Journal of Universal Computer Science, vol. 22, no. 8 (2016), 1123-1147
submitted: 9/9/15, accepted: 29/7/16, appeared: 1/8/16 © *J.UCS*

A Semantic Filtering Mechanism Geared Towards Context Dissemination in Ubiquitous Environments

Guilherme Melo e Maranhão, Renato de Freitas Bulcão-Neto
(Instituto de Informática
Universidade Federal de Goiás
Goiânia-GO, Brazil
guilhermemaranhao@inf.ufg.br, renato@inf.ufg.br)

Abstract: A great challenge in context-aware computing is dealing with the heterogeneity and volume of sensors data. A problem regarding that scenario is to notify context-aware applications, which have distinct interests of context events in terms of volume, semantic and complexity, in an efficient and relevant manner. Aiming to solve this problem, this research focuses on a new approach for filtering semantic context towards supporting context dissemination. This mechanism is to be aligned with the reasoning capabilities of a context-aware solution and also be maintainable and extensible to efficiently support changes in an ontological model. A performance evaluation is carried out in a simulated scenario of vital signs monitoring in Intensive Care Units and wards. Hermes Interpreter's behaviour is analysed when dealing with filters of different complexities and also an increasing number of subscribers per vital sign. Results demonstrate the high cost of the semantic filtering mechanism in comparison with pure context reasoning activities.
Key Words: Filtering, Semantics, Context Dissemination, Experimentation
Category: H.3.3, J.3, L.7.0

1 Introduction

The data's life cycle in context-aware systems usually contains four phases: acquisition, modelling, reasoning and dissemination [Perera et al. 2014]. These phases have been strongly influenced by the heterogeneity of published data, as a consequence of the increasing deployment of sensors as well as the increasing amount of shared data among systems.

The heterogeneity is mainly related to the different information types, e.g., geographic coordinates, personal profiles, vital sign measures, humidity or urban traffic. To describe this type of complex domain, the literature has reported ontology-based models as those that better fulfill these issues, due to its high formalism and expressiveness [Bettini et al. 2010, Perera et al. 2014].

As more sensors with distinct purposes are deployed, more new concepts (measures, units or relationships) are introduced, which in turn produce a major impact on the context domain. For instance, imagine that a hospital has acquired a new type of sensor to monitor body temperature in ICU patients. An ontological context model for that hospital would have to be changed to support that new sensor data.

Therefore, in order to provide high maintainability and extensibility of both systems and context models it is recommended that the context representation should be independent and decoupled from the context-aware systems themselves [Perera et al. 2014]. The ontological modelling also enables this separation of concerns, because it makes the domain knowledge (domain concepts) independent from the operational knowledge (executable code).

In addition, such type of modelling enables reasoners to discover new facts based on explicitly stated context data against the semantic and logic of a ontological model. Combining that reasoning technique with rules also enables the detection of high level context situations, also called high level events, such as the health state of patients based on vital signs measurements [Bettini et al. 2010].

However, applications may have different needs in a context-aware environment in such way that a high level event which is relevant to one application, may be not to other ones. Therefore, before the dissemination phase, a filtering process aligned with an ontological model is recommended in order to deliver the correct data to the right application [Perera et al. 2014].

Currently, context-aware systems lack on providing solutions that enable applications to choose their context of interest in a semantic and dynamic way. For instance, in the financial stock market, each broker-agent has her own events of interest, most of them related to price movements, market trends associated to companies' business, and all complex pattern events which concerns this scenario. In a shopping center scenario, consumers have interest in different products, and for each product, a set of specific details such as price, color, model, and so on.

Another motivating scenario is the vital sign monitoring in hospital ICUs. There are physicians and nurses with distinct events of interest in terms of patients and their respective treatments. For instance, a nurse aids a patient with blood pressure and oxygen saturation issues, whereas a physician attends a newly operated patient with pulse rate limitations. Dealing with all these different needs to deliver a consistent and relevant context to applications is the problem that this paper addresses.

Regarding context-aware systems which execute upon the publish-subscribe paradigm, we advocate the combination of a semantic filtering approach with the subscription of topics, because these usually correspond to static and high granularity structures, which do not satisfy subscribers with dynamic and very specific demands.

Another reason is to increase the filtering possibilities so that subscribers might request for concepts and terms according to the semantics described in an ontology. For instance, instead of subscribing for events based on syntactical analysis, e.g. *"patient_name=John"* or *"oxygen_saturation ≤ 90%"*, subscribers may request for events based on an ontological class or property, e.g. *":pulse_rate rdf:type :tachycardia"* or *":patient :isLocatedAt :room4"*.

In this paper we present a new mechanism to support the data's life cycle in context-aware systems, specially the context dissemination phase. We propose to align our results according to the design principles recommended by the literature and also evaluate them in a case study, that enables expliciting the contributions for the context reasoning and dissemination phases.

The contributions include: (i) a semantic filtering mechanism for context data to notify the context-aware applications according to their events of interest; (ii) the *Hermes Interpreter*[1] component, which reasons and filters semantic context data as well as manages the events dissemination; (iii) a comparison study between the amount of time spent by the filtering and the reasoning processes in a scenario of vital signs monitoring of patients; and (iv) a scalability experiment which analyses the HI's behaviour in response to an increasing number of subscribers per vital sign, with filters of different complexities.

The remaining of this paper is organized as follows: Section 2 overviews specifications for semantic modelling used in the case study; Section 3 details the semantic filtering mechanism implemented into *Hermes Interpreter*; Section 4 presents the case study with a study of performance evaluation and a scalability evaluation; Section 5 discusses the results; Section 6 discusses related work, and Section 7 presents the concluding remarks and future work.

2 Theoretical Foundation

2.1 Semantic Modelling Specifications

The Semantic Web is a web of data described and linked in ways to establish context or semantics that adhere to defined grammar and language constructs [Hebeler et al. 2009]. The Semantic Web body of knowledge is composed of modelling specifications that form the building blocks of this work: RDF (Resource Description Framework) [Wood et al. 2014], OWL2 (Ontology Web Language) [Hitzler et al. 2012], SWRL (Semantic Web Rule Language) [Horrocks et al. 2004] and SPARQL [Harris and Seaborne 2013].

RDF provides a graph-based data representation in which nodes represent a *subject* and an *object* linked to the latter. The edges linking subjects and objects are defined as *predicates*. An RDF statement is composed of a triple used to describe any knowledge domain as well as to be shared among systems. For instance, an RDF statement (:patient, :hasPulseRate, "120") means that the pulse rate (predicate) of a particular patient (subject) is 120 bpm (object).

OWL2 is the standard language to define complex ontologies in the Semantic Web. Its semantics provides constructors to represent axioms and facts, which provide information about classes, properties, and individuals in an ontology.

[1] Throughout this paper, *Hermes Interpreter* will also be referred as HI.

Those constructors allow to specify symmetric, transitive and inverse properties, disjoint classes and properties, equivalent classes and individuals, among others. The OWL2 expressiveness, formality and reasoning support are of great significance to model and reason about the semantics of context information.

SWRL is an ongoing effort towards the standardization of a language for describing rules in the Semantic Web. SWRL complements the semantics provided by OWL2 with terms and relations that are not supported by OWL2, or even hard to be expressed in OWL2. The structure of SWRL rules includes the *body* $\rightarrow$ *head* format: the former contains the conditions clauses whereas the latter describes the consequent clauses. By extending the semantics of OWL2, SWRL has been a feasible alternative to enrich the semantics of concepts.

