Designing a Federated Multimedia Information System on the Semantic Web

Richard Vdovjak, Peter Barna, and Geert-Jan Houben

Eindhoven University of Technology,
POBox 513, 5600 MB Eindhoven,
The Netherlands
{r.vdovjak, p.barna, g.j.houben}@tue.nl

Abstract. A federated Web-based multimedia information system on one hand gathers its data from various Web sources, on the other hand offers the end-user a rich semantics describing its content and a user-friendly environment for expressing queries over its data. There are three essential ingredients to successfully deploy such a system. First, one needs a design methodology identifying different design phases and their underlying models which serve as a framework for the designer. Second, there must be a set of tools that are able to execute the design, i.e. they serve as the back-end of the information system instantiating the models with data coming from various Web sources. Third, there also must be an entry point for the end-user, where he is able to explore what the system can do for him and where he can formulate his queries. This paper is a follow-up of our previous work describing the Hera design methodology and contributes to all three issues above. In particular, it refines the existing methodology by presenting an explicit RDFS-based integration model and explains how the mediator uses this model to obtain query results. The issue of a user-friendly front-end is addressed by introducing our interface for browsing and querying RDFS-based ontologies.

1 Introduction

Designing a Web-based federated information system (IS) that deals with multimedia items requires the consideration of both the IS back-end that integrates various Web sources, and the IS front-end that serves as an entry point for the user query. If this query is issued not only against text-like data but also against multimedia data, the semantic issues become crucial. Consider for instance the following user query: *"Give me all pictures of Niagara Falls taken from the Falls Avenue in Niagara Falls, Canada with a telelens"*. The first part of the query denotes the subject (waterfalls) being photographed. The second identifies the position of the photographer. Note the ambiguity of the "Niagara Falls" collocation, first denoting the waterfalls, and then being a part of the position description as a town name. Moreover, there is the third part of the query, imposing an additional lens constraint which basically says that we are only interested in those images that provide enough details and a narrow perspective achievable

J. Eder and M. Missikoff (Eds.): CAiSE 2003, LNCS 2681, pp. 357–373, 2003.

only by using a lens of that focal length. It is evident that queries of this kind are not likely to be satisfactorily answered by keyword based search engines. By translating this query into a set of keywords and trying a keyword based search engine we either obtained an empty set of results (e.g. Google Image Search) or a countless number of irrelevant pictures featuring big photo lenses (e.g. AltaVista Image Search retrieved over 1.4 million images and the top scored ones were indeed mostly showing long lenses and other photo equipment). Examples like this show a clear need for something more powerful than keywords. The use of ontologies, taxonomies of classes within a certain domain linked by properties indicating different relations among those classes, would enable to enhance queries and improve both the precision and the recall. Ontologies are becoming the essence of Web portals offering integrated views over various domains.

There are three important ingredients to successfully deploy such systems. First, one needs a design methodology identifying different design phases and their underlying models/ontologies which serve as a framework for the designer. Second, there must be a set of tools that are able to execute the design, instantiating the models with data coming from various Web sources. Third, there also must be an entry point for the end-user, where he is able to explore what the system can do for him and where he can formulate his queries. This paper is a follow-up of our previous work describing the Hera design methodology [1] and [2]. It contributes to all three issues above. In particular, it refines the existing methodology by presenting an explicit RDFS-based integration model, and explains how the mediator uses this model to obtain query results. The issue of a user-friendly front-end is addressed by introducing our interface for browsing and querying RDFS-based ontologies. A design of a Web-based photo portal is used as our case example for illustrating our ideas.

2 Hera Methodology

A primary focus of the Hera project is to support Web-based information system (WIS) design and implementation. A WIS generates a hypermedia presentation for data that in response to a user query is retrieved from the data storage. This entire process of retrieving data and presenting it in hypermedia format needs to be specified during the design of the WIS.

The typical structure of the WIS design in the Hera perspective consists of three layers:

- Semantic Layer: defines the content that is managed in the WIS in terms of a conceptual model; this layer includes the definition of the process of integration needed to gather the data from different sources; if the data is made available from outside the WIS, a search agent or information retrieval engine could be the interface to the WIS.
- Application Layer: defines the abstract hypermedia (navigation) view on the data in terms of an application model, which represents the structure shown to the user in the hypermedia presentation; this layer includes the definition

of the adaptation in the hypermedia generation, e.g. based on a user model and user/platform profile.
- Presentation Layer: defines those details that together with the definitions from the Application Layer are needed to generate a presentation for a concrete browsing platform, e.g. HTML, WML or SMIL.

In this paper we focus mainly on the integration and data retrieval process of the Hera methodology. This combined phase helps to make the data available from different sources, such that in response to a *user query* a *conceptual model instance* is generated that contains the data for which the application is going to generate a presentation: see Figure 1[1].

