UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2569

Performance Evaluation of Ubiquitous Inference Services: Reasoning-Related Issues

Renato de Freitas Bulcão Neto
Maria da Graça Campos Pimentel

Nº 272

RELATÓRIOS TÉCNICOS

São Carlos – SP
Abr./2006

Performance evaluation of ubiquitous inference services: reasoning-related issues

Renato de Freitas Bulcão Neto, Maria da Graça Campos Pimentel[1]

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)
Caixa Postal 668 – 13560-970 – São Carlos – SP – Brazil

Abstract. *Considering the current demand for efforts that deal with performance evaluation of ubiquitous computing services, this paper focuses on evaluating a context inference service that can be configured according to applications' reasoning requirements. After detailing a performance evaluation of a context inference service with multiple inference engines over semantic context information, we contribute with a set of general reasoning-related issues that developers must take into account when building ontology-based ubiquitous applications.*

1. Introduction

Context-aware computing is a research theme in ubiquitous computing which investigates problems related to supporting applications in customizing their behaviors based on context information sensed from instrumented environments. A classic definition of context is "any relevant information about the user-application interaction, including the user and the application themselves" [Dey 2001].

As the development of context-aware applications is quite complex and time-consuming [Helal 2005], there is a need for formal context models that facilitate its sharing, reuse and processing, as well as the development of service infrastructures to support applications with respect to context management. The more formal a context model is, the better is the ability for applications to reason about context.

Ontologies have been elected by several authors as the representation formalism for context models (e.g. [Ranganathan et al. 2004] [Chen et al. 2004] [Tan et al. 2005] [Bulcão Neto et al. 2005b]) . A software infrastructure built on top of such context models can then provide applications with services for context sharing, reuse and reasoning.

The literature recognizes the need for research with respect to quantifying and measuring various aspects of ubiquitous computing so as to obtain a deeper understanding of those aspects as well as rich insights into its implications [Satyanarayanan 2005]. Metrics can let us to compare ubiquitous systems and quantify the corresponding benefits and costs of individual algorithms, services, and complete systems. Considering mobile wireless telecommunications, for instance, the performance of mobile agents is evaluated in terms of response time for service provision in 3G networks [Baousis et al. 2005]. Regarding semantic context information, reasoning response times are measured over context knowledge bases in [Tan et al. 2005] and [Gu et al. 2005].

In this paper we report our investigations regarding features pertinent to the performance evaluation of reasoners other than reasoning response time.

To carry out our investigation, we exploit a service infrastructure for context management called *Semantic Context Kernel* (SCK), which is composed of configurable

semantic services for context storage, query and reasoning as well as service discovery [Bulcão Neto et al. 2005b]. Built upon an ontology-based context model, the SCK provides a general vocabulary that lower ontologies can import for particular domains. SCK is appropriate for this study given that its inference service provides applications with customizable types of reasoning about semantic context such as ontology-based and rules-based reasoning.

This paper presents a performance evaluation of the SCK context inference service configured to run with different ontology-based inference engines with distinct capabilities. The context inference service is evaluated against semantic context information from ontologies as well as from a Web-based application. We investigated not only total reasoning response times, but also other timing information related to ontology-based reasoning, such as the time for consistency checking and for merging ontologies that import higher level ones. Such time-based considerations are expected when using the ontology formalism based on Description Logics (DL) [Baader et al. 2003].

From this study, we contribute with a set of general reasoning-related issues that developers must consider when building ontology-based ubiquitous applications such as: what reasoning time is composed of, other reasoning-related timing information that is worth attention, the expressiveness of the ontologies used, and the range of services provided by reasoners.

The remaining of this paper is organized as follows. Section 2 discusses related work. Section 3 presents an overview on the Semantic Web as well as background information on the context inference service and the ontologies we used. Section 4 details the performance evaluation carried out. Section 5 presents concluding remarks.

2. Related work

In this section, we compare our evaluation work with others with respect to reporting the performance evaluation of context inference services.

The GaiaOS middleware is a component-based middleware that handles context information represented in first-order predicates [Ranganathan and Campbell 2003]. GaiaOS also enables applications to reason about uncertain contexts by means of techniques of probabilistic logic, fuzzy logic, and Bayesian networks [Ranganathan et al. 2004]. The CoBrA (Context Broker Architecture) provides an agent-based architecture for acquiring, managing and reasoning about ontology-based context [Chen et al. 2004]. Its main advantage is the ability to handle user's privacy control, which is a relevant issue for the development of context-aware systems. However, neither GaiaOS nor CoBrA report an evaluation that could measure the performance of any services which they are composed of, or the feasibility of exploiting ontologies and reasoning techniques in ubiquitous environments.

The SOCAM (Service-Oriented Context-Aware Middleware) provides applications with independent services for context acquisition, discovery, query and reasoning [Wang et al. 2004]. Reasoning response times are measured using a reasoner based on first-order logic (FOL) over increasing context knowledge bases. Results show that reasoning times increase with the size of context knowledge bases, the complexity of rules, and CPU speed [Gu et al. 2005]. In addition, an ontology-based reasoner and a

FOL-based reasoner are also compared against the same context knowledge bases. Their conclusions include the fact that ontology-based reasoning takes much more time than rules-based reasoning, since the amount of rules to be processed by the former is much greater than in the case of the latter.

