M-FIRE: A Metaphor-Based Framework for Information Representation and Exploration

Matteo Golfarelli, Andrea Proli, and Stefano Rizzi

DEIS, University of Bologna
Via Risorgimento 2, Bologna, Italy
{mgolfarelli,aproli,srizzi}@deis.unibo.it

Abstract. An open problem in the construction of an environment for visualizing and navigating information in the context of the Semantic Web is to guarantee a satisfactory compromise between expressivity and domain-independence. In this paper we first introduce M-FIRE, a configurable framework for instantiating visualization and navigation systems based on the adoption of custom metaphors: metaphors drive the process for obtaining a visual representation of a given piece of information and define how queries are generated upon user actions. Then, the paper describes in detail how presentation is achieved. The possible applications for our framework range from semantic browsing to ontology-enabled Web site design.

Keywords: Ontology, visualization, navigation, metaphor, RDF.

1 Introduction

The Semantic Web vision is built upon the ability of formally defining and processing the semantics of knowledge. Knowledge definition languages of increasing expressivity, allowing for increasingly sophisticated (and computationally complex) reasoning, have been developed by the W3C organization: they are structured as a layered tower of languages, where each layer builds on the lower one as its *semantic extension* (W3C, 2004). At the bottom of the tower is RDF, which allows for the assertion of *statements* about *resources* and their *properties*.

Since expressive languages like RDF Schema (RDFS) and OWL are semantic extensions of RDF, RDFS and OWL documents share the same syntax and structure of RDF documents. Though the syntax of RDF is designed to be human-readable, most end-users are not familiar with it, and (most important) neither they are with the semantics of abstract concepts like 'restriction', 'specialization', 'cardinality', and so on: because of this, they should be provided with tools that (1) translate low-level statements into easy-to-interpret visual renderings and (2) translate user actions performed on those visual renderings into low-level queries over the underlying knowledge base.

An open problem in the field of visualization and navigation of RDF documents is to guarantee a satisfactory compromise between expressivity and domain-independence. The former is meant as the capability of delivering an intuitive representation of knowledge and some tailored navigation primitives to end-users working in a given application domain, while the latter is aimed at accomplishing a high degree of reusability.

J. Filipe, J. Cordeiro, and V. Pedrosa (Eds.): WEBIST 2005/2006, LNBIP 1, pp. 214–227, 2007.

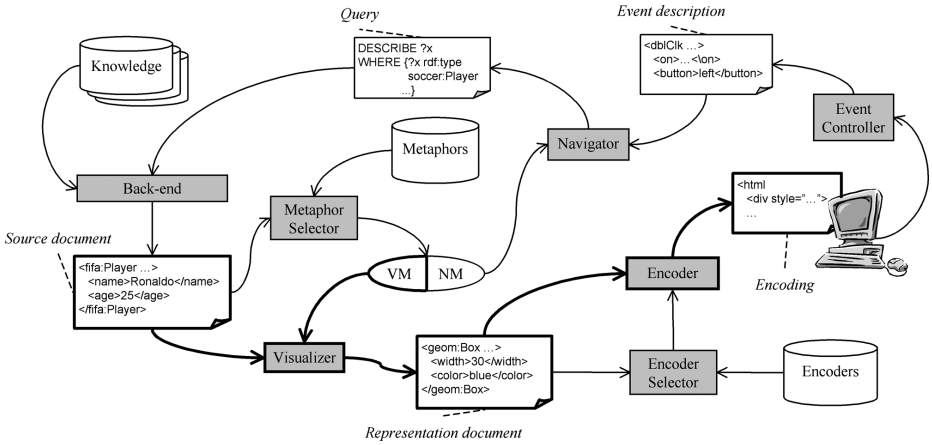

Fig. 1. Overall functional architecture of M-FIRE

Most existing tools favor domain-independence, and represent entities in a way that is closer to the abstract form used to formally define them: in fact, they adopt visual items to represent things – such as classes, properties, specializations, and instantiation relationships – that are familiar to knowledge engineers (a narrow category of end-users) but not to domain experts. Indeed, though domain-specific formalisms have a lower degree of reusability, they provide graphically richer constructs allowing for a representation that is closer to how entities appear in the application domain.

An approach to address this issue is to decouple the mechanism for transforming RDF documents into an expressive representation from the criteria that drive the representation itself. In this paper, we present M-FIRE, an original, configurable framework for easily instantiating visualization and navigation systems for RDF-based knowledge, relying on the adoption of custom *metaphors*. Metaphors drive the process through which visual representations are obtained for a given document, and define how queries are generated upon user actions. This allows users to perform semantic browsing by relying on intuitive concept representations and to interact in a simple manner with complex knowledge.

