UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

ERNESTO FONSECA VEIGA

Serviços de Representação, Agregação e Histórico de Contexto Ontológico em Apoio ao Desenvolvimento de Aplicações

Goiânia
2016

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR AS TESES E DISSERTAÇÕES ELETRÔNICAS 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), regulamentada pela Resolução CEPEC nº 832/2007, 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

Nome completo do autor: Ernesto Fonseca Veiga

Título do trabalho: Serviços de Representação, Agregação e Histórico de Contexto Ontológico em Apoio ao Desenvolvimento de Aplicações

3. Informações de acesso ao documento:

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF da tese ou dissertação.

Ernesto Fonseca Veiga

_____

Assinatura do (a) autor (a)

Data: 01 / 09 / 2016

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

AUTORIZAÇÃO PARA PUBLICAÇÃO DE DISSERTAÇÃO EM FORMATO ELETRÔNICO

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: Serviços de Representação, Agregação e Histórico de Contexto Ontológico em Apoio ao Desenvolvimento de Aplicações

Autor(a): Ernesto Fonseca Veiga

Goiânia, 19 de Julho de 2016.

Ernesto Fonseca Veiga – Autor

Dr. Renato de Freitas Bulcão Neto – Orientador

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Veiga, Ernesto Fonseca
Serviços de representação, agregação e histórico de contexto ontológico em apoio ao desenvolvimento de aplicações [manuscrito] / Ernesto Fonseca Veiga. - 2016.
CXII, 112 f.: il.

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, 2016.
Bibliografia. Apêndice.
Inclui siglas, gráfico, tabelas, algoritmos, lista de figuras, lista de tabelas.

1. Sistema de Gerenciamento de Contexto. 2. Serviços. 3. Ontologia. 4. Representação . 5. Agregação e Histórico. I. de Freitas Bulcão Neto, Renato, orient. II. Título.

CDU 004

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UFG

ATA Nº 12/2016

ATA DA SESSÃO DE JULGAMENTO DA DISSERTAÇÃO DE MESTRADO DE ERNESTO FONSECA VEIGA

Aos dezenove dias do mês de julho de dois mil e dezesseis, às catorze horas, na sala 151 do Instituto de Informática da Universidade Federal de Goiás, Campus Samambaia, reuniu-se a banca examinadora designada na forma regimental pela Coordenação do Curso para julgar a dissertação de mestrado intitulada **"Serviços de Representação, Agregação e Histórico de Contexto Ontológico em Apoio ao Desenvolvimento de Aplicações"**, apresentada pelo aluno Ernesto Fonseca Veiga como parte dos requisitos necessários à obtenção do grau de Mestre em Ciência da Computação, área de concentração Ciência da Computação. A banca examinadora foi presidida pelo orientador do trabalho de dissertação, Professor Doutor Renato de Freitas Bulcão Neto (INF/UFG), tendo como membros os Professores Doutores Fernando Antonio Mota Trinta (DC/UFC) e Fábio Moreira Costa (INF/UFG). Aberta a sessão, o candidato expôs seu trabalho. Em seguida, o aluno foi arguido pelos membros da banca e:

(X) tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **aprovação** do candidato, sem restrições.

( ) tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **aprovação** do candidato, condicionado a satisfazer as exigências listadas na Folha de Modificação de Dissertação de Mestrado anexa à presente ata, no prazo máximo de 60 dias, a contar da presente data, ficando o professor-orientador responsável por atestar o cumprimento dessas exigências.

( ) não tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **reprovação** do candidato.

Os trabalhos foram encerrados às 17 horas. Nos termos do Regulamento Geral dos Cursos de Pós-Graduação desta Universidade, lavrou-se a presente ata que, lida e julgada conforme, segue assinada pelos membros da banca examinadora.

Prof. Dr. Renato de Freitas Bulcão Neto Renato Bulcão Neto

Prof. Dr. Fernando Antonio Mota Trinta Fernando Antonio Mota Trinta

Prof. Dr. Fábio Moreira Costa

ERNESTO FONSECA VEIGA

Serviços de Representação, Agregação e Histórico de Contexto Ontológico em Apoio ao Desenvolvimento de Aplicações

Dissertação 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 Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Renato de Freitas Bulcão Neto

Goiânia
2016

ERNESTO FONSECA VEIGA

Serviços de Representação, Agregação e Histórico de Contexto Ontológico em Apoio ao Desenvolvimento de Aplicações

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 19 de Julho de 2016, pela Banca Examinadora constituída pelos professores:

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

Prof. Dr. Fábio Moreira Costa
Universidade Federal de Goiás – UFG

Prof. Dr. Fernando Antonio Mota Trinta
Universidade Federal do Ceará – UFC

Ernesto Fonseca Veiga

Bacharel em Informática pelo Instituto Federal de Goiás e Mestre em Ciência da Computação, pela Universidade Federal de Goiás. Sua pesquisa teve foco em serviços para gerenciamento de contexto baseado em ontologias e Engenharia de Software. Atualmente é pesquisador e desenvolvedor no MediaLab-L3P/UFG. Suas áreas de interesse incluem Web Semântica, Engenharia de Software e Computação Sensível a Contexto.

Dedico este trabalho à minha família, de modo especial aos meus pais, primeiros responsáveis pela minha formação, que me ensinaram ainda criança a importância dos estudos, me incentivando no caminho até aqui.

Agradecimentos

À Deus, pela vida e por tudo que conquistei até aqui.

À minha família por ser a base da minha educação e por sempre me apoiar em meus objetivos. Aos meus pais, Raimundo e Eva, por todos os ensinamentos e exemplos, sem os quais eu não teria chegado a este momento. E aos meus irmãos, Estevão e Jaqueline, pelo companheirismo e amizade. Sou muito grato a vocês, e tenho orgulho de fazer parte desta família.

Ao meu amor, e antes de tudo minha amiga, Karlla Loane, por me dar suporte nessa jornada e nos momentos em que mais precisei. Obrigado por estar ao meu lado sempre, fazendo os meus dias mais especiais.

Ao meu orientador, Prof. Dr. Renato de Freitas Bulcão Neto, por todo o incentivo, conselhos e ensinamentos. Além de me proporcionar formação e capacitação para conclusão deste trabalho, por meio das nossas conversas e reuniões pude crescer tanto no âmbito acadêmico quanto pessoal. Tenho hoje um grande amigo e exemplo para a vida.

Aos meus amigos e colegas de pesquisa, que compartilharam comigo as experiências do mestrado, contribuindo com o êxito deste trabalho: Cleon Xavier, Guilherme Maranhão, Guilherme Marques, Heyde Francielle, Márcio Sena e Marcos Alves.

Aos meus tios Geraldo e Noélia, por sempre me abrirem as portas, me apoiando nesse projeto mesmo antes de seu início.

Aos orientadores e colegas de trabalho do L3P (UFG), pelo apoio na reta final da minha pesquisa. Agradeço a Luís Neto e aos professores Marcel e Dalton pela ajuda sempre pronta.

Aos profissionais de enfermagem do HC/UFG, pelas informações e conhecimento compartilhados, e a todos que contribuíram de alguma forma para esta pesquisa.

Resumo

Veiga, Ernesto Fonseca. **Serviços de Representação, Agregação e Histórico de Contexto Ontológico em Apoio ao Desenvolvimento de Aplicações**. Goiânia, 2016. 107p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Sistemas de Gerenciamento de Contexto (CMS) são responsáveis pelo processamento do contexto ao longo do seu ciclo de vida, que é realizado em quatro etapas definidas pela literatura: aquisição, modelagem, raciocínio e disseminação. A etapa de modelagem possui grande relevância neste processo, sendo responsável por transformar os dados adquiridos de sensores em contexto de alto nível. Porém, esta etapa tem sido impactada pela crescente implantação de sensores e o grande volume de informações que são disponibilizadas, gerando a necessidade de prover serviços que realizem a manipulação dessas informações, oferecendo representação adequada e apoio ao desenvolvimento de aplicações. Neste sentido, a representação baseada em ontologias apresenta grandes vantagens, entre as quais destacam-se: alta expressividade, padronização e interoperabilidade. Diante dos requisitos destacados pela literatura, este trabalho apresenta a construção dos serviços relacionados à etapa de modelagem do ciclo de vida do contexto: representação, agregação e histórico, baseados em ontologias. Estes serviços foram projetados e implementados como componentes que integram um CMS denominado *Hermes*. Sendo assim, compõem o escopo deste trabalho: *i*) *Hermes Widget*: representação ontológica de contexto independente de domínio, baseada no padrão *Stimulus-Sensor-Observation*; *ii*) *Hermes Aggregator*: agregação de contexto representado ontologicamente e conversão do contexto para domínios de aplicação; e *iii*) *Hermes History*: histórico de contexto para os diferentes níveis de expressividade produzidos pelos demais componentes. Estes serviços foram validados e avaliados em um cenário de monitoramento de sinais vitais humanos.

Palavras–chave

Sistema de Gerenciamento de Contexto, Serviços, Ontologia, Representação, Agregação, Histórico

Abstract

Veiga, Ernesto Fonseca. **Ontological Context Representation, Aggregation and History Services for Supporting Application Development**. Goiânia, 2016. 107p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

Context Management Systems (CMS) are responsible for processing the context throughout its life cycle, which is performed in four steps defined in the literature: acquisition, modeling, reasoning and dissemination. The modeling step has great relevance in this process, being responsible for transforming the data acquired from sensors in high-level context. However, this step has been impacted by the increasing deployment of sensors and the large volume of information that is available, creating the need to provide services to carry out the handling of such information, providing adequate representation and supporting the development of applications. In this sense, the ontology-based representation has major advantages, including: high expressiveness, standardization and interoperability. Before highlighted requirements by the literature, this work presents the construction of services related to the context life cycle modeling step: representation, aggregation and history, based on ontologies. These services have been designed and implemented as components that integrate a CMS called Hermes. Therefore, the scope of this work is composed by: i) *Hermes Widget*: ontological representation of independent domain context, based on *Stimulus-Sensor-Observation* standard; ii) *Hermes Aggregator*: ontologically represented context aggregation and its conversion for application domains; and iii) *Hermes History*: context history for the different levels of expression produced by other components. These services have been validated and evaluated in a human vital signs monitoring scenario.

Keywords

Context Management System, Services, Ontology, Representation, Aggregation, History

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Códigos de Programas

Lista de Siglas

API	*Application Programming Interface*
CMS	*Context Management Systems*
CSV	*Comma-separated values*
DDL	*Data Definition Language*
DDS	*Data Distribution Service*
IEC	*International Electrotechnical Commission*
IRI	*Internationalized Resource Identifier*
ISO	*International Organization for Standardization*
JSON	*JavaScript Object Notation*
HA	*Hermes Aggregator*
HB	*Hermes Base*
HH	*Hermes History*
HI	*Hermes Interpreter*
HW	*Hermes Widget*
MIMIC	*Multiparameter Intelligent Monitoring in Intensive Care*
MSVH	Monitoramento de Sinais Vitais Humanos
OMG	*Object Management Group*
OWL	*Web Ontology Language*
POJO	*Plain Old Java Object*
RDF	*Resource Description Framework*
RDFS	*Resource Description Framework Schema*
SEVOCAB	*Software and Systems Engineering Vocabulary*
SIG	*Software Improvement Group*
SSN	*Semantic Sensor Network*
SSO	*Stimulus-Sensor-Observation*
SPARQL	*Simple Protocol and RDF Query Language*
SQL	*Structured Query Language*
SWRL	*Semantic Web Rule Language*
Turtle	*Terse RDF Triple Language*
UTI	Unidade de Terapia Intensiva
VSO	*Vital Sign Ontology*
W3C	*World Wide Web Consortium*
XML	*Extensible Markup Language*

Lista de Publicações

A seguir é apresentada a lista de publicações já realizadas a partir deste trabalho:

1. VEIGA, E. F.; BULCÃO-NETO, R. F. **Um Serviço de Representação Ontológica de Contexto Baseada no Padrão de Projeto Estímulo-Sensor-Observação** In: *8° Simpósio Brasileiro de Computação Ubíqua e Pervasiva*, p. 1–10, 2016.

2. VEIGA, E. F.; MARANHÃO, G. M.; BULCÃO-NETO, R. F. **Development and scalability evaluation of an ontology-based context representation service.** *IEEE Latin America Transactions*, 14(3):1372–1379, March 2016.

3. VEIGA, E. F.; BULCÃO-NETO, R. F. **An ontological approach for information management of public transport networks.** In: *Proceedings of the Annual Conference on Brazilian Symposium on Information Systems: A Computer Socio-Technical Perspective - Volume 1*, p. 66:493–66:500, 2015.

4. VEIGA, E. F.; SENA, M. V. O.; BULCÃO-NETO, R. F. **Padrões, ferramentas e boas práticas no desenvolvimento de software para web semântica.** *XV Simpósio Brasileiro de Sistemas de Informação*, p. 25–31, 2015.

5. VEIGA, E. F.; MARANHÃO, G. M.; BULCÃO-NETO, R. F. **Apoio ao desenvolvimento de aplicações de tempo real sensíveis a contexto semântico.** *IX Workshop de Teses e Dissertações do XX Simpósio Brasileiro de Sistemas Multimídia e Web*, p. 1–4, 2014. (*Best Paper*)

Além dos trabalhos já publicados, o seguinte artigo foi aceito e se encontra no estado de *major revisions*:

- VEIGA, E. F.; BULCÃO-NETO, R. F. **A Systematic Review on the Modeling Phase of Context Life Cycle: Ontology-based Approach.** *IEEE Latin America Transactions*, 2016.

CAPÍTULO 1

Introdução

A *Computação Sensível ao Contexto* é a área de pesquisa que estuda os mecanismos para fornecimento e utilização do contexto a fim de oferecer serviços e informações relevantes a usuários e a aplicações na realização de alguma tarefa [12]. A sensibilidade ao contexto (*context awareness*) é considerada o núcleo dos sistemas de computação ubíqua [56, 39].

Várias definições de contexto foram utilizadas na literatura por pesquisadores, sendo uma das mais aceitas aquela cunhada por Anind Dey, oficialmente apresentada em sua tese e em seu trabalho clássico de 2001, intitulado "*A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications*" [11, 12]:

> *"Contexto é qualquer informação que possa ser usada para caracterizar a situação de uma entidade, sendo que essa entidade pode ser uma pessoa, lugar ou objeto considerado relevante para a interação entre o usuário e a aplicação, incluindo o próprio usuário e a aplicação."*

Além de definir contexto, Dey et al. [12] também foram precursores em elencar os requisitos para a construção de software que o utiliza. A identificação dos problemas básicos quanto à especificação, aquisição e ação em relação ao contexto, juntamente com uma análise minuciosa do processo de projeto de aplicações, permitiram a definição e implementação de um *framework* precursor no apoio ao desenvolvimento de aplicações sensíveis ao contexto, o *Context Toolkit*.

O arcabouço criado por Dey et al. [12] trouxe contribuições quanto ao projeto, desenvolvimento e evolução de aplicações sensíveis ao contexto [54]. Segundo os autores, a utilização de uma infraestrutura de apoio para lidar com os detalhes de aquisição, modelagem, raciocínio e disseminação do contexto, auxilia na redução da complexidade do desenvolvimento de aplicações.

A ebulição das pesquisas nesta área, impulsionada por contribuições como as do *Context Toolkit*, permitiu que novas perspectivas pudessem ser almejadas para os

denominados Sistemas de Gerenciamento de Contexto (CMS[1]) [13, 2, 4]. Estes sistemas, bem como as tecnologias utilizadas na sua construção, estão em constante processo de evolução, alinhando-se a novos requisitos identificados pela literatura e, dessa forma, contribuindo para o avanço em sistemas sensíveis a contexto [7, 39].

1.1 Motivação

Com o desenvolvimento e popularização das tecnologias de computação embarcada, há uma grande facilidade quanto à implantação de sensores para as mais diversas finalidades, tais como: monitoramento de sinais vitais, localização, segurança, trânsito, entretenimento, entre outras [44]. Este cenário, no entanto, produz domínios contextuais cada vez mais complexos, devido à grande distinção *semântica* entre as informações produzidas por esta gama de sensores [39].

Com o aumento da complexidade dos ambientes de contexto, novos requisitos têm sido identificados no intuito de permitir a evolução dos CMS. O *aumento na expressividade das informações de contexto*, *o uso de técnicas de inferência* e *o gerenciamento de histórico de contexto* são exemplos de necessidades destacadas pela literatura [7, 39].

Os projetos de CMS, em geral, adotam um ciclo de vida de contexto que inclui etapas às quais as informações de contexto estão submetidas, desde a aquisição (provedores), até sua entrega às aplicações (consumidores). Perera et al. [39] apresentam uma abordagem geral que sintetiza as principais etapas de ciclo de vida do contexto, como representado na Figura 1.1.

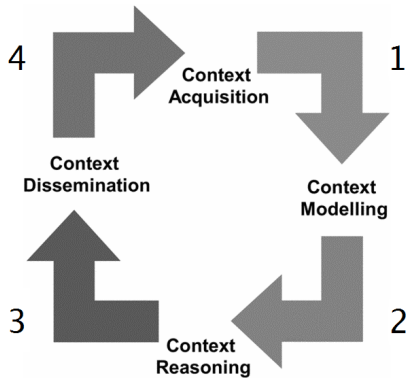


Figura 1.1: *Ciclo de vida do contexto: etapas essenciais [39].*

Na primeira etapa, de *aquisição* (1), os CMS têm sido exigidos a lidar com um número crescente de sensores presentes nos ambientes de contexto. Duas consequências diretas desse cenário, de acordo com Sehic et al. [43] e Perera et al. [39], são o grande

[1]No decorrer do trabalho será utilizada a sigla CMS, adaptada do inglês *Context Management Systems*. O termo *infraestrutura* também será utilizado como equivalente a CMS.

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.

A etapa seguinte, de *modelagem* (2), possui um papel central no ciclo de vida do contexto. Nela, os dados coletados dos provedores de contexto são representados segundo técnicas de modelagem, que agregam significado e permitem que, ao serem processados, esses dados possam gerar informação de contexto de alto nível.

Algumas das principais necessidades identificadas para essa etapa são a *evolução quanto à expressividade*, *formalidade* e *utilização de padrões para a representação de informação contextual* [7]. Representar o contexto e realizar outros serviços como a agregação e histórico em relação a esta quantidade de dados de fontes e domínios heterogêneos tem sido um dos grandes desafios da Computação Sensível ao Contexto [43].

Dentre as principais técnicas de modelagem de contexto apresentadas na literatura [47, 2], são várias as razões para a utilização de ontologias [36, 55, 39], a saber: *i*) compartilham uma interpretação consensual da estrutura das informações, seja entre pessoas ou agentes de software; *ii*) formalizam o conhecimento do domínio; *iii*) separam o conhecimento do domínio (regras de negócio) do conhecimento operacional (implementação); *iv*) permitem a reutilização do conhecimento de um domínio; e *v*) possibilitam a inferência de contexto de alto nível.

A terceira etapa do ciclo de vida de contexto é a de *raciocínio* (3), que está diretamente ligada à etapa de modelagem, uma vez que as técnicas de inferência sobre o contexto dependem de como essa informação foi anteriormente representada. Encerrando este ciclo de vida, a etapa de *disseminação* (4) é responsável pela entrega das informações de contexto relevantes aos consumidores interessados, por exemplo, aplicações ou serviços computacionais.

1.2 Justificativa

Os CMS em geral possuem similaridade quanto aos principais componentes que os constituem, assemelhando-se nas tarefas centrais que realizam, tais como: aquisição, modelagem, armazenamento, raciocínio e disseminação de contexto. Entretanto, esses sistemas se diferenciam nas características específicas que oferecem, tais como: tipo de modelagem adotada, apoio a agregação, histórico de contexto e questões de projeto arquitetural para disponibilização dos serviços [35].

Perera et al. [39], com base na literatura [41, 5, 34, 7], sumarizam os principais princípios de projeto para CMS, a saber: *projeto arquitetural em camadas e componentes*, *escalabilidade e extensibilidade*, *gerenciamento automático do ciclo de vida do contexto*, *independência do modelo de contexto*, *modelagem expressiva e abrangente*, *verificação da consistência das informações de contexto instanciadas em relação ao modelo*, *com-*

partilhamento de informação por meio do fornecimento de histórico de contexto. Estes princípios de projeto estão alinhados com a literatura, que recomenda a modelagem baseada em ontologias [7, 39].

Para explorar a semântica e o alto nível de expressividade da modelagem ontológica de contexto, devem ser levadas em consideração as relações e associações que podem ser geradas pelo CMS a partir dos dados de provedores. A quantidade e o *tipo de processamento* realizado sobre o contexto pode resultar em diferentes tipos de informação ou diferentes *níveis de expressividade do contexto* [46, 45].

Os CMS podem gerenciar diferentes tipos de informação contextual [39], sendo as principais: *i*) representação do contexto obtido de provedores de contexto; *ii*) agregação do contexto de diferentes provedores sobre uma mesma entidade e; *iii*) informações de contexto inferidas por meio de técnicas de raciocínio. O serviço de representação do contexto, reforçando a importância da etapa de modelagem, é a base para o processamento dos demais tipos de informação no CMS. Por exemplo, para que uma agregação seja realizada, é necessário que as informações sejam anteriormente representadas individualmente. Da mesma forma, para que novas informações sejam raciocinadas em relação a um modelo de contexto, elas devem estar devidamente representadas.

Como CMS precursores em utilizar a modelagem ontológica, sistemas como Gaia [42], CoBrA [9] e SOCAM [19] já possuíam também em seu projeto repositórios ou bases de dados para as informações de contexto. Porém, nenhum destes sistemas realiza em seu escopo o gerenciamento do histórico de contexto, não explorando este aspecto que é necessário para que os diferentes tipos de informações de contexto possam ser posteriormente recuperadas e reutilizadas.

Verificando essa necessidade, Hydra [1] e CROCO [35] destacam a importância do tratamento do histórico de contexto em suas arquiteturas. O CMS CROCO fornece um componente para armazenamento do histórico que associa informação temporal ao contexto. Mitschick et al. [35] evidenciam ainda a necessidade de aprimorar os registros de contexto, permitindo que o histórico seja melhor utilizado e aproveitado por aplicações sensíveis ao contexto.

Quanto ao projeto arquitetural de um CMS, cada etapa do ciclo de vida apresentado pode ser desmembrada em vários serviços, que realizam tarefas específicas dentro dos objetivos da etapa principal. A representação e a agregação de contexto são exemplos de serviços da etapa de modelagem. Por outro lado, há ainda tarefas ou serviços que podem permear mais de uma das etapas do ciclo de vida, por exemplo, o gerenciamento do histórico de contexto, que precisa ser ser transversal às etapas de modelagem e raciocínio, tratando as informações geradas em ambas as etapas.

Quanto à disponibilização destes serviços, muitos sistemas e aplicações sensíveis ao contexto implementam suas funcionalidades dispersas por todo o código fonte,

causando dificuldade na sua utilização, entendimento e manutenabilidade. A técnica de *Programação Orientada a Aspectos*, em Engenharia de Software, objetiva modularizar essas funcionalidades em nível de código fonte, permitindo a separação de conceitos. Etapas do gerenciamento de contexto, tais como representação e agregação podem ser modularizadas e encapsuladas como diferentes aspectos de contexto [40].

A partir do exposto é apresentado o problema de pesquisa deste trabalho:

> **Problema:** *Soluções para gerenciamento de contexto e desenvolvimento de aplicações não tratam a expressividade e a padronização das informações coletadas do ambiente em seus diferentes aspectos (representação, agregação e histórico) envolvidos na etapa de modelagem do ciclo de vida do contexto e seus requisitos.*

A modelagem baseada em ontologias pode ser explorada no intuito de padronizar a representação de contexto, independente do domínio, permitindo que diferentes sistemas e aplicações possam consumir essas informações de maneira uniforme e interoperável. A expressividade oferecida por essa técnica de modelagem permite uma descrição detalhada da informação de contexto e possibilita a aplicação de técnicas como raciocínio para inferências de informações de alto nível sobre um determinado contexto.

Complementando e estendendo a representação, a agregação de contexto ontológico permite disponibilizar informações em um nível ainda mais alto de expressividade. Essa técnica facilita o acesso ao contexto por parte das aplicações, que precisam realizar menos interações com um CMS. Por ser baseado em uma representação padronizada, o contexto agregado pode ainda ser facilmente convertido para o domínio de aplicação específico.

O histórico é uma importante técnica utilizada em CMS para permitir consultas e análise sobre as informações de contexto. Com diferentes aspectos de contexto, o histórico deve lidar com os diferentes tipos de informações processadas para possibilitar consultas adequadas às necessidades de sistemas e aplicações que o utilizam.

A pesquisa apresentada neste trabalho, alinhada com a literatura, tem como foco a *etapa de modelagem* do ciclo de vida de contexto, utilizando ontologia como técnica de modelagem, e o desenvolvimento de serviços relacionados a essa etapa. Estes serviços são parte integrante de um CMS denominado *Hermes* [52, 33, 50], apresentado em detalhes na Seção 2.3, que está sendo projetado para fornecer os serviços necessários a todas as etapas do ciclo de vida do contexto.

Previamente a esta pesquisa, foram desenvolvidos dois componentes do CMS *Hermes*: *Hermes Interpreter* [33] e *Hermes Base* [33], responsáveis pelas etapas de raciocínio e disseminação do ciclo de vida do contexto, respectivamente. As contribuições deste trabalho em relação à etapa de modelagem foram adicionadas a *Hermes* e validadas em conjunto com os demais componentes do CMS.

1.3 Objetivos

O objetivo geral deste trabalho consiste na elaboração de uma abordagem baseada em ontologias para apoio ao gerenciamento de contexto e desenvolvimento de aplicações, com foco na etapa de modelagem do ciclo de vida do contexto, abordando os serviços de representação, agregação e histórico das informações de contexto.

Neste intuito, os objetivos específicos da pesquisa são:

- Integrar os serviços de representação, agregação e histórico ao CMS *Hermes* para que possam contribuir no gerenciamento do contexto em relação à etapa de modelagem.
- Validar e/ou avaliar os serviços propostos a partir de um estudo de caso e desenvolvimento de aplicações que os utilizam, no intuito de verificar o cumprimento dos requisitos identificados.

1.4 Metodologia

Esta seção apresenta as atividades realizadas, bem como a metodologia utilizada para alcançar os objetivos deste trabalho.

1. **Revisão Bibliográfica**: é constante durante todas as etapas do trabalho, sendo que, ao início da pesquisa, foi planejada e conduzida uma revisão sistemática sobre a etapa de modelagem do ciclo de vida do contexto, que foi continuada ao decorrer das demais etapas.
    - Os *requisitos* identificados nos trabalhos [12, 7, 39], serviram como ponto de partida para a investigação sobre o ciclo de vida do contexto e Sistemas de Gerenciamento de Contexto (CMS).
    - A *revisão sistemática* da literatura foi conduzida no intuito de identificar princípios de projeto, questões em aberto e trabalhos relacionados. Foram definidos como foco da pesquisa a etapa de modelagem do ciclo de vida do contexto e os serviços relacionados.
2. **Definição do escopo de serviços necessários à proposta**: por meio da revisão sistemática realizada, a *representação de contexto*, a *agregação de contexto* e o *histórico de contexto* foram identificados como principais serviços relacionados à etapa de modelagem.
3. **Definição dos requisitos e projeto arquitetural dos componentes**: com base na literatura revisada, os componentes *Hermes Widget*, *Hermes Aggregator* e *Hermes History* tiveram seus requisitos e projeto arquitetural definidos.

4. **Definição da abordagem e de ferramentas necessárias**: metodologias, tecnologias e padrões adotados na pesquisa.
    - A *modelagem baseada em ontologias* foi adotada nesta pesquisa para representação de contexto. Por meio deste tipo de modelagem, o contexto pode ser representado com *formalismo* através de padrões, o que permite a utilização de técnicas de *raciocínio* sobre o contexto e a sua *interoperabilidade* entre diferentes sistemas.
    - Foi definida a utilização de *padrões* consolidados da Web Semântica tais como RDF, OWL e SPARQL para a representação do contexto, realização de consultas e processamento da semântica baseada em ontologias.
    - Para oferecer uma representação independente de domínio, que expresse as características das informações de contexto coletadas dos sensores, foi adotada a ontologia *Semantic Sensor Network* [10] como padrão para representação de contexto.
5. **Desenvolvimento, validação e avaliação do serviço de representação ontológica de contexto**:
    - Desenvolvimento do componente *Hermes Widget*, que oferece o serviço de representação ontológica do contexto, que é base para os demais componentes [50, 49].
    - Validação da solução no cenário de *monitoramento de sinais vitais humanos*, domínio adotado para a elaboração de casos de teste. Foram desenvolvidos *Hermes Widgets* para diferentes sinais vitais.
    - Avaliação da solução proposta de acordo com características de qualidade definidas pela ISO 25010 [25]: manutenabilidade e escalabilidade.
6. **Desenvolvimento e validação do serviço de agregação de contexto**: desenvolvimento do componente *Hermes Aggregator*, que oferece o serviço de agregação de contexto e sua validação no cenário de monitoramento de sinais vitais.
7. **Desenvolvimento, validação e avaliação do serviço de histórico de contexto**:
    - Desenvolvimento do componente *Hermes History* para gerenciamento do histórico de contexto, que persistirá o contexto produzido pelos demais componentes: HW (representação), HA (agregação) e HI (interpretação).
    - Validação das funcionalidades do componente e avaliação de escalabilidade no intuito de verificar sua capacidade quanto ao gerenciamento e persistência das informações de contexto recebidas.
8. **Desenvolvimento de aplicações sensíveis ao contexto**: foram desenvolvidas duas aplicações sensíveis ao contexto que utilizam os componentes implementados nesta proposta, validando suas funcionalidades.

1.5 Estrutura da Dissertação

Esta dissertação está estruturada em 5 capítulos, além desta introdução. O Capítulo 2 aborda os fundamentos teóricos acerca dos principais assuntos envolvidos neste trabalho. No Capítulo 3 é apresentada a proposta e o desenvolvimento da pesquisa: os requisitos, arquitetura e detalhes de implementação e validação dos serviços de representação, agregação e histórico de contexto. O Capítulo 4 discorre sobre as avaliações realizadas para os componentes *Hermes Widget* e *Hermes History*, também tratando das questões relevantes ao desenvolvimento de aplicações. No Capítulo 5 são apresentados os trabalhos relacionados a esta proposta e uma análise comparativa. E o Capítulo 6, encerra esta dissertação elencando suas contribuições, limitações e trabalhos futuros. Complementando esta pesquisa, o Apêndice A apresenta os detalhes da avaliação de manutenabilidade realizada para o componente *Hermes Widget*.