Previous work in SOCAM was extended to support distributed event-based context interpretation, which they argue offer better performance in terms of flexibility, scalability and processing time [Tan et al. 2005]. However, their evaluation is not backed up by a performance evaluation study as in our case.

Overall, the results we report in this paper are broader than those found in the literature since we detail what context reasoning time is composed of, we discuss the importance of the expressiveness of the ontologies used, and we exploit the range of services provided by the reasoners used.

3. Background

This section gives an overview of Semantic Web concepts and describes the context inference service and the ontologies we used in our performance evaluation study.

3.1. Semantic Web overview

The Semantic Web is a step further regarding the current Web in which information is given semantics, better enabling computers to process that information [Berners-Lee et al. 2001]. The RDF specification is one attempt to allow information to be described with explicit meaning on the Semantic Web [Klyne and Carroll 2004]. It provides a generic data model that consists of nodes connected by labeled arcs, where nodes represent resources, and arcs represent properties or relations describing resources. A resource combined with a property and the value of that property for that resource is called an RDF statement. Those three individual parts of a statement form the RDF triple model.

The RDFS ontology language conveys the semantics of RDF metadata by means of mechanisms for describing classes of resources, relationships between resources, and restrictions on properties [Brickley and Guha 2004]. RDFS is the least common denominator in terms of Semantic Web ontology languages.

The OWL ontology language builds on top of RDF and RDFS and adds a richer vocabulary for describing properties and classes including, among others, relations between classes (e.g. disjointness), (in)equality of concepts (e.g. equivalent classes), cardinality restrictions (e.g. at least one), richer typing of properties, and characteristics of properties (e.g. transitivity) [Bechhofer et al. 2004]. OWL is divided into three sublanguages, or species, with increasingly expressiveness, as detailed below. Further information about differences between OWL subsets are found in [Bechhofer et al. 2004].

OWL Lite provides primarily a classification hierarchy and simple constraints. In addition, OWL Lite forbids the use of some constructs such as set operations (e.g. union, complement) and disjointness.

OWL DL includes all OWL language constructs, but they can be used only under certain restrictions. For instance, OWL DL requires that every resource has an explicit type: if a resource referenced by X is used where a class is expected, the RDF graph should contain a triple (X, rdf:type, owl:Class).

OWL Full relaxes some of the constraints in OWL DL so as to make available features which may be of use to many knowledge representation systems, but which violate the constraints of DL reasoners.

As different sets of constructs are provided, each sub-language is, therefore, designed for different applications' reasoning requirements. OWL Lite is well-suited to easy implementation and to provide users with a functional starting point in the use of the language. OWL DL is designed for users who want the maximum expressiveness and desirable computational properties for reasoning systems such as completeness[1] and decidability[2]. Finally, OWL Full is intended for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees — this means that it is unlikely that any OWL reasoner will be able to support complete reasoning for every feature of OWL Full [Bechhofer et al. 2004].

OWL DL is so named due to its correspondence with Description Logics (DL), which is a mature knowledge representation technique that represent a subset of first-order logic [Baader et al. 2003]. DL form the formal foundation of OWL balancing expressiveness with decidability. The DL expressivity of OWL ontologies varies according to the OWL constructs being used for ontology developers. A notation for representing DL expressivity is detailed below, so as to indicate, for instance, that an ontology with ALH(D) expressivity means that it includes constructs of attribute logic (AL), properties arranged in hierarchies (H), and datatypes (D).

```
AL (Attribute Logic): conjunction, universal value
                      restriction and limited
                      existential quantification
C  (Complement): AL plus disjunction and full
                 existential quantification
R+ : Transitive property
S  : a shortcut to all ALCR+
H  : Property hierarchy
I  : Inverse property
F  : Functional property or a cardinality
     restriction with 1
O  : Nominal such as enumeration of individuals
     (i.e. instances of classes) or of data values
N  : Unqualified number restrictions such as
     cardinality restrictions greater than 1
D  : Datatypes
```

As DL reasoners support different levels of DL expressivity, the expressiveness criteria should be also taken into account by developers when choosing a reasoner. Currently, Pellet is the only sound and complete DL reasoner that can handle the full expressivity of OWL DL including reasoning about nominals [Sirin et al. 2006]. More details about Pellet and other reasoners are discussed in Section 4.

3.2. The Semantic Context Model

The Semantic Context Model (SeCoM), illustrated in **Figure 1**, represents the basic concepts of actors, location, time, devices, events and activities (dark ovals in the figure), and the relationships between these concepts (in the figure, an arrow indicates when a lower ontology imports concepts from an upper ontology) [Bulcão Neto and Pimentel 2005]. We elected SeCoM to this study because it provides a set of OWL-based generic classes,

[1] All conclusions are guaranteed to be computable.

[2] All computations will finish in finite time.

