

Error-Tolerant RDF Subgraph Matching for Adaptive Presentation of Linked Data on Mobile

Luca Costabello

▶ **To cite this version:**

Luca Costabello. Error-Tolerant RDF Subgraph Matching for Adaptive Presentation of Linked Data on Mobile. European Semantic Web Conference, ESWC 2014, May 2014, Anissaras, Greece. 10.1007/978-3-319-07443-6_4 . hal-01188204

HAL Id: hal-01188204
https://inria.hal.science/hal-01188204

Submitted on 28 Aug 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Error-Tolerant RDF Subgraph Matching for Adaptive Presentation of Linked Data on Mobile

Luca Costabello

Inria, Sophia Antipolis, France
luca.costabello@inria.fr

Abstract. We present PRISSMA, a context-aware presentation layer for Linked Data. PRISSMA extends the Fresnel vocabulary with the notion of mobile context. Besides, it includes an algorithm that determines whether the sensed context is compatible with some context declarations. The algorithm finds optimal error-tolerant subgraph isomorphisms between RDF graphs using the notion of graph edit distance and is sublinear in the number of context declarations in the system.

Keywords: #eswc2014Costabello

1 Introduction

Semantic Web mobile applications might not have built-in assumptions about the schemas of the data they consume, as data models could be unknown a-priori, and provided by heterogeneous sources: users might consume any type of data, as long as it is relevant to their context [17]. To improve the effectiveness of Linked Data consumption, content adaptation must be adopted, i.e. the process of selecting, generating, or modifying content units in response to a requested URI[1]. Essential in the mobile Web, such process is driven by the multifaceted notion of *client context* [9]. Content adaptation reduces the fan-out of RDF entities, and provides coherent information by using context as a dynamic filter. Furthermore, it orders, groups, and formats triples, thus creating "optimized" content units ready for user consumption.

This paper addresses the question of *how to enable context-aware adaptation for Linked Data consumption*. We split up the problem in two sub-questions: i) *how to model context for Linked Data presentation* and ii) *how to deal with context imprecision to select proper presentation metadata at runtime*. Modelling context-aware presentation concepts for Linked Data needs a proper ontology that fills the gap between traditional context ontologies and the Web of Data (e.g. support for future extensions, adoption of a lightweight vocabulary instead of a vast, monolithic context ontology, etc). The selection of presentation metadata is complicated by a series of constraints: first, the intrinsic imprecision of context data determines the need for an error-tolerant strategy that takes into account

[1] See *Adaptation* definition: http://www.w3.org/TR/di-gloss/#sec-glossary

possible discrepancies between context descriptions and actual context. Second, this error-tolerant mechanism must support heterogeneous context dimensions (e.g. location, time, strings). Third, since the procedure must run on the client-side - to avoid disclosing sensitive context data - we must design a mobile-friendly algorithm, with acceptable time and space complexity. Finally, the adopted strategy must support runtime updates of RDF graphs, as context descriptions might be fetched from remote repositories and added to the selection process at runtime, and the sensed context may change at any time.
Our contribution is PRISSMA, a context-aware presentation framework for Linked Data. PRISSMA answers our two-fold research question with the following contributions: i) a vocabulary for describing context conditions, compatible with Fresnel [19], and ii) an error-tolerant subgraph matching algorithm that determines whether the sensed context is compatible with context declarations.

In Section 2 we present the state-of-the-art presentation-level frameworks for the Semantic Web, along with an overview of error-tolerant matching techniques. Section 3 describes the PRISSMA vocabulary and explains the error-tolerant presentation metadata selection algorithm. The algorithm experimental evaluation results are described in Section 4.

2 Related Work

As shown in Table 1a, none of the existing presentation frameworks for Linked Data [1, 4, 8, 11, 12, 15, 20] completely supports context awareness. One of these works is Fresnel [19], a rendering engine for RDF. Fresnel is built on the assumption that data and its related schema do not carry sufficient information for representing triples, hence it provides additional presentation-level knowledge. Developers create Fresnel declarations for RDF instances or classes that will be displayed by their applications using the Fresnel vocabulary, an ontology built on the separation between *data selection* and *formatting*. Data selection and filtering is implemented by Fresnel *Lenses*, while *Formats* define how to present data.

Castano et al. [3] provide an overview of matching techniques for RDF instances; most of these works stem from ontology matching strategies [10]. Figure 1b compares error-tolerant works closer to our requirements: iSPARQL [16] is designed for error-tolerant matching, but it neither supports heterogeneous dimensions (such as location), nor is it designed for computationally-constrained mobile platforms. The Silk framework [24] includes geographical and time distances but such metrics do not consider data imprecision. Furthermore, Silk is not designed to run on mobile devices. RDF semantics states that two RDF graphs are semantically equivalent if they entail one another[2], and, as underlined by Carrol [2], the important concept for entailment between RDF graphs is *subgraph isomorphism*, known to be NP-complete. Subgraph isomorphism is at the heart of a recent pattern matching engine for SPARQL by Zou et al. [25]. Unfortunately, the authors do not provide an *error-tolerant* version of their algorithm. It has been proved [6] that finding the optimal error-tolerant subgraph isomorphism

