Presenting and Navigating Styled RDF using Context-maps

Matthias Palmér, Ambjörn Naeve, and Mikael Nilsson

CID/Nada at KTH Royal insitute of Technology
Lindstedtsvägen 5, 100 44 Stockholm, Sweden

Abstract. This paper introduces context-maps based on the concept browser Conzilla as a user-friendly interface to formal knowledge that is expressed in RDF. Context-maps provide ways to visualize and edit RDF as graphs, where selected resources and statements may be presented in a styled fashion. An important problem to adress is the fact that RDF expressions are verbose, complex and not very intuitive to work with directly.
Our solution, context-maps, allows parts of RDF graphs to be summarized, suppressed and presented in a form that is more comprehensible to humans. Also, context-maps support the use of richer visual languages such as UML in our presentations.
In order to support various author friendly features, such as extensibility, annotation, reuse, etc., context-maps are expressed in RDF. This allows them to be embedded in the knowledge they describe, and to be processed by generic RDF tools such as low level editors and flexible search engines. Through the use of RDF, context-maps are given the necessary power and flexibility to maintain a one-to-one relationship between the visual interface and the represented formal knowledge.
Context-maps make heavy use of referring techniques and matching patterns in which variables are introduced. Context-maps are additionally extended with a navigational structure which is inspired by hyperlinks.

1 Background

Context-maps and our tool Conzilla originated from a need to build and present complicated knowledge structures that can enhace the learning process in various ways. The resulting expressions were inspired by object oriented modeling, most notably UML. However, they soon expanded to include navigational support like hyperlinks, occurrence relations similar to Topic Maps and support for filtering of such occurrences. Since the user groups included teachers, students as well as other non-expert groups, an effort was made to keep the interfaces intuitive and attractive. This led to the use of a more linguistically coherent modeling technique called Unified Language Modeling[5], which focuses on depicting how we speak about concepts and their relations. In practise we developed surfable context-maps containing concepts, n-ary role-based concept-relations and a basic type system, all expressed as interrelated XML-documents using Uniform

Resource Names. The resulting concept browser prototype, Conzilla-1, was a combined browser and editor for these types of knowledge expressions[13]. Because of a need for more flexible authoring and extensions to richer modeling languages, as well as for reasons of scalability, harmonization / standardization / interoperability, we have since then investigated how to change to another backend for knowledge representation. We found that the semantic web and most notably RDF[9] was appropriate to our needs, and we were a little surprised that from our point of view, it still seemed to lack good, generic user interfaces.

We chose to equip Conzilla with an RDF backend because we wanted to strengthen the bridge between human- and machine- semantics. Moreover, RDF represents an important step in the right direction with regard to scalability and extensibility. In general, with millions of users on the web - users with diverse and sometimes very conflicting opinions - it is crucial that RDF is designed to support knowledge representation in a scalable manner. It is also very important that items of knowledge can refer to other such items. Otherwise, questions like 'who said this' will not be answerable in a standardized manner. From a learning perspective these issues are equally important, especially since we believe that learning should not be regarded as an activity that is separated from other activities.

However, this paper will not focus on learning issues, but it will rather describe how context-maps are defined in RDF in order to enable generic user interfaces to edit and present knowledge expressed in RDF. Inspired by the needs of technology enhanced learning environments, we will show how to design context-maps that allow parts of RDF expressions to be summarized, suppressed and presented in a form that is more comprehensible to humans. An effort is made to design the context-maps so that they can be kept in tune with the various knowledge sources.

Nearly all the designs discussed in this paper have been implemented in Conzilla-2 where we have aimed for a full-fledged RDF presentation- and editing-tool. Presently Conzilla-2 is both an editor and a browser for context-maps, and extensions that will allow generic meta-data and query-editing are about to appear. Many of the implementation issues we have faced when developing Conzilla-2 has been used as a source of inspiration for the ideas and problems presented in this paper.

In preparing the figures in this paper we have used Conzilla as a modeling tool, e.g. the Figure 3 is then already a ready to use RDF Schema.

2 State of the Art

As a remedy for the web's lack of semantics - as well as to meet the rapidly increasing need to express meta-data about web resources - the W3C has defined *RDF* [9]. At its basic level RDF has very little semantics. However, with the help of the RDF Vocabulary description language [3], people are encouraged to introduce new layers of semantics on top of the old ones. For more complicated data the Web Ontology Language, OWL[2] is probably more suitable. It is im-

portant to notice that the base of RDF is agnostic to the kind of knowledge that is actually expressed.

