Providing the Semantic Layer for WIS Design

Richard Vdovjak and Geert-Jan Houben

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

Abstract. Designing Web-based information systems requires the use of a thorough design methodology. Particularly, when the content of the system is gathered from different information sources available via the Web, the specification of how data is to be retrieved requires appropriate design tools. Concretely, a solution to the problem of how to facilitate the design of integrating heterogeneous information sources is needed, in order to be able to provide a uniform access to data gathered from different sources. In this paper we propose the use of the Hera methodology extended with the Semantic Layer, which concentrates on the integration aspect. The presented integration framework provides a coherent and meaningful (with respect to a given conceptual model) view of the integrated heterogeneous information sources.

1 Introduction and Related Work

The WWW has become one of the most popular information channels of today. This results into an ever-growing demand for new Web applications. The Web is, however, becoming a victim of its own success. Ad hoc hacking without a prior design, using no rigorous methodology is currently the common Web development practice. This approach fails to meet the demand for high quality data-driven Web applications such as Web-based Information Systems (WIS).

WIS applications use Web technologies to fulfill the needs of professional information systems. The specific role of modern WIS asks for a highly structured and controlled approach to Web engineering [1]. Many WIS have a data-driven nature, that requires a process of automatically generating hypermedia or multimedia (Web) presentations for the data to be output.

To facilitate the engineering of these data-driven Web applications there is an obvious need for a design framework. This framework should allow designers to specify and reason about WIS in an appropriate level of abstraction depending on the different stages of the engineering project (requirements analysis, design, and implementation), but also on the different dimensions of the problem area (e.g. ata integration and modeling, hyperspace navigation, user/platform adaptation, layout design etc.).

The specification of artifacts (models) of the different stages of the design process can benefit a lot from technologies like RDF(S)[2,3], introduced with the Semantic Web initiative [4], where the main idea is to allow for the data

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 584–599, 2002.

that resides on the WWW to be targeted not only for humans but to become also machine-understandable. The goal is to achieve semantic interoperability, which would facilitate creation of new services such as smart search engines, automated information integration and exchange, presentation generation, etc. We argue that semantic interoperability will have a positive impact on the design and use of WIS, especially those constructed from several heterogeneous sources.

Existing methodologies for designing Web applications, such as RMM[5], OOHDM[6], WebML[7], do not explicitly cover the integration phase. We believe that the integration issues (Conceptual Model design, Schema integration, and Data integration) should be taken into consideration during the design of those data-driven Web applications, for which the content is coming from heterogeneous data sources.

In previous work [8,9] we suggested a methodology inspired by RMM for the design of automatically generated data-driven Web presentations for ad-hoc user queries. This paper is a follow-up extending the methodology with the Semantic Layer, so that it can be used as a basis for engineering WIS-like applications. We focus on designing those WIS that are built from collections of heterogeneous information sources. The paper addresses problems of information integration (both schema and data), such as syntactic and semantic reconciliation of different sources.

The rest of the paper is structured as follows. Section 2 presents the different steps of our design methodology and the underlying software suite that provides the means for executing the specifications resulting from the design process. In section 3 we focus on the Semantic Layer and we cover in detail the issues regarding the design of the models that create the foundation of this layer. Section 4 introduces a software architecture that implements the Semantic Layer. We conclude the paper by section 5 with a short summary and future work.

2 Hera Design Methodology

The Hera design methodology has its origins in RMM[5]. Similarly to RMM, it distinguishes several steps to be followed during the design process of a Web application[1]. Each design step produces as its outcome a specification with a certain level of abstraction based on the separation-of-concerns principle. The sequence of the main steps is depicted in figure 1.[2]

The first step, the Requirements Analysis clarifies and captures the needs and demands that the client has for the future application, into a requirements document, which serves as a "wish list" that has to be fulfilled by the designer(s) and as a "check list" for testing, once the application is developed.

[1] Although the methodology is general and can be applied to many (types of) Web applications, here we primarily concentrate on data-driven Web applications of the typical WIS nature.

[2] Note that the arrows in the picture denote only the general flow; it is possible to have feedback loops in the process.