The integration is in principle performed before querying, as opposed to the retrieval and presentation generation that are performed for every query. It represents the data stored and therefore uses an integration model to map the data from the different sources into concepts of the conceptual model. From a mapping at schema (ontology) level a mapping at instance level is derived. This mapping is needed whenever for a given query the instances that compose the query result are to be retrieved. These instances need to be extracted by the mediator from the different *source ontology instances*. The role of the integration is to make these source ontology instances available(on-demand).

The data retrieval handles the reception of the *user query*, and in response produces a *conceptual model instance* for the query result. It starts with the translation of the query formulated by the user into a query that can act as a retrieval request on the data stored. This translation takes into account that while the user is allowed to formulate a query by mentioning items from the conceptual model or application model, the application model defines exactly which concepts need to be retrieved in connection with the items mentioned: this is known as query extension. Subsequently, using the query engine, the mediator retrieves the data from the sources and provides the query result. Finally, this query result needs to be transformed into the conceptual model instance that is passed on to the phase of presentation generation[2] .

2.1 Related Work

Most of the web engineering approaches (e.g. UWE [3] or XWMF [4]) do not explicitly consider integration and neither the user support for the query generation process, as opposed to what is the case in Hera. Given its RDF-based nature we address XWMF here in more detail. The eXtensible Web Modeling Framework (XWMF) [4] consists of an extensible set of RDF schemas and descriptions to model web applications. The core of the framework is the Web Object Composition Model (WOCM), a formal object-oriented language used

[1] The ellipses denote the transformations (in XSLT or Java) and the rectangles denote models or data. The shapes in grey denote application-independent items, the shapes with bold lines are query independent, while the others are query-dependent items.

[2] Details of this phase are beyond the scope of the paper, interested reader is referred to [2].

to define the structure and content of a web application. WOCM is a directed acyclic graph with complexons as nodes and simplexons as leaves. Complexons define the application's structure while simplexons define the application's content. Simplexons are refined using the subclassing mechanism in different variants corresponding to different implementation platforms. While Hera provides both a modeling framework and a methodology for developing web applications, XWMF appears to be only a modeling framework.

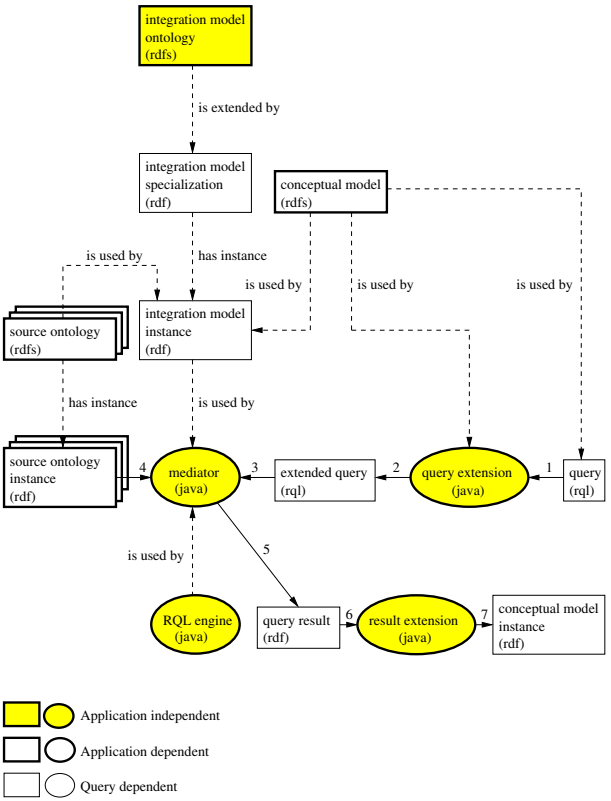

Fig. 1. Integration and data retrieval

3 RDF, RDFS, RQL and Their Role in Hera

The Resource Description Framework (RDF) [5] is a general-purpose data language issued as a W3C standard. An RDF model consists of resources, named properties, and property values. RDF Schema (RDFS) [6], an extension to RDF which is itself expressed in RDF terms, provides a support for creating vocabularies at the type (schema) level. RDFS defines a modeling language by assigning

a special semantics to several (system) resources and properties amongst which *rdfs:Class*, *rdfs:Property*, *rdfs:subClassOf*, *rdfs:subPropertyOf*.

In Hera, RDF(S) is the main format used for the different models in the design phases. One of the reasons for choosing RDF(S) is that it is flexible (supporting schema refinement and description enrichment) and extensible (allowing the definition of new resources/properties) Moreover, it is coming with the promise of web application interoperability. Hera model instances are represented in plain RDF validated against their associated models (schemas) represented in RDFS.

