os eventos, simples ou complexos, são previamente gerados e descritos em regras e na camada *evento* da ontologia proposta, de forma tal que, quando um novo contexto, ou evento simples, é acessado, após as etapas de inferência e filtragem, são identificadas as instâncias de eventos complexos geradas e que serão notificadas. Trata-se, portanto, de uma abordagem *proativa* com respeito à notificação de eventos, ou seja, esses eventos já estavam previamente descritos na ontologia *evento* utilizada pelo sistema.

Em *Hermes Intepreter*, por outro lado, os eventos complexos notificáveis são originários das parametrizações de contexto fornecidas pelas aplicações. Esses eventos se referem às situações de contexto das quais essas aplicações necessitam ser notificadas, sendo, portanto, uma abordagem *reativa* desse problema, adequada para cenários de alta dinamicidade e de difícil previsão.

São, dessa forma, implementações complementares, cujas respectivas relevâncias serão determinadas pelo cenário de uso. Retomando o exemplo da sensibilidade ao contexto na área da saúde, a abordagem proativa auxilia os profissionais nas tomadas de decisões, enquanto a reativa, alerta-os sobre as situações de seus interesses em determinados cenários.

Outra diferença entre os trabalhos é que, apesar de citar dois exemplos de aplicação da sua arquitetura, Teymourian et al. não a avaliam quanto a sua capacidade de desempenho, visando atender às exigências de tempo de resposta dos cenários propostos.

## 6.2 EXEHDA-UC

EXEHDA-UC [26] é uma arquitetura distribuída que provê gerenciamento de contexto e fornece serviços básicos a aplicações sensíveis a contexto, móveis e *web*. O componente que realiza inferência de contexto é o *Interpreter*, integrante do módulo *Context Server*, e se assemelha ao *Hermes Interpreter* no quesito *Monitoramento e Detecção de Eventos*. Ambas soluções possuem módulos específicos para efetuarem as tarefas de comunicação, detecção de eventos e notificação. Enquanto EXEHDA-UC apresenta somente a técnica de inferência baseada em regras, *Hermes Interpreter* suporta inferência ontológica, devido à modelagem semântica suportada por sua arquitetura e, também, a técnica baseada em regras.

Além disso, EXEHDA-UC não propõe solução arquitetural para contemplar outros mecanismos de inferência. EXEHDA-UC oferece aos usuários interface para parametrizar regras de contexto de interesse, com a finalidade de cruzar informações de diferentes sensores. A diferença dessa funcionalidade para a filtragem de contexto provida por *Hermes Interpreter* é que em EXEHDA-UC os parâmetros são comparados apenas por operadores de comparação como >, >=, <, <=, = *ou !*=, enquanto que no *Hermes Interpreter*, além de operadores de comparação, são suportadas também as propriedades definidas na ontologia, ampliando a capacidade de filtragem de contexto.

O trabalho é avaliado por meio da abordagem TAM (*Technology Acceptance Model*), em que se verifica a melhoria que a tecnologia oferece aos usuários na realização de suas atividades. *Hermes Interpreter*, por outro lado, não pode ser instanciado em cenário real de uso, apenas simulado, razão pela qual não foi possível fazer uma avaliação como essa. Porém, essa simulação avaliou o componente quanto a sua capacidade de atender aos usuários em tempo aceitável para desempenho de suas atividades.

## 6.3 TrailM

Em Martins et al. [28], é apresentado o modelo *TrailM* para gerenciamento de preferências comerciais de usuários em ambientes comerciais ubíquos. O módulo responsável pela inferência do contexto é o *Opportunity Consultant* que toma decisões a partir de consultas a informações armazenadas em banco de dados as quais se referem aos perfis comerciais dos usuários e de suas localidades percorridas. A inferência baseada em ontologia implementada no *Hermes Interpreter* é mais robusta devido a seu poder

de raciocínio a partir de informações implícitas, enquanto que em banco de dados, a informação deve estar explicitamente registrada. Essa característica permite que novos conhecimentos possam ser produzidos em maior quantidade do que a partir de um modelo relacional.

A disseminação do contexto em TrailM ocorre levando-se em consideração os trechos percorridos pelos usuários negociadores. Para isso, o sistema realiza o cruzamento das informações de vendedores e compradores de produtos em comum que compartilham os mesmos espaços físicos, tais como lojas ou estações de trem. Esses locais são registrados automaticamente em seus respectivos perfis de locais preferidos, ou mais comparecidos.

A proposta de filtragem de contexto apresentada por TrailM contempla dados geográficos e o item de compra a ser negociado. Contudo, não é um mecanismo parametrizável e extensível quanto *Hermes Intepreter*, tanto do ponto de vista do acréscimo de novos filtros, resultante de alterações no domínio do contexto, quanto das possibilidades que oferece aos usuários de especificarem os eventos que desejam ser notificados. Além disso, a filtragem não está alinhada à semântica das informações de contexto, o que limita capacidade de seleção da informação frente a cenários complexos que exigem inferência de situações de mais alto nível, como os observados no monitoramento de anormalidades de sinais vitais.

Por estar em fase inicial na época da publicação do artigo, TrailM assim como o *Hermes Intepreter*, foram validados em cenário de simulação, mas contendo dados reais sobre as localizações geográficas dos trechos. Todavia, não foi avaliado quanto ao desempenho, como *Hermes Interpreter* o foi em cenário simulado. Porém, Martins et al. consideram realizar essa avaliação futuramente.

## 6.4 E-health

Em Guermah et al. [16], é proposta uma arquitetura baseada em ontologia para o desenvolvimento de serviços sensíveis a contexto, com aplicação em um sistema de saúde nomeado *E-health*. Dentre as contribuições do trabalho, está o componente de inferência *Context Reasoning Engine* que é composto por três módulos: *Reasoner*, *Adaptation Inference Rule* e *Knowledge Base*. O módulo *Adaptation Inference Rule* possibilita que regras sejam construídas utilizando dados de sensores, dados persistidos ou derivados de ambos. Além disso, a arquitetura não demonstrou ser extensível para incluir novas técnicas de inferência.

As regras que compõem o *E-health* se assemelham com as regras para detecção de anormalidades dos sinais vitais registradas em SWRL de *Hermes*. E, assim como *Hermes*, as regras podem ser criadas e editadas em tempo de execução de forma desacoplada

do código-fonte. A diferença é que *Hermes Interpreter* possibilita que além das regras, que em geral apresentam alta granulosidade, os usuários podem especificar os dados que desejam ser informados como intervalo de idade ou médico responsável, à parte das regras pré-definidas. No sistema *E-health*, não há processamento de filtros de dados à parte das regras. Além disso, a pesquisa não avalia a capacidade de desempenho do *E-health* em atender aos seus usuários em tempo aceitável com o monitoramento de sinais vitais, como foi realizado na simulação de *Hermes* deste trabalho.

## 6.5 Park et al.

Park et al. [32] apresentam solução para criação de rede de relacionamentos, utilizando informações de dispositivos móveis, tais como dados de sensores e *logs* de utilização. A partir dessas informações, a aplicação consegue inferir situações contextuais de alto nível, como reconhecimento de atividades, emoções e nível de relacionamentos entre usuários, como amigos próximos, colegas, conhecidos de trabalho e outros. A inferência é fundamentada em *Redes Bayesianas* [23], apropriada para contextos imprecisos como os produzidos por dispositivos móveis.

Arquiteturalmente, o sistema contém uma camada cliente que contempla um *browser map* para exibir localização de usuários e uma lista de usuários relacionados, chamada *Context Viewer*, cujas informações de usuários compartilhadas estão associadas ao grau de proximidade, inferido pelo sistema, com o usuário da aplicação.

A inferência é implementada no módulo *Context Recognizer*, porém o trabalho não descreve detalhes de sua arquitetura interna. Por isso, não é possível comprovar a possibilidade de extensão do sistema para suportar outros mecanismos de inferência. O sistema filtra os dados registrados no aparelho para realizar as inferências, porém não há possibilidade de os usuários selecionarem os dados de contexto que desejam ser notificados.

Park et al. [32] submeteram o sistema à avaliação de usabilidade por meio do questionário SUS (*System Usability Scale*). *Hermes Intepreter* foi executado em cenário simulado de acordo com demandas reais apresentadas por profissionais em potencial do sistema. Visando avaliar o sucesso do componente para atender aos requisitos temporais exigidos, também foi realizada uma validação de desempenho para verificar o grau de escalabilidade da solução.

## 6.6 DOHA

O *middleware DOHA - Dynamic Open Home-Automation*, desenvolvido por Rodríguez et al. [39], tem como finalidade apoiar o desenvolvimento de aplicações

sensíveis a contexto e prover o gerenciamento de contexto por meio de uma camada de software que utiliza ontologia genérica para modelar a informação de contexto, independente da aplicação. A informação de contexto é subdividida em três categorias: modelo de serviço, modelo de espaço e modelo de usuário. O primeiro descreve e classifica os elementos que serão representados como serviços e processados pelo sistema. O segundo descreve o espaço físico ao qual o sistema está sendo executado e localiza o modelo de serviço nesse espaço. O último, modelo de usuário, descreve o conhecimento do usuário e preferências.

Para tratamento da informação de contexto, o *middleware* trabalha com os conceitos de *perfil de dado* e processamento de *Qualidade de Dado* que visa adequar as informações contextuais para que sejam melhor entregues às aplicações. Os perfis de dados são serviço, processo, entidade, elemento de informação ou requisitos. O *perfil de dado* é modelado em ontologias que, no código de DOHA, é representado por classes. Essa modelagem permite um desacoplamento entre o modelo do *middleware* e das aplicações sensíveis a contexto as quais se comunicam com ele. Assemelha-se ao *Hermes Interpreter* por inferir o contexto baseado em ontologia e regras. Entretanto, *DOHA* não apresentou solução para processamento de dados de contexto por meio de filtragem.

DOHA foi avaliado em contexto de saúde, indústria e automação doméstica, porém, diferente de *Hermes Intepreter*, essa avaliação não se propôs a evidenciar o comportamento do *middleware* quanto ao atendimento a possíveis requisitos temporais exigidos por esses cenários.

## 6.7 COPAL

Outra solução para computação sensível a contexto é *COPAL* (COntext Provisioning for ALL), apresentado por Fei et al. [25]. Trata-se de um *middleware* e *framework* para provisão de dados de contexto. Assim como *Hermes*, sua arquitetura é componentizada para permitir a implantação de novas fontes de contexto, criação de novos modelos de contexto e suporte a filtragem de contexto. O *middleware* faz parte do projeto *SM4ALL*[1] e possibilita a comunicação entre fontes de contexto, dispositivos de usuários e serviços de contexto em ambientes domésticos inteligentes. Os serviços de contexto são *Listeners* os quais COPAL notifica caso algum filtro previamente definido por eles tenha sido atendido pelos dados do contexto.

Por ser baseado no paradigma *publish/subscribe*, a sua organização de classes e relacionamentos é semelhante ao DDS, *Hermes Base* e *Hermes Interpreter*: um *ContextEvent*, por exemplo, é um tópico que está associado a um *ContextType* específico. Para

[1] http://www.sm4all-project.eu

cada *ContextEvent*, podem ser definidos vários *Listeners* que estão associados a um *ContextQuery*. Portanto, a cada evento notificado por *ContextEvent*, as consultas dos objetos *ContextQueries* são executadas e para as que forem atendidas, os respectivos *Listeners* são acionados. Como o contexto de COPAL não é modelado semanticamente, as consultas geradas pelos *ContextQueries* são no formato EPL - *Event Processing Language* que é análogo à linguagem SQL.

O texto não explicita se o *middleware* fornece mecanismos para que as consultas dos *Listeners* sejam geradas de forma desacoplada do código-fonte. Também não é esclarecida a técnica de inferência que COPAL utiliza para gerar as informações de contexto de alto nível.

COPAL é instanciado em serviço doméstico, porém sem uma análise de desempenho que demonstre a capacidade de atender aos seus assinantes de contexto.

## 6.8 Feel@Home

O projeto *Feel@Home* [18] apresenta uma arquitetura para gerencimento de contexto em domínios locais, como LANS, como também para diferentes domínios de alcance, como WANS. O projeto é destinado a soluções domésticas inteligentes e sensíveis a contexto, cujos consumidores e provedores de contexto podem ser locais ou remotos. Para atender a esses requisitos, são considerados três gerenciadores de contexto: *Home context manager*, *Office context manager* e *Mobile context manager*.

O mecanismo de consulta das aplicações requisitantes de contexto é influenciado pela diversidade de consumidores e provedores de contexto que podem ser internos ou externos. Se local, a consulta é explícita para algum provedor de contexto, tal como informações sobre o *nível de luz em um quarto da casa*. Se remota, como *localização de um usuário fora da casa*, utiliza-se um mecanismo transparente de roteamento da consulta para o correto gerenciador de contexto.

A arquitetura do sistema é semelhante a de *Hermes*: Ambas possuem contexto modelado por ontologia e possuem alguns componentes internos com finalidades parecidas, tais como os *context wrappers*, que são componentes associados aos sensores e possuem funcionalidades parecidas com os *Hermes Widgets*. Possui também os *context aggregators*, que são componentes resultantes da associação de vários *context wrappers*, semelhante aos componentes *Hermes Aggregators* e por último, o *context reasoner*, responsável pela inferência ontológica dos dados de contexto, que é semelhante ao componente *Hermes Interpreter* que também suporta mecanismo de inferência baseado em regras.

O projeto também oferece uma camada de consulta, chamada *Query*, para que aplicações e outros serviços de contexto possam acessar e enviar consultas para

obterem dados de interesse, tais como localização ou atividades de usuários, presença de moradores na casa etc. Nesse quesito, *Feel@Home* se diferencia de *Hermes*, pois suas consultas têm finalidades distintas. Em *Hermes*, o objetivo é filtrar, dentre os dados inferidos, os eventos de contexto para atender aos assinantes da infraestrutura.

Em *Feel@Home*, o propósito é acessar informações de contexto atuais persistidas em Base de Conhecimento com a finalidade de se realizar alguma atividade, como por exemplo: se tocam a campainha, o serviço de contexto acessa a camada *Query* para verificar se há algum morador disponível para atendê-la. Do ponto de vista do usuário, não há meios para que ele filtre dados de contexto de seu interesse para ser notificado, pois os próprios serviços de contexto já contêm as consultas específicas para cada requisição.

*Feel@Home* é avaliado funcionalmente em um cenário que notifica os moradores sobre a chegada do carteiro na residência. Entretanto, é mencionado na pesquisa que futuramente a solução também será avaliada do ponto de vista de desempenho. *Hermes Interpreter*, por sua vez, é avaliado em cenário simulado, porém tal avaliação inclui testes de funcionalidade e escalabilidade.