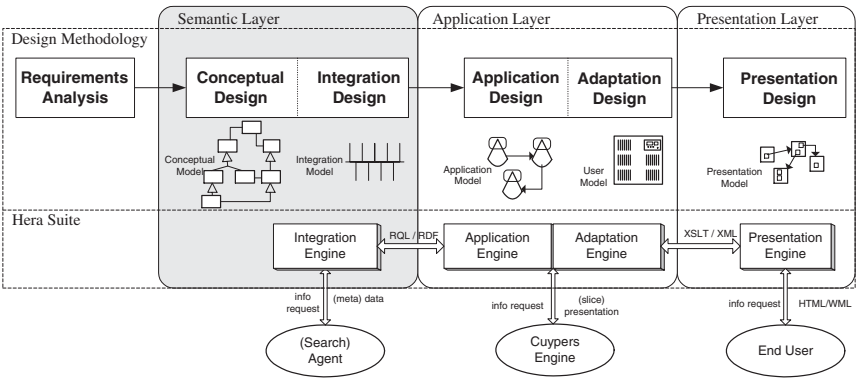

Fig. 1. Design Methodology, and underlying set of data-processing engines

In the Conceptual Design step the application domain is captured into a Conceptual Model (CM) consisting of a hierarchy of concepts, their properties, and relations. Within an information system one usually distinguishes a data repository, which contains the data that is available for querying: each answer to a query results in the generation of a (Web) presentation. The purpose of the CM is to describe which information is available inside this data repository.

Note that if the data is coming from different (possibly heterogeneous) sources, the data repository can be virtual (not materialized). In this case an Integration Model (IM) must be introduced, which selects sources and articulates (translates) the relevant concepts from them into the CM. It is evident that the design of CM influences the creation[3] of IM (IM mappings depend on the CM), but the dependency here is mutual. There is not much use of introducing concepts into the CM if they cannot be populated with data instances if we do not have appropriate data sources for them in the IM. So, it is often the case that the CM design and IM creation are intertwined and accomplished in an iterative manner.

In the Application Design step, the concepts from the CM are transformed into slices. In analogy to RMM, we use the notions of slice and slice relationship to model how the concepts from the CM will be presented. While the CM gives a *semantic description* of the information that is available, the Application Model (AM) [9] gives a *navigational description* of that information. Having in mind that Web presentations will be generated for queries against this information, the navigational description specifies (a blueprint for) the hypermedia nature of those presentations. The AM does not yet include all the rendering details covered in the next steps of the methodology, but it does allow for a first, logical sketch of the presentation to be generated. This implies the translation of

[3] We make a distinction between "creating" and "designing" in a sense that the first can be to a large extent automated while the second is more "human dependent".

concepts into "pages" and the establishing of a navigational structure between those pages.

The era of "one size fits all" Web applications seems to be vanishing and personalization and adaptation (one-to-one content delivery) is becoming an issue for an increasing number of applications. A User Adaptation Model (UAM) supporting the adaptation is built during Adaptation Design; in line with the AHAM reference model [10] this UAM includes the specification of a user model and adaptation rules. The design of UAM is intertwined and associated with the Application Design, just like in the case of IM creation and CM design.

The Presentation Design step introduces a rendering independent Presentation Model (PM)[9], which focuses on layout, hyperlinking, timing and synchronization issues. The Presentation Design elaborates on the Application Design and bridges it with the actual code generation for the desired platform, e.g. TML, WML, SMIL etc.

The Hera suite (the lower part of figure 1) is a collection of engines (software programs), which can interpret the specifications (or models) provided by the designer during the different phases of the design process.

The suite is split into several layers each of them processing a different model and thus reflecting a different design phase.

- The Semantic Layer, which is the primary focus of this paper, integrates the data that is available in a collection of heterogeneous external data sources into one Conceptual Model (CM) with well-understood semantics. The Mediator as a main component of this layer, provides mediation services for the Application Layers and offers also a connectivity for the "outside world" of software agents.
- The Application Layer provides the logical functionality of the application, in the sense that it exploits the structure of the CM to lay a foundation for the hypermedia nature of the output in the form of the AM. The Application Engine and the Adaptation engine are two main components of this layer: the first generates a relevant (depending on the given query) "subset" of the AM, which is then populated with data coming from the Semantic Layer; the second interprets the adaptation rules and updates the UAM accordingly.
- The Presentation Layer of the framework is responsible for the transformation of the Application Model (possible including the User Adaptation Model) into a chosen rendering platform (e.g. TML, WML or SMIL). In order to achieve this, the Presentation Engine uses a relevant part (again depending on the given query) of the implementation independent Presentation Model provided by the designer, which is populated with data coming from the previous layer. This is then transformed by means of XSLT [11] to a chosen rendering platform.