There are many applications for working with RDF and the more specific languages expressed with the help of it. General frameworks such as *RedLand*[1], *KAON*[2] and *Jena*[3] for parsing, storage, querying and connections to inference engines are sufficient for their respective purposes. However, generic authoring tools, especially visual graph-based interfaces for end users, still seem to be lacking. Let us list some general features of authoring tools to clarify what sort of tools we are talking about:

#	Approach 1	Approach 2
1	Works directly with RDF	Native 'compatible' representation
2	Edits generic RDF	Restricted to e.g. specific set of properties
3	Manages multiple RDF sources	Manages one RDF source at a time
4	Can suppress information	Presents everything in the RDF source(s)
5	Visual - graph like - views	Interface based on forms or similarly
6	Has editing support in views	Views are for presentation only
7	Customizable layout and style	Automatic layout, fixed style

E.g. *Protege*[4] , a widely used tool for managing ontologies uses approach 1 for feature 3, 4 and the *OntoViz Tab plugin* manages to be configurable to some extent regarding 7. *IsaViz*[5] on the other hand takes approach 1 for 2, 4, 5, 6, and - with upcoming version 2 including Graph Style Sheets - also 7. With the latest improvements, this tool is pretty capable. Its main drawback is that it doesn't work directly with RDF. Hence, it cannot be used to present or annotate existing knowledge without loosing the connection to the original source. There are several other tools that provide editing facilities for RDF in a visual manner such as, *FenFire Loom*[6], *RDFAuthor*[7], the *XWMF*[8] tools etc. However, none of them take approach 1 for features 1, 2 and 5 at the same time. After thorough investigation, we have found that how to provide layout information to RDF from the outside is not a trivial task to solve. As a solution we will discuss in Chapter 4 how to refer to RDF constructs in detail. Before that we will in Chapter 3 investigate the features above in more detail. Then, in Chapter 5 we will see how the solution context-maps can be further extended with nice features, in this case some form of navigation. Finally, we present our conclusions and future work in Chapter 6.

[1] Redland http://www.redland.opensource.ac.uk/
[2] KAON http://kaon.semanticweb.org/
[3] Jena http://www.hpl.hp.com/semweb/jena.htm
[4] Protege http://protege.stanford.edu/
[5] IsaViz http://www.w3.org/2001/11/IsaViz/
[6] FenFire http://savannah.nongnu.org/files/?group=fenfire
[7] RDFAuthor http://rdfweb.org/people/damian/RDFAuthor/
[8] XWMF http://nestroy.wi-inf.uni-essen.de/xwmf/

3 Presenting RDF

Basic RDF statements come in chunks called RDF *graphs*[9] . Typically such graphs reside in databases or files that are retrievable over the Internet. The RDF semantics [15] specifies how to deal with RDF graphs, i.e. how to combine and split them, but not what it means for triples to reside in a specific RDF graph. In order to enable access to RDF graphs in a visual manner we will consider two very different strategies for performing presentations of RDF.

The first strategy regards the RDF graph as a suitable chunk for straightforward *visualization*. All statements in the RDF graph are treated equally and are included in the view. A visualization is typically a graph-view generated by some layout algorithm, something which several tools accomplish already, e.g. IsaViz. Typically *Graphviz*[10] is used to give the graph a nice layout in order to enhance human readability. However, in spite of nice layouts, graphs are often hard to read, mainly due to size and verbosity.

The second, and much more interesting strategy includes choosing a suitable set of statements from one or several RDF graphs and presenting those individual statements differently depending on what predicates they use, in which combination they occur etc..

We will now discuss the second strategy more carefully.

3.1 Introducing Contexts - Why We Need to Choose Statements

RDF graphs, often have many shortcomings with respect to presentation. First, they may be very large, e.g. the RDF graph for content descriptions of the Netscape Open Directory Project[11] amounts to 1200 Mb, and obviously this graph cannot be visualized directly. Second, and closely related, RDF graphs can be extremely verbose, expressing details that - for a given situation - are often of minor importance. Third, RDF graphs may be incomplete - in fact most RDF graphs implicitly use external schemata such as Dublin Core[1]. This is of course a good thing, since it encourages the use of standardized schemata / vocabularies. Hence, in a presentation it may be necessary to include them in order to get a complete picture. Fourth, a single RDF graph constitutes an information source wherein either an individual or a group is in control of the information. However, presentations often need to include relevant information regarding thoughts on the same subject originating from different sources which implies different RDF graphs. In order to be able to deal with these issues, we now define *context* to be a set of statements chosen to be presented together.

It is not an acceptable solution to form a new context by copying all the chosen statements to a new RDF graph since it would rapidly become incoherent with the original source. Instead, a context should directly reference parts of

[9] we will not use the name RDF *model* since it might be confused with the term model in logics.

[10] Graphviz http://www.research.att.com/sw/tools/graphviz/

[11] Open Directory http://dmoz.org/

the original RDF graphs. With this approach it is clear that reasonable changes (originating from separate editing etc.) in the RDF graph sources, will not corrupt or out date the context that easily. Preferably, an editor of the context should be able to 'edit through' directly into the original RDF graphs.