CAPÍTULO 2

Fundamentação Teórica

A proposta apresentada neste trabalho está diretamente ligada à etapa de modelagem do ciclo de vida do contexto, realizada no CMS *Hermes*. O estado da arte em Computação Sensível a Contexto recomenda os CMS baseados em ontologias, uma vez que suas características apoiam os requisitos evidenciados pela literatura. A fim de tornar explícitos estes conceitos e suas relações, são apresentados neste capítulo: os fundamentos de Web Semântica, tecnologias associadas e ontologias utilizadas na realização desta proposta, e uma visão geral de gerenciamento de contexto e do CMS *Hermes*.

2.1 Web Semântica

A Web Semântica é uma Web de dados descritos e interligados de maneira a se estabelecer um contexto ou semântica que adere a uma linguagem e regras gramaticais bem definidas [6]. Para atingir esse propósito, foram criadas especificações para representar informação na Web de forma padronizada, e desenvolvidos *frameworks*, processadores de consultas e máquinas de inferência com suporte a tais especificações [20].

A proposta da Web Semântica é reestruturar e expandir a Web tradicional, estabelecendo contexto e semântica associados aos dados e às suas relações [51]. Por exemplo, na Web tradicional o termo *professor*, descrito isoladamente dentro de uma página, só possui significado para um ser humano que compreende 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. Dessa forma, o termo *professor* não seria um elemento isolado, mas estaria associado a diversos outros elementos como *curso*, *disciplinas que leciona*, entre outros.

A arquitetura em camadas para a Web Semântica, apresentada na Figura 2.1, foi proposta para que a Web se torne compreensível para sistemas computacionais e seus mecanismos de busca. Cada camada já implementada dessa arquitetura está associada

a padrões, por exemplo: representação de identificação única (IRI), estrutura (XML), sintaxe (RDF), consulta (SPARQL) e semântica de informação (OWL) [20].

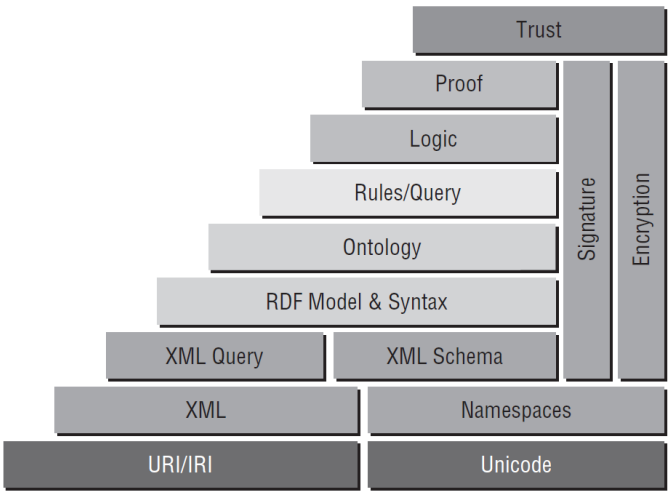


Figura 2.1: *Arquitetura da Web Semântica [20].*

A seguir serão descritas as camadas e padrões necessários para o entendimento da proposta deste trabalho.

2.1.1 Padrão IRI e a Tríade XML

Uma URI (*Uniform Resource Identifier*) corresponde a uma cadeia de caracteres, com uma sintaxe particular, para identificar unicamente um recurso na Web. Entende-se por recurso qualquer porção de conteúdo, seja uma página de texto, um clipe de vídeo ou de áudio, um programa, ou uma imagem.

Atualmente, as URI's são definidas como IRI's [1] (*Internationalized Resource Identifier*), em função de seus caracteres seguirem o padrão internacional Unicode de codificação de caracteres. O principal papel do padrão IRI na Web Semântica é fornecer uma codificação universal para nome e localização de recursos, pois cada recurso na Web é identificado unicamente por meio de sua IRI.

O primeiro elemento da tríade XML é a própria linguagem de marcação XML [2] (*Extensible Markup Language*) que, além de extensível, é flexível, permitindo ao usuário definir suas próprias tags, na ordem desejada e da maneira como quer que sejam processadas ou apresentadas. Todo documento XML é representado em memória como uma árvore de elementos (e atributos, se cabível), e obedece às seguintes regras de sintaxe: *i*) sendo uma árvore, possui uma única raiz, sem ciclos; *ii*) cada nó (exceto o raiz) tem um único pai; *iii*) elementos devem ter uma tag de fechamento; *iv*) tags dos elementos são

[1] *Internet Engineering Task Force*, RFC 3987: http://tools.ietf.org/html/rfc3987
[2] *Extensible Markup Language (XML) 1.0 (Fifth Edition)*: http://www.w3.org/TR/xml/

sensíveis à caixa; *v*) a ordem do aninhamento dos elementos é importante; *vi*) a ordem dos atributos não é relevante; e *vii*) valores dos atributos devem estar entre aspas.

Por causa de suas características, o principal papel do padrão XML na Web Semântica é fornecer suas regras de sintaxe para a construção de outras especificações da arquitetura citada, como RDF, RDFS e OWL. Entretanto, a flexibilidade permitida pela linguagem XML, de que qualquer pessoa, organização ou software possa especificar suas próprias *tags* para descrever conteúdo, pode resultar em duplicações ou conflitos de nomes de tags.

Para resolver estes conflitos, os documentos XML devem fazer uso dos espaços de nomes XML (do inglês XML *namespaces*), segundo membro da tríade, que funcionam como um repositório no qual se define a sintaxe e a estrutura de elementos e atributos de documentos XML. Para isso, deve-se associar um espaço de nomes a uma IRI, e a esse espaço de nomes atribuir um prefixo (ou alias). Na prática, essa associação é realizada por meio da sintaxe `xmlns: prefixo="IRI"`.

Os espaços de nomes XML têm grande importância para integração e compartilhamento de dados na Web Semântica, pois previnem colisões de nomes de recursos do ponto de vista sintático, previnem ambiguidades do ponto de vista semântico e garantem a unicidade dos recursos de uma declaração em escala global, segundo seus nomes e respectivos espaços de nomes.

O terceiro elemento que forma a tríade XML é a especificação esquema XML (do inglês XML *Schema*), uma linguagem para a definição da estrutura de documentos XML (ordem, encadeamento e cardinalidade de elementos, e obrigatoriedade de elementos e atributos), bem como dos tipos de dados armazenados em cada elemento ou atributo. Ou seja, um esquema XML é um documento que especifica as regras gramaticais de validação de um ou mais documentos XML.

O principal papel do esquema XML para a Web Semântica é o de fornecer um conjunto sofisticado e padronizado de tipos de dados para a descrição de conteúdo, usado por outras especificações da arquitetura citada, como RDF e OWL.

2.1.2 *Resource Description Framework* (RDF)

Embora a linguagem XML seja a linguagem franca para estruturação e intercâmbio de dados na Web, há características que a tornam inviável para intercâmbio de dados na Web Semântica [20]:

- A dificuldade de se determinar exatamente o significado da hierarquia entre as *tags*, bem como dos dados que formam o conteúdo dessas *tags*;
- O fato de uma mesma informação poder ser representada de múltiplas maneiras dada a estrutura de árvore de documentos XML.

Portanto, a Web Semântica demanda um modelo de dados que possa representar qualquer informação que se deseje expressar de maneira universal, legível por máquinas e fácil para integrar dados de fontes diferentes [53]. Nesse interim, surge o modelo de dados RDF, que é a especificação padrão do Consórcio W3 (W3C) para intercâmbio de dados na Web Semântica.

O modelo de dados RDF é baseado em triplas, cujo conceito é considerado chave na Web Semântica. Triplas são declarações constituídas por três elementos (*sujeito*, *predicado* e *objeto*), e podem ser mapeadas diretamente para grafos dirigidos, nas quais:

- O *sujeito* representa uma IRI de um *recurso*;
- O *predicado* pode representar uma IRI de uma *propriedade* de um *recurso*, ou uma IRI de um *relacionamento* entre dois recursos;
- O *objeto* pode representar um *valor literal* (quando se liga ao sujeito por meio de um predicado-propriedade), ou uma IRI de um *recurso* (quando se liga ao sujeito por meio de uma propriedade-relacionamento).

Ressalta-se também que todos os predicados descritos em RDF são binários, ligando sempre *um sujeito* a *um objeto*. Conclui-se, pois, que o modelo de dados RDF é voltado para a estruturação e o intercâmbio universal de dados, e não para sua exibição.

A Figura 2.2 apresenta a representação gráfica de uma tripla que tem como sujeito o recurso *Renato*, como predicado a propriedade *leciona* e como objeto o recurso *Engenharia de Software*.

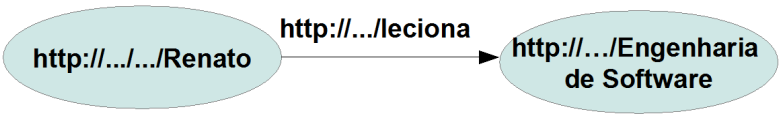


Figura 2.2: *Exemplo de grafo que representa uma tripla [53].*

Chama-se atenção também ao fato de que mesmo que grafos RDF sejam fáceis de visualizar, entender e programar, deve existir uma representação desses grafos para facilitar ainda mais o processamento e o intercâmbio de dados entre aplicações. Ou seja, necessita-se de sintaxes de serialização de dados RDF.

Dentre as várias sintaxes para serializar triplas RDF, destaca-se a serialização *Turtle* (*Terse RDF Triple Language*). Trata-se de uma sintaxe que permite que um grafo RDF seja escrito sob uma forma textual compacta, simples de aprender e compatível com os padrões IRI, espaços de nomes XML e esquema XML. O Código 2.1 ilustra uma representação RDF serializada em Turtle.

Código 2.1 Exemplo de serialização RDF *Turtle* [53].

```
1 @prefix : <http://www.inf.ufg.br/node#>    .
2 @prefix acm: <http://www.acm.org/acmcs#>     .
3 @prefix xsd: <http://w3.org/2001/XMLSchema#> .
4 :Renato :leciona acm:EngenhariaDeSoftware    ;
5         :idade   "40"^^xsd:positiveInteger   .
```

A linha 1 associa um prefixo vazio à IRI que delimita o espaço de nomes nos quais os recursos *Renato* e *leciona* foram definidos. A linha 2 faz o mesmo com o recurso `Engenharia de Software`, cujo espaço de nomes é associado ao prefixo `acm:` . A linha 3 associa o prefixo `xsd:` à IRI do espaço de nomes padrão em que os tipos de dados do esquema XML estão definidos. A linha 4 corresponde à declaração de que "*Renato leciona Engenharia de Software*", enquanto que a linha 5 atribui ao sujeito `Renato` o predicado `idade` com objeto de valor "*40*", a ser interpretado como um número inteiro positivo, segundo a especificação do esquema XML.

O exemplo apresentado pode ser contextualizado neste trabalho como uma descrição de contexto da entidade *Renato*, que tem como propriedades a *idade* e a *disciplina que leciona*. Cada uma das informações possui uma IRI, que a identifica, e faz parte de uma tripla de dados. Dessa forma, o contexto é representado de acordo com o padrão RDF e se torna interoperável entre diferentes sistemas, que necessitam apenas conhecer o modelo de contexto do qual essa informação é uma instância.

2.1.3 *Web Ontology Language* (OWL)

Apesar de fornecer um formato de representação universal para expressar declarações sobre recursos, o padrão RDF não oferece uma linguagem para modelagem da semântica desses recursos [53]. Neste sentido, o vocabulário RDF *Schema* (RDFS) permite expressar a semântica de conceitos, mas ainda é simples para permitir a construção de ontologias de maior expressividade. Para cobrir esta lacuna foi criada a linguagem OWL, que atualmente está em sua segunda versão, chamada OWL2.

Ao fazer uso das especificações IRI, XML, RDF e RDFS, a OWL2 é a linguagem padrão para construção de ontologias para a Web Semântica [20]. Para isso, em acréscimo ao vocabulário da RDFS, a OWL2, por exemplo, classifica as propriedades como *atributos* ou *relações*. Às do tipo atributo, associam-se os tipos de dados definidos no esquema XML; e às do tipo relação, pode-se classificá-las quanto a suas características lógicas, como relações transitivas, simétricas, reflexivas, inversas, funcionais, entre outras. Ainda em comparação à RDFS, a linguagem OWL2 oferece:

- Suporte a restrições sobre valores e cardinalidades de propriedades, como o uso de quantificações universal e existencial e cardinalidades exata, mínima e máxima;
- Suporte à modelagem de classes disjuntas, i.e. aquelas cujos indivíduos não podem ser de ambas as classes;
- Construtores para criar classes segundo a teoria de conjuntos, bem como para declarar equivalência entre classes, propriedades e indivíduos.

Um simples exemplo da expressividade e da capacidade de inferência da linguagem OWL2 é apresentado no Código 2.2.

A linha 4 define que o conceito `Pessoa`, definido no espaço de nomes XML de prefixo `foaf:` , é uma classe de indivíduos. O mesmo acontece na linha 5, i.e. o conceito `Disciplina` é modelado também como uma classe. Nas linhas 6 a 8, o conceito `leciona` é representado como uma propriedade, vinculada a indivíduos da classe `Pessoa`, e indivíduos da classe `Disciplina` como valores possíveis dessa propriedade.

Na linha 9, a relação `leciona` é declarada como equivalente à relação `ministra`: ou seja, afirma-se que sujeitos e objetos interligados pela propriedade `leciona`, deverão ser também pela propriedade `ministra`. Na linha 10 a relação `ehLecionadaPor` é declarada como inversa da relação `leciona`: ou seja, sujeitos e objetos interligados pela propriedade `leciona` serão invertidos na declaração da relação `ehLecionadaPor`. Ao aplicar a semântica desses construtores OWL2 à declaração da Figura 2.2, uma aplicação poderá inferir, além das deduções obtidas pela semântica da RDFS, que "*Renato ministra Engenharia de Software*" que, por sua vez, "*é lecionada por Renato*".

Código 2.2 Exemplo de modelagem OWL [53].

```
1  @prefix owl: <http://www.w3.org/2002/07/owl#> .
2  @prefix acm: <http://www.acm.org/acmcs#> .
3  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
4  foaf:Pessoa  a  rdfs:Class .
5  acm:Disciplina  a  rdfs:Class .
6  acm:leciona  a  rdfs:Property   ;
7               rdfs:domain  foaf:Pessoa ;
8               rdfs:range   acm:Disciplina .
9  acm:ministra owl:equivalentProperty acm:leciona.
10 acm:ehLecionadaPor owl:inverseOf acm:leciona .
```

A partir de um mesmo modelo ontológico podem ser representadas informações de contexto relativas a diferentes entidades, que instanciam as classes e utilizam as propriedades que compõem a ontologia. Dessa forma, para que um sistema compreenda e utilize uma informação de contexto ontológica, basta que este conheça o modelo de

dados, que é independente de qualquer implementação e pode ser acessado e incorporado por qualquer sistema.

2.1.4 *Simple Protocol and RDF Query Language* (SPARQL)

Como complemento aos esforços de padronização do RDF e OWL, o Consórcio W3 criou de uma linguagem padrão para a escrita de consultas sobre triplas segundo o modelo de dados RDF (sujeito, predicado, objeto): a linguagem SPARQL.

A linguagem SPARQL permite quatro formas de consulta a triplas RDF, a saber [14]: SELECT, ASK, CONSTRUCT e DESCRIBE. O Código 2.3 apresenta um exemplo da forma de consulta mais comum (SELECT) sobre o qual serão exploradas as regras da sintaxe da linguagem SPARQL.

Código 2.3 Exemplo de consulta SPARQL SELECT [53].

```
1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX ex: <http://example.org/>
3 SELECT ?person ?name
4 WHERE {
5     ?person   rdf:type   ex:Person  .
6         ?person   ex:name  ?name
7 }
```

As linhas 1 e 2 descrevem prefixos, por meio da cláusula PREFIX, dos espaços de nomes XML utilizados na consulta, no caso, `rdf:` e `ex:` . A cláusula SELECT, na linha 3, identifica uma ou mais variáveis nas quais devem ser armazenados os valores de retorno da consulta, no caso, `?person` e `?name`. Destaque para a cláusula WHERE, nas linhas 4 a 7, que descreve um padrão de grafo (do inglês *pattern graph*) sob a forma de uma conjunção lógica de triplas RDF escritas em Turtle.

Esse padrão de grafo (ou subgrafo) representa a informação que deverá ser buscada na fonte de dados RDF em questão. Para isso, um processador de consultas SPARQL visa identificar um "casamento de triplas" (do inglês *triple matching*), i.e. localizar triplas do subgrafo que combinam estrutural e/ou lexicalmente com triplas da fonte de dados. No exemplo anterior, deseja-se o identificador (IRI de sujeito) e o nome (objeto literal) de um recurso do tipo Pessoa. Os valores encontrados para `?person` e `?name` na fonte de dados que "casam" com a posição dessas variáveis no subgrafo (sujeito e objeto, respectivamente) serão retornados nessas variáveis de consulta da cláusula SELECT. Caso haja nenhum "casamento" entre o subgrafo e as triplas da fonte de dados, o retorno da consulta é nulo.

Outro tipo de consulta importante para a compreensão deste trabalho é utilizando a CONSTRUCT. As consultas que utilizam esta cláusula retornam um conjunto de triplas, que podem ser originadas da própria fonte de dados ou triplas novas, por exemplo, convertidas para um novo modelo de dados, conforme o exemplo apresentado na Figura 2.4. O resultado de uma consulta com CONSTRUCT é um novo grafo contendo as triplas retornadas.

Código 2.4 Exemplo de consulta SPARQL CONSTRUCT.

```
1  PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
2  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
3  CONSTRUCT {
4    ?X vCard:FN ?name .
5    ?X vCard:URL ?url .
6    ?X vCard:TITLE ?title .
7  }
8  FROM <http://www.w3.org/People/Ernesto-Veiga/card>
9  WHERE {
10   ?X foaf:name ?name .
11   ?X foaf:homepage ?url .
12   ?X foaf:title ?title .
13 }
```

Neste exemplo, as informações de um indivíduo, representadas conforme o modelo de dados da ontologia FOAF, estão sendo traduzidas para uma representação baseada na ontologia VCARD. Das linhas 4 a 6, dentro da cláusula CONSTRUCT, está sendo descrito o modelo de triplas VCARD que se quer retornar, enquanto das linhas 10 a 12, estão representadas as triplas equivalentes a estas no modelo FOAF. Dessa forma, onde for descrita no modelo consultado uma tripla com a propriedade `foaf:name`, será retornada a tripla, substituindo esta propriedade pela sua equivalente, `vCard:FN`.

Assim como em SQL, há cláusulas da SPARQL que alteram o resultado de uma consulta, podendo inclusive melhorá-lo quanto a precisão e desempenho. Nesse contexto, cita-se como exemplo as cláusulas FILTER, OPTIONAL e ORDER BY, bem como as expressões sobre agrupamentos via GROUP BY, tais como HAVING, SUM, AVG, entre outras [14, 20].

2.2 Ontologias

Esta seção realiza uma breve apresentação das principais ontologias utilizadas no desenvolvimento desta proposta, estudo de caso, validações e avaliações.

2.2.1 *Semantic Sensor Network* (SSN)

As tecnologias semânticas podem auxiliar no gerenciamento, consulta e combinação de sensores e dados de sensoriamento. Definições semânticas compartilhadas podem ajudar não só com a integração de dados de diferentes fontes, mas também na integração de novos dados espaciais e temporais, relacionados ao histórico de contexto. Criada pelo W3C *Incubator Group*, a SSN tem como objetivo central descrever as relações entre sensores, estímulos e observações, conforme o padrão de projeto *Stimulus-Sensor-Observation* (SSO) [26].

A ontologia SSN pode ser utilizada com um foco específico, dentro do domínio de rede semântica de sensores, ou combinar as diferentes perspectivas para as quais oferece apoio:

- *Perspectiva de sensor*: onde o foco é o sensor, como ele realiza o sensoriamento, e o que é detectado;
- *Perspectiva de dados ou observação*: com foco nas observações e metadados relacionados;
- *Perspectiva do sistema*: cujo foco é um sistema de sensores;
- *Perspectiva de entidades e propriedades*: onde as entidades e propriedades são o foco, bem como o que estas propriedades expressam.

Para permitir a utilização destes recursos a SSN é composta por um conjunto de módulos que permitem agrupar ou refinar estas diferentes perspectivas sobre os sensores ou sobre as informações detectadas, fornecendo um nível de descrição que pode ser abstrato ou detalhado. O conjunto de módulos, bem como as principais classes e propriedades da SSN são apresentados na Figura 2.3.

O padrão de projeto SSO, que visa apoiar todo tipo de sensor e observação baseado em ontologias ou vocabulários de sensores para Web Semântica, foi desenvolvido com base nos princípios ontológicos que o tornam reutilizável para diferentes áreas de aplicação [26]. Independente de ontologias de alto nível, introduz um conjunto de classes e relações voltadas para a descrição de estímulos, sensores e observação, onde:

- **Estímulos**: são mudanças detectáveis no ambiente, seja este físico ou lógico. São o ponto de partida de uma medição, podendo ser direta ou indiretamente relacionadas a propriedades ou características de interesse.
- **Sensores**: são objetos que realizam observações, por exemplo, transformando um estímulo de entrada em uma representação digital. Objetos são considerados sensores enquanto realizam algum tipo de sensoriamento.
- **Observações**: atuam como a relação entre o estímulo de entrada, o sensor e a saída do sensor. Podem também descrever outros parâmetros, tais como tempo e localização.

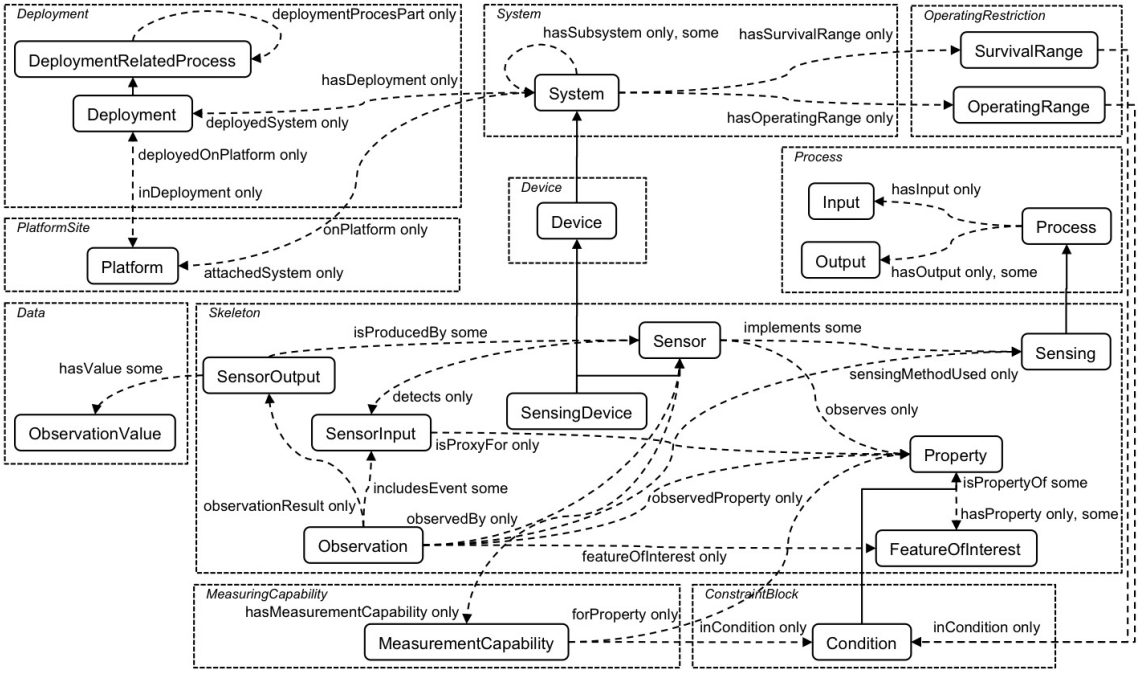


Figura 2.3: *Módulos, classes e propriedades da SSN.*

A Figura 2.4 apresenta a implementação do padrão SSO na ontologia SSN.

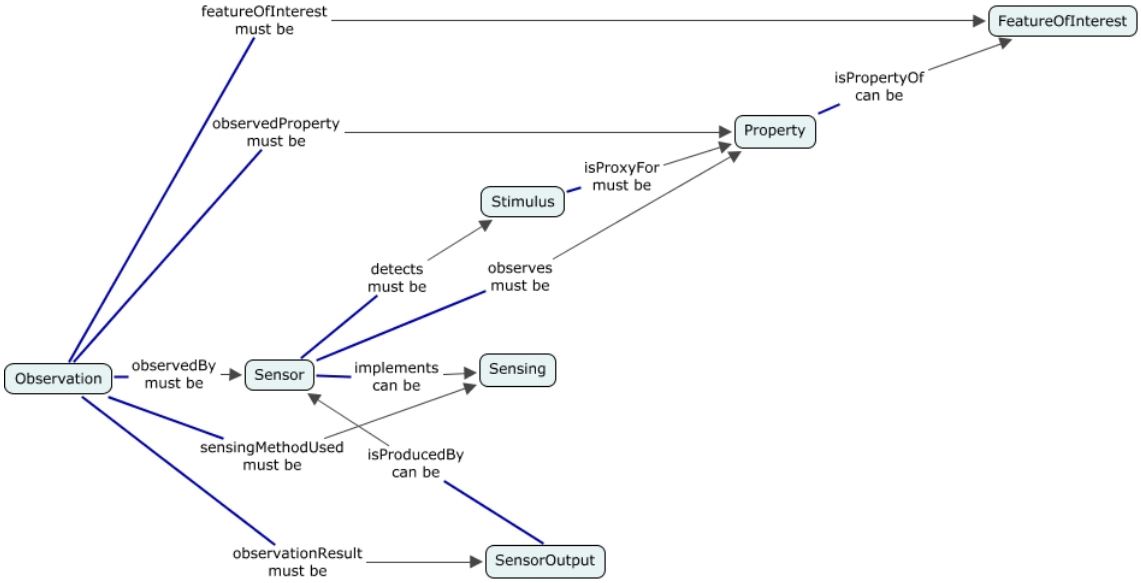


Figura 2.4: *Implementação do padrão de representação Stimulus-Sensor-Observation na ontologia SSN [28].*

A classe *ssn:Observation* na ontologia, oferece a estrutura para representar uma única observação. Uma observação é a representação que descreve uma (única) entidade observada, uma (única) propriedade, um (único) sensor com seu método de sensoriamento, e um (único) valor observado para a propriedade descrita. Observações de múltiplas entidades, ou de múltiplas propriedades de uma entidade devem ser representadas ou

como entidades e propriedades compostas ou como múltiplas observações, agrupadas em uma estrutura adequada.

A SSN define diversas propriedades para uma observação, entre elas:

- *ssn:featureOfInterest*: relaciona a observação à entidade de interesse que está sendo observada;
- *ssn:observedProperty*: representa uma propriedade específica de uma entidade de interesse, que pode ser observada por um sensor;
- *ssn:observedBy*: representa o sensor que produz uma observação;
- *ssn:observationResult*: representa o resultado de uma observação, que é expresso por uma instância da classe *ssn:SensorOutput*;
- *ssn:observationResultTime*: representa o instante em que o resultado da observação se tornou disponível.

A propriedade *ssn:observationResultTime* é definida como uma propriedade de objeto, uma vez que a SSN não determina ou modela um formato específico para a representação de instantes de tempo.

O resultado de uma observação, que é uma instância de *ssn:SensorOutput*, está relacionado às propriedades: *ssn:isProducedBy*, que relaciona o resultado da observação ao sensor que o produziu; e *ssn:hasValue*, que representa o valor da observação (por exemplo: “30°C”, “60 km/h”, etc). O valor de uma observação é expresso na SSN como uma instância da classe *ssn:ObservationValue*. A ontologia não restringe o formato do valor observado, que pode ser definido pelo usuário ou importado de uma outra ontologia.

A ontologia SSN foi utilizada como base para a padronização do serviço de representação ontológica de contexto, núcleo da proposta deste trabalho.

2.2.2 Monitoramento de Sinais Vitais Humanos (MSVH)

A ontologia MSVH [3] modela os sinais humanos consensualmente classificados na literatura especializada como sinais vitais, bem como o conhecimento relacionado a este domínio. No seu projeto foram definidas 37 questões de competência que guiaram a definição dos conceitos e relações modeladas pela ontologia, em consenso com uma equipe de enfermagem do Hospital das Clínicas da Universidade Federal de Goiás.

Entre as principais questões de competência tratadas pela MSVH estão: *i*) faixas de valores populacionais que identificam normalidade ou anormalidade para cada sinal vital; *ii*) faixas de valores personalizados de acordo com as características do paciente; *iii*) medições de sinais vitais de um paciente durante um determinado período de tempo; e *iv*) instante do tempo em que um paciente teve uma medição de sinal vital fora da normalidade.

Para a construção do modelo proposto foram reutilizadas ontologias existentes, como sugere a literatura [36]. Foram escolhidas para integrarem-se à MSVH as ontologias do modelo SeCoM (*Semantic Context Model*) [8] e a VSO (*Vital Sign Ontology*) [17]. A Figura 2.5 apresenta o modelo MSVH, com as suas principais classes e relações.