Since RDF(S) is used as our vehicle for capturing semantics of different domains within the IS being designed, we also need a query language that would enable us to retrieve the information of interest from these knowledge bases. The most advanced RDF(S) query language to date is RQL [7]. It covers both queries over RDF schemas and RDF instances. RQL queries that we will consider consist, similarly to SQL quires, of `SELECT-FROM-WHERE` clauses. The `SELECT` clause specifies resources (variables) which are of interest. The `FROM` part is the core of the query and specifies one or more path expressions (i.e. subgraphs of the entire schema graph) where the variables are being bound. Finally, the `WHERE` clause contains filtering conditions that are being applied on the variables bound in the `FROM` statement.

```
SELECT PHOTO
FROM   {PHOTO:GeoPhoto}depictsTheme{THEME:Universe},
       {PHOTO}takenFromPlace{PLACE:AddressablePlace}.country{COUNTRY},
       {PLACE}town{TOWN},{PLACE}street{STREET},
       {PHOTO}takenWithSettings.usedLens{LENS:TeleLens}
WHERE THEME like "*Niagara Falls" and
      COUNTRY like "Can*" and
      TOWN like "Niagara*" and
      STREET like "Falls Av*"
```

Fig. 2. User query in RQL

Despite its relative intuitiveness, RQL is still too low-level for an average end-user who wants to formulate queries over RDFS-based ontologies. It would be unreasonable to expect the end-user to type in a query like the one depicted in Figure 2, which is the RQL syntax of the "Niagara picture request" presented in the introduction. We address this issue later in section 5.2 by introducing our visual interface that enables the user to generate such RQL queries in a point-and-click manner.

RQL in combination with its java-based interpreter called Sesame [8] proved to be useful when building our retrieval engine, which in fact acts (with some limitations) as a distributed RQL query engine.

4 Integration and Data Retrieval

The main task of this design phase is to connect the application's conceptual model with several autonomous sources by creating channels through which the data will populate on request the concepts from the conceptual model. This involves identifying the right concepts occurring in the source ontologies and relating them to their counterparts in the conceptual model. Note that as opposed to classical database schema integration we do not aim at integrating all source concepts, but rather select only those that are relevant with respect to the defined conceptual model.

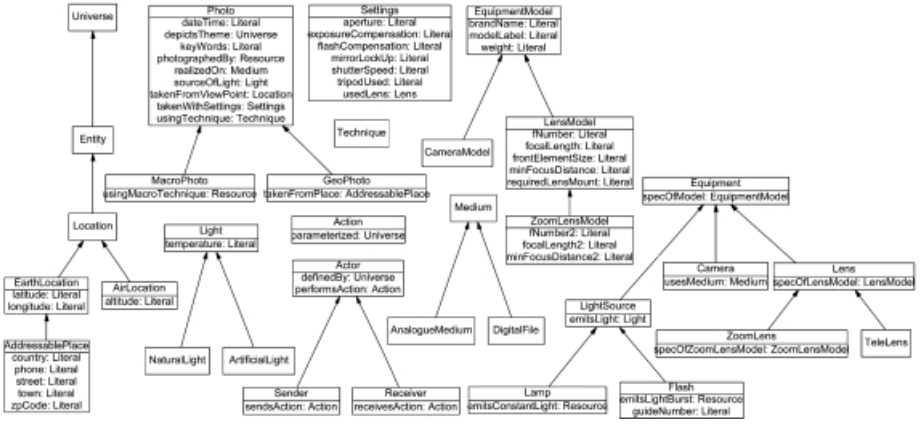

Fig. 3. Conceptual model

4.1 Conceptual Model

The conceptual model (CM) provides a uniform semantic view over multiple data sources. The CM serves as an interface between data retrieval and presentation generation. The CM is composed of concepts and concept properties that together define the domain ontology. There are two types of concept properties: concept attributes which associate media items to the concepts and concept relationships that define associations between concepts.

The running example used throughout the paper describes the design of a federated multimedia information system (MIS) implementing a photo library portal that allows the user to create on-the-fly photo exhibitions (browseable presentations of images of interest), study the photo technique behind these images and help him to rent or buy the necessary photo equipment in order to be able to achieve similar results.

The data is assembled on demand, based on the visitor's query, from the photos coming from different (online) photostock agencies, annotated with relevant

descriptions. The photo equipment data is gathered from (online) catalogues and matched with (online) photo rental offers. All this data is accessible from a single entry point, semantically represented by the CM. Figure 3 presents an excerpt of this CM which is composed of several sub-ontologies covering the semantics of different sub-domains:

- The `photo` sub-ontology consists of terms coming from the photo domain. It describes things like different kinds of `Light`, various photo `Techniques`, different camera `Settings` etc. The cornerstone of this ontology is the class `Photo` which is linked by its properties with the aforementioned classes. There is also a property called `depictsTheme` that connects the `Photo` class from the photo ontology with the `Entity` class from the general ontology.
- The `equipment` sub-ontology focuses on the photo hardware, describing and classifying different kinds of lenses, cameras and other related accessories.
- The general sub-ontology consists of all terms one can possible take a picture of, the most general term being the `Entity` class.
- The ternary sub-ontology serves as a means to describe a story captured on the photograph, where there are two or more actors that perform (either send or receive) an action. For instance a photo depicting a man biting a dog is certainly a different story than that depicting a dog biting a man.