SPARQL is the standard language to query over RDF statements (or triples). A SPARQL query can be converted in a graph G that will be matched against the data in an RDF data source Gr (which is also a graph). In other words, a SPARQL query result includes all triples of G representing a subgraph of Gr. Using that same graph matching mechanism, the SPARQL syntax provides four different query forms, such as the CONSTRUCT form. It allows you to transform a query result into any kind of RDF graph you can design, as long as each triple is valid. The CONSTRUCT query form provides an easy and powerful way not only to transform data from one RDF graph or OWL ontology into another graph, but also to add triples to RDF repositories or even combine them with other RDF graphs.

2.2 MSVH Ontology

The MSVH ontology models the semantics of the human vital signs monitoring activity, particularly in Intensive Care Units and wards [Bastos et al. 2014]. The ontology represents five vital signs (blood pressure, body temperature, oxygen saturation, pulse rate and respiratory rate). Besides, it models related concepts including the measured value with temporal information, measurement unit, the minimum and maximum parameterized values for a particular patient, among others. An excerpt of the MSVH ontology is described in Figure 1.

MSVH also defines rules formatted in the SWRL specification to complement its OWL2 semantic concepts. These rules express abnormalities associated with each vital sign. For instance, there is a rule to detect the *tachycardia* state associated with the pulse rate vital sign (greater or equal than 100) described in Listing 1. MSVH is the core ontology in the case study presented in Section 4.

Listing 1: SWRL rule for detecting tachycardia [Bastos et al. 2014].

```
'pulseRate'(?pr), 'pulseRateMeasurementDatum'(?prd),
isPulseRateMeasurement(?prd,?pr), valuePulseRate(?prd,?vpr),
nonNegativeInteger[>= 100](?vpr) -> Tachycardia(?prd)
```

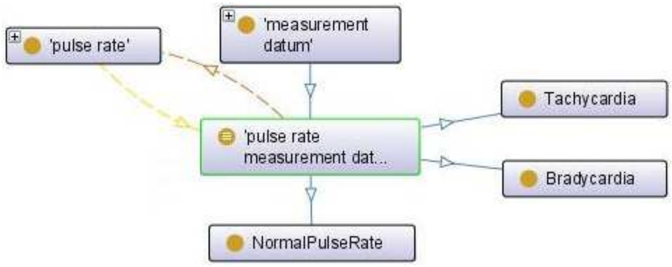

Figure 1: The OWL2 representation for the MSVH 'pulse rate' vital sign, measurement data, and corresponding health states (normal, bradycardia and tachycardia) [Bastos et al. 2014].

2.3 Data Interchange Format

Context-aware applications should make use of a flexible and structured mechanism to interchange and describe data [Perera et al. 2014]. The JSON format (Javascript Object Notation) is a lightweight and human-readable standard for data interchanging and description. It is programming language independent, easy to read and retrieve information, and also built upon universal structures including nested name/value pairs and ordered list of values.

Using the JSON format, the semantic filtering mechanism proposed associates in a decoupled way each filter parameter with the respective graph pattern G in a SPARQL query. Besides, JSON keeps the maintainability and extensibility of the semantic specifications in the filtering mechanism, and also enables the reasoning flexibility over different topics.

3 The Semantic Filtering Mechanism

As depicted in Figure 2, the semantic filtering mechanism proposed works as a layer between context-aware applications and a publish-subscribe middleware, offering semantic services such as semantic reasoning and filtering. It is implemented in a software component called *Hermes Interpreter*, which is part of the *Hermes* semantic context management system [Maranhão et al. 2014, Veiga et al. 2014]. *Hermes* manages the context data life cycle, from acquisition to dissemination, and its communication support is based on the publish-subscribe paradigm (applications or *Hermes* services subscribe for topics of interest).

Besides context filtering, the HI component provides context reasoning capabilities based on the semantics of OWL2 ontologies and SWRL rules. The HI reasoning feature has been proven to be maintainable and extensible to deal with changes on the context model as well as flexible to configure the reasoning technique according to subscribers' filters.

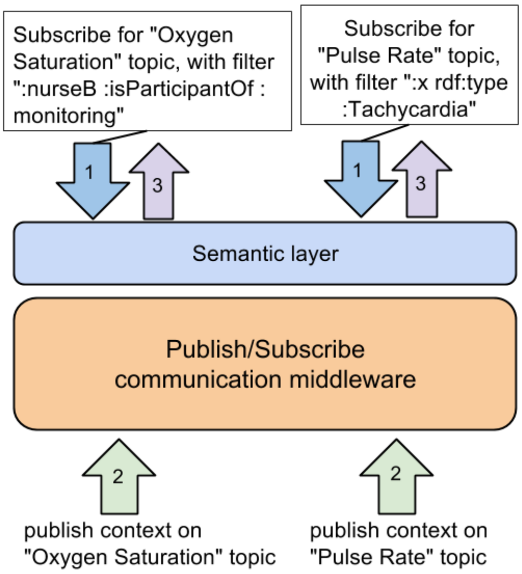

Figure 2: The semantic filtering mechanism.

In Figure 2, an application subscribes (1) topics for the oxygen saturation and the pulse rate vital signs, respectively. The first subscription filters every oxygen saturation monitoring whom a particular nurse is responsible for, whereas the second subscription filters every pulse rate measurement describing *tachycardia*.

The first subscriber is notified (3) when a context provider publishes (2) information (i.e. graph Gr) that matches the SPARQL filter (i.e. graph G). In this case, reasoning support is not required. On the other hand, the *tachycardia* state required for the second application will be notified (3) only if the reasoning support is set. Every pulse rate measurement published (2) is then evaluated against the SWRL rule describing *tachycardia* shown in Section 2.2.

3.1 Requirements

Regarding the *reasoning* characteristic, HI should support multiple types of reasoning techniques. Due to the increasing of complexity related to application domains, different high level context situations emerge to be reasoned, and each situation requires an appropriate technique to be used. For instance, rules-based reasoning is recommended for the detection of high level events, whereas ontology-based reasoning is suitable for the automatic derivation of new knowledge about the current context [Hebeler et al. 2009, Bettini et al. 2010].

Concerning the *filtering* mechanism, HI should also (i) support ontological models filtering; (ii) decouple source code from an ontological model; (iii) prioritize the execution of filters whose parameters are explicitly known; and (iv) be aligned to reasoning techniques supported by a context-aware solution.

The *first* requirement aligns the filtering process with the semantics of context, enabling subscribers to ask for events related to the knowledge domain, e.g., the occurrence of a particular ontological class or property in the published context in which a subscriber is interested. Heterogeneous data produces complex domains and, consequently, complex filtering needs. Bringing the semantics of context to the filtering capabilities will help achieving these needs.

Regarding the *second* requirement, the purpose is to ease the maintenance and extension of the filters supported even when an ontological model changes. As domains become more complex, a demand for the maintenance and extension of ontological models also arises. These context model changes directly influence the filtering mechanism.

The *third* requirement is related to the differences of complexity among the possible filters subscribed for each topic. Some of them do not have to be executed after the reasoning phase, because the knowledge required is already described in the published context data, e.g., a subscriber may simply require to be notified if the measure of a vital sign is greater than a constant. However, subscribers may also require for even more complex filters in which their execution must happen after the reasoning phase, e.g., when the information is not available in the current context data yet. Therefore, it is reasonable that simplest filters be executed so that their subscribers are notified soon. Considering the ontology-based reasoning of complex ontological models, the amount of data inferred after this step considerably increases filtering time.