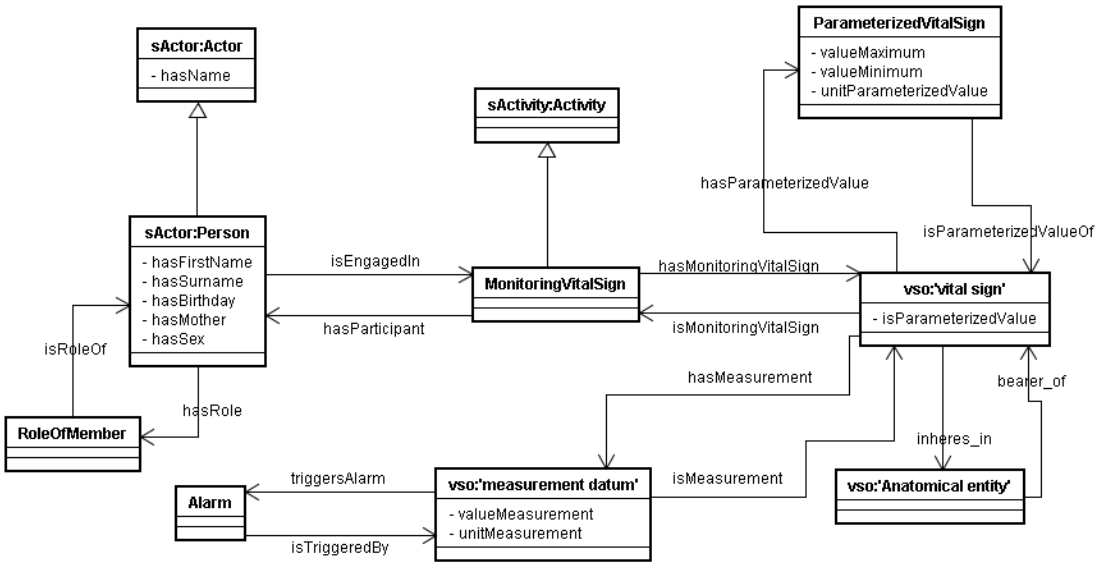


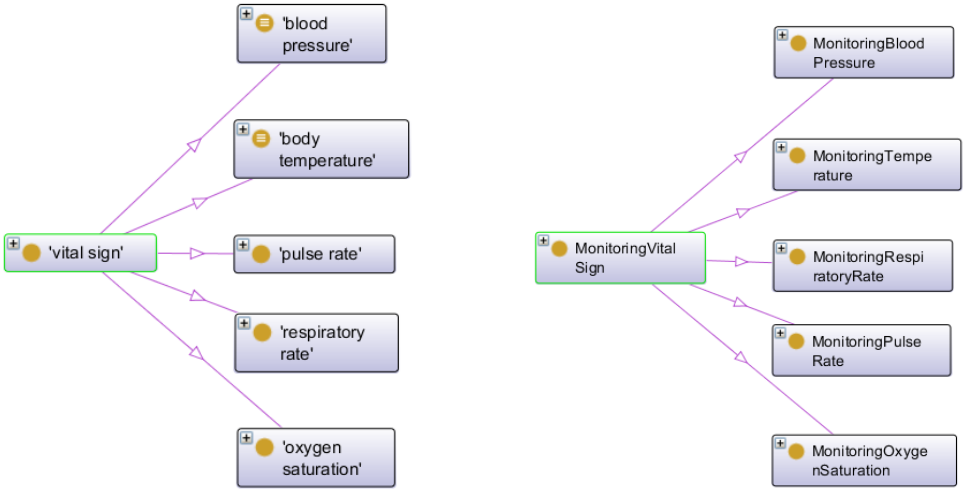
Figura 2.5: *Visão geral do modelo MSVH.*

A ontologia VSO modela os sinais vitais consensualmente adotados pela literatura médica, com exceção de "*oxigen saturation*", que foi incluído no desenvolvimento de MSVH como recomendação dos profissionais de saúde envolvidos. A principal classe da ontologia MSVH, *MonitoringVitalSign*, é superclasse das classes que modelam o monitoramento dos sinais vitais e gerencia as relações com as demais informações de contexto. Estes relacionamentos, modelados na ferramenta Protégé [3] são apresentados na Figura 2.6.

O uso de ontologias permite a formalização de um domínio de interesse, descrevendo a semântica explícita de seus conceitos, o seu compartilhamento e reúso por outros membros que tenham interesse nesse domínio. O modelo ontológico da MSVH permite inferir informações sobre as medições de sinais vitais humanos, disparando alarmes que podem alertar uma equipe de saúde quando ocorrerem desvios de normalidade dos valores de referência e/ou individualizados dos sinais vitais do paciente

Também foram desenvolvidas 80 regras na linguagem SWRL, que definem alarmes relacionados a diferentes tipos de anormalidades que podem ocorrer de acordo com a medição de cada sinal vital, segundo os parâmetros consensuais da literatura. A ontologia MSVH foi validada através de consultas SPARQL desenvolvidas para cada

[3] http://protege.stanford.edu/



(a) (b)

Figura 2.6: *Modelagem dos sinais vitais pela ontologia VSO (a) e do monitoramento dos sinais vitais por MSVH (b).*

questão de competência modelada, demonstrando a capacidade de representação do domínio e a recuperação de informações representadas.

A ontologia MSVH foi utilizada neste trabalho como ontologia de domínio para conversão das informações de contexto independentes de domínio, representadas com SSN, para o domínio de monitoramento de sinais vitais humanos. Esta conversão permite aumentar a expressividade do contexto e realizar inferências específicas sobre o domínio ao qual este pertence.

2.3 Sistema de Gerenciamento de Contexto *Hermes*

Com a crescente incorporação da *sensibilidade ao contexto* em sistemas de software, tornando-a um serviço que pode ser utilizado nos mais diversos tipos de aplicação, o gerenciamento de contexto tem se caracterizado como uma funcionalidade essencial [39]. Para demonstrar como os dados se movem fase a fase nestes sistemas e apoiar o seu gerenciamento, é utilizada a abordagem de ciclo de vida do contexto, apresentada na Seção 1.1.

São denominados Sistemas de Gerenciamento de Contexto (CMS), os sistemas que coletam, modelam, interpretam e fornecem contexto para aplicações [48]. A literatura recomenda a realização independente de cada etapa do ciclo de vida do contexto, porém, estas etapas devem ser gerenciadas em conjunto para oferecer de forma transparente todos os serviços necessários à aplicações sensíveis ao contexto [41].

2.3.1 Requisitos para Gerenciamento de Contexto

Perera et al. [39] sumarizam os mais importantes princípios de projeto de CMS de acordo com a literatura. Entre eles:

1. **Arquitetura em camadas e componentes**: as funcionalidades precisam ser divididas em camadas e componentes de maneira significativa, onde cada componente é responsável por executar uma determinada tarefa, que deve ser realizada de forma independente das demais.
2. **Escalabilidade e extensibilidade**: um componente pode ser adicionado ou removido dinamicamente, sem afetar os componentes existentes, o que implica na utilização de padrões no desenvolvimentos destes componentes.
3. **Gerenciamento automático do ciclo de vida do contexto**: CMS devem manipular as informações de contexto, de acordo com as etapas do seu ciclo de vida, e oferecê-las com o mínimo de intervenção humana.
4. **Independência do modelo de contexto**: a modelagem e armazenamento do contexto deve ser realizada de forma independente do CMS, permitindo que ambas as partes possam ser alteradas de maneira independente.
5. **Modelagem expressiva e abrangente**: modelos de contexto devem ser facilmente extensíveis, uma vez que há um número cada vez maior de dispositivos de aquisição (sensores) e domínios específicos de contexto.
6. **Compartilhamento de informações (tempo real e histórico)**: o compartilhamento de contexto deve acontecer em diferentes níveis, seja entre os componentes do CMS, ou destes para as aplicações. A independência do modelo de contexto é crucial para o compartilhamento.

Para cumprir estes requisitos, é recomendada pela literatura *a abordagem baseada em ontologias para o gerenciamento do ciclo de vida do contexto* [7, 39]. Os requisitos 1 a 3, mesmo sendo diretamente ligados ao projeto arquitetural de um CMS, são facilitados quando há *a independência do modelo de contexto*, proposta no requisito 4. A utilização de ontologias separa a modelagem do contexto do seu gerenciamento, contribuindo para *o melhor desacoplamento entre os componentes da arquitetura*.

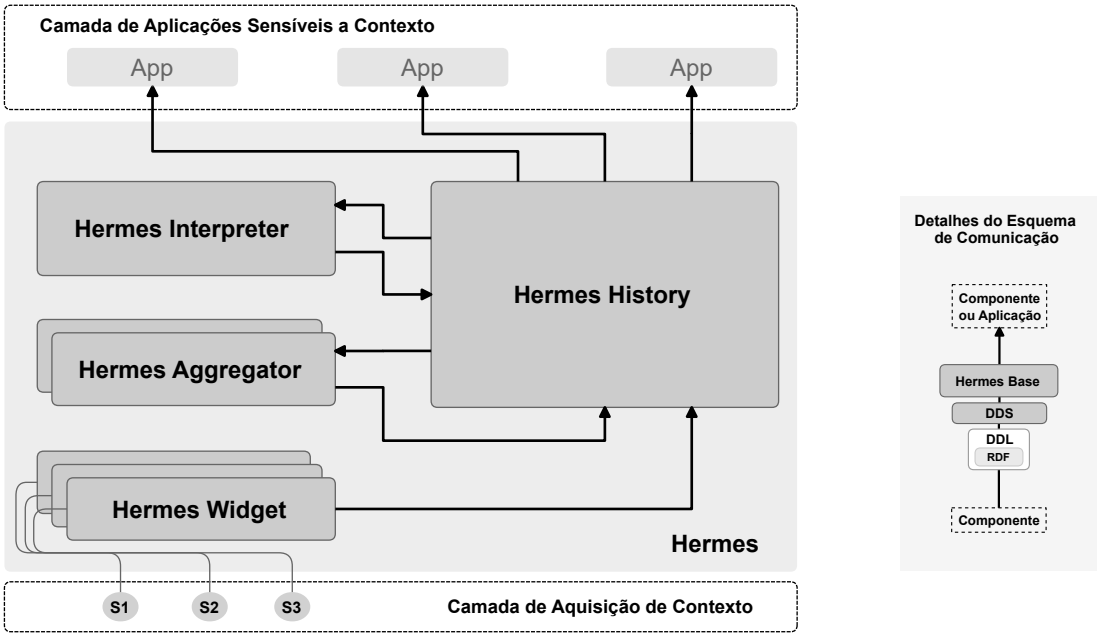
O requisito 5 está diretamente ligado à utilização de ontologias, que permite *a representação mais expressiva do contexto*, além de *possibilitar técnicas de raciocínio* e *possuir forte apoio de padrões e ferramentas*. O último requisito, da mesma forma, necessita da independência do modelo de contexto, fornecida pela abordagem baseada em ontologias para permitir *o compartilhamento do contexto manipulado pelo CMS, em diferentes níveis de abstração*. Este contexto deve ser disponibilizado tanto em tempo real, como também na forma de histórico, o que contribui para a identificação de padrões e preferências de usuários/aplicações.

2.3.2 Projeto Arquitetural de *Hermes*

Hermes [4] é um CMS baseado em ontologias proposto por Veiga et al. [52] e Maranhão et al. [33]. Fundamentado de acordo com o estado da arte de Computação Sensível a Contexto e seguindo os princípios de projeto apresentados, os principais requisitos para sua construção são:

- Oferecer serviços de apoio ao gerenciamento do ciclo de vida das informações de contexto (aquisição, modelagem, interpretação e disseminação) [12, 39];
- Infraestrutura baseada em componentes independentes [39];
- Serviços apoiados na semântica das informações de contexto [7, 39].

A Figura 2.7(a) apresenta a arquitetura de *Hermes* com seus componentes, a saber: i) *Hermes Base*, responsável pela comunicação componente/componente e componentes/aplicações na disseminação de contexto [33]; ii) *Hermes Widget*, representação ontológica de contexto coletado de sensores [50]; iii) *Hermes Aggregator*, agregador para geração de contexto com maior nível de expressividade; iv) *Hermes Interpreter*, interpretação e filtragem semântica de contexto [33] e v) *Hermes History*, gerenciador de histórico, consultas e acesso ao contexto.



(a) (b)

Figura 2.7: *Infraestrutura Hermes, onde: (a) apresenta o projeto arquitetural de componentes e (b) em detalhes, o esquema de comunicação.*

[4] O nome *Hermes* vem da mitologia grega, nome dado ao mensageiro dos deuses. Nesta proposta, como CMS, representa a ideia de uma comunicação de informações rápida e eficiente entre diferentes partes.

No intuito de simplificar a visualização do fluxo de contexto dentro do CMS *Hermes*, a Figura 2.7(b) apresenta separadamente o esquema de comunicação. O componente *Hermes Base*, responsável pela comunicação em *Hermes*, encapsula o acesso a um *middleware* de comunicação DDS (*Data Distribution Service*) que, por sua vez, permite o envio de modelos RDF serializados. Este esquema é utilizado na arquitetura de *Hermes* para comunicação entre os componentes e destes com as aplicações sensíveis a contexto.

2.3.3 Componentes de *Hermes*

Esta subseção apresenta os componentes de *Hermes* preexistentes a esta pesquisa. Os demais componentes, parte do escopo deste trabalho, serão apresentados em detalhes no Capítulo 3.

Hermes Base

Base da distribuição de contexto na infraestrutura *Hermes*, o componente *Hermes Base* [33] oferece uma interface para que os demais componentes se comuniquem entre si e também com as aplicações sensíveis a contexto através do acesso a um *middleware* de comunicação DDS. A especificação DDS, da OMG, é o primeiro padrão internacional aberto de *middleware* direcionado para comunicação segundo o paradigma *publish/subscribe* para sistemas embarcados.

Em ambientes pervasivos e distribuídos, onde 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*. O desacoplamento realizado por *Hermes Base*, por sua vez, confere a *Hermes* a possibilidade de conectar outros sensores (com seus respectivos *Hermes Widgets*), *Hermes Interpreters* e os demais componentes e aplicações sem que estes(as) conheçam o *middleware* DDS, ou qualquer outro *middleware* que forneça a integração.

O componente *Hermes Base* foi projetado de forma a centralizar as complexidades de acesso ao fornecedor DDS, neste caso o CoreDX [5], e expor classes e métodos simplificados e independentes do fornecedor, para o restante da infraestrutura e aplicações. Dessa forma, *Hermes Base* facilita o trabalho de desenvolvimento de aplicações e outros componentes, abstraindo a complexidade do serviço de comunicação. A Figura 2.8 apresenta as interfaces expostas pelo componente de comunicação.

O elemento central de *Hermes Base* é a interface *Hermes Base Manager*, que expõe os principais métodos para registro, publicação e assinatura de tópicos. Dentre estes, o método *createNotificationTopic* é utilizado por todos os componentes e aplicações

[5] http://www.coredx.com/

que desejam receber contexto para a criação de *tópicos de notificação*. A interface *Notification Listener* complementa este serviço, capturando os eventos de publicação de contexto em tópicos criados. Essa interface é exposta às aplicações e componentes que podem implementar o método *handleContext()* para tratar o contexto conforme suas necessidades.

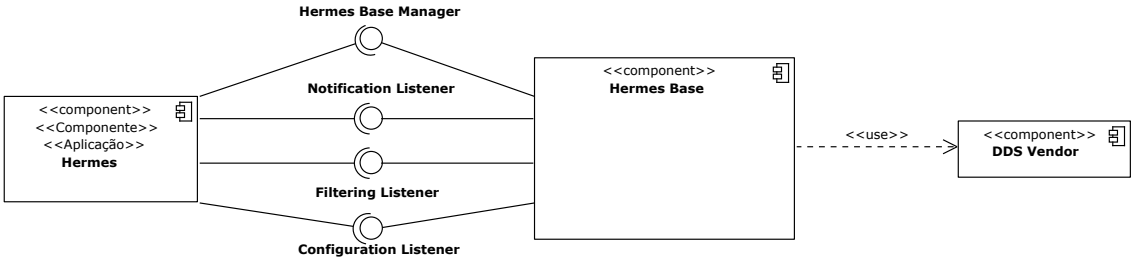


Figura 2.8: *Diagrama de componentes de Hermes Base.*

O método *handleContext()* possui os seguintes atributos transmitidos em uma notificação:

- *id_entidade*: principal elemento no contexto notificado, que objetiva identificar a instância do objeto publicado;
- *nome_topico*: nome do tópico onde o contexto foi publicado;
- *complemento_topico*: utilizado para publicar eventos de contexto filtrados;
- *caminho_ontologia*: localização do *schema* ou modelo ontológico que descreve a semântica do contexto representado, que é transmitido em uma publicação;
- *rdf_serializado*: informação mais relevante dentre os argumentos desse método, trata-se do modelo de contexto RDF publicado;
- *tipo_serializacao*: trata-se do tipo de serialização aplicada no modelo RDF, por exemplo, Turtle, Json, entre outras.

Após a assinatura dos tópicos através dos *listeners*, componentes e aplicações são notificados conforme ocorrem as publicações de contexto nos respectivos tópicos.

Hermes Interpreter

A partir de notificações de contexto representado ontologicamente, interpreta as informações conforme um ou mais mecanismos de inferência e produz informações de mais alto nível do que as representadas pelos *Hermes Widgets*. 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* [33].

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. [39], 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.

As tarefas de interpretação 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.

2.4 Considerações Finais

Este capítulo apresentou os conceitos fundamentais para a compreensão dos serviços de gerenciamento de contexto que compõem a proposta deste trabalho. Os fundamentos e tecnologias de Web Semântica estão intrínsecos no projeto e desenvolvimento destes serviços. As ontologias apresentadas, SSN e MSVH, são a base para a representação de contexto e criação de cenários de aplicação. Também foi apresentada uma visão geral da infraestrutura *Hermes*, uma vez que este sistema para gerenciamento de contexto é composto por diferentes componentes e serviços.

Dos componentes de *Hermes*, estão no escopo desta dissertação:

1. *Hermes Widget*: componente para representação ontológica de contexto;
2. *Hermes Aggregator*: componente para agregação de contexto, baseado em representações realizadas por *Hermes Widget*;
3. *Hermes History*: componente para persistência, histórico e consultas de contexto processado pela infraestrutura *Hermes*. Também participa no gerenciamento da distribuição de contexto.

CAPÍTULO 3

Serviços para Representação, Agregação e Histórico de Contexto Ontológico

Uma grande parte da pesquisa sobre computação sensível a contexto está voltada para a investigação de abordagens para a etapa de modelagem do ciclo de vida das informações que são utilizadas por aplicações sensíveis a contexto [7]. A importância e os benefícios de uma representação formal das informações de contexto têm sido destacados na literatura, que recomenda a modelagem baseada em ontologias para prover uma representação altamente expressiva e interoperável do contexto, que permite técnicas de raciocínio sobre a semântica dessas informações [22, 7, 39].

A agregação de contexto, além de abstrair a complexidade da utilização do contexto por parte das aplicações, oferece informações com um nível de expressividade maior, realizando a fusão de dados de diferentes provedores. O histórico, por sua vez, deve persistir os diferentes tipos de informação de contexto processadas e gerenciadas pelos serviços de um CMS e oferecer a capacidade de recuperação dessas informações.

No intuito de cumprir a proposta de pesquisa, fundamentada na necessidade do desenvolvimento de serviços relacionados à etapa de modelagem do ciclo de vida do contexto [39], compõem o escopo deste trabalho:

- *serviço de representação ontológica de contexto* independente do domínio de aplicação;
- *serviço de agregação de contexto* para componentes ou aplicações interessadas;
- *serviço de histórico* de informações de contexto.

De acordo com o problema descrito e contextualizado no Capítulo 1, aqui revisitado, e com base na fundamentação teórica apresentada no Capítulo 2, essa solução pretende desacoplar do projeto de aplicações sensíveis a contexto os serviços de representação, agregação e histórico, apoiando o gerenciamento de contexto e reduzindo desta forma a complexidade inerente ao desenvolvimento de aplicações sensíveis a contexto.

3.1 *Hermes Widget*: representação ontológica de contexto

Hermes Widget (HW) [50, 49] é o componente responsável por abstrair o serviço de representação ontológica do contexto. Este serviço é fundamental dentro do CMS *Hermes* ou de qualquer outra infraestrutura de gerenciamento de contexto, uma vez que oferece as informações de entrada necessárias à realização dos demais serviços e também às aplicações. A representação baseada em ontologias marca em *Hermes* a primeira etapa de transformação dos dados sem processamento (como são adquiridos de seus provedores) em contexto de alto nível, que pode ser utilizado por outros componentes e por aplicações sensíveis a contexto.

O restante desta seção apresenta os detalhes do HW. São abordados requisitos, questões de projeto e arquitetura, serviços disponibilizados, tecnologias utilizadas e a validação do componente em um cenário de aplicação.

3.1.1 Requisitos

O HW, como parte do CMS *Hermes*, atende aos requisitos já destacados na Seção 2.3, sendo os principais: *arquitetura em camadas com componentes reutilizáveis*, *extensibilidade* e *independência do modelo de contexto*. Seu principal requisito funcional é a *representação ontológica*, cuja importância é fundamentada no papel central da etapa de modelagem no ciclo de vida do contexto.

Os principais requisitos para o desenvolvimento do HW incluem:

1. **Permitir a especificação de um sensor e do processo de sensoriamento**: o componente de representação deve permitir a descrição das informações relacionadas a um processo de sensoriamento e de suas relações, tratando termos e eventos deste domínio tais como: sensor, propriedade, entidade de interesse, observação (ou sensoriamento), resultado, entre outros.
2. **Representação ontológica das informações de contexto**: a representação realizada por HW deve representar formalmente as informações de contexto. A representação ontológica complementa o requisito de independência do modelo de contexto através da utilização de ontologias para modelagem e representação de informações, sendo a base para a representação de instâncias de contexto, trazendo expressividade, formalidade e padronização para a realização deste serviço. Dessa forma, *HW pode ser utilizado para representar contexto de diferentes domínios de aplicação.*
3. **Disseminação de informações de contexto para demais componentes e aplicações**: após representar ontologicamente o contexto recebido da camada de aquisi-

ção de dados de sensores, o HW deve disponibilizar essas informações para demais componentes e/ou aplicações interessados.

São acrescentados a estes, os seguintes requisitos não-funcionais, descritos de acordo com a norma internacional SEVOCAB [1]:

- **Manutenabilidade**: *"é a facilidade com que um sistema ou componente de software pode ser modificado para alterar ou adicionar recursos, corrigir falhas ou defeitos e otimizar o desempenho ou outros atributos"*.
- **Interoperabilidade**: *"grau em que dois ou mais sistemas, produtos ou componentes podem intercambiar informações entre si"*. O modelo de contexto gerado por *Hermes Widget* é consumido por demais componentes e aplicações e, dessa forma, precisa ser disponibilizado em um formato interoperável.
- **Flexibilidade**: é a *"capacidade que um sistema ou componente tem para ser modificado para uso em aplicações ou ambientes diferentes para os quais eles foram inicialmente projetados"*. A utilização de ontologias separa o modelo de contexto das regras de negócio do sistema, o que torna o componente reutilizável, minimizando as alterações necessárias quando há mudança no domínio de aplicação.
- **Escalabilidade**: sub-característica de adaptabilidade, é o *"grau com que um produto ou sistema pode efetivamente e eficientemente manipular uma porção crescente de trabalho de maneira uniforme"*.

3.1.2 Arquitetura

A Figura 3.1 apresenta as camadas arquiteturais do componente de representação de contexto. Este projeto foi definido com base nos requisitos de *Hermes Widget* juntamente com os requisitos do próprio CMS *Hermes*, que direcionam as questões de configuração e comunicação, de maneira geral, para todos os seus componentes.

Em um nível mais baixo que a representação, a camada de aquisição realiza a coleta dos dados de sensores que detectam as alterações de contexto. Após a aquisição, a representação é realizada com base em um modelo ontológico, conforme especificado nos requisitos, e o contexto produzido é gerenciado pelas camadas de configuração, notificação e comunicação, que realizam os serviços necessários à sua disponibilização.

Esse modelo arquitetural, também utilizado como base para os demais componentes deste trabalho, visa separar os diferentes serviços realizados dentro de um componente em camadas bem definidas que devem ser implementadas de maneira desacoplada.

[1] http://pascal.computer.org/sev_display/

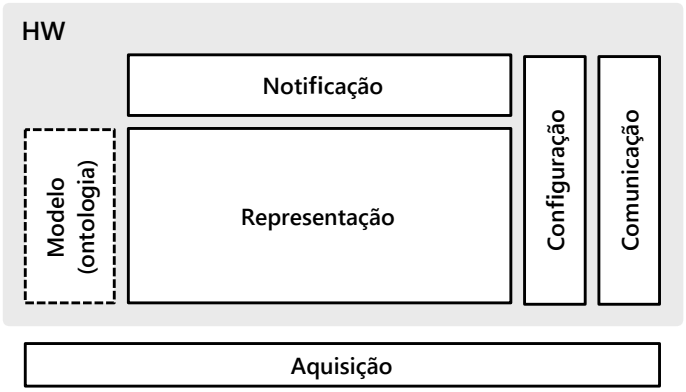


Figura 3.1: *Arquitetura em camadas do Hermes Widget.*

Camada de Aquisição

Antes que o serviço de representação do HW seja iniciado, é necessário que as informações de contexto sejam coletadas de seus provedores. Este processo *ocorre na interação entre a camada de aquisição e os sensores que produzem o contexto, externamente ao HW*, como mostra a Figura 3.1.

Os provedores de contexto, fontes de dados para HW, podem ser sensores *físicos*, *virtuais* ou *lógicos* [24]:

- *Sensores físicos* são aqueles que coletam dados fisicamente observáveis, tais como: sinais vitais, temperatura ambiente e luminosidade.
- *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.
- *Sensores virtuais* coletam informações que não podem ser medidas fisicamente, tais como contatos telefônicos, preferências de configuração de usuários e informações de dispositivos computacionais.

Camada de Representação

O serviço de representação, realizado por esta camada, permite que, além das próprias informações do sensor, incluindo os dados de contexto providos por ele, outras informações possam ser acrescentadas ao modelo de contexto que será posteriormente notificado. Esta característica enriquece o contexto em relação ao seu nível de expressividade, tratando semanticamente o domínio.

A Figura 3.2 apresenta mais detalhes do serviço de representação ontológica de contexto. As aferições de contexto recebidas de um sensor são transformadas em informações de entrada para o serviço de representação, através do serviço de aquisição. O serviço de representação é composto de duas partes principais:

1. **Gerente de Representação**: é a etapa do serviço que realiza a representação ontológica do contexto adquirido, e tem como base para esta finalidade a ontologia que modela o domínio das informações adquiridas.
2. **Gerente de Contexto**: prepara as representações para o serviço de notificação, na forma de instâncias de contexto devidamente serializadas.

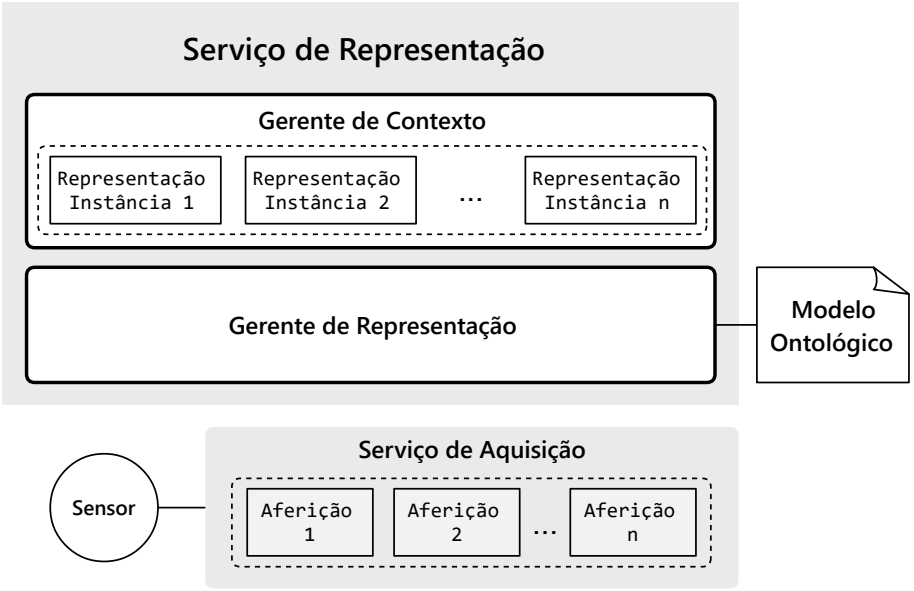


Figura 3.2: *Serviço de representação ontológica de contexto.*

O gerente de representação é responsável por receber os dados de contexto de baixo nível, obtidos da camada de aquisição, e representar essas instâncias de acordo com um modelo de contexto definido. Após a representação, o gerente de contexto envia as novas instâncias para publicação, que é realizada posteriormente através da camada de notificação.

Camadas de Configuração, Comunicação e Notificação

Hermes Widget, bem como os demais componentes de *Hermes*, implementa uma interface fornecida pelo componente de comunicação *Hermes Base* [33], descrito na Subseção 2.3.3, para notificar o contexto representado aos seus assinantes. O HB abstrai a complexidade de acesso a um *middleware publish/subscribe* utilizado para a comunicação entre componentes do CMS *Hermes* e entre estes e as aplicações sensíveis a contexto.

O serviço de configuração de cada HW acessa um arquivo JSON que descreve os parâmetros de configuração do componente. Estas definições tratam de maneira específica a criação do tópico que receberá as informações de contexto representadas e que será assinado por aplicações e outros componentes interessados, que passarão a receber as notificações de contexto desse tópico.

O Código 3.1 apresenta o arquivo JSON de configuração para um *Hermes Widget* que representa o monitoramento de temperatura corpórea. Este arquivo, estruturado na forma de pares chave-valor, determina para:

- *tipo*: o tipo do tópico criado (“notificação”);
- *registrar*: o nome do tópico que será registrado no serviço de comunicação (neste exemplo é o identificador do sinal vital pressão arterial, como descrito na ontologia MSVH, apresentada na Subseção 2.2.2, concatenado ao sufixo ‘H’, indicando que o histórico do contexto deverá ser mantido);
- *publicar*: o nome do tópico onde HW irá publicar (VSO_0000005H) e um complemento do tópico (“reason”).

O complemento do tópico é uma configuração complementar (e opcional) utilizada para finalidades como filtragem de contexto, onde é necessário restringir determinadas notificações a determinados componentes. Este tipo de configuração é principalmente necessária em conjunto com o componente *Hermes Interpreter* [33], apresentado na Subseção 2.3.3.

