Conzilla - a Conceptual Interface to the Semantic Web

Matthias Palmér, Ambjörn Naeve

No Institute Given

Abstract. This paper has two foci that are intended to be complementary. First, it describes Conzilla as an incarnation of a concept browser. More specifically, as a technical solution for expressing context-maps, concepts, concept relations etc. Second, it introduces Conzilla as a fairly complete RDF editor which combines graph- and form-based manipulation of RDF-graphs.
Apart from these foci, the main requirements for the Conzilla design is: It should serve as a collaboration tool for more or less formalized modeling techniques, most notably UML-dialects. It should simplify the task of creating information according to various metadata standards. It should support customized presentations of existing information without requiring duplication or modification of information sources. These requirements are fullfilled by choosing a three layered approach for working with semantic web information in Conzilla, i.e. the *information*, *presentation* and *style* layers.

1 Introduction

Our concept browser[8] Conzilla[10] originated from a need to build and present complicated knowledge structures that can enhance 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[13] and support for filtering of such occurrences. Since the user groups included teachers and 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* (ULM) [8], which focuses on depicting how we speak about concepts and their relations. In practice 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, URN:s. The resulting concept browser prototype, Conzilla1, was a combined browser and editor for these types of knowledge expressions[10]. 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[5] was appropriate for our needs, and we were a little surprised to find that from our point of view, it still seemed to lack good, generic user interfaces.

We have chosen to equip Conzilla with an RDF backend because we wanted to strengthen the bridge between human- and machine-understandable 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, rather will it 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. Furthermore, Conzilla2 is still a concept browser and provides functionality such as hyperlinks between context-maps and the specific separation between content and context. We believe that these features beyond being useful in the setting of a concept browser are truly beneficial for the Semantic Web as they contribute to providing a better overview of the rather verbose and obsure expressions in RDF.

Conzilla2 has also replaced the hardcoded IMS metadata editor of Conzilla1 with SHAME[4], a configurable meta-data editing, presentation and querying library. SHAME uses a sort of application profile to generate interfaces to nearly all kinds of metadata as long they are expressed in RDF. Many of the implementation issues we have faced when developing Conzilla2 have 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 and drawing tool.

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 metadata about web resources - the W3C has defined RDF[5]. At its basic level RDF has very little semantics. However, with the help of the RDF Vocabulary description language[2], people are encouraged to introduce new layers of semantics on top of the old ones. For more complicated data, the Web Ontology Language, OWL[12] is probably more suitable. It is important to notice that the base of RDF is agnostic to the kind of informtion 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], Jena2[3], Sesame[4][3] and SCAM[5] [11] for parsing, storing, querying and making 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. In figure 1 we compare the performance of our tools Conzilla2[6] Meditor[7] with IsaViz[8], VUE[9] RDFAuthor[10], InferEd[11], Protege[12] with respect to RDF editing and/or conceptual modeling capabilities. We claim neither that this list of tools is complete, nor that the features considered are exhaustive with respect to their capabilities. Instead, the features brought up reflect what we believe to be important in the design of Conzilla2. Hence it is no mystery that Conzilla2 stands out as the most feature-rich tool in this comparision.

Let us perform a more qualitative investigation of IsaViz and VUE. We select IsaViz because it is a capable graph-based RDF-editor, and we select VUE because it is a capable knowledge-structuring and modeling tool.

VUE is a concept-mapping tool that connects to - and filters - the content of various digital repositories. The focus is on maps and the management of content in nodes and links. It has many nice features, including the filtering of content and presentation paths through maps. However, in comparison with Conzilla2 it lacks a visual language strong enough for modeling in e.g. UML and since nodes cannot occur in several maps the navigational aspects (feature 9) are rather weak compared to the capabilities of a concept browser. Furthermore the separation between information and presentation is done only for content, while the nodes and links appear in the presentation layer only. From the perspective of comparing VUE with Conzilla2 it would be interesting if the presentation file format used RDF. And provided machine understandable semantics for the concepts, concept relations, navigational aspects and the relation to content. However, as it stands now, VUE is not an RDF tool (feature 1,2, and 3), even though it might integrate content from repositories that uses RDF internally.