3.2 Layout of RDF Presentations

For several reasons we have chosen not to rely on layout algorithms for presentations. First of all, aesthetics is very hard to code in algorithms. It might also change over time or vary substantially between individuals. Second, the semantics of an RDF graph is rarely formalized in a way that it is available to a layout algorithm. Even if it were formalized, this would not necessarily imply that the algorithm would know what to do with it. Existing layout algorithms almost exclusively rely on information coded in the structure of the graph, so that e.g. textual descriptions will not matter. Third, human knowledge about a specific presentation is often available, i.e. the chosen context is in most cases subjective and hence not formalized and accessible by algorithms without extensive heuristics. We are not claiming that knowledge should be kept non-formalized; rather we see formalizations - i.e. encodings - of knowledge as an important step to allow communication between humans as well as between humans and machines. However, we acknowledge the fact that informal knowledge nearly always arises before any formal knowledge is encoded. Hence, layout is a powerful tool for communication that we want to preserve.

Unfortunately editing layout by hand is a very time consuming process. Therefore, a combined approach with layout algorithms as input to human editing can be very helpful. As layout methods we have considered - but not limited ourself to - absolute/relative positioning.

3.3 Blocking and Style

How would a presentation of a context look like, e.g. is graph-like presentations good enough? Here we have been inspired by *frame-based* representations [11], since, in the presentation, we would like to group properties around resources in a *frame-like* manner. Moreover, we have found much inspiration in the de facto industry standard UML [14]. However, frame based thinking is somewhat conflicting to the common usage of RDF [8]. This needs some elaboration:

First, RDF inherently represents an *open world*[12] of knowledge, i.e. anyone can add new knowledge and you can never be sure that you have found all of it. Second, much information expressed in RDF also makes use of a 'RDF schema' expressing classes and properties defined with the help of the RDF vocabulary description language [3] and possibly also with Web Ontology Language [2]. Now, since the schemata themselves are expressed in RDF, anyone can extend established schemata, e.g. adding new properties to classes defined elsewhere. This

[12] see definition in [3, 2] and discussion in [6]

is also recognized in the semantics for how the vocabulary description language works [15].

Hence, being an instance of a class will not suffice as a rule in defining a unique frame-like presentation, since the amount of properties applicable depends on which property definitions that have been found for this class. A solution would need to include a matching pattern for allowed properties.

Given a piece of a RDF graph constituting a frame, applications still need guidelines for customizing the presentation. For instance, the title should appear inside the box, and the description should be made available via a pop-up. Furthermore, the box should have rounded corners and the title should be centered in the north etc etc. Such hints should be separated in a *style* used by specific instances, or more preferably depending on the class of those instances.

4 Context-map Design

In this chapter we will consider a design for what we call *context-maps.* These context-maps will include a realization of the context concept introduced in the previous chapter. This will include some nitty-gritty details of how RDF represents knowledge and how we can refer to that knowledge via referring to RDF constructs. Since the context-maps will be expressed in RDF as well, we will need to be careful with what we talk about. The knowledge - expressed in RDF - that we want to present will be called *source knowledge*, the RDF representation of that knowledge will be called *source RDF* and the RDF expression of the context-maps will be referred to as the *context RDF*.

4.1 The Context-map Construct

Since we do not want to encode contexts by copying triples in the source RDF, we need a construct that refers to triples and resources expressed in other RDF graphs. For several reasons we have chosen to express these constructs (the context-maps) in RDF. First, it enables good integration with the source RDF using internal referencing techniques such as URIs and the reification mechanism. Second, it allows inference engines to easily make use of the combination of source RDF and context RDF. Third, it allows context-maps to be extended and reused in other contexts. Fourth, it allows flexible authoring and annotation of the context-maps themselves, effectively allowing statements like, 'I agree with what was said about that source knowledge' and so on.

A context-map contains a tree of *layouts* - which are RDF resources from where properties expressing graphical layout information originates - as well as a reference to the source RDF it presents. Different layouts - even in the same context-map - are allowed to refer to the same source RDF, i.e. the same triples or resources. This is useful both from pedagogical and aesthetic perspectives[13], but it is also a matter of showing a resource or a triple from different viewpoints.

[13] Typically you want to avoid crossing lines

Another advantage of using layouts is that context-specific information can be added on the layout level, avoiding to rely on keeping the RDF graphs separate.

Clearly, by this design, it is possible to store the context RDF, i.e. the layouts, in-line with the source RDF which might simplify the editing process. It is important that this be possible without changing the semantics of the source RDF. However, for practical purposes, this is not always possible such as when the RDF source is read-only or when there are several RDF sources. Keeping the context RDF separate, even though it does not affect the semantics of the source RDF, may even be a crucial principle for keeping the storage of source RDF pure and minimal. From this we recognize the need for references to RDF constructs across the borders between RDF graphs.