## 6.9 CAF

Badii et al. [3] apresentam o *framework Context Awareness Framework (CAF)* para o projeto de *middleware HYDRA*. Esse *middleware* tem como finalidade prover uma camada de comunicação para ambientes inteligentes, sensíveis a contexto e baseada em arquitetura orientada a serviços. O *framework CAF* oferece suporte para o gerenciamento de contexto para aplicações que se comunicam ao *HYDRA* através de arquitetura baseada em componentes. Possui como principais componentes os *Context Providers*, *Data Acquisition Component (DAqC)*, *Context Manager (CM)* e os *Context Consumers*. Os provedores de contexto obtêm dados de serviços *web* ou sensores. O componente *DAqC* acessa esses dados por consulta ou assinatura de eventos e os envia para o componente *CM*, assinante desses eventos.

Após a notificação, o componente *CM* infere as situações contextuais de alto nível, por meio da execução de regras, descritas no motor de regras *Drools* e, também, por meio do modelo ontológico de contexto. Essas informações de alto nível produzem eventos que devem ser alertados às aplicações assinantes, mediante o componente *Event Manager*.

Outra forma de obtenção de dados de contexto é por meio do componente *Context Querying*, em que aplicações solicitam o estado de contexto de determinadas entidades através de *Named Queries*.

A solução se assemelha a *Hermes* tanto com respeito aos propósitos, quanto à sua organização interna. Porém, *Hermes*, por meio do componente *Hermes Interpreter*,

possibilita que as aplicações parametrizem, dentre os eventos de interesse, o tipo de contexto que desejam ser alertadas. A possibilidade de diminuir essa granularidade dos eventos, por meio de filtragens dinâmicas, não é oferecida pelo componente *Context Manager*, do *framework CAF*.

## 6.10 Fatma et al.

Fatma et al. [12] apresentam um sistema sensível a contexto que adapta as aplicações, conforme alguns parâmetros de contexto, aos dispositivos em que estão sendo executadas. As informações de contexto se referem a dados do usuário, rede, localização, dispositivo, serviços e aplicações, as quais estão modeladas ontologicamente e, respectivamente, descrevem *web services* semânticos com os quais as aplicações dos dispositivos se conectam para notificar os dados de cada tipo de contexto. Até o momento, o processo de adaptação se restringe ao contexto do dispositivo que inclui informações relativas a dados de bateria, conexão *bluetooth*, características tais como, brilho e dimensão da tela, velocidade do processador, informações de volume, câmeras, memória, linguagem, dentre outras.

A arquitetura do sistema contém um aplicativo cliente que envia as informações de contexto para o respectivo *web service*. Cada *web service*, ao receber essas informacões, cria modelos RDF para que sobre eles sejam processadas regras pré-definidas, utilizando a API Jena, da linguagem de programação Java. Um exemplo de regra definida para o contexto do dispositivo é:
$((BatteryLevel(50) \wedge DeviceCharacteristics.ScreenBrightness(100)) \longrightarrow (DeviceCharacteristics.ScreenBrightness(50) \wedge Volume.ActualVolume(5.0))$.

Caso as condições das regras sejam atendidas, o sistema retorna ao aplicativo cliente um documento RDF contendo as novas configurações que devem ser assumidas pelo dispositivo.

Apesar de conter alguns exemplos da capacidade de adaptação do sistema, ele não é avaliado quanto ao tempo de resposta quando os eventos são identificados, como esta pesquisa o faz com o componente *Hermes Interpreter*.

O sistema apresentado por Fatma et al. [12] e a infraestrutura *Hermes* se assemelham quanto aos padrões e métodos utilizados para executarem suas atividades: trabalham com modelagem de contexto ontológica e realizam inferência baseada em regras.

Porém, se diferenciam quanto aos seus propósitos. *Hermes* se propõe a ser uma infraestrutura sensível a contexto semântica capaz de prover serviços tais como inferência, filtragem e persistência dos dados de negócio das aplicações, de forma transparente para

elas. Por outro lado, Fatma et al. [12] fornecem serviços de adaptação dos dispositivos conforme alguns tipos de contexto.

Além dessas diferenças, Fatma et al. [12] realizam comunicação de contexto por meio de *web services*, enquanto *Hermes* utiliza o *middleware* de comunicação DDS. Fatma et al. [12] não apresentam solução para que novos mecanismos de inferência sejam incorporados ao sistema e, devido aos propósitos do sistema, não implementam qualquer método de filtragem de contexto. O trabalho de Fatma et al. [12] apresenta uma visão geral da arquitetura do sistema, como disposição dos componentes e comunicação entre eles, mas não descreve de maneira detalhada as camadas arquiteturais que compõem o componente *processador das regras*, assim como o componente *Hermes Interpreter* é descrito neste trabalho.

## 6.11 Aman et al.

Aman et al. [2] definem um *schema* ontológico para adaptação automática de software de acordo com requisitos de segurança no cenário da IoT - *Internet of Things*. Apresentam também um modelo de software que provê esse serviço, baseado em arquitetura orientada a eventos. As alterações de segurança são coletadas em tempo real pela plataforma OSSIM - Open Source Security Information Management[2], que provê filtragem e normalização de eventos primitivos coletados a partir dos provedores de contexto, que podem ser dispositivos, aplicações ou ferramentas de segurança monitorados. O trabalho abrange três áreas: monitoramento de eventos, correlação de eventos e adaptação de segurança.

Como parte do monitoramento de eventos, está incluída a etapa de filtragem de eventos, cujo objetivo é descartar os eventos indesejados ou redundantes. Para isso, compara-se os dados de contexto dos dispositivos monitorados com *expressões regulares*, a fim de identificar tais eventos e reduzir a perda de informações relevantes durante o processo.

Após a seleção dos eventos relevantes, é executada a etapa de análise, realizada pelo componente *Analyzer*. Essa etapa consiste em ordenar os eventos conforme prioridade, a qual é determinada pelo risco que o evento oferece para a segurança do sistema. Classificados os eventos, o modelo realiza as correlações de eventos, as quais estão especificadas em regras descritas em diretivas XML. Essas regras comparam os atributos dos eventos com a finalidade de produzir situações de alto nível que podem estar associados a alarmes que precisam ser gerenciados.

[2] https://www.alienvault.com/open-threat-exchange/projects

Na última etapa do modelo, ocorre o processo de adaptação de segurança com respeito aos eventos pré-identificados. O contexto, nessa etapa, está modelado em uma ontologia que descreve três tipos de contexto, a saber: usuário, aplicações/dispositivos e segurança. Após a análise do contexto, o componente *Adaptation Engine* então, estabelece as ações de segurança que devem ser providenciadas pelos atuadores instalados nos provedores de contexto monitorados. A comunicação entre o *Adaptation Engine* e os autuadores acontece por meio de mensagens enviadas pelo protocolo de transporte MQTT (*MQ Telemetry Transport*)[3].

Assim como o trabalho de Fatma et al. [12], Aman et al. [2] expõem um modelo sensível a contexto para possibilitar a adaptação de elementos monitorados, de acordo com informações de contexto. Em Aman et al. [2], a adaptação tem a finalidade de adequar os dispositivos monitorados com respeito a ameaças de segurança. Para isso, o modelo de adaptação trabalha com contexto descrito ontologicamente e utiliza inferência baseada em regras, ambas soluções contempladas por *Hermes*.

Apesar dessas semelhanças, o propósito dos trabalhos são distintos, ou seja, *Hermes* oferece uma infraestrutura para provimento de contexto independente de contexto ontológico, enquanto que em Aman et al. [2] o contexto se limita à ontologia de segurança. *Hermes* também provê filtragem semântica parametrizada pelas necessidades das aplicações assinantes, com relação aos dados de contexto pelos quais desejam ser notificadas. Em Aman et al. [2], por sua vez, a filtragem ocorre por comparação entre os dados dos eventos e expressões regulares pré-definidas para cada dispositivo monitorado, visando selecionar os eventos que serão manipulados posteriormente. Conclui-se, então, que as filtragens de contexto presentes nos trabalhos têm finalidades diferentes. Além dessa diferença entre os trabalhos, em Aman et al. [2], os componentes *Analyzer* e *Adaptation Engine*, que realizam interpretação de contexto baseada em regras e ontológica, respectivamente, não têm seus modelos arquiteturais demonstrados de forma detalhada como ocorre neste trabalho, com o componente *Hermes Interpreter*.

Aman et al. realizam estudo de caso em cenário de adaptação de segurança para aplicações sensíveis a contexto da área de saúde. Porém, não há validação do tempo de resposta do sistema frente aos requisitos da área, como foi realizado na simulação de *Hermes Interpreter*.

## 6.12 Yang et al.

Yang et al. [51] apresentam solução para inferir a quantidade de pessoas presentes em ambientes domésticos, além de suas respectivas identidades, baseadas em dados

[3] http://mqtt.org/

de contexto coletados por sensores implantados por empresas, tais como sensores de movimento e medidores de eletricidade inteligentes. O objetivo do trabalho é evidenciar os riscos de segurança aos quais os cidadãos estão expostos devido à presença de tais sensores em suas residências que podem denunciar os comportamentos e rotinas das famílias, expondo a privacidade do lar. Essas informações, por isso, podem ser usadas tanto para fins de publicidade, quanto para atividades maliciosas, tais como assaltos.

O trabalho consiste em realizar inferências baseadas em aprendizado supervisionado e baseadas em regras sobre informações coletadas por sensores em cômodos da casa.

O propósito do trabalho de Yang et al. [51] se difere de *Hermes*, pois a sua solução visa atender a um objetivo específico, que é avaliar o potencial das informações inferidas quanto à privacidade de moradores e não fornecer uma infraestrutura para implantação de aplicações sensíveis a contexto. Apesar disso, trata-se de uma solução sensível a contexto, pois gerencia todo o ciclo de vida do contexto. Assemelha-se com *Hermes Intepreter* no que diz respeito ao atendimento a múltiplas técnicas de inferência, com a finalidade de atender objetivos distintos. Yang et al. [51] realizam filtragem de sensores, com a finalidade de aperfeiçoar e selecionar os dados coletados, porém não realizam filtragem de eventos parametrizadas, pois não há esse tipo de interação com aplicações assinantes de contexto.

A proposta de Yang et al. é instanciada em vários cenários domésticos, variando as técnicas de modelagem e inferência juntamente com a quantidade de sensores e ocupantes dos aposentos da casa. Porém, não foi propósito da validação evidenciar o tempo de resposta da solução, mas verificar a corretude dos dados inferidos para cada contexto.

## 6.13 Resumo comparativo

As tabelas a seguir comparam os trabalhos relacionados ao *Hermes Interpreter* sob alguns critérios explorados neste trabalho. A Tabela 6.1 relaciona os princípios de projeto que devem estar presentes em soluções sensíveis a contexto e outras características consideradas relevantes para a pesquisa na área.

A Tabela 6.2 compara os trabalhos de acordo com especificações técnicas, explorando algumas soluções que foram utilizadas para atender aos requisitos. Em ambas as tabelas, as descrições das colunas são apresentadas inicialmente para melhor visualização das respectivas colunas.

**P1** Extensibilidade

**P2** Modelo de contexto independente

**P3** Suporte a múltiplas técnicas de interpretação de contexto
**P4** Monitoramento e detecção de eventos
**P5** Verificação de consistência entre fatos e modelo de contexto
**P6** Filtragem de contexto
**P7** Filtragem semântica de eventos de contexto
**P8** Apresenta arquitetura interna do componente de interpretação de contexto
**P9** Arquitetura reutilizável
**P10** Manutenibilidade
**P11** Flexibilidade
**P12** Avaliação de desempenho da solução

Se o atendimento ao referido princípio não for explicitado no trabalho ou não estiver entre seus objetivos, será utilizado o marcador '-' na respectiva célula da tabela. Se tiver sido atendido, será marcado com 'S'. Se o respectivo princípio tiver sido explicitamente relacionado como *trabalho futuro*, ele será descrito como 'TF'. Para os projetos que não possuem um nome específico, será nomeado com o nome dos autores, utilizado na referência bibliográfica.

**Tabela 6.1:** *Comparativo conforme princípios de projeto e características da pesquisa*

| Nome | **Ano** | **P1** | **P2** | **P3** | **P4** | **P5** | **P6** | **P7** | **P8** | **P9** | **P10** | **P11** | **P12** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Teymourian et al. | 2009 | - | S | S | S | S | S | S | S | S | - | - | - |
| *Feel@Home* | 2010 | S | S | - | S | S | - | - | - | S | - | - | TF |
| *CAF* | 2010 | S | S | S | S | S | - | - | - | S | - | - | - |
| *COPAL* | 2010 | S | S | - | S | S | S | - | - | S | - | S | - |
| Park et al. | 2011 | TF | TF | - | S | S | S | - | - | S | - | - | - |
| *TrailM* | 2012 | S | - | - | S | - | S | - | - | S | - | - | TF |
| *EXEHDA-UC* | 2013 | - | - | - | S | - | S | - | - | S | - | S | - |
| Guermah et al. | 2013 | S | S | S | S | S | - | - | - | S | S | S | - |
| *DOHA* | 2013 | S | S | S | S | S | - | - | - | S | - | S | - |
| Fatma et al. | 2014 | - | S | S | S | S | - | - | - | - | - | - | - |
| Aman et al. | 2014 | - | S | S | S | S | S | - | - | S | - | S | - |
| Yang et al.. | 2014 | S | - | S | S | - | - | - | - | - | - | S | - |
| ***Hermes Interpreter*** | **2015** | **S** | **S** | **S** | **S** | **S** | **S** | **S** | **S** | **S** | **S** | **S** | **S** |

A listagem abaixo, semelhante a presente no trabalho de Perera et al. [34], descreve as características tecnológicas implementadas pelos trabalhos. Na sequência, na Tabela 6.2, será apresentado um comparativo entre os trabalhos relacionados nesse capítulo. Para esse comparativo, também serão incluídos o componente *Hermes Base* juntamente com o componente *Hermes Interpreter*.