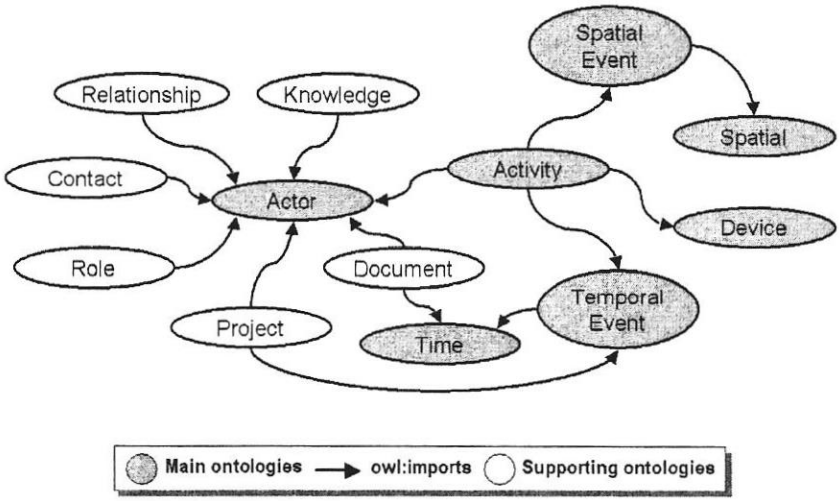

Figure 1. An overview of the Semantic Context Model(SeCoM).

properties, relations, and restrictions that applications can import and/or extend for particular domains. Moreover, SeCoM's ontologies use only the OWL Lite and the OWL DL constructs, which guarantee computation in finite time.

The *Actor* ontology models the profile of entities that can perform actions in a ubiquitous environment such as people, groups and organizations. This ontology is imported by other ontologies that we have built to deal with actors' profiles including social role, social relationship, knowledge, contact information, document and project.

The *Spatial* ontology describes the whereabouts of actors. It models virtual and real-world indoor and outdoor places, containment and spatial relations between places, geographic coordinates, among others. Spatial events are those that have spatial extensions and can be classified as virtual and physical events. Such kind of events are modeled in the *Spatial Event* ontology.

The *Time* ontology represents temporal information in terms of temporal instants and intervals. It models relations between instants and intervals, relations between intervals, and calendar and clock information. Events with temporal extensions are modeled in the *Temporal Event* ontology, and can be classified as instant or interval events.

The *Device* ontology describes devices by means of their hardware and software platforms. The former includes information about storage and battery capacity, multimedia support, and network connectivity, whereas the latter allows to describe operating systems and browsers supported, virtual machines installed, among others.

The *Activity* ontology describes actions that actors do or cause to happen. Activity is modeled as relevant spatiotemporal events that characterizes it including the corresponding actors and devices involved in. Hence, it imports the following SeCoM ontologies: *Actor*, *Spatial Event*, *Temporal Event*, and *Device*. Activities are modeled as impromptu (or informal) and scheduled (i.e. planned in terms of time and place).

3.3. The Semantic Context Kernel

Our investigation required a service infrastructure for context management with an inference service providing applications with customizable types of reasoning about semantic context. We elected the Semantic Context Kernel(SCK) for this study since its architecture provides configurable semantic services as follows [Bulcão Neto et al. 2005b].

Figure 2 depicts the Semantic Context Kernel architecture [Bulcão Neto et al. 2005b]. Context information is provided by *Context Sources*, which may be represented by applications, web services, and physical sensors. *Context Transducers* convert the information captured from context sources into a common semantic representation: the RDF triple model.

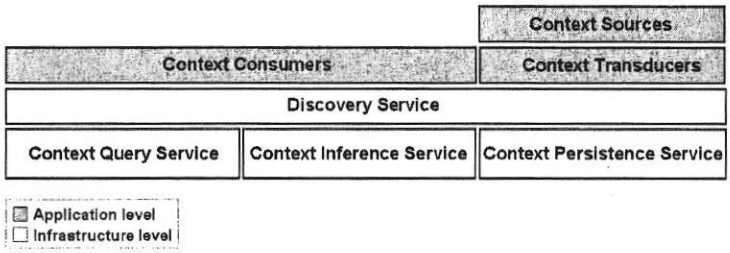

Figure 2. The SCK architecture.

Context Consumers make use of context information stored by context sources so that the former can adapt themselves according to the current situation (e.g. applications). The *Discovery Service* provides context transducers and every service layer with an advertising mechanism so as to allow context consumers to locate services. The *Context Query Service* allows context consumers to query context via declarative languages for RDF models based on simple conjunctive triple patterns. In general, query expressions are represented as a matching of an RDF triple pattern against an input source RDF graph.

The *Context Persistence Service* allows developers to choose the type of persistent storage. It allows context sources to store context information in relational databases or on a *context file*. The latter is an alternative for applications that do not require database functionalities such as consistency of data or transactions. Both context information and ontologies content are stored on the context file. When storing context on databases, ontologies are stored on separate files and only read when necessary.

The *Context Inference Service* provides context consumers with a configurable inference support over context. It allows developers to exploit two types of reasoning which include ontology-based and rules-based reasoning. When using ontology-based reasoning, the inference service supports the RDFS and OWL ontology languages with different levels of expressiveness.

Three types of ontology-based reasoners have been integrated in the inference service:

Transitive reasoner infers the generalization and specialization relations between classes and properties via RDFS constructs for defining subclasses and subproperties.