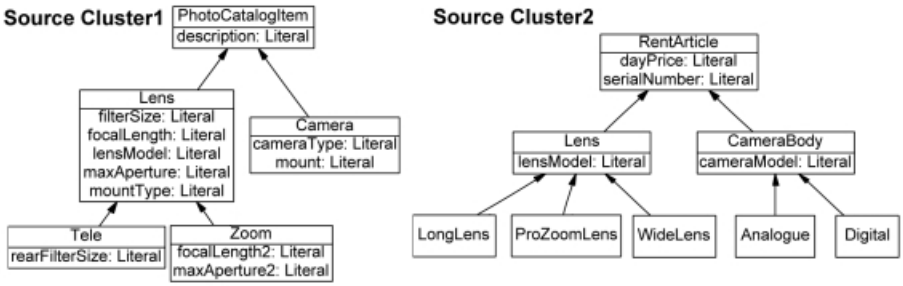

Fig. 4. Integrated sources: Photo Equipment Catalogue, Photo Equipment Rental

4.2 Sources

In our previous research [1] [9] we discussed how to overcome the syntactic heterogeneity of different source formats by introducing a layered approach starting with a layer of wrappers. In this paper we focus on the issues regarding the semantic heterogeneity. We consider for integration only those sources that are capable of exporting their schema in an RDFS based ontology and their data upon request (an RQL query) in RDF. In other words, we assume that each source offers its data on the Semantic Web platform providing RQL query services.

It is often the case on the Web that the information is duplicated and offered (possibly with a different flavor) from several sources. We group such sources into semantically close clusters and provide a means to order them dynamically within a cluster, based on several notions of quality introduced by the designer. Sources within a cluster do not necessarily have the same structure but should provide approximately the same semantic content. Sometimes for the sake of simplicity we abuse the word source when in fact we mean a whole cluster represented by that source.

As already mentioned, the data in our photo portal is coming from three sources/clusters: an online photo catalogue, rental offers, and online photostock agenicies. The abbreviated ontologies describing schemas of the first two clusters are depicted in Figure 4. The third kind of sources describing photostock agencies is omitted due to space limitations and we will consider it hereafter identical with the relevant parts of the CM.

The photo catalogue source (Figure 4 cluster 1) captures specifications of two sorts of catalogue items: lenses and cameras together with their respective subclasses. A seemingly similar class hierarchy featuring lenses and camera bodies exists also in the second source representing rental offers (Figure 4 cluster 2). There is, however, a considerable semantic gap between the two sources. The first source considers lenses or cameras as models, i.e. a virtual set of features and specifications. The second source refers to lenses and cameras as physical entities, i.e. concrete lenses and cameras (including their serial number) that are available for renting.

Suppose that the user, after reading details about a nice picture, decides to take similar pictures and he wants to rent the necessary photo equipment. When renting a lens from one source and a camera body from another source (remember there are more sources in one cluster) one must assure that the lens and the camera are compatible. In other words, that they share the same mount specification. As this information is not available in the rental offers the system has to look it up in the photo catalogue. In the following we present the integration model which helps to resolve such issues.

4.3 Integration Model

The integration model (IM) addresses the problem of relating concepts form the source ontologies to those from the CM. This problem can also be seen as the problem of merging or aligning ontologies. The approaches to automate the solution to this problem are usually based on lexical matches, relying mostly on dictionaries to determine synonyms and hyponyms; this is however often not enough to yield good results. Even when the structure of ontologies is taken into account the results are often not satisfactory especially in the case of uncoordinated development of ontologies across the Web [10]. For this reason and for the fact that every mistake in the integration phase will propagate and get magnified in all the subsequent phases, we currently rely on the designer or a domain expert to articulate CM concepts in the semantic language of sources.

What we offer the designer, is an integration ontology by the instantiation of which he specifies the links between the CM and the sources.

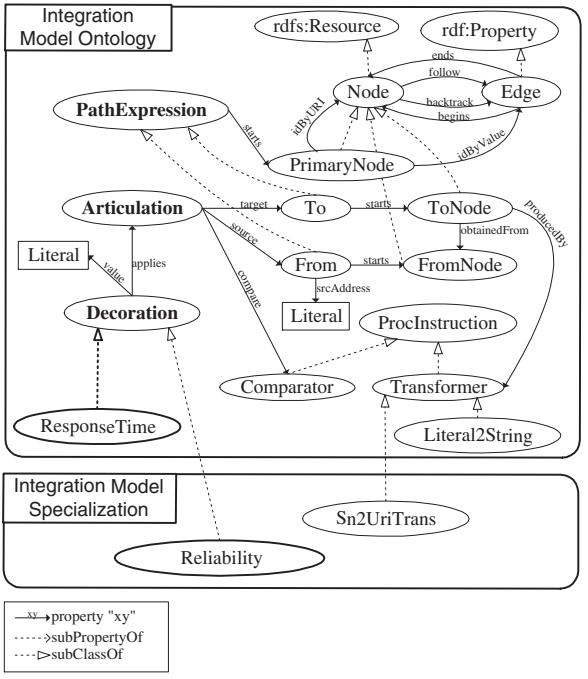

Fig. 5. Integration model ontology and its specialization

Integration Model Ontology. The integration model ontology (IMO) depicted in Figure 5 is a meta-ontology[3] describing integration primitives that are used both for ranking the sources within a cluster and for specifying links between them and the CM. The IMO is expressed in RDFS allowing the designer to tailor it for a particular application. The main concepts in the IMO are `Decoration` and `Articulation`.