4.2 Referring To Resources and Triples

As pointed out above, the layouts of a context-map refers to the source RDF, i.e. resources and triples. In a context-map a triple is shown with the help of three layouts, two *layout-resources* and one *layout-triple*. Because triples have no identifiers the layout-triple refers, via a reification, to a triple in the source RDF[14]. Furthermore, since a resource may be presented by several layout-resources, the layout-triple must indicate which layout-resources it makes use of. The layout-triples indicated layout-resources should match the ends of the reification referred to by the layout-triple. If they do not match, the layout-triple is incorrectly constructed, (see Figure 1).

Notice, that the layout-resources refer to a resource reference in the source RDF. Where the resource reference in the source RDF simply is a URI. However, since it is meaningless to reference a reference, we just use the same URI[15] as in the source RDF. Hence we access the resource in the source knowledge for a layout-resource by following its reference and applying the *interpretation function* once[16]. For a layout-triple we need to apply the interpretation function twice, since we first have to interpret the reification resource in order to get to the triple in the source RDF, and then we must interpret the triple in order to get to the source knowledge that it expresses.

4.3 Referring Literals and Anonymous Resources from Layouts

It is not enough that context-maps can present resources and their connecting triples. Both literals and anonymous resources occur frequently and therefore we should be able to present them individually. Hence we must find some way to refer to them from layouts.

[14] In a given RDF graph there is only one triple with a given subject, predicate and object. Hence, in this graph, this triple is uniquely referenced by a reification. However, nothing (except good practises) prevents the same triple to occur in several RDF graphs or several reifications to refer the same triple.

[15] Compare with a triple referring its subject, predicate and object via their URIs.

[16] as described in the model theory [15].

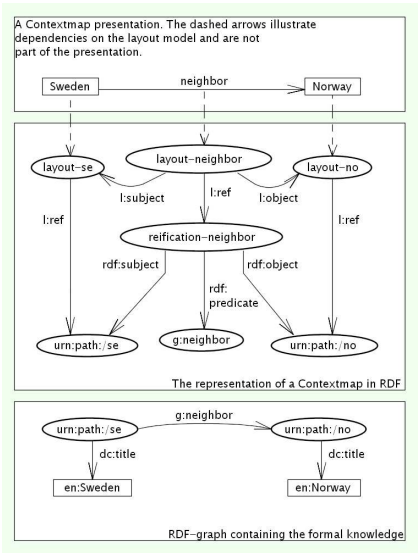

Fig. 1. Here we show how layouts refer to source knowledge about a relation between Sweden and Norway. The resulting (minimal) context-map is presented at the top. All properties regarding graphics have been excluded on the presentation of the layouts.

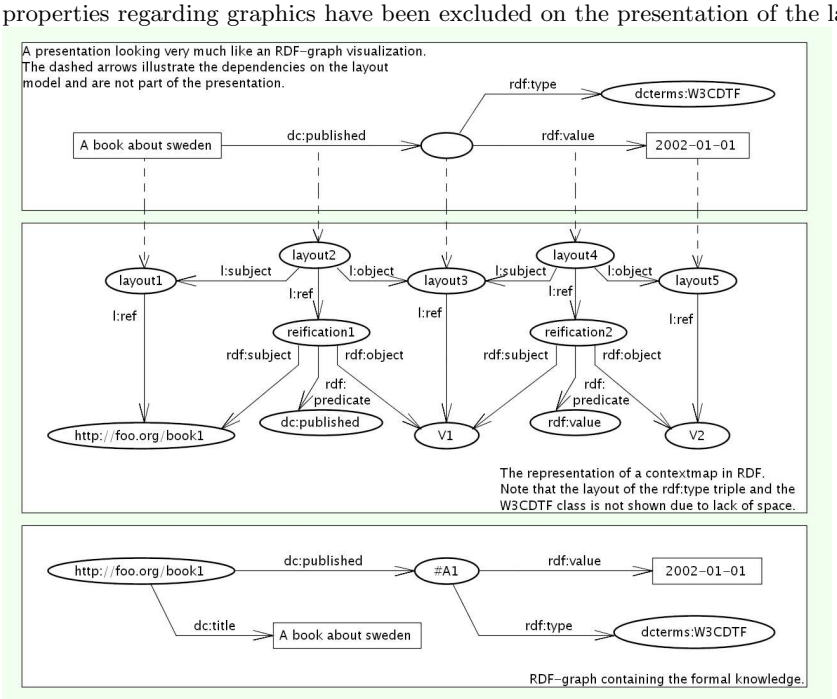

Fig. 2. Here the published date for a book about Sweden is presented via several layouts that together reference a matching pattern.