Finally, the *forth* requirement aims to make the HI component flexible regarding reasoning techniques and previously subscribed filters. For instance, if a subscribed filter demands a context situation which is described in a pre-defined rule, HI should use the rule-based technique to infer the data required by that subscriber when a publication on the respective topic occurs.

3.2 Architecture

The HI's architectural model is depicted in Figure 3. There are two main flow of events concerning the HI execution: one flow starts when a context-aware application subscribes for a specific topic; the other one starts when context publishers notify data gathered to the Hermes infrastructure.

Regarding the first flow, the *HI Filter Service* communicates with the *HI Situation Factory* to obtain the appropriate *HI Filter Situation* so that a new filter can be built for this subscriber. Using the JSON format, this layer describes filter parameters available to a respective topic including a SPARQL clause to query against the ontological context model, the reasoning type, among others exposed in Table 1. By taking advantage of the JSON format features, HI has

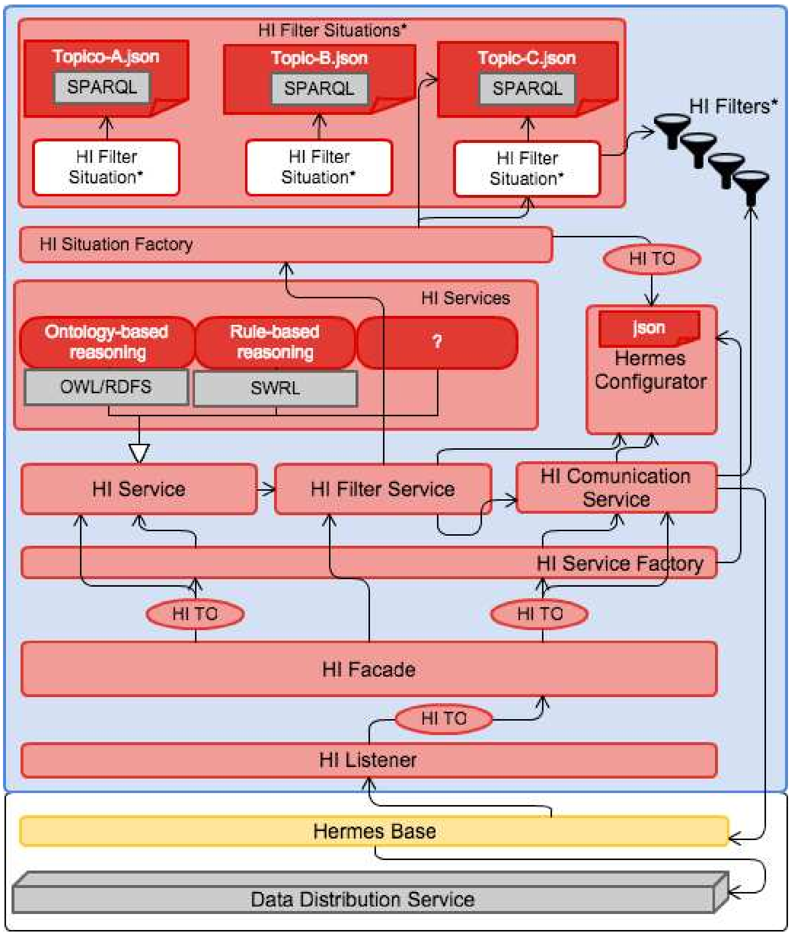

Figure 3: The *Hermes Interpreter* architecture [Maranhão et al. 2014].

high maintainability and extensibility because the description of filter parameters are kept apart from the executable code. In the next step, the freshly built filter is inserted into a queue corresponding to its topic, and it will be executed when a notification to this topic occurs.

With respect to HI's second flow, the *HI Service* provides a single interface for the reasoning service aiming to support other inference mechanisms. The *HI Filter Service* manages (creation and execution) subscribers' filters. In both cases, the *HI Filter Service* calls the *HI Situation Factory* layer, which returns the corresponding *HI Filter Situation* to the current topic. Once again, JSON enables the architecture to be decoupled from ontology concepts, mapping the filter parameters with the required reasoning type.

Considering the requirements described in Section 3.1, the *HI Filter Situations* layer addresses the filtering of ontological models and the decoupling of source code from ontological models. By distributing subscribers' filters to queues according to its complexity, the *HI Filter Service* layer prioritizes the execution of filters whose parameters are explicitly known such as those that do not require a reasoning step. Finally, the filtering mechanism is aligned with the reasoning techniques supported by means of the *HI Service* layer.

As previously stated, there are metadata supporting the filtering mechanism for each filter parameter registered in a JSON filter file. The Table 1 presents the *ApneaAlarm* parameter with its respective metadata. The *ApneaAlarm* parameter is one of the filters supported by the *Respiratory Rate* topic. The aim of each filter parameter in Table 1 is described next.

Table 1: *ApneaAlarm* filter parameters and corresponding metadata.

Metadata	**Value**
has disjoint class	true
has parameter	false
clause	?x msvh:hasMonitoringRespiratoryRate ?m . ?m msvh:hasMeasurementRespiratoryRate ?measure . ?measure rdf:type msvh:ApneaAlarm . ?sAlarm rdfs:subClassOf msvh:RespiratoryRateAlarm . ?measure rdf:type ?sAlarm .
filter clause	null
reasoning technique	rule-based

- **has disjoint class**: it informs whether disjoint classes[2] on the ontological schema exist or not. If there are triples with disjoint classes in the graph G, HI must include a UNION clause in the SPARQL query to separate them;
- **has parameter**: it informs whether the query parameter requires a SPARQL FILTER clause or not, which allows flexibility with fine-grained results;
- **clause**: it describes the RDF graph pattern G to be matched against the RDF source graph Gr;
- **filter clause**: it describes a SPARQL filter clause template; e.g. FILTER (?measure >"some_value"), in the case of value comparisons;
- **reasoning technique**: it informs one of the reasoning techniques supported that HI should set to a corresponding topic.

[2] Disjoint classes are those that do not share individuals or instances.

3.3 Filter Categories

Three categories of filters are supported to deal with different demands in terms of filter complexity and the instant of execution of the filtering mechanism:

- **FBR - Filter before reasoning**: it is composed of filters processed *before* the reasoning phase, because they require information already known by HI, or data already published in the original RDF context model. This type of filter includes the clauses AND and FILTER of the SPARQL syntax;
- **FAR - Filter after reasoning without disjoint classes**: it is composed of filters processed *after* the reasoning phase, because they require unknown data, i.e. not previously known in the current RDF context model. It also implements the clauses AND and FILTER of the SPARQL syntax;
- **FARD - Filter after reasoning with disjoint classes**: similar to the FAR filters, this type of filter also demands a previous reasoning phase. In addition to the SPARQL support of the FBR and FAR filters, this filter implements the UNION clause to separate disjoint classes in the query filter.

4 Case Study

In this section, we present experiments in a vital signs monitoring scenario in which HI's reasoning and filtering capabilities are evaluated.

The MSVH ontology, presented in Section 2, works as the underlying ontological context model. The current version of the HI component is based on the Apache Jena framework, which provides developers with APIs and a query processor with support to the RDF, SPARQL and OWL2 specifications [Apache Jena 2014]. Besides, the Pellet reasoning engine [Sirin et al. 2007] is integrated to the HI component to support both the OWL2 ontology-based and the SWRL rule-based reasoning processes.

Although the role of both context providers and applications are simulated by programs developed by the authors, real-world vital signs are used from the MIMIC public database [Goldberger et al. 2000], which contains registers from patients during their admission in ICUs and wards.