The overall functional architecture of our framework is sketched in Figure 1. First of all, we postulate the existence of a *back-end* module that, given a query in a supported query language, returns a result as an RDF document (from now on, the *source document*). The *metaphor selector* component takes the source document and returns the best suited metaphor for its visualization (*visualization metaphor*, VM in Figure 1) and navigation (*navigation metaphor*, NM in Figure 1); many criteria could drive this choice, for instance the vocabulary of the source document. The chosen visualization metaphor is then given as input to the *visualizer* module, which applies the directives contained in the visualization metaphor to generate a *representation document* for the source document; the representation document describes the visualization to be produced in an abstract form, independently of any implementation detail. Then, a

properly chosen *encoder* translates the representation document into a concrete form, called *encoding* (e.g., an HTML document), which can be given as input to the end-user's visualization program (e.g., a Web browser). The choice of the best suited encoder for a given representation document is carried out by the *encoder selector* and could depend, again, on different decision criteria. The reason for this two-layer visualization architecture (visualization followed by encoding) is to obtain flexibility and independence with respect to any visualization device, program, and language.

Once rendering has been completed by the visualization program, end-users are allowed to interact with the produced visualization. Events generated by user actions are captured by the *event controller*, which creates an *event description* in the form of an OWL document describing the occurred event (for instance, a user's double click on an icon representing a soccer player). The event description is then given as input to the *navigator* module together with the chosen navigation metaphor. In the same way as the visualization metaphor tells the visualizer which representation must be produced for a given RDF document, the navigation metaphor tells the navigator which query must be formulated for a given event. The resulting query is then forwarded to the back-end, and the process is repeated.

In this paper we focus on presentation, meant as the process through which an encoding is generated starting from the source document returned by the back-end. Such path is tracked in Figure 1 by means of thick lines.

2 Related Work

Several general-purpose tools currently exist, that highlight the semantics of a few basic terms from the RDF(S) or OWL vocabulary only (e.g., instances of rdfs:subClassOf, instances of rdfs:Class, and so on) and allow users to perform simple navigation actions, mainly resulting in graph node expansion. This is a reasonable limitation if we consider that those formalisms are intended to give a presentation for a wide range of RDF-based documents, no matter what their application domain is. However, they fail in offering end-users intuitive representations of the described objects and concepts, thus missing the main goal of our approach. The front-ends of well-known ontology management systems such as KAON (Volz et al., 2003), Spectacle (Harmelen et al., 2001), Ontolingua (Farquhar et al., 1995), Ontosaurus (Swartout et al., 1996), Ontorama (Furnas, 1986), and WebOnto (Domingue et al., 1999) all fall into this category; the same also applies to Protégé plug-ins for ontology visualization and navigation, except for Jambalaya (Storey et al., 2001).

Jambalaya is a significant step toward the development of a highly configurable tool, in that a visualization for the semantics of a given piece of information is generated according to user-specified parameters. In Jambalaya, graphical containment is used, by default, to encode the semantics of both sub-classing and instantiation, but users are allowed to modify this behavior by associating whatever property they prefer to the graphical containment drawing primitive. Our approach generalizes the one of Jambalaya one by allowing any kind of semantic structure to be rendered by any kind of graphical structure.

In the context of the W3C's IsaViz visual environment for browsing and authoring RDF documents[1], a language called Graph Stylesheets (GSS) is defined as a way to associate style to node-edge representations of RDF graphs and to offer alternative layouts for some elements. Actually, with respect to our approach, GSS plays three roles at the same time: it resembles representation metaphors because it allows to associate graphical styles to some semantic patterns; it can also be seen as a representation language, since it defines a vocabulary of graphical styles interpreted by a rendering engine; last, it can be classified as an encoding language because it directly feeds a visualization program (the IsaViz tool). Our approach aims at disambiguating this multifold nature by introducing a clean separation of each aspect, and generalizes GSS in that representations are not limited to graph-based drawing primitives.

FRESNEL[2] is a simple vocabulary for specifying which parts of an RDF graph are displayed and how they are styled using existing style languages like CSS. There are analogies between M-FIRE and FRESNEL, but also significant differences. First, the representation paradigm of FRESNEL is centered on *resources*, while the one of M-FIRE is centered on *statements*: this means that in FRESNEL representations are generated for individuals, while in M-FIRE representations are associated to (sets of) RDF triples. The approach based on statements is more general, because the representation of an existing resource X can always be obtained as the representation of the statement X rdf:type rdf:Resource, while representing an arbitrary statement in the resource-based approach requires reification. Moreover, M-FIRE allows the same graphical drawing to be the representation of more than one statement: for instance, a picture may represent both the fact that a person is a soccer player and the fact that he plays in a particular soccer team. Besides, in M-FIRE navigation metaphors are independent from representation, while the two aspects are not well separated in FRESNEL. Finally, FRESNEL comes with a built-in vocabulary for formatting displayed information in a browser-independent way, while M-FIRE, as a pure framework, does not commit to any graphical vocabulary and relies on the existence of a proper encoder capable of translating the produced representation into the chosen format.