[2] `http://www.w3.org/TR/rdf-concepts/`

	Haystack Ozone[15]	Noadster[22]	Surrogates[12]	Fresnel[19]	Xenon[20]	Tal4Rdf[4]	LESS[1]	Hide the Stack[8]	LDVM[11]	**PRISSMA**
Declarative approach	•		•	•	•	•	•	•	•	•
Domain Independence	•	•	•	•	•	•			•	•
Standard Languages	•		•	•	•			•	•	•
Context Awareness			○							•
Automatic stylesheets			○							
Evaluation								•		•
Distribution							•			○
Multimodality			•	•		○	○			•

(a)

	iSPARQL[16]	Silk[24]	Zou et al.[25]	Messmer and Bunke[18]	**PRISSMA**
RDF-specific	•	•	•		•
Data Heterogeneity		○		○	•
Client-side Execution				○	•
Incremental index updates	•			•	•
Selective matching cache				○	○

(b)

Table 1: A comparison of presentation layers for the Semantic Web (a) and of error-tolerant matching techniques for RDF (b). Full support is identified by •, partial support by ○, no support by the empty cell.

between two graphs can be reduced to the computation of *graph edit distance*: the idea is that differences between graphs can be modelled in terms of operations to apply to graphs, such as adding a node or modifying an arc. Graph edit distance provides the required flexibility for building an error-tolerant subgraph matching algorithm, and supports customized and heterogeneous cost functions (comparing contexts means dealing with data such as location, time, string literals, URIs). Nevertheless, computational complexity is exponential in the number of graph nodes, since graph edit distance algorithms assume that every node can be mapped on every node of another graph. Although context descriptions are rather small graphs, computing graph edit distance remains a computationally expensive task, in particular on mobile devices. Traditional approaches to compute graph edit distance between an input graph and a set of reference graphs apply a pairwise comparison, but such methods do not scale well and badly perform with runtime updates [13]. Messmer and Bunke [18] adopt a different strategy: they fragment directed, labelled graphs into smaller subgraphs, and store them into a single data structure, to avoid duplicates. Given an input graph, an online search algorithm searches for the error-tolerant subgraph isomorphisms with the lowest edit costs.

3 Prism Selection Algorithm

We extend the Fresnel presentation-level ontology with context awareness. The PRISSMA vocabulary[3] (Figure 1) broadens the semantics of `fresnel:Purpose` to delegate the selection of Lenses and Formats to a broader and more expressive

[3] `http://ns.inria.fr/prissma`

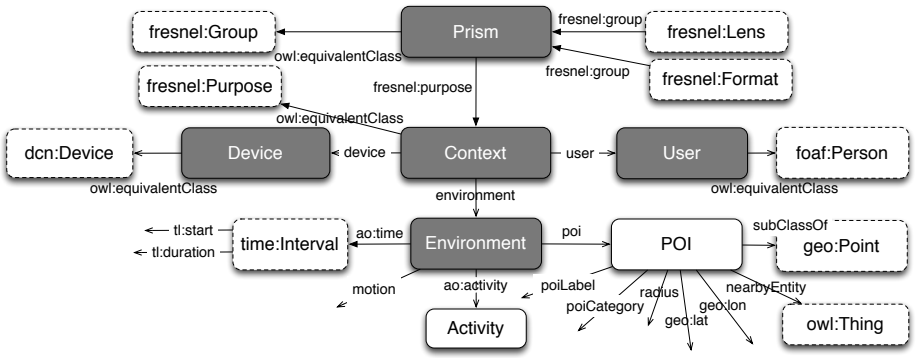


Fig. 1: The PRISSMA vocabulary

definition of mobile context, modelled by the `prissma:Context` class. PRISSMA is not meant to provide yet another mobile context model, as that is out of the scope of our work. Instead, we reuse and combine well-known vocabularies: we are based on the widely-accepted formalization of context proposed by Dey [9] and we extend the W3C Model-Based User Interface Incubator Group[4] proposal, that models mobile context as the sum of the *User* model, the *Device* features, and the *Environment* in which the action is performed (we have described the PRISSMA vocabulary in further detail in [7]). To wrap up each context-aware presentation-level unit of information, the concept of *Prism* is introduced (a Prism is `owl:equivalentClass` to a `fresnel:Group`):

Definition 1 (Prism). *A Prism P is an RDF graph that describes the context conditions under which a given RDF presentation must be activated.*

Fig. 2 shows the sample Prism `:museumPrism`. The Prism styles `dbpedia:Museum` instances when requested by art-loving users walking in Paris. Fresnel lenses and formats (8-27) are coupled to the PRISSMA context description of lines 29-43.

Before rendering an RDF resource with Fresnel, PRISSMA-equipped applications search the available Prisms and select the better match for the context in which the desired resource is accessed, thus our second research question: *how to select the proper context description at runtime?* The most relevant challenge of this task is the imprecise and incomplete nature of context data, that complicates the matching procedure between declared and sensed contexts, and requires an error-tolerant approach. Context data is riddled by the following issues:

Ambiguity. Some RDF Entities and literals used in PRISSMA declarations might not match with the actual context entities. Nevertheless, in some cases entities and literals might be *similar*.