RDFS reasoner implements a configurable subset of the RDFS entailments such as subclasses, subproperties, as well as domain and range of properties.

OWL reasoners support all RDFS entailments and different subsets of the OWL language. The choose of an appropriate OWL reasoner relies on the OWL constructs involved in the inference process.

All those types of inference engines derive new RDF assertions (or facts) which are entailed from some base RDF together with any optional ontology information and the axioms and rules associated with the reasoner. A context knowledge base is, therefore, the combination of instances of context information (i.e. assertions) and ontologies which these instances are compliant with.

When developers want to exploit high-level context reasoning, they must set the rules-based reasoning feature of the context inference service. Developers should then define their own rules which are stored on a configuration file called *context rule* — typically, one file per application. In this case, the inference service reads ontology facts into memory represented as RDF triples and parse those rules to validate them.

Developers can also choose different chaining of rules such as forward, backward and hybrid. Example 1 illustrates a forward-chaining rule describing that if a graduate student and his supervisor are in the meeting room, then they are attending a meeting.

```
<!--            Example 1            -->
Person(A), Person(B), isLocatedIn(A,C), isLocatedIn(B,C), MeetingRoom(C)
         , hasRole(A,D), hasRole(B,E), Graduate(D), FacultyMember(E)
         , isSupervisorOf(E,D) --> attendingMeeting(A,B)
```

The context inference service infers that a meeting is taking place if not only every predicate shown in the premises (e.g. Person) is represented in the current ontology, but also the individuals are asserted in the input RDF graph.

4. Performance evaluation of the SCK context inference service

This section is organized as follows. First, we describe the hardware and software configuration used in the experiments. Second, we show the SCK inference service using SeCoM ontologies as test data. Third, we evaluate the performance of the SCK inference service running against increasing context knowledge bases of a context-aware web page recommender system. Finally, we give attention to some points about inference making process that we consider important for developers of semantic-enabled ubiquitous applications.

4.1. Experiment configuration

The experiments that we report were performed on a Linux box featuring an Intel$^{(R)}$ Pentium$^{(R)}$ 4 at 2.66 GHz and 1 GiB of main memory. The Semantic Context Kernel infrastructure is implemented in Java using J2SE 1.5.0_02, and the context inference service is implemented on top of the Jena 2.3 inference subsystem [Carroll et al. 2004].

Three types of ontology-based reasoners were used on the experiments: both the *transitive reasoner* and the *OWL reasoner* bundled with Jena 2.3 inference subsystem, and the *Pellet 1.3beta2 reasoner* [Sirin et al. 2006]. In order to prevent lack of memory for the reasoners, considering the dataset used, the Java heap size was set to 64 MiB of main memory (using -Xmx option).

Ontology	OWL species	Expressivity	Triples	Classes	Properties	Instances
Actor	Lite	ALR+IF(D)	72	9	9	0
Time	DL	SHOIF(D)	414	10	47	19
Spatial	DL	SHOIN(D)	181	31	14	8
Temporal Event	DL	SHOIF(D)	432	13	47	19
Contact	DL	SOIF(D)	164	17	21	0
Document	DL	SHOIF(D)	600	27	75	19
Knowledge	Lite	ALR+IF(D)	91	12	13	0
Project	DL	SHOIF(D)	566	54	64	19
Relationship	Lite	ALR+HIF(D)	109	11	19	0
Role	Lite	ALR+IF(D)	104	15	14	0

Table 1. Characterization of SeCoM ontologies using Pellet.

4.2. SeCoM ontologies as test data

The use of ontologies as the representation formalism for context models often arises questions with respect to the supposed overhead to process knowledge. In order to investigate this, we experimented 10 SeCoM ontologies as test data for the context inference service as follows: *Actor*, *Time*, *Spatial*, *Temporal Event*, *Contact*, *Document*, *Knowledge*, *Project*, *Relationship* and *Role*.

For this experiment, we set the context inference service to use only the Pellet reasoner — chosen because it is the only reasoner that guarantees maximum decidability and completeness regarding OWL DL. More specifically, Pellet supports full reasoning on SHOIN(D)[3] DL expressivity. In addition, Pellet performs special services when reasoning about ontologies: it validates their OWL species, informs their DL expressivity, and tries to turn ontologies into OWL DL when they are classified as OWL Full. **Table 1** illustrates the characterization of the SeCoM ontologies used in this experiment. The program that generated the results presented in **Table 1** is included in the Pellet distribution: for every ontology it was obtained its corresponding OWL species (Lite, DL or Full), DL expressivity, and number of triples, classes, properties and individuals (or instances).

Observe that most SeCoM ontologies are encoded in OWL DL, what can suggest that they may require a DL reasoner. The DL expressivity of SeCoM ontologies is another feature that is worth mentioning. For instance, the *Spatial* ontology has SHOIN(D) DL expressivity, what suggests that it may require a complete DL reasoner such as Pellet. Information about the number of triples, classes, properties and individuals of each ontology may also be useful [Tempich and Volz 2003] when evaluating timing information such as illustrated in **Table 2**.