IsaViz on the other hand has a strong focus on the information representation as is mainly an RDF editor. It has a lot of nice features like a zoomable interface and the ability to apply Graphical Stylesheets[14] to customize the presentation of individual graphs or schemas. However, because of how RDF is designed,

[1] `http://www.redland.opensource.ac.uk`
[2] `http://kaon.semanticweb.org/`
[3] `http://jena.sourceforge.net/`
[4] `http://www.openrdf.org/`
[5] `http://scam.sourceforge.net/`
[6] `http://www.conzilla.org/`
[7] Meditor is an application of the SHAME framework `http://kmr.nada.kth.se/shame`
[8] `http://www.w3.org/2001/11/IsaViz/`
[9] `http://vue.tccs.tufts.edu/`
[10] `http://rdfweb.org/people/damian/RDFAuthor/`
[11] `http://www.intellidimension.com/pages/site/products/infered/default.rsp`
[12] `http://protege.stanford.edu/`

Feature number:	Feature 1 to 3 requires that the tools can edit or export to RDF			4	Feature 5 to 9 requires that the tool supports a visual graph view				
	1	2	3		5	6	7	8	9
Feature description: Tools:	X = Works natively with RDF O = Some 'compatible' representation - = Not applicable	X = Edits any kind of RDF O = Editing restricted somehow - = Not applicable	X = Combines multiple RDF models O = Model centric - = Not applicable	X = Visual – graph like view O = Only text oriented view - = Not applicable	X = Has WYSIWYG editing O = Separated editing - = Not applicable	X = Can suppress information O = Presents everything - = Not applicable	X = Customizable layout O = Automatic layout only - = Not applicable	X = Appearance controlled by style O = Appearance fixed by the tool - = Not applicable	X = Navigation between views O = No navigation - = Not applicable
Conzilla2	X	O	X	X	X	X	X	X	X
IsaViz	O	X	O	X	X	X	X	X	O
VUE	-	-	-	X	X	X	X	O	X
RDFAuthor	O	O	O	X	X	O	X	O	O
InferEd	X	X	O	O	-	-	-	-	-
Protege	O	O	X	X	O	O	O	O	O
Meditor	X	O	X	O	-	-	-	-	-

Fig. 1. A feature overview of RDF and/or conceptual modeling tools.

there are problems with referring to parts of graphs from the outside. This has as a consequence that the IsaViz designers have chosen to use their own format (feature 1) for saving graphs whenever the appearance is customized (e.g. when doing your own layout or suppressing information). Unfortunately this has some serious drawbacks, e.g. you cannot customize the presentation of an RDF graph without making a snapshot of it, and therefore, independent subsequent changes cannot be incorporated without starting all over. IsaViz is oriented around RDF, not around maps and consequently has no navigational primitives beyond the navigation within the customized view of a single RDF-graph (feature 9).

3 Short Conzilla Interface Overview

3.1 Basic browsing and editing

Conzilla2 allows you to browse context-maps and inspect concepts and concept-relations. Figure 2, depicts two context-maps in browse mode. Here we see several transparant popups with parts of the Dublin Core metadata fields [1] for concepts and concept-relations. In the context-map in front we have brought up a pop-up menu, which contains the alternatives *surf*, *view* and *info*. The info alternative would bring up a more complete metadata inspector, that allows you to view other parts of the descriptions that are suppressed in the map. You can change your language preference in the settings menu.