**T1** Modelagem do contexto: ontológica (O), relacional (R), orientada a objetos (OO) ou *schemas* de marcação (M)
**T2** Técnica de Inferência: baseada em regras (R), ontologia (O) ou aprendizado supervisionado (S), como redes bayesianas

**T3** Distribuição e aquisição de contexto: paradigma *publish/subscribe* (P) ou *query* (Q)

**T4** Arquitetura: baseada em componente (1), distribuída (2), baseada em serviços (3), baseada em nó (4), centralizada (5) e cliente-servidor (6)

**T5** Histórico e Persistência: se implementada de alguma forma, 'S'; 'N', não disponível e '-' não informado

**T6** Base de Conhecimento: se implementada de alguma forma, 'S'; 'N', não disponível e '-' não informado

**T7** Qualidade de Contexto: pode ser suporte a resolução de conflito (C) decorrente de incertezas na aquisição do contexto ou suporte à validação de contexto (V), conforme a modelagem, tipos de dados, cardinalidade e outras validações especificadas

**T8** Processamento dos dados: oferece funcionalidades para agregação de contexto (A) ou filtragem de contexto (F)

**Tabela 6.2:** *Comparativo conforme características técnicas*

| Nome | **T1** | **T2** | **T3** | **T4** | **T5** | **T6** | **T7** | **T8** |
|---|---|---|---|---|---|---|---|---|
| Teymourian et al. | O | R,O | P | 2 | S | S | V | A,F |
| *Feel@Home* | O | O | P,Q | 2,4 | S | S | V | A |
| *CAF* | O | R,O | P,Q | 1,3 | S | S | V | A |
| *COPAL* | OO,R | - | P,Q | 1 | S | - | C,V | F |
| Park et al. | - | S | Q | 6 | S | S | C,V | A,F |
| *TrailM* | R | R | P,Q | 1,3,6 | S | N | - | - |
| *EXEHDA-UC* | R | R | P,Q | 1,2,6 | S | - | - | A,F |
| Guermah et al. | O | R,O | P | 1,3 | S | S | V | A |
| *DOHA* | O | R,O | - | 1,3 | - | S | V | A |
| Fatma et al. | O | R,O | P | 3 | - | - | V | - |
| Aman et al. | O,R | R,O | Q | 6 | - | - | C,V | F |
| Yang et al. | - | R,S | P | - | S | S | C | A |
| ***HermesInterpreter e HermesBase*** | **O** | **R,O** | **P** | **1,2** | **N** | **N** | **V** | **F** |

## 6.14 Considerações Finais

Esse capítulo apresentou os trabalhos relacionados à infraestrutura *Hermes*, os quais descrevem soluções de software em geral para computação sensível a contexto, tais como *middlewares*, arcabouços, arquiteturas e infraestruturas. Esses trabalhos foram resumidamente descritos, destacando-se seus propósitos e extraindo as respectivas soluções implementadas para realizar a interpretação de informações de contexto, a fim de inferir situações de alto nível de contexto. Esses módulos ou componentes *interpretadores de contexto* foram, então, comparados ao componente *Hermes Interpreter* de acordo com seus requisitos atendidos, principais características e funcionalidades.

Na sequência do capítulo, foi exibido na Tabela 6.1, um resumo das comparações anteriormente realizadas. Devido à grande quantidade desses requisitos, nem todos foram

minuciosamente utilizados nas comparações ao longo do texto, mas foram identificados nessa tabela, caso presentes nos respectivos trabalhos. Como parâmetros de comparação, utilizaram-se alguns dos princípios de projeto citados por Perera et al. [34] e Bettini et al. [6] que devem guiar o desenvolvimento das soluções para computação sensível a contexto. Foram também acrescentados requisitos elencados neste trabalho para suprir necessidades específicas na computação sensível a contexto, tais como a *filtragem semântica de eventos*, a *manutenibilidade* e a *flexibilidade*.

Desse resumo comparativo, observou-se que, conforme Perera et al. [34] e Bettini et al. [6] atentaram, a *modelagem de contexto independente de aplicação, o monitoramento e detecção de eventos, a verificabilidade de consistência dos fatos de contexto com o modelo e a reusabilidade da arquitetura* foram os princípios mais atendidos.

Em contrapartida, a filtragem de contexto, com ou sem suporte semântico, pouco é explorada nos trabalhos. Em geral, as soluções utilizam o paradigma *publish/subscribe*, porém poucas complementam a atividade de monitoramento e detecção de eventos com a etapa de filtragem. Além disso, não foi observado em nenhum trabalho, a filtragem como uma alternativa parametrizável para as aplicações assinantes em ambientes pervasivos. Esse requisito, ou era associado à filtragem de dados de contexto com a finalidade de resolver conflitos e tratar os dados conforme redundância ou erros, ou era associado a filtros de eventos já prescritos para os respectivos tópicos ou contextos.

Uma contribuição deste trabalho que, dentre os trabalhos relacionados, só foi observada em Teymourian et al. [46], é a arquitetura interna do módulo de interpretação de contexto. Esse componente, em geral, é apresentado somente como uma "caixa preta", em que as mensagens fluem por ele, mas não se conhece o seu funcionamento e camadas internas. Imagina-se que o motivo pelo qual esse contéudo não foi apresentado pelos autores seja porque o objetivo era apresentar a arquitetura completa do sistema sensível a contexto e não detalhar um componente apenas. Descrições da arquitetura como essas favorecem a reusabilidade de software entre projetos e a melhoria das soluções atuais e futuras.

Finalizando os princípios da Tabela 6.1, o requisito *manutenibilidade* pouco foi mencionado nos trabalhos. Provavelmente, devido à generalidade do termo que poderia ser confundido com o termo *flexibilidade*, ou por não ter sido um fator relevante explorado pelos respectivos trabalhos.

Por último, foi apresentada a Tabela 6.2 que exibiu as principais soluções tecnológicas implementadas de cada trabalho. Dessas características técnicas, observou-se que a modelagem ontológica de contexto é bastante difundida entre os trabalhos da área, fato já identificado pelo trabalho de Perera et al. [34]. Devido a essa característica de modelagem, grande parte dessas soluções formam bases de conhecimento para futuras

inferências ontológicas.

Com relação ao item *técnica de inferência*, grande parte das soluções que utiliza modelagem ontológica do contexto, acrescenta *regras* para ampliar suas capacidades de inferência. Com essa característica, esses trabalhos se enquadram como aqueles que suportam múltiplas técnicas de interpretação de contexto.

Outras soluções técnicas bastante contempladas pelos trabalhos são o paradigma *publish/subscribe* para a distribuição e aquisição de contexto; a persistência de dados de contexto e o suporte à validação de contexto, característica associada ao princípio *Verificação de consistência entre fatos e modelo de contexto*, apresentado na Tabela 6.1.

Por último, os trabalhos não apresentam a avaliação de desempenho dos seus respectivos componentes de interpretação e/ou disseminação do contexto. Sistemas sensíveis a contexto, na maioria dos casos, demandam por soluções que ofereçam resposta em tempos aceitáveis [34] [31] . A validação de desempenho, portanto, além de comprovar a eficácia do produto com respeito a esse requisito em seu cenário de aplicação, também contribui para orientar pesquisadores e desenvolvedores da área no tratamento desse requisito em seus respectivos trabalhos.

CAPÍTULO 7

# Conclusões, Limitações e Trabalhos Futuros

A computação sensível a contexto é a área da Ciência da Computação que estuda os meios para possibilitar a interação entre dispositivos computacionais e o ambiente, ou contexto, em que estão inseridos.

A literatura tem demandado soluções que sejam capazes de lidar com a crescente implantação de sensores e a consequente grande produção de dados heterogêneos, o que requer soluções eficientes para uma complexa manipulação desses dados a fim de atender às necessidades das aplicações sensíveis a contexto e, consequentemente, de seus usuários finais [34] [6]. Para isso, são requeridas infraestruturas que, de maneira transparente e reutilizável, realizem modelagem, inferência e disseminação, com relevância, dessas informações.

Diante desse cenário, este trabalho apresentou uma nova abordagem para as etapas de interpretação e disseminação do ciclo de vida de contexto. Essa abordagem, materializada como um componente de software denominado *Hermes Interpreter* (HI), apresenta uma característica que a difere de outros trabalhos da literatura de computação sensível a contexto: o suporte à filtragem semântica de contexto.

O componente HI fornece mecanismos para que as aplicações associem filtros aos tópicos por elas assinados, refinando ainda mais os dados de contexto pelos quais necessitam ser notificadas. Esses filtros são construídos utilizando a própria semântica da ontologia de contexto da infraestrutura *Hermes*, da qual o *Hermes Interpreter* é parte integrante. Essa característica propicia que tais filtros se beneficiem da estrutura e da semântica representadas para um respectivo domínio de aplicação, permitindo que ofereçam capacidade de filtragem compatível com a semântica do contexto. Em função do desacoplamento fornecido pela modelagem ontológica de contexto, o mecanismo de filtragem desenvolvido goza das características de extensibilidade e manutenibilidade.

Outra contribuição dessa pesquisa é o modelo arquitetural de *Hermes Interpreter*, contendo suas camadas e respectivos padrões de projeto e tecnologias que foi detalhadamente apresentado com o intuito de auxiliar no desenvolvimento de novos interpretadores de contexto que reúnam os princípios de projeto que nortearam a condução desta pesquisa, como o suporte a múltiplas técnicas de inferência, filtragem semântica, dentre outros.

Para comunicação dos dados entre seus componentes e aplicações, *Hermes* foi desenvolvida sob o paradigma *publish/subscribe* e utiliza, para isso, os serviços de um *middleware* que implementa a especificação OMG DDS. Para prover o desacoplamento entre esse *middleware* e os demais elementos de *Hermes*, foi desenvolvido o componente *Hermes Base* que oferece uma API para publicação e assinatura de contexto.

A infraestrutura *Hermes*, em especial o componente HI, foi instanciada em cenário de monitoramento de sinais vitais que simulou a assistência de enfermeiros plantonistas a pacientes internados em UTIs e enfermarias. Essa validação fundamentou-se em orientações da equipe de enfermagem do Hospital das Clínicas da Universidade Federal de Goiás. Como os testes não foram realizados em cenário real, utilizou-se uma base de referência internacional de sinais vitais coletados de pacientes em UTIs.

Para validar os componentes, foram descritos casos de teste que foram executados no cenário proposto. Pela simulação, foi possível constatar a contribuição da infraestrutura para auxiliar os profissionais no desempenho de suas atividades, ao realizar a inferência automática de anormalidades dos sinais vitais dos pacientes e notificação dos eventos de seus interesses durante suas atividades de plantão.

Os resultados demonstraram o quão complexa e relevante é a atividade de filtragem de contexto dentro do ciclo de vida de informações de contexto. Apesar dessa atividade não estar explicitamente relacionada na Figura 1.1, este trabalho apresentou a sua relevância para aplicações que utilizam o paradigma *publish/subscribe* para distribuição dos dados de contexto em cenários em que os assinantes possuem necessidades distintas entre si. A sua relevância pode ser comprovada tanto pela validação funcional, ao notificar os assinantes conforme os perfis de contexto desejados, quanto pelas validações de escalabilidade e de comparação, em que se atestou o custo das tarefas de filtragem desempenhadas pelo componente *Hermes Interpeter*. De acordo com os resultados desta pesquisa, é recomendável a inclusão da etapa de *filtragem* entre as etapas de inferência e disseminação de contexto, como ilustrado a Figura 7.1.

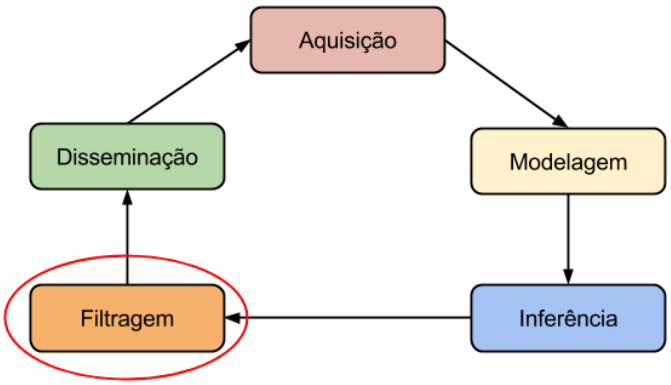


**Figura 7.1:** *Etapa de filtragem no ciclo de vida de contexto.*

Como apresentado neste trabalho, para que as informações de alto nível produzidas na etapa de *inferência* sejam entregues de forma eficiente às aplicações sensíveis a contexto, faz-se necessário um mecanismo dinâmico, flexível e extensível de manipulação de filtros dos assinantes. Esse mecanismo se torna mais eficiente se alinhado com as especificações semânticas contextuais da infraestrutura sensível a contexto.

## 7.1 Limitações

Apesar das contribuições apresentadas, os componentes *Hermes Interpreter* e *Hermes Base* possuem limitações quanto a determinados cenários de uso, tecnologias e implementação, as quais podem ser exploradas em pesquisas futuras. As limitações do componente *Hermes Interpreter* são as seguintes:

- O modelo RDF notificado aos assinantes contém exatamente os mesmos dados incluídos no filtro, juntamente com os originalmente publicados pelos componentes *Hermes Widgets*, como por exemplo: Se o filtro é por anormalidade e idade do paciente, o modelo conterá somente essas informações complementadas com as originais. O componente não oferece meios para que, à parte do filtro escolhido, os assinantes também detalhem as informações de contexto de que desejam ser notificados. Só é possível especificar os eventos de contexto de interesse;
- Os contextos inferidos não são reutilizáveis por não haver persistência dos mesmos. Essa limitação impossibilita a criação de filtros que remetem a informações históricas, por exemplo: paciente com hipotensão que apresentou hipertensão no dia anterior. Entretanto, para as validações apresentadas nesta pesquisa, foram apenas consideradas as situações de contexto momentâneas. Outra consequência dessa limitação, é que os componentes *Hermes Widgets* deveriam sempre publicar todos os dados referentes aos perfis dos pacientes e enfermeiros. Isso não afetou os experimentos porque eles não se propuseram a avaliar o tráfego na rede que seria influenciado por essa quantidade extra de informações;
- Se a execução de alguma aplicação assinante for interrompida, *Hermes Interpreter* continuará a notificá-la, mesmo que o *middleware* DDS a exclua de sua lista de assinantes;
- Se a execução de *Hermes Interpreter* for interrompida, todos os assinantes deverão repetir a assinatura dos tópicos.
- Apesar de ter sido projetado para ser extensível quanto a novas técnicas de inferência, não foi possível validar essa característica da arquitetura. O trabalho se restringiu a implementar as interpretações de contexto ontológicas e baseada em regras.

- Como ainda não existem aplicações reais se comunicando com *Hermes*, os parâmetros suportados pelo HI estão estáticos nas aplicações. A inclusão de um novo parâmetro, como a *pressão arterial média*, abordada no estudo de caso, não é notificada pelo HI para as aplicações.