Decorations. Decorations serve as a means to label "appropriateness" of different sources (and their concepts) grouped within one semantically close cluster. By having a literal property `value` they offer a simple way of ranking otherwise equivalent sources from several different points of view. There are some general decoration classes that are predefined in the framework (e.g. `ResponseTime`).

[3] IMO is a meta-ontology in the sense that its instances are dealing with concepts from other ontologies (source ontologies and the CM).

However, which ordering criteria are of interest depends mostly on the application. That is why the concept of `Decoration` is meant to be extended (specialized) by the designer. In this way we allow the designer to capture in the IM his (mostly background) knowledge regarding the sources.

For instance in our photo portal we example sources in the `Cluster 2` are graded based on the reliability of the equipment they offer for rent. Hence the `Reliability` decoration is introduced as shown in Figure 5 bottom.

The idea behind decorations is to capture the "reputation" of sources in different areas. By making this explicit, the mediator which is responsible for evaluation of queries can consult the relevant sources in the optimal way w.r.t. the chosen ordering criterion.

Articulations. Articulations describe actual links between the CM and the source ontologies and clarify also the notion of the concept's uniqueness which is necessary to perform joins from several sources.

Before we explain the concept of `Articulation` (see Figure 5) we need to introduce the notion of a path expression. A path expression is a chain of concepts (represented by the class `Node`) connected by their properties (represented by the class `Edge`). If the property has the given node as its domain (in other words we follow the arrow in the RDF graph) we connect them with the `follow` meta-property. If the property has the given node as its range (going against the arrow in the graph), we connect the two by the `backtrack` meta-property. This allows us to define inverse relationships even in the case when they are not present in the source ontologies.

Each path expression starts with a link to `PrimaryNode` which is a special node that can be uniquely identified either by a URI (`idByURI`) or by value (`idByValue`). The first points to a resource whose URI serves as an ID, the second points to a property (the `Edge` type) the value[4] of which serves as an ID.

In our example we chose to identify source concepts with identification-by-value due to the simpler way of deciding whether two resources describe the same thing. In the case of URI identification the decision whether two URIs refer to the same real-life object would require some sort of a web institution that normalizes several URIs to a canonical URI. This is in our opinion rather restrictive and a value-based identification is likely to work better when looking at the data coming from the Web source. The only exception where we chose to use `idByURI` is the local (mediator's) CM instance of the Equipment concept; this URI is generated from the `pr:serialNumber`.

An articulation contains two path expressions: the target path expression `To` pointing into the CM and the path expression called `From` pointing to a source (note the `srcAddress` property, value of which is the source URL). The target path expression contains nodes of type `ToNode`[5] that extends the `Node` with two

[4] By a value of a property we mean the object in RDF terminology.

[5] The ToEdge property is defined in a similar way but was omitted in order to simplify the figure.

properties: `obtainedFrom` and `producedBy`. The first links this node to its counterpart in the `From` path expression, the second points to a converting processing instruction called `Transformer`, which is called by the mediator to transform the source into the target. Processing instructions are resources containing a piece of Java code, an XSLT transformation, an RQL query or a combination of those. They are used by the mediator for changing and comparing values. Some general processing instructions are provided by the framework (e.g. the `Literal2String` transformer); those that are application-dependent are introduced in the specialization of IMO by the designer (e.g. the `Sn2UriTrans` transformer).