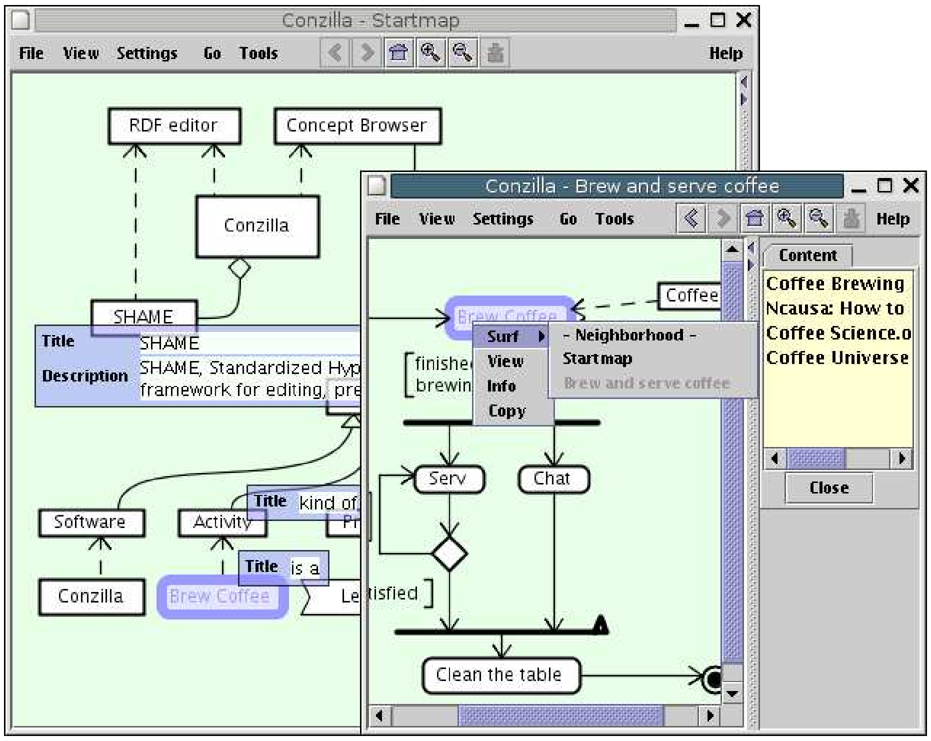
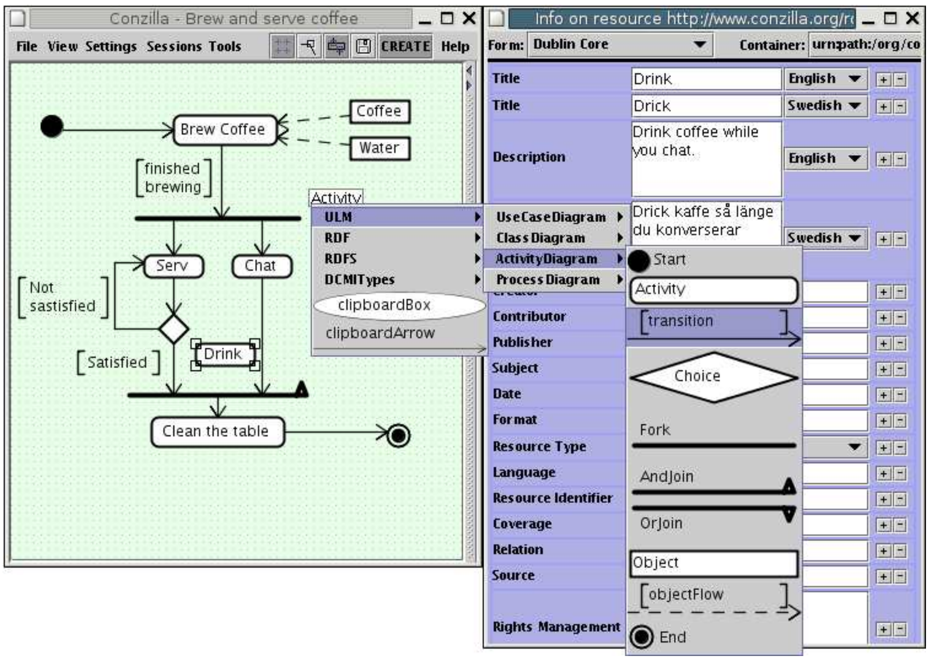


Fig. 2. The context-map in the back displays a ULM class/instance diagram with three transparent pop-ups showing Dublin Core information around the concept 'SHAME', a relation named 'kind of' (similar to rdfs:subclassof), and a relation named 'is a' (similar to rdf:type). The concept 'Brew Coffee' is shown in blue (in both maps) to indicate that it is selected. In the context-map in front we see a ULM activity diagram with a popup-menu showing the three alternatives, *surf*, *view* and *info* on the selected concept. The surf-submenu shows the conceptual neighborhood with the present context-map (Brew and Serve Coffee) grayed out. Since the info alternative was previously chosen, the list of content-components for the concept "Brew Coffee" is still shown to the right.