Pellet has another advantage over other DL reasoners: it provides a customizable set of operations available as API so as to get timing information involved in context reasoning. In order to get that information from Pellet, we instrumented the SCK inference service with a program included in the Pellet distribution (*Benchmark.java*).

As input parameters, the inference service gets the Pellet reasoner configuration and the SeCoM ontology file location. For each ontology, we ran the experiment 10 times

[3]Attribute Logic, Complement, Transitive Property, Property hierarchy, Inverse property, Nominals, Unqualified number restrictions, and Datatypes (see Section 3.1)

Ontology	MLT	SVT	ST	CT	MT	RT
Actor	48.5	122.8	12.6	35.8	0	0
Time	103	439.5	32.2	54.3	0	6.5
Spatial	65.8	271.7	83.7	133	0	6.1
Temporal Event	98	474	25	54	2	6.6
Contact	71.2	210.4	22.6	56.8	9.4	0
Document	115.1	610.3	39.7	102.8	23.7	10.5
Knowledge	47.6	137.6	14.6	42.8	5.7	0
Project	111.7	596.3	44.3	100.8	20.8	12
Relationship	46.6	153.9	12.7	37.6	2.6	0
Role	45.8	144.9	19.3	49.6	6.5	0

Table 2. Mean time of reasoning tasks using the SCK inference service and Pellet (in ms).

following the approach used by [Pan 2005]: each parameterized execution of the context inference service was processed after a fresh start. Hence, there was no need to clear both the main memory and the cache memory used by Pellet to speed up reasoning tasks. For each ontology, we collected reasoning-related timing information and measured the corresponding mean time, maximum time, and standard deviation time.

Table 2 presents the mean time of several aspects related to context reasoning — all timing information is in milliseconds:

Model loading time (MLT) is the amount of time spent to load an RDF triple model representing an ontology to reasoner's internal memory — the higher the number of triples of ontologies, the higher the model loading time (see *Document*, *Project*, *Temporal Event* and *Time* ontologies). Specifically for the SeCoM ontologies, the model loading time for the reasoning process is 16.5%, on average.

Species validation time (SVT) is the time interval in which a reasoner validates the OWL sublanguage of ontologies. Similarly to model loading time, species validation time also increases with the number of triples of ontologies. Once developers already know the OWL species and the DL expressivity of their ontologies, we recommend they disable the species validation service because it can save an average of 60.5% over the entire reasoning process.

Satisfiability time (ST) is the time interval in which a reasoner determines if the ontology definition is generally satisfiable (or consistent), i.e., if an ontology does not contain any contradictory facts. We noticed that satisfiability time is higher when the DL expressivity of an ontology increases. For instance, the *Spatial* ontology has the most complex DL expressivity and its corresponding mean satisfiability time is, in some cases, two times greater than ontologies with close DL expressivity (see *Document* and *Project* ontologies). Although it is one of the primary reasoning services, the satisfiability service spends an average of 6.5% of the entire reasoning process.

Classification time (CT) is the time interval in which a reasoner computes the subclass relation between each named class to create the class hierarchy, which is used to answer queries such as getting all or only the direct subclasses. There are optimizations in Pellet so that we can speed up classification time by turning off

nominal reasoning support. However, as some SeCoM ontologies have nominals — e.g. all with SHOIF(D) DL expressivity — we used the Pellet's default configuration. **Table 2** shows that, for our dataset, the more complex the DL expressivity, the higher classification time (see *Spatial*, *Document* and *Project* ontologies). However, this result cannot be generalized since classification time can be minimized due to optimizations coming from other services such as the satisfiability service [Baader et al. 2003]. In our experiment, classification time represents an average of 14% of the entire reasoning process.

Merging time (MT) is the amount of time spent to merge ontologies that import others. As shown in **Figure 1**, SeCoM ontologies merge themselves — *Temporal Event* imports *Time* while *Document* imports *Actor* and *Time*. It is important to observe that the higher the number of triples in the ontologies imported, the higher the merging time (see *Document* and *Project* ontologies). As a result, merging time takes 10 ms on average for SeCoM ontologies, what means an average of 1.5% of the entire reasoning process. SeCoM has been modelled so that applications can instantiate and/or extend it with their own vocabulary, and the observed merging time corroborates the feasibility of such a two-layer ontology-based approach.

Realization time (RT) is the time interval in which the reasoner computes the most specific classes that an individual is an instance of. It should be done after classification since direct types are defined with respect to a class hierarchy. **Table 2** shows that the more individuals in an ontology, the higher the realization time. So, ontologies with no individuals do not spend time for realization (see *Actor*, *Contact*, *Knowledge*, *Relationship*, and *Role* ontologies). In the case of SeCoM, realization time represents an average of 1% of the entire reasoning process.

4.3. Context inference case study: WebMemex

Regarding performance evaluation of a ubiquitous inference service, a question that may arise is how the SCK context inference service behaves with a real context-aware application such as WebMemex [Macedo et al. 2003, Truong and Abowd 2004], a system that allows groups of users to share Web browsing experiences by means of automatic recommendation of pages based on similar contents.

As logged users browse on the Web, WebMemex captures each request from their browsers, extracts metadata and content of web pages, and automatically recommends web pages that are somehow related to the current web page browsed by a user. Figure 3 depicts the user interface of the WebMemex application.