Incompleteness. The authors of PRISSMA context declarations might omit or forget certain properties, when describing a context. Nevertheless, in certain

[4] `http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui/`

```
# Styles a Museum when walking in Paris
:museumPrism a prissma:Prism ;
            a fresnel:Group ;
 fresnel:purpose :walkingInParisArtLover ;
 fresnel:stylesheetLink <style.css>.

# Fresnel presentation-level triples
:museumlens a fresnel:Lens;
   fresnel:group :museumPrism;
   fresnel:classLensDomain dbpedia:Museum;
   fresnel:showProperties (
                    dbpprop:location
                    dbpprop:publictransit
                    ex:telephone
                    ex:openingHours
                    ex:ticketPrice ) .

:addressFormat a fresnel:Format ;
   fresnel:group :museumPrism ;
   fresnel:propertyFormatDomain
                    dbpprop:location ;
   fresnel:label "Address" ;
   fresnel:labelStyle
       "css-class1"^^fresnel:styleClass ;
   fresnel:valueStyle
       "css-class2"^^fresnel:styleClass .

# [...]

# PRISSMA context description
:walkingInParisArtLover a prissma:Context ;
   prissma:user :artLover ;
   prissma:environment :parisWalking .

:artLover a prissma:User ;
   foaf:interest "art".

:parisWalking a prissma:Environment ;
   prissma:poi :paris ;
   prissma:motion "walking" .

:paris geo:lat "48.8567" ;
   geo:long "2.3508" ;
   prissma:radius "5000" .
```

Fig. 2: A sample Prism (prefixes are omitted)

cases the context graph, although topologically different, should still be considered as a valid candidate by the selection algorithm.

Sensor noise. Onboard sensors might provide erroneous information that will be part of the actual context graph [14]. This is a well-known problem when determining geographic location (e.g. weak GPS signal, indoor location, etc).

To overcome such issues, we extended and adapted to RDF the Messmer and Bunke error-tolerant algorithm for finding *optimal* subgraph isomorphisms for labelled, directed graphs [18].

3.1 Definitions

Before describing the adapted algorithm, we remind some useful definitions provided in Messmer [18], adjusting them to our scenario:

Definition 2 (RDF Graph). *An RDF graph is a set of RDF triples* $G = \{(s_1, p_1, o_1) \dots (s_n, p_n, o_n)\} = (V, E)$ *where* s_t *is the subject,* p_t *the property and* o_t *the object of each triple* t*.* V *is the set of labelled vertices and contains the elements* s_t *and* o_t*, that are entities or literals.* E *is the set of directed edges and contains all the triple properties* p_t*.*

Definition 3 (Graph Edit Operation). *Given an RDF graph* $G = (V, E)$*, a graph edit operation* $\delta(G)$ *is one of the following:*

- $v \to v',\ v \in V, v' \in V$ *(substituting an RDF entity or literal)*
- $e \to e',\ e \in E$ *(substituting an RDF property)*
- $v \to \epsilon,\ v \in V$ *(deleting an RDF instance or literal)*
- $e \to \epsilon,\ e \in E$ *(deleting an RDF property)*
- $\epsilon \to e$ *(adding an RDF property between existing nodes)*

where ϵ is an empty RDF entity, literal, or property.

These five edit operations are sufficient to transform any graph G into a subgraph of any graph G'. Note that the algorithm searches for subgraph isomorphisms from a model graph to the input graph, hence there is no need to consider exterior RDF instances or literals in the input graph, i.e. there is no need for a $\epsilon \rightarrow v,\ v \in V$ operation.

Definition 4 (Edited Graph). *Given an RDF graph G and a sequence $\Delta = (\delta_1, \delta_2, \ldots, \delta_n)$ of edit operations, the edited graph $\Delta(G) = (\Delta(V), \Delta(E))$ is the graph $\Delta(G) = \delta_n(\ldots \delta_1(G))$.*

Definition 5 (Error-Tolerant RDF Subgraph Isomorphism). *Given two RDF graphs $G = (V, E)$ and $G' = (V', E')$, an error-tolerant RDF subgraph isomorphism f from G to G' is a two-tuple $f = (\Delta, f_\Delta)$ where:*

- *Δ is a sequence of graph edit operations that transforms G in $\Delta(G)$.*
- *f_Δ is an injective function $f_\Delta : \Delta(V) \rightarrow V'$ such that $\exists$ a graph isomorphism[5] from $\Delta(G)$ to a subgraph $S \subseteq G'$.*

We now introduce the definition of *cost of error-tolerant subgraph isomorphism*, preceded by the *cost* of an edit operation:

Definition 6 (Cost of Edit Operation). *Given an edit operation δ_i, the cost of δ_i is a value $C(\delta_i) \in [0, 1]$.*

The cost $C(\delta_i)$ of an edit operation δ_i varies according to the type of edit operation (e.g. instance substitution, property deletion, etc.) and the nature of the involved RDF element. We cover in more details $C(\delta_i)$ in Section 3.4.

Definition 7 (Cost of Error-Tolerant RDF Subgraph Isomorphism). *Given an error-tolerant RDF subgraph isomorphism $f = (\Delta, f_\Delta)$, its cost $C(f)$ is defined as the normalized cost of the sequence of edit operations $\Delta = (\delta_1, \ldots \delta_n)$, $C(f) = \frac{C(\Delta)}{n} = \frac{\sum_{i=1}^{n} C(\delta_i)}{n}$.*

The cost of error-tolerant subgraph isomorphism described in Definition 7 adopts the arithmetic mean to normalize the cost of the sequence of edit operations. Other strategies might be adopted, such as using a weighted mean or the maximum cost in the sequence.