By definition, anonymous resources have no identifiers, and therefore they cannot be referred to directly - except from triples in the same RDF graph. This causes a problem, since we require that context-maps should be able to reside in separate RDF graphs from those containing the source RDF that should be referred to. In order to solve this problem we need to invent an indirect referencing technique, somewhat similar to how reifications indirectly refer to triples. A reification describes the structure of a triple by listing its subject, predicate and object. We need to describe the anonymous node position in the RDF graph without using any anonymous identifiers. The triples that the anonymous node takes part in, or even only one of them, should suffice in most cases. If all triples are common for two anonymous nodes, then they are not distinguishable at all - or only via their relations to other anonymous nodes that are distinguishable directly. For clarity, two anonymous nodes are indistinguishable if the graphs they occur in are isomorphic (disregarding the anonymous nodes identifiers). In such cases the referencing techniques break down. However, for presentation purposes this does not matter, since there is no information that is presented falsely (anonymous node identifiers should never be presented since they change all the time). When it comes to editing, this does not matter as long as the system makes sure to bind the indistinguishable anonymous nodes separately in order to avoid inconsistent editing. The only case when this approach breaks down is when someone changes the source RDF without knowledge of the context-maps so that anonymous nodes that were distinguished using one graph pattern now are distinguishable in another non compatible graph pattern. This will result in a broken presentation. With the current design of RDF this problem has no solution, hence we will not discuss this problem further. Instead we will focus on how to describe the matching pattern of an anonymous resource.

In order to identify an anonymous resource we cannot use a regular reification, since one of its triples would have the anonymous resource as its object. This would require that we could refer to the anonymous resource, which is impossible. Instead, we use reification as a base, but instead of the anonymous resource we put in a dummy resource and type it as a variable. In this way we obtain a matching pattern for triples. Several triple matching patterns can share the same variable, creating in effect a matching pattern for a subgraph. Hence, when we want to refer to an anonymous resource, we simply refer to a variable in a matching pattern. In this way, applying the interpretation function on the variable yields the anonymous resource. The variable for an anonymous resource will be referred by the layout for the anonymous resource, called a *layout-bnode*, as well as by the reifications for the layout-triples where the anonymous resource occurs. In most cases there is no need to provide a specific matching pattern for an anonymous resource, since such a pattern is provided by the referencing techniques used by the surrounding layouts. As an example, (Figure 2) the variable V1 matches an anonymous resource. Here the reifications referred by layout 2 and layout 4 provide the matching pattern for V1.

Let us now take a look at literals. A literal is always a part of a triple, which expresses something about its subject. We could let the layout of this triple be

responsible for the layout of this literal. However, using a 'special case layout' and a regular reification has two important drawbacks. First, the reification would need to repeat the literal, thereby increasing the risk of incoherence between the context-map and the source RDF, especially if there are several context-maps for the same source RDF. Second, layout-triples with literal support would be more complicated than other layouts and would have to be dealt with separately. So instead we use the same solution as for anonymous resources, i.e. a variable and matching patterns. If there are several triples differing only in the object literal, a constraint can be added on the variable in order to distinguish them.

Whenever a context-map references variables, it contains an implicit query, which in most cases is very simple to resolve. However, in a worst case scenario, when there are many anonymous resources to resolve, it might require a full-blown query engine to process it. In Figure 2 we see that the variable V2 should match the date in the form of a literal, no extra constraint is needed. It is not uncommon with triples of the form ¡[anonymous resource], [rdf:value], [literal]¿. Hence reification2 will probably not give a unique match for V2. However, if reification1 is matched first, the specific anonymous resource matched for variable V1 will probably cut down the matching triples to a unique match for variable V2.

Observe that we treat reifications as matching structures where the match of a reification is a triple and not a resource. Hence the reification resource is really a variable. That means that the rdf:Statement class should be a subclass of the *v:Variable* class. The reifications that allow pointing to variables instead of resources can then be seen as belonging to a subclass of Statement. From now on, we will refer to this subclass as a *v:GraphStatement*. Moreover, there should be three extra variable classes, corresponding to variables that match resources, literals and bnodes. This would suggest an RDF schema for variables as shown in Figure 3.

The approach described here is largely similar to the Edutella[12] project query exchange language QEL3. QEL3 provides datalog style rules on top of GraphStatements (named QueryStatements) as well as constraints on variables. We will harmonize schemata with the Edutella project as far as possible.

4.4 Frame-like Views

So far we have seen how a context-map can present plain nodes and arcs. The next step is to allow frame-like and styled presentations. Consider a typical situation where we want to present information about e.g. Sweden.