The context-aware WebMemex application

We build a context-aware version of WebMemex in which, when a user wants to recommend a Web page to other users, they explicitly recommend it to a target group, to all users, or to a group of users following a social network criteria (e.g. friendship) [Bulcão Neto et al. 2005a].

Our implementation of WebMemex extends four of the SeCoM ontologies for creating the WebMemex ontology: (i) *Actor* for representing WebMemex users; (ii) *Relationship* for describing the social relationship between users; (iii) *Document* for modeling Web pages; and (iv) *Time* for registering users' history of browsing and recommendation. **Table 3** characterizes the WebMemex ontology. As it imports DL-based ontologies with

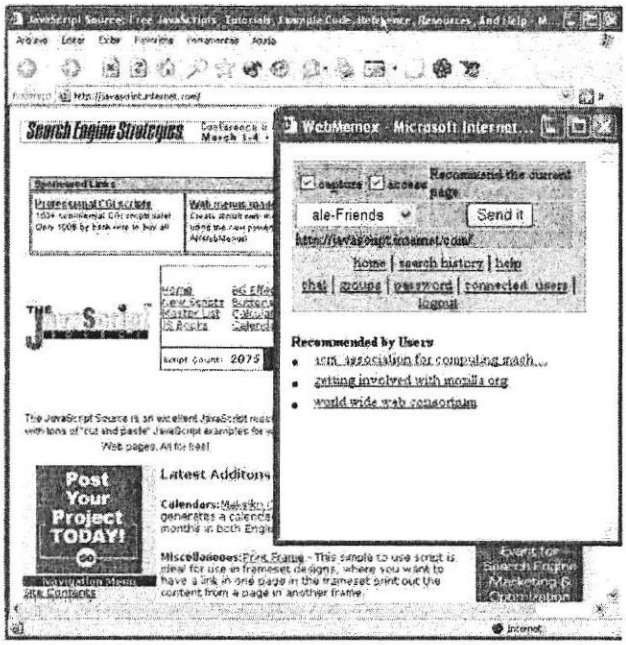

Figure 3. The WebMemex interface (foreground): the combo box corresponds to the groups a user belongs to.

Ontology	OWL species	Expressivity	Triples	Classes	Properties	Instances
WebMemex	DL	SHOIF(D)	720	62	94	19

Table 3. Characterization of the WebMemex application using Pellet.

SHOIF(D) expressivity (i.e. *Document* and *Time*), the WebMemex ontology inherits the same characteristics. As stated earlier, the program that generated those results is included in the Pellet reasoner distribution.

We ran the SCK context inference service against the WebMemex application ontology so as to obtain all the time-based information related to a reasoning process. As input parameters, the inference service gets the Pellet reasoner configuration and the WebMemex ontology file location. We also ran this experiment 10 times always after a fresh start. **Table 4** presents the mean time (in milliseconds) of each reasoning-related task about the WebMemex ontology.

Ontology	MLT	SVT	ST	CT	MT	RT
WebMemex	198.3	609.9	54.7	121.7	26.7	13.4

Table 4. Mean time of reasoning tasks about the WebMemex ontology (in ms).

An important result is that the analysis with respect to timing information relative the SeCoM ontologies also applies. In this case, species validation also poses a high percentage (60% on average), what means that this service should be avoided when using Pellet for reasoning. Merging time confirmed the feasibility of the two-layer ontological approach for context modeling with an average of 2.6% of the entire reasoning process.

WebMemex reasoning regarding semantic context

When creating a group, a user chooses the social network criteria such as *isFriendOf*, *worksWith* or *cooperatesWith*. These relationships are modeled in the *Relationship* ontology as subproperties of *knows*, which is a symmetric property that connects the class *Person* defined in the *Actor* ontology. Hence, if a person X is friend of (or knows) a person Y, then the reciprocal is true.

Every new member in a group is socially related to the creator of that group. Here there is a tradeoff between using OWL-based or transitive reasoners in WebMemex: the former reduces the amount of information stored because it already supports the *owl:SymmetricProperty* construct, whereas the latter offers greater performance, but it requires to store both directions of a symmetric relation. This shows that even a reasoner with less expressivity can infer over ontologies with high DL expressivity. Hence, we add both directions of social network-related properties when adding new members to groups (e.g. 'A isFriendOf B and B isFriendOf A') [Bulcão Neto et al. 2005a].

When a user recommends a Web page to a particular group of users, there is no need for reasoning. On the other hand, if the target group includes *ALL* groups, for instance, the context inference service calls the transitive reasoner to infer over the hierarchy of the superproperty *knows*. The list of people resulting from the reasoning process is then input for the context persistence service to update the corresponding Web logs.

WebMemex and SCK with different reasoners

To the evaluation of the context inference service against with the WebMemex application, we used not only Pellet but also the transitive and the OWL reasoners of the Jena 2.3 distribution. This is a real scenario since WebMemex's reasoning requirements may demand the inference over facts that the transitive reasoner is not able to achieve. As a result, it is an appropriate example in the investigation of how the context inference service behaves with different reasoners over a same knowledge base.