It is evident that there might exist multiple sequences Δ of edit operations from graph G to graph G', each with a different cost: we are interested in finding the *optimal error-tolerant subgraph isomorphism*, i.e. the error-tolerant subgraph isomorphism with the least expensive sequence of edit operations. In other words, we want to find the minimum amount of distortion needed to transform a Prism into the actual mobile context, thus computing their graph edit distance [21]:

[5] Definition of graph isomorphism provided in [18].

Definition 8 (Optimal Error-Tolerant RDF Subgraph Isomorphism). *Given the set of error-tolerant subgraph isomorphisms $F = f_1 \dots f_n$ between two graphs, the optimal error-tolerant subgraph isomorphism f_{opt} is the element of F with cost $C(f_{opt}) = \min_{f_i \in F} C(f_i)$.*

3.2 Decomposition

The context-related triples included in each Prism are split in subgraphs and saved in a structure called *decomposition*, a recursive partitioning of a set of RDF models (Prisms). The decomposition algorithm works on the set of Prisms pre-loaded by the PRISSMA-equipped mobile application. The idea is building the decomposition by detecting and merging common subgraphs: in the decomposition, subgraphs duplicated in different Prisms are collapsed and represented only once, thus providing a compact representation of possible contexts. As remarked by Messmer and Bunke, there exists more than one decomposition for a set of graphs: the adopted strategy does not provide an optimal decomposition (e.g. in the number of elements), but it is computationally inexpensive compared to other strategies [18].
The elements of a decomposition are tuples that include graph patterns sharing the same topology and whose RDF elements have the same classes. Among the decomposition elements, some consist in groups of non-decomposable, atomic graph patterns called *context units*:

Definition 9 (Context Unit). *A context unit is an RDF graph $U = (V_U, E_U)$ representing atomic context information. A context unit U consists in either a single class, or a single RDF entity, or a single literal, or in a graph that describes an atomic context information.*

In the original proposition, Messmer and Bunke deal with graphs with a limited range of discrete values, thus they decompose graphs up to single nodes. We compare more complex structures, hence the need to preserve context units (e.g. we cannot split latitude, longitude, and radius when comparing locations). Thus, different types of context units have been defined, according to the type of context information: *Class* context units consist in core PRISSMA classes (e.g. `prissma:Context`, `prissma:User`, `prissma:Environment`, and `prissma:Device`). *Class* context units are created by a preliminary step, where instances of core PRISSMA classes are substituted by their class. This operation decreases the size of the decomposition without losing information, since the URIs of such core instances are not important for matching purposes. *Entity* context units are RDF entities, whose classes are not among PRISSMA core classes. Entity context units may be blank nodes. *Geo* context units represent geographic locations, while *Time* elements include temporal information. Both Geo and Time context units may be blank nodes. *String* and *Numeric* context units are associated to string and numeric literals. The decomposition is formally defined as follows:

Definition 10 (Decomposition). *Given a set of Prisms $P = \{P_1, \dots, P_n\}$, the decomposition $D(P)$ is a set of 4-tuple (G, G', G'', E) where:*

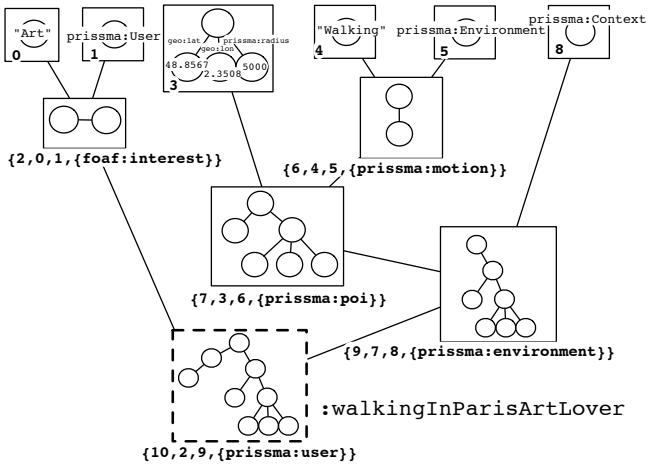


Fig. 3: A decomposition of context data of `:museumPrism` showed in Figure 1.

1. *G, G', G'' are RDF graphs, with $G', G'' \subset G$*
2. *E is a set of RDF properties such that $G = G^{'} \cup_E G''$, where $\cup_E$ is the union of G' and G'' properties.*
3. *for each P_i there exists a 4-tuple $(P_i, G', G'', E) \in D(P)$*
4. *for each 4-tuple (G, G', G'', E) there exists no other 4-tuple $(G_1, G'_1, G''_1, E) \in D(P)$ with $G = G_1$*
5. *for each 4-tuple $(G, G', G'', E) \in D(P)$*
   (a) *if G' is not a context unit, there exists a 4-tuple $(G_1, G^{'}_1, G^{''}_1, E) \in D(P)$ such that $G' = G_1$*
   (b) *if G'' is not a context unit, there exists a 4-tuple $(G_2, G^{'}_2, G^{''}_2, E) \in D(P)$ such that $G'' = G_2$*
   (c) *if G' is a context unit, there exists no 4-tuple $(G_3, G^{'}_3, G^{''}_3, E) \in D(P)$ such that $G' = G_3$*
   (d) *if G'' is a context unit, there exists no 4-tuple $(G_4, G^{'}_4, G^{''}_4, E) \in D(P)$ such that $G'' = G_4$*

Figure 3 shows the decomposition of `:walkingInParisArtLover`, the Prism in Figure 2. Uppermost elements are context units: 0 and 4 are "String" context units, 1, 5, and 8 are "Class" context units, and 3 is a "Geo" context unit. Each decomposition elements contains the IDs of the ancestors (G', G'') and the set of connecting RDF properties E. Element 10 represents the complete `prissma:Context` graph.