*Hermes Base* também apresenta algumas limitações se comparado a outros *middlewares* com o mesmo propósito. A razão para isso é que não é objetivo deste trabalho realizar um estudo exaustivo sobre *middlewares* de comunicação. Por isso, algumas funcionalidades do DDS foram exploradas apenas superficialmente com o objetivo de possibilitar a comunicação de acordo com paradigma *publish/subscribe*, para o cenário de simulação de monitoramento de sinais vitais. As limitações são as seguintes:

- Acoplado somente a uma implementação do *middleware* DDS, no caso o produto *CoreDX DDS*. A alteração dessa tecnologia implicará na manutenção do componente
- Políticas estáticas de QoS para publicadores e assinantes
- Acessível somente por aplicações JAVA
- Restrito a um domínio DDS. DDS é organizado em domínios e cada domínio possui seus tópicos, publicadores e assinantes específicos, conforme explicado no Capítulo 3. Os domínios dentro de uma aplicação DDS podem ser acessados simultaneamente de forma que as publicações e tópicos não fiquem visíveis entre domínios distintos

## 7.2 Trabalhos Futuros

Novas pesquisas podem ser acrescentadas ao trabalho, explorando melhorias tanto em *Hermes Interpreter* quanto em *Hermes Base*. Os pontos em aberto são:

**Persistência dos contextos inferidos**: As informações inferidas pelo *Hermes Interpreter* são utilizadas somente para filtragem e notificação naquele exato momento em que a publicação ocorre. Após isso, a informação é perdida, pois não é mantida nem em memória pelo componente. Um trabalho futuro é implementar a persistência dos dados de contexto a cada publicação de *Hermes Widgets*, para possibilitar a consulta, inferência e filtragem sobre dados históricos. Para essa implementação, deve-se atentar para a quantidade máxima de filtros FCID que o componente deve suportar, para atender às exigências de tempo de resposta do respectivo cenário em que *Hermes Intepreter* estiver operando.

**Parametrização de políticas de qualidade de serviço**: Evolução do *Hermes Base* para possibilitar a parametrização de políticas de QoS. No momento, esse componente efetua essa comunicação através de políticas estáticas, imutáveis e idênticas entre os

comunicantes. Porém, em determinados cenários, essas políticas devem ser variadas, devido às características das aplicações comunicantes, como por exemplo: um *smartphone* com pouca capacidade de hardware, executando uma aplicação assinante de *Hermes*, poderia parametrizar determinadas políticas de publicação de contexto para enviar dados de acordo com sua capacidade de processamento. Ou, ainda, o próprio HB se executando em uma rede *wi-fi* com pouca banda de dados disponível, poderia, automaticamente, configurar as políticas de publicadores para notificarem com intervalo maior do que em cenário com alta disponibilidade de tráfego.

**Priorização de filtros de acordo com políticas de QoS**: Uma vez que as políticas de QoS se tornem parametrizadas, um ponto a ser abordado é viabilizar para que o HI tenha conhecimento das políticas exigidas pelos assinantes de contexto. Baseado nisso, priorizar a execução dos filtros com mais urgência de resposta.

**Divulgação dos filtros suportados**: A cada inclusão ou remoção de parâmetro de filtro suportado pelo HI, esse deve notificar as aplicações, ou por meio de *queries*, ou por publicação em tópicos. Isso torna a infraestrutura *Hermes* mais dinâmica quanto à manutenção dos filtros.

**Permitir a parametrização de dados de contexto notificados**: Apesar de realizar a filtragem de eventos para aplicações, o modelo semântico notificado às aplicações sempre contém os dados originalmente publicados pelos HWs juntamente com os fatos inferidos, se estiverem dentre os parâmetros dos filtros. Possibilitar que as aplicações também parametrizem os dados que desejam ser notificadas, à parte dos eventos, trará mais benefícios para o desempenho de suas atividades. Outra possibilidade, é permitir que o próprio HI determine os dados que devem ser publicados, baseado na quantidade de assinantes para o tópico. Com isso, para cenários de pouca disponibilidade de banda e/ou com vários assinantes, o componente pode ser flexível para retornar modelos RDF de tamanhos compatíveis com o cenário.

**Suporte a outros mecanismos de interpretação de contexto**: Validar a extensibilidade do *Hermes Interpreter* quanto à inclusão de novas técnicas de interpretação de contexto e torná-lo mais próximo do que recomenda a literatura quanto ao suporte a múltiplas técnicas de interpretação de contexto [6] [34]. Essa ampliação possibilitará à infraestrutura *Hermes*, cada vez mais, se submeter a contextos de características diferentes das atuais, em que técnicas específicas de interpretação são mais apropriadas do que as implementadas até o momento. No âmbito de aplicações sensíveis a contexto da área da saúde, por exemplo, o *aprendizado supervisionado* seria adequado para monitoramento de quedas de idosos em ambiente *homecare*; um outro exemplo é a técnica *lógica probabilística*, aplicável no apoio a diagnósticos.

***Hermes Interpreter* adaptativo em tempo de execução**: No momento, cada tópico de *Hermes Interpreter* está associado a uma técnica de inferência previamente

configurada. O componente poderia ser evoluído para decidir, em tempo de execução, qual(is) mecanismo(s) utilizar para o contexto a ele notificado.

***Hermes Intepreter* como *endpoint***: Já que os dados de contexto são persistidos, *Hermes Interpreter* pode ser evoluído para se tornar um *SPARQL endpoint*. Um *SPARQL endpoint* é um serviço que recebe consultas SPARQL e retorna resultados em diferentes formatos, dependendo do tipo da consulta [20]. No contexto da saúde, *Hermes Interpreter* poderia ser um prontuário médico eletrônico acessível globalmente, resguardando as políticas de privacidade estabelecidas pela comunidade médica.

**Distribuição do processamento de inferência e filtragem de contexto**: Paralelizar essas tarefas em ambiente distribuído para melhorar o desempenho do componente.

**Modelo de distribuição de carga entre diferentes HIs**: Ao invés de uma abordagem centralizada do *Hermes Interpreter*, uma instância de *Hermes* pode conter vários HIs com o intuito de distribuir entre eles os tópicos existentes em uma rede DDS;

**Interface amigável para configuração de ontologias e filtros**: Do ponto de vista do usuário administrador de *Hermes*, um possível trabalho futuro é a criação de interfaces amigáveis para manutenção das ontologias de contexto suportadas como também dos filtros mantidos nos arquivos JSON. Uma interface para manter os parâmetros dos filtros, com seus respetivos metadados e valores, traria agilidade na execução de *Hermes* em um ambiente real.

## 7.3 Considerações Finais

Esse capítulo descreveu a conclusão da pesquisa, com destaque para a relevância da filtragem para disseminação de contexto nos cenários com comunicação *publish/subscribe* e com assinantes com necessidades distintas. O mecanismo se apoia na semântica do modelo contextual, propiciando filtragens mais ricas e expressivas se comparadas às meramente sintáticas existentes na comunicação *publish/subscribe* convencional.

*Hermes Interpreter* foi validado em cenário de saúde somente, entretanto sua arquitetura pode ser reutilizada em outros cenários de computação sensível a contexto. Essa generalização se dá pelo desacoplamento entre o código-executável do componente e a estrutura que mantém os filtros associados aos tópicos. Outros cenários passíveis para sua utilização são as áreas financeira, conforme abordado por Teymourian et al. [46] e militar, os quais são caracterizados pela existência de vários assinantes que possuem necessidades de contexto distintas entre si.

Ao final do capítulo, foram apresentados as limitações da pesquisa e os trabalhos futuros que podem ser explorados na área de web semântica e computação sensível a contexto.

# Referências Bibliográficas


[1] ALUR, D.; CRUPI, J.; MALKS, D. **Core J2EE Patterns**. Elsevier, 2004.

[2] AMAN, W.; SNEKKENES, E. **Event driven adaptive security in internet of things**. *UBICOMM 2014, The Eighth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, p. 7–15, Aug. 2014.

[3] BADII, A.; CROUCH, M.; LALLAH, C. **A context-awareness framework for intelligent networked embedded systems**. In: *Advances in Human-Oriented and Personalized Mechanisms, Technologies and Services (CENTRIC), 2010 Third International Conference on*, p. 105–110, Aug 2010.

[4] BASTOS, A. B. **Uma Abordagem Ontológica Baseada em Informações de Contexto para Representação de Conhecimento**. In: *Dissertação de Mestrado, Instituto de Informática da Universidade Federal de Goiás*. 2013. 1–137.

[5] BASTOS, A. B. ; SENE JÚNIOR, I. G. . B. A.-N. R. F. **Modelagem e inferência baseadas na semântica de monitoramento de sinais vitais humanos**. In: *XX Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia)*, p. 13–16, João Pessoa-PB, 2014.

[6] BETTINI, C.; BRDICZKA, O.; HENRICKSEN, K.; INDULSKA, J.; NICKLAS, D.; RANGANATHAN, A.; RIBONI, D. **A survey of context modelling and reasoning techniques**. *Pervasive and Mobile Computing*, 6(2):161 – 180, 2010.

[7] BIKAKIS, A.; PATKOS, T.; ANTONIOU, G.; PLEXOUSAKIS, D. **A survey of semantics-based approaches for context reasoning in ambient intelligence**. In: *Constructing Ambient Intelligence*, volume 11 de **Communications in Computer and Information Science**, p. 14–23. Springer Berlin Heidelberg, 2008.

[8] BOUFLEUER, R.; ROMERO, B. A.; NETO, A. D. F.; LIMA, J. C. D.; AUGUSTIN, I.; PETRY, M. T.; CARLESSO, R. **Desenvolvimento de um medidor de umidade e pluviometria baseado em uma arquitetura de sensoriamento remoto sensível ao contexto para agricultura de precisão**. In: *Sistemas Sociais e Eventos de Grandes*

*Massas: Ampliando Desafios da Computação, SBCUP - VI Simpósio Brasileiro de Computação Ubíqua e Pervasiva*, p. 852–861, Julho 2014.

[9] CAROTHERS, G.; PRUD'HOMMEAUX, E. **RDF 1.1 turtle**. W3C recommendation, W3C, Feb. 2014. http://www.w3.org/TR/2014/REC-turtle-20140225/.

[10] DEY, A. K.; ABOWD, G. D.; SALBER, D. **A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications**. *Human-computer interaction*, 16(2):97–166, 2001.

[11] DOUKAS, C.; MAGLOGIANNIS, I.; TRAGAS, P.; LIAPIS, D.; YOVANOF, G. **Patient fall detection using support vector machines**. In: Boukis, C.; Pnevmatikakis, A.; Polymenakos, L., editors, *Artificial Intelligence and Innovations 2007: from Theory to Applications*, volume 247 de **IFIP The International Federation for Information Processing**, p. 147–156. Springer US, 2007.

[12] FATMA, A.; ANIS, J.; FAIEZ, G. **The pervasive information system adaptation: Android device context**. *UBICOMM 2014, The Eighth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, p. 27–34, Aug. 2014.

[13] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns - Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1994.

[14] GOLDBERGER, A.; AMARAL, L.; GLASS, L.; HAUSDORFF, J.; IVANOV, P.; MARK, R.; MIETUS, J.; MOODY, G.; PENG, C.; STANLEY, H. **Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals**. In: *Circulation Electronic Pages*, volume 23, p. 215?220, 2000.

[15] GRUBER, T. **A translation approach to portable ontologies**. Knowledge Acquisition, 1993.

[16] GUERMAH, H.; FISSAA, T.; HAFIDDI, H.; NASSAR, M.; KRIOUILE, A. **Context modeling and reasoning for building context aware services**. In: *ACS International Conference on Computer Systems and Applications*, p. 1–7, 2013.

[17] GUHA, R.; BRICKLEY, D. **RDF schema 1.1**. W3C recommendation, W3C, Feb. 2014. http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[18] GUO, B.; SUN, L.; ZHANG, D. **The architecture design of a cross-domain context management system**. In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, p. 499–504, March 2010.

[19] HARRIS, S.; SEABORNE, A. **SPARQL 1.1 query language**. W3C recommendation, W3C, Mar. 2013. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[20] HEBELER, J.; FISHER, M.; BLACE, R.; PEREZ-LOPEZ, A. **Semantic Web Programming**. John Wiley & Sons, 2009. 652 pages.

[21] HEITLAGER, I.; KUIPERS, T.; VISSER, J. **A practical model for measuring maintainability**. In: *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*, p. 30–39, Sept 2007.

[22] HORROCKS, I.; PATEL-SCHNEIDER, P. F.; BOLEY, H.; TABET, S.; GROSOF, B.; DEAN, M. **SWRL: A semantic web rule language combining OWL and RuleML**. W3C recommendation, W3C, May 2004. http://www.w3.org/Submission/SWRL/.

[23] KORB, K. B.; NICHOLSON, A. E. **Bayesian Artificial Intelligence**. Taylor e Francis, 2003. 392 pages.

[24] KOREL, B. T.; KOO, S. G. M. **A survey on context-aware sensing for body sensor networks abstract**, 2010.

[25] LI, F.; SEHIC, S.; DUSTDAR, S. **Copal: An adaptive approach to context provisioning**. In: *Wireless and Mobile Computing, Networking and Communications (WiMob), 2010 IEEE 6th International Conference on*, p. 286–293, Oct 2010.

[26] LOPES, J.; GUSMAO, M.; SOUZA, R.; DAVET, P.; SOUZA, A.; COSTA, C.; BARBOSA, J.; PERNAS, A.; YAMIN, A.; GEYER, C. **Towards a distributed architecture for context-aware mobile applications in ubicomp**. In: *Proceedings of the 19th Brazilian Symposium on Multimedia and the Web*, p. 43–50, Salvador-BA, Brazil, 2013.

[27] MARANHÃO, G. M.; SENE JÚNIOR, I. G.; BULCÃO NETO, R. F. **Anatomia de um interpretador de contexto semântico com suporte à notificação de eventos em tempo real**. In: *XX Simpósio Brasileiro de Sistemas Multimídia e Web (Webmedia)*, p. 1–4, João Pessoa-PB, Brasil, 2014.

[28] MARTINS, C.; ROSA, J. A.; FRANCO, L.; BARBOSA, J.; BEZERRA, E. **Towards a model to explore business opportunities in trail-aware environments**. In: *Proceedings of the 18th Brazilian Symposium on Multimedia and the Web*, p. 143–150, São Paulo-SP, Brazil, 2012.

[29] MATTHIAS, G.; BOAZ, A. M.; DWAYNE, R. W. **Improving alarm performance in the medical intensive care unit using delays and clinical context anesthesia e analgesia**. 108(5):1546–1552, May 2009.

[30] MOODY, G.; MARK, R. **A database to support development and evaluation of intelligent intensive care monitoring**. In: *Computers in Cardiology*, volume 23, p. 657?660, 1996.

[31] NETO, R. F. B. A.; DA GRAÇA PIMENTEL, M. **Performance evaluation of inference services for ubiquitous computing**. In: *Proceedings of the 12th Brazilian Symposium on Multimedia and the Web*, WebMedia '06, p. 27–34, New York, NY, USA, 2006. ACM.