Código 3.1 Arquivo de configuração para um *Hermes Widget* que realiza a representação ontológica do sinal vital *pressão arterial*.

```
1  {   "topicos":[
2          {"tipo":"notificacao",
3           "registrar":"VSO_0000005H",
4           "publicar":[
5              {"nome":"VSO_0000005H",
6               "complementoTopico":"reason"}]
7          }
8      ]
9  }
```

No caso específico de *Hermes Widget*, este componente apenas cria (registra) *tópicos para publicação*, uma vez que não depende de informações processadas por nenhum outro componente de *Hermes* para realizar seus serviços.

3.1.3 Detalhes de Implementação

Em sua implementação a arquitetura do HW foi inicialmente desmembrada em duas partes principais e totalmente desacopladas, a saber:

- ***Hermes Widget Manager***: fornece os serviços básicos do HW, incluindo *comunicação*, *configuração* e *notificação*; fornece as camadas básicas de gerenciamento do componente, com base em padrões de projeto arquiteturais.

- ***Hermes Widget Sensor***: expõe a API do HW e estende as camadas de gerenciamento de *Hermes Widget Manager*, acrescentando a este o serviço de *representação ontológica do contexto*.

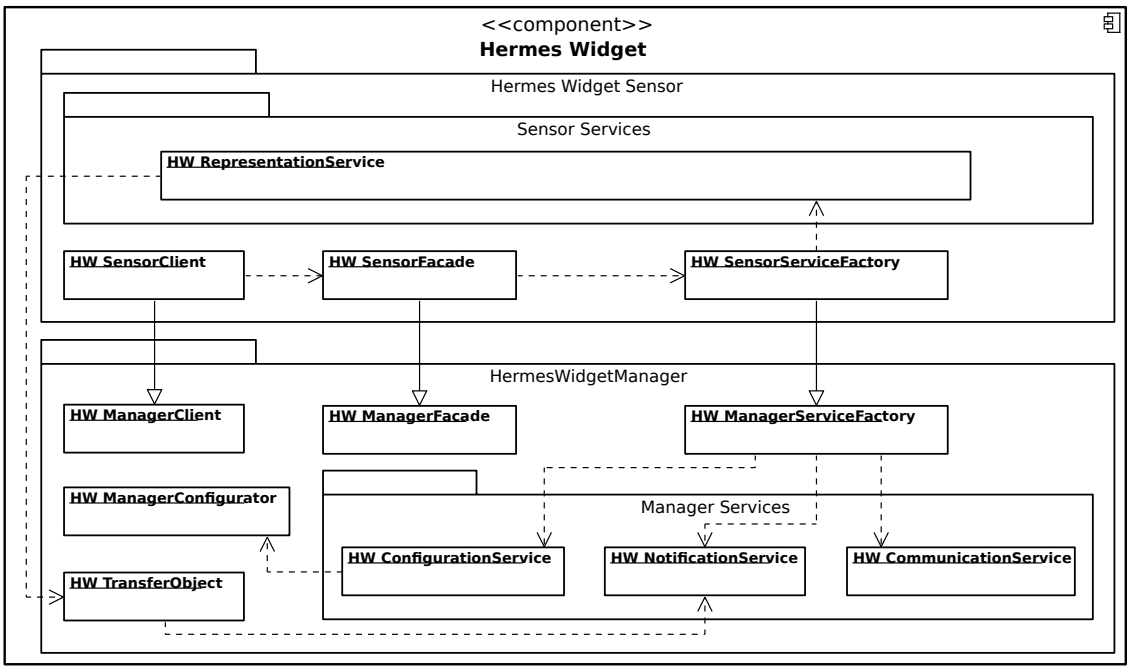


Figura 3.3: *Implementação do componente Hermes Widget.*

O objetivo dessa decisão de projeto é desacoplar a camada e o serviço de representação, que depende do domínio para o qual o componente será utilizado, dos demais serviços realizados. Dessa forma, independente do domínio em que HW seja utilizado, as interações necessárias com o componente e a utilização de sua API só se darão em *Hermes Widget Sensor*.

Essa estratégia de projeto mantém as camadas de gerenciamento (comunicação/configuração/notificação) transparentes aos desenvolvedores de aplicações sensíveis a contexto, contribuindo para a flexibilidade e reusabilidade do componente, e para a disponibilização de uma API simplificada, requisito destacado por Perera et al. [39].

De acordo com o domínio de aplicação, o trabalho do desenvolvedor em relação à interface fornecida por *Hermes Widget Sensor* é descrever as triplas que representam a parte de interesse do domínio necessária aos objetivos das aplicações que receberão o contexto. O intuito é tornar o componente flexível quanto ao domínio, visando deixar o mínimo de responsabilidade de implementação para o desenvolvedor.

Para realizar a implementação do serviço de representação, que na implementação é denominada *HW_RepresentationService*, foi utilizado o *framework* Jena [2], da

[2]Jena: https://jena.apache.org/

Apache, tecnologia *open source* que oferece API's para a manipulação de informação ontológica, tais como modelos RDF, ontologias OWL, consultas SPARQL, entre outras ferramentas. A construção das instâncias de contexto é baseada nas ontologias MSVH e SSN.

Por limitações na utilização de *hardware* específico para aquisição, este serviço é abstraído em HW como uma camada de *software* que não interage com sensores físicos. O processo de coleta de dados é simulado através de um sistema de leitura de dados de sensores previamente armazenados em arquivos, cujas aferições são enviadas ao serviço de representação do HW.

Toda implementação realizada foi baseada em padrões de projeto [15] para garantir a manutenabilidade do componente, além da alta coesão com baixo acoplamento entre as camadas.

3.1.4 Validação do Serviço de Representação de Contexto

Esta subseção apresenta a instanciação do componente *Hermes Widget* em um cenário de testes a fim de realizar sua validação funcional.

Cenário de testes

Para esta validação foram implementados 5 (cinco) *Hermes Widgets*, sendo um para cada sinal vital, a saber: *pressão arterial*, *temperatura corpórea*, *frequência de pulso*, *frequência respiratória* e *saturação de oxigênio*. O objetivo foi reproduzir o ambiente de monitoramento real de uma Unidade de Terapia Intensiva (UTI), com apoio da equipe de enfermagem do Hospital das Clínicas da UFG.

Como especificado, o componente *Hermes Widget* expõe uma API que oferece o serviço de representação, onde são descritas as relações que serão representadas a partir dos dados de contexto. Através dessa interface, cada *Hermes Widget* pôde ser configurado com as representações para cada sinal vital supracitado.

Não dispondo de sensores físicos para medição, nem do acesso a uma UTI real para integração dos sensores com os componentes *Hermes Widget*, os trabalhos e testes relacionados ao serviço de aquisição foram realizados na forma de simulação. Essa simulação ocorreu através da utilização de leituras de sinais vitais reais de pacientes em UTIs, disponíveis na base pública MIMIC [16].

A Tabela 3.1 contém amostras dos registros de um paciente identificado por *033n*. Essa amostra condiz com a frequência de monitoramento definida para os testes com as instâncias do HW, que foi o intervalo de 1 segundo entre aferições. A primeira coluna apresenta o momento da medição a partir do início do monitoramento, e as demais apresentam os dados aferidos para cada sinal vital, representado pelas sua respectiva sigla:

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

Tabela 3.1: *Amostra de registro da base MIMIC para o paciente 033n.*

Intervalo	***ABPMean***	***ABPsys***	***ABPdias***	***Pulse***	***Resp***	***SpO2***	***Temp***
sec	mmHg	mmHg	mmHg	bpm	bpm	%	°C
1	78	134	54	53	25	94	37.5
2	78	134	53	53	30	94	37.5
3	79	135	53	52	36	94	37.5
4	78	133	53	52	26	94	37.5

A partir dos dados importados da base MIMIC foram implementados sensores virtuais que disponibilizam as medições de acordo com a frequência de leitura também presente nos dados dessa base. Cada arquivo contém as leituras de todos os sinais vitais de um mesmo paciente, registrando 12 horas de monitoramento com intervalo de 1 segundo entre as aferições.

A instância de cenário para esta validação foi de 15 pacientes, sendo que cada um possui um total de 5 *Hermes Widgets* monitorando suas aferições de sinais vitais através dos sensores virtuais que realizam a leitura da base MIMIC para cada paciente.

Para cada sinal vital monitorado de um paciente foi implementado um sensor virtual. Por sua vez, cada sensor virtual instancia um *Hermes Widget* para o sinal vital específico que este representa. Dessa maneira, se um paciente tem cinco sinais vitais monitorados, ele possui cinco *Hermes Widgets* instanciados para o seu monitoramento, recebendo as informações dos sensores virtuais.

Representação ontológica com padronização SSN

Hermes Widget utiliza o padrão *Stimulus-Sensor-Observation* (SSO) [26], implementado pela ontologia SSN, apresentada na Subseção 2.2.1, que oferece os principais conceitos necessários para a representação ontológica de um processo de sensoriamento. O Código 3.2 apresenta um exemplo de representação dos dados de contexto de um sensoriamento do sinal vital de pressão arterial.

Código 3.2 Representação do sensoriamento de pressão arterial utilizando a ontologia SSN.

```
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix ssn:     <http://purl.oclc.org/NET/ssnx/ssn#> .

ssn:sensor-VSO_0000005-0e7e220a-2779-488d-942b-1e8f1f88a4db
    a   ssn:SensingDevice ;
    ssn:observes
        ssn:property-PresSang-6a705df7-b69c-44db-a09e-345f109b4b8c .

ssn:observation-PresSang-451d2ca4-fc90-4f42-9782-a0d193371ce3
    a   ssn:Observation ;
    ssn:featureOfInterest
        ssn:041n-6f5c59fe-c015-411d-bfb0-0baaf2991285 ;
    ssn:observationResult
        ssn:sensorOutput-VSO_0000005-2ac099c9-f9d5-4c97-ad99-3d479625d092 ;
    ssn:observationResultTime
        "2016-05-20T15:15:25.959-03:00"^^xsd:dateTime ;
    ssn:observedBy
        ssn:sensor-VSO_0000005-0e7e220a-2779-488d-942b-1e8f1f88a4db ;
    ssn:observedProperty
        ssn:property-PresSang-6a705df7-b69c-44db-a09e-345f109b4b8c .

ssn:property-PresSang-6a705df7-b69c-44db-a09e-345f109b4b8c
        a   ssn:Property .

ssn:041n-6f5c59fe-c015-411d-bfb0-0baaf2991285
    a   ssn:FeatureOfInterest .

ssn:sensorOutput-VSO_0000005-2ac099c9-f9d5-4c97-ad99-3d479625d092
    a   ssn:SensorOutput ;
    ssn:hasValue
        ssn:observationValue-3994bfe6-5ad8-4d8a-9d1a-83b537d3d6dc ;
    ssn:isProducedBy
        ssn:sensor-VSO_0000005-0e7e220a-2779-488d-942b-1e8f1f88a4db .

ssn:observationValue-3994bfe6-5ad8-4d8a-9d1a-83b537d3d6dc
    a   ssn:ObservationValue ;
    ssn:hasOutputUnit
        "mmHg"^^xsd:string ;
    ssn:hasOutputValue
        "95"^^xsd:nonNegativeInteger ;
    ssn:hasOutputValueAux
        "49"^^xsd:nonNegativeInteger .
```

Neste exemplo de representação podem ser identificadas as diferentes classes e propriedades da ontologia SSN que são utilizadas pelo HW para descrever a informação de contexto. De acordo com o padrão SSO, para cada instância de contexto aferida é criado um indivíduo da classe `ssn:Observation`, ao qual serão adicionadas as seguintes propriedades:

- `ssn:featureOfInterest`: relaciona uma instância de observação a um indivíduo do tipo `ssn:FeatureOfInterest`, que representa a entidade que está sendo monitorada, por exemplo, uma pessoa, um lugar, ou objeto;
- `ssn:observationResult`: liga a observação ao seu resultado, que é indivíduo da classe `ssn:SensorOutput`;
- `ssn:observationResultTime`: que tem como *range* o horário em que o sensoreamento foi realizado, `xsd:dateTime`;
- `ssn:observedBy`: liga o indivíduo da observação à instância do sensor que fornece o dado de contexto representado, `ssn:SensingDevice`;
- `ssn:observedProperty`: relaciona a observação ao tipo da propriedade que foi observada, `ssn:Property`.

No exemplo apresentado no Código 3.2, a propriedade `ssn:Property` descrita é a *pressão arterial*, representada pela URI `ssn:property-PresSang-6a705df7...` . O sensor `ssn:SensingDevice` que observa esta propriedade é descrito pela instância `ssn:sensor-VSO_0000005-0e7e220a...` . A entidade observada, neste caso o paciente ao qual a informação de contexto está relacionada, é representada pela instância `ssn:041n-6f5c59fe...`, do tipo `ssn:FeatureOfInterest`.

O indivíduo `ssn:sensorOutput-VSO_0000005-2ac099c9...` , da classe `ssn:SensorOutput`, representa o resultado de saída deste monitoramento, que possui um valor de observação `ssn:ObservationValue` para a medição de pressão arterial aferida. Neste caso específico da propriedade pressão arterial, o resultado da observação é composto, sendo formado pelos valores 95 e 49 (mmHg), que representam a pressão sistólica e a pressão diastólica, respectivamente.

Todas estas propriedades do sensoriamento são relacionadas à instância `ssn:observation-PresSang-451d2ca4...`, da classe `ssn:Observation`, que descreve a observação realizada.

A Figura 3.4 apresenta o grafo gerado a partir da representação de contexto do Código 3.2. Os nós deste grafo estão separados por cores, onde cada cor representa um espaço de nomes diferente. Como esta representação é em sua totalidade baseada na ontologia SSN, somente os dados literais (valores, unidade de medida e horas) possuem cor diferente dos demais nós, pois não são recursos e, portanto, não pertencem ao espaço de nomes da SSN. As arestas que representam as relações entre os nós e assumem a cor do nó de destino, que representa o *range* da propriedade em questão.

O tamanho de cada nó é proporcional ao seu grau, ou seja, a soma da quantidade de arestas que chegam e partem deste. Se uma aresta parte de um nó, este é o sujeito da tripla, a aresta é o predicado, e o objeto da relação é o nó de chegada da aresta, que pode ser um valor ou um recurso. Com essa informação podem ser visualizados os principais nós do grafo, que também são aqueles que mais atuam como *domain* ou *range* para as propriedades utilizadas na representação. As arestas, que representam as propriedades, que são o predicado das triplas, seguem a cor do nó objeto.

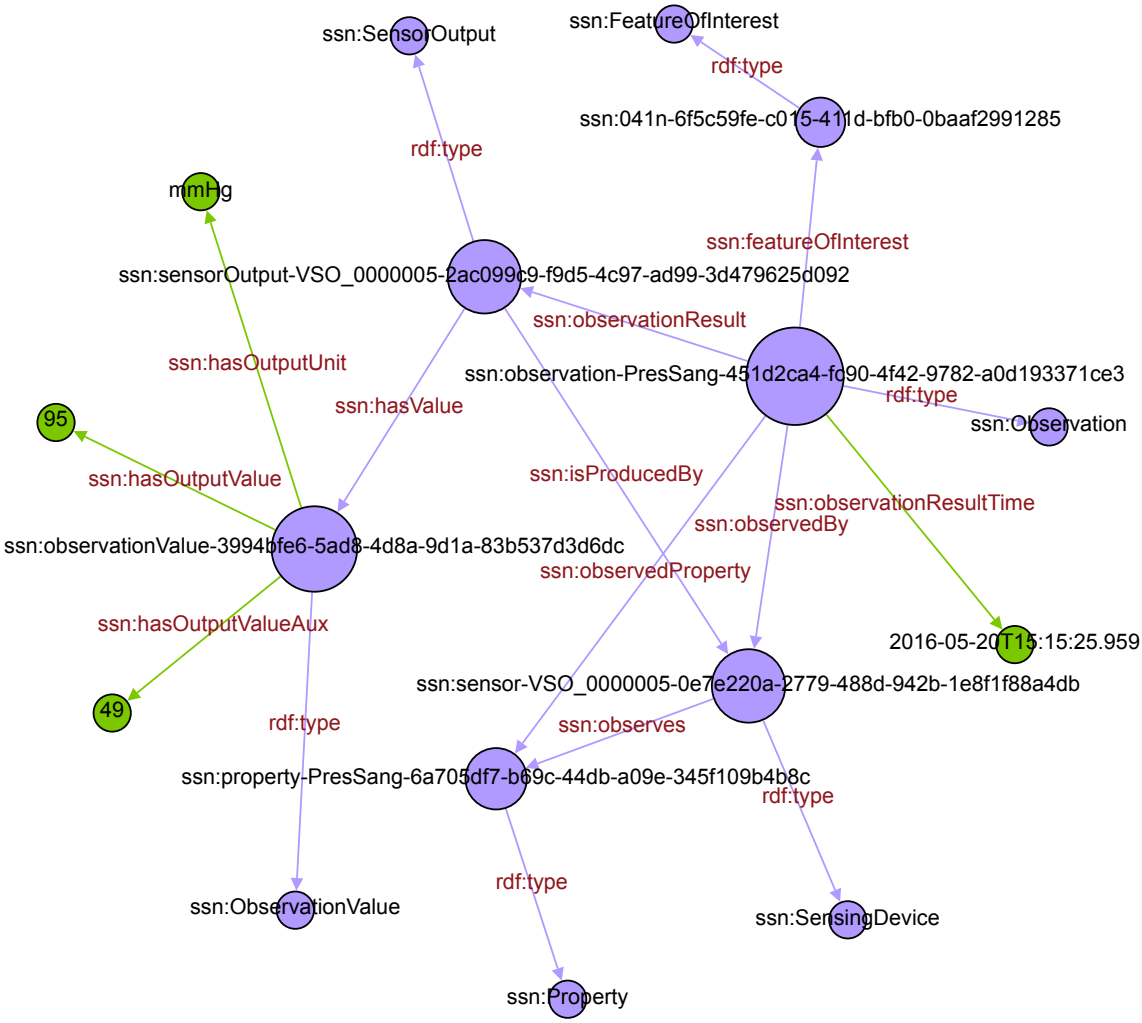


Figura 3.4: *Grafo da representação de contexto utilizando SSN.*

3.2 *Hermes Aggregator*: agregação de informações de contexto ontológicas

O componente *Hermes Aggregator* (HA) é responsável por reunir/agregar em um único modelo o contexto de diferentes fontes (*Hermes Widgets*), relacionadas a uma mesma entidade. Esse serviço aumenta o nível de expressividade do contexto, pois faz

com que um modelo carregue mais informações relacionadas a uma entidade, e também simplifica o acesso de aplicações a um conjunto de informações de contexto, diminuindo a granularidade em relação ao componente de representação.

3.2.1 Requisitos

1. **Permitir a especificação de uma entidade**: deve ser possível descrever o tipo de uma entidade e configurar quais são os atributos de interesse para agregação de contexto.
2. **Agregar contexto de diferentes provedores sobre uma mesma entidade**: o contexto de uma entidade pode ser composto por diferentes partes específicas. Por exemplo, a entidade *paciente* pode ter diferentes informações de contexto relacionadas, tais como: *temperatura corpórea*, *frequência de pulso* (e demais sinais vitais), localização, profissional de saúde responsável, entre outras.
3. **Elevar o nível de expressividade do contexto, diminuindo sua granularidade**: enquanto cada HW é responsável por descrever uma parte específica do contexto, coletada por um sensor, o HA é responsável por reunir/agregar várias partes diferentes da informação de contexto relacionada a uma entidade, oferecendo um modelo de contexto mais completo, com um maior nível de expressividade.
4. **Reduzir a complexidade no desenvolvimento de aplicações sensíveis a contexto**: o contexto relacionado a uma mesma entidade pode ter origem em diferentes provedores/sensores, que fornecem partes específicas dessa informação. O HA deve atuar de forma a evitar uma quantidade excessiva de interações com os provedores de contexto. Dessa forma, o processo de agregação simplifica e reduz o trabalho de acesso das aplicações ao contexto.

Os requisitos não funcionais descritos para HW também se aplicam aos demais componentes propostos neste trabalho, *Hermes Aggregator* e *Hermes History*, e por este motivo não serão citados novamente.

O conceito de entidade assumido neste trabalho é o mesmo que o utilizado por Dey [12] em sua definição de contexto, onde uma entidade pode ser qualquer pessoa, lugar ou objeto relevante para uma aplicação.

A utilização de técnicas de agregação possibilita a produção de novo conhecimento a partir de contexto preexistente, que pode ser tanto através da agregação lógica quanto de técnicas baseadas em semântica [4]. Diferentes soluções para gerenciamento de contexto adotam técnicas de agregação com dois objetivos principais: *i*) reduzir os requisitos de memória através da obtenção de um resumo do contexto adquirido; e *ii*) para aumentar o conhecimento do sistema através da introdução de novo contexto de alto nível.

Com acesso a um serviço de agregação, uma aplicação que deseja diferentes informações de contexto sobre uma mesma entidade só precisa se comunicar com um único componente responsável pela entidade que se tem interesse. Sem suporte a agregação, uma aplicação precisa combinar diferentes assinaturas ou consultas, a diferentes provedores/componentes, aumentando o número de requisições e a complexidade necessária para tratar as informações de contexto [12]. Dessa forma, a agregação de contexto permite reduzir a sobrecarga computacional e também de comunicação [31].

3.2.2 Arquitetura

O componente de agregação de contexto ontológico proposto neste trabalho possui 4 camadas, como mostra a Figura 3.5.

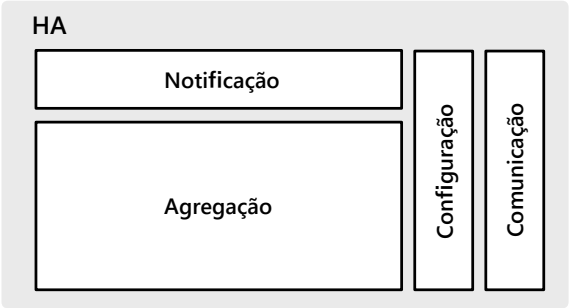


Figura 3.5: *Arquitetura em camadas do Hermes Aggregator.*

Comunicação, configuração e notificação

Da mesma forma que ocorre em HW, o HA segue os princípios de baixo acoplamento e alta coesão entre suas camadas internas. Este componente também possui a tríade de camadas básicas para gerenciamento de contexto em *Hermes*: comunicação, configuração e notificação.

Através do serviço de comunicação o HA interage com o restante do sistema de gerenciamento de contexto, *assinando os tópicos de interesse para agregação e criando os tópicos onde as informações de contexto agregadas serão publicadas*. As informações de tópicos para assinatura e publicação são gerenciadas pelo serviço de configuração. Após criados e devidamente configurados, o serviço de notificação pode receber o contexto assinado e publicar o contexto processado pelo HA nos respectivos tópicos.

O Código 3.3 apresenta o conteúdo do arquivo de configuração para um HA. Enquanto um HW precisa especificar somente o(s) tópico(s) onde irá publicar, um HA precisa configurar tanto os tópicos para publicação quanto os tópicos para assinatura, uma vez que depende dos dados de contexto representados por HW's para realizar seus serviços. No exemplo apresentado, o HA criado é configurado para assinar os tópicos para

os sinais vitais pressão arterial (VSO_0000005) e temperatura corpórea (VSO_0000008), e publicar o contexto agregado no tópico AGG_0508H.

Código 3.3 Arquivo de configuração para um *Hermes Aggregator*.

```
{   "topicos":[
        {"tipo":"notificacao",
         "registrar":"VSO_0000005",
         "assinar":[
                {"nome":"VSO_0000005"}]
        },
        {"tipo":"notificacao",
         "registrar":"VSO_0000008",
         "assinar":[
                {"nome":"VSO_0000008"}]
        },
        {"tipo":"notificacao",
         "registrar":"AGG_0508H",
         "publicar":[
                {"nome":"AGG_0508H",
                 "complementoTopico":"reason"}]
        }
    ]
}
```

Camada de Agregação

Cada informação de contexto recebida pelo componente *Hermes Aggregator* descreve uma parte específica da informação de contexto relacionada a uma *entidade*. Dessa forma, o contexto de uma entidade é o conjunto de informações que a descreve. Para o serviço de agregação de contexto, cada informação específica relacionada à descrição de uma entidade é denominada *propriedade*.

O componente HW, descrito na Seção 3.1, é responsável pela representação ontológica do contexto relacionado a uma propriedade, ou seja, uma parte específica do contexto de uma entidade, como: temperatura corpórea, pressão arterial, frequência de pulso, entre outras. Cada representação de contexto, produzida por um HW e recebida por um HA é considerada uma *instância*. Assim, cada propriedade de contexto de uma entidade pode ter diferentes instâncias associadas e, à medida em que há alterações de contexto no ambiente que são representadas, estas são enviadas para agregação.

Com base nesses conceitos, o serviço de agregação e o gerenciamento de contexto em *Hermes Aggregator* é realizado por 3 (três) gerentes de contexto, a saber:

- **Gerente de Entidades**: oferece a interface para criação de entidades que serão a base para a agregação de contexto.

- **Gerente de Propriedades**: toda entidade deve estar associada a um conjunto de *n* propriedades, que são as informações específicas de contexto que devem ser agregadas em um único modelo.
- **Gerente de Instâncias**: cada propriedade pode ter diversas instâncias de contexto relacionadas a si, onde cada instância representa uma informação de contexto recebida.

A associação de uma entidade a um conjunto de propriedades configura o escopo de uma agregação. O serviço de agregação pode gerenciar o contexto relacionado a *n* entidades, onde cada entidade possui seu conjunto de propriedades e cada propriedade seu próprio conjunto de instâncias.

A Figura 3.6 representa o serviço de agregação e seus respectivos gerentes de contexto.

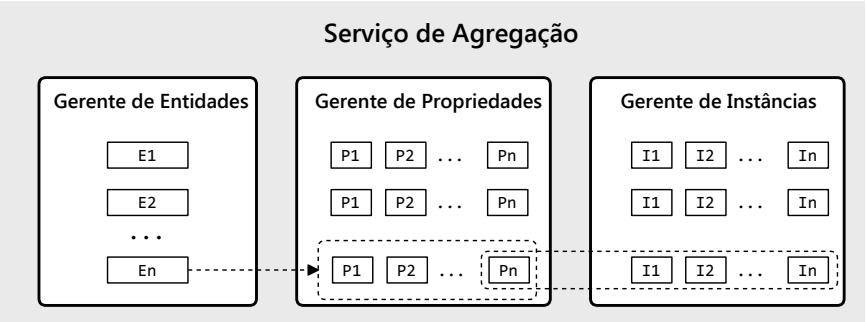


Figura 3.6: *Funcionamento do serviço de agregação de contexto.*

Por meio do serviço de configuração, é realizada uma validação a cada instância de contexto recebida pelo serviço de agregação, verificando se um novo modelo de agregação está completo. Quando o componente HA recebe uma nova instância de contexto para cada uma das propriedades especificadas para uma entidade, uma agregação de contexto é criada e notificada aos componentes e aplicações interessadas.

3.2.3 Validação do Serviço de Agregação de Contexto

Para validação do HA foi adotado o mesmo cenário de testes apresentado na Subseção 3.1.4, que tem como domínio o monitoramento de sinais vitais humanos.

Conversão de contexto SSN para um domínio de aplicação

Oferecendo um serviço de representação independente de domínio, HW pode representar, de modo geral, qualquer tipo de informação de contexto. Porém, uma aplicação ou componente para realização de raciocínio e inferências pode necessitar de informações de contexto representadas de acordo com o domínio ao qual este contexto pertence.

Por este motivo, além de agregar as diferentes partes do contexto associado a uma entidade, o serviço de agregação de contexto permite que seja realizada uma conversão das informações representadas com SSN pelos HW's para o domínio de aplicação a qual elas pertençam, com base em uma ontologia específica do domínio.

Para realizar essa conversão de contexto ontológico são utilizados os recursos semânticos do próprio padrão de representação RDF e também da linguagem de consulta SPARQL. Através da descrição do padrão de triplas que se deseja realizar a conversão entre os domínios e da realização de uma consulta do tipo CONSTRUCT, com base na representação SSN das informações de contexto, pode-se gerar um novo modelo representado de acordo com a ontologia de domínio, neste caso de testes, a MSVH.

O Código 3.4 apresenta a consulta SPARQL utilizada para converter as informações básicas da representação de contexto SSN para a representação baseada na ontologia MSVH.

Código 3.4 Consulta Construct para conversão da representação de contexto.

```
1    " PREFIX msvh: "
2  + " <http://www.semanticweb.org/ontologies/2013/1/Ontology1361391792831.owl#> "
3  + " PREFIX time: <http://linkserver.icmc.usp.br/ckonto/time#> "
4  + " PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#> "
5  + " CONSTRUCT "
6  + " { "
7  + " msvh:"+dateTime+" time:instantCalendarClockDataType ?dateTime . "
8  + " msvh:"+instance+" msvh:value"+property+" ?value ; "
9  + "                  msvh:unit"+property+" ?unit . "
10 + " } "
11 + " WHERE "
12 + " { "
13 + " ?sensing ssn:observationResultTime ?dateTime . "
14 + " ?instance ssn:hasOutputValue ?value ; "
15 + "           ssn:hasOutputUnit ?unit . "
16 + " } ";
```

Das linhas 1 a 4 são descritos os espaços de nomes utilizados dentro da consulta. As linhas 5 a 10 descrevem as triplas do modelo a ser construído, com base na ontologia MSVH. As variáveis concatenadas à consulta, por exemplo `dateTime` e `instance` representam as novas URI's que descrevem estes recursos, padronizadas com a MSVH. Nas demais linhas, de 11 a 16, são descritas as triplas equivalentes da representação SSN que deverão ser convertidas.

O serviço de agregação ainda oferece uma interface que permite a inserção de demais triplas à agregação de contexto, de maneira a validar essa representação em relação ao seu modelo ontológico e também aumentar a expressividade na descrição da agregação, conforme necessário.

Agregação de contexto ontológico com MSVH

No serviço de agregação de contexto utilizando a ontologia MSVH, cada entidade (paciente) criada para o processo de agregação é um indivíduo da classe `msvh:Person`. Da mesma forma, para cada propriedade de contexto sensoreada (temperatura corpórea, pressão arterial, etc.) é criado um indivíduo da classe específica para o seu tipo na MSVH. Cada instância de contexto (aferição de um sensor) está relacionada a uma respectiva propriedade que, por sua vez, está ligada a uma entidade que é o ponto central de um modelo de agregação.