3 Visualization

We now provide a simplified definition of an RDF document which is useful for our purpose of detailing how the framework works.

Definition 1 (RDF document). *A* statement *is a triple* $\langle subj, pred, obj \rangle$ *where the* subject $subj$ *is a* resource *identified by a* URI *(Berners-Lee et al., 1998), the* predicate $pred$ *is a* property *identified by a URI too, and the* object obj *is either a resource or* literal *(e.g. a string). An* RDF document d *is a set of statements, and its* vocabulary*, denoted by* $Voc(d)$*, is the union of the set of the URIs and literals appearing as the subject, object, or predicate of its statements.*

Visualization is the process of obtaining a representation document in which certain graphical drawings are associated to certain (kinds of) statements belonging to the

[1] http://www.w3.org/2001/11/IsaViz/

[2] http://www.w3.org/2005/04/fresnel-info/

source document, according to the directives contained in a visualization metaphor. In other words, disjoint subsets of the source document d_s are defined such that statements in the same subset are represented by instantiating the same type of representation. The problem of classifying statements in d_s raises expressiveness and tractability issues concerning the complexity of queries which can be formulated to select them.

In order to provide a very general solution, visualization in M-FIRE is conceived as a two-step process. Although statement selection is performed through conjunctive queries over RDF triples by relying on a structural pattern matching engine without any reasoning capability (see Subsection 3.2), we allow for a sort of preprocessing step, called *enrichment* and described in Subsection 3.1, during which the source document is augmented with new concept definitions of arbitrary complexity; reasoning w.r.t. such concept definitions allows to infer useful classifications for existing resources in d_s, which are then exploited to formulate expressive statement selections.

Thus, the visualizer module shown in Figure 1 actually consists of two separate modules: the *enricher*, which augments the source document with new classifications and concept definitions for obtaining an *enriched document*, and the *representer*, carrying out the association of particular visual items and graphical styles to certain kinds of statements in the enriched document, in order to produce the representation document (an RDF document describing the drawing that must be presented to the end-user). Both the new concept definitions and the styling instructions come from the metaphor: indeed, parallel to the above separation, visualization metaphors are divided into two components, namely the *enrichment metaphor*, which drives the transformation of the source document into the enriched document, and the *representation metaphor*, which drives the production of a representation document for the enriched document. The vocabulary of the representation document is called the *representation vocabulary* and is not bound to any predefined set of URIs (this will be further discussed in Section 4).

3.1 Enrichment

An enrichment metaphor em is a pair $\langle o_{em}, r_{em} \rangle$, where o_{em} is an OWL document containing concept definitions and r_{em} is a compatible reasoner providing classification services. Ontology o_{em} can only contain concept definitions of the form $A = C$, where A is a concept name and C is a complex concept defined through concept constructors provided by the language that r_{em} supports.[3] The enricher merges o_{em} to the source document and lets r_{em} add new classifications to produce the enriched document d_e.

Example 1. Let d_s be the source document in Figure 2(a), and let o_{em} be the OWL ontology in Figure 2(b). If r_{em} is an OWL-DL reasoner, then the enriched document d_e will include both statements in $(d_s \cup o_{em})$ and ⟨soccer:ply01 rdf:type soccer:Goalscorer⟩. □

As previously discussed, visualization is split into two phases because the query language used to select those statements for which a certain representation must be drawn

[3] This limitation is necessary to enable query unfolding during navigation, which is out of the scope of this paper.

(*a*)	(*b*)
soccer:ply01 rdf:type soccer:Player, soccer:ply01 soccer:hasName “Zlatan Ibrahimovic”	soccer:Goalscorer rdf:type owl:Restriction
soccer:ply01 soccer:playsIn soccer:juve, soccer:juve soccer:hasNation soccer:cty07	soccer:Goalscorer owl:onProperty soccer:hasScored
soccer:ply01 soccer:hasNation soccer:cty15, soccer:cty15 soccer:hasFlag “Sweden.bmp”	soccer:Goalscorer owl:minCardinality 1
soccer:cty07 soccer:hasFlag “Italy.bmp”, soccer:ply01 soccer:hasScored soccer:goal15A4	

Fig. 2. Sample source document from the soccer domain (a) and enrichment ontology for the same domain (b)

might have limited expressiveness. Our implementation of the representer module is based on SPARQL[4]; as a plain RDF query language, SPARQL does not support any kind of non-trivial reasoning. Should SPARQL interpreters have a built-in support for reasoning with (suppose) OWL-DL vocabulary, enrichment could be unnecessary, as implicit classification could be inferred by the reasoner.