[32] PARK, H.-S.; OH, K.; CHO, S.-B. **Bayesian network-based high-level context recognition for mobile context sharing in cyber-physical system**. *International Journal of Distributed Sensor Networks*, 2011.

[33] PEIZHI, L.; JIAN, Z. **A context-aware application infrastructure with reasoning mechanism based on dempster-shafer evidence theory**. In: *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE*, p. 2834–2838, May 2008.

[34] PERERA, C.; ZASLAVSKY, A.; CHRISTEN, P.; GEORGAKOPOULOS, D. **Context aware computing for the internet of things: A survey**. *IEEE Communications Surveys Tutorials*, 16(1):414–454, 2014.

[35] PÉREZ, J.; ARENAS, M.; GUTIERREZ, C. **Semantics and complexity of sparql**. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, Sept. 2009.

[36] PICALAUSA, F.; VANSUMMEREN, S. **What are real sparql queries like?** In: *Proceedings of the International Workshop on Semantic Web Information Management*, SWIM '11, p. 7:1–7:6, New York, NY, USA, 2011. ACM.

[37] RANGANATHAN, A.; CAMPBELL, R. H. **A middleware for context-aware agents in ubiquitous computing environments**. In: *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, Middleware '03, p. 143–161, New York, NY, USA, 2003. Springer-Verlag New York, Inc.

[38] RIVA, O.; DI FLORA, C.; RUSSO, S.; RAATIKAINEN, K. E. E. **Unearthing design patterns to support context-awareness**. In: *PerCom Workshops'06*, p. 383–387, 2006.

[39] RODRÍGUEZ VALENZUELA, S.; HOLGADO TERRIZA, J.; PETKOV, P.; HELFERT, M. **Modeling context-awareness in a pervasive computing middleware using ontologies and data quality profiles**. In: *PerCAM 2013, 3rd International Workshop on Pervasive and Context-Aware Middleware*, p. 271–282, 2013.

[40] ROSSI, G.; GORDILLO, S.; LYARDET, F. **Design patterns for context-aware adaptation**. In: *Applications and the Internet Workshops, 2005. Saint Workshops 2005. The 2005 Symposium on*, p. 170–173, Jan 2005.

[41] SCHLESSELMAN, J.; PARDO-CASTELLOTE, G.; FARABAUGH, B. **Omg data-distribution service (dds): architectural update**. In: *Military Communications Conference, 2004. MILCOM 2004. 2004 IEEE*, volume 2, p. 961–967 Vol. 2, Oct 2004.

[42] SEHIC, S.; LI, F.; NASTIC, S.; DUSTDAR, S. **A programming model for context-aware applications in large-scale pervasive systems**. In: *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on*, p. 142–149, Oct 2012.

[43] SHUAI, J.; HUAXIN, M. **Design patterns in object oriented analysis and design**. In: *Software Engineering and Service Science (ICSESS), 2011 IEEE 2nd International Conference on*, p. 326–329, July 2011.

[44] SOUZA, A.; MESQUITTA, F.; LOPES, J.; SOUZA, R.; PERNAS, A.; YAMIN, A.; GEYER, C. **Uma abordagem ubíqua consciente de situação para avaliação de metas terapêuticas em ambiente hospitalar**. In: *Sistemas Sociais e Eventos de Grandes Massas: Ampliando Desafios da Computação, SBCUP - VI Simpósio Brasileiro de Computação Ubíqua e Pervasiva*, p. 921–930, Julho 2014.

[45] TERZIYAN, V.; KAYKOVA, O.; ZHOVTOBRYUKH, D. **Ubiroad: Semantic middleware for context-aware smart road environments**. In: *Proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services*, ICIW '10, p. 295–302, Washington, DC, USA, 2010. IEEE Computer Society.

[46] TEYMOURIAN, K.; STREIBEL, O.; PASCHKE, A.; ALNEMR, R.; MEINEL, C. **Towards semantic event-driven systems**. In: *Proceedings of the 3rd International Conference on New Technologies, Mobility and Security*, NTMS'09, p. 347–352, Piscataway, NJ, USA, 2009. IEEE Press.

[47] VEIGA, E. F.; MARANHAO, G. M.; BULCÃO NETO, R. F. **Apoio ao desenvolvimento de aplicações de tempo real sensíveis a contexto semântico**. In: *XX Simpósio Brasileiro de Sistemas Multimidia e Web*, 2014.

[48] WEISER, M. **The computer for the 21st century**. *Scientific American*, 265(3):66–75, 1991.

[49] WELTY, C.; MCGUINNESS, D. **OWL web ontology language guide**. W3C recommendation, W3C, Feb. 2004. http://www.w3.org/TR/2004/REC-owl-guide-20040210/.

[50] WOOD, D.; LANTHALER, M.; CYGANIAK, R. **RDF 1.1 concepts and abstract syntax**. W3C recommendation, W3C, Feb. 2014. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[51] YANG, L.; TING, K.; SRIVASTAVA, M. **Inferring occupancy from opportunistically available sensor data**. In: *Pervasive Computing and Communications (PerCom), 2014 IEEE International Conference on*, p. 60–68, March 2014.

[52] YUAN, B.; HERBERT, J. **Web-based real-time remote monitoring for pervasive healthcare**. In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*, p. 625–629, March 2011.

[53] ZHANG, C.; BUDGEN, D.; DRUMMOND, S. **Using a follow-on survey to investigate why use of the visitor, singleton amp; facade patterns is controversial**. In: *Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium on*, p. 79–88, Sept 2012.