In figure 3, we see one of the context-map from figure 2 - but this time in edit mode. We also see a Dublin Core metadata editor for the newly created concept of type Activity with the English title 'Drink'. If you would like to edit other metadata fields than those provided by Dublin Core you can change metadata-form in the choice-box where, it says Dublin Core. Only a few metadata-forms are provided by default e.g. Dublin Core, LOM and FOAF. But, via the *Formulator* application of the SHAME framework, you can create new metadata-forms or reuse parts from established standards / schemas. Which metadata-form that should be used in the browse mode can be specified in the style layer.



Form:	Dublin Core	Container: urn:path:/org/co
Title	Drink	English
Title	Drick	Swedish
Description	Drink coffee while you chat.	English
	Drick kaffe så länge du konverserar	Swedish
Contributor		
Publisher		
Subject		
Date		
Format		
Resource Type		
Language		
Resource Identifier		
Coverage		
Relation		
Source		
Rights Management		

Fig. 3. A context-map showing how to brew coffe as a ULM Activity diagram. The activity 'Drink' has just been inserted and the user is choosing the property 'transition' in the type menu to be able to connect the activity to the synchronizationbar[14]. On the side we see the Dublin Core metadata editor for the newly inserted activity with Swedish translations of the 'title' and the 'description'.

3.2 Advanced usage

All editing is done in *sessions*. Among other things, a session specifies *containers*[15] for storing concepts and concepts-relations as well as graphics and style information. From a collaboration perspective it is important that you are able to use sessions with different containers for the same context-map. If the different containers are loaded the map shows everything combined, in editing mode you still see everything, but you can only edit according to the priviledges of your session. The *container-manager* can be used in order to investigate context-maps that have mixed origin. The concepts and concept-relations can also be grouped into different *layers* in order to simplify the editing process or aid in a certain presentation perspective.

[15] for the moment can containers only be regular RDF/XML files locally or via ftp, we plan to support remote RDF-stores like SCAM or SESAME

4 Context-map design

Fundamentally, a context-map should present concepts and concept relations in a manner that fullfills the requirement of a concept browser. Also, we have made it a priority that the information that makes up a concept or concept-relation is expressed in RDF, is separated from its graphical layout, and is allowed to reside in several RDF graphs. Support for the reasoning behind this design can be found e.g in [15] and [6]. Our motivation can summarized in the following design goals:

1. Concepts and concept-relations are human expressions that may be more or less strict. It is important that the expression has the potential for beeing machine readable.
2. Expressing information is often a task that requires collaboration or at least positioning within a larger setting. Hence, it is important that authors can work with separate information sources and still have a unified view.
3. Human expression is seldom uncontroversial, allowing the same information to be viewed from another perspective without loosing sight of the similarities is important. Hence it is important to provide good separation between the information and presentation layers of a context-map.

It is noteworthy that from a technical perspective, the nature of RDF allows information to be spread out. Hence, in order to make full use of the strength of RDF, we need to ensure that context-maps are able to collect information from disparate sources and assemble this information into a unified presentation.

In the following subsections we will describe the context-map design in terms of three different layers, an *information*, a *presentation* and a *style* layer. This division in layers is inspired by how the regular web has evolved into a division between data storage (typically a database), HTML (typically generated via some template language) and a stylesheet (typically CSS or XSL).

4.1 Information Layer

The information in the information layer is made up of statements around resources expressed in RDF. In most cases the actual resources have an extent that cannot be captured in the information layer. This extent may include things with digital representations like documents or pictures or it might equally well include things like ideas, processes or people. A concept in a context-map is constituted of a subgraph of RDF-statements centered around a specific RDF-resource. In a similar manner a concept-relation in a context-map is constituted of a subgraph of RDF-statements centered around the reification of a specific RDF statement.

The expression of the information may follow a schema, a standard or be locally defined, it depends on what is suitable for the domain and the user.