3 Designing the Semantic Layer

As already suggested in the previous section, the main purpose of the Semantic Layer is to provide a semantically unified interface for querying (selected) het-

erogeneous information sources. We do not aim at merging all possible sources together to provide a cumulated view of all attributes. We argue that such an approach offers weak semantics, where the understanding of the semantic structure of all integrated sources is effectively left up to the user who is issuing the query against such a view. Instead, we chose to provide the user with a semantic entry point in terms of the CM, which is interconnected with the underlying sources by means of the IM.

Because of the Web nature of our target applications in combination with the dynamically changing data in the underlying sources, we chose to base our integration framework on the lazy retrieval paradigm [12] assuring that the delivered data is always up-to-date. To provide support for interoperability we combine this retrieval with the technologies introduced with the Semantic Web initiative, namely RDF(S)[3,2]. There are several reasons why we decided to use RDF(S):

- Compared to a traditional database schema, it deals better with the semi-structured nature of Web data.
- RDF(S), a W3C standard for metadata, and its extensions such as DC[13], CC/PP[14], RSS[15] etc. bring us closer to interoperability at least as far as descriptive metadata is concerned.
- On top of RDF(S) high level ontology languages (e.g. AML+OIL[16]) are (becoming) available, which allow for expressing axioms and rules about the described classes giving the designer a tool with larger expressive power. Choosing RDF(S) as the foundation for describing the CM, enables a smooth transition in this direction.

When looking at integration as a (design) process, the following two principal phases can be distinguished: designing the Conceptual Model and designing/deriving the Integration Model. Further on, we detail these phases.

3.1 Conceptual Model Design

The Conceptual Model (CM) provides a uniform interface to access the data integrated within a given Web application. It corresponds to a mediated schema that is filled with data during query resolution. The application user is assumed to be familiar with the semantics of terms within his field of interest, and the function of the CM is to offer the user a uniform semantic view over the different sources, that usually use different terms and/or different semantics.

The CM consists of hierarchies of concepts relevant within the given domain, their properties, and relations. As already mentioned above it is expressed in RDF(S).

Although RDF(S) seems to be a promising modeling tool it does not provide all modeling primitives we demand. Namely, the notion of cardinality and inverse relationship is missing and there is also a lack of basic types. There are more ways how to approach this problem, for example, in [17] it is shown how to combine the data semantics, expressed in RDF(S), with the data constraints, modeled in XML Schema [18]. As there is no clear W3C Recommendation on

this subject yet, we chose for the purist approach, i.e. to model both data constraints and data semantics in RDF(S) itself, by extending it with the mentioned modeling primitives (the extensions are recognizable in the actual encoding with the abbreviated namespace prefix "*sys:*").