This patient monitoring scenario is contribution of the Nursing department of the Clinics Hospital of the Federal University of Goiás, Brazil. For clarification purposes, we describe this case study by means of Figures 4, 5 and 6.

Every step which *Hermes* executes to make the user app's subscription is depicted in Figure 4. A DDL object is shown with two filter parameters: age and abnormality (step 1). Next, HI is notified (2) to create a filter to that subscriber. The *HI Situations* fabric is instantiated (3) to obtain an instance of the *HI Filter Situation* responsible for the respective topic. The fabric returns that object and a JSON file which describes the SPARQL templates to create the required filter (4 and 5). A *Filter* object is instantiated, in which the SPARQL query just created is added (6). This *Filter* object is added to the correspondent list, according to its complexity (7). As it needs the rule step to be executed, its list is the one that contains filters to be executed after the reasoning step. Finally, the topic *Pulse Rate* is set to rule reasoning type, due to the existence of this newly filter (8 and 9).

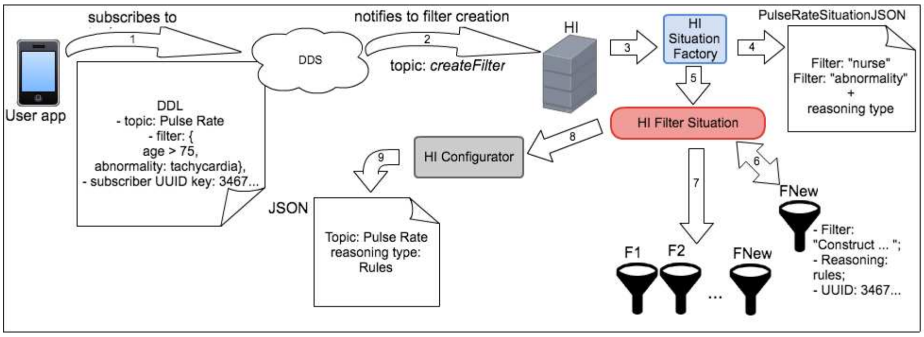

Figure 4: User application subscribes for a specific topic containing a parameterized filter – in this case, it is to be notified when 75 year old patients get tachycardia.

Figure 5 depicts the context filtering process without reasoning step. The *HI Rule Service* layer (steps 1 and 2) communicates the original context in the RDF format to the *Filter Service* object, so that it performs the filtering execution of the FBR filters (3). For each matched filter, it requests the *HI Communication Service* to proceed with the context publication (4 to 6).

Finally, Figure 6 refers to the reasoning step so that *Hermes* can identify new context facts according to the ontology modeling. *HI Rule Service* instantiates the reasoning engine (step 1) which identifies new facts (2) related to the context data (green nodes and edges). The *Filter Service* object gets the FAR and FARD filters (3 and 4), which are kept by the respective topic's *HI Situation*. Then, for each matched filter, the *Communication Service* is invoked (5) to publish the context in the DDS network so that the respective subscribers could be notified (6 and 7).

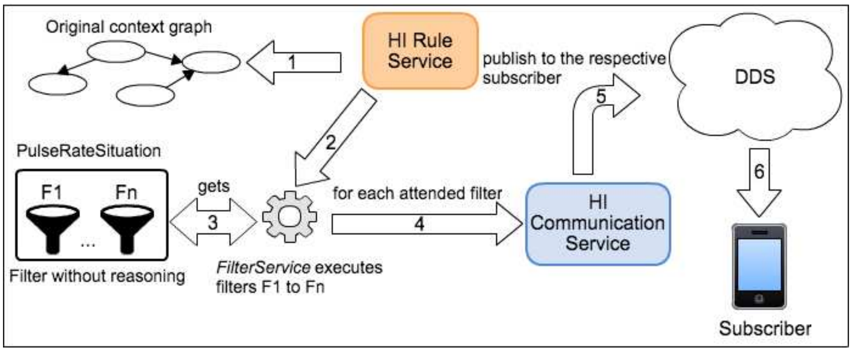

Figure 5: Context filtering **without** a reasoning step.

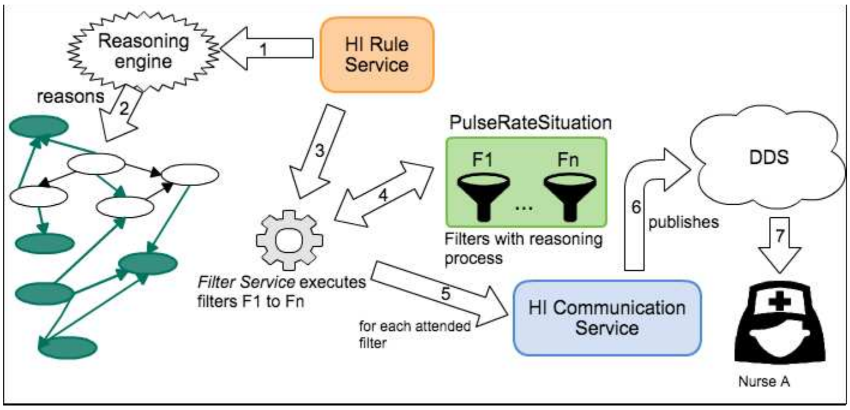

Figure 6: Context filtering **with** a reasoning step.

4.1 Functional Validation

This experiment aims to validate the HI's behaviour in a scenario proposed by a Nursing staff, in which the maintainability and the extensibility of the component is validated along the changes in the ontology model. Besides, the flexibility related to reasoning types and topics are demonstrated. Finally, the main purpose is to validate how the component notifies multiple subscribers with distinct interests of context.

The scenario includes 14 patients assisted by 3 nurses for 6 hours: Nurse "A" and Nurse "B" aid 5 patients each, whereas Nurse "C" aid 4 patients. Every subscriber's filter should contain the following parameters: (i) the patient ID or the responsible nurse; (ii) monitoring date and time; and (iii) a vital sign reference value or a vital sign abnormality.

The following is described a filter subscribed by nurse "A" to the *pulse rate* topic:

- responsible nurse: Nurse "A";
- abnormality: *bradycardiaalarm* or *tachycardiaalarm*;
- monitoring interval: between 12 pm and 5 pm of 03/14/2015.

Three scenarios are presented next to explore HI's functionalities: (i) topic subscription with filter, (ii) context publishing, and (iii) context dissemination after changes in the ontological model.

4.1.1 Topic Subscription with Filter

This experiment aims to validate a topic subscription with filter. The evidence is the creation of a filter (a SPARQL query) to a respective topic. As a successful result, the subscriber is notified if the situations that the nurse is looking for occur. Consider for this test the nurse "A" subscription to the *pulse rate* topic presented earlier.

The subscription result is a filter related to the respective subscriber, including the topic name, an UUID key for the subscriber identification, and a SPARQL query, which includes the filter parameters specified, as shown in Listing 2.

According to the subscribed filter, HI should detect the occurrence of *bradycardiaalarm* or *tachycardiaalarm* in the published data, which are ontological classes with SWRL rules associated in the MSVH ontology.

For the detection of these classes in an RDF context model published by the context providers, the corresponding JSON *HI Filter Situation* of the "pulse rate" topic contains the value "rule-based" for the "reasoning technique" tagged key associated to the filter parameters *bradycardiaalarm* and *tachycardiaalarm*.