As mentioned above, information presented in one context-map may reside in several RDF graphs.

4.2 Presentation Layer

The focus of the presentation is the idea of a context-map as introduced in [8]. In short it is a map of concepts and connecting concept-relations presented by boxes and lines. There might be text in the form of a label within the box and on the side of the line, both excavated from the information layer.

Every concept or concept-relation has its own integrity, i.e. its existence is not bound to a specific context-map. The concepts or concept-relations are included in a context-map via *layouts*. Layouts are intermediate resources holding information regarding position, size, and in-line styles such as text-alignment, visibility etc. Observe that the same concept or concept-relation may be inserted several times in the same map yielding different layouts. There is a special kind of layout, which groups other layouts. Such layouts are typically used for creating context-map layers.

In addition to the context-map presentation it is possible to interact with concepts and concept-relations in various ways. The three interactions presented below are central to the the idea of a concept browser and quite useful for an RDF editor as well:

Surfing First of all, there are hyperlinks on concepts and concept-relations leading to other context-maps. A hyperlink is stored on the layout, hence for a concept / concept-relation there might be different hyperlinks depending on where it is encountered. Second, since the same concept (or concept-relation) may occur in different context-maps it is possible to provide a *contextual neighborhood* [8] which is a list of all context-maps where the given concept (or concept-relation) occurs.

Since it is possible to include concepts / concept-relations in context-maps without anyone else's knowledge, it is impossible to be sure that you have a complete listing of contextual neighbourhoods. We have considered and tested to use the p2p network Edutella [9] for finding contextual neighbourhoods. However, in the long run it might be neccessary to develop a specific service which specializes in keeping track of contextualizations of concepts / concept-relations for efficiency reasons. The current implementation of Conzilla2 only checks all RDF-graphs that are loaded already.

Viewing Content One of the main design principles of a concept browser is to have content-components separated from their initial context. Content-components can then be assigned to relevant concepts or concept-relations as e.g. explanations, examples, motivations, discussions, or knowledgable people. It is possible to assign content-components to a concept or concept-relation within the scope of a context-map. These content-components will not be visible on that concept in any other context-map.

Just like a concept or a concept-relation a content-component might have an extent which goes beyond what is expressible in the information layer. Hence, if a content-component is detectable as a retrievable digital resource, it can be shown in a *content browser*. In most cases a regular web browser is suitable as a content browser, at least as an intermediate step in order to launch a better suited application.

Inspecting Further Information Presenting labels on concepts and concept-relations is nice but in general just represents “a scratch on the surface”. For example, imagine that we are using the RDF version of the metadata standard Dublin Core [1] to express e.g. title, description, creator, creation date, subject, relations, and rights. The title is obviously suitable to use as a label and the relations can clearly be shown as conceptual relations. A context-map can, if needed, present the other fields in a graph manner as well, e.g. the date as a relation to a literal. However, in most cases it is more suitable to present metadata fields with string values as pure text in a form.

A set of fixed forms, to cover all possible situations, is not flexible enough. Instead we rely on the SHAME framework[16][4] to generate forms from small form-snippets called *formlets*. Much like how Conzilla separates information from its presentation SHAME formlets uses *queries* to capture elements of the RDF-graph and *form templates* to generate actual forms. Which formlets that should be used in a certain setting might be specified statically in the context-map or triggered via a type as specified in a stylesheet. Furthermore, by manually switching among available forms and relevant RDF graphs a more full investigation is possible. Another approach, yet to be perfected, is to let relevant formlets be automatically detected from the information itself.

In the editing mode of Conzilla, SHAME is also used as an metadata editor, see figure 3.

4.3 Style Layer

A *style* describes the apperance of boxes and lines, typically their form, linetype, linewith, text alignment etc. A *local style* is a style applied to a specific layout of a given resource. A *global class style* is a style that is applied to an RDF Class or an RDF Property. A *global instance style* is a style that is applied to a specific resource, independently of context. More specifically, a global class style applies to *all* concepts or concept-relations that are expressed as instances of the RDF Class or RDF Property in this global class style. When detecting instances, we also take into account RDF Schema information, i.e. *subClassOf* and *subPropertyOf*. OWL ontologies are not yet taken into account because it is still unclear to us how they relate to the scope of context-maps, which relies heavily on an open-world assumption.