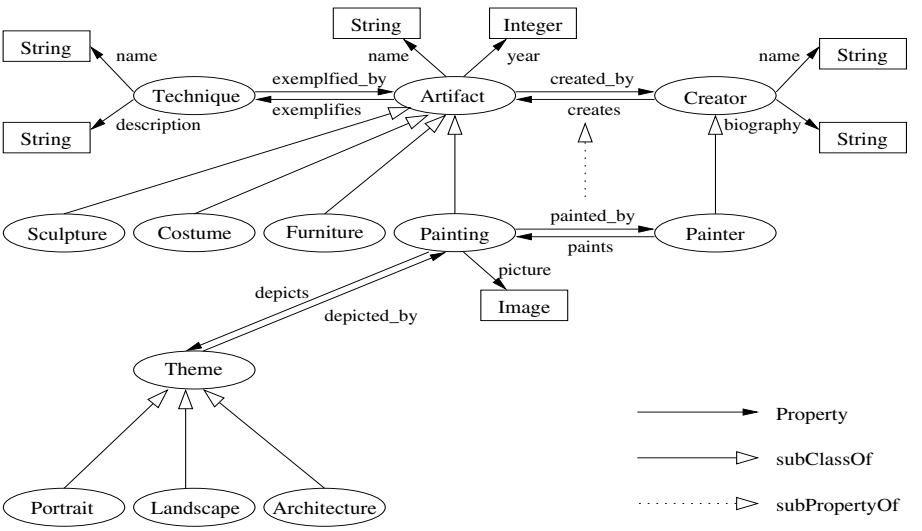

Fig. 2. Conceptual Model

The running example that we use throughout the paper describes the design of a virtual museum Web information system, which allows visitors to choose (query) their favorite artist(s) and/or pieces, and to browse exhibitions (Web presentations) assembled from exhibits coming from different (online) museums and art galleries, and possibly annotated with relevant descriptions from an online art encyclopedia. All this data is offered from a single entry point, semantically represented by a CM. In order to create a good CM it usually requires both the domain expertise and modeling know-how. In this example, we let ourselves be inspired by an existing CM. The Conceptual Model of our application roughly corresponds to the actual museum catalog of the Rijksmuseum in Amsterdam[4].

A part of this CM is depicted in a graphical notation in figure 2:

- Concepts are depicted as ovals; the more abstract ones such as *Artifact* and *Creator* are on the top part of the model. From those, more concrete concepts are inherited, e.g. *Painting* and *Painter*.
- Relationships are denoted with full arrows and are modeled as RDF properties. Similarly to concepts there are abstract relationships (properties) such

[4] `www.rijksmuseum.nl`

as *created_by*, *creates* and their more concrete subrelationships (subproperties) *painted_by* and *paints*.
– Attributes, e.g. *Creator.name: String*, are treated as properties having as its domain a concept, e.g. *Creator*, and as its range a basic type, e.g. *String* (indicated with the rectangle).

```
<Technique rdf:about="Technique_ID01_Pointillism">
    <name>
        <sys:String>
             <sys:data>Pointillism</sys:data>
        <sys:String>
    </name>
    <description>
        <sys:String>
             <sys:data>
                 Pointillism is a painting technique in which
                 the use of tiny primary-color dots is used
                 to generate secondary colors...
              </sys:data>
        </sys:String>
    </description>
</Technique>
<Painting rdf:about="Painting_ID01_ BanksoftheSeine ">
    <name>
        <sys:String>
             <sys:data>Banks of the Seine</sys:data>
        <sys:String>
    </name>
    <year>
        <sys:Integer>
             <data>1887</data>
        </sys:Integer>
    </year>
    <exemplifies rdf:resource="#Technique_ID01_Pointillism"/>
    <painted_by rdf:resource="#Painter_ID01_VincentvanGogh"/>
</Painting>
```

Fig. 3. Examples of RDF instances of the CM

Note that the graphical notation of the CM directly corresponds to an RDF(S) graph. RDF(S) as such has not only graphical but also a textual (XML) syntax. This XML-RDF(s) serialization is used in our software prototype to represent the CM.

During the actual run-time process of presentation generation the CM is on request populated with instances (RDF statements), which represent the query

result coming from the Application Layer. Note that the source data generally comes from different (heterogeneous) sources and that the actual instance retrieval is performed by the Integration Engine. Figure 3 presents in a XML-RDF(S) syntax a concrete example of generated instances adhering to our CM, as a response to the given query which propagated from the Application Layer.

3.2 Integration Model Design/Creation

The task of building the Integration Model is rather complex. It includes steps like finding relevant information sources, parsing, understanding and articulating the relevant parts of their schemas in terms of the CM. An articulation that connects the CM with a concrete source is implemented as a set of mappings, which relate (express) concepts from the source to (in terms of) concepts in the CM. These articulations are then used during the data integration step to provide the actual data response to a given query. Some of the above steps can be automated, some still need a human interaction. In the following, we look at them in more detail.