Dessa forma, cada agregação relacionada a uma entidade é um conjunto de instâncias das propriedades de contexto que foram previamente configuradas para agregação. Cada nova instância de contexto recebida pelo HA é adicionada ao modelo de agregação a qual pertence, que é então verificado e, se completo de acordo com as configurações definidas, é disponibilizado para os assinantes da respectiva agregação.

O Código 3.5 apresenta um exemplo de agregação dos sinais vitais temperatura corpórea e pressão arterial para um determinado paciente. De acordo com a MSVH, a representação ontológica do monitoramento de um sinal vital relaciona ao modelo de contexto as seguintes informações: *i*) *participantes do monitoramento*, que podem ser o paciente monitorado e também os profissionais de saúde envolvidos; *ii*) o *horário da aferição do contexto*; e *iii*) o *tipo do sinal vital monitorado*; e *iv*) o *contexto aferido pelo sensor*, que inclui o valor lido e a unidade da medição para o sinal vital.

No exemplo apresentado, os principais recursos que descrevem o monitoramento dos sinais vitais agregados são:

- `msvh:VSO_0000005` e `msvh:VSO_0000008`: identifica, o tipo do sinal vital agregado, respectivamente *pressão arterial* e *temperatura corpórea*;
- `activity:hasParticipant`: relaciona o indivíduo de monitoramento à entidade cujos sinais vitais estão sendo monitorados, identificada pela URI `041n-9525de38`;
- `msvh:unitTemperature` e `msvh:unitBloodPressure`: relaciona uma instância de contexto à unidade de medida utilizada para identificar a escala a que pertence uma medição, neste caso, "`Celsius`" para temperatura corpórea e "`mmHg`" para pressão arterial;
- `tEvent:startDateTime`: propriedade que representa o instante em que a leitura da medição foi realizada.

A Figura 3.7 apresenta o grafo relativo à agregação de contexto do Código 3.5. As cores e tamanho dos nós, bem como a cor das arestas, seguem as mesmas regras aplicadas ao grafo da Figura 3.4, apresentado na Subseção 3.1.4.

Código 3.5 Agregação de contexto utilizando a ontologia MSVH.

```
msvh:041n-9525de38-3a06-42fe-abd7-b15315b4af3d>
    a   msvh:Person .

msvh:InstanceBloodPressure-7b59ca07-72b5-4866-a9d3-83cdf141deff
    a   msvh:VSO_0000005 ;
    msvh:isMeasurementBloodPressure
        msvh:bloodPressure-d361ea6f-b865-48af-9f12-3ab9f20ee6ac ;
    msvh:unitBloodPressure
        "mmHg"^^xsd:string ;
    msvh:valueDiastolicBloodPressure
        "56"^^xsd:nonNegativeInteger ;
    msvh:valueSystolicBloodPressure
        "111"^^xsd:nonNegativeInteger .
msvh:InstanceTemperature-5f3db9b1-aecb-4ab4-8c9e-bcc05ff23d9a
    a   msvh:VSO_0000008 ;
    msvh:isMeasurementTemperature
        msvh:temperature-6f3d4796-8c00-42fc-a00f-4d7ce4ff9897 ;
    msvh:unitTemperature
        "Celsius"^^xsd:string ;
    msvh:valueTemperature
        "37.300"^^xsd:float .

msvh:MonitoringBloodPressure-29fdba4b-5334-4400-a874-7554a9cb437e
    a   msvh:MonitoringBloodPressure ;
    acti:hasParticipant
                msvh:041n-9525de38-3a06-42fe-abd7-b15315b4af3d> ;
    tEvent:startDateTime
        msvh:DateTime-0aeaf604-d01a-4032-88c9-d654c3e91d54 ;
    msvh:hasMonitoringBloodPressure
        msvh:bloodPressure-d361ea6f-b865-48af-9f12-3ab9f20ee6ac .
msvh:MonitoringTemperature-5faf4c44-d489-41f8-9370-0468727a5135
    a   msvh:MonitoringTemperature ;
        acti:hasParticipant
                msvh:041n-9525de38-3a06-42fe-abd7-b15315b4af3d> ;
    tEvent:startDateTime
        msvh:DateTime-e7df2130-6fd8-4205-a674-3766d837598b ;
    msvh:hasMonitoringTemperature
        msvh:temperature-6f3d4796-8c00-42fc-a00f-4d7ce4ff9897 .

msvh:DateTime-0aeaf604-d01a-4032-88c9-d654c3e91d54
    a   tEvent:InstantEvent ;
    time:instantCalendarClockDataType
        "2016-05-20T15:15:26.377-03:00"^^xsd:dateTime .
msvh:DateTime-e7df2130-6fd8-4205-a674-3766d837598b
    a   tEvent:InstantEvent ;
    time:instantCalendarClockDataType
        "2016-05-20T15:15:50.333-03:00"^^xsd:dateTime .
```

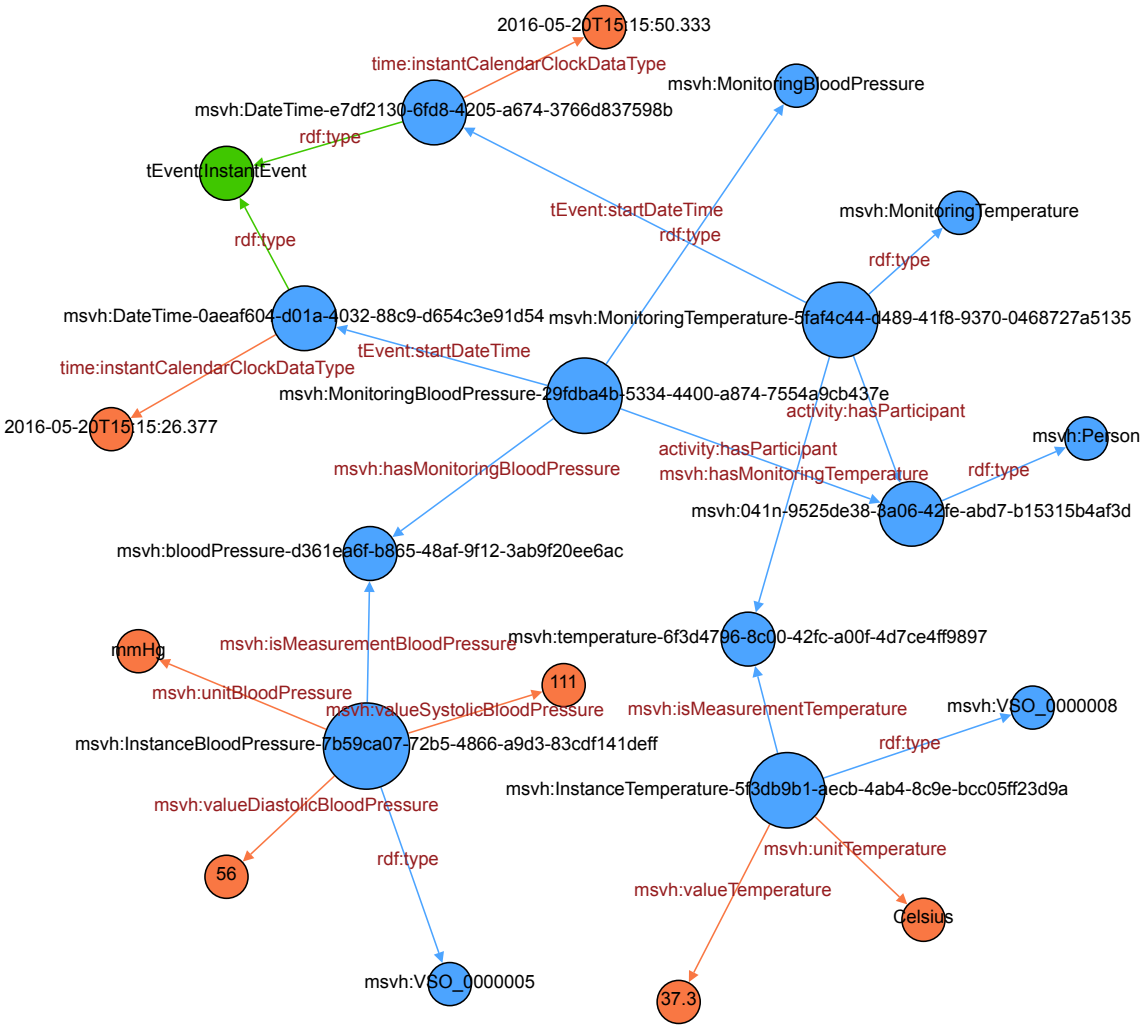


Figura 3.7: *Grafo da agregação de contexto utilizando MSVH.*

Os demais sinais vitais que compõem este cenário foram representados segundo o mesmo modelo, baseado na MSVH.

3.3 *Hermes History*: histórico, consultas e gerenciamento ao acesso de contexto ontológico

Hermes History (HH) é o componente que oferece os serviços de persistência e consulta na infraestrutura *Hermes*. Além disso, é responsável por gerenciar o acesso ao contexto processado pelos demais componentes. A persistência de contexto é realizada em três diferentes níveis de expressividade: representação ontológica (HW), agregação (HA) e raciocínio de contexto (HI). Esses níveis de expressividades são separados em diferentes repositórios, cujo histórico pode ser explorado por meio de consultas.

Todo contexto gerenciado pelo CMS *Hermes*, seja pelo serviço de representação ontológica, de agregação ou de raciocínio, deve ser persistido para consultas e serviços

baseados em histórico. Por este motivo, todo o fluxo de contexto de *Hermes* deve passar pelo HH, persistindo toda informação antes que esta seja disponibilizada, garantindo assim a consistência do contexto.

3.3.1 Requisitos

1. **Receber contexto ontológico como entrada de dados**: o HH deve receber dados de contexto devidamente representados e serializados no formato RDF, com base em ontologias previamente especificadas, de acordo com o domínio de aplicação.
2. **Identificar e persistir o contexto de acordo com a sua origem e nível de expressividade**: as fontes de contexto do HH são os três componentes que também representam os diferentes níveis de expressividade de contexto na infraestrutura *Hermes*; o HW, que oferece contexto representado com base em ontologias; o HA, que fornece contexto agregado de diferentes fontes sobre uma mesma entidade; e o HI, que entrega novas informações de contexto oriundas de inferências sobre contexto representado e/ou agregado.
3. **Oferecer interface para consulta e recuperação de informações de contexto**: o HH deve fornecer uma interface para facilitar a realização de diferentes tipos de consultas semânticas aos dados do histórico.
4. **Garantir a consistência entre os dados persistidos e disponibilizados**: o HH deve gerenciar o contexto processado pelos demais componentes de *Hermes* para que nenhuma informação seja publicada a consumidores antes de ser devidamente persistida para consultas posteriores.

3.3.2 Arquitetura

A Figura 3.8 apresenta a arquitetura em camadas do HH.

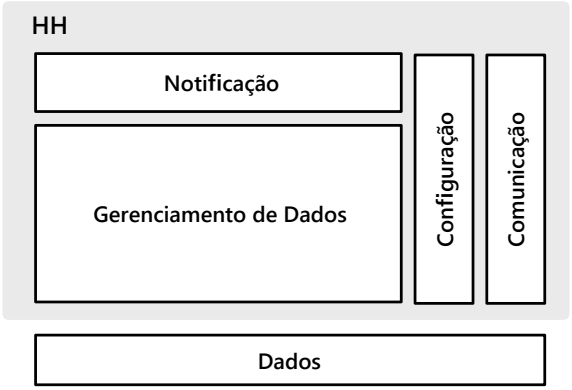


Figura 3.8: *Arquitetura em camadas do Hermes History.*

Como nos demais componentes apresentados, ele conta com as camadas de comunicação, configuração e notificação. Os serviços de histórico e consulta são realizados pela camada de gerenciamento de dados, que se comunica com um repositório de dados.

Gerenciamento de Acesso a Informações de Contexto

Para garantir que toda informação de contexto gerenciada por *Hermes* seja persistida para histórico, foi criado um *sistema para gerenciamento de tópicos que envolve os serviços de comunicação, configuração e notificação* de HH. A ideia central desse sistema é que toda informação de contexto processada por *Hermes*, antes de ser notificada às aplicações ou componentes assinantes, seja persistida para consultas e histórico.

Como os demais componentes de *Hermes*, o HH utiliza os serviços de comunicação oferecidos por *Hermes Base*. Estes serviços incluem a criação de tópicos para interoperabilidade de contexto entre os componentes, e entre estes e as aplicações. Os tópicos utilizados para comunicação de contexto em *Hermes*, denominados *tópicos de notificação* são classificados em dois tipos: *tópicos públicos* e *tópicos privados*.

- **Tópicos públicos**: são tópicos acessíveis a todos os componentes e aplicações interessados em uma determinada informação de contexto. Todo contexto publicado neste tipo de tópico deve, obrigatoriamente, já ter sido persistido para garantir a consistência das notificações com o histórico.
- **Tópicos privados**: são aqueles onde toda informação de contexto processada por *Hermes* é inicialmente publicada. Por padrão, o único assinante de tópicos privados em *Hermes* é o HH, que realiza a persistência das informações recebidas, e então republica o contexto em tópicos públicos.

A tarefa de gerência de tópicos em HH, bem como nos demais componentes de *Hermes*, é dividida entre os serviços de comunicação, configuração e notificação. Inicialmente, para a criação dos tópicos, públicos ou privados, é necessária uma instância do serviço de comunicação que, por sua vez, utiliza as definições geradas através do serviço de configuração. Com os tópicos já criados e devidamente configurados, o serviço de notificação pode ser utilizado para entrega de contexto aos componentes e aplicações assinantes.

Diferentes situações ou cenários podem surgir de acordo com a utilização dos componentes de *Hermes*. Um HA precisa das informações de contexto processadas por HW's para realização de sua função. Já um HI necessita ou do contexto representado por HW's, ou de contexto agregado por HA's.

Neste interim, uma vez que todas as informações de contexto processadas por componentes de *Hermes* precisam ser persistidas antes de serem notificadas para os componentes/aplicações assinantes, o componente *Hermes History*, em conjunto com

o *Hermes Base*, opera como um *blackboard* de contexto para a infraestrutura *Hermes*. Assim, toda informação de contexto é publicada primeiro em tópicos privados, que são assinados exclusivamente pelo componente *Hermes History*. O contexto recebido por tópicos privados é persistido em repositórios de contexto ontológico, e depois é republicado em tópicos públicos, que são assinados pelos demais componentes.

A Figura 3.9 apresenta a estrutura *blackboard* de tópicos em HH. Todos os tópicos privados da infraestrutura de contexto são assinados por HH, que recebe as publicações de HW's, HA's e HI. De maneira análoga, HH também é publicador de todos os tópicos públicos, sendo que, para todo tópico privado assinado há um tópico público relativo, pelo qual as informações são disponibilizadas para os demais componentes e aplicações após serem persistidas.

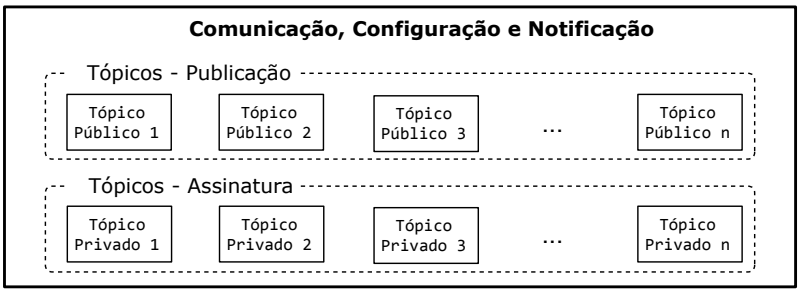


Figura 3.9: *Blackboard de tópicos em Hermes History.*

A Figura 3.10 apresenta o esquema geral para gerenciamento de acesso ao contexto em *Hermes*. É válido ressaltar que o componente *Hermes Base*, representado nesta e nas demais figuras a seguir, é abstraído por cada um dos demais componentes na forma de uma camada de comunicação. Uma vez que todos os demais componentes e também aplicações necessitam do serviço de comunicação, para demonstrar as etapas de comunicação, *Hermes Base* foi representado como o componente central no gerenciamento de tópicos.

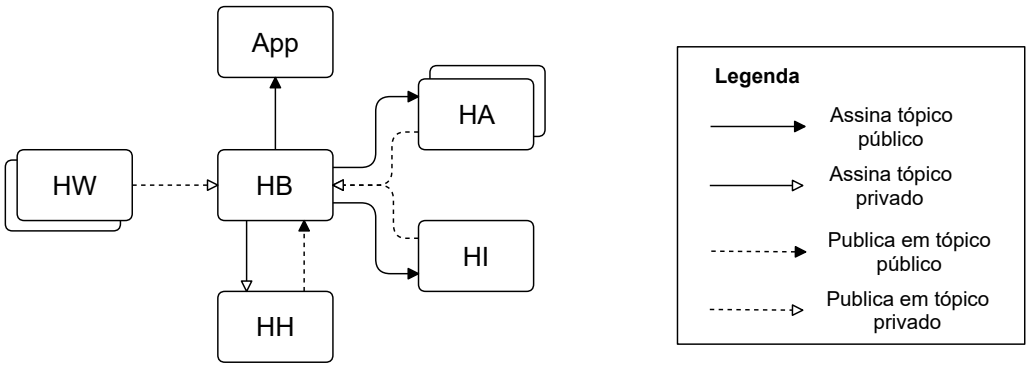


Figura 3.10: *Esquema geral de acesso ao contexto em Hermes.*

A partir dessa representação podem ser identificadas as características gerais que são comuns a qualquer cenário em que *Hermes* pode ser utilizado, a saber:

- ***Hermes Widget***: *sempre publica em tópicos privados*, assinados por HH e não assina nenhum tópico.
- ***Hermes Aggregator*** e ***Hermes Interpreter***: *sempre publicam em tópicos privados*, da mesma forma que o HW, porém *sempre assinam tópicos públicos* para receber o contexto de entrada necessário para a realização de seus serviços.
- ***Hermes History***: é o único que *assina todos os tópicos privados* do sistema, e o único que *publica em tópicos públicos*, que podem ser assinados pelos demais componentes e aplicações, e que são o meio para que estes recebam contexto de entrada.
- **Aplicações**: *sempre assinam tópicos públicos*, onde HH publica contexto processado pelos demais componentes, de acordo com as configurações especificadas.

Partindo desde esquema geral, três tipos principais de cenários podem ser produzidos de acordo com a combinação dos componentes de *Hermes*.

a) Cenário 1

No primeiro cenário, que é também o mais simples, o foco são as instâncias do componente HW, que recebem contexto de sensores, representam ontologicamente e publicam essas informações (1). Neste cenário, HH assina os tópicos privados onde cada HW publica (2), persiste o contexto e então publica novamente este contexto em tópicos públicos assinados pelas aplicações (3), que recebem o contexto (4).

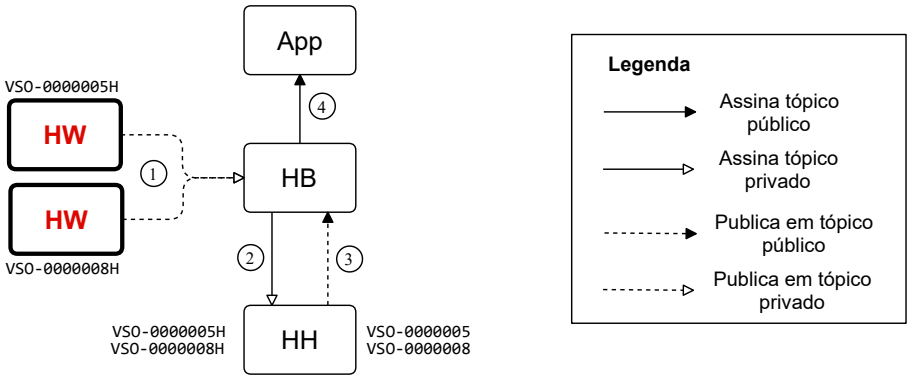


Figura 3.11: *Esquema de tópicos para o Cenário 1.*

b) Cenário 2

No segundo cenário, apresentado na Figura 3.12 é acrescentado o HA. No primeiro passo, HWs representam o contexto e o publicam em tópicos privados (1), assinados por HH (2). Em seguida, o componente de histórico persiste as informações e as republica em tópicos públicos (3), assinados pelo componente HA (4). Assim que o serviço de agregação é realizado, HA publica o contexto agregado em um tópico privado

(5), assinado por HH (6), que realiza novamente o processo de persistência e publica o contexto agregado em um tópico público (7), que é assinado por aplicações sensíveis a contexto (8).

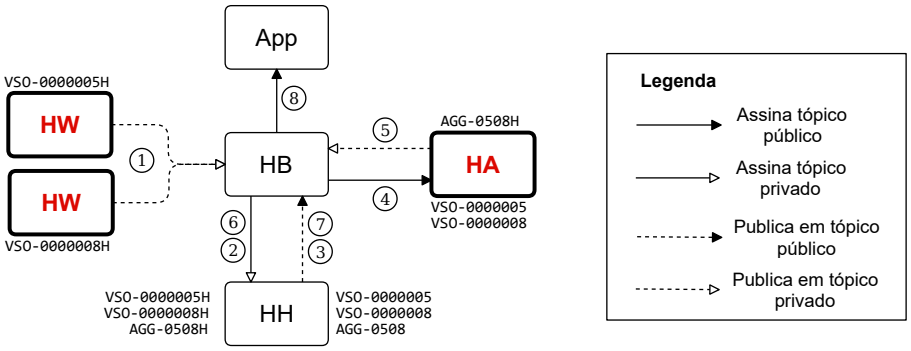


Figura 3.12: *Esquema de tópicos para o Cenário 2.*

c) Cenário 3

O último cenário acrescenta o HI à infraestrutura de contexto. Este cenário possui duas variações, sendo a primeira sem o serviço de agregação, e a segunda com a utilização do HA. Da mesma forma que nos cenários anteriores a informação de contexto é inicialmente representada pelos *Hermes Widgets*, publicada em tópicos privados (1) e assinada pelo *Hermes History* (2). Sem o serviço de agregação, na Figura 3.13, o componente de histórico publica o contexto em um tópico público (3), que é assinado pelo HI (4). O serviço de raciocínio é realizado e o contexto inferido é publicado em um tópico privado (5) assinado por HH (6), que persiste as novas informações e publica em tópico público (7), para consumo das aplicações (8).

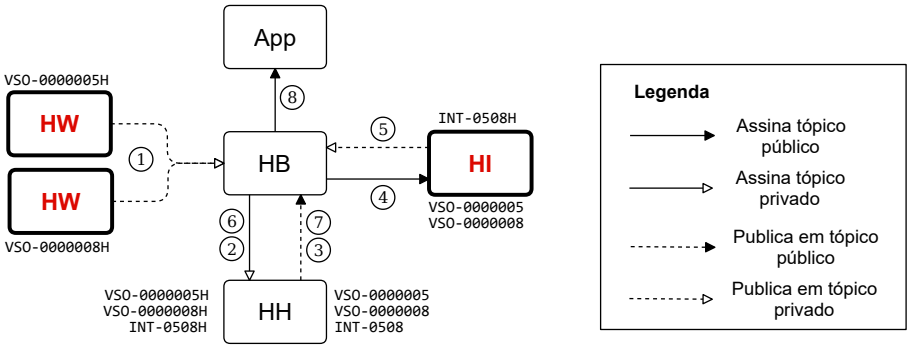


Figura 3.13: *Esquema de tópicos para o Cenário 3, exemplo 1.*

Com a utilização do serviço de agregação, apresentado na Figura 3.14 o componente *Hermes History* publica o contexto recebido dos *Hermes Widgets* (2) em tópicos públicos (3) que são assinados por um *Hermes Aggregator* (4). Depois de agregado e persistido (5) (6) (7), de maneira idêntica ao Cenário 2, o contexto é recebido pelo componente *Hermes Interpreter* (8), que realiza seus serviços e publica as novas informações

em um tópico privado (9) assinado por *Hermes History* (10), que persiste e republica o contexto para as aplicações (11).

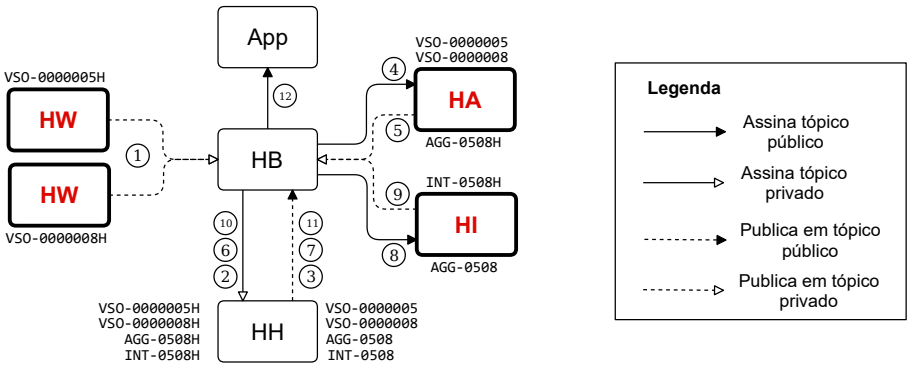


Figura 3.14: *Esquema de tópicos para o Cenário 3, exemplo 2.*

Camada de Gerenciamento de Dados

O núcleo do serviço de gerenciamento de dados, que também pode ser descrito como serviço de histórico e consultas, é constituído de três gerentes:

- **Gerente de Atualizações**: recebe o contexto publicado em tópicos privados e envia este contexto para o gerente de histórico;
- **Gerente de Histórico**: recebe o contexto do gerente de atualizações e acessa a camada de dados, persistindo as informações em seu formato padrão RDF em repositórios de contexto.
- **Gerente de Consultas**: oferece interface para a execução de consultas SPARQL aos repositórios de contexto, na camada de dados.

A Figura 3.15 apresenta uma visão interna do serviço de histórico e consultas do componente *Hermes History*, com os gerentes que o compõem.

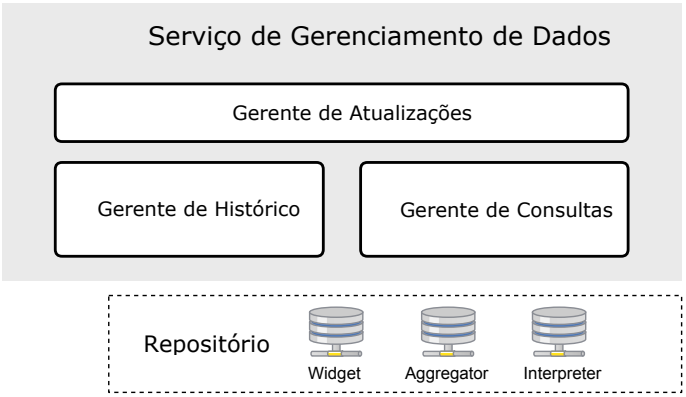


Figura 3.15: *Serviço de histórico e consultas.*

Este serviço comunica com uma camada de dados, onde as informações recebidas são persistidas em formato nativo, ou seja, na forma de triplas (sujeito, predicado,

objeto) do padrão RDF. Também através deste serviço, mais especificamente através do gerente de consultas, as informações já persistidas podem ser recuperadas, de acordo com as necessidades dos componentes ou aplicações interessadas.

O Código 3.6 apresenta um exemplo de consulta às informações de contexto representadas por HW's, e armazenadas no seu respectivo repositório.

Código 3.6 Consulta ao repositório de HW, para representações baseadas em SSN, buscando o(s) valor(es) aferido(s) para todas as instâncias.

```
1  PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
2  SELECT ?person ?property ?outputValue ?outputValueAux ?outputUnit ?data
3  WHERE {
4          ?observation ssn:featureOfInterest ?person .
5          ?observation ssn:observedProperty ?property .
6          ?observation ssn:observationResultTime ?data .
7          ?observation ssn:observationResult ?result .
8          ?result ssn:hasValue ?value .
9          ?value ssn:hasOutputValue ?outputValue .
10         ?value ssn:hasOutputUnit ?outputUnit .
11         OPTIONAL {?value ssn:hasOutputValueAux ?outputValueAux} .
12 }
```

Por ser uma consulta a uma representação totalmente baseada em SSN, este é único espaço de nomes declarado, na linha 1. Na linha 2 são descritos os valores a serem retornados pela consulta, através da cláusula `SELECT`, neste caso: nome do paciente, propriedade observada, valor de saída do sensoriamento, valor de saída auxiliar (opcional, para medições que possuem dois valores que compõem uma mesma aferição), unidade de medida e a data/hora da aferição.

Nas linhas 3 a 12 é descrito o padrão de triplas buscado pela consulta, dentro da cláusula `WHERE`. A tripla apresentada na linha 11, dentro da cláusula `OPTIONAL` é opcional, ou seja, não elimina os resultados que não a possuem. Esse exemplo é válido para a busca de informações como pressão arterial, cujo resultado do sensoriamento é composto por dois valores (sistólica e diastólica). Para as demais representações de contexto essa tripla não é utilizada.

O Código 3.7 apresenta um exemplo de consulta criada para acessar as informações do repositório do HA e, neste caso específico, o contexto de pressão arterial.

Código 3.7 Consulta ao repositório de HA, para representações baseadas em MSVH, buscando aferições do sinal vital *pressão arterial*.

```
1  PREFIX msvh: <http://www.semanticweb.org/ontologies/2013/1/Ontology1361391792831.owl#>
2  PREFIX acti: <http://linkserver.icmc.usp.br/ckonto/activity#>
3  PREFIX tEvent: <http://linkserver.icmc.usp.br/ckonto/tEvent#>
4  PREFIX time: <http://linkserver.icmc.usp.br/ckonto/time#>
5  SELECT ?person ?valorSistolica ?valorDiastolica ?pressure ?unidade ?valorData
6  WHERE {
7          ?monitoring acti:hasParticipant ?person .
8          ?monitoring msvh:hasMonitoringBloodPressure ?bloodPressure .
9          ?monitoring tEvent:startDateTime ?date .
10         ?pressure msvh:isMeasurementBloodPressure ?bloodPressure .
11         ?pressure msvh:valueSystolicBloodPressure ?valorSistolica .
12         ?pressure msvh:valueDiastolicBloodPressure ?valorDiastolica .
13         ?pressure msvh:unitBloodPressure ?unidade .
14         ?date time:instantCalendarClockDataType ?valorData .
15 }
```

As linhas 1 a 4 da consulta declaram os espaços de nomes utilizados. O principal prefixo declarado neste exemplo é o `msvh:`, que representa a URI da ontologia de monitoramento de sinais vitais humanos. Os demais prefixos representam as ontologias auxiliares *Activity*, *Temporal Event* e *Time*, importadas pela MSVH.