If a local style and a global style are simultaneously relevant, then the local style takes precedence. Observe that we have to override all parts of a style explicitly. As an example, if we provide a local style of a concept, which changes the label alignment but not the form of the box, this form will be determined by the global style - if one has been provided. Moreover, Conzilla offers a “fallback style” that applies if there is no other style that does so.

[16] Documentation for SHAME can be found at `http://kmr.nada.kth.se/shame`

The present lack of an intermediate style level means that a context-maps cannot be associated with a specific and reusable set of styles. Of course you could force inline styles everywhere but that is not a recommended approach. Instead we plan to introduce a *style set* which would be the equivalence of a CSS style sheet. A style set would need to have a selector construction - similar to the ones in CSS or preferrably GSS [14]. A style set would override a global style, but be overridden by an local style.

Currently the style information makes use of a fixed set of hardcoded graphical primitives.

5 Context-maps Expressed in RDF

In this section we will consider the RDF-expressions of the three layers in more detail. This will also include some nitty-gritty details of how RDF represents information in the information layer and how we can refer to that information via referring to RDF-constructs. In order to avoid confusion, we will refer to the RDF-expression of the information as *information triples*, to the RDF expression of context-maps as the *presentation triples* and to the RDF expression of styles as *style triples*.

5.1 General Thoughts on the Context-map Construction in RDF

There are several reasons why we have chosen to express context-maps in RDF. First, this enables good integration with the information triples using internal referencing techniques such as URIs and the reification mechanism. Second, it allows inference engines to easily make use of the combination of information and presentation triples. 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 information”.

An important feature of context-maps is that they are able to present information without changing it. There are several reasons for this. The simplest reason is that you may have only read access to the information triples. Hence, it is neccessary that the presentation triples can be located in other RDF-graphs than the information triples. Equally important is that the presentation triples of the context-maps do not express anything that would change the semantics of the information triples. This is important if the two graphs are to be stored together or managed by tools without prior knowledge of context-maps. Observe that if the intention of the user is to express information that adds to - or changes - the semantics of existing information, he or she should of course be allowed to do this. But the presentation triples of context-maps should not do this automatically by their mere existence.

From this we clearly recognize the need for references to RDF-constructs across the borders of RDF-graphs.

5.2 Referring To Resources and Triples

As pointed out above, the layouts of a context-map should refer to the information triples. But we also need to reference the resources that are spoken about in the triples. Lets first note that a resource is something that is in general outside of RDF, it is merely referenced by a URI. You have to use a domain specific interpretation function[7] to get from the URI to the actual resource. This is quite natural since URIs can denote anything from e.g. a car to the idea of a perfect circle.

Hence, a layout-resource references a resource simply via its URI just like how the information triples references it. However, since we actually are interested in information around a resource as well as the resource itself we have to add a reference to the *container* where the information triples is stored. Currently the reference to the container is calculated[17] rather than explicitly stored on the layout or context-map.

triples, by default, have no identifiers, instead a layout-triple refers to a reification which in RDF is a standardized and identifiable representation for a specific triple[18]. Since a resource may be presented by several layout-resources, the layout-triple must indicate which layout-resources it refers to. Obviously, the layout-triples indicated layout-resources should match the ends of the reification referred to by the layout-triple, see Figure 4. If they do not match, the layout-triple is incorrectly constructed. 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 information triples, and then we must interpret the triple in order to get to the information that it expresses.

5.3 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 should try to find some way to refer to them from layouts.

Unfortunately, there is no perfect solution in standard RDF unless we force the presentation and information triples to be stored together. However, since the requirement to be able to keep them separate is vital, we choose to live with imperfect solutions and encourage people to use the integrated form based approach based on SHAME for displaying anonymous nodes and literals in connection with non anonymous resource. In appendix A we scetche an approach which have been partially implemented in Conzilla as of when this paper was published.

[17] In some cases this is not possible, it remains to define an unambigous scheme for when it has to be expressed and when it can be calculated.