Source Discovery As it usually takes the human insight of the designer(s) to create a (good) CM, finding relevant sources to populate the CM with data is currently also mostly done by humans. When creating the CM, the designer can be helped by available ontology engineering tools, which allow for export in the RDF/S format. However, in the search for suitable sources the designer is currently left alone in the vast space of the World Wide Web. We envision that when the idea of the Semantic Web becomes reality (i.e. hen most of the Web sources would provide a machine understandable semantics of their data and services), search agents and information brokers will be capable of finding (recommending) suitable sources to populate the CM.

Schema Integration An essential prerequisite to achieve interoperability is to be able to identify and link together semantically similar data coming from different sources. To fully automate this task is very difficult, especially in the context of the semistructured Web sources. Even in the more structured database world, this has been recognized as a not always solvable problem [19]. We try to split this problem into two. The first one, easier to solve, deals with syntactical issues. The second, more complicated one, deals with the reconciliation of semantic discrepancies.

- Syntactic issues
  Unlike the case of database integration, where the schemas of integrated sources are assumed to be known explicitly, in the Web environment we often have to do the pre-integration schema discovery. In other words, it is needed to identify and make explicit what classes/concepts are provided by the integrated sources. In the ideal world of semantically annotated sources,

which export their schemas in RDF/S, this would be a question of parsing an RDF file, so it can be fully automated.

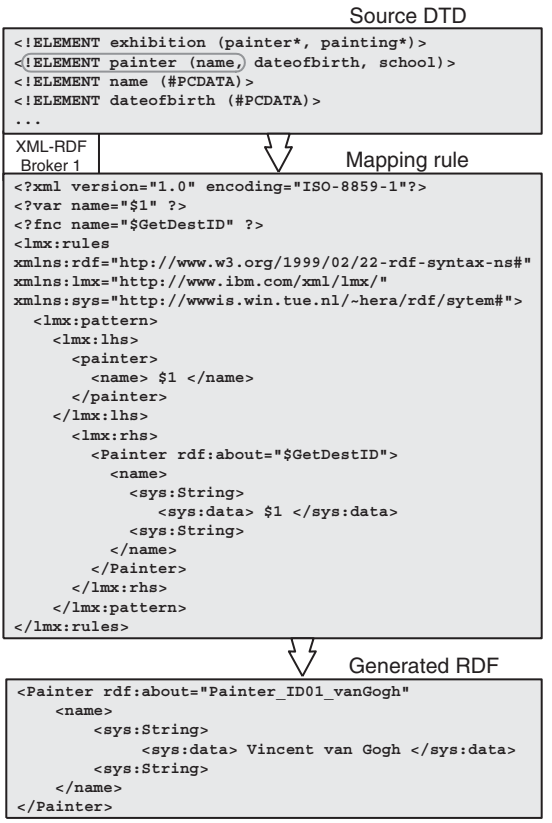

Fig. 4: The concept extraction from a DTD and its corresponding mapping to the CM

At the present however, most of the Web sources do not provide more than an XML description of their content. In the following we consider sources having at least this capability[5].

Even if the sources are encoded in XML, assuming no particular XML structure, it is still difficult to automatically recognize concepts and in order to do so, the insight of an application designer is often needed. The difficulty here varies based on the way in which the sources are encoded in XML.

[5] In case of HTML-only sources, a wrapping process is required to extract the source's content to an XML form.