The recursive function *decompose*() (Algorithm 1) is executed on each Prism G in the decomposition D. The function searches in the decomposition for S_{max}, the biggest subgraph of G (lines 3-5): the goal is to determine if there exists a

Alg. 1: decompose(G,D)

```
   Data: a Prism G, the decomposition D
   Result: The updated decomposition D
1  S_max = ∅
2  if G not context unit then
3      foreach (G_i, G'_i, G''_i, E_i) do
4          if G_i is a subgraph of G and S_max smaller than G_i then
5              S_max = G_i
6      if S_max is isomorphic to G then
7          exit
8      if (S_max = ∅) then
9          choose subgraph S_max, priority to PRISSMA properties
10         decompose(S_max)
11     decompose(G − S_max)
12     add (G, S_max, G − S_max, E) to D
```

graph pattern in common with the decomposition[6]. If S_{max} is isomorphic[6] to G, then G is already represented in D and the algorithm stops (lines 6-7). If no subgraph is found, and G can be further decomposed (i.e. it is not a context unit), the procedure chooses S_{max} (line 9) and recursively decomposes it (line 10). The choice of S_{max} is determined by a list of ordered RDF properties, with a priority for PRISSMA background ontology core properties (e.g. `prissma:user`, `prissma:environment`, `prissma:device`, etc). This enhances the chances of merging decomposition elements, thus resulting in a more compact structure. The procedure is invoked recursively on $G - S_{max}$, the part of G not yet decomposed (line 11). Finally, $(G, S_{max}, G - S_{max}, E)$ is added to D.

3.3 Search Algorithm

Every significant context change[7] detected by the device triggers the search for Prisms that fit the updated context requirements. PRISSMA carries out this operation with an adapted version of Messmer and Bunke online search algorithm [18]. The algorithm detects *optimal* error-tolerant subgraph isomorphisms between the graph of the sensed context and the Prisms stored in the decomposition. The algorithm first computes edit operations between context units in the decomposition D and context units of the input graph. Second, it combines such edit operations to obtain optimal error-tolerant subgraph isomorphisms for larger patterns, up to complete Prisms (Algorithm 3). To avoid combinatorial explosion, the concatenation of error-tolerant subgraph isomorphisms includes only the cheapest error-tolerant graph isomorphisms: this guarantees to find optimal error-tolerant subgraph isomorphisms.

[6] The graph isomorphism and the exact subgraph isomorphism operations are delegated to off-the-shelf algorithms, such as [23] whose description is out of the scope of this work.

[7] The notion of *significant* context change is scenario-dependent, and it is not investigated in this paper.

Algorithm 2 presents the *search* procedure: first, it finds the error-tolerant subgraph isomorphisms from each context unit S of the decomposition D to the input context graph G_I and stores them in the list $candidates(S)$ (lines 1-2). This operation is performed by the *context_unit_matching()* function. From line 3 to 12 such error-tolerant subgraph isomorphisms are concatenated to find error-tolerant subgraph isomorphisms for larger graphs, up to Prisms:

Alg. 2: $search(G_I, D)$

```
   Data: a Decomposition D, a context graph G_I
   Result: the result set R containing selectable Prisms
1  foreach S context unit in D do
2      candidates(S) = context_unit_matching(S, G_I)
3  while choose S_1 | ∃ f_1 ∈ candidates(S_1) with C(f_1) minimal in D and C(f_1) ≤ T do
4      winners(S_1) = winners(S_1) ∪ {f_1}
5      candidates(S_1) = candidates(S_1) − {f_1}
6      if S_1 is a Prism then
7          R = R ∪ {S_1}
8      foreach (S, S_1, S_2, E) ∈ D || (S, S_2, S_1, E) ∈ D do
9          foreach f_2 ∈ winners(S_2) do
10             f = combine(S_1, S_2, E, G_I, f_1, f_2)
11             if f ≠ ∅ then
12                 candidates(S) = candidates(S) ∪ {f}
13 return R
```

Alg. 3: $combine()$

```
   Data: S_1, S_2, E, G_I, f_1 =
         (Δ_1, f_Δ1), f_2 =
         (Δ_2, f_Δ2), Δ_1 =
         (Δ_V1, Δ_E1), Δ_2 = (Δ_V2, Δ_E2)
   Result: f
1  if f_Δ1(Δ_V1) ∩ f_Δ2(Δ_V2) ≠ ∅ then
2      exit
3  foreach v ∈ (Δ_V1 ∪ Δ_V2) do
4      if v ∈ V_Δ1 then
5          f_Δ(v) = f_Δ1(v)
6      else if v ∈ V_Δ2 then
7          f_Δ(v) = f_Δ2(v)
8  Δ = Δ_1 + Δ_2 + Δ_E
9  return f = (Δ, f_Δ)
```