[18] In a given RDF graph there is only one triple with a given subject, predicate and object.

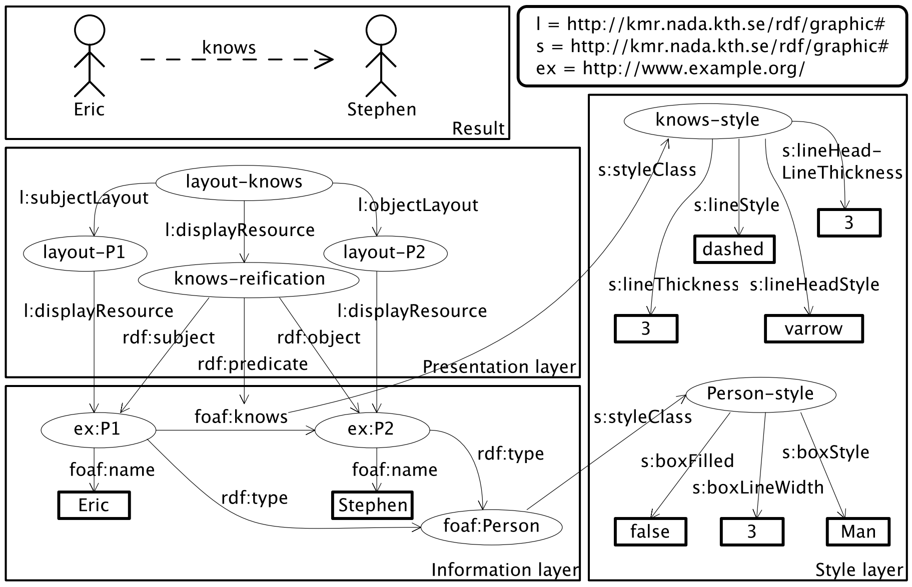


Fig. 4. An overview of the three layers (information, presentation and style) behind the fact that “Eric knows Stephen”.

5.4 Style expression

Figure 4 shows two different global class styles - one for the RDF Class foaf:Person and one for the RDF Property foaf:knows. The RDF-expressions for a global instance style and local style are almost identical. The only difference is that they are connected via the styleInstance property to the corresponding resources and layouts. The RDF-design of style sets remain to be developed.

6 Conclusions and future work

In this paper we have shown how context-maps can be designed and implemented (in Conzilla2) in order to effectively present information expressed on the semantic web. The context-maps have been designed and implemented in a manner that allows WYSIWYG editing. Moreover, the context-maps have been expressed via a three-layered solution, where the lowest layer - the information layer - has been allowed to remain independent of the other two layers - the presentation layer and the style layer. In the presentation layer, the author is allowed to combine several sources into one or several maps, as well as to customize the layout and to specify the navigation through hyperlinks or contextual neighbourhoods. The style layer provides the final touch - together with the form-based metadata displays that are leveraged by the SHAME framework.

Future work will include the investigation of a better design for styles that are bound to maps, which we presently think of in terms of style sets. We will also think more about how to reference anonymous resources and literals. We also want to provide integration with RDF-based storage-and-access solutions like SCAM and Sesame. Also, in order to support inference-type business rules, the relations to OWL will be worked out, and in order to enable process management, a workflow engine will be interfaced. And of course, the interface of Conzilla2 needs improvements, which will be achieved through user-testing and feedback. We also plan to develop thin clients for use in browsers or mobiles. In fact, preparatory work in this direction is already under way.