APÊNDICE A

# Diagramas de classe e sequência dos cenários de Hermes Base

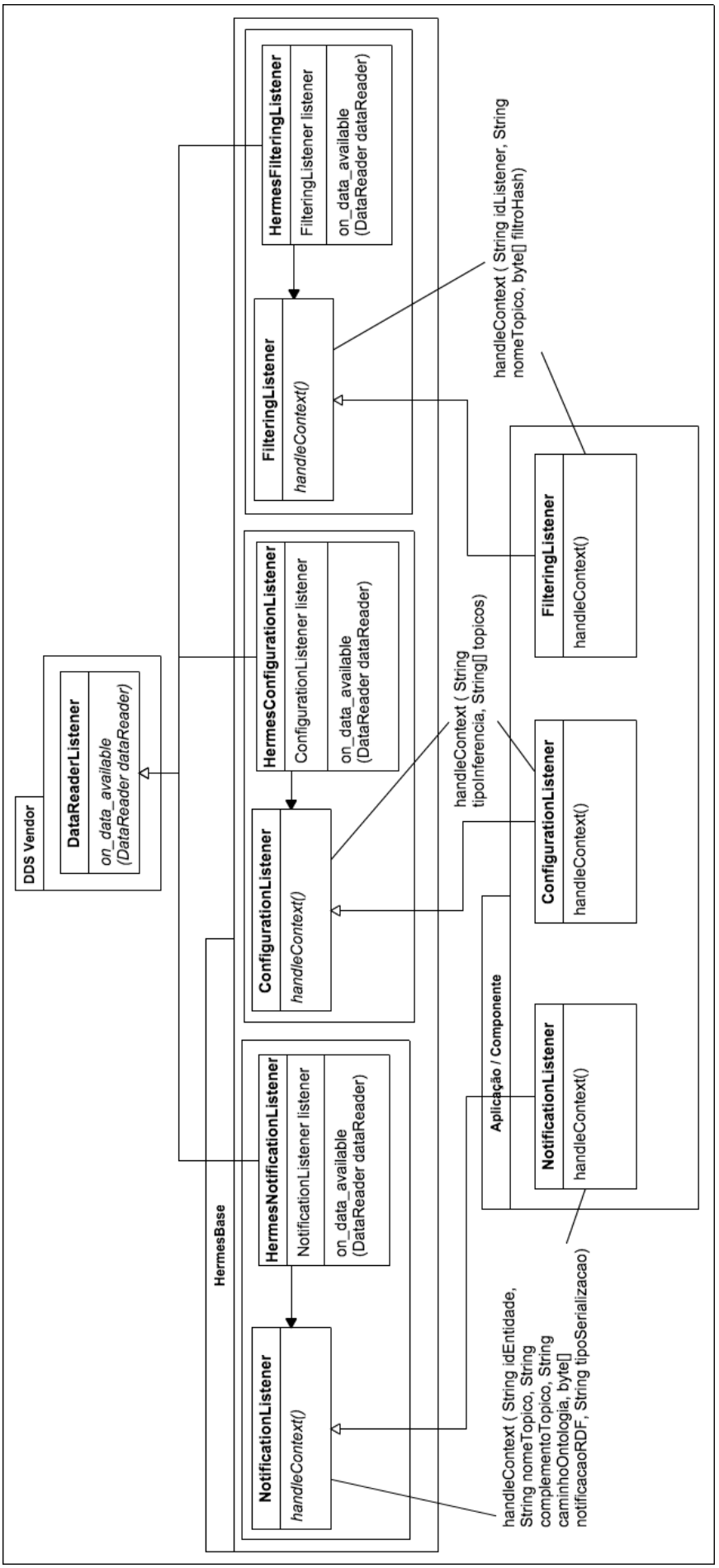


**Figura A.1:** *Diagrama de classe dos Listeners de Hermes Base*

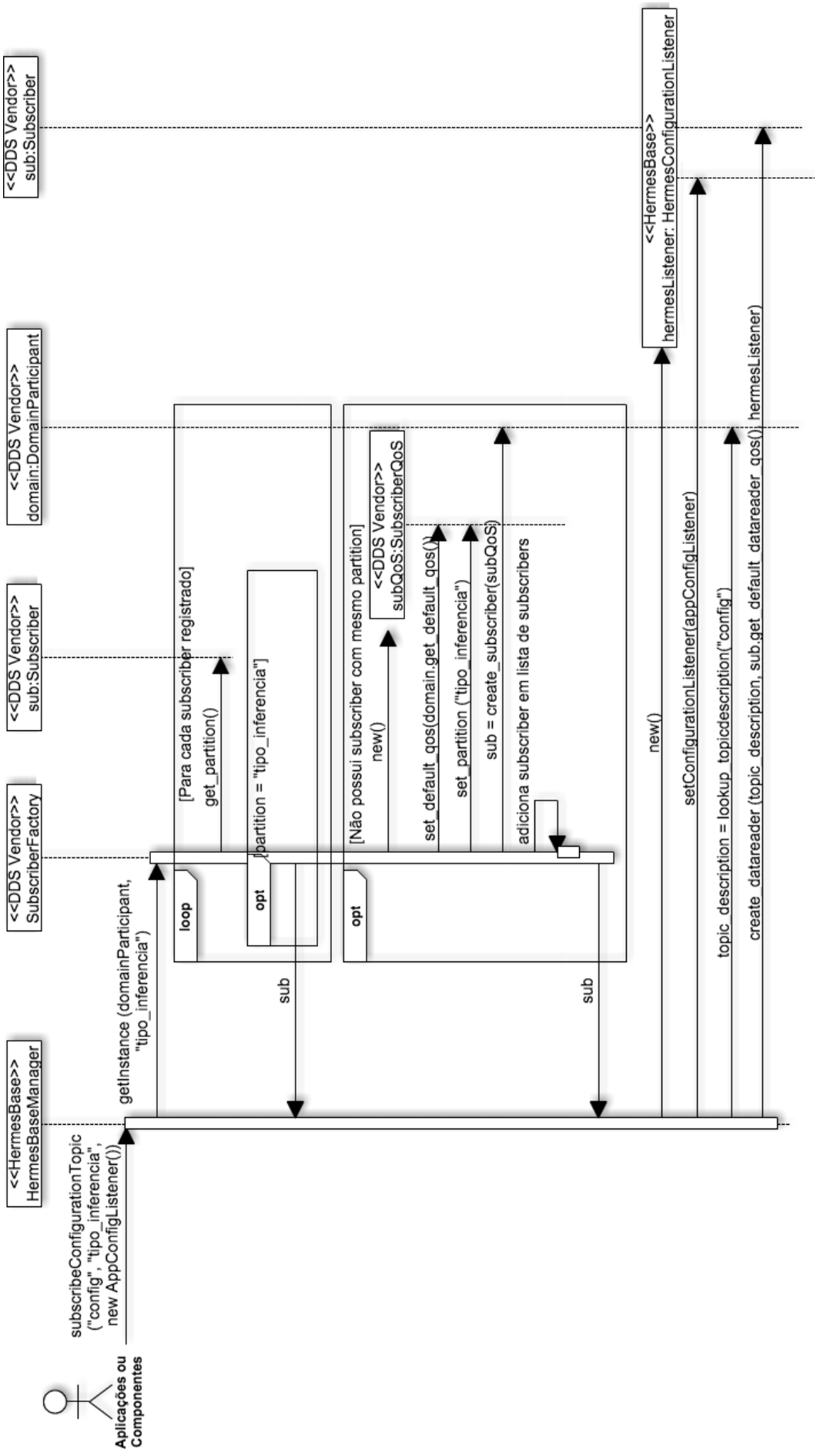


**Figura A.2:** *Assinatura em tópico de configuração*

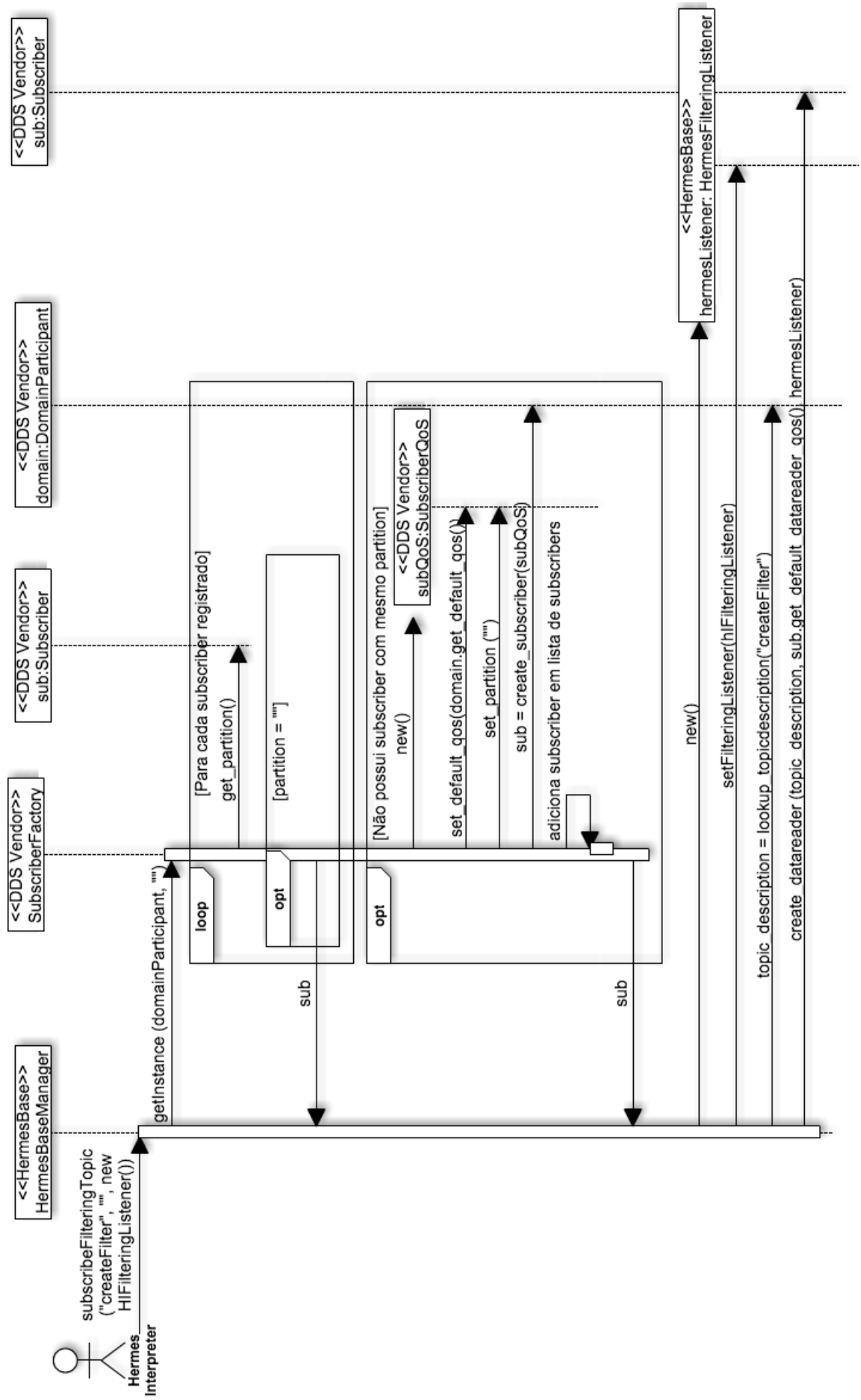


**Figura A.3:** *Assinatura em tópico de filtragem*

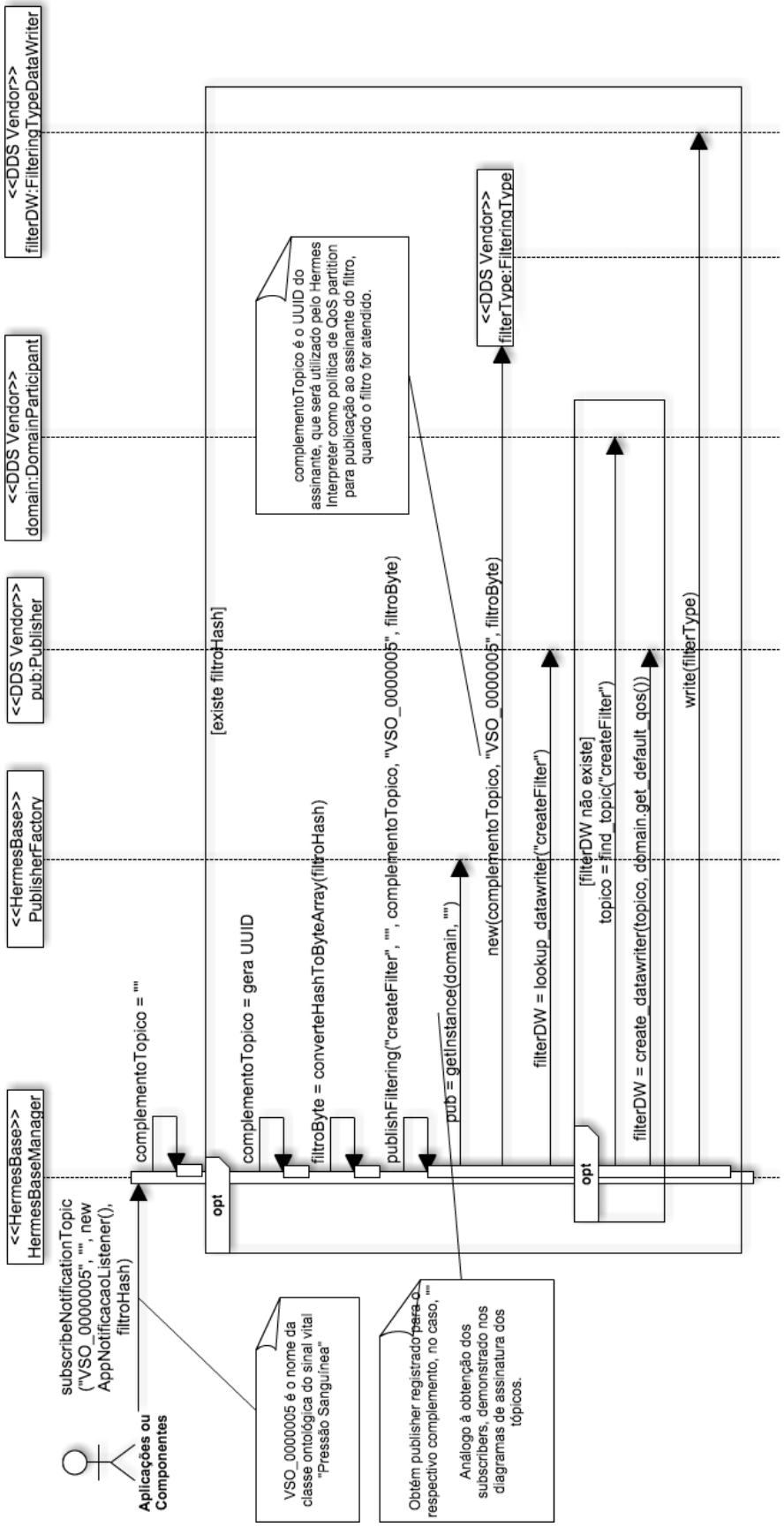


**Figura A.4:** *Assinatura em tópico de notificação - parte 1 de 2*

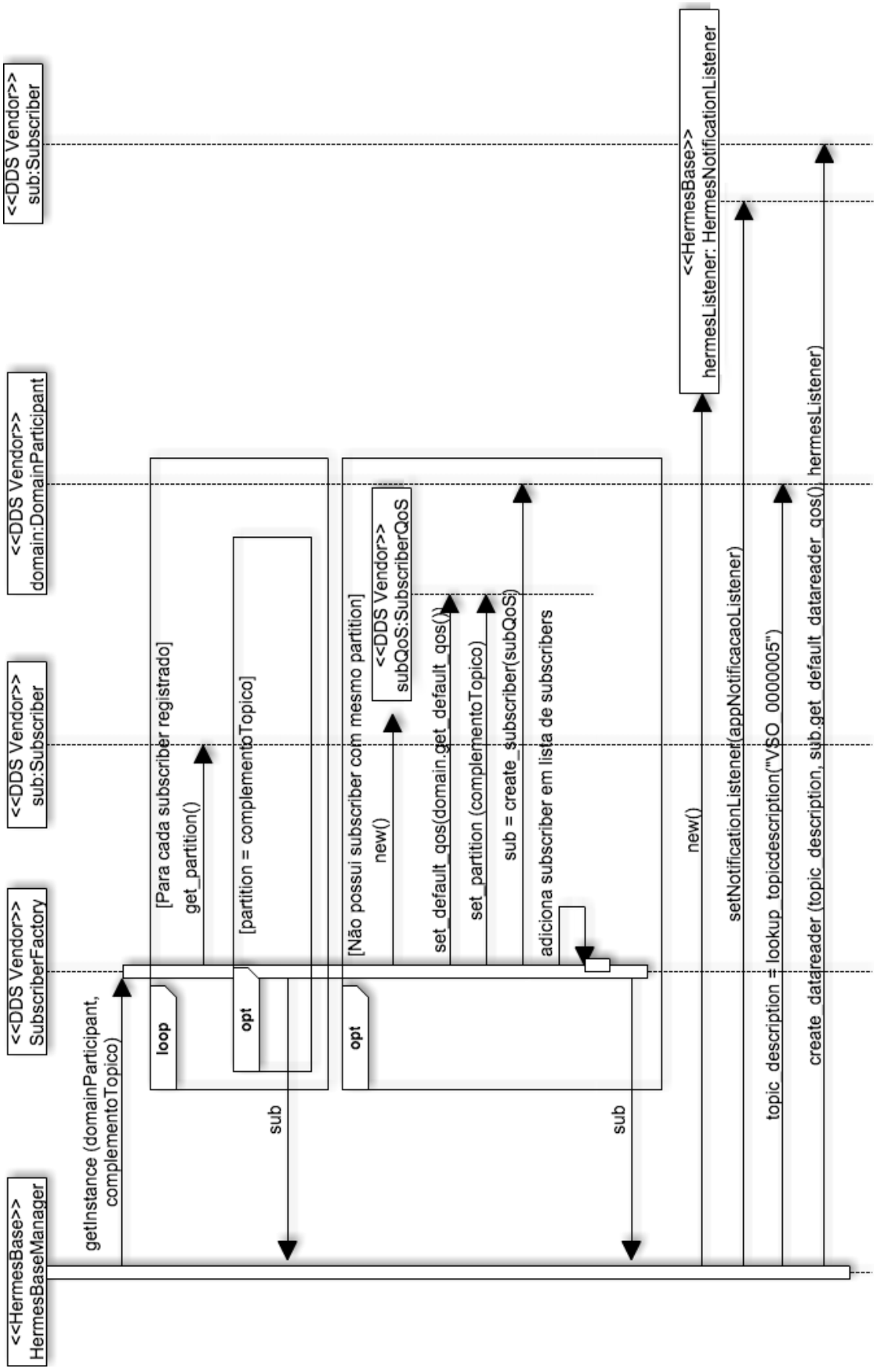


**Figura A.5:** *Assinatura em tópico de notificação - parte 2 de 2*

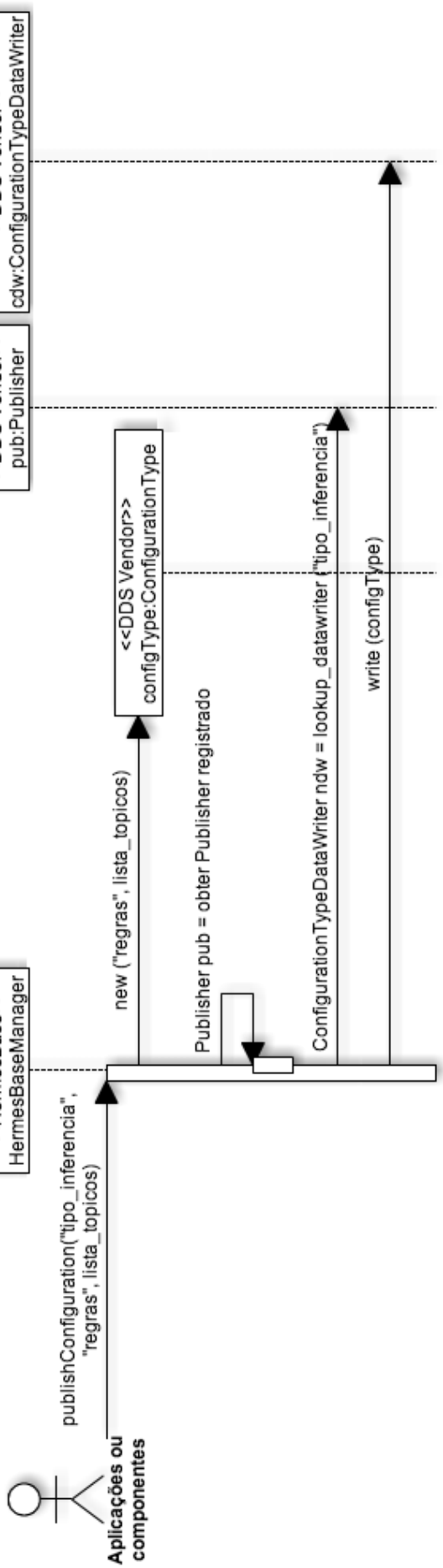


**Figura A.6:** *Publicação em tópico de configuração*

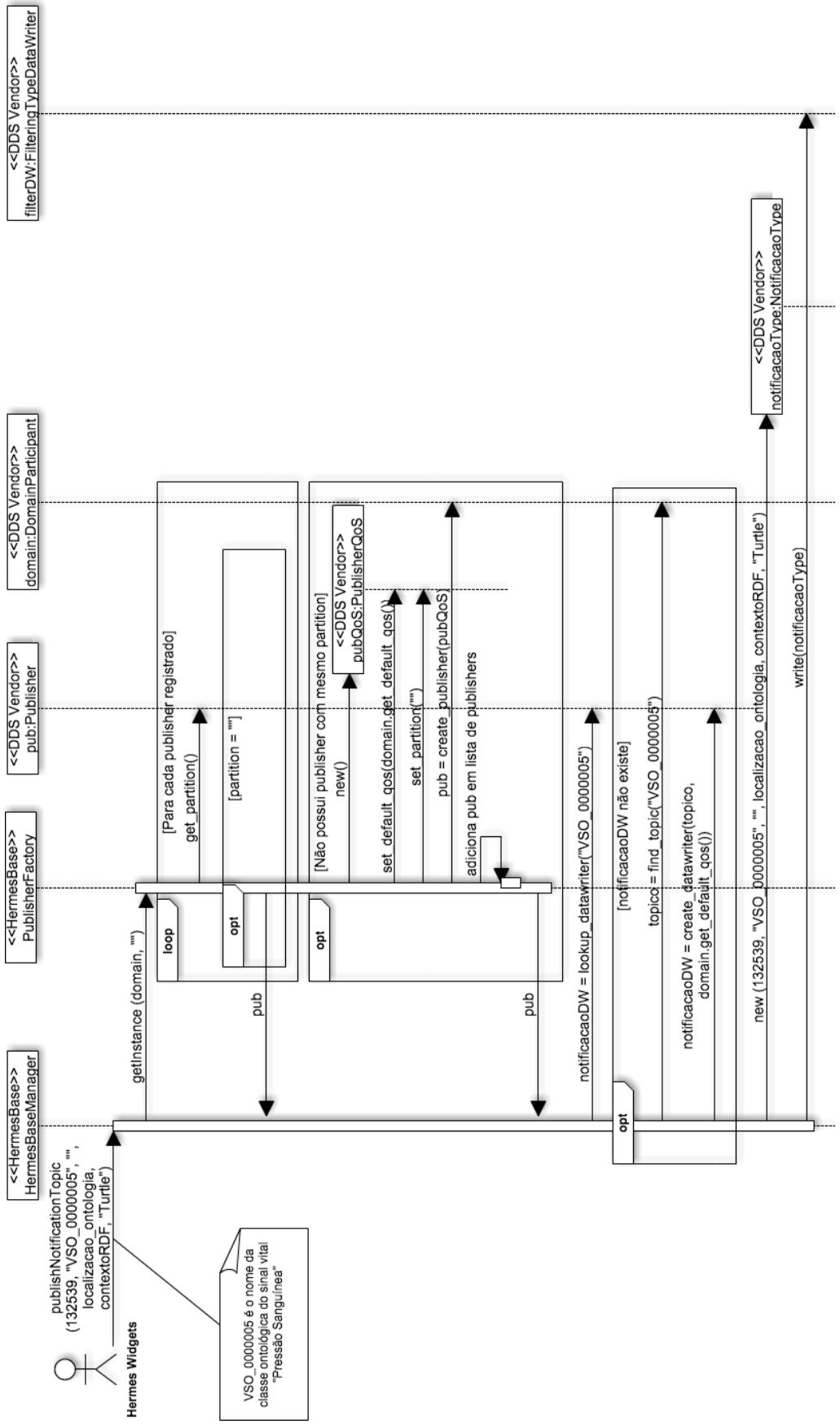


**Figura A.7:** *Publicação em tópico de notificação*

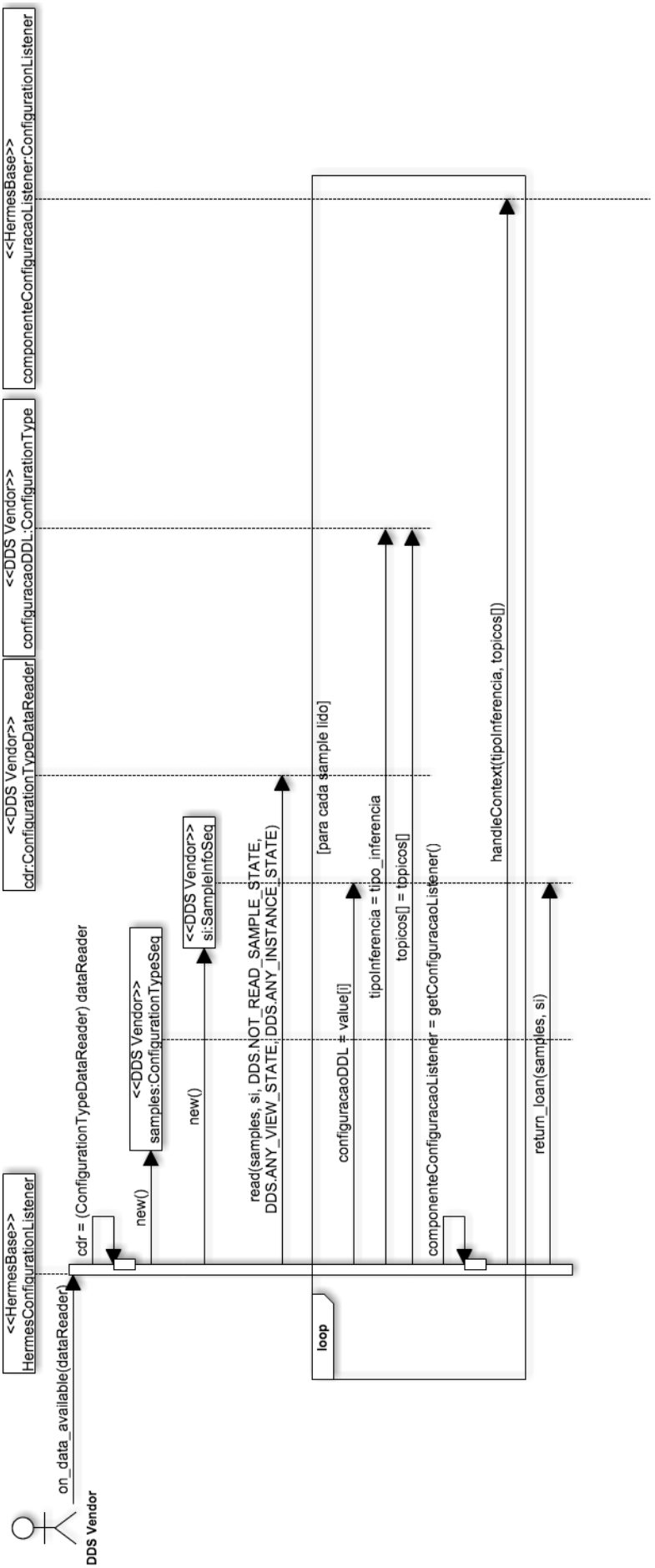


**Figura A.8:** *Listener de Configuração invocado pelo DDS*

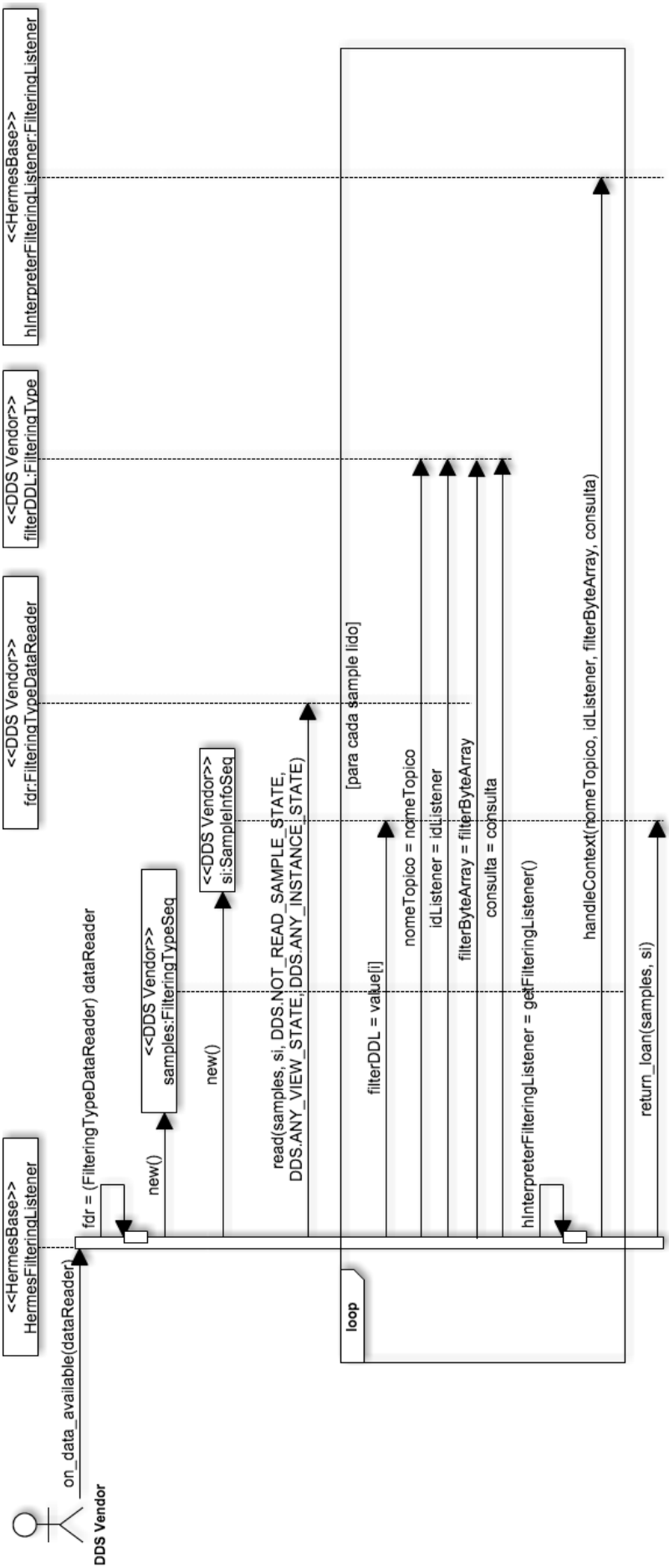


**Figura A.9:** *Listener de Filtragem invocado pelo DDS*

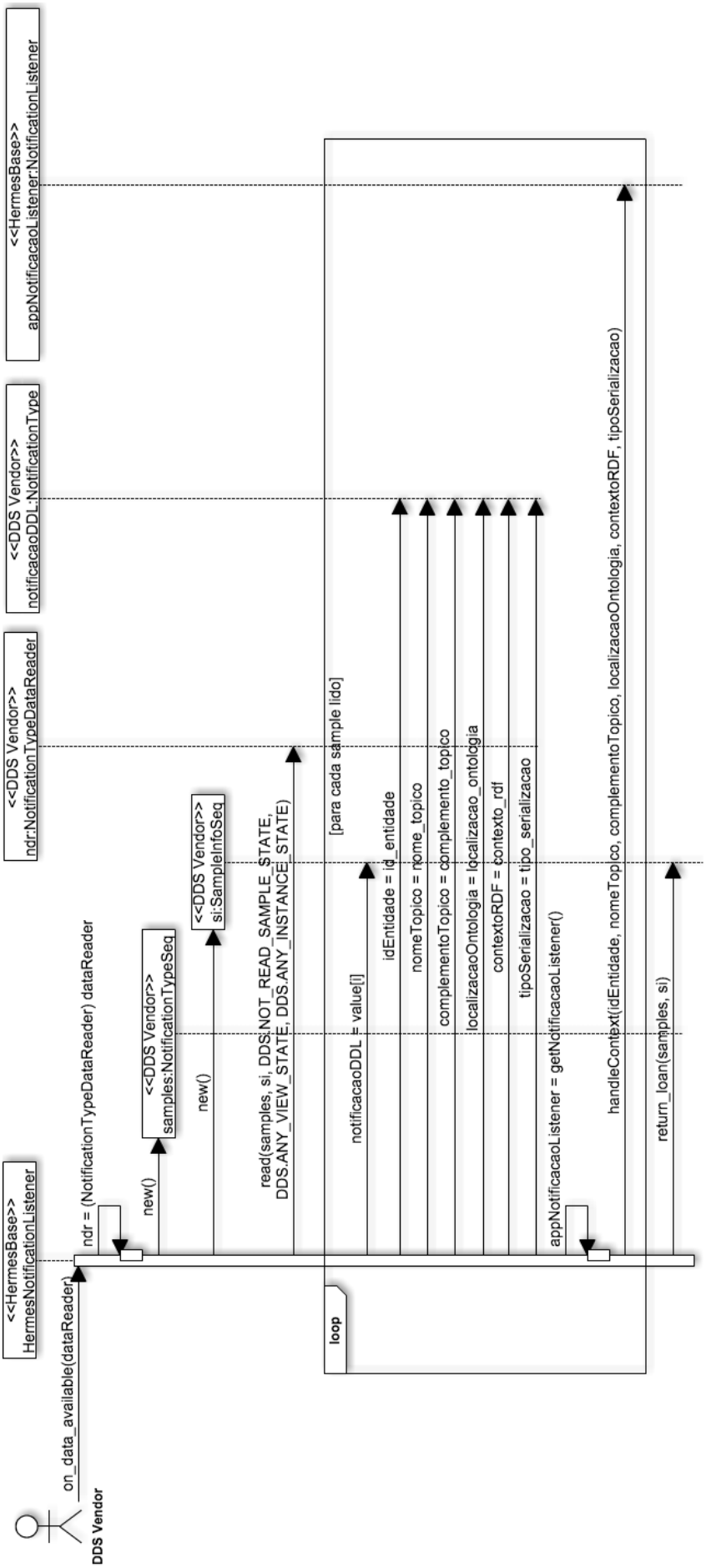


**Figura A.10:** *Listener de Notificação invocado pelo DDS*

APÊNDICE B

# Casos de teste de Hermes Base e Hermes Interpreter

**Tabela B.1:** *Hermes Base: Criar Tópico, cenário 1*

| | |
|---|---|
| **Caso de teste** | Criar tópico de notificação |
| **Pré-condições** | O tipo de dado *DDL* para notificação já deve ter sido criado |
| **Entrada** | Nome do tópico |
| **Observações** | A saída do caso de teste será exibida no console de execução do componente que requisitou a operação |
| **Pontos de controle** | Não se aplica |
| **Resultados esperados** | **Positivo:**<br>1) O tópico é criado pelo middleware DDS<br><br>**Negativo**:<br>1) O tópico não é criado pelo middleware DDS |
| **Pós-condições** | **Positivo:**<br>1) Hermes Base está apto para publicar e assinar em tópico recém-criado<br><br>**Negativo:**<br>1) Não poderá haver publicações e assinaturas no tópico |

**Tabela B.2:** *Hermes Base: Criar Tópico, cenário 2*

| | |
|---|---|
| **Caso de teste** | Criar tópico de filtragem |
| **Pré-condições** | O tipo de dado *DDL* para filtragem já deve ter sido criado |