A linha 5, através da cláusula `SELECT`, identifica quais valores devem ser retornados pela consulta. Neste exemplo a consulta deverá retornar o nome do paciente, os valores das aferições de pressão sistólica e diastólica, o identificador da instância representada, a unidade de medida do valor aferido e a data completa relativa ao instante da aferição. Nas linhas 6 a 16 é descrita a conjunção ou o padrão de triplas que deve ser localizado para atender a consulta.

A nível de implementação, para oferecer acesso distribuído aos repositórios de contexto foi utilizado o *endpoint* SPARQL Fuseki, parte da API Jena. O Fuseki permite a criação e gerenciamento de repositórios TDB, oferecendo serviços de atualização, remoção e também de consultas a estes repositórios.

3.4 Considerações Finais

Este capítulo apresentou os serviços de representação, agregação e histórico de contexto, respectivamente realizados pelos componentes *Hermes Widget*, *Hermes Aggregator* e *Hermes History*. Foram descritos os requisitos e arquitetura de cada componente, bem como detalhes do funcionamento interno dos serviços projetados e implementados. A Tabela 3.2 apresenta um resumo sobre os componentes: requisitos, projeto arquitetural e detalhes de implementação.

Tabela 3.2: *Requisitos e solução de implementação dos serviços propostos.*

Comp.	Requisito	Camada	Implementação	Solução de projeto
Hermes Widget	Permitir a especificação de um sensor e do sensoriamento	Representação	*HW Representation Service Sensor*	*Jena RDF*
	Representação formal e expressiva do contexto (baseada em ontologias)	Representação	*HW Representation Service Domain*	*Jena RDF*
	Disseminação de contexto através de tópicos	Comunicação e Notificação	*HW Communication Service* e *HW Notification Service*	*Singleton*, *CoreDX DDS*
Hermes Aggregator	Permitir a especificação de uma entidade	Agregação	*HA Entity Model*, *HA Property Model* e *HA Instance Model*	*Jena RDF*
	Agregar contexto de diferentes provedores sobre uma mesma entidade	Agregação	*HA Aggregation Service*	*Jena RDF*
	Elevar o nível expressividade e diminuir a granularidade do contexto	Agregação	*HA Aggregation Service*	*Jena RDF*
	Reduzir a complexidade no desenvolvimento de aplicações sensíveis a contexto	Agregação	—	—
Hermes History	Receber contexto ontológico como entrada de dados	Histórico	*HH History Service*	
	Identificar e persistir o contexto de acordo com a sua origem e nível de expressividade	Histórico	*HH History Service*	*Jena TDB*
	Oferecer interface para consulta e recuperação de contexto	Histórico	*HH History Service*	*Jena Fuseki (Endpoint SPARQL)*
	Garantir a consistência entre os dados persistidos e disponibilizados	Histórico, Configuração e Notificação	*HH History Service*, *HH Configuration Service* e *HH Notification Service*	*Jena Fuseki (Endpoint SPARQL)*
Hermes	Manutenabilidade	Geral	*HW HA e HH Facade*, *HW HA e HH Transfer Object* e *HW HA e HH Communication Service*	*Facade*, *Transfer Object*, *Factory*, *Domain Object* e *Singleton*
	Escalabilidade	Geral	—	—
	Interoperabilidade	Comunicação, Configuração e Notificação	*HW HA e HH Communication Service*	*Jena RDF*
	Extensibilidade	Geral	—	—

CAPÍTULO 4

Experimentação e Avaliação

Na etapa de modelagem do ciclo de vida do contexto, com ênfase no serviço de representação ontológica, além dos requisitos funcionais destacados na Seção 3.1, os requisitos não funcionais de qualidade de software possuem grande relevância para atender as demandas de aplicações sensíveis ao contexto [39]. Estes requisitos devem ser tratados e atendidos pelos componentes de um CMS responsáveis por oferecer os serviços relacionados a esta etapa.

Cada característica relevante da qualidade de um produto de software deve ser especificada e avaliada utilizando medidas validadas ou consensualmente aceitas [25]. No componente *Hermes Widget*, projetado para atuar em diferentes domínios de aplicação, as características de qualidade devem ser devidamente tratadas para garantir a eficácia na realização do serviço de representação ontológica e demais serviços oferecidos. Da mesma forma, os demais componentes também devem cumprir na prática as obrigações determinadas por seus requisitos, especificados no Capítulo 3.

A primeira parte desta seção apresenta o processo para desenvolvimento de aplicações sensíveis a contexto baseadas em *Hermes*, criadas para validar a utilização dos serviços oferecidos pelos componentes desta proposta. Em seguida são apresentadas as avaliações dos serviços de representação (HW) e histórico de contexto (HH). O foco destas avaliações foi o requisito não funcional de *escalabilidade*. Complementando este capítulo, foi ainda realizada uma avaliação da manutenabilidade do componente *Hermes Widget*, que é apresentada no Apêndice A.

4.1 Desenvolvimento de aplicações sensíveis a contexto

Esta seção apresenta a estrutura de uma aplicação sensível a contexto desenvolvida com base em *Hermes*, e a sua integração com o CMS para utilização dos serviços oferecidos.

4.1.1 Arquitetura de aplicações *Hermes*

Apresentado o cenário em que os serviços propostos foram experimentados, foi construída uma aplicação cujo objetivo é consumir o contexto processado por *Hermes*. Essa aplicação assina as informações de contexto desejadas, e utiliza de forma transparente os serviços de representação, agregação e histórico de contexto.

A Figura 4.1 apresenta a estrutura de uma aplicação sensível a contexto desenvolvida para utilizar o CMS *Hermes*:

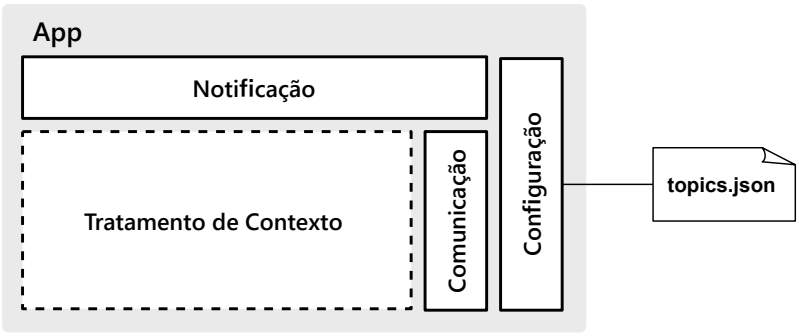


Figura 4.1: *Arquitetura em camadas de uma aplicação sensível a contexto.*

Uma aplicação sensível a contexto neste trabalho é composta das seguintes camadas:

1. Camada de comunicação: realiza a interface com o componente de comunicação *Hermes Base*;
2. Camada de configuração: realiza a leitura das configurações de assinatura de tópicos de uma aplicação, discriminando os tópicos assinados por esta;
3. Camada de notificação: implementa um *listener* que utiliza a camada de comunicação para recebimento do contexto publicado pelos componentes de *Hermes*;
4. Camada de tratamento de contexto: realiza as operações sobre o contexto recebido.

As camadas de comunicação, configuração e notificação das aplicações são denominadas em conjunto como *base de aplicações*, uma vez que todas as aplicações baseadas em *Hermes* as utilizam como padrão de projeto. Dessa forma, os desenvolvedores de aplicação podem manter o foco na camada de tratamento de contexto, implementando a utilização deste contexto de acordo com os objetivos da aplicação.

4.1.2 Integração com *Hermes* e os serviços de contexto

Para atender as demandas das aplicações, é necessário instrumentar o experimento com os componentes desenvolvidos: *Hermes Widget(s)*, para representação do

contexto aferido; *Hermes Aggregator(s)* para combinação das informações; e *iii*) *Hermes History* para armazenar as informações representadas e agregadas.

A integração de uma aplicação com o CMS *Hermes* e os serviços disponibilizados por este acontece através do componente de comunicação. Todos os componentes, e também aplicações que compõem o ambiente de contexto utilizam o HB, como demonstrado anteriormente na Figura 2.8.

A partir deste projeto, os componentes ficam totalmente desacoplados entre si, mesmo havendo a comunicação direta conforme é demonstrado na arquitetura apresentada na Seção 2.3. Assim, qualquer alteração de projeto ou implementação no componente de comunicação não gera qualquer impacto para os demais componentes, desde que sua interface seja mantida.

4.2 Avaliação do componente *Hermes Widget*

Além da característica de manutenabilidade, é necessário analisar a escalabilidade do componente *Hermes Widget* quando executado em um ambiente real (ou em simulações de ambientes reais), situação na qual este componente deve lidar com os dados de contexto fornecidos pelos provedores. A escalabilidade está incluída no escopo da sub-característica *adaptabilidade* do modelo de qualidade apresentado na ISO/IEC 25010 [25].

No cenário de monitoramento de sinais vitais, e em diversos outros cenários de contexto onde *Hermes Widget* pode ser aplicado, uma das situações recorrentes é a alta frequência com que os dados são recebidos para serem representados. Com base nesta constatação, o objetivo desta avaliação é verificar a escalabilidade do componente *Hermes Widget* ao receber grandes quantidades de contexto e realizar o serviço de representação ontológica. Neste intuito foram lançadas as seguintes hipóteses para esta avaliação:

1. O componente *Hermes Widget* é escalável quanto ao gerenciamento de dados de contexto recebidos de provedores.
2. O serviço de representação ontológica é escalável quanto ao número de solicitações recebidas.
3. Os tempos para representação ontológica mantém crescimento linear dado o aumento exponencial de dados de contexto.

Por provedores entende-se neste trabalho qualquer tipo de sensor que obtenha ou produza dados através de suas aferições. Nesta avaliação foram utilizadas as simulações de sensores para o cenário de monitoramento de sinais vitais. Apesar destes sensores simulados não coletarem aferições diretamente dos pacientes, os dados transmitidos por eles são obtidos de uma base de dados de pacientes reais que foram monitorados em

UTIs. Esta base de dados, denominada MIMIC (Seção 3.1), traz as informações de monitoramento do paciente, tais como horário da aferição, intervalo entre aferições, sinal vital monitorado, entre outras.

No cenário de monitoramento de sinais vitais, os dados de contexto podem ser monitorados em tempo real ou com uma frequência mínima estipulada por profissionais especializados para cada caso específico, uma vez que as alterações podem refletir uma situação de risco iminente ao estado de saúde de um paciente. Por este motivo o componente *Hermes Widget deve garantir a representação do contexto independente da frequência com que este seja recebido* e *permitir que a notificação do contexto ontológico representado seja realizada dentro do tempo de resposta adequado*.

4.2.1 Configurações dos experimentos

Na avaliação de escalabilidade foram utilizados os *Hermes Widgets* implementados para cada um dos 5 sinais vitais modelados na MSVH. Neste cenário, cada paciente da base MIMIC é associado a instâncias de *Hermes Widgets* específicas de acordo com os sinais vitais contemplados em seus registros. Dessa forma, se um paciente possui 5 sinais vitais em seu monitoramento, terá 5 *Hermes Widgets* associados a ele.

A base de dados MIMIC possui diferentes versões de arquivos de monitoramento de sinais vitais, com diferentes parâmetros e frequência de aferição. Para esta avaliação foi utilizada uma versão que realizou a persistência das aferições do monitoramento de sinais vitais segundo a segundo (tempo real). Esta escolha se deu no intuito de se trabalhar com um número expressivo de medições para verificar a escalabilidade do *Hermes Widget* quanto ao processamento de grandes volumes de dados.

Foram utilizados para os experimentos os dados de 15 pacientes que possuem informações de monitoramento relativas ao intervalo de 12 horas (em média 42 mil aferições por sinal vital para cada paciente). Este número de pacientes foi escolhido para simular o ambiente de uma UTI real e por fornecer dados de monitoramento suficientes para verificar a escalabilidade do componente *Hermes Widget*.

Todos os dados processados nas simulações realizadas são dados reais de pacientes da base MIMIC. Porém, no intuito de verificar a escalabilidade do componente, o volume de dados enviados para processamento pelos *Hermes Widgets* foi maior que o de um ambiente de monitoramento real.

Os dados de pacientes da base MIMIC utilizada foram monitorados e coletados com o intervalo real de uma aferição por segundo. Para este teste de escalabilidade, no entanto, estes dados foram enviados para o componente *Hermes Widget* em lotes de 30, 300, 3000 e 30000 aferições para realização do serviço de representação ontológica. Após

representada, cada aferição se torna um modelo de contexto que é notificado de acordo com a frequência real do monitoramento.

Cada aferição representada por um *Hermes Widget* neste cenário gera um modelo ontológico que possui em média 30 triplas, as quais descrevem a aferição do sinal vital, os dados da aquisição e demais informações de contexto do paciente. Como descrito nas hipóteses, pretende-se avaliar a escalabilidade do componente *Hermes Widget* para representar grandes volumes de dados ontológicos, que tem como base o modelo de triplas.

No intuito de simular o ambiente mais completo de gerenciamento de contexto, foram inseridas aplicações assinantes e o componente de raciocínio *Hermes Interpreter*. Porém, em se tratando de uma avaliação do componente *Hermes Widget*, os tempos de execução para os demais elementos (aplicações e *Hermes Interpreter*) não foram averiguados neste trabalho. Para o componente *Hermes Interpreter* foram assumidos os tempos de interpretação e filtragem do contexto apresentados por Maranhão [33], que avalia este componente.

Na realização dos experimentos foram utilizadas as seguintes máquinas para os respectivos componentes:

- *Hermes Widgets*: Windows 10 de 64 bits, com processador Intel(R) Core(TM) i7-3537U, de 2,00GHz cada e memória RAM de 8GB (foram instanciados 61 *Hermes Widgets* no total, não simultaneamente);
- *Hermes Interpreter*: Windows 7 de 64 bits, com processador Intel(R) Core(TM) 2 Duo, de 2,2GHz cada e memória RAM de 4GB (foi utilizado apenas uma instância do *Hermes Interpreter* para os testes);
- *Aplicações sensíveis a contexto*: Windows 7 de 64 bits, com processador Intel(R) Core(TM) 2 Duo, de 2,2GHz cada e memória RAM de 4GB (foram instanciadas duas aplicações com parâmetros diferentes para assinatura).

4.2.2 Análise dos resultados

A representação ontológica de diferentes sinais vitais gera modelos com número de triplas distintos, uma vez que, mesmo sendo modelados de acordo com a mesma ontologia, informações diferentes precisam ser representadas para cada sinal vital. Por exemplo, para a representação de temperatura, apenas uma aferição é representada em cada modelo. No entanto, para pressão sanguínea, é necessário que sejam incluídas em uma mesma representação as aferições de pressão sanguínea sistólica e diastólica (sendo ainda desejável a pressão sanguínea média), aumentando dessa forma a quantidade de triplas representadas.

A Tabela 4.1 apresenta os tempos de representação coletados de *Hermes Widgets* para os sensores de pressão sanguínea e temperatura, para diferentes lotes de entradas. Esta comparação foi realizada no intuito de demonstrar a diferença entre os tempos de processamento para os *Hermes Widgets* que realizam a representação do maior e do menor número de triplas, respectivamente para as aferições de pressão sanguínea e de temperatura.

Tabela 4.1: *Tempos para representação de diferentes entradas para temperatura e pressão sanguínea.*

	Entradas			
	30	300	3000	30000
Temperatura	1,91s	7,05s	38,94s	321,27s
Pressão Sanguínea	1,95s	7,67s	41,53s	338,45s

A Figura 4.2 apresenta a comparação gráfica entre as medições dos dois *Hermes Widgets* supracitados. Pode-se constatar, a partir do gráfico, que o crescimento do tempo de processamento para representação se manteve linear, apesar do aumento da quantidade de aferições e também de triplas representadas. Este teste prova a hipótese de que o serviço de representação é escalável quanto ao aumento de dados de contexto.

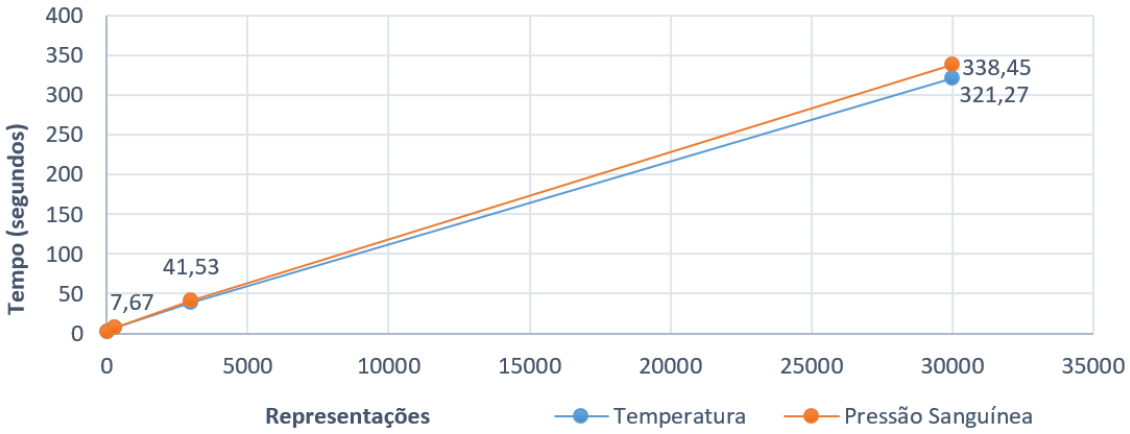


Figura 4.2: *Tempos de representação para os sinais de pressão sanguínea e temperatura.*

Ressalta-se que, para cada lote de dados de entrada (30, 300, 3000 e 30000) foi realizado um teste particular, no qual o componente foi instanciado recebendo uma quantidade x distinta de aferições para representação. Dessa forma, para cada sinal vital monitorado de um paciente foram realizados quatro testes diferentes.

Tendo sido realizados os testes com todos os *Hermes Widgets*, para cada um dos sinais vitais de cada paciente monitorado, a Tabela 4.2 apresenta o tempo médio gasto por um *Hermes Widget* para representar cada lote de entradas neste cenário. Estes dados podem ser analisados graficamente na Figura 4.3.

Tabela 4.2: *Tempos médios para representação de diferentes lotes de entradas.*

	Entradas			
	30	300	3000	30000
Tempo (s)	1,92	7,08	38,97	321,45

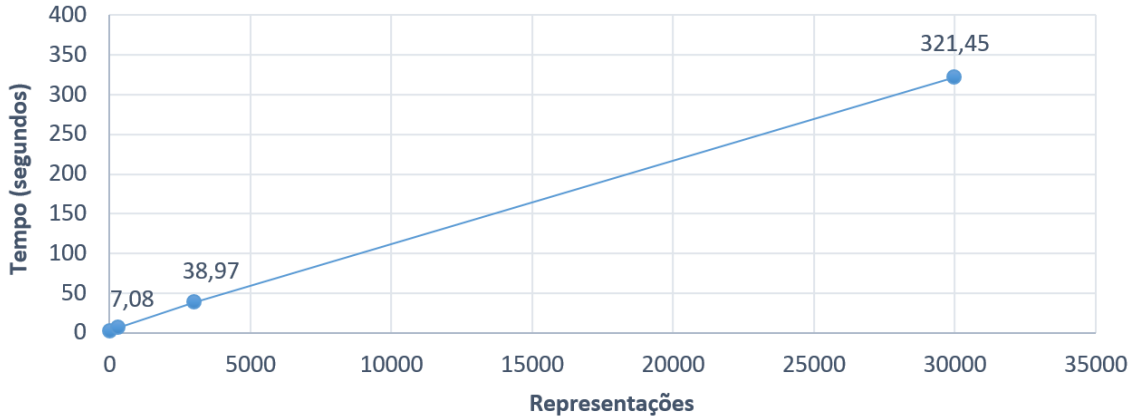


Figura 4.3: *Tempo gasto em segundos para representação de 30, 300, 3000 e 3000 medições.*

Através deste gráfico, comprova-se o crescimento linear de tempo de processamento em relação ao aumento exponencial da quantidade de entradas que devem ser processadas e representadas pelos *Hermes Widgets*. De acordo com estes resultados, foi calculada ainda a média de tempo gasto para a realização de cada representação, apresentada na Tabela 4.3 e no gráfico da Figura 4.4.

Tabela 4.3: *Tempos médios para cada representação.*

	Entradas			
	30	300	3000	30000
Tempo (ms)	64,07	23,59	12,98	10,71

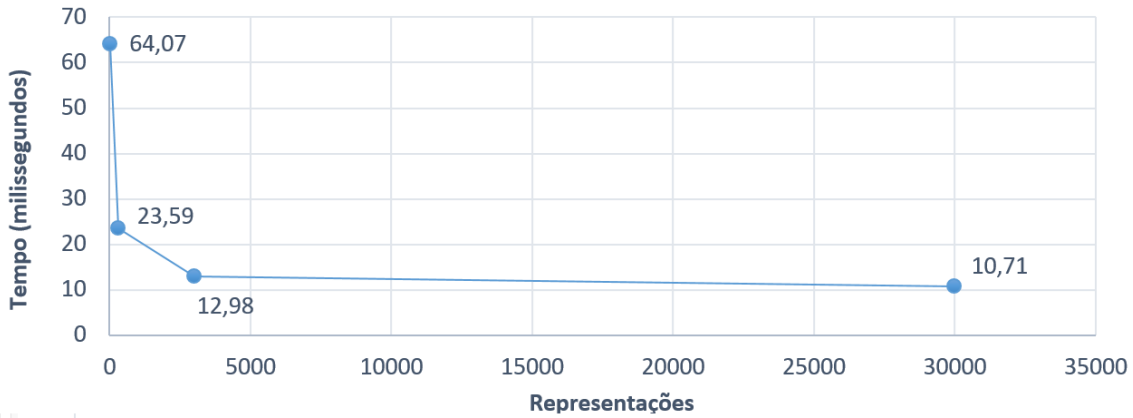


Figura 4.4: *Tempo médio em milissegundos por representação para 30, 300, 3000 e 30000 medições.*

Constatou-se que a média de tempo para representação dos menores lotes de dados é discretamente mais elevada que a média para os lotes maiores. Este comportamento se deve ao processamento necessário para realizar uma operação de aquivo para carregar o modelo ontológico no início da execução de um *Hermes Widget*, antes do serviço de representação. Por este motivo, as primeiras representações possuem um tempo médio de execução mais alto do que as representações posteriores.

Ainda neste gráfico, observa-se que, à medida em que aumenta a quantidade de entradas, o tempo de representação torna-se cada vez mais uniforme. Por exemplo, entre 3000 e 30000 entradas a diferença média de tempo de representação foi de 2,27 milissegundos, enquanto no intervalo anterior, entre 300 e 3000, esta diferença foi de 10,61 milissegundos. Este comportamento ocorre pois, após o carregamento inicial do modelo ontológico, todo o processamento é realizado com o modelo carregado em memória, melhorando o tempo de execução.

Analisando dados gerados pelo componente *Hermes Widget*, conclui-se que os tempos de execução atendem com folga a demanda da literatura para este cenário, que determina o limite de 19 segundos para a notificação de alarmes a partir de uma medição de sinal vital. Porém, uma vez que o componente *Hermes Widget* é responsável somente pela etapa de representação, para que esta afirmação possa ser válida é necessário que sejam somados a estes tempos os tempos da etapa de raciocínio, realizada pelo componente *Hermes Interpreter* [32]. A soma das duas etapas reflete o tempo necessário para a geração de alarmes a partir de inferência sobre os modelos de contexto representados.

De acordo com Maranhão [32], para raciocínio de filtros com inferência utilizando o componente *Hermes Interpreter*, os tempos médios foram respectivamente: 234 milissegundos para 30 entradas, 1495 milissegundos para 300 entradas e 15879 milissegundos para 3000 entradas. Maranhão [32] não apresenta testes com o *Hermes Interpreter* para 30000 entradas. A Tabela 4.4 apresenta a soma dos tempos médios de execução dos componentes *Hermes Widget* e *Hermes Interpreter* para as diferentes entradas.

Tabela 4.4: *Tempo total de representação e raciocínio.*

	Entradas			
	30	300	3000	30000
Hermes Widget	64,07ms	23,59ms	12,98ms	10,71
Hermes Interpreter	234ms	1495ms	15879ms	–
Total	**298,07ms**	**1518,59ms**	**15891,98ms**	–

A partir dos resultados obtidos pode-se afirmar que, para até 3000 entradas, o somatório dos tempos de execução dos componentes *Hermes Widget* e *Hermes Interpreter* está abaixo do máximo estipulado pela literatura [18]. Considerando-se somente o componente *Hermes Widget*, até 30000 entradas são processadas com tempo médio de execução válido para o monitoramento de sinais vitais. Os resultados apresentados nesta

análise comprovam a escalabilidade do componente *Hermes Widget*, validando as hipóteses levantadas.

4.3 Avaliação do componente *Hermes History*

A avaliação descrita nesta seção tem como objetivo analisar o comportamento do componente *Hermes History* em função do crescimento do número de entradas recebidas para persistência de contexto. Como *blackboard* de informações do CMS *Hermes*, todos os modelos de contexto produzidos pelos demais componentes passam em algum momento pelo serviço de histórico, onde são persistidos antes de sua disseminação.

As seguintes hipóteses foram levantadas para esta avaliação:

1. O componente *Hermes History* é escalável quanto ao serviço de persistência de contexto recebido dos demais componentes do CMS *Hermes*.
2. O tempo gasto por *Hermes History* para realizar seus serviços cresce de maneira linear conforme o aumento exponencial de entradas recebidas.
3. *Hermes History* atende os requisitos do domínio de monitoramento de sinais vitais humanos, destacados pela equipe de enfermagem do Hospital das Clínicas da UFG.

O método utilizado para verificação das hipóteses consistiu nos seguintes passos: *i*) criação de um cenário de aplicação; *ii* execução experimentos com os componentes de acordo com as configurações do cenário; *iii*) coleta de *logs* dos tempos de execução para cada entrada (modelo de contexto) recebida; e *iv*) análise dos dados coletados.

4.3.1 Configurações dos experimentos

Hermes History foi projetado e implementado para persistir e recuperar três tipos de entradas de contexto: *i*) representações, produzidas por *Hermes Widgets*; *ii*) agregações, produzidas por *Hermes Aggregators*; e *iii*) inferências, produzidas por *Hermes Interpreter*. Ao receber uma nova entrada de contexto, HH a identifica pelas informações do tópico e gera uma requisição para persistência da informação de contexto.

O cenário para os testes realizados estende as configurações utilizadas para avaliação do componente HW, apresentada na Seção 4.2, incluindo os componentes *Hermes Aggregator* e *Hermes History*, que é o elemento central nestes experimentos. De maneira mais específica, HH recebe entradas de todos os componentes em execução, conforme foi descrito na Subseção 3.3.2. Para os testes e avaliação do HH, além de uma instância deste componente, foram utilizadas duas instâncias de HW, uma instância de HA, e duas instâncias de aplicações sensíveis a contexto.

Cada HW representa o contexto aferido por um sensor específico e relacionado a uma propriedade, neste caso um sinal vital, sendo os sinais vitais monitorados a *pressão arterial* e a *temperatura corpórea*, os quais são ainda agregados pelo HA. O contexto produzido é publicado em tópicos privados de HH, que persiste e republica em tópicos públicos para os demais assinantes. Cada aplicação assina um tipo distinto de contexto: a primeira assina as representações produzidas por HW e a segunda, as agregações produzidas por HA.

Para executar os experimentos, os componentes foram executados nas seguintes máquinas:

- *Hermes History*: Windows 10 de 64 bits, com processador Intel(R) Core(TM) i7-3537U, de 2,00GHz cada e memória RAM de 8GB;
- *Hermes Aggregator*, *Hermes Widgets* e aplicações: Windows 7 de 64 bits, com processador Intel(R) Core(TM) 2 Duo, de 2,2GHz cada e memória RAM de 4GB.

Por ser o objeto central desta avaliação, o componente *Hermes History* foi executado isoladamente em uma máquina, à parte dos demais componentes e aplicações, no intuito de evitar a concorrência por memória e processamento que teriam influência nos resultados do experimento.

4.3.2 Análise dos resultados

As entradas de contexto recebidas e persistidas por HH nos testes realizados para este experimento possuem as seguintes configurações:

- Cada entrada de HW para temperatura corpórea, representada com SSN: 16 triplas de dados;
- Cada entrada de HW para pressão arterial, representada com SSN: 17 triplas;
- Cada entrada de HA para agregação de temperatura corpórea e pressão sanguínea, representada através de conversão para MSVH: 22 triplas de dados.

Para todos os exemplos de teste foram utilizados dois (2) HW's, para temperatura corpórea e pressão arterial, e um HA para agregação do contexto produzido para os dois sinais vitais mencionados. Estes HW's enviam representações de contexto para HH com a frequência de 1 publicação a cada segundo. A cada par de representações dos dois tipos de sinal vital mencionados, uma agregação também é publicada para persistência.

O gráfico da Figura 4.5 apresenta a quantidade de triplas persistidas por *Hermes History* em cada teste realizado.