To carry out this experiment, we have used five size-increasing WebMemex's context knowledge bases, which differ at the rate of 1K triples each. For each reasoner, the SCK context inference service gets the reasoner configuration and the context knowledge base it should infer over. In order to achieve higher performance, the OWL reasoner was set to its micro configuration — which means that the reasoner can handle all OWL Lite species, though a few OWL DL constructs with constraints.

For each knowledge base and context inference service configuration, we ran the experiment 10 times towards tuning configuration parameters. As we stated earlier, each parameterized execution was processed after a fresh start.

Figure 4 depicts the mean reasoning response time of the SCK context inference service set to each reasoner over the five sets of knowledge bases (reasoning response times are in milliseconds). Given that the transitive and the OWL (Micro) reasoners do not provide mechanisms that allow developers to get reasoning-related timing information as does Pellet, we only obtained the entire reasoning response time for each reasoner.

Despite of being less expressive than Pellet and OWL Micro, it is noticeable that the transitive reasoner outperforms both reasoners in all cases. As we stated earlier, in order to exploit this lightweight approach with the SCK context inference service, we

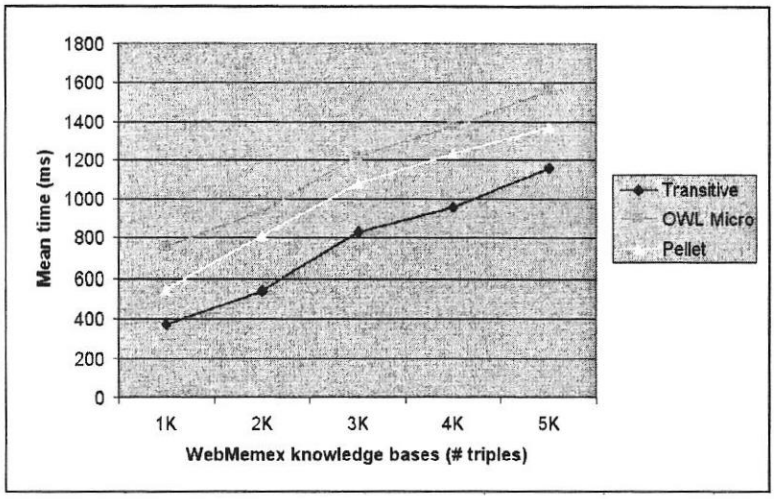

Figure 4. Multiple configurations of the SCK inference service over WebMemex knowledge bases.

had to store both directions of a symmetric property. The feasibility of this approach is sustained as long as WebMemex maintains its current requirement of simple inference over social network-related properties.

From a quantitative point of view, the OWL Micro reasoner is usually 40% slower than the transitive reasoner, and 15% slower than Pellet. It is important to observe that, with the species validation service turned off, Pellet ran an average of 25% faster than the transitive reasoner.

Even though it is possible to change the list of OWL constructs that a developers wants the OWL reasoner to deal with, the OWL reasoner is not recommended for real-world ontologies — Jena developers suggest that the OWL reasoner should be used for early experimental investigations only.

4.4. Discussion

This section presents a summary of the discussion relative to both experiments.

First, we presented how the reasoning time is divided with respect to model loading time, species validation time, satisfiability time, classification time, merging time, and realization time — considering the SeCoM ontologies as test data and the SCK as inference service. This is an important input since developers should fully understand the impact of each phase towards proper application development and ontology reuse, for instance.

Second, we evaluated the reasoning response times of the SCK inference service set to different reasoners over context knowledge bases of a context-aware application. We observed that for all reasoners in our experiment, the ontology-based reasoning response times are somewhat proportional to the size of the knowledge base.

Third, given the amount of time spent for an ontology-based reasoning process, our study also shows the feasibility of an independent module for context reasoning such as the SCK context inference service.

There is, however, a step before the reasoning time starts that we have not considered yet. A reasoner should load and parse the ontology file before it loads the corresponding RDF triple model into memory (i.e. model loading time). The process of loading ontology file has a great impact on reasoning because it has a high variability of loading times both for local and remote ontology files. Regarding reasoning, ontology file loading spends an average of 82% for SeCoM ontologies and 72% for the WebMemex application ontology — for remote ontology loading in both cases. Although this considerable amount of loading time, this can be reduced, for instance, if the inference service implements a fetching mechanism of commonly used ontologies.

Despite the range of services provided by reasoners (e.g. species validation and ontology satisfiability), it is not possible to compare reasoners that make use of different reasoning techniques to support different expressiveness. Considering that they may use different classification algorithms to minimize the number of issued subsumption queries, or even adopt optimizations techniques *so as to support different expressiveness*, a qualitative comparison between the different reasoners used in our performance evaluation is not possible.

5. Concluding remarks

In this paper we identified some reasoning-related issues for which we have discussed the importance for developers of semantic ubiquitous applications. The investigation of how reasoning time is subdivided into important processes has provided valuable information about how ontology-based reasoning time can be improved.

We have also shown that even the reasoner with the simplest capabilities can infer over ontologies with high level of expressiveness. In other words, developers must choose what is the well-suited reasoner according to applications' reasoning requirements.