Besides, moving the reasoning step out of the representation process into the enrichment phase enables the support of new languages (or new reasoners) by simply defining different metaphors. Thus, a crisp separation between enrichment and representation gives metaphor designers a better control over the kind of inferences and classifications that are performed, also increasing the flexibility and the modularity of the design.

3.2 Representation

Representation is the process of associating proper graphical structures to certain semantic structures, and is carried out by the representer. More precisely, the representer produces a representation document for the enriched document d_e, where sets of resources in the former represent single statements in the latter, by interpreting the directives contained into the representation metaphor. A representation metaphor is a pair $rm = \langle R, F \rangle$, where R is a set of *representation rules*, and F is a set of *fusion rules*. In order to provide a formal account for the semantics of representation and fusion rules, we need some auxiliary definitions that recall the usage of graph patterns and templates in SPARQL. Due to space limitations, however, we present the syntax for expressing rules through illustrative examples instead of providing the rigorous grammar definition.

Definition 2 (RDF Document Pattern). *A* statement pattern *is a statement where variable names can appear instead of URIs and literals as the subject, the object, and the predicate. The set of variable names appearing in a statement pattern sp is denoted by $Var(sp)$. An* RDF document pattern *dp is a set of statement patterns, and we define $Var(dp) = \bigcup_{sp_i \in dp} Var(sp_i)$. Given an RDF document pattern dp and an RDF document d, we say that d* matches *dp iff there exists at least one $d' \subseteq d$ and a mapping $\beta : Var(dp) \rightarrow Voc(d')$ such that d' is obtained from dp by replacing each variable name $v \in Var(dp)$ with $\beta(v)$. Mapping β is called a* binding*, and d' is a* solution for *dp* in *d.*

Definition 3 (RDF Document Template). *A* statement template *is a statement pattern where new names used as* resource templates *can appear in place of URIs and variable*

[4] http://www.w3.org/TR/rdf-sparql-query/

(*a*)	(*b*)
REPRESENTATION RULE PlayerInItaly	REPRESENTATION RULE Nationality
FOR PATTERN { ?x rdf:type soccer:Player . ?x soccer:playsIn ?y .	FOR PATTERN { ?a misc:hasNation ?b . ?b misc:hasFlag ?c . }
?x misc:hasName ?z . }	?c misc:hasSourceFile ?d . }
WITH { #b rdf:type geom:Box . #b geom:hasColor "Blue" . #b geom:hasText ?z . }	WITH { #i rdf:type geom:Img . #i geom:hasSrc ?d . #i geom:nextTo #p . }

Fig. 3. Two sample representation rules for the soccer domain

(*a*)	(*b*)
geom:b1 rdf:type geom:Box	geom:i1 rdf:type geom:Image
geom:b1 geom:hasColor "Blue"	geom:i1 geom:hasSource "Sweden.bmp"
geom:b1 geom:hasText "Zlatan Ibrahimovic"	geom:i1 geom:nextTo geom:p1

Fig. 4. Partial representations generated by rules in Figure 3(a) and 3(b) when applied to Example 1

names as the subject and the object only. The set of resource templates appearing in a statement template st is denoted by $Tem(st)$. An RDF document template *dt is a set of statement templates, and we define $Tem(dt) = \bigcup_{st_i \in dt} Tem(st_i)$. Given an RDF document template dt, an RDF document d is said to be an* instance of *dt iff there exists a mapping $\tau : Tem(dt) \rightarrow U$, where U is the set of URIs contained in $Voc(d)$, such that d is obtained from dt by replacing each resource template $t \in Tem(dt)$ with $\tau(t)$. Mapping τ is called the* instantiation function *from dt to d.*

Intuitively, representation rules in R are used to create one *partial representation document* for each set of statements in d_e matching a particular document pattern, while fusion rules in F properly merge multiple such documents whenever a condition expressed over the sets of statements they represent is satisfied.

Definition 4 (Representation Rule). *A* representation rule *is a pair $r = \langle ss, rt \rangle$, where ss (*statement selector*) is a document pattern, and rt (*representation template*) is an RDF document template, with $Var(rt) \subseteq Var(ss)$.*

Example 2. With reference to the enriched document from Example 1, representation rule PlayerInItaly *in Figure 3(a) defines the visualization of soccer players who play in an Italian team, while rule* Nationality *in Figure 3(b) defines a visualization for the nationality of a person, a team or anything else. The first rule generates, for each graph describing the fact that a soccer player with a certain name plays in an Italian team, an RDF document describing a blue box which contains his name; the second one represents the fact that a person (or a team) belongs to a nation which is symbolized by a flag, and whose picture is stored in a file, by generating a description where the picture of the flag is placed next to the person's representation.* □

Clauses FOR PATTERN and WITH in Figure 3 denote, respectively, the statement selector and the representation template for a representation rule. Names beginning with '?' and '#' are, respectively, variables and resource templates.