A resource has a dc:title 'Sweden' and is an instance of a class with an rdf:label 'Country'. Instead of presenting the rather unintelligible URIs of the resource and the class, the strings 'Sweden' and 'country' are displayed in their respective boxes. The instantiation triple rdf:type is drawn as a directed relation in this situation with a v-shaped arrow and a dotted line - as a replacement for writing out the predicate name rdf:type. Another triple on the resource - with a predicate geography:populationCount and a value of 9000000 - might be shown

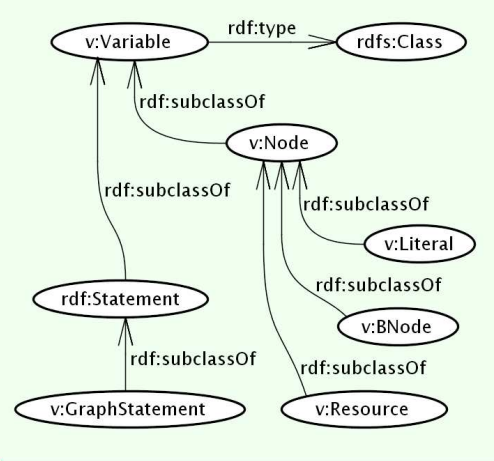

Fig. 3. Variable classes constituting part of a suggested RDF-schema for context-maps and Edutella.

as a UML-like attribute within the box. The attribute tag could be fetched from the rdf:label of the predicate.

Triples or subgraphs used within frames could - and in most cases probably should - be removed from the plain graph-view in order to improve readability of the presentation. When removed, verbosity is traded for a richer - and maybe more complicated - visual language, which should be defined by a style. In order to be able to present source RDF in a frame-like fashion, we need to extract relevant pieces of the RDF graphs, and therefore we need to find a graph-matching mechanism that enables this.

The approach described in Section 4.3 will work fine in this situation as well. However, in this case we do not reuse the implicit matching pattern of a context-map. Instead we provide a specific matching pattern separately. Furthermore, since the demands might be quite complex, conditional rules and constraints will be needed. In principle, a fullfledged Edutella query[12] might be called for. We would like to specify a matching pattern for a specific type of frame, i.e. a class, and then apply it to all instances. For this kind of reuse of matching patterns, the resource to be presented as a frame has to be inserted at run-time in place of a specific parameter variable. Of course, matching patterns work equally well for reifications of triples. In this case the parameter variable is replaced by the reification resource[17].

Since there are probably different subgraphs around each resource, the reusable matching pattern cannot be expected to provide an exact match. For instance, a very common pattern would be to ask for the dc:title and the dc:description. If both of these exist, we would like to match them both, but if only one of them

[17] which is a special kind of variable ,which has a matching pattern, that matches triples (see figure 3)

is expressed, we would like to match only that one. This can be achieved by performing a disjunctive query covering all three cases, i.e. *dc:title and dc:description*, *only dc:title* or *only dc:description*. Clearly, when the number of separate matching patterns grows, this approach leads to 'combinatorial explosion'. However, this can be avoided by doing an *outer join* of the separate matching patterns. Since Edutella queries presently do no support outer joins, we have to extend the underlying query language. For the time being we have defined a specific type of query containing GraphStatements and constituting a tree, where outer join is performed between the branches[18].

Now, when we know how to match subgraphs, we must define how to style the matched information. Such styles would define e.g. that matchings to variable X should be presented as a string within a box, whereas matchings to variable Y should be presented as a pop-up etc. The current solution for styles is a proof-of-concept implementation that is under further investigation.

4.5 Multiple RDF Graphs

We will now investigate the effect of having several RDF graphs for the presentation perspective and the editing perspective.

One of the layout design issues was to be able to refer to constructs within RDF graphs from outside, but do we really need to know which RDF graph we are referring to? First, layout-resources only refer to resources, that are entities outside of RDF graphs. Second, reifications referred to by layout-triples need not refer to specific RDF graphs since they provide no extra information if found in a specific RDF graph or not. We know the form of the triple anyway. Third, matching patterns referred to by layout-bnodes and layout-literals may need to indicate which RDF graph that they match if in case the matching pattern cannot be made exact enough. Fourth, frame-like views allow matchings from several RDF graphs to be combined, at least between branches where outer joins are performed. Hence, the referencing techniques almost always can function without telling which RDF graph they reference. However, which RDF graphs a system has access to[19] when displaying a context-map, will have both explicit and implicit consequences for the presentation. If sources are missing, then part of the context-map will be broken. If there is a more subtle change in appearance, it is more likely that references such as graph patterns will be broken or styles will be missing. This brings us to the conclusion that context-maps in general should be able to specify a suitable set of RDF graphs to be loaded before display. This is discussed further in Section 5.3. A system that displays context-maps should - as a favour to the user - be very clear about which knowledge sources it has made use of in the assembling of the context-map. If not by other reasons it should be provided in order to answer questions of the type 'who said this' or possibly 'where was this said'. A simple 'knowledge source manager' is currently in place

[18] This will be discussed in a separate paper.

[19] or chooses to access

where sources can be inactivated or reordered so that the user can discover from where information is retrieved and how it is prioritised.