Alg. 4: $context_unit_matching(U, G_I)$

```
   Data: context unit U, input context graph
         G_I = (V_I, E_I)
   Result: the list of error-tolerant subgraph
           isomorphisms F
1  F = ∅
2  foreach context unit U_I ∈ G_I do
3      generate an error-tolerant subgraph
       isomorphism f between U and U_I
4      F = F ∪ {f}
5  f' = (Δ', f'_Δ) with Δ' = (v → ϵ) and
   f'_Δ = ∅
6  F = F ∪ {f'}
7  return F
```

in line 3 we select the subgraph S_1 whose error-tolerant subgraph isomorphism f_1 has the minimum cost in D. Note that $C(f_1)$ must be lower than a threshold $T \in [0, 1]$. The error-tolerant subgraph isomorphism f_1 is removed from $candidates(S_1)$ in line 5 and added to the list $winners(S_1)$, the container of error-tolerant subgraph isomorphism chosen to be combined. If S_1 is a Prism, the algorithm has found a result (lines 6-7). Otherwise, we generate error-tolerant subgraph isomorphisms for each subgraph S having S_1 as ancestor (lines 8-12). Such generation is done with the *combine()* function that concatenates f_1 to each $f_2 \in winners(S_2)$, where S_2 is the other ancestor of S. If a combination is feasible, the resulting error-tolerant subgraph isomorphism is added to $candidates(S)$ (line 12).

Algorithm 3 details the *combine()* procedure: first (line 1), the function tests if f_1 and f_2 do not contain mappings to the same node in G_I (this is necessary because

subgraph isomorphisms are injective functions [18]). If this condition is satisfied, an error-tolerant subgraph isomorphism is constructed as a concatenation of the edit operations of f_1 and f_2 and of the edit operations on the edge between S_1 and S_2, Δ_E (line 8). Mappings are chosen among the mappings of f_1 and f_2 (lines 3-7).

We now discuss in further detail *context_unit_matching()*, the function used by the search algorithm to compute error-tolerant subgraph isomorphisms for context units (Algorithm 4). Given a context unit U and an input context graph G_I, the procedure finds the edit operations from U to each context unit of G_I (line 2-3) and stores them as error-tolerant subgraph isomorphisms. Moreover, the deletion of U is considered (line 5).

Worst case computational complexity analysis shows that the complexity of the search procedure varies from $O(Lm^n n^2)$ when Prisms in the decomposition are completely different, to $O(m^n n^2)$ when Prisms are highly similar (L is the number of Prisms in the decompositions, m the number of vertices of the input context graph and n the number of vertices of each Prism included in a decomposition D made of Prisms with same number of nodes). Hence, the search algorithm is sublinear in the number L of Prisms included in the decomposition (for a detailed theoretical analysis of the computational complexity of the search algorithm, see [18]). This is an important property of the algorithm, since the number of Prisms in the system can be potentially high and unknown a priori.

3.4 Cost of Edit Operations

Each graph edit operation δ computed by the Prism selection algorithm is associated to a cost $C(\delta) \in [0, 1]$. Unlike Messmer and Bunke that only consider topological differences and limit to graphs with discrete node values, in our scenario cost functions are influenced by the presence of heterogeneous context dimensions.

Topology. The algorithm assigns the highest cost $C(\delta) = 1$ to the substitution of "Class" context units, core PRISSMA vocabulary properties (such as `prissma:environment`), and to the deletion of "Class", "Geo" or "Time" context units. Hence, whenever an input context graph needs such edit operations, the cost of the resulting error-tolerant subgraph isomorphism will be higher than the threshold T. The algorithm assigns lower costs for edit operations on non-core properties, and on "Entity" context units (e.g. a missing `foaf:interest` property in the user dimension may not prevent a Prism match). Such cost is determined by the $C_{topology} \in [0, 1]$ parameter. Note that the presence of additional properties between two context units is not considered to affect global cost, and is therefore assigned cost 0.

Location. A "Geo" context unit is a subgraph composed by `geo:lat`, `geo:long` and a `prissma:radius` (e.g. context unit 3 in Fig. 3). The cost of the substitution of a location context unit depends on the geographic distance. We first compute the distance d of the two points using the Haversine formula. If d is within the declared radius, the edit operation has cost $C(\delta) = 0$.

Otherwise, PRISSMA features an exponential decay function to smooth the transition between a perfect match and a mismatch[8]:

$$C_{geo}(d) = \begin{cases} 0 & \text{if } d < d_{radius} \\ e^{\frac{-d}{\lambda_{geo}}} & \text{if } d > d_{radius} \end{cases}$$

Time. Temporal context units include a start timestamp t_{start} and a duration Δ_t (Figure 1). The cost of the substitution of a temporal pattern is computed to an exponential decay function:

$$C_{time}(t) = \begin{cases} e^{\frac{t-t_{start}}{\lambda_{time}}} & \text{if } t < t_{start} \\ 0 & \text{if } t_{start} < t < t_{start} + \Delta_t \\ e^{\frac{-t+t_{start}+\Delta_t}{\lambda_{time}}} & \text{if } t > t_{start} + \Delta_t \end{cases}$$

Strings. The cost C_{string} of substituting a string literal is computed with an approximate string matching strategy, to overcome problems such as spelling variants (to date, the Prism selection algorithm focus only on this string similarity problem). The algorithm adopts the Monge-Elkan distance function (according to Cohen et al. [5] such function outperforms other approaches when dealing with spelling variants).

Precision-recall analysis[9] has been carried out to assess the validity of the Prism selection algorithm with different cost functions parameters, and with different similarity thresholds T. Future work will include a thorough campaign evaluation to assess the algorithm performance on a wider scale, and will involve PRISSMA-enabled applications users in the loop.