The semantics of representation rules can be described by illustrating how the representer exploits them for generating partial representation documents. Iteratively, a representation rule r is extracted from R, and the set Z_r of solutions for $r.ss$ in the enriched

document d_e is computed. For each solution $z \in Z_r$, there exists exactly one set of statements in d_e that matches $r.ss$. The binding that corresponds to z, say β_z, is then used to transform the representation template $r.rt$ into a matching RDF document template t_z by replacing each variable name $v \in Var(r.rt)$ with $\beta_z(v)$ (with reference to Example 2, statement ⟨#b geom:hasText ?y⟩ becomes ⟨#b geom:hasText "Zlatan Ibrahimovic"⟩). Thus, for every solution z, we obtain an RDF document template t_z. An instance is then created for all such t_z, where the images of the corresponding instantiation functions are pairwise disjoint (thus producing, e.g. geom:123 geom:hasText "Zlatan Ibrahimovic").

The procedure is repeated until all of the representation rules in R have been processed.

Example 3. The partial representation document shown in Figure 4(a) is generated by rule PlayerInItaly *in Figure 3(a), and the partial representation document in Figure 4(b) is generated by rule* Nationality *in Figure 3(b), when applied to the enriched document obtained in Example 1.* □

This way, for each representation rule $r \in R$, many partial representation documents are instantiated (one for each set of statements in d_e matching the statement selector of r). Once all $r \in R$ have been processed, fusion rules come into play.

Definition 5 (Fusion Rule). *A* fusion rule *is a triple $f = \langle S, fp, ft \rangle$, where the* fusion set *S is a multiset containing representation rules in R, fp is an RDF document pattern called the* fusion pattern*, and ft is an RDF document template called the* fusion template*. Variables in the fusion template must also appear in the fusion pattern: formally, $Var(ft) \subseteq Var(fp)$.*

The representer uses fusion rules to link partial representation documents, among those created by representation rules in S, whose represented sets of statements meet the join conditions expressed in the fusion pattern *fp*. A helpful analogy can be set up with relational algebra, where a join operator merges several tables (and the same table can be included multiple times with different roles in the FROM clause) much like a fusion rule merges the existing partial representation documents generated by a number of representation rules; the difference here is that applying fusion rules leads to the creation of new sets of statements linking the partial representation documents, whereas a join operation in relational algebra produces a mere concatenation of tuples, without new information being created. Representation rules in S thus correspond to the joined tables, and the partial representation documents generated by them correspond to the instances of those tables (tuples).

Fusion pattern *fp* (corresponding to the join predicate in our analogy with relational algebra) can refer variables used in the statement selectors of any representation rule in S, thus allowing to express a cross-representation condition, involving their represented statements, that must be satisfied in order to instantiate *ft*; similarly, fusion template *ft* can refer resource templates used in the representation templates of any representation rule in S, allowing to create a connection among the partial representation documents previously produced (see Example 4 below).

Example 4. The following fusion rule establishes an identity equivalence between the blue box generated by representation rule PlayerInItaly *and the graphical placeholder*

```
// d_s is the document to represent, vm is the visualization metaphor
// This function returns the final representation document
RDFDocument visualizer(d_s,vm) {
    RDFDocument d_e = enricher(d_s, vm.em);
    RDFDocument r = representer(d_e, vm.rm);
    return r;
}

// d_s is the source document, em is the enrichment metaphor
// This function returns the enriched document
RDFDocument enricher(d_s,em) {
    RDFDocument d_merge = merge(em.o_em, d_s)
    RDFDocument d_e = em.r_em.inferClassifications(d_merge);
    return d_e;
}

// d_e is the enriched document, rm is the representation metaphor
// This function returns the final representation document
RDFDocument representer(d_e,rm) {
    Set A = ∅; // Stores semantic annotations
    for each r ∈ rm.R { // Apply representation rules
        Set Z_r = match(d_e, r.ss); // Stores the matching solutions
        for each z ∈ Z_r { // A solution is a set of represented statements
            Binding β_z = z.getBinding();
            RDFDocument repr = β_z.bind(r.rt).instantiate();
            // Associates a partial representation to the set of statements
            // it represents, and to the rule it was generated by.
            Annotation a = new Annotation(repr, z, r); A = A ∪ a;
        }
    }
    for each f ∈ rm.F { // Apply fusion rules
        for each ⟨a_1, ..., a_|f.S|⟩ ∈ A^|f.S| { // Assume a_i.rule = f.S_i
            Set Z_f = match(⋃_{i=1,...,|f.S|} a_i.solution, f.sp);
            for each z ∈ Z_r { // Instantiate the fusion template
                Binding β_z = z.getBinding();
                RDFDocument repr = β_z.bind(f.ft).instantiate();
                Annotation a = new Annotation(repr, z, f); A = A ∪ a;
            }
        }
    }
    RDFDocument repr_final = ∅;
    for each a ∈ A { // Merge the created partial representations
        repr_final = merge(repr_final, a.representation);
    }
    return repr_final;
}
```