Listing 2: Nurse "A" SPARQL filter.

```
CONSTRUCT {
        ?x  rdf:type  msvh:MonitoringPulseRate .
        ?x  activity:hasParticipant  ?n .
        ?n  msvh:hasRole    msvh:nurse .
        ?n  actor:hasName    ''NurseA''^^string .
        ...
        ?measure  rdf:type  ?sAlarm .
}

WHERE {
        ?x  activity:hasParticipant  ?n .
        ?n  msvh:hasRole  msvh:nurse .
        ?n  actor:hasName  ''NurseA''^^string .
        ...
```

```
        {
          ?x   rdf:type   msvh:MonitoringPulseRate .
          ?x   msvh:hasMonitoringPulseRate   ?m .
          ?m   msvh:hasMeasurementPulseRate   ?measure .
          ?measure   rdf:type          msvh:TachycardiaAlarm .
          ?sAlarm    rdfs:subClassOf msvh:PulseRateAlarm .
          ?measure   rdf:type          ?sAlarm .
        }
 UNION  {
          ?x   rdf:type   msvh:MonitoringPulseRate .
          ?x   activity:hasParticipant   ?n .
          ?n   msvh:hasRole    msvh:nurse .
          ?n   actor:hasName    ``NurseA''^^string .
          ...
          ?x   msvh:hasMonitoringPulseRate    ?m .
          ?m   msvh:hasMeasurementPulseRate   ?measure .
          ?measure   rdf:type            msvh:BradycardiaAlarm .
          ?sAlarm    rdfs:subClassOf   msvh:PulseRateAlarm .
          ?measure   rdf:type            ?sAlarm .
        }
}
```

4.1.2 Publication of Context

This test aims to verify the HI's behaviour with respect to the vital signs published by the context provider components. Among the various measurements collected from the MIMIC database, here we consider *pulse rate* measurements.

In order to be published to the "pulse rate" topic, the measurement data acquired, which is originally described in the CSV format, is converted to the RDF triple format. The measurement value chosen is 53 bpm, which is modelled according to the MSVH ontology: the date and the time of the measurement as well as some context data about the monitored patient, e.g., her identification and her nurse assistant.

Once this data is published, HI is notified and performs the reasoning and filtering processes. As detailed above, the reasoning technique to be used for the "pulse rate" topic is set to "rule-based". Next, HI calls the *HI Rules Service* to reason about the recent published context. After this process, HI calls the *HI Filter Service*, which executes each of the"pulse rate" subscribed filters so that it can identify their parameters' occurrences in the inferred RDF model.

As the reasoning process deduces that the 53 bpm published measurement means a *bradycardia* abnormality (through the corresponding SWRL rules), the "nurse" filter detects the presence of the *bradycardiaalarm* ontological class in the respective individual measurement. Besides, it verifies that the data collected is in the interval parameterized by the filter and "nurse A" is the assistant nurse. Therefore, HI creates a new RDF model, according to the specified by the SPARQL CONSTRUCT clause, to be notified to the subscriber "nurse".

4.1.3 Context Dissemination After Changes in the Ontological Model

This scenario illustrates the HI's extensibility and maintainability at run-time. To validate its extensibility, nursing professionals suggested to include the "mean blood pressure" information to help them to better assist the patients. This new requirement was implemented as a new data property, named *:hasMeanBloodPressure*, which was defined to the *:MonitoringBloodPressure* ontological class. To validate the HI's maintainability, the property *:hasName*, in the ontological class *:Person*, was renamed to *:hasPersonName*.

After the changes described in the ontological model, the JSON filter files were changed to include the parameter <*MeanBloodPressure*> and also to overwrite the property *:hasName* with *:hasPersonName*. In order to signal HI about changes in the MSVH ontology, the tagged information *updated ontology* was set to true. Based on this information, as soon as HI is notified about a new context, it updates all the pre-defined subscribers' filters so that they can include the recent ontological changes in SPARQL queries.

Continuing the validation, one subscribed for the *Blood Pressure* topic with the *MeanBloodPressure* filter parameter, as shown in Listing 3:

Listing 3: SPARQL query for the new filter.

```
CONSTRUCT {
        ...
        ?x  msvh:hasMeanBloodPressure  ?measure .
        ?n  actor:hasPersonName        ''NurseC''^^string .
 }
 WHERE {
        ...
        ?n  actor:hasPersonName        ''NurseC''^^string .
        ?x  msvh:hasMeanBloodPressure  ?measure .
        FILTER (?measure >= ''105''^^int ) .
 }
```

Then, as soon as the context providers start publishing information about the "mean blood pressure", if it is greater than 105 mmHg a new subscriber is notified, as described in the following program output sample. VSO_0000005 refers to the *BloodPressure* ontological class, and the *systolic* and *diastolic* measurements describe a *mean blood pressure measurement* >= 105.

```
*** FILTER ACCOMPLISHED ****
UUID → 8d6f98fe-4dbd-4946-8145-bef4e2dbacfc
Vital Sign VSO_0000005 measurement from patient 226n_VSO_0000005 at Sat Mar 14
14:10:53 BRT 2015
VSign measurement (valueSystolicBloodPressure) → 155 mmHg
VSign measurement (valueDiastolicBloodPressure) → 84 mmHg
Type: VSO_0000005
```

4.2 Reasoning Time Versus Filtering Time

It is reasonable that the new filtering mechanism deals with case study time restrictions. For that reason, this second experiment aims at comparing the amount of time spent by the filtering process in comparison with the reasoning time. We also consider the subdivision of the ontology-based reasoning time into four main phases, as described in [Bulcão Neto and Pimentel 2006]:

- **MLT (Model Loading Time)**: the amount of time spent to load an ontological model;
- **ST (Satisfiability Time)**: the amount of time spent to validate whether ontology facts are in accordance with ontology schema or not;
- **CT (Classification Time)**: the amount of time consumed to determine the individuals hierarchy in accordance with ontology classes and properties;
- **RT (Realization Time)**: the amount of time consumed to identify the most specific class for each ontological individual.

Different subscriber filters are defined for each filter type (FBR, FAR and FARD) as follows. Based on Nursing staff's feedback, 30 subscribers for each filter is a reasonable value for our purposes in the experiment: FBR (pulse rate topic), FAR (respiratory rate topic), and FARD (blood pressure topic).

Figure 7 summarizes the results obtained for the FBR, FAR and FARD filters, respectively. The legends in Figure 7 mean the *total average time* and the respective *standard deviation* (both in milliseconds) for each process (MLT, ST, CT, RT, and filters supported).

Results show that the filtering phase represents a significant part in the HI's processing, specially concerning the filters with disjoint classes. This cost may be more expressive if a larger number of subscribers is required.

The hardware and software settings used in this experiment are as follows:

- the HI component runs on an Intel Core 2 Duo 2,2GHz with 2 cores, 4GB RAM, Windows 7 64 bit;
- the context providers (including MIMIC-based vital signs) run on an Intel Core i5-3330 3GHz x 4, 8 GB RAM, Linux Ubuntu 64 bit; and
- the applications (representing nurses' demands) run on an Intel Core i5 2,3 GHz with 2 cores, 2GB RAM, Windows XP 32 bit virtual machine, Oracle VM VirtualBox6.