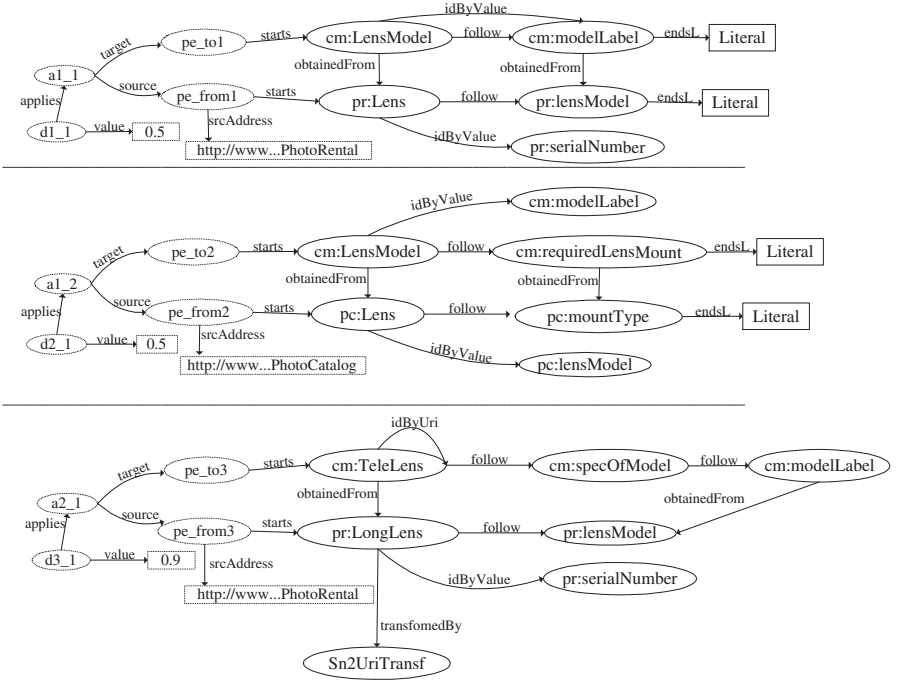

Fig. 6. Articulations, integration model instance

4.4 Integration Model Instance

The integration model instance is produced by the designer by instantiating the IMO[6]. Even though it is an ontology instance, it deals with the sources and the CM at the schema level, i.e. it makes statements about their concepts, not about instances. The choice of using RDF(S) as our underlying format proved to be

[6] Currently we are involved in building tools that allow the designer to specify the links and generate the articulations in the RDF(S) format automatically.

very useful since it is easy in its distributed fashion to make statements about other resources. Figure 6 shows three articulation examples.

The first is a simple articulation linking the `cm:LensModel` and its property `cm:modelLabel` with their counterparts from the photo rental source. The prime nodes are defined by the value of the properties: `cm:modelLabel` and `pr:serialNumber`.

The second articulation defines the lens property `cm:requiredLensMount` by linking it to the property `pc:mountType` from the catalogue source. Note the match between the identifying properties both referring to the model label.

The third articulation maps the concept `cm:TeleLens` into its counterpart `pr:LongLens` from the photo rental source. Note that these articulations were simplified in the sense that the links to the processing instructions (i.e. transformers that transform source values into the target values) are mostly omitted.

4.5 Data Retrieval

While the integration phase (instantiating the IM) is performed only once, prior to the user asking the query, the data retrieval phase is performed for every query. In this phase, the query is split into several sub-queries which are then routed to the appropriate sources. Subsequently, the results are gathered and transformed into a CM instance. Figure 1 shows the dataflow of this phase with three processing blocks involved: the query extension, the mediator, and the result extension.

Query Extension. As already mentioned the query language used in our system is RQL [7]. In this sub-phase the RQL user query, an example of which is depicted in Figure 2, is extended to contain all relevant data which is used by the presentation generation engine[2]. Note that in order to create a hypermedia presentation containing also links to relevant information that was not explicitly mentioned in the user query we must extend this query. The extension algorithm traverses the CM from a given concept(s) (`PHOTO: GeoPhoto`) and adds all concepts and/or literal types that can be reached by following property edges in the CM graph.

Mediator. The mediator is responsible for finding the answer to the query by consulting the available sources based on the integration model instance. As shown in Figure 1 the mediator takes the extended query as its input. Then it proceeds as follows: for every variable occurring in the `select` clause of this query it locates an articulation(s) which contains this variable. From this articulation the mediator determines the name of the concept occurring in the source and also the way how to obtain that concept, i.e. the necessary transformer(s) for the concept values, the address of the source, and the path expression to the concept of interest within the source schema. This path expression can be seen as a query executed on a particular source. Hence, consulting articulations in the IM instance in fact means query unfolding as it is known in the GAV approach.

The acknowledged disadvantage of the GAV approach is that in principle it requires changing the definition of the global schema (in our case the CM) each time a new source is added. This is however clearly not the case in our framework, since the only thing which changes when a new source is added or removed is the IM instance (new articulations are added or removed). From this point of view, we keep the CM independent from the sources, similarly to the LAV approach. Details concerning these approaches are beyond the scope of the paper, for details see [11]. If there are more articulations found for a given variable, that means there are several competing sources offering values for this variable. In this case the decorations attached to each articulation are used to decide the order in which the sources will be consulted.

After the sources are consulted, i.e. appropriate RQL queries are routed to them, the mediator waits for the response. Subsequently, it collects the results and assembles them into an answer which consists of a collection of tuples (in RDF terminology a bag of lists).

Result Extension. The answer provided by the mediator is a valid response to the RQL query that was asked, however it is not yet a CM instance. The result extension module transforms the "flat" collection of tuples by adding the appropriate properties into a valid RDF graph which adheres to the CM. This (query-dependent) CM instance serves as a basis for the presentation generation phase.

5 Hera Front-End

In the above sections we discussed how to design a data retrieval back-end for a multimedia information system. Here we focus on the front-end, the part of the system that interacts with the end-user. To bring an existing MIS to the end-user, two prerequisites are essential. Firstly, the users have to become familiar with the structure of the CM, i.e. to explore the ontology that captures the semantics of the MIS. Secondly, the system should help the users in generating the queries over this ontology, and visualize the results.