Fig. 5. The visualization algorithm

generated by the #p *resource template in representation rule* Nationality*, whenever their represented set of statements describe, respectively, the fact that a person is a soccer player in an Italian team, and the fact that the same person belongs to a certain nation. As a result, taking into account the semantics of the* owl:sameAs *property, the two graphical objects described by those resources are identified as one – so, practically, they are merged.*

```
FUSION RULE PlayerWithNationality
JOINS PlayerInItaly AS P, Nationality AS N
WHEN { P.?x mfire:sameAs N.?a . }
GENERATES { P.#b owl:sameAs N.#p . }
```

The above fusion rule specifies how to graphically link any two partial representations, generated by rules PlayerInItaly *and* Nationality*, that represent two corresponding sets of statements where the former describes a soccer player and the latter describes his nationality. The statement pattern* P.?x mfire:sameAs N.?x *is intended to ensure that, for two given partial representations obtained by the above representation rules, the fusion template is instantiated only if the resource that was bound to variable* ?x *during the generation of the former actually is the same resource that was bound to variable* ?a *during the generation of the latter. Identifiers* P.#b *and* N.#p *denote the resources that were generated, in the corresponding partial representations, as instances of resource templates* #b *and* #p*, respectively.* □

In Example 4, clause JOINS denotes the fusion set S, clause WHEN defines the fusion pattern fp, and clause GENERATES defines the fusion template ft. From a high-level perspective, the application of fusion rules does not differ significantly from the application of representation rules: iteratively, a rule f is extraced from F and a corresponding set of $|f.S|$-ples is computed, such that the elements of each tuple are partial representation documents that were generated by rules in $f.S$, and that match the join condition expressed by $f.fp$. For every such tuple, the corresponding set of solutions and related bindings is found, and the RDF document template $f.ft$ is eventually instantiated.

Example 5. With reference to the partial representation documents obtained in Example 3, the application of the fusion rule PlayerWithNationality *in Example 4 yields to the generation of the following partial representation document, consisting of a single RDF statement:*

geom:b1 owl:sameAs geom:p1 □

Finally, after all fusion rules have been processed, partial representation documents created by representation and fusion rules are all merged together in order to obtain the final representation document. Figure 5 summarizes the overall visualization process by means of an illustrative algorithm.

4 Encoding

The encoding process is carried out by the encoder module, that is entitled to translate the representation document into a concrete form, i.e., a document that can be parsed by a proper program to produce a visualization.

In principle, many formats could be used to encode a drawing described in the representation document. Two colored circles connected by a dotted, directed edge could be encoded as both a GraphML[5] document and an SVG[6] document; a table containing names and photos could be encoded as an SVG document as well as an HTML document.

[5] http://graphml.graphdrawing.org/
[6] http://www.w3.org/Graphics/SVG/

The choice of the proper encoding can be carried out by considering many criteria, among which user preferences. As users of the Web usually retrieve information by means of a Web browser parsing HTML documents, an HTML encoder could be an option if the drawings described by the representation document can be rendered in HTML. Still, the main decision criterion is, of course, the content of the representation document: if the vocabulary contains instances of a class named geom:DottedArrow (from a fictive geom namespace) which designate edges connecting pairs of objects, it is preferable to drop HTML and to target a visualization program which is able to reproduce graph structures, thus choosing its corresponding encoding format (for instance, GraphML). The encoder selector module solves the aforementioned task: given a set of encoders, it extracts the most suited one according to user preferences, document representation content, and so on.

Formally, an encoder can be defined as a triple $e = \langle p, f, V \rangle$, where p is a program translating the input representation document into the final encoding, f is the target encoding format, and V is the *encoder vocabulary*, i.e. the set of class URIs for which the encoder is able to perform a translation into the target format. Let $E = \{\langle p_i, f_i, V_i \rangle\}_{i=1,\ldots,n}$ be a set of available encoders, and let d_r be a given representation document. Then, a valid decision criterion would be to select the encoder for which the value $|\,Voc(r) \cap V_i|/|\,Voc(r)|$ is maximum, to have the broadest representation vocabulary coverage. Other methods could assign different weights to the URIs in the representation vocabulary, so that some drawing primitives are considered more important than others.