In editing sessions, dealing with many RDF graphs at the same time may be confusing. The concept of editing through into the source RDF when working with a context-map might have strange effects. E.g. when you change a title in one box you edit one RDF graph, when you change a title in another box you edit another RDF graph. The task of creating new knowledge is also hard to make intuitive, since it is unclear which RDF graph should be chosen to be edited. We are investigating the introduction of the metaphor of a *project* as a possible solution, where information such as default RDF graph is recorded for editing purposes.

A special case of editing is 'annotation mode', which is accomplished by making a specific RDF graph active, where personal annotations can be expressed as extensions to other knowledge sources. This approach is close to the solutions of the Annotea[7] initiative, where users can annotate material on the web via XPointer[4] and RDF graphs - either locally or on special servers. In principle, the Annotea technology can be made to work with context-maps. Actually it is simpler with context-maps than with, let's say XML documents, since everything has a URI already, and hence there is no need to calculate an XPointer. Unfortunately, Annotea is limited to a specific schema for doing annotations, which our 'unrestrictive import' of RDF graphs in 'annotation mode' is not. However, the idea of having special annotation servers is very interesting and worth to explore further.

5 Context-map Navigation

Now, when the basic problems of how to define context-maps in order to provide context views of various sources of knowledge is solved, we will shift our focus to what can be done with the context-maps themselves. Here we will restrict ourselves to one important application of context-maps: hyper-linking and navigation.

5.1 Hyper-linked Context-maps

Comparing with HTML4, context-map hyperlinks should resemble the *A* element combined with the specific link-type semantics[20] specified in the header specific *LINK* element. In practice, we define a new property, *navigation:hyperlink* as a subproperty of rdfs:seeAlso. The definition of rdfs:seeAlso in [3] says that it 'is used to indicate a resource that might provide additional information about the subject resource' which is vague enough[21]. However, the domain and range for the hyperlink property should be more restrictive. More specific link-semantics

[20] There is a closed set of types available by default. Alternatively the extension mechanism defined by the *PROFILE* element can be used to define new link types.

[21] the hyperlink should in principle be allowed to be free of semantics just like the A element.

such as 'details', 'overview', 'inspired by' etc. can be accomplished by defining new sub-properties of the hyperlink property.

In a context-map, hyperlinks should be allowed on virtually anything, including all kinds of layouts. Hyperlinks are added on the context RDF, not the source RDF. There are two reasons for this. First, hyperlinks provide navigational information about context-maps, thar have no meaning in the knowledge source, so we don't want to pollute the knowledge source. Second, this allows context-maps to have different hyperlinks on the same source RDF which is crucial to allow different perspectives.

Therefore hyperlinks are properties with layouts as domain but what about the range? We see two possibilities here: either you link to an entire context-map or to a specific layout within a context-map - similarly to how you can hyperlink to anchors in HTML. Observe that a layout does not need to have a *URI-reference* [16] with the same base as the URI-reference of the context-map[22]. Hence if you want to link directly to a layout, you need to additionally specify which context-map it occurs in, since this is not implicit in the URI-reference. In this case the hyperlink points to a structured resource, with rdf:value pointing to the link target, and a *navigation:contextMap* pointing to the containing context-map.

5.2 Contextual Neighborhoods

We have seen how hyperlinks can be provided in order to create connections between various context-maps. Another approach would be to acknowledge the implicit navigational structures that already exist. If we consider a specific piece of source knowledge expressed by some source RDF, the set of context-maps where this knowledge is referred is defined as its *contextual neighborhood*[5]. In practise, on a layout object in a context-map, a list of its contextual neighborhood context-maps are reachable. Clearly, there is no way to generate a full list, unless a closed world is assumed. Even then, due to the complexity of referring to source RDF, contextual neighborhoods should probably only be listed from explicitly referred source RDF, i.e. objects that are referred through variables and matching patterns should not be considered. In practice, an application might keep track of its recently visited context-maps and use this information to generate 'reasonable' contextual neighborhoods. This is the current approach. The Edutella peer-to-peer network can function as another source for discovering contextual neighborhoods when more complete listings are required.

5.3 How to Load a Context-map

Notice that a single RDF graph may contain several context-maps and a specific context-map may be spread out over several RDF graphs. Hence, in general, given a URI for a context-map, in order to be able to load it, we need to know

[22] Compare with HTML documents, where anchors are always identified via URI-references constructed by adding a fragment identifier to the document's URI.

which RDF graphs it is expressed in. For simplicity we require that retrieval of RDF graphs be specified by URLs or at least indirectly via URIs. We define a property *navigation:includeContainer* which has the context-map to be loaded as its subject and the URI of an RDF graph as its object. In an RDF graph where a hyperlink is expressed, several such includeContainer-triples may exist. When new RDF graphs are loaded, new includeContainer-triples referring to the same context-map may be found and - if followed - may lead to new includeContainer-triples and so on. If you are unlucky, the number of new RDF graphs to load may explode. A wise policy would be to only follow those includeContainer-triples that are specified in the RDF graph where the hyperlink is expressed, and then follow all includeContainer-triples found in the newly loaded RDF graphs. This approach works fine in the current implementation. Furthermore, different principles may be used for accessing RDF graphs, e.g. using caches or preload-mechanisms. Presently, a cache is in place.