| **Entrada** | Nome do tópico |
| **Observações** | A saída do caso de teste será exibida no console de execução do componente que requisitou a operação |
| **Pontos de controle** | Não se aplica |
| **Resultados esperados** | **Positivo:**<br>1) O tópico é criado pelo middleware DDS<br><br>**Negativo**:<br>1) O tópico não é criado pelo middleware DDS |
| **Pós-condições** | **Positivo:**<br>1) Hermes Base está apto para publicar e assinar em tópico recém-criado<br><br>**Negativo:**<br>1) Não poderá haver publicações e assinaturas no tópico |

**Tabela B.3:** *Hermes Base: Criar Tópico, cenário 3*

| | |
|---|---|
| **Caso de teste** | Criar tópico de configuração |
| **Pré-condições** | O tipo de dado *DDL* para configuração já deve ter sido criado |
| **Entrada** | Nome do tópico |
| **Observações** | A saída do caso de teste será exibida no console de execução do componente que requisitou a operação |
| **Pontos de controle** | Não se aplica |
| **Resultados esperados** | **Positivo:**<br>1) O tópico é criado pelo middleware DDS<br><br>**Negativo**:<br>1) O tópico não é criado pelo middleware DDS |
| **Pós-condições** | **Positivo:**<br>1) Hermes Base está apto para publicar e assinar em tópico recém-criado<br><br>**Negativo:**<br>1) Não poderá haver publicações e assinaturas no tópico |

**Tabela B.4:** *Hermes Base: Assinar tópico, cenário 1*

| | |
|---|---|
| **Caso de teste** | Assinar tópico de notificação sem filtro |
| **Pré-condições** | O tópico assinado já deve ter sido criado |
| **Entrada** | Nome do tópico |
| **Observações** | A saída do caso de teste será exibida no console de execução do componente que requisitou a operação |
| **Pontos de controle** | Não se aplica |
| **Resultados esperados** | **Positivo:**<br>1) O tópico é assinado pelo middleware DDS<br><br>**Negativo**:<br>1) O tópico não é assinado pelo middleware DDS |
| **Pós-condições** | **Positivo:**<br>1) O assinante será notificado das publicações no referido tópico<br><br>**Negativo:**<br>1) O assinante não será notificado das publicações no referido tópico |