A crucial issue involving the encoding process is *semantic annotation*. Semantic annotation traces a mapping between the graphical items that are used to represent a given set of statements and the represented statements themselves. Such mapping is first established at a conceptual level by the representer (see the pseudo-code in Figure 5), by linking each partial representation document to the set of statements it represents. Annotations are then parsed by the encoder, which has the responsibility of embedding them into the encoding document. There, such information can be exploited by the end-user's visualization program to integrate graphical drawings with their semantics. Notably, an HTML encoder would be able to translate representation documents into automatically annotated (pieces of) Web pages.

5 Implementation

The architecture of our framework has been designed in order to maximize flexibility, reusability and extensibility. As to the visualization process, the framework provides an implementation for the enricher and the representer modules, but only defines the interface for the metaphor selector, the encoder and the encoder selector: visualization systems are instantiated by plugging custom implementations of these components into the framework.

We have built the presentation engine (see Figure 1) as a simple Java program taking as input an RDF document and forwarding it to the metaphor selector program. The metaphor selector returns the most suited visualization metaphor for the source

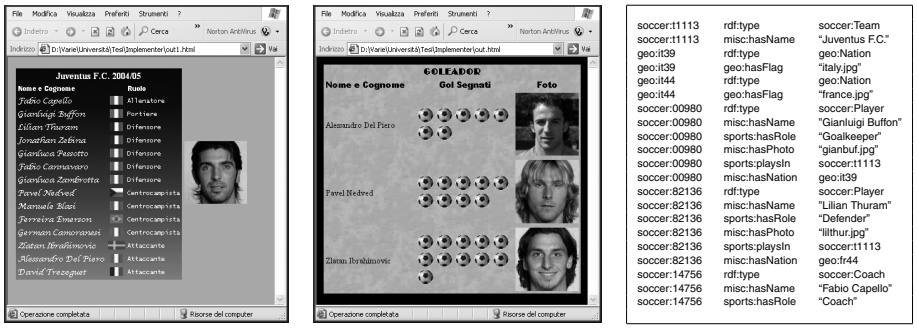

Fig. 6. Rendering of the HTML encoding of two representation documents obtained by applying different visualization metaphors to the same (RDFS) source document containing information about Juventus players, of which an excerpt is shown in (c). In (a), soccer teams are the focus of interest and they are rendered as a list of players (in this case, only Juventus appears); in (b), goalscorers are the relevant information to be highlighted and they are shown together with their score.

document, which is then given as input to the visualizer together with the source document itself. In our prototypical implementation, the metaphor selector does nothing more than presenting the user a list of metaphors, so the decision on which metaphor to apply is actually delegated to the end-user (though, this is neither a desirable nor a realistic behavior). The visualizer is in turn coded as two independent Java programs, implementing the routines listed in Figure 5.

Both the enricher and the representer rely on the Jena[7] library for generic processing of RDF documents; the enricher makes use of Pellet[8] for supporting OWL(-DL and -Lite) reasoning, while the representer module makes additional use of the ARQ[9] SPARQL engine provided by Jena; representation rules are executed by internally translating them into SPARQL queries with a CONSTRUCT clause, which are then parsed by the ARQ interpreter for instantiating the representation templates and generating the partial representation documents.

Since HTML is the standard format used for presenting information across the World Wide Web, we have chosen to implement an HTML encoder. Our HTML encoder is a Java program based on Jena; screenshots in Figure 6(a) and (b) depict the encoding of two representation documents, obtained by applying two different visualization metaphors to the same source document (partly) listed in Figure 6(c).

Since RDF documents can be conceptually understood as graphs, and most tools render them as such, we are currently working on the implementation of a GraphML encoder too.

[7] http://jena.sourceforge.net/
[8] http://www.mindswap.org/2003/pellet/
[9] http://jena.sourceforge.net/ARQ/

6 Conclusions

In this paper we presented M-FIRE, an original approach to RDF-based knowledge visualization and navigation where ad-hoc presentations of contents are generated according to different metaphors. We believe that the strength of M-FIRE lies in its ability to deliver expressive, domain-specific presentations to end-users without affecting reusability, which constitutes a significant advance over existing approaches. Navigation primitives are also expressed by metaphors and complete the framework by providing a unified approach to knowledge fruition.

As to the presentation process, we have shown that the architectural design of our framework enjoys a double degree of reusability and flexibility: during visualization, different metaphors could be applied to the same source document (flexibility), and the same metaphor could be applied to two different source documents (reusability); during encoding, different encoders could be used to translate the same representation document (flexibility), and the same encoder could be used to translate two different representation documents (reusability). Figures 6 and 7 provide an illustrative example of this potential.