5.1 Visualizing the CM

As already mentioned, the CM is expressed in RDF(S). The problem of visualizing RDF(S) lies in the fact that it is difficult to show the whole expressive power of RDF(S) and at the same time to keep the user interface (UI) still comprehensible, easy to use, browse and navigate.

The two main approaches currently used: the tree-based approachand the graph-based approach, do not in our opinion address the above issues completely. The tree metaphor, though very familiar as UI, does not help the user in grasping other concept relationships than that used to construct the tree structure (most of the time being the rdfs:subClassOf relationship). The graph metaphor, on the other hand, displays all concept relationships but as a result introduces the

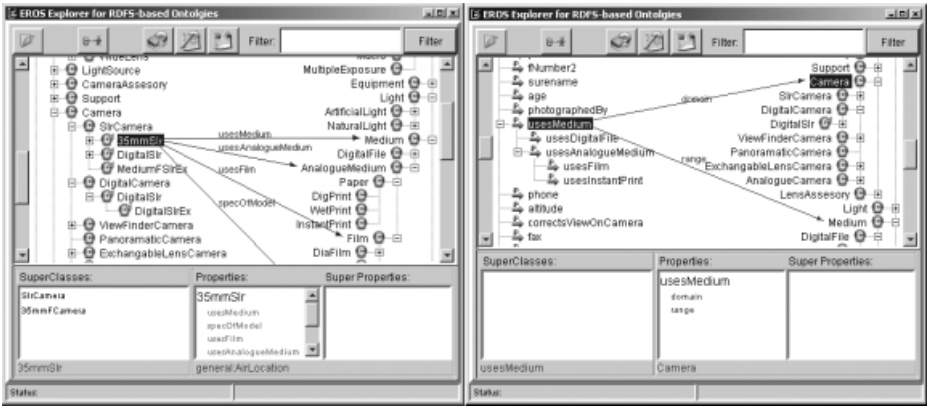

Fig. 7. The EROS interface: Class-centric view, Property-centric view

full complexity of a directed labeled graph in which is very difficult to spot the hierarchical structure of the ontology "hidden" behind the special edges and not reflected by the position of the class nodes.

Combining the advantages of the two mentioned UI metaphors, one would desire the simplicity of a browsable tree and at least a part of the expressiveness of the graph based approach. This is exactly what we tried to achieve by the EROS[7] interface.

5.2 The EROS Interface

The main idea behind this interface is to consider properties as partial mappings that map (some) elements (classes) from the class hierarchy into other (possibly identical) elements within the same hierarchy. Note that the set of all elements from the hierarchy serves two purposes: firstly as a (potential) domain of all properties and later as their (potential) range. This double purpose inspired us to have two (almost) identical hierarchy trees in our interface, the left tree being the domain ("from") tree, and the right tree being the range ("to") tree. Properties themselves are depicted as arrows connecting the classes from the left / domain tree with the classes from the right / range tree. Note that this approach makes it possible to display for a certain property at the same time both the context (the neighboring class hierarchy) of the domain class and the context of the range class. Cases of multiple inheritance are handled by displaying a list of super classes for the currently selected class. This approach, illustrated in Figure 7(left) can be considered as "class-centric" since the key transitive property on which both trees are built is the *subClassOf* relation.

As already mentioned in section 3 properties in RDF(S) are first-order citizens. So, if the user prefers to view the ontology with the "property-centric

[7] Explorer for Rdfs-based OntologieS

optics" and desires to explore the tree hierarchy based on the relation *subPropertyOf* the EROS interface can easily accommodate this demand by imposing that the left tree hierarchy is built based on the *subPropertyOf* relation and the right tree (still hierarchically based on *subClassOf*) represents the domains and ranges of the properties from the left tree as depicted in Figure 7(right).

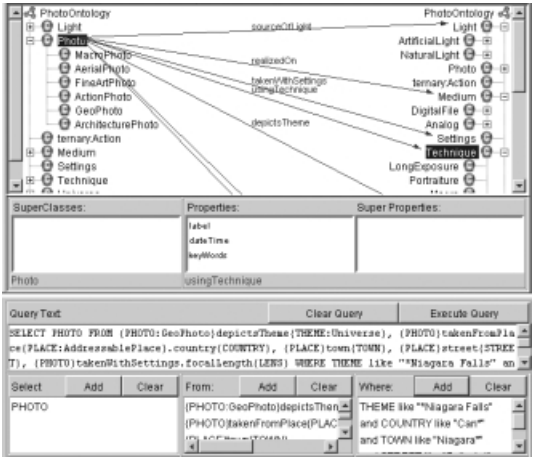

Fig. 8. The EROS interface: Query Building Mode

5.3 Query Generation Support

When in query mode, the EROS interface, guides the user in building RQL queries, i.e. specifying the `SELECT-FROM-WHERE` clauses. As the most tricky clause to build is the `FROM` part, the EROS interface starts by assisting the user in the formulation of a single path expression of the form: $\{VAR_i : DC_i\}p_i\{VAR_j : RC_j \mid L\}, \ldots$ This expression can be combined in a so-called chained expression which takes the following form: $\{VAR_i : DC_i\}p_i\{VAR_j : RC_j \mid L\}.p_j\{VAR_m : RC_m \mid L\} \ldots$ Where $VAR_x : Y$ denotes that a variable of type Y, p_x represents a property, DC_i and RC_j denote classes serving as a domain and range respectively; L represents a literal.