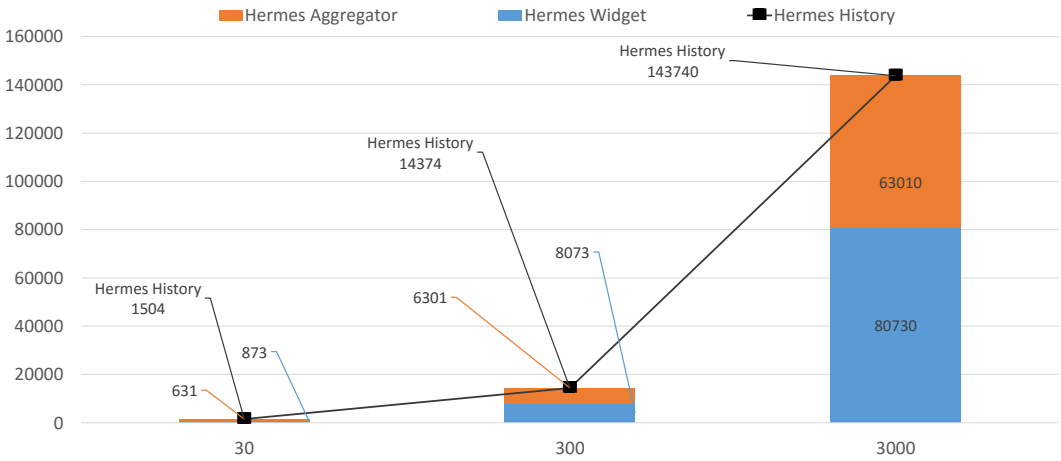


Figura 4.5: *Quantidade de triplas armazenadas para cada entrada de contexto.*

As Figuras 4.6, 4.7, 4.8 e 4.9 apresentam o tempo gasto por *Hermes History* para processar e persistir cada entrada de contexto recebida. Uma vez que *Hermes History* executa as entradas de contexto recebidas como *thread*, ressalta-se que o tempo total para processamento e persistência de todas as entradas não é igual ao somatório do tempo de execução das mesmas.

A Figura 4.6 apresenta o teste para dois HWs publicando 30 representações cada, e um HA que gera um total de 30 agregações. Dessa forma, o componente HH processa 90 *threads* relativas às entradas recebidas dos dois componentes. Neste teste o tempo total para o processamento das 90 *threads* foi de 38,338 segundos, com uma média de 412,1 milissegundos por *thread* e desvio padrão de 302,55 milissegundos.

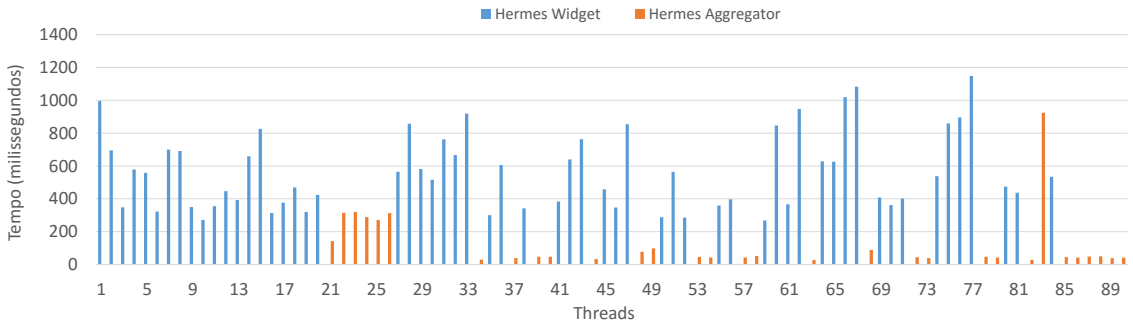


Figura 4.6: *Tempos de execução para 90 threads recebidas por Hermes History.*

A partir do gráfico pode-se constatar que, quando há um maior número de entradas de um determinado tipo, como é o caso dos modelos enviados por HWs (60), o tempo de processamento para persistência destes é mais elevado em relação aos modelos enviados pelo HA (30). Com um número maior de publicações, e consequentemente um menor intervalo entre o recebimento dos modelos, há mais concorrência por processa-

mento das entradas, o que contribui para que as *threads* de modelos recebidos de HWs sejam processadas com uma maior demanda de tempo.

A Figura 4.7 apresenta o teste que foi realizado para dois HWs publicando 300 representações cada, e um HA que realiza 300 agregações. Neste caso, o componente HH processa 900 *threads* no total, para o contexto recebido dos dois componentes. O tempo total para o processamento de todas as entradas foi de 5,6 minutos, com uma média de 433,35 milissegundos por *thread* e desvio padrão de 376,38 milissegundos.

Da mesma forma que ocorreu no teste com 90 *threads*, com o crescimento exponencial para 900 *threads* evidencia-se que o tempo gasto para o processamento de entradas vindas de HWs é relativamente maior que o tempo gasto para as entradas vindas de um HA. A média e o desvio padrão tiveram um crescimento de 21 e 74 milissegundos respectivamente, demonstrando baixo impacto causado pelo crescimento do número de entradas.

O teste apresentado na Figura 4.8 teve como entrada dois HWs publicando 3000 representações cada, e um HA que processou 3000 agregações. Neste cenário, HH processa um total de 9000 *threads* relativas às publicações recebidas. Para estas *threads*, o tempo total de processamento foi de 52,2063 minutos, com uma média de 436,2 milissegundos por *thread* e desvio padrão de 385,86.

Este experimento teve como resultado um comportamento similar aos testes anteriores, também apresentando um baixo crescimento da média de tempo de processamento e do desvio padrão, respectivamente 3 e 9 milissegundos. Os números obtidos demonstram o baixo impacto do crescimento exponencial de 900 para 9000 entradas, quanto ao tempo de processamento e persistência das informações de contexto.

Nos testes com um total de 90000 *threads* foi observada uma sobrecarga de processamento e memória quanto ao *endpoint* utilizado para acesso aos repositórios. Verificou-se que a operação de persistência do contexto nos repositórios é a parte do processamento realizado por HH que possui mais demanda de recursos, e consequentemente influencia no tempo de processamento das entradas.

Devido à quantidade de entradas, e do curto intervalo de tempo entre publicações (1 segundo, para cada HWs executado em paralelo), a concorrência por processamento elevou a média de tempo para persistência de cada entrada de contexto. Apesar do crescimento linear, como mostra a Figura 4.9, os tempos de execução se mostraram significativamente mais elevados em comparação aos testes anteriores.

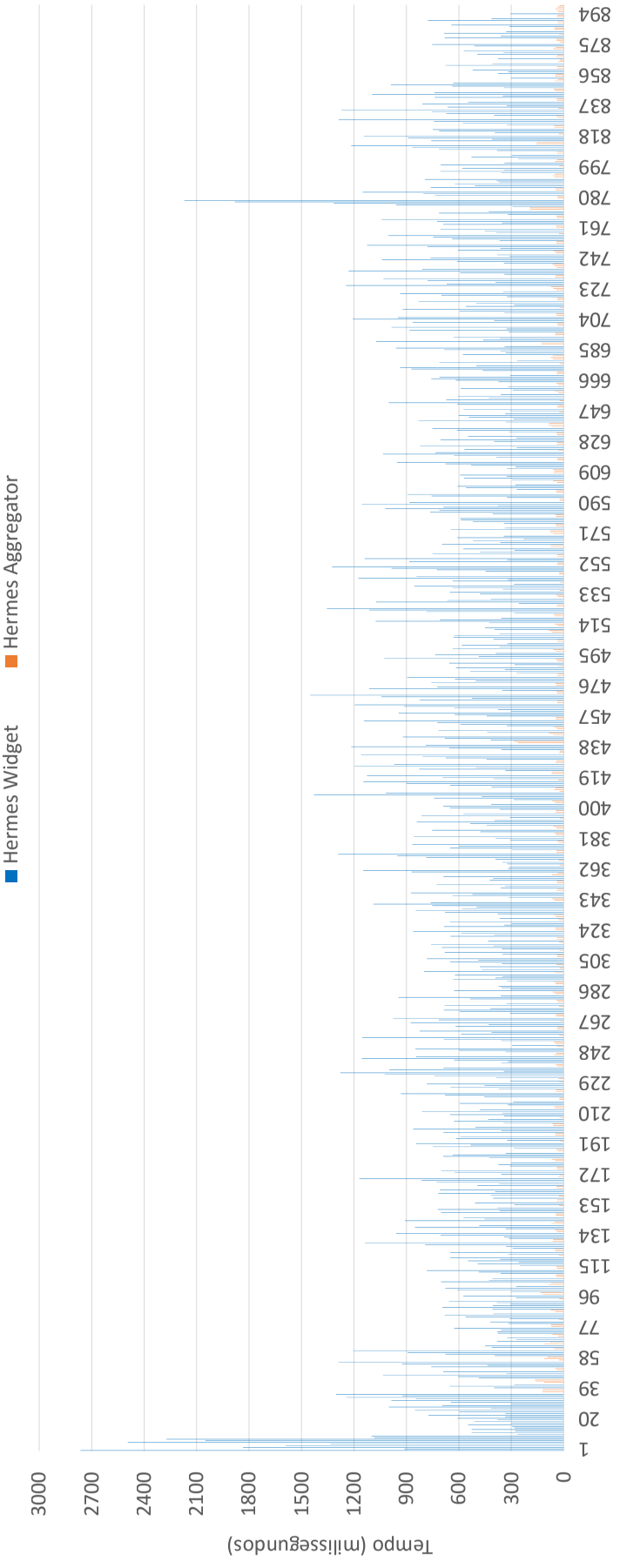


Figura 4.7: *Tempos de execução para 900 threads recebidas por Hermes History.*

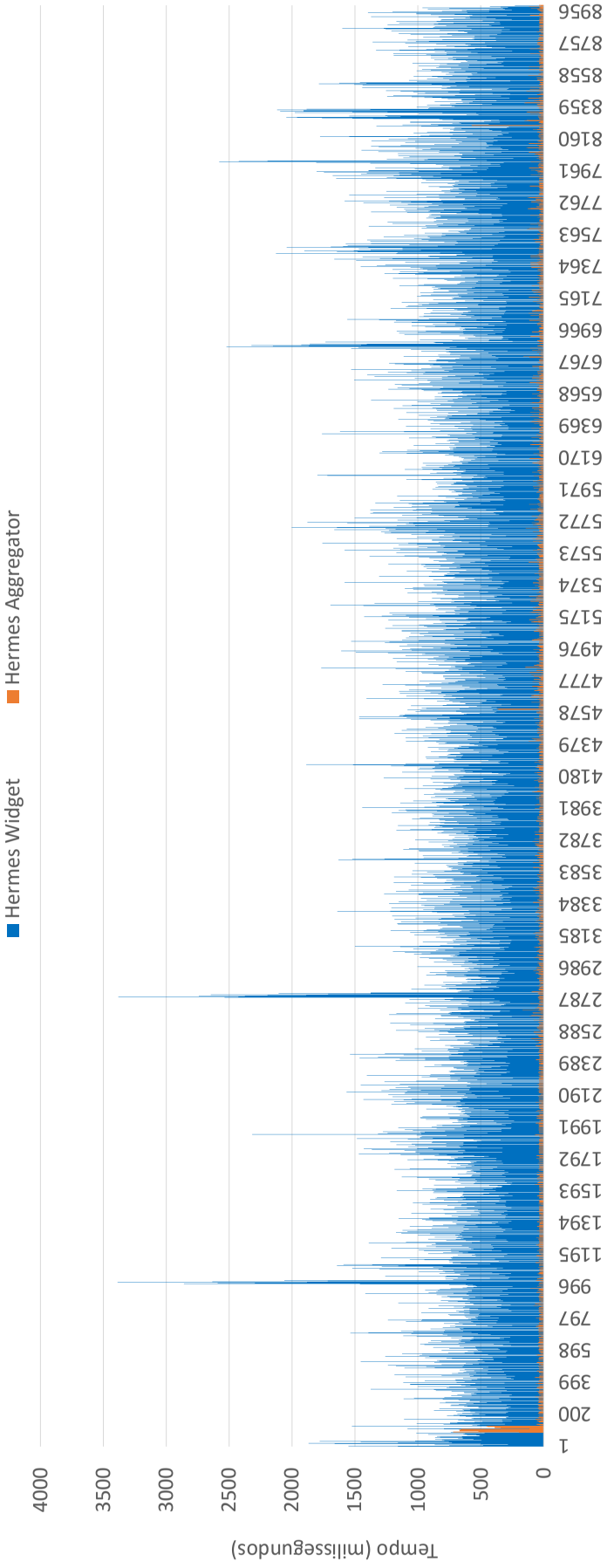


Figura 4.8: *Tempos de execução para 9000 threads recebidas por Hermes History.*

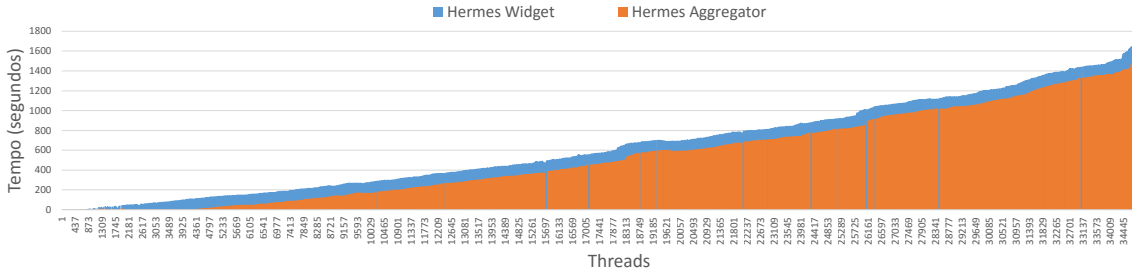


Figura 4.9: *Tempos de execução para teste com 90000 threads recebidas por Hermes History.*

Para o cenário de monitoramento de sinais vitais humanos, de acordo com a equipe de enfermagem que apoiou esta pesquisa, a idade [1] das aferições dos sinais vitais recebe as seguintes classificações, apresentadas na Tabela 4.5.

Tabela 4.5: *Relevância do contexto de sinais vitais humanos.*

Sinal Vital	**Unidade**	***Muito relevante***	***Relevante***	***Irrelevante***
Temperatura	minutos	x <= 5	x <= 60	x > 60
Pressão Arterial	minutos	x <= 5	x <= 60	x > 60
Freq. Respiratória	minutos	x <= 5	x <= 30	x > 30
Freq. Cardíaca	minutos	x <= 1	x <= 30	x > 30
Saturação de Oxigênio	minutos	x <= 1	x <= 30	x > 30

Enquanto a idade de uma aferição do sinal vital temperatura corpórea, por exemplo, considerada *muito relevante* pela equipe de enfermagem é de até 5 minutos (ou seja, uma entrada é recebida a cada 5 minutos, que representa sua frequência do monitoramento), os experimentos realizados e apresentados nesta seção adotaram a frequência de 1 segundo entre aferições, explorando o serviço de histórico em um nível de exigência exponencialmente superior ao do cenário real. Em um exemplo comparativo isso significa que, para cada uma (1) aferição persistida segundo os parâmetros do cenário real, cerca de 300 aferições são processadas e armazenadas com as configurações dos experimentos realizados.

Com as configurações do cenário real, apresentadas na Tabela 4.5, o monitoramento do sinal vital temperatura corpórea por 12 horas geraria um total de apenas 144 aferições, cumprindo a idade de referência para que todas as aferições sejam classificadas como *muito relevantes*. Este valor, somado com o número de aferições de outro sinal vital (ex.: a pressão sanguínea), e ainda com o total de agregações destes, geraria 360 entradas de contexto para HH. Este número de entradas se encaixaria nos parâmetros do segundo teste, apresentado na Figura 4.7, com um total inferior a 900 aferições. Porém, enquanto no cenário real seria necessário processar um total de 360 entradas no espaço de tempo de 12 horas, no experimento realizado 900 entradas foram processadas em 5,6 minutos.

[1] Como *idade* entende-se o tempo decorrido desde que uma aferição foi realizada.

Dessa forma, pode-se concluir que, mesmo adotando uma frequência de monitoramento muito superior à de um ambiente real, os tempos de processamento de HH são satisfatórios para entrega do contexto enquanto este ainda possui sua melhor avaliação de qualidade. Estes resultados permitem afirmar ainda que diversos outros domínios de aplicação, inclusive ainda mais críticos e com níveis de exigência maiores do que o monitoramento de sinais vitais, podem ser tratados com a utilização dos serviços apresentados.

4.4 Considerações Finais

Este capítulo apresentou as avaliações realizadas para os componentes *Hermes Widget* e *Hermes History*. Para HW, foi realizada uma avaliação de manutenabilidade, no intuito de verificar este requisito tanto no projeto quanto na implementação do componente; bem como uma avaliação de escalabilidade, para verificar o comportamento do componente quanto ao crescimento de dados de contexto enviados para representação.

Quanto ao HH, o foco foi também uma avaliação da escalabilidade, mostrando as capacidades do componente no gerenciamento das entradas de contexto (receber, persistir e publicar). Os resultados obtidos foram comparados aos dados fornecidos pela Equipe de Enfermagem do Hospital das Clínicas, comprovando a eficiência do HH no cenário de monitoramento de sinais vitais humanos.

Uma avaliação específica para o componente HA não foi realizada no escopo deste trabalho, uma vez que este componente pode ser considerado uma extensão do HW, e realiza um processamento similar a este. De maneira indireta, o HA foi analisado durante os experimentos para avaliação da escalabilidade do HH, possibilitando a constatação de que este realiza seus serviços de maneira eficiente, sem gerar qualquer tipo de gargalo ou sobrecarga quanto ao tempo total de processamento do contexto.

CAPÍTULO 5

Trabalhos Relacionados

Este capítulo apresenta os trabalhos relacionados de maneira geral e a sua relação específica com os serviços de representação, agregação e histórico propostos nessa dissertação. A análise realizada teve como base os principais requisitos elencados para *Hermes* e para os componentes propostos, apresentados respectivamente da Seção 2.3 e no Capítulo 3. Para comparação foram considerados softwares tais como *middlewares*, infraestruturas, CMS no geral, que apoiam o gerenciamento do contexto e o desenvolvimento de aplicações.

5.1 Descrição dos Trabalhos

Esta seção apresenta brevemente os trabalhos relacionados a *Hermes Widget*, *Hermes Aggregator* e *Hermes History*.

CroCo

CroCo, acrônimo para *Cross-application Context management Service* [35] é um CMS baseado em ontologias projetado para oferecer serviços de gerenciamento de contexto independente de aplicação (*cross-application*). Como base para o serviço de modelagem de contexto é utilizada uma ontologia de alto nível, denominada CroCoOn (*Cross-application Context Ontology*).

Por questões de eficiência e reusabilidade, Mitschick et al. [35] destacam a necessidade de separar das aplicações sensíveis a contexto os problemas relacionados à aquisição e modelagem de contexto, mas se preocupando em oferecer informações de contexto para aplicações de diferentes domínios. Para realizar este propósito e construir os serviços necessários é adotada a modelagem de contexto baseada em ontologias.

Em relação a representação ontológica, CroCo assume que seus provedores já enviem o contexto na forma de triplas RDF, que são recebidos pelo serviço *Context Update Service*. Este serviço adiciona informações como identificador do provedor e instante do recebimento, às triplas recebidas. Posteriormente, o serviço *Consistence*

Manager verifica a consistência dos modelos recebidos de acordo com a ontologia CROCOON.

Em *Hermes*, o componente *Hermes Widget* oferece o serviço de representação ontológica, onde as informações de baixo nível recebidas de provedores são representadas em RDF de acordo com o domínio e as ontologias que o modelam. Juntamente com a representação do domínio é realizada a representação do sensoriamento, que adiciona as informações do provedor e do tipo de de contexto ao modelo recebido.

CROCO destaca ainda a importância do histórico de contexto para que um serviço mais sofisticado de raciocínio de contexto possa ser realizado. O CMS conta com um componente para gerenciamento de histórico, mas os autores destacam limitações e desafios como: a realização de um registro mais sofisticado ou a integração de mecanismos para previsão de futuras alterações contexto.

O serviço de histórico de CROCO armazena os dados de contexto, processados e representados em RDF, em um banco de dados MySQL. De acordo com a literatura e a própria API *Jena*, utilizada na implementação de CROCO, bancos de dados relacionais não são meio mais eficiente e escalável para o armazenamento de triplas. Neste intuito, devem ser utilizados repositórios otimizados para o armazenamento e consulta de triplas.

Hermes History também utiliza a API Jena, porém, ao invés da sua versão SDB, para armazenamento em bancos de dados relacionais, foi utilizada a API TDB para armazenamento nativo de triplas. Essa solução otimiza as questões de armazenamento, criando um conjunto de arquivos adequado para informações em triplas, ao invés de um mapeamento adaptado em tabelas relacionais, e também de consulta, oferecendo desempenho significamente mais elevado que a solução relacional.

infinitum

Com base nos requisitos de gerenciamento de contexto distribuído e escalável, disseminação de contexto independente de domínio e aplicação de políticas de segurança, *infinitum* [27] é um CMS que utiliza os benefícios da modelagem baseada em ontologias e inferência semântica. Entre as principais preocupações de *infinitum* está a grande quantidade de informação contextual produzida por provedores de contexto.

Em *infinitum* o gerenciamento de contexto em cada domínio é realizado por um *Context Server* (CS) específico para este domínio, conceito que se assemelha ao dos *Hermes Widgets* do CMS *Hermes*. Dessa forma, *infinitum* gerencia o contexto adquirido de diferentes CSs (e, portanto, diferentes domínios), além de fornecer um serviço de consultas e serviço de inferência, ambos independentes de domínio.

A fim de armazenar as informações de contexto relacionadas a um domínio, é proposto um repositório baseado em espaço de tuplas, chamado *Context Wave* (CW). Um

Context Wave é um documento XML hospedado por um *Context Server*. Cada CW possui um único ID global e consiste em um consunto de informações de contexto, chamadas *contextlets*.

A arquitetura de *infinitum* é distribuída e consiste de um conjunto de nós *Context Servers* interligados via Internet, utilizando o protocolo *Google Wave Federation Protocol* para a troca de informações de contexto. Cada CS é composto por um *Context Manager*, um *Wave Server* e um *Web Server*.

Uma limitação de *infinitum* é o fato de que a ligação entre o componente de gerenciamento e as ontologias, repositório e serviço de inferência são diretamente realizadas pela API Jena, sem um componente que realize esta abstração e o desacoplamento, visando as questões de manutenabilidade da infraestrutura. Outra diferença, em relação à proposta deste trabalho, é que *infinitum* não se preocupa com histórico de contexto ou tratamento dos diferentes níveis de expressividade das informações que gerencia.

CASUP

Hong et al. [23] destacam a importância da utilização do contexto para predizer as preferências de usuários no intuito de prover serviços personalizados. Nesta lacuna de pesquisa na área de computação sensível a contexto, é proposto um *framework* para o provimento de serviços personalizados baseados no histórico de contexto.

CASUP (*Context-aware system considering user preference*) é um CMS que possui quatro camadas principais:

1. *Camada de aquisição de contexto*: que coleta os dados de sensores (contexto não processado), dados de usuários (perfis) e dados de serviços;
2. *Camada de gerenciamento de contexto*: infere contexto de alto nível a partir de contexto de baixo nível, armazena as informações coletadas no componente de histórico de contexto e classifica os perfis de usuários e os serviços selecionados de acordo com o contexto de alto nível;
3. *Camada de gerenciamento de preferências*: raciocina as preferências de usuários a partir do histórico de contexto e gerencia a associação de regras para recomendar os próximos serviços;
4. *Camada de aplicações*: fornece serviços personalizados de acordo com as regras e associações extraídas no gerenciamento de preferências.

Em cada camada supracitada, agentes realizam tarefas específicas relacionadas ao seu escopo. Na camada de gerenciamento de contexto, o agente *context aggregator* funciona como um ponto de controle que encontra *context wrappers* (provedores de contexto) e coleta o contexto produzido por eles. De maneira similar, o componente

Hermes Aggregator realiza de forma extensível, a fusão de *Hermes Widgets* para gerar modelos de contexto de alto nível.

Para Hong et al. [23], o histórico de contexto é composto por: perfis de usuários, o contexto atual destes usuários e os serviços utilizados por eles. O histórico de contexto é modelado através de ontologias OWL e é utilizado para raciocinar regras de preferência e recomendar serviços personalizados para usuários.

É destacado ainda o desafio do gerenciamento de grandes quantidades de informações de contexto. Para reduzir essa quantidade de informações de contexto e a carga de processamento necessária, é utilizada uma abordagem hierárquica que consiste de: uma ontologia comum, que gerencia informações gerais como conceitos básicos comumente utilizados em diferentes ambientes; e uma ontologia específica do domínio.

UCAM

Complementando os requisitos para CMS elencados na Seção 2.3, Oh et al. [37] destacam ainda em sua proposta: *i*) o relacionamento transparente entre fontes de contexto heterogêneas (sensores, serviços, entre outras); *ii*) a sincronização dos dados contextuais utilizados em diferentes sistemas sensíveis a contexto; e *iii*) arquitetura com inteligência para gerenciar múltiplos comportamentos de usuários.

Segundo os autores, um CMS deve possuir um método de representação que apoie completamente a informação contextual provida por sensores e serviços. Fusão e raciocínio de contexto realizam a integração de contexto de diferentes domínios e melhoram a capacidade de inteligência através da extração de contexto de alto nível.

A arquitetura proposta, denominada UCAM (*Unified Context-Aware Application Model*) [37], utiliza como base para representação o modelo de contexto 5W1H, que consiste de: elemento de contexto (formato estruturado), contexto (dados) e memória de contexto (repositório). Este modelo é constituído por um conjunto de ontologias que contém as definições explícitas para raciocínio de contexto dos usuários.

UCAM classifica o contexto em dois níveis de abstração:

- *Contexto de baixo nível*: ou contexto *explícito*, é o contexto primariamente representado a partir dos sensores;
- *Contexto de alto nível*: também chamado contexto *implícito*, é o aquele extraído do contexto de baixo nível através dos processos de fusão e raciocínio de contexto.

O UCAM apoia a fusão de dados de diferentes fontes, integrando informações tais como: identidade do usuário, localização, atividades, comportamento e padrões. O raciocinador é utilizado para predição de novas informações de contexto através da reutilização de informações de contexto anteriores (histórico de contexto) e criar regras reconfiguráveis de acordo com o ambiente.

LoCCAM

O LoCCAM (*Loosely Coupled Context Acquisition Middleware*) é uma infraestrutura de gerenciamento de contexto que apoia o desenvolvimento de aplicações sensíveis ao contexto em dispositivos móveis (e.g., aplicações de anotações contextuais em imagens, aplicações de recomendação de conteúdo baseado na localização do usuário) [30].

O *middleware* realiza o intermédio para aquisição das informações contextuais e provê desacoplamento entre o código das aplicações e sensores de aquisição de contexto. O desacoplamento ocorre devido ao uso de CACs (Componentes de Aquisição de Contexto), que são os componentes responsáveis pela captura das informações contextuais.

Cada CAC encapsula um sensor, que pode ser tanto um sensor físico (ex.: GPS), lógico (ex.: perfil do usuário) ou virtual (ex.: um serviço de meteorologia). O LoCCAM gerencia todos os CACs em execução, possibilitando sua instalação, execução, parada e remoção. Como uma forma de possibilitar a comunicação entre o *middleware* e as aplicações, foi desenvolvido o conceito de *Context Keys*. É por meio do uso das *Context Keys* que as aplicações identificam as informações contextuais que desejam sem precisar especificar ou instanciar o código que provê a informação.

A modelagem de contexto em LoCCAM é baseada em um vocabulário de referência, organizado na forma de uma árvore de conceitos. O mecanismo para representar a informação de contexto é baseado na abordagem atributo-valor, por exemplo, o atributo `context.device.location.relativelocation.inside` pode ter o valor "`home`".

De modo similar aos CAC's de LoCCAM, o componente *Hermes Widget* separa o tratamento do contexto adquirido de sensores da sua utilização por demais componentes e/ou aplicações. Porém, *Hermes Widget* realiza a representação ontológica dessas informações, padronizando-as de acordo com a ontologia SSN, enquanto LoCCAM utiliza um vocabulário próprio para realização desse serviço.

EXEHDA-UC

Considerando a alta distribuição, heterogeneidade, dinamicidade e mobilidade de ambientes ubíquos, o trabalho de Lopes et al. [29] apresenta o modelo arquitetural para consciência de contexto denominado EXEHDA-UC (*Execution Environment for Highly Distributed Applications - Ubiquitous Context awareness*). Na forma de *middleware*, EXEHDA oferece a aplicações apoio à aquisição, persistência e processamento da informação contextual.

A implementação de EXEHDA é baseada em duas partes principais: *Border Server* e *Context Server*, respectivamente responsáveis pela interação com o ambiente através dos sensores e processamento da informação contextual. O *Translate Component*,

um dos componentes internos de *Border Server*, é responsável pela *adequação do dado coletado à natureza da aplicação*.

Enquanto *Hermes Widget* realiza a representação das informações coletadas, com base em um modelo ontológico, o *Translate* realiza uma conversão, ou adequação dos dados de acordo com a aplicação, produzindo um dado simples ou um resultado através da aplicação de regras. Por exemplo, um conjunto de valores de temperatura pode gerar o contexto "temperatura alta" ou "temperatura baixa" através de uma regra predefinida.

A falta de um modelo formal para representação do contexto em EXEHDA dificulta a sua interoperabilidade e utilização por diferentes sistemas e/ou aplicações. *Hermes Widget* realiza a representação ontológica formal do contexto, gerando uma informação independente de aplicações e com significado próprio.

EXEHDA mantém um repositório de contexto, que persiste as informações contextuais obtidas por um *Collector*, utilizando um modelo relacional. Não são apresentados detalhes da base de dados utilizada, nem de como esta pode ser consultada. Por outro lado *Hermes History* oferece um serviço de histórico independente porém integrado de maneira transparente aos demais componentes, que persiste o contexto em repositórios de triplas e permite sua consulta através de uma interface para essa finalidade.

CaMP

O trabalho de Paspallis e Papadopoulos [38] apresenta uma arquitetura de *middleware* baseada em componentes cujo objetivo é facilitar o desenvolvimento e implantação de aplicações sensíveis a contexto através de componentes reutilizáveis. São citadas como principais contribuições desse trabalho a combinação de uma metodologia de desenvolvimento juntamente com a arquitetura de *middleware* que trazem juntas um valor significativo para os desenvolvedores de aplicações.

O projeto e implementação de aplicações sensíveis a contexto é facilitado pelos componentes reutilizáveis, chamados de *context plug-ins*. O conceito de *plugabilidade* proposto implica que os provedores de contexto sejam concebidos e desenvolvidos como componentes independentes e dinamicamente conectáveis, facilitando a sua implantação. De maneira similar, os componentes de *Hermes* também visam facilitar a integração entre si e com as aplicações, oferecendo uma interface simples e padronizada.