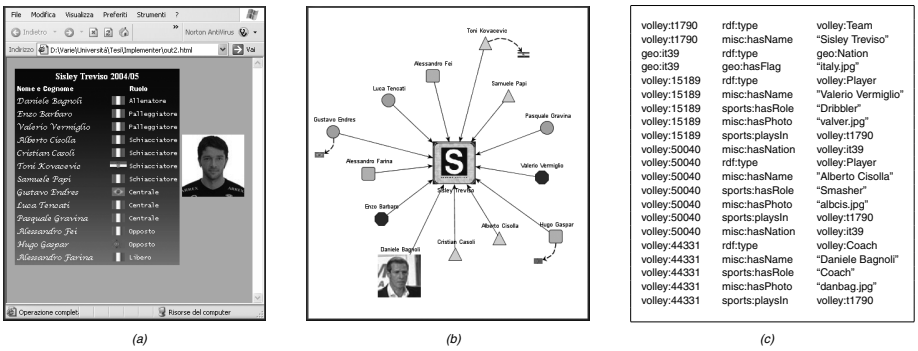

Fig. 7. Rendering of two encodings, obtained by applying different visualization metaphors to the same (RDFS) source document (c) containing information about a volley team. In (a), the same visualization metaphor and encoder were used as in Figure 6(a). In (b), the applied visualization metaphor produced a representation document describing a graph, which was then processed by a GraphML encoder.

Regarding the creation of metaphors, defining representation and fusion rules is perhaps as difficult as writing a SPARQL query: transformations of source documents into representation documents are expressed by means of a declarative language which recalls the CONSTRUCT form of SPARQL queries. Nonetheless, the same visualization metaphor could be defined by different sets of rules, and the quality of design could have an impact on modularity, reusability, readability, and extendibility of the metaphor. We plan to develop a visual tool aimed at assisting metaphor designers in the definition of rules, as well as in building coherent visualization metaphors.

M-FIRE is a very general environment, aimed at supporting several different classes of applications. Among those, ontology browsing is probably the one which most

promises to impact on the deployment of the Semantic Web. A key feature of ontology browsers is that of allowing users to switch between multiple fruition paradigms, either transparently or intentionally; this is accomplished in M-FIRE through metaphors and by plugging different metaphor selectors, that perform either automatic or manual selection of metaphors, into the framework. In case of manual selection, a desirable feature is that of proposing a list of valid metaphors, integrated with legends and previewing functionalities.

Another interesting field of application for M-FIRE, aimed at enhancing the potentiality of current Web technologies, is the support to the development of semantically annotated Web sites. In fact, M-FIRE would allow a domain-dependent translation of RDF documents into HTML encodings to be easily specified. We believe that this would push Web site owners to publish their content by (1) directly implementing knowledge bases on ontology-enabled repositories and (2) translating them into HTML documents whose elements are semantically annotated with the information they represent. Thus, for instance, it would be possible to drag an image from a Web browser and drop it into an ontology editor for exploring the formal description of the resource that image depicts. In general, semantic annotation will allow RDF-aware presentation programs to be integrated with other tools capable of handling the semantics of the underlying information, thus forming a rich environment for knowledge fruition and retrieval.

References

Berners-Lee, T., Fielding, R., Irvine, U., Masinter, L.: Uniform resource identifiers (URI): Generic syntax (1998), http://www.ietf.org/rfc/rfc2396.txt

Domingue, J., Motta, E., Garcia, O.: Knowledge Modelling in WebOnto and OCML: A User Guide. Knowledge Media Institute, Milton Keynes, UK (1999)

Farquhar, A., Fikes, R., Pratt, W., Rice, J.: Collaborative ontology construction for information integration (1995)

Furnas, G.W.: Generalized fisheye views. SIGCHI Bull. 17(4), 16–23 (1986)

Harmelen, F.V., et al.: Ontology-based information visualisation. In: Proc. Workshop on Visualisation of the Semantic Web, pp. 546–554 (2001)

Storey, M. et al.: Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protégé. In: Proc. Workshop on Interactive Tools for Knowledge Capture, Victoria, Canada (2001)

Swartout, B., Patil, R., Knight, K., Russ, T.: Toward distributed use of large-scale ontologies. In: Proc. 10th Knowledge Acquisition Workshop, Banff, Canada (1996)

Volz, R., Oberle, D., Staab, S., Motik, B.: KAON SERVER - a semantic web management system. In: Proc. WWW, Budapest, Hungary (2003)

W3C, Rdf semantics (2004), http://www.w3.org/TR/rdf-mt/