EROS allows the user to generate such expressions by selecting a node in the graph (a variable of a certain type), then selecting a property (p_I) of the chosen node that navigates the user to the destination node (again a variable). If the destination node is not of a literal type, the user can choose another property of that node traversing the graph further (building a chained path expression) or he can create a new path expression concatenating it with the previous ones.

Filling in the `SELECT` clause consists of choosing variables that are of interest from the list of variables introduced in the `FROM` clause. Similarly to the `SELECT` clause, the `WHERE` clause starts by selecting variables bound in the `FROM` clause

Fig. 9. The Resulting Hypermedia Presentation

and then building a Boolean expression by utilizing an offered list of appropriate operators. The queries that can be built in this way represent a large class of practical queries for ontology exploration and data retrieval as illustrated in Figure 2. However, we acknowledge that we do not cover the complete expressive power of RQL (e.g. nested queries nor implicit schema queries are not supported). Figure 8 depicts the query building part of the EROS interface. Note that the built query is that presented in Figure 2 and after its execution, the Hera system offers the query results in the form of a multimedia presentation depicted in Figure 9.

6 Conclusions and Future Work

As the nature of applications changes under the influence of the Semantic Web (SW) initiative, the need to capture the semantics of the application data increases. In the typical application representing SW ideas, the semantic annotation of Web resources is crucial. As the illustrative example we showed how the Hera methodology addresses this issue and how RDF(S) can help to improve the querying multimedia information systems. We presented an RDF(S)-based integration model and illustrated its use in a multimedia federated IS.

Essential in the use of RDF(S) are the ontologies. They help to organize the resources in terms of their semantics, and thus offer a nice specification of the semantics of the entire application. However, for many applications this specification in terms of ontologies needs to be used by humans. The end-users use such an ontology to find or search for terms and to mentally reason about

these terms. Before they are constructing actual queries on the RDF metadata they need to familiarize with the ontology. For this purpose an effective visual representation of ontologies is vital. We addressed this issue providing a visual interface in which the user is able to view the ontology both from the viewpoint of classes and that of properties. This interface also assists in the generation of RQL queries.

In the future we plan to help also the designer by turning this interface into an authoring tool which would allow for specification of the models used in the Hera methodology. From the integration perspective we intend to address the issue of query relaxation which would allow to retrieve approximate results in the case when the exact query does not succeed to find an answer.

References

1. Richard Vdovjak and Geert Jan Houben. Providing the semantic layer for wis design. In *Advanced Information Systems Engineering, 14th International Conference, CAiSE 2002*, volume 2348 of *Lecture Notes in Computer Science*, pages 584–599. Springer, 2002.
2. Flavius Frasincar, Geert Jan Houben, and Richard Vdovjak. Specification framework for engineering adaptive web applications. In *The Eleventh International World Wide Web Conference, Web Engineering Track*, 2002. `http://www2002.org/CDROM/alternate/682/`.
3. Nora Koch, Andreas Kraus, and Rolf Hennicker. The authoring process of the uml-based web engineering approach. In *First International Workshop on Web-Oriented Software Technology*, 2001.
4. Reinhold Klapsing and Gustaf Neumann. Applying the resource description framework to web engineering. In *Electronic Commerce and Web Technologies, First International Conference, EC-Web 2000*, volume 1875 of *Lecture Notes in Computer Science*, pages 229–238. Springer, 2000.
5. Ora Lassila and Ralph R. Swick. Resource description framework (rdf) model and syntax specification. W3C Recommendation 22 February 1999.
6. Dan Brickley and R.V. Guha. Rdf vocabulary description language 1.0: Rdf schema. W3C Working Draft 30 April 2002.
7. Gregory Karvounarakis, Vassilis Christophides, Dimitris Dimitris Plexousakis, and Sofia Alexaki. Querying rdf descriptions for community web portals. In *17iemes Journees Bases de Donnees Avancees*, pages 133–144, 2001.
8. J. Broekstra and A. Kampman. Sesame: A generic architecture for storing and querying rdf and rdf schema. Aidministrator Nederland b.v. October 2001.
9. Richard Vdovjak and Geert Jan Houben. Rdf-based architecture for semantic integration of heterogeneous information sources. In *International Workshop on Information Integration on the Web*, 2001.
10. Natalya F. Noy and Mark A. Musen. Anchor-prompt: Using non-local context for semantic matching. In *Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence*, 2001.
11. Jeffrey D. Ulman. Information integration using logical views. In *Proceedings of the 6th Int. Conference on Database Theory, ICDT'97*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 1997.