Entre os principais serviços oferecidos estão o `IContextModel`, que permite a representação do contexto baseada em ontologias ou implementada via código, para situações mais simples; e os serviços `IContextRepository` e `IContextQuery`, que possibilitam respectivamente a persistência do contexto para histórico e o processamento de consultas ao contexto na linguagem CQL (*Context Query Language*). O serviço de consulta é definido como opcional para a infraestrutura, sendo os demais obrigatórios.

No processo de persistência, inicialmente o contexto é disponibilizado e acessado na forma de instâncias de objetos, que precisam ser serializadas para persistência. A implementação do repositório de contexto necessita que um sistema SGBD básico seja utilizado, para facilitar a customização do *middleware* para a plataforma. Sendo instanciado para Android, o *middleware* utiliza um banco de dados SQLite nativo.

Em *Hermes*, o contexto após representado por *Hermes Widget* já é disponibilizado para histórico e distribuição na forma de um modelo RDF devidamente serializado. Por utilizar um modelo de repositório orientado a triplas, nenhum outro tipo de processamento precisa ser realizado sobre a informação contextual, que pode ser diretamente persistida por *Hermes History*, sem gargalos quanto a serialização ou conversão da informação para o modelo de banco de dados.

Como estudo de caso, o trabalho de Paspallis e Papadopoulos apresenta o desenvolvimento de uma aplicação denominada CaMP (*Context-aware Media Player*), que estende as funcionalidades de um reprodutor de mídia padrão, que se torna ciente, por exemplo, de quando o usuário está digitando ou ausente, de modo a interromper e retomar suas atividades de maneira automática de acordo com as atividades do usuário.

5.2 Resumo Comparativo

A Tabela 5.1 relaciona os *princípios de projeto* (P*n*) que devem estar presentes em soluções sensíveis a contexto e outras *características* (C*n*) consideradas relevantes para a pesquisa nesta área.

- **P1** – Arquitetura em camadas
- **P2** – Componentes independentes
- **P3** – Representação baseada em ontologias
- **P4** – Agregação de contexto
- **P5** – Histórico de contexto
- **P6** – Histórico com tratamento de níveis de expressividade do contexto
- **C1** – Avaliação de manutenabilidade
- **C2** – Avaliação de escalabilidade
- **C3** – Microarquitetura dos componentes

Estes parâmetros para comparação são baseados na literatura que destaca os princípios de projeto que devem guiar o desenvolvimento de soluções para computação sensível a contexto [7, 39]. Também foram acrescentados requisitos elencados nesta proposta para suprir as necessidades do desenvolvimento de uma abordagem para fase de modelagem do ciclo de vida do contexto e dos serviços que a compõem.

Se um referido princípio não for explicitado e atendido no trabalho ou não estiver entre seus objetivos, é utilizado o marcador '–' na respectiva célula da tabela. Se um princípio ou característica foi parcialmente atendido, ele será identificado como '+' e, se completamente atendido, será utilizado o marcador '++'. A Tabela 5.1 apresenta uma análise comparativa dos trabalhos relacionados em relação aos princípios e características que deverão ser cumpridos pelos componentes de *Hermes* no escopo desta pesquisa.

Tabela 5.1: *Análise comparativa de Hermes em relação aos trabalhos relacionados.*

P/C / **CMS**	P1	P2	P3	P4	P5	P6	C1	C2	C3
CROCO	++	+	++	–	++	–	–	–	–
infinitum	++	–	++	–	++	–	–	+	–
CASUP	++	+	++	++	++	–	–	–	–
UCAM	–	–	++	–	++	–	–	–	–
LoCCAM	++	+	–	++	++	–	–	–	–
EXEHDA-UC	++	++	–	+	++	–	–	–	–
CaMP	++	++	+	–	++	–	–	–	–
Hermes	++	++	++	++	++	++	++	++	++

Esta análise comparativa com os demais trabalhos foi realizada em relação a *Hermes*, porém, destacamos que somente foram relacionados e comparados os itens que estão dentro do escopo deste trabalho. Princípios e características de componentes pré-existentes de *Hermes* não foram considerados.

5.3 Considerações Finais

Finalizando este capítulo é constatada a ligação entre os princípios de projeto de CMS e a modelagem ontológica de contexto, bem como a importância dos diferentes níveis de expressividade gerados a partir do processamento do contexto. A proposta deste trabalho, que envolve os serviços para representação, agregação e histórico de contexto, segue os princípios destacados pela literatura, no intuito de fornecer os serviços necessários para o desenvolvimento de aplicações sensíveis a contexto.

CAPÍTULO 6

Conclusões

Com a crescente implantação de sensores e a consequente produção de dados heterogêneos, a literatura tem demandado por soluções eficientes que consigam manipular esses dados e atender as necessidades das aplicações sensíveis a contexto e seus usuários [7, 39]. Nesse sentido, são necessários Sistemas Gerenciadores de Contexto que ofereçam serviços para obtenção e utilização do contexto, de maneira transparente e reutilizável, tratando cada etapa do ciclo de vida do contexto.

Diante desse cenário, este trabalho apresentou componentes que atuam sobre o contexto em relação à etapa de modelagem: *Hermes Widget*, que oferece o serviço de representação ontológica; *Hermes Aggregator*, que realiza o serviço de agregação de diferentes informações de contexto; e *Hermes History*, que funciona como um *blackboard* do CMS *Hermes*, oferecendo o gerenciamento do acesso e a persistência ao contexto processado pelos demais componentes.

6.1 Contribuições

Como principais contribuições deste trabalho destacam-se:

1. **Um serviço de representação ontológica de contexto independente de domínio e baseado em padrões**: o componente *Hermes Widget* permite que informações coletadas de sensores, com baixo nível de expressividade, sejam representadas com base em uma ontologia voltada para o sensoriamento, gerando uma informação que expressa a semântica do contexto que a envolve, descrevendo características como: tipo da propriedade aferida, unidade de medida, sensor responsável pela aferição, entidade de interesse (a qual a informação de contexto pertence), entre outras. A representação de contexto gerada por este serviço é baseada no padrão de projeto Estímulo-Sensor-Observação, núcleo da ontologia SSN, e uma proposta do W3C para padronização e expressividade de informações em sistemas de sensoriamento.
2. **Um serviço de agregação de contexto e conversão para domínio de aplicação**: o componente *Hermes Aggregator* permite que diferentes informações de contexto

que descrevem uma mesma entidade sejam reunidas em uma única representação também baseada em ontologia, facilitando, dessa forma, a utilização do contexto por aplicações e demais componentes, que recebem uma informação de contexto com maior nível de expressividade do que a gerada pelos *Hermes Widgets*. Este serviço permite ainda a conversão do contexto representado com base na SSN, para o domínio de aplicação específico onde estas informações serão utilizadas.

3. **Um serviço de histórico baseado no nível de expressividade do contexto com suporte a consultas semânticas e gerenciamento de acesso às informações**: o componente *Hermes History* realiza a persistência de todas as informações de contexto processadas pelos componentes HW, HA e HI, criando uma base de dados para cada nível de expressividade atingido dentro de *Hermes*: representação ontológica independente de domínio, agregação com conversão para o domínio de aplicação e inferências obtidas através da realização de raciocínio sobre as informações de contexto. Este componente ainda garante a consistência do contexto consumido pelos componentes e aplicações, no sentido de persistir todas as informações produzidas antes de disponibilizá-las para serem consumidas.

A representação de contexto baseada em ontologia, como descrito no início da proposta deste trabalho, contribui para questões de suma importância no âmbito de desenvolvimento de sistemas e aplicações sensíveis a contexto, como: modelo de contexto independente, interoperabilidade e expressividade. *Hermes Widget* se preocupa ainda com a independência de domínio do contexto, para que o serviço oferecido possa ser reutilizado em diferentes domínios de aplicação, representando as principais informações relacionadas a uma aferição de contexto.

Hermes Aggregator, além de minimizar o número de interações necessárias entre as aplicações e *Hermes* para obtenção de contexto, entrega uma informação com maior nível de expressividade, uma vez que agrega diferentes dados relacionados a uma mesma entidade de interesse e representa estas informações de acordo com o domínio de aplicação. Este serviço contribui significativamente para a redução da complexidade no desenvolvimento de aplicações sensíveis a contexto.

Em uma infraestrutura de contexto, o histórico das informações processadas possui grande importância para diferentes tipos de aplicações, por exemplo, aquelas que necessitam interpretar as alterações em relação a uma entidade ao longo do tempo. Toda informação de contexto em *Hermes* é persistida para consultas posteriores por meio do componente *Hermes History*, que atua como *blackboard* de contexto do CMS e garante que toda informação distribuída seja antes armazenada para manter a consistência das informações recebidas pelos demais componentes e aplicações.

Todos os componentes propostos neste trabalho ainda se preocuparam com as perspectivas da Engenharia de Software, no intuito de oferecer uma arquitetura adequada

a sistemas sensíveis a contexto, baseada nos requisitos encontrados na literatura [7, 39]. Dessa forma, os componentes projetados e implementados oferecem facilidade tantos aos desenvolvedores de aplicações sensíveis ao contexto, quanto a desenvolvedores avançados que precisem estender e/ou manter os componentes já desenvolvidos.

6.2 Limitações e Trabalhos Futuros

São identificadas como limitações dos componentes e serviços propostos nesta dissertação:

- A verbosidade das amostras de contexto ontológico (representado por HW e agregado por HA) ainda é muito alta se comparada a outras técnicas de representação, o que pode influenciar em serviços como de comunicação, aumentando o tráfego de dados em rede.
- Complementando a primeira limitação, o consumo de banda não foi avaliado nos cenários simulados. No entanto, isso não afetou nenhum dos experimentos, uma vez que não se propunham a avaliar características como tráfego de rede.
- A comunicação entre os componentes de *Hermes* e aplicações estão atualmente restritas a comunicação em rede local. Esse fato se deve à solução de projeto adotada para implementação do serviço de comunicação que, se necessário, pode ser substituída para atender este requisito.
- O único domínio de aplicação explorado foi o monitoramento de sinais vitais humanos. Dessa forma não foi possível realizar testes com aplicações interdomínio.
- A versão atual dos serviços ainda não está compatível para instalação e utilização em dispositivos móveis.
- Os serviços não foram integrados a sensores físicos para testes, os quais foram realizados apenas por meio de simulações.

Como trabalhos futuros desta pesquisa destacam-se:

1. Explorar histórico do contexto para oferecer novos serviços e informações relevantes, tais como tendências, inferências e possíveis previsões de contexto.
2. Investigar o tratamento adequado das questões de segurança e privacidade em sistemas de gerenciamento de contexto, tanto em nível de distribuição de contexto quanto de acesso ao contexto persistido nas bases de dados.
3. Otimizar o sistema de repositórios utilizado por *Hermes History*, tratando questões como compactação dos dados e disponibilização de serviços de raciocínio sobre as informações, no intuito de gerar novo conhecimento a partir de contexto já adquirido anteriormente.

4. Projetar e oferecer um serviço de *cache* para manter informações temporariamente em memória, otimizando o processamento consultas e minimizando o acesso a disco e questões de concorrência.
5. Investigar técnicas de fusão de dados mais sofisticadas que possam estender/evoluir o processo de agregação proposto, explorando a representação ontológica do contexto.
6. Evoluir o processo de distribuição dos serviços, com ênfase para o histórico de contexto em cenários distribuídos e heterogêneos.
7. Utilizar os serviços propostos na a criação de um *framework* para geração de aplicações (linha de produto de software), facilitando e estimulando a utilização de *Hermes* para o desenvolvimento de aplicações sensíveis a contexto.
8. Disponibilizar novos *Hermes Widgets*, sejam de âmbito geral, tais como para informações espaciais e temporais, ou para domínios de aplicação específicos e os respectivos sensores envolvidos.
9. Prover o tratamento de contexto de diferentes domínios de aplicação (serviços e aplicações interdomínio).

6.3 Considerações Finais

As soluções apresentadas nesta dissertação atenderam aos objetivos, em resumo, a construção de serviços para representação, agregação e histórico de contexto em apoio ao desenvolvimento de aplicações sensíveis ao contexto. Este capítulo, que encerra o trabalho realizado, apresentou as contribuições, limitações e trabalhos futuros, vislumbrando a possibilidade de melhorias, evolução e continuação desta pesquisa.

Além dos artigos já publicados, que refletem contribuições específicas deste trabalho, principalmente voltadas para o serviço de representação ontológica de contexto, estão previstos como metas desta pesquisa a escrita e submissão de artigos sobre as demais contribuições ainda não exploradas e as contribuições gerais, que deverão ser submetidos para os seguintes periódicos/congressos:

- *Journal of Computer Science* (JCS): B3
- *IEEE International Conference on Pervasive Computing and Communications* (Percom): A1
- *ACM Symposium on Applied Computing* (ACM SAC): A1

Referências Bibliográficas

[1] 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.

[2] BALDAUF, M.; DUSTDAR, S.; ROSENBERG, F. **A survey on context-aware systems**. *Int. Journal of Ad Hoc Ubiquitous Computing*, 2(4):263–277, June 2007.

[3] 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.

[4] BELLAVISTA, P.; CORRADI, A.; FANELLI, M.; FOSCHINI, L. **A survey of context data distribution for mobile ubiquitous systems**. *ACM Comput. Surv.*, 44(4):24:1–24:45, Sept. 2012.

[5] BERNARDOS, A.; TARRIO, P.; CASAR, J. **A data fusion framework for context-aware mobile services**. In: *Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on*, p. 606–613, Aug 2008.

[6] BERNERS-LEE, T.; HENDLER, J.; LASSILA, O.; OTHERS. **The semantic web**. *Scientific american*, 284(5):28–37, 2001.

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

[8] BULCAO-NETO, R. F.; PIMENTEL, M. G. C. **Toward a domain-independent semantic model for context-aware computing**. In: *Web Congress, 2005. LA-WEB 2005. Third Latin American*, p. 10 pp.–, Oct 2005.

[9] CHEN, H.; FININ, T.; JOSHI, A.; KAGAL, L.; PERICH, F.; CHAKRABORTY, D. **Intelligent agents meet the semantic web in smart spaces**. *Internet Computing, IEEE*, 8(6):69–79, Nov 2004.

[10] COMPTON, M.; BARNAGHI, P.; BERMUDEZ, L.; GARCÃA-CASTRO, R.; CORCHO, O.; COX, S.; GRAYBEAL, J.; HAUSWIRTH, M.; HENSON, C.; HERZOG, A.; HUANG, V.; JANOWICZ, K.; KELSEY, W. D.; PHUOC, D. L.; LEFORT, L.; LEGGIERI, M.; NEUHAUS, H.; NIKOLOV, A.; PAGE, K.; PASSANT, A.; SHETH, A.; TAYLOR, K. **The SSN ontology of the W3C semantic sensor network incubator group**. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17:25 – 32, 2012.

[11] DEY, A. K. **Providing architectural support for building context-aware applications**. PhD thesis, Georgia Institute of Technology, 2000.

[12] 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.

[13] DOURISH, P. **What we talk about when we talk about context**. *Personal Ubiquitous Comput.*, 8(1):19–30, Feb. 2004.

[14] DUCHARME, B. **Learning Sparql**. O'Reilly Media, Inc., 2013.

[15] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design patterns: elements of reusable object-oriented software**. Pearson Education, 1994.

[16] GOLDBERGER, A. L.; AMARAL, L. A. N.; GLASS, L.; HAUSDORFF, J. M.; IVANOV, P. C.; MARK, R. G.; MIETUS, J. E.; MOODY, G. B.; PENG, C.-K.; STANLEY, H. E. **PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals**. *Circulation*, 101(23):e215–e220, 2000 (June 13).

[17] GOLDFAIN, A.; SMITH, B.; ARABANDI, S.; BROCHHAUSEN, M.; HOGAN, W. R. **Vital sign ontology**. *Workshop on Bio-Ontologies, ISMB, Vienna*, p. 71–74, 2011.

[18] GÖRGES, M.; MARKEWITZ, B. A.; WESTENSKOW, D. R. **Improving alarm performance in the medical intensive care unit using delays and clinical context**. *Anesthesia & Analgesia*, 108(5):1546–1552, 2009.

[19] GU, T.; PUNG, H. K.; ZHANG, D. Q. **A service-oriented middleware for building context-aware services**. *J. Netw. Comput. Appl.*, 28(1):1–18, Jan. 2005.

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

[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] HONG, J.-Y.; SUH, E.-H.; KIM, S.-J. **Context-aware systems: A literature review and classification**. *Expert Systems with Applications*, 36(4):8509–8522, May 2009.

[23] HONG, J.; SUH, E.-H.; KIM, J.; KIM, S. **Context-aware system for proactive personalized service based on context history**. *Expert Systems with Applications*, 36(4):7448–7457, May 2009.

[24] INDULSKA, J.; SUTTON, P. **Location management in pervasive systems**. In: *Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003 - Volume 21*, ACSW Frontiers '03, p. 143–151, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[25] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 25010 – Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models**. p. 1–43. ISO/IEC, 2008.

[26] JANOWICZ, K.; COMPTON, M.; ROURE, K. T. A. A. D. D. **The stimulus-sensor-observation ontology design pattern and its integration into the semantic sensor network ontology**. In: *The 3rd International workshop on Semantic Sensor Networks 2010 (SSN10) in conjunction with the 9th International Semantic Web Conference (ISWC 2010)*, 2010.

[27] JAROUCHEH, Z.; LIU, X.; SMITH, S. **An approach to domain-based scalable context management architecture in pervasive environments**. *Personal and Ubiquitous Computing*, 16(6):741–755, 2012.

[28] LEFORT, L.; HENSON, C.; TAYLOR, K.; BARNAGHI, P.; COMPTON, M.; CORCHO, O.; GARCIA-CASTRO, R.; GRAYBEAL, J.; HERZOG, A.; JANOWICZ, K.; OTHERS. **Semantic sensor network xg final report**. *W3C Incubator Group Report*, 28, 2011.

[29] LOPES, J. A.; 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*, WebMedia '13, p. 43–50, New York, NY, USA, 2013. ACM.

[30] MAIA, M. E. F.; FONTELES, A.; NETO, B.; GADELHA, R.; VIANA, W.; ANDRADE, R. M. C. **Loccam - loosely coupled context acquisition middleware**. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, p. 534–541, New York, NY, USA, 2013. ACM.

[31] MAKRIS, P.; SKOUTAS, D.; SKIANIS, C. **A survey on context-aware mobile and wireless networking: On networking and computing environments' integration**. *Communications Surveys Tutorials, IEEE*, 15(1):362–386, First 2013.

[32] MARANHÃO, G. M. **Interpretação e disseminação de contexto: filtragem semântica, projeto arquitetural e estudo de caso em Saúde**. In: *Dissertação de Mestrado, Instituto de Informática da Universidade Federal de Goiás*. 2015. 1–156.

[33] MARANHÃO, G. M.; SENE JÚNIOR, I. G.; BULCÃO NETO, R. D. F. **Anatomy of a semantic context interpreter with real-time events notification support**. In: *Proceedings of the 20th Brazilian Symposium on Multimedia and the Web*, WebMedia '14, p. 159–162, New York, NY, USA, 2014. ACM.

[34] MARTIN, D.; LAMSFUS, C.; ALZUA, A. **Automatic context data life cycle management framework**. In: *Pervasive Computing and Applications (ICPCA), 2010 5th International Conference on*, p. 330–335, Dec 2010.

[35] MITSCHICK, A.; PIETSCHMANN, S.; MEIβNER, K. **An ontology-based, cross-application context modeling and management service**. *International Journal on Semantic Web & Information Systems*, 6(1):39–54, Jan. 2010.

[36] NOY, N. F.; MCGUINNESS, D. L.; OTHERS. **Ontology development 101: A guide to creating your first ontology**, 2001.

[37] OH, Y.; HAN, J.; WOO, W. **A context management architecture for large-scale smart environments**. *Communications Magazine, IEEE*, 48(3):118–126, March 2010.

[38] PASPALLIS, N.; PAPADOPOULOS, G. A. **A pluggable middleware architecture for developing context-aware mobile applications**. *Personal and Ubiquitous Computing*, 18(5):1099–1116, 2014.

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

[40] PREUVENEERS, D.; NOVAIS, P. **A survey of software engineering best practices for the development of smart applications in ambient intelligence**. *J. Ambient Intell. Smart Environ.*, 4(3):149–162, Aug. 2012.

[41] RAMPARANY, F.; POORTINGA, R.; STIKIC, M.; SCHMALENSTROER, J.; PRANTE, T. **An open context information management infrastructure the ist-amigo project**.

In: *Intelligent Environments, 2007. IE 07. 3rd IET International Conference on*, p. 398–403, Sept 2007.

[42] ROMAN, M.; HESS, C.; CERQUEIRA, R.; RANGANATHAN, A.; CAMPBELL, R.; NAHRSTEDT, K. **A middleware infrastructure for active spaces**. *Pervasive Computing, IEEE*, 1(4):74–83, Oct 2002.

[43] SEHIC, S.; LI, F.; NASTIC, S.; DUSTDAR, S. **A programming model for context-aware applications in large-scale pervasive systems**. In: *Proceedings of the 2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, WIMOB '12, p. 142–149, Washington, DC, USA, 2012. IEEE Computer Society.

[44] SHETH, A. **Internet of things to smart iot through semantic, cognitive, and perceptual computing**. *IEEE Intelligent Systems*, 31(2):108–112, Mar 2016.

[45] SIGG, S.; GORDON, D.; VON ZENGEN, G.; BEIGL, M.; HASELOFF, S.; DAVID, K. **Investigation of context prediction accuracy for different context abstraction levels**. *Mobile Computing, IEEE Transactions on*, 11(6):1047–1059, June 2012.

[46] SIGG, S. **Development of a novel context prediction algorithm and analysis of context prediction schemes**. PhD thesis, University of Kassel, 2008.

[47] STRANG, T.; LINNHOFF-POPIEN, C. **A context modeling survey**. In: *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.

[48] SUBBARAJ, R.; VENKATRAMAN, N. **A systematic literature review on ontology based context management system**. In: *Emerging ICT for Bridging the Future - Proceedings of the 49th Annual Convention of the Computer Society of India CSI Volume 2*, volume 338 de **Advances in Intelligent Systems and Computing**, p. 609–619. Springer International Publishing, 2015.

[49] VEIGA, E. F.; BULCAO-NETO, R. F. **Um Serviço de Representação Ontológica de Contexto Baseada no Padrão de Projeto Estímulo-Sensor-Observação**. *VIII Simpósio Brasileiro de Computação Ubíqua e Pervasiva (SBCUP)*, p. 1–10, 2016.

[50] VEIGA, E. F.; MARANHAO, G. M.; BULCAO-NETO, R. F. **Development and scalability evaluation of an ontology-based context representation service**. *IEEE Latin America Transactions*, 14(3):1372–1379, March 2016.

[51] VEIGA, E. F.; BULCAO-NETO, R. F. **An ontological approach for information management of public transport networks**. In: *Proceedings of the Annual Conference*

on Brazilian Symposium on Information Systems: Information Systems: A Computer Socio-Technical Perspective - Volume 1, SBSI 2015, p. 66:493–66:500, Porto Alegre, Brazil, Brazil, 2015. Brazilian Computer Society.

[52] VEIGA, E. F.; MARANHÃO, G. M.; BULCÃO-NETO, R. D. F. **Apoio ao desenvolvimento de aplicações de tempo real sensíveis a contexto semântico**. *IX Workshop de Teses e Dissertações do XX Simpósio Brasileiro de Sistemas Multimídia e Web*, p. 1–4, 2014.

[53] VEIGA, E. F.; SENA, M. V. O.; BULCÃO-NETO, R. D. F. **Padrões, ferramentas e boas práticas no desenvolvimento de software para web semântica**. *XI Simpósio Brasileiro de Sistemas de Informação*, p. 25–31, 2015.

[54] VIEIRA, V.; TEDESCO, P.; SALGADO, A. C. **Designing context-sensitive systems: An integrated approach**. *Expert Systems with Applications*, 38(2):1119–1138, 2011.

[55] WANG, X.; ZHANG, D. Q.; GU, T.; PUNG, H. **Ontology based context modeling and reasoning using owl**. In: *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, p. 18–22, March 2004.

[56] WEISER, M. **The computer for the 21st century**. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, July 1999.

APÊNDICE A

Avaliação de Manutenabilidade do Componente *Hermes Widget*

Os requisitos relacionados ao projeto arquitetural do componente *Hermes Widget* estão ligados de forma intrínseca às sub-características de manutenabilidade, como apresentado na Tabela A.1. Dessa forma, a manutenabilidade de *Hermes Widget* reflete no cumprimento destes requisitos, validando seu projeto arquitetural.

Tabela A.1: *Relações entre manutenabilidade e os requisitos de projeto de Hermes Widget.*

Requisito	**Manutenabilidade**
Arquitetura em camadas	– *Modularidade*: aplicada tanto internamente nas camadas do componente *Hermes Widget*, como também externamente, na relação do componente com o restante da arquitetura do CMS *Hermes*. – *Analisabilidade*: a divisão do software em camadas bem definidas contribui para a identificação de possíveis problemas e manutenção do componente. – *Conformidade*: utilização de padrões de projeto como *Facade*, *Transfer Object* e *Factory*, que contribuem para o desacoplamento entre as camadas.
Componentes reutilizáveis	– *Reusabilidade*: o componente *Hermes Widget*, que possui como principal serviço a representação ontológica de contexto, pode ser reutilizado em diferentes domínios de aplicação. – *Capacidade de mudança*: o componente oferece interface para ser modificado de acordo com o domínio de aplicação. – *Estabilidade de modificação*: independente das modificações realizadas no serviço de representação o componente oferece todos os serviços sem nenhum impacto e pode ser validado em diferentes cenários.

Propriedades do código fonte do *Hermes Widget*

A seguir serão apresentadas as relações entre cada sub-característica de manutenabilidade e as propriedades do código fonte do componente de software *Hermes Widget*, extraídas através da ferramenta de análise *SonarQube* [1]. Os referenciais para esta análise são as sub-características de manutenabilidade e o *Modelo de Manutenabilidade SIG* [21].

1. **Volume**: através da ferramenta *SonarQube* verificou-se que *Hermes Widget* possui 849 linhas de código-fonte, sendo classificado como um sistema *muito pequeno* (+ +). Quanto maior a quantidade de linhas, mais difícil se torna a análise do sistema para identificação de anomalias ou implementação de modificações, bem como o seu reuso. De acordo com a avaliação de volume do modelo SIG, o componente *Hermes Widget* atende as sub-características *analisabilidade* e *reusabilidade*.
2. **Complexidade por unidade**: *Hermes Widget* possui 849 linhas de código-fonte e 49 métodos, distribuídos em 21 classes no total. A partir dos dados gerados pelo *SonarQube*, foi verificado que 192 linhas de código são consideradas moderadas (22,61%) e 657 (77,39%) são consideradas simples, indicando que o software é classificado como *simples* (+ +), satisfazendo as sub-características *analisabilidade*, *testabilidade* e *modularidade*.
3. **Duplicação**: de acordo com a ferramenta *SonarQube* não foram encontradas duplicidades no componente *Hermes Widget* (0,0%). Dessa maneira, o software foi classificado como (+ +) nesta propriedade, atendendo as sub-características de manutenabilidade supracitadas e evidenciando a qualidade de projeto do componente.
4. **Tamanho por unidade**: uma vez que a maior parte dos métodos se enquadram na categoria simples (com raras exceções de métodos de complexidade moderada), *Hermes Widget* é classificado como (+ +) neste quesito.
5. **Teste de unidade**: Não foram desenvolvidos testes automatizados para o *Hermes Widget*, dessa forma, o componente foi classificado como (– –) nesta propriedade.

Análise da manutenabilidade

A Tabela A.2 apresenta uma análise detalhada das propriedades do código-fonte e a atribuição do indicador de manutenabilidade a cada camada do componente *HermesWidget*, de acordo com o Modelo de Manutenabilidade SIG [21]. Uma vez que não foram encontradas *duplicidades* no código, essa propriedade não foi adicionada à tabela. A propriedade *teste de unidade* também não foi incluída pois não foram realizados testes automatizados, sendo uma limitação desta avaliação.

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

Tabela A.2: *Manutenabilidade por camadas do Hermes Widget.*

Camada *Hermes Widget*	Volume (LOC)	Tamanho da unidade	Complexidade da unidade	Indicador
HW_RepresentationServiceDomain	192	10	19	+
HW_SensorDomain	92	2	9	+ +
HW_TransferObject	74	0	0	+ +
HW_RepresentationServiceSensor	52	3	7	+ +
HW_HistoryService	44	5	5	+ +
HW_NotificationService	36	2	3	+ +
HW_ManagerConfigurator	35	2	4	+ +
HW_ManagerServiceFactory	31	4	7	+ +
HW_SensorClient	23	4	4	+ +
HW_RepresentationService	23	1	2	+ +
HW_ManagerFacade	21	4	4	+ +
HW_ConfigurationService	15	1	2	+ +
HW_CommunicationService	11	2	2	+ +
HW_SensorFacade	9	1	1	+ +
HW_ManagerClient	9	3	0	+ +
HW_SensorServiceFactory	9	1	1	+ +

A Tabela A.3 sintetiza a avaliação realizada apresentando a relação entre cada propriedade do código-fonte com as sub-características de manutenabilidade, bem como a avaliação do componente *Hermes Widget*, de acordo com a análise das propriedades e também das sub-características, apresentadas nesta seção.

Tabela A.3: *Avaliação da manutenabilidade de Hermes Widget.*

		Modelo de Manutenabilidade SIG					
		Volume	Complexidade	Duplicação	Tamanho da unidade	Teste de unidade	***Hermes Widget***
ISO/IEC 25010	Analisabilidade	✓		✓	✓	✓	✓
	Capacidade de Mudança		✓	✓			✓
	Estabilidade de Modificação					✓	
	Testabilidade		✓		✓	✓	✓
	Modularidade		✓		✓		✓
	Reusabilidade	✓		✓			✓
	Conformidade	✓	✓	✓	✓		✓
	Hermes Widget	+ +	+ +	+ +	+ +	− −	