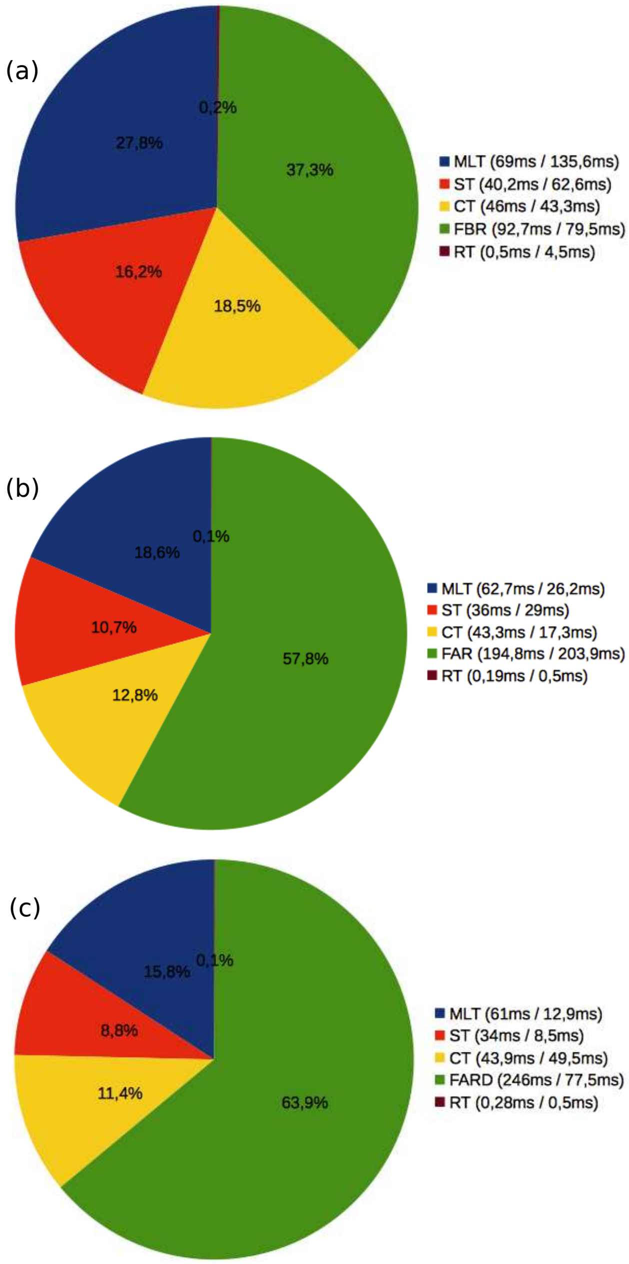

Figure 7: Comparison among reasoning variables and FBR (a), FAR (b) and FARD (c) filters, respectively.

4.3 Scalability Evaluation

Besides the comparison between the time costs of reasoning and filtering, a third experiment analyses the HI's behaviour along the time, considering the increasing of subscribers per vital sign. In terms of the case study, the response time must be acceptable according to the medical literature.

A reference response time is suggested in [Matthias et al. 2009], which propose a delay time to reduce the alarm fatigue occurrence in ICUs. They prove that 77% of alarms are ineffective or even ignored, and also claim that 67% of those alarms may be removed if there is a delay of 19s to sound the sign alarm.

As it is not a purpose of the *Hermes Interpreter* to solve the alarm fatigue problem, this response time reference is used as a parameter, i.e. an acceptable value to process the vital sign context data in an ICU monitoring scenario. Based on this, the HI's scalability is evaluated in order to obtain the amount of subscribers supported in terms of the reference response time. The hardware and software settings used in this experiment are as follows:

– the HI component runs on an Intel(R) Core(TM) 2 Duo (2,2 GHz each), 4GB RAM, Windows 7 64 Bits;

– *Hermes Widgets*: Linux Ubuntu 12,4 LTS 64 Bits, Intel Core i5-3330 with 3GHz x 4 and 7,7 GB RAM

– Context-aware applications (subscribers): Windows 7 32 Bits, Intel(R) Dual Core, 1,73 GHz and 1,5 GB RAM. All subscribers ran on this machine

As defined in the functional validation, each subscriber describes three types of filter parameters: patient ID, monitoring interval and a vital sign measure or abnormality. The association between filter categories and vital signs is FBR (pulse rate), FAR (respiratory rate), and FARD (blood pressure).

For each aforementioned pair of filter and vital sign, three experiments are performed increasing the number of subscribers (e.g. 30, 300 and 3,000). Figure 8 depicts the average processing times of FBR, FAR and FARD filters, respectively.

Table 2 summarizes the results of this experiment including the average processing times and respective standard deviation for FBR, FAR and FARD filters.

5 Discussion

This section summarizes the discussion about the experiments described in the previous section: functional validation, comparative analysis between reasoning and filtering times, and scalability evaluation.

Regarding functional validation, we show how the filtering mechanism supports subscribers' needs in a semantic fashion. In a heterogenous scenario, tagged filters are defined to enable the subscribers to combine their contexts of interest.

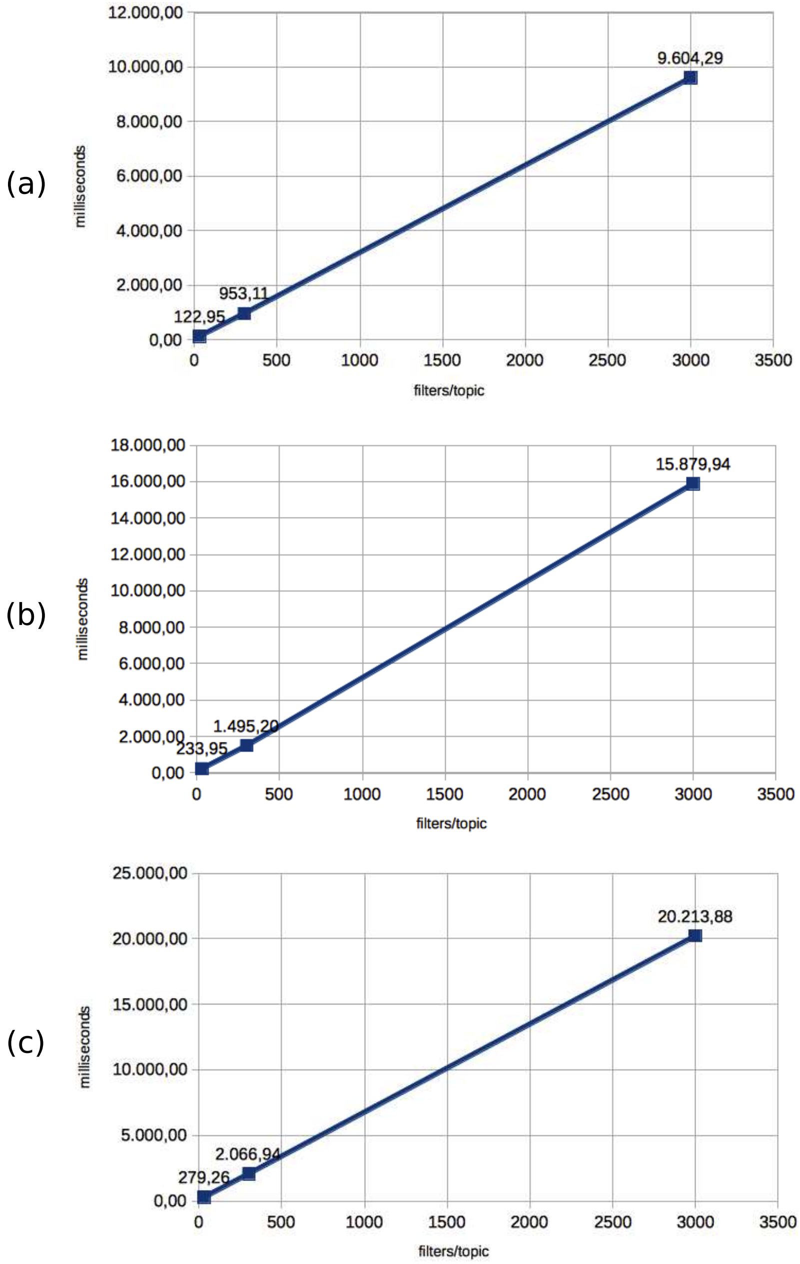

Figure 8: HI's average processing time for FBR (a), FAR (b), and FARD (c) filters, respectively.