**Tabela B.5:** *Hermes Base: Assinar tópico, cenário 2*

| | |
|---|---|
| **Caso de teste** | Assinar tópico de notificação com filtro |
| **Pré-condições** | O tópico assinado já deve ter sido criado |
| **Entrada** | Nome do tópico e filtro |
| **Observações** | A saída do caso de teste será exibida no console de execução do componente que requisitou a operação |
| **Pontos de controle** | Hermes Base publica em tópico de filtragem para notificar Hermes Interpreter sobre a necessidade de criar filtro para assinante |
| **Resultados esperados** | **Positivo:**<br>1) O tópico é assinado pelo middleware DDS<br>2) O filtro é criado pelo componente Hermes Interpreter<br>**Negativo**:<br>1) O tópico não é assinado pelo middleware DDS<br>2) O filtro não é criado por Hermes Interpreter |
| **Pós-condições** | **Positivo:**<br>1) O assinante será notificado das publicações no referido tópico, quando houver publicações nele, que atendam ao filtro especificado<br>2) O assinante será notificado das publicações no referido tópico, quando houver publicações nele, que atendam ao filtro especificado<br>**Negativo:**<br>1) O assinante não será notificado das publicações no referido tópico<br>2) O assinante não será notificado das publicações no referido tópico<br><br>*Nota:* Para todas as pós-condições, executar casos de teste B.6, B.7, B.8 e B.9 |

**Tabela B.6:** *Hermes Intepreter: Criar filtro semântico, cenário 1*

| | |
|---|---|
| **Caso de teste** | Criar filtro que não necessite de inferência para execução |
| **Pré-condições** | O tópico *createFilter* deve ter sido assinado |
| **Entrada** | Dados de contexto do assinante: *hashmap* do filtro, serializado em array de bytes, com nenhum parâmetro que exija algum tipo de inferência; o tópico assinado e o *UUID* do assinante |
| **Observações** | A saída do caso de teste será exibida no console de execução do Hermes Interpreter |
| **Pontos de controle** | Não se aplica |
| **Resultados esperados** | **Positivo:**<br>1) O filtro é criado e incluído na lista de filtros que não exigem inferência para o respectivo tópico<br><br>**Negativo**:<br>1) O arquivo json de templates de filtro para o referido tópico não é localizado<br>2) O filtro é incluído em lista que exige inferência |
| **Pós-condições** | **Positivo:**<br>1)O componente aguarda novas notificações de Hermes Base<br><br>**Negativo:**<br>1) O sistema retorna mensagem de que arquivo não foi localizado<br>2) A execução da consulta ocorrerá após a etapa de inferência, o que é desnecessário para o filtro. Executar testes B.11 e B.12 |

**Tabela B.7:** *Hermes Intepreter: Criar filtro semântico, cenário 2*

| | |
|---|---|
| **Caso de teste** | Criar filtro que necessite de inferência para execução |
| **Pré-condições** | O tópico *createFilter* deve ter sido assinado |
| **Entrada** | Dados de contexto do assinante: *hashmap* do filtro, serializado em array de bytes, com pelo menos um parâmetro que exija algum tipo de inferência; o tópico assinado e o *UUID* do assinante |
| **Observações** | A saída do caso de teste será exibida no console de execução do Hermes Interpreter |
| **Pontos de controle** | Não se aplica |
| **Resultados esperados** | **Positivos:**<br>1) O filtro é criado e incluído na lista de filtros que exigem inferência para o respectivo tópico<br><br>**Negativos**:<br>1) O arquivo json de templates de filtro para o referido tópico não é localizado<br>2) O filtro é incluído em lista que não exige inferência |
| **Pós-condições** | **Positivos:**<br>1)O componente aguarda novas notificações de Hermes Base<br><br>**Negativos:**<br>1) O sistema retorna mensagem de que arquivo não foi localizado<br>2) A execução da consulta não retornará o resultado esperado, pois o modelo de contexto não conterá os dados resultantes da etapa de inferência. Executar testes B.11 e B.12 |

**Tabela B.8:** *Hermes Intepreter: Criar filtro semântico, cenário 3*

| | |
|---|---|
| **Caso de teste** | Criar filtro que não altera técnica de inferência configurada para tópico |
| **Pré-condições** | O tópico *createFilter* deve ter sido assinado |
| **Entrada** | Dados de contexto do assinante: *hashmap* do filtro, serializado em array de bytes, sendo que nenhum parâmetro provoque alteração em técnica de inferência configurada para o tópico; o tópico assinado e o *UUID* do assinante |
| **Observações** | A saída do caso de teste será exibida no console de execução do Hermes Interpreter e comprovada no arquivo json do tópico |
| **Pontos de controle** | Não se aplica. |
| **Resultados esperados** | **Positivos:**<br>1) O filtro é criado e incluído na lista de filtros para o respectivo tópico<br>2) A técnica de inferência configurada para o filtro não é alterada<br><br>**Negativos**:<br>1) O arquivo json de templates de filtro para o referido tópico não é localizado<br>2) O componente altera a técnica de inferência configurada o tópico |
| **Pós-condições** | **Positivos:**<br>1) O componente aguarda novas notificações de Hermes Base<br>2) O componente continuará a utilizar a técnica de inferência previamente configurada para o tópico em questão<br><br>**Negativos:**<br>1) O sistema retorna mensagem que arquivo não foi localizado<br>2) A execução da consulta possivelmente não retornará o resultado esperado, ou seja, o dado solicitado não será localizado no modelo RDF. Executar testes B.11 e B.12 |

**Tabela B.9:** *Hermes Intepreter: Criar filtro semântico, cenário 4*

| | |
|---|---|
| **Caso de teste** | Criar filtro que altera técnica de inferência configurada para tópico |
| **Pré-condições** | O tópico *createFilter* deve ter sido assinado. A tópico deve estar configurado com técnica de inferência *ontológica* |
| **Entrada** | Dados de contexto do assinante: *hashmap* do filtro, serializado em array de bytes, com pelo menos um parâmetro que provoque alteração em técnica de inferência configurada para o tópico; o tópico assinado e o *UUID* do assinante |
| **Observações** | A saída do caso de teste será exibida no console de execução do Hermes Interpreter e comprovada no arquivo json do tópico |
| **Pontos de controle** | Se a técnica de inferência exigida para o filtro for mais abrangente do que a registrada para o tópico, o componente deve alterar a configuração de inferência para o tópico |
| **Resultados esperados** | **Positivos:**<br>1) O filtro é criado e incluído na lista de filtros para o respectivo tópico<br>2) A técnica de inferência configurada para o filtro é alterada<br><br>**Negativos**:<br>1) O arquivo json de templates de filtro para o referido tópico não é localizado<br>2) O componente não altera a técnica de inferência configurada o tópico<br>3) O componente altera a configuração para a técnica de inferência errada para o tópico |
| **Pós-condições** | **Positivos:**<br>1) O componente aguarda novas notificações de Hermes Base<br>2) As notificações subsequentes para o referido tópico serão inferidas com a nova técnica de inferência configurada<br><br>**Negativos:**<br>1) O sistema retorna mensagem que arquivo não foi localizado<br>2) A execução da consulta possivelmente não retornará o resultado esperado, ou seja, o dado solicitado não será localizado no modelo RDF. Executar testes B.11 e B.12<br>3) A execução da consulta possivelmente não retornará o resultado esperado, ou seja, o dado solicitado não será localizado no modelo RDF. Executar testes B.11 e B.12 |

**Tabela B.10:** *Hermes Base: Publicar contexto*

| | |
|---|---|
| **Caso de teste** | Publicar contexto em tópico de notificação |
| **Pré-condições** | O tópico assinado já deve ter sido criado |
| **Entrada** | Contexto referente à medida de sinal vital |
| **Observações** | A saída do caso de teste será exibida no console de execução do componente que requisitou a operação |
| **Pontos de controle** | Não se aplica |
| **Resultados esperados** | **Positivo:**<br>1) O tópico é publicado pelo middleware DDS<br>**Negativo**:<br>1) O tópico não é publicado pelo middleware DDS |
| **Pós-condições** | **Positivo:**<br>1) O assinante é notificado da publicação, caso o filtro solicitado tenha sido atendido<br>**Negativo:**<br>1) O assinante não é notificado da publicação no referido tópico<br><br>*Nota:* Para todas as pós-condições, executar casos de teste B.11, B.12, B.13 e B.14 |

**Tabela B.11:** *Hermes Intepreter: Inferir situações de contexto, cenário 1*

| | |
|---|---|
| **Caso de teste** | Inferir situações de contexto com técnica *ontológica* |
| **Pré-condições** | O tópico referente ao *sinal vital* notificado deve ter sido assinado. O tópico deve estar configurado com técnica de inferência *ontológica* |
| **Entrada** | Dados de contexto do sinal vital |
| **Observações** | A saída do caso de teste será exibida no console de execução do Hermes Interpreter e das aplicações notificadas |
| **Pontos de controle** | Não se aplica |
| **Resultados esperados** | **Positivos:**<br>1) Os assinantes cujos filtros contemplem as informações de contexto inferidas são notificados<br><br>**Negativos:**<br>1) O template SPARQL está mal formatado<br>2) O tipo de dado requerido para o filtro não é suportado pelo componente<br>3) O template SPARQL não está atualizado com o schema ontológico |
| **Pós-condições** | **Positivos:**<br>1) Hermes Interpreter aguarda novas notificações de Hermes Base<br><br>**Negativos:**<br>1) A consulta do filtro retornará um erro quando for executada<br>2) A consulta do filtro possivelmente retornará um erro quando for executada, pois o parâmetro do filtro é diferente do descrito no schema ontológico<br>3) A consulta do filtro possivelmente retornará um erro quando for executada |

**Tabela B.12:** *Hermes Intepreter: Inferir situações de contexto, cenário 2*

| | |
|---|---|
| **Caso de teste** | Inferir situações de contexto com técnica baseada em *regras* |
| **Pré-condições** | O tópico referente ao *sinal vital* notificado deve ter sido assinado. O tópico deve estar configurado com técnica de inferência *regras* |
| **Entrada** | Dados de contexto do sinal vital |
| **Observações** | A saída do caso de teste será exibida no console de execução do Hermes Interpreter e das aplicações notificadas |
| **Pontos de controle** | Não se aplica |
| **Resultados esperados** | **Positivos:**<br>1) Os assinantes cujos filtros contemplem as informações de contexto inferidas são notificados<br><br>**Negativos:**<br>1) O template SPARQL está mal formatado<br>2) O tipo de dado requerido para o filtro não é suportado pelo componente<br>3) O template SPARQL não está atualizado com o schema ontológico |
| **Pós-condições** | **Positivos:**<br>1) Hermes Interpreter aguarda novas notificações de Hermes Base<br><br>**Negativos:**<br>1) A consulta do filtro retornará um erro quando for executada<br>2) A consulta do filtro possivelmente retornará um erro quando for executada, pois o parâmetro do filtro será diferente do descrito no schema ontológico<br>3) A consulta do filtro possivelmente retornará um erro quando for executada |

**Tabela B.13:** *Hermes Intepreter: Inferir situações de contexto, cenário 3*

| | |
|---|---|
| **Caso de teste** | Inferir situações de contexto após alteração de schema ontológico |
| **Pré-condições** | O tópico referente ao *sinal vital* notificado deve ter sido assinado. O atributo *atualizada*, no arquivo *topics.json* referente à ontologia do tópico, deve estar com valor *false*. |
| **Entrada** | Dados de contexto do sinal vital |
| **Observações** | A saída do caso de teste será exibida no console de execução do Hermes Interpreter e das aplicações notificadas. A atualizacão do status da ontologia poderá ser verificada no arquivo *topics.json* |
| **Pontos de controle** | Após refatoração dos filtros semânticos, atributo *atualizada* referente à ontologia é atualizado para *true* |
| **Resultados esperados** | **Positivos:**<br>1) Os assinantes cujos filtros contemplem as informações de contexto inferidas são notificados.<br>2) Os filtros associados aos tópicos da ontologia atualizada foram refeitos<br>3) O atributo *atualizada* da ontologia no arquivo *topics.json* foi alterado para *true*<br><br>**Negativos:**<br>1) O template SPARQL está mal formatado<br>2) O tipo de dado requerido para o filtro não é suportado pelo componente<br>3) O template SPARQL não está atualizado com o schema ontológico |
| **Pós-condições** | **Positivos:**<br>1) Hermes Interpreter aguarda novas notificações de Hermes Base<br>2) Os novos contextos notificados para os tópicos da ontologia atualizada serão filtrados com a nova versão dos respectivos filtros. Testar casos de teste B.11 e B.12<br>3) Até que o status da ontologia não seja atualizado para *false*, o componente não solicitará que os filtros dos tópicos associados à ontologia sejam refeitos<br><br>**Negativos:**<br>1) A consulta do filtro retornará um erro quando for executada<br>2) A consulta do filtro possivelmente retornará um erro quando for executada, pois o parâmetro do filtro será diferente do descrito no schema ontológico<br>3) A consulta do filtro possivelmente retornará um erro quando for executada |

**Tabela B.14:** *Hermes Intepreter: Inferir situações de contexto, cenário 4*

| | |
|---|---|
| **Caso de teste** | Inferir situações de contexto após inclusão de novo schema ontológico |
| **Pré-condições** | O tópico referente a nova ontologia deve ter sido assinado. O tópico e a localização da nova ontologia devem estar registrados no arquivo *topics.json* |
| **Entrada** | Dados de contexto referentes ao novo tópico. Dentre essas informações, deve constar a localização da nova ontologia |
| **Observações** | A saída do caso de teste será exibida no console de execução do Hermes Interpreter e das aplicações notificadas |
| **Pontos de controle** | Não se aplica. |
| **Resultados esperados** | **Positivos:**<br>1) Os assinantes cujos filtros contemplem as informações de contexto inferidas são notificados<br><br>**Negativos:**<br>1) O template SPARQL está mal formatado<br>2) O tipo de dado requerido para o filtro não é suportado pelo componente<br>3) O template SPARQL não está atualizado com o schema ontológico |
| **Pós-condições** | **Positivos:**<br>1) Hermes Interpreter aguarda novas notificações de Hermes Base<br><br>**Negativos:**<br>1) A consulta do filtro retornará um erro quando for executada<br>2) A consulta do filtro possivelmente retornará um erro quando for executada, pois o parâmetro do filtro será diferente do descrito no schema ontológico<br>3) A consulta do filtro possivelmente retornará um erro quando for executada |