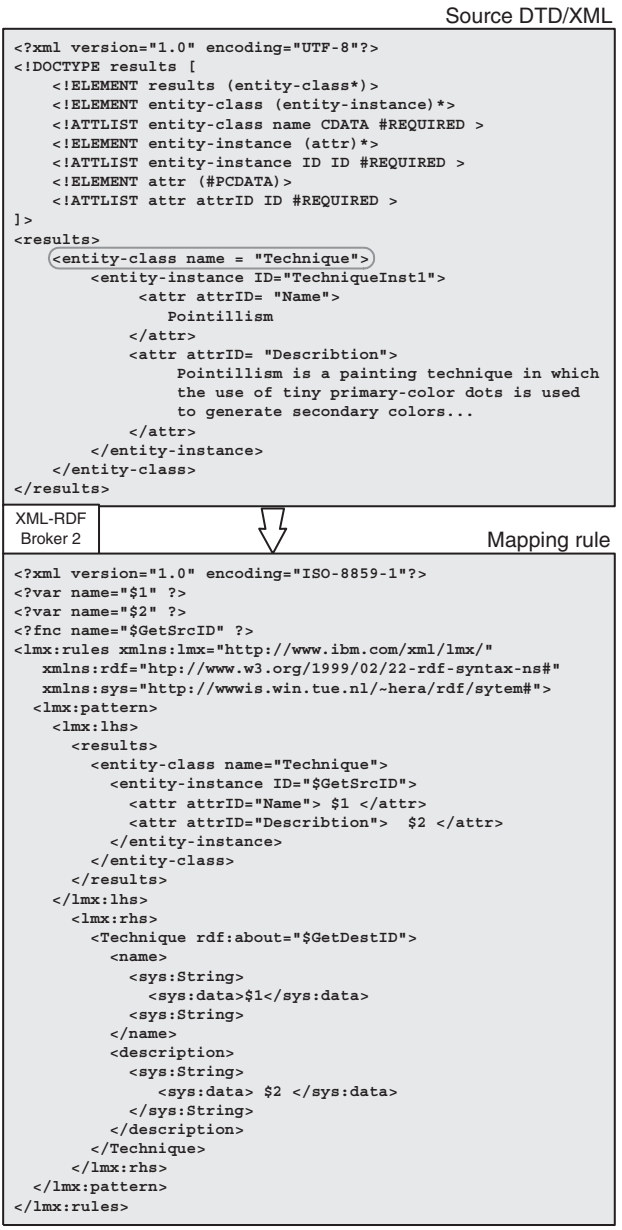

Fig. 5: The concept extraction from XML data and its corresponding mapping to the CM

For example, assume that one of the online galleries that we integrate in our virtual museum provides a description of the available data in the form

of a DTD, a part of which is shown in figure 4 (top). The designer can conclude directly from this DTD that the element *painter* together with its subelement *name* is to be extracted as a relevant concept and mapped to our CM. However, looking only at DTDs (XMLSchemas) is sometimes not enough. Assume for instance that an online art encyclopedia encodes its concepts as XML attribute values, providing only a general DTD as shown in figure 5 (top). In this case, the designer must also examine the actual XML data to conclude that the encyclopedia offers a concept called *Technique*, which is to be linked to our CM.

– Semantic issues
  After reconstructing the source schemas, the main problem is to choose the right concepts occurring in the source schemas and to relate them to the concepts from the CM (recall that unlike in classical database schema integration, we do not integrate all concepts from sources, but rather select the relevant ones with respect to the defined CM).
  The problem of relating concepts form the source schemas to the ones from the CM can be mapped to the problem of merging or aligning ontologies. The approaches to solve the 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.
  In [20] the structure of ontologies is also taken into account when searching for corresponding concepts. Neither of these approaches, however, delivers satisfactory results especially if ontologies are constructed differently, e.g. having a very different structure or differing in the depth of the class hierarchy. This is often the case in uncoordinated development of ontologies across the Web and that is why ontology aligning is currently mostly a manual process, where a domain expert identifies the concepts that are similar and records the mappings between them. In our framework, we also currently rely on the designer who provides such mappings[6].
  As mentioned above an articulation is implemented as a set of mappings, which relate concepts from a source to the concepts from the CM. In case of integrating an RDF source the mappings are mostly straightforward and usually they are also expressible in RDF. However, if we want to integrate an XML source we need something that transforms the source's XML to the RDF(S) of the CM. For this kind of transformation we use mapping rules that extract the relevant portions of information [7] from the (XML) sources and relate them to the CM.
  These mappings are expressed in the form of LMX[8][21] rules consisting of a left-hand side, the "*from* part", and the right-hand side, the "*to* part". The data that is to be transferred is specified by positioning variables denoted as *$x*. These are declared at the beginning as processing instructions

[6] We intend to re-address this issue in the future by providing the designer with a tool, which would suggest initial correlations.

[7] It is often the case that only some attributes from the source concept are linked to a concept from the CM.

[8] Language for Mapping XML documents.