Table 2: A summary of the scalability evaluation experiment.

Filter	Subscribers	Average Time	Std. Deviation
FBR	30	0.12s	0.82s
	300	0.9s	0.23s
	3000	9.6s	2.36s
FAR	30	0.23s	0.26s
	300	1.5s	1.65s
	3000	15.8s	16.6s
FARD	30	0.28s	0.9s
	300	2s	0.56s
	3000	20.2s	5s

The separation of concerns between domain model (i.e. the MSVH ontology) and executable code is obtained through the markup scheme tagged encoding. For that reason, HI is benefited with an extensible and maintainable solution for context reasoning and filtering. Another advantage is the transparent semantic layer which the filtering mechanism builds upon the publish-subscribe middleware, offering to subscribers a rich solution to deal with complex domains.

In terms of the comparative analysis between reasoning subprocesses and filter categories, it demonstrates how expensive is to manipulate those filters in a complex scenario such as the vital signs monitoring.

Besides, three distinct behaviours are identified concerning the three filter types: FBR filters spend less time because they use the original context data (i.e. a smaller RDF triple model); FAR and FARD filters, in turn, take longer, because both are executed over a huge RDF model with many new context data inferred. The execution of FARD filters consume more time than the FAR filters, because the query processor has to analyse all triples of the RDF source graph Gr in order to detect the occurrence of all disjoint classes.

The results obtained by the scalability evaluation show that only FAR and FARD categories with 3,000 subscribers do not meet the response time suggested by [Matthias et al. 2009]. Observe that the average time to the FAR category is 15.8s, but its high standard deviation (16.6s) makes it higher than the 19s used as reference in the experiment.

All filter types present a linear growth with the increase of subscribers. This is because the input size for each filter query was not increased along vital signs' notifications, so that the RDF data source and the SPARQL query for each filter type were always the same, varying the parameters values only. As verified by the functional validation, this was not a limitation in the HI operation.

On the other hand, if the RDF data source was increased along context notifications, FBR and FAR filters would continue being executed at polynomial time [Picalausa and Vansummeren 2011]. However, FARD filters would have their execution time exponentially increased. These studies define that SPARQL queries containing only AND and FILTER operators (i.e. FBR and FAR) are polynomially executable with the increase of input size. However, queries with UNION and AND operators are NP-Complete problems.

Therefore, if HI made use a knowledge base for storing context notifications, the maximum number of subscribers to attend the 19s reference would be smaller, because the input size would be increased along the notifications as well as the response time. Due to its exponential behaviour, a FARD filter would have its number of subscribers more influenced than the FAR and the FBR ones.

6 Related Work

In this section, we compare our work in terms of filtering capabilities and non-functional requirements (maintainability, extensibility and flexibility). None of the following related works evaluated their solutions with respect to filtering time and number of subscribers to meet use scenario's requirements.

Comparatively, Teymourian et al. present an architectural model to be used in semantic event-driven systems including an ontology to describe high-level events [Teymourian et al. 2009]. As *Hermes Interpreter*, that architectural model also detects events based on ontology-based and rule-based techniques, but its event filtering mechanism lacks of reactive approach to this problem. Reactivity is useful for unpredictable and highly dynamic scenarios, when subscribers need to specify their events of interest.

EXEHDA-UC provides context management services to mobile and web context-aware applications [Lopes et al. 2013]. Although EXEHDA-UC and *Hermes* share architectural similarities, EXEHDA-UC lacks on ontology reasoning, as long as this is one of the main activities supported by *Hermes Interpreter*. The filtering capabilities proposed by EXEHDA-UC are limited to comparisons operators, e.g. $>$, $<$, $\geqslant$, $\leqslant$, $=$ or $\neq$, whereas *Hermes Interpreter* also includes the properties described in the ontology model. Both studies also differ in terms of the evaluation method: while EXEHDA-UC makes use of the TAM (*Technology Acceptance Model* approach, a performance evaluation was carried to test *Hermes Interpreter*'s capacity to attend users' needs.

Aman and Snekkenes's work include an ontological *schema* for information security and event monitoring based on context filtering to discard redundant and unwanted events [Aman and Snekkenes 2014]. To achieve this, the research uses regular expressions to parse the received context data. Both *Hermes* and Aman and Snekkenes's work deal with semantic context modeling, but the former is is a general purpose system, whereas the latter is specifically for security issues. Besides, the filtering mechanism in *Hermes* is semantically associated with an ontology model and subscribers' needs. In Aman and Snekkenes's work, however, the filtering process occurs by comparison among context data and regular expressions, pre-defined to each monitored device. Those authors also make a case study in a security adaptation scenario to context-aware applications in a health case. However, it does not accomplish response time validation according to the research field requirements.

Balakrishnan and Nayak combines context dissemination with interoperability concerns and dynamic adaptation to tackle with the problem of how the contents of a rapidly evolving context entity is propagated among interested context entities [Balakrishnan and Nayak 2014]. The main objective is to meet heterogeneous entities' demands for customization of context services such as a multimedia traffic routing entity, which adapts itself according to needs of user devices. By making use of networks connectivity and user's resources to customize context data, we consider it as complementary work because *Hermes* deals with context dissemination from the user perspective, i.e., in terms of which context data are of user's interest in a given moment. On the other hand, although Balakrishnan and Nayak also present a comparative evaluation among main events of context dissemination, these are related to the phases of a context dissemination protocol, whereas in *Hermes* they are linked to filtering and inference of context data.

TrailM is a context management system that handles the preferences of dealer users in commercial ubiquitous environments [Martins et al. 2012]. Context data includes users' identification, the products that they want to buy or to sell, and their paths in the ubiquitous environment. Using such information information, TrailM notifies a dealer when another dealer with similar interests is nearby. TrailM's context model lacks of reasoning support because it is solely modelled into relational databases. Regarding the context dissemination issues, TrailM only considers dealers' paths and preferences, while HI offers a dynamic mode to the users parameterize their own events of interest, based on the semantic of the context. Besides that, TrailM's filtering mechanism is static concerning the supported parameters, while HI's is extensible and maintainable according to the changes on the ontology model.

Similar to our work, the E-health system is built upon an ontology-based architecture for the development of context-aware services [Guermah et al. 2013]. E-health aims to notify users about predefined events related to the patients' situations. Although the architecture is extensible to support new rules, it does not provide support to the specification of filters to attend the diversity of subscribers' interests, which is required in highly dynamic scenarios such as the case study described earlier.

Table 3 summarizes the comparison of related work. It describes some design principles which have to be presented in context-aware solutions and other relevant features concerning the research field.

P1 - Independent context model

P2 - Support to multiple context reasoning techniques

P3 - Consistency verification among facts and context model

P4 - Context filtering

P5 - Semantic filtering of context events

P6 - Performance evaluation

Concerning the table data, if the related work does not explicitly mention the principle or if it is not among its goals, it is defined with the '-' marker, in the respective cell. If it is presented, it is marked as 'Y'. If the principle is explicitly described as future work, it is described as 'FW'.

Table 3: Comparison: Design principles and research features.