In OWL[2] there is a construct, that requires other ontology containers to be included. Unfortunately, in the current OWL specification, an ontology is a specific document containing RDF in some form. Hence, the identifier for the ontology and the document are defined to be the same, which is not the case for context-maps. Consequently, the import property in OWL cannot be used, since its semantics is slightly wrong. An important reason for this is that ontologies are often crafted with minute care and can therefore be trusted to have well defined stable URLs. Context-maps, on the other hand, may be created in one place, published in another, moved several times and retrieved in many ways.

6 Conclusions and Future Work

In this paper we have described how knowledge expressed in RDF can be presented via context-maps. The context-maps have been designed to allow customizable presentations with respect to both size and appearance. It is important to emphasize that the knowledge expressed in the RDF graphs need not be changed or duplicated because of presentational issues. At the same time, however, the context-maps are not disconnected from the RDF graphs they present. In fact, they may be used as a user-friendly interface in order to work with knowledge expressed in RDF graphs. We have also described how a navigational layer can be added to context-maps. Future work includes (at least):

1. Investigate in more detail how the style of a context-map should be expressed in order to allow maximal flexibility.
2. Improve the user interface and perform usability tests, especially regarding the handling and editing of multiple RDF sources.
3. Harmonize queries with the Edutella query language.
4. Investigate how to edit OWL-ontologies with the help of styled context-maps.
5. Continue to look at how frame-based knowledge like UML should be represented in RDF and presented via styled context-maps.
6. Continue to work on new ways of navigating and searching for context-maps.

Our main goal - which is to show how non-experts can access and work [10] with the semantic web in a graph-like manner, has been partly fulfilled with the design of context-maps and the Conzilla-2 prototype. Our hope is that styled context-maps with navigational capabilities and flexible but precise searches will give rise to a new generation of user-friendly tools that have the potential of bringing the power of the semantic web to a much broader audience.

References

[1] Dublin core metadata element set, version 1.1: Reference description. http://dublincore.org/documents/2003/02/04/dces/.

[2] OWL Web Ontology Language Semantics and Abstract Syntax. http://www.w3.org/TR/2003/WD-owl-semantics-20030331/.

[3] RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/2003/WD-rdf-schema-20030123/.

[4] W3C XML Pointer, XML Base and XML Linking. http://www.w3.org/XML/Linking.

[5] Ambjörn Naeve. The concept browser a new form of knowledge management tool. In *Proceedings of the 2 nd European Web-based Learning Environments Conference (WBLE 2001)*, 2001.

[6] Tim Berners-Lee. What the Semantic Web can represent. http://www.w3.org/DesignIssues/RDFnot.html, 1998.

[7] Jos Kahan and Marja-Ritta Koivunen. Annotea: an open rdf infrastructure for shared web annotations. In *Proceedings of the tenth international conference on World Wide Web*, pages 623–632. ACM Press, 2001.

[8] K. Baclawski, M. Kokar, P. Kogut, L. Hart,.J. Smith, W. Holmes, J. Letkowski, M. Aronson. Extending UML to Support Ontology Engineering for the Semantic Web. http://ubot.lockheedmartin.com/ubot/papers/publication/UMLOntology.pdf, 2001.

[9] O. Lassila and R. Swick. Resource description framework (rdf) model and syntax specification. http://www.w3.org/TR/REC-rdf-syntax/.

[10] Mikael Nilsson, Matthias Palmer, Ambjörn Naeve. Semantic Web Meta-data for e-Learning - Some Architectural Guidelines. In *Proceedings of the 11th World Wide Web Conference (WWW2002)*.

[11] M. Minsky. A framework for representing knowledge. In D. Metzing, editor, *Frame Conceptions and Text Understanding*, pages 1–25. de Gruyter, Berlin, 1980.

[12] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. Edutella: A p2p networking infrastructure based on rdf. In *Proceedings of the 11th World Wide Web Conference (WWW2002)*, 11 2001.

[13] Mikael Nilsson. The conzilla design - the definitive reference. http://kmr.nada.kth.se/papers/ConceptualBrowsing/conzilla-design.pdf, 2000.

[14] OMG. Omg unified modeling language specification. http://cgi.omg.org/docs/formal/01-09-67.pdf.

[15] Patrick Hayes. Rdf semantics. http://www.w3.org/TR/2003/WD-rdf-mt-20030123/.

[16] T. Berners-Lee, R. Fielding, U.C. Irvine, L. Masinter. URI Generic Syntax. www.ietf.org/rfc/rfc2396.txt, 1998.