4 Evaluation

The PRISSMA decomposition and selection algorithms have been implemented as an Android library[9]. The library is showcased by the PRISSMA Browser[9], a mobile Linked Data browser enhanced with PRISSMA context-aware adaptation (Figure 5).

The first test analyses the decomposition memory consumption (Figure 4a). The test measured the decomposition size of groups of Prisms with a variable number of identical context units. Groups included 20 Prisms, each containing 10 context units. Overall, test Prisms accounted for 340 triples. The percentage of identical context units in each group of Prisms is progressively increased, ranging from 10% to 100% (where the latter means that all Prisms in the group are represented by the same decomposition item).

We assigned an arbitrary size of 30 Bytes to context units (we consider UTF-8 strings with an average length of 30 characters), and 42 Bytes to intermediate decomposition elements (one integer ID, two integer ancestors IDs, and a list of

[8] More refined geospatial matching techniques are out of the scope of this work, e.g. `http://linkedgeodata.org/`.

[9] Binaries, code, and evaluation results available at: `http://wimmics.inria.fr/projects/prissma`

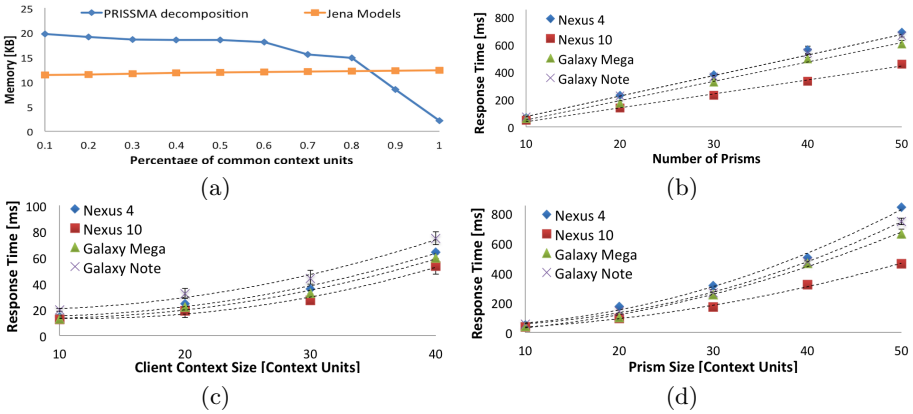
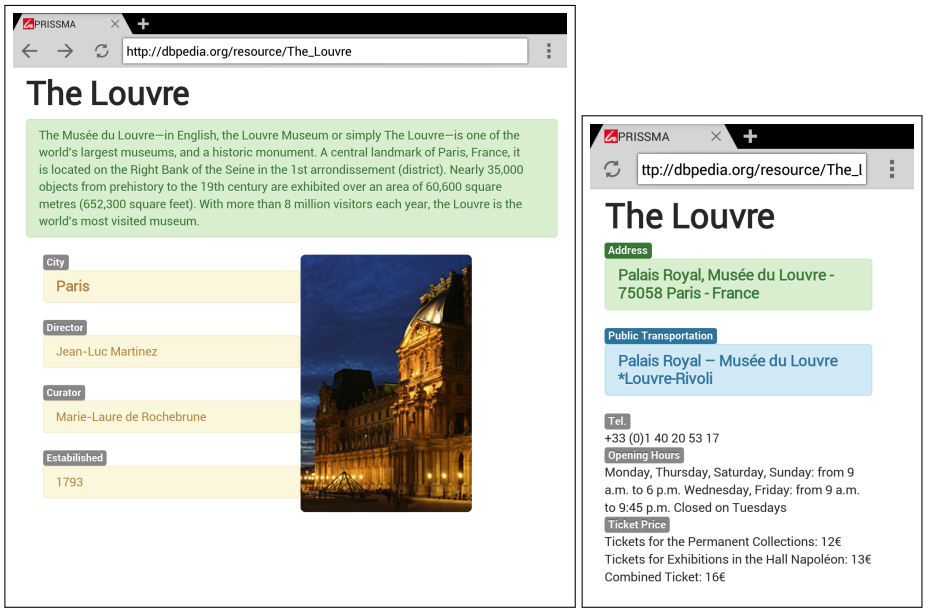


Fig. 4: Memory consumption (a) and Response time (b,c,d) of Prism selection algorithm.

connecting edges. Each edge includes a triple of estimated size 90 characters). The size of PRISSMA decompositions are compared with the retained size of a group of Jena Model[10], each containing a test Prism. As expected, with higher common context units percentages, we have lower decomposition memory footprints. Nevertheless, the memory size of PRISSMA decomposition is in the same order of magnitude as the Jena models size.

A series of tests have been run to prove the computational complexity analysis of the search algorithm. The algorithm response time has been tested with the proof-of-concept PRISSMA Browser on a group of Android mobile devices (Google Nexus 4, Google Nexus 10, Samsung Galaxy Mega, and Samsung Galaxy Note). All phones were running Android 4.2.2. Figure 4b shows the relationship between L, the number of Prisms in the decomposition, and response time. Prisms in each group are all different (thus testing the worst case decomposition configuration). Prisms contain $n = 10$ context units and the test context to be matched is made of $m = 10$ context units. Five independent runs have been executed for each group of Prisms, thus computing average response time measurements. Results prove a linear dependency, thus confirming the worst case complexity analysis of the search algorithm for what concerns the number of Prisms $O(L)$. Figure 4c shows how the size of the incoming context graph impacts on response time. In this case, each run varied the size m of input context (ranging from 10 to 50 context units) using a fixed group of $L = 5$ Prisms each made of $n = 2$ context units. Results match computational complexity analysis, thus giving a $O(m^n)$ relationship (experimental setup shown in Figure 4c has $n = 2$, thus giving a $O(m^2)$ relationship). Unlike the number of Prisms, the growth associated to the size of the incoming context graph suggests that the size of the latter must be kept as small as possible, to consistently reduce response time. Finally, in Figure 4d the size n of each Prism has been tested. Five independent test runs