(together with functions such as *GetDestId*, which is used for a coordinated assignment of IDs) to make the application aware of them. In figure 4 we present an example of such a mapping rule together with the RDF instances it yields. Another example is presented in figure 5; this rule maps the *Technique* element coming from the online art encyclopedia to the corresponding concept in our CM. The result (an RDF instance) of this mapping is shown in figure 3 (top). Given the mappings, the semantic reconciliation is performed by the XML2RDF brokers described in section 4.

Data Integration Once the relevant high level sources have been identified and mappings to the CM established, the user can ask a query. The integration system must reformulate it into a query that refers directly to the source schemas taking into account the source query capabilities. Then the results are collected and presented to the user. Note that sometimes it is also the case that different sources provide information about the same real world entities but refer to these entities differently; this is also called a designation conflict. The integration system has to determine such cases and consider them adequately.

The query processing, and the composition of results is a task performed by the mediator described in section 4.

4 Implementing the Semantic Layer

The notion of mediator as introduced in [22] laid the foundation for building mediating architectures that have the ambition to overcome the semantic heterogeneity and facilitate a seamless access to data coming from many heterogeneous sources. In order to provide the Semantic Layer, our prototype implements such a mediating architecture. To address the issues mentioned in the previous sections, the architecture is split from software point of view into separate independent (sub)layers (figure 6), each them responsible for a different task and together creating a well-knit integration framework. We describe each of the layers further.

– Source Layer
  The Source Layer contains external data sources such as relational or object databases, HTML pages, XML repositories, or possibly RDF (ontology) based sources. This layer provides the content to be integrated. Our target applications can be built from fairly general sources that can be distributed across the Web. As already mentioned, we assume that the sources have the capability of exporting their data in XML serialization. However, due to heterogeneity, we do not impose any particular structure the XML data sources should comply to. This allows us to leave the XML wrapping process for the source providers.
  Note that by choosing RDF(S) as the foundation for the CM and by exporting it via the mediator to the outside world, one instance of our framework

can qualify as a resource for another framework, hence providing composability [9].

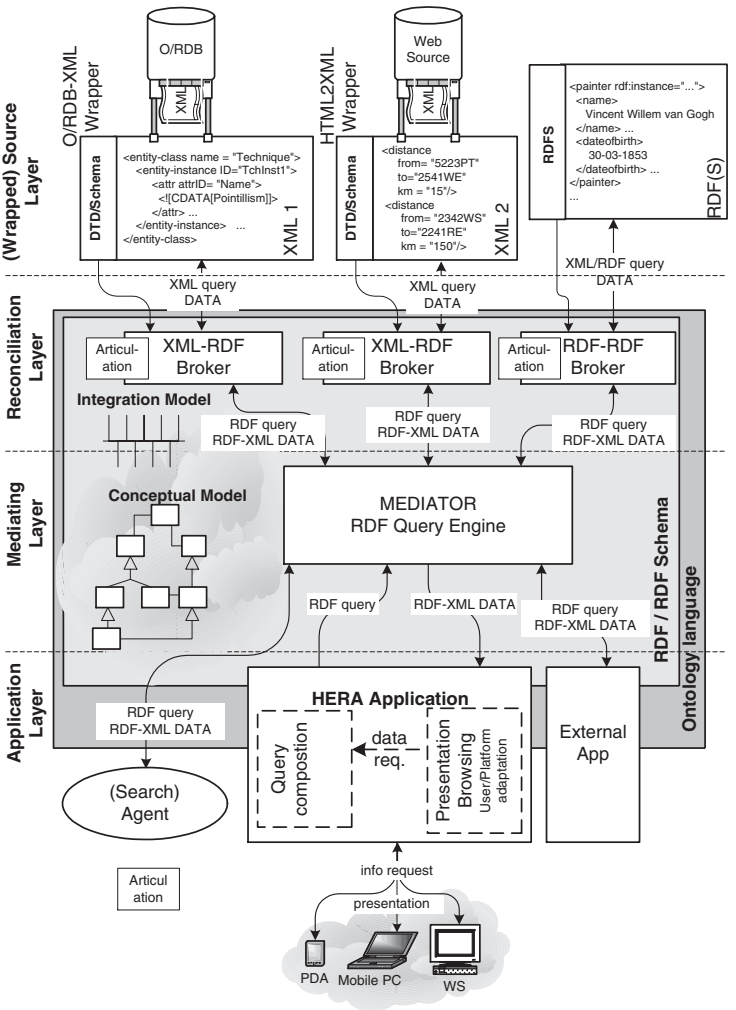

Fig. 6: Architecture facilitating semantic integration of heterogeneous information sources

– Reconciliation Layer

[9] We acknowledge that implementing our architecture on newly appearing middleware standards such as J2EE from Sun Microsystems could bring composability and interoperability even further.