Research	**Year**	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**
[Teymourian et al. 2009]	2009	Y	Y	Y	Y	Y	-
[Lopes et al. 2013]	2013	-	-	-	Y	-	-
[Aman and Snekkenes 2014]	2014	Y	Y	Y	Y	-	-
[Balakrishnan and Nayak 2014]	2014	-	-	-	Y	-	Y
[Martins et al. 2012]	2012	-	-	-	Y	-	FW
[Guermah et al. 2013]	2013	Y	Y	Y	-	-	-
Hermes Interpreter	**2015**	**Y**	**Y**	**Y**	**Y**	**Y**	**Y**

7 Conclusions and Future Work

This paper focuses on context data's life cycle, especially on the context dissemination phase. The authors claim that solutions for context dissemination must deal with the diversity of the subscribers' interests regardless the application domain such as health [Guermah et al. 2013, Souza et al. 2014], finance [Teymourian et al. 2009], or shopping [Martins et al. 2012].

In order to address that issue, this paper presents a filtering mechanism so that applications describe semantic events dynamically. This mechanism is maintainable and extensible to support changes on an ontological context model as demonstrated in vital signs monitoring scenario.

Results also demonstrate the feasibility of a context filtering layer between the context reasoning phase and the context dissemination phase. In order to better deal with changes in the ontology model and to adapt a filter category to a proper reasoning type, this new layer should keep apart the filters supported, the semantic model and the execution code itself. The comparative test shows the cost of introducing a semantic filtering mechanism depends on the number of subscribers and the complexity of their required filters. Finally, the scalability evaluation shows how many subscribers HI would support to attend the medical reference response time for notifying context in a vital sign monitoring scenario.

As future work, we intend to integrate the filtering mechanism with a knowledge base, to enable both context reasoning and filtering about past context events as well as to investigate the cost of filtering past events.

Hermes Interpreter could also be evolved to provide context reasoning and filtering to others events related to the health monitoring (e.g. elderly fall) using supervised learning techniques or probabilistic logic to support patient's diagnosis.

Acknowledgements

The authors would like to thank the Nursing department of the Clinics Hospital of the Federal University of Goiás for its valuable support.

References

[Aman and Snekkenes 2014] Aman, W., Snekkenes, E.: "Event driven adaptive security in internet of things"; Proceedings of the Eighth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, (2014), 7-15.

[Apache Jena 2014] Apache Software Foundation: Apache Jena Semantic Web Framework. (2014). `http://jena.apache.org/`.

[Balakrishnan and Nayak 2014] Balakrishnan, D., Nayak, A.: "Adaptive Context Dissemination in Heterogeneous Environments"; IEEE Transactions on Mobile Computing, 13, 6 (2014), 1173-1185.

[Bastos et al. 2014] Bastos, A.B., Sene Júnior, I.G., Bulcão-Neto, R.F.: "Modeling and inference based on the semantics of monitoring of human vital signs"; Proceedings of the 20th Brazilian Symposium on Multimedia and the Web, (2014), 13-16.

[Bettini et al. 2010] Bettini, C., Brdiczka O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D.: "A survey of context modelling and reasoning techniques"; Pervasive and Mobile Computing, 6, 2 (2010), 161-180.

[Bulcão Neto and Pimentel 2006] Bulcão Neto, R.F., Pimentel, M.G.C.: "Performance evaluation of inference services for ubiquitous computing"; Proceedings of the 12th Brazilian Symposium on Multimedia and the Web, (2006), 27-34.

[Goldberger et al. 2000] Goldberger, A., Amaral, L., Glass, L., Hausdorff, J., Ivanov, P., Mark, R., Mietus, J., Moody, G., Peng, C., Stanley, H.: "PhysioToolkit and PhysioNet: Components of a new research resource for complex physiologic signals"; Circulation Electronic Pages, 23, 6 (2000), 215-220.

[Guermah et al. 2013] Guermah, H., Fissaa, T., Hafiddi, H., Nassar, M., Kriouile, A.: "Context modeling and reasoning for building context aware services"; Proceedings of the International Conference on Computer Systems and Applications, (2013), 1-7.

[Harris and Seaborne 2013] Harris, S., Seaborne, A.: SPARQL 1.1 Query Language W3C Recommendation. (2013). `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`.

[Hebeler et al. 2009] Hebeler, J., Fisher, M., Blace, R., Perez-Lopez, A.: "Semantic Web Programming"; (2009) John Wiley e Sons, isbn 978-0-470-41801-7.

[Hitzler et al. 2012] Hitzler, P., Krotzsch, M., Parsia, B., Patel-Schneider, P.F., Rudolph, S.: OWL2 Web Ontology Language Primer (Second Edition) W3C Recommendation. (2012). `http://www.w3.org/TR/owl2-primer/`.

[Horrocks et al. 2004] Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web rule language combining OWL and RuleML. (2004). `http://www.w3.org/Submission/SWRL/`.

[Lopes et al. 2013] Lopes, J., Gusmão, 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"; Proceedings of the 19th Brazilian Symposium on Multimedia and the Web, (2013), 43-50.

[Maranhão et al. 2014] Maranhão, G.M., Sene Júnior, I.G., Bulcão-Neto, R.F.: "Anatomy of a semantic context interpreter with real-time events notification support"; Proceedings of the 20th Brazilian Symposium on Multimedia and the Web, (2014), 159-162.

[Martins et al. 2012] Martins, C., Rosa, J., Franco, L., Barbosa, J., Bezerra, E.: "Towards a model to explore business opportunities in trail-aware environments"; Proceedings of the 18th Brazilian Symposium on Multimedia and the Web, (2012), 143-150.

[Matthias et al. 2009] Matthias, G., Boaz, A.M., Dwayne, R.W.: "Improving alarm performance in the medical intensive care unit using delays and clinical context"; Anesthesia e Analgesia, 108, 5 (2009), 1546-1552.

[Perera et al. 2014] Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, D.: "Context aware computing for the Internet of Things: A survey"; IEEE Communications Surveys Tutorials, 16, 1 (2014), 414-454.

[Picalausa and Vansummeren 2011] Picalausa, P., Vansummeren, S.: "What Are Real SPARQL Queries Like?"; Proceedings of the International Workshop on Semantic Web Information Management, (2011), 1-6.

[Sirin et al. 2007] Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: "Pellet: A practical OWL-DL reasoner"; Web Semantics: Science, Services and Agents on the World Wide Web, 5, 2 (2007), 51-53.

[Souza et al. 2014] Souza, A., Mesquitta, F., Lopes, J., Souza, R., Pernas, A., Yamin, A., Geyer, C.: "A situation-aware ubiquitous approach for the evaluation of therapeutic goals in a hospital environment"; Proceedings of the 6th Brazilian Symposium on Ubiquitous and Pervasive Computing, (2014), 921-930.

[Teymourian et al. 2009] Teymourian, K., Streibel, O., Paschke, A., Alnemr, R., Meinel, C.: "Semantic event-driven systems"; Proceedings of the 3rd International Conference on New Technologies, Mobility and Security, (2009), 347-352.

[Veiga et al. 2014] Veiga, E.F., Maranhão, G.M., Bulcão-Neto, R.F.: "Supporting the development of real-time, semantic context-aware applications"; Proceedings of the 20th Brazilian Symposium on Multimedia and the Web, (2014), 1-4.

[Wood et al. 2014] Wood, D., Lanthaler, M., Cyganiak, R.: RDF 1.1 Concepts and Abstract Syntax W3C Recommendation. (2014). `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`.