References

1. The Dublin Core Metadata Initiative. `http://dublincore.org`.
2. D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.
3. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF.
4. H. Eriksson. Query Management For The Semantic Web. `http://kmr.nada.kth.se/papers/SemanticWeb/CID-216.pdf`.
5. K. Graham and J. J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`.
6. T. Gruber. Every Ontology is a treaty - a social agreement - among people with some common motive in sharing. AIS Sigsemis Bullenting 1(3), October 2004.
7. Patrick Hayes. RDF Semantics. `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`.
8. 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).
9. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. EDUTELLA: A P2P Networking Infrastructure Based on RDF. In *Proceedings of the 11th World Wide Web Conference*, 2002.
10. Mikael Nilsson. The conzilla design - the definitive reference. `http://kmr.nada.kth.se/papers/ConceptualBrowsing/conzilla-design.pdf,2000`.
11. Matthias Palmér, Ambjörn Naeve, and Fredrik Paulsson. The scam framework: Helping semantic web applications to store and access metadata. In *ESWS*, pages 167–181, 2004.
12. F. P. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. `http://www.w3.org/TR/2004/REC-owl-semantics-20040210/`.
13. S. Pepper. The TAO of Topic Maps. `http://www.ontopia.net/topicmaps/materials/tao.html`.
14. E. Pietriga. Graph Stylesheets (GSS) in IsaViz. `http://www.w3.org/2001/11/IsaViz/gss/gssmanual.html`.
15. A. Sheth. The Informations Systems Perspective on Semantic Web Research. AIS Sigsemis Bullenting 1(1), April 2004.

Appendix - An Approach to Referencing Literals and Anonymous nodes

By definition, *anonymous* resources have no identifiers outside of the RDF-graph where they occur. Therefore an anonymous resource cannot be referenced directly - except from triples in the same RDF-graph. If we want to refer to an anonymous resource, we need to invent an indirect referencing technique. Below we describe an approach called *graph-patterns* that captures anonymous resources and literals as a special case.

Referencing Anonymous Resources We will here define and state some findings without proof, a more formal treatment remains to be done.

Def: The *marked-graph* of a anonymous resources A consists of the RDF-graph where A occurs, and an extra marker-triple wherein A is the subject[19]. It follows that two anonymous resources are *indistinguishable* if their marked-graphs are *isomorphic* (defined in [7]). Even though it is not always possible, it is nevertheless interesting to consider the graph-pattern[20] that provides the most detailed matching of an anonymous resource.

Def: for an anonymous resource A, we say that a graph-pattern is *complete* relative to A, if the graph pattern captures the *anonymous closure*[21] of A. It can be shown that a complete graph pattern for A matches A - as well as each of its indistinguishable anonymous resources - which is the best you can expect under these circumstances.

Moreover, small independent changes to the surrounding RDF-graph should not invalidate the graph-pattern and break the references to its anonymous resources. From this perspecitve we can identify two inherently conflicting requirements on graph-patterns:

1. If we want to reference anonymous resources uniquely, the best we can do is use complete graph-patterns.
2. If we want to minimize the risk of broken references, a smaller graph-pattern reduces this risk.

It should be noted that in most practical situations complete graph-patterns are quite small and hence the conflict between the two requirements is neglible. A second somewhat weaker approach, which in most practical situations coincides with complete graph-patterns, is to use graph-patterns that capture all incoming and outgoing triples. A third - and even weaker - approach, is to use graph-patterns that only capture the incoming and outgoing triples that are shown in the context-map from where the anonymous resource is being referenced. In the last approach, the context-map itself can function as the graph-pattern, no secondary expression is neccessary. Whenever it is not enough to reuse context-maps as graph-patterns, an external query language is needed. The Edutella Query Language (QEL) [9] is a good alternative. In fact, the RDF-expression of QEL uses reifications that refer to *variables* instead of fixed resources, which is precisely how we will reference anonymous nodes in the triple- and resource-layouts described in section 5.2.

[19] the predicate and object should not have been introduced in the RDF-graph already.

[20] a graph-pattern is a query where the required anonymous resource is captured in a specific variable

[21] i.e. reachable graph from a node where only anonymous nodes may be traversed.

Referencing Literals Referencing literals constitutes a special case of referencing anonymous resources. In the case where you do not have an anonymous resource as the subject of the triple where the literal is expressed as object, the graph-pattern will be comparable to a reification. Moreover, if literals change often compared to resource URIs it is a bad idea to rely on exact string matching of the literal in the graph-pattern. The simplest solution is to replace the literal with a variable. However, this does not work when there are several triples that differ only in their literals. In such cases a constraint could be added on the variable in order to distinguish the literal in question. However, this approach would have to rely on heuristics and further investigation is needed to investigate whether it would be worth the effort.