[10] `http://jena.apache.org/documentation/notes/model-factory.html`



(a) (b)

Fig. 5: Two screenshots of the PRISSMA Browser Android application. Content and layout in (a) are optimized for tablets, while (b) is optimized for users walking in Paris (see the Prism in Figure 2).

assessed response time using an incoming context graph of $m = 50$ context units and a decomposition made of $L = 5$ Prisms. Results confirm the complexity analysis $O(n^2)$. As for the case of incoming context graph, the size of Prisms impacts with a quadratic growth on response time, thus it is important to avoid defining useless context conditions in Prisms to lower response time.

5 Conclusions

Extending Fresnel with context awareness favours the sharing and reuse of prisms across applications, does not introduce new formalisms, and is extensible to domain-specific context data. Operating on the client side guarantees privacy preservation, because context data does not have to be disclosed to third-party adaptation servers. Moreover, the decomposition structure supports incremental updates. Memory consumption tests show that the decomposition structure reduces memory usage when Prisms contain repeated subgraphs. Response time test campaign shows the sublinear dependence on the number of Prisms in the system. The main limitation of PRISSMA is the need for a proper parametrization of the selection algorithm. This is a well-known issue of strategies based on graph edit distance, that will be addressed in future work with machine learning

techniques. Additional cost functions will be added (e.g. semantic distance between URIs). Response time comparison with cited state-of-the-art solutions is envisaged, although experimental conditions vary, making the task tricky. Future work will also include deal user acceptability evaluation campaigns. Prisms distribution has not been examined yet: PRISSMA might support multiple strategies for discovery, retrieve, and consume Prisms published as Linked Data.

References

1. S. Auer, R. Doehring, and S. Dietzold. LESS - Template-Based Syndication and Presentation of Linked Data. In *In Procs of ESWC*, pages 211–224, 2010.
2. J. J. Carroll. Matching RDF Graphs. In *Procs of ISWC*, pages 5–15, 2002.
3. S. Castano, A. Ferrara, S. Montanelli, and G. Varese. Ontology and instance matching. volume 6050 of *LNCS*, pages 167–195. Springer, 2011.
4. P.-A. Champin. T4R: Lightweight presentation for the Semantic Web. In *Scripting for the Semantic Web, workshop at ESWC*, 2009.
5. W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.
6. D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, pages 265–298, 2004.
7. L. Costabello. DC proposal: PRISSMA, towards mobile adaptive presentation of the web of data. In *Procs of ISWC*, pages 269–276, 2011.
8. A.-S. Dadzie, M. Rowe, and D. Petrelli. Hide the Stack: Toward Usable Linked Data. In *Procs of ESWC*, volume 6643 of *LNCS*, pages 93–107. Springer, 2011.
9. A. K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, 2001.
10. J. Euzenat and P. Shvaiko. *Ontology matching*. Springer, 2007.
11. J. M. B. Fernéandez, S. Auer, and R. Garcia. The linked data visualization model. In *ISWC (Posters & Demos)*, 2012.
12. F. L. Gandon. Generating surrogates to make the semantic web intelligible to end-users. In *Web Intelligence*, pages 352–358, 2005.
13. X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Analysis & Applications*, 13(1):113–129, 2010.
14. K. Henricksen and J. Indulska. Modelling and using imperfect context information. In *PerCom Workshops*, pages 33–37, 2004.
15. D. Huynh, D. R. Karger, and D. Quan. Haystack: A platform for creating, organizing and visualizing information using rdf. In *Semantic Web Workshop*, 2002.
16. C. Kiefer, A. Bernstein, and M. Stocker. The fundamentals of iSPARQL: A virtual triple approach for similarity-based semantic web tasks. In *ISWC*. 2007.
17. m. c. schraefel and L. Rutledge. User interaction in semantic web research. *J. Web Sem.*, 8(4):375–376, 2010.
18. B. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998.
19. E. Pietriga, C. Bizer, D. Karger, and R. Lee. Fresnel: A browser-independent presentation vocabulary for RDF. In *Procs of ISWC*, pages 158–171. 2006.
20. D. Quan and D. R. Karger. Xenon: An RDF stylesheet ontology. In *Procs of WWW*, 2005.
21. K. Riesen, X. Jiang, and H. Bunke. Exact and inexact graph matching: Methodology and applications. In *Managing and Mining Graph Data*, pages 217–247. 2010.

22. L. Rutledge, J. van Ossenbruggen, and L. Hardman. Making RDF presentable: integrated global and local semantic web browsing. In *WWW*, 2005.
23. J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1), 1976.
24. J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and maintaining links on the web of data. In *Procs of ISWC*, pages 650–665, 2009.
25. L. Zou, L. Chen, M. T. Özsu, and D. Zhao. Answering pattern match queries in large graph databases via graph embedding. *VLDB J.*, 21(1):97–120, 2012.