In practice, the expressivity and efficiency are not the only qualities for choosing a reasoner. Developers may also prefer a reasoner for other reasons, for instance, when it requires few lines of code to use, or it is written in a easily portable programming language.

As future work, we plan to investigate benchmarks for evaluating context inference services. We believe that similar evaluation studies regarding rule-based reasoning in context-aware computing should also be conducted. We also plan to investigate approaches for evaluating distributed inference services, as well as other context-related services such as query and discovery.

Acknowledgments

Renato Bulcão Neto is a PhD candidate supported by the FAPEMA funding agency (n.03/345). Maria da Graça Pimentel is supported by FAPESP and CNPq. We would like to thank to Rodrigo Mello and Daniel Lobato for their valuable comments on this technical report.

References

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Baousis, V., Kyriakakos, M., Hadjiefthymiades, S., and Merakos, L. (2005). Performance evaluation of a mobile agent-based VHE architecture in 3G networks. In *Proceedings of the IEEE International Conference on Pervasive Services (ICPS'05)*, pages 353–360. IEEE Computer Society.

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2004). OWL Web ontology language reference. http://www.w3.org/TR/owl-ref/.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5):35–43.

Brickley, D. and Guha, R. V. (2004). RDF vocabulary description language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/.

Bulcão Neto, R. F., Macedo, A. A., Camacho-Guerrero, J. A., and Pimentel, M. G. C. (2005a). Configurable semantic services leveraging applications context-aware. In *Proceedings of the 11th Brazilian Symposium on Multimedia and Web Systems (WebMedia'05)*, pages 1–9. ACM Press.

Bulcão Neto, R. F. and Pimentel, M. G. C. (2005). Toward a domain-independent semantic model for context-aware computing. In *Proceedings of the 3rd Latin American Web Congress (LA-Web'05)*, pages 61–70. IEEE Computer Society.

Bulcão Neto, R. F., Teixeira, C. A. C., and Pimentel, M. G. C. (2005b). A Semantic Web-based infrastructure for supporting context-aware applications. In *Proceedings of the IFIP International Conference on Embedded and Ubiquitous Computing (EUC'05)*, volume 3824 of *Lecture Notes on Computer Science*, pages 900–909.

Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., and Wilkinson, K. (2004). Jena: implementing the Semantic Web recommendations. In *Proceedings of the 13th International World Wide Web Conference (WWW'04)*, pages 74–83.

Chen, H., Finin, T., Joshi, A., Perich, F., Chakraborty, D., and Kagal, L. (2004). Intelligent Agents Meet the Semantic Web in Smart Spaces. *IEEE Internet Computing*, 8(6):69–79.

Dey, A. K. (2001). Understanding and using context. *Personal and Ubiquitous Computing Journal*, 5(1):4–7.

Gu, T., Pung, H. K., and Zhang, D. Q. (2005). A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1):1–18.

Helal, S. (2005). Programming pervasive spaces. *IEEE Pervasive Computing*, 4(1):84–87.

Klyne, G. and Carroll, J. J. (2004). Resource Description Framework (RDF): concepts and abstract syntax. http://www.w3.org/TR/rdf-concepts/.

Macedo, A. A., Truong, K. N., Camacho-Guerrero, J. A., and Pimentel, M. G. C. (2003). Automatically sharing web experiences through a hyperdocument recommender system. In *Proceedings of the ACM Conference on Hypertext and Hypermedia*, pages 48–56.

Pan, Z. (2005). Benchmarking DL reasoners using realistic ontologies. In *Proceedings of the International Workshop on OWL: Experiences and Directions (OED'05)*. 7 pages.

Ranganathan, A., Al-Muhtadi, J., and Campbell, R. H. (2004). Reasoning about uncertain contexts in pervasive computing environments. *IEEE Pervasive Computing*, 3(2):62–70.

Ranganathan, A. and Campbell, R. H. (2003). An infrastructure for context-awareness based on first order logic. *Personal and Ubiquitous Computing Journal*, 7(6):353–364.

Satyanarayanan, M. (2005). Metrics and benchmarks for pervasive computing. *IEEE Pervasive Computing*, 4(3):4–6.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., and Katz, Y. (2006). Pellet: A practical OWL-DL reasoner. Submitted for publication to Journal of Web Semantics.

Tan, J. G., Zhang, D., Wang, X., and Cheng, H. S. (2005). Enhancing semantic spaces with event-driven context interpretation. In *Proceedings of the Third International Conference on Pervasive Computing*, pages 80–97.

Tempich, C. and Volz, R. (2003). Towards a benchmark for semantic web reasoners - an analysis of the DAML ontology library. In *Proceedings of the International Workshop on Evaluation of Ontology-based Tools (EON'03)*. 12 pages.

Truong, K. N. and Abowd, G. D. (2004). INCA: A software infrastructure to facilitate the construction and evolution of ubiquitous capture & access applications. In *Proceedings of the International Conference on Pervasive Computing*, pages 140–157.

Wang, X., Dong, J. S., Chin, C., Hettiarachchi, S., and Zhang, D. Q. (2004). Semantic Space: An infrastructure for smart spaces. *IEEE Pervasive Computing*, 3(3):32–39.