This layer is responsible for the semantic reconciliation of the sources with respect to the CM and consists of XML2RDF brokers, which provide the bridge between the XML instances from the previous layer and the mediator. The designer tailors each XML2RDF broker to its source by specifying an articulation that maps XML sources to the underlying CM. This articulation is implemented as a set of mapping rules, which are used by the XML2RDF broker while resolving a query coming from the mediator. Providing the actual response to a mediator's query requires the broker to poll the source for data and to create RDF statements, that is riplets (*subject*, textitpredicate, textitobject).

– Mediating Layer
  This layer is responsible for the query services and data integration, its central component being the mediator. The mediator can be considered as a secondary source which does not have a data repository; instead, it maintains the CM. From the technical point of view the mediator contains a query decomposition module and an RDF query engine [23]. After the mediator receives a query from the Application Layer it proceeds as follows.
  First, it decomposes the query into subqueries [24] and distributes them among the brokers trying to push the processing of the query as much as possible on the sources, while taking into account the source query capabilities. The actual querying is triggered by a navigation request coming from the Application Layer.
  Then it collects the data from the brokers, applies a field matching algorithm citeAEMCPE:96 to resolve possible designation conflicts, constructs the response and sends it to the Application Layer.
– Application Layer
  The Application Layer utilizes the Mediating Layer by asking queries against the CM. For the Application Layer, the Mediating Layer is actually the front-end of the Semantic Layer introduced in section 2. From the mediating point of view it is not important whether the application generates (Web) presentations for the end-user (e.g. s the Hera suite does) or it only processes the pure information provided by this layer (e.g. s a search agent does): both cases are served in the same manner by answering queries from the CM.

5 Conclusions and Future Work

Creating Web-based information systems requires the use of a thorough design methodology. Specially, when the content of the system is gathered from different information sources available via the Web, the specification of how data is retrieved requires appropriate design tools. Concretely, a solution to the problem of integrating heterogeneous information sources is needed in order to be able to provide a uniform access to data gathered from the different sources. In this paper we have extended the Hera design methodology with the Semantic Layer concentrating on its integration aspect. The proposed integration framework combines semantic metadata with on-demand retrieval. It offers a composable semantic interface for (dynamic) access to heterogeneous information sources.

In terms of the software, on-going experiments in the context of the Hera project are verifying our ideas about integration specification with an implementation of the framework's prototype. In the future we would like to extend our framework with an algorithm that would deliver starting correlations between source concepts and the concepts from the CM, which can be then improved by the designer. Next, we also intend to extend the conceptual model with a high level ontology language allowing for expressing inference rules.

References

1. Deshpande, Y., Ginige, A., Hansen, S., Murugesan, S.: Web Engineering: A New Discipline for Web-Based System Development. Proc. of the First ICSE Workshop on Web Engineering, ACM, 1999 584
2. Lassila, O., Swick, R. R.: Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, 1999 `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222` 584, 588
3. Brickley, D., Guha, R. V.: Resource Description Framework (RDF) Schema Specification 1.0. W3C Candidate Recommendation, 2000 `http://www.w3.org/TR/2000/CR-rdf-schema-20000327` 584, 588
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American, 2001 May, `http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html` 584
5. Balasubramanian, P., Isakowitz, T., Stohr, E. A.: RMM: A Methodology for Structured Hypermedia Design. Communications of the ACM, Vol. 38, No. 8, 1995 585
6. Barbosa, S. D. J., Rossi, G., Schwabe, D.: Systematic Hypermedia Application Design with OOHDM. Proc. of the Seventh ACM Conference on Hypertext, 1996 585
7. Bongio, A., Ceri, S., Fraternali, P.: Web Modeling Language (WebML): a modeling language for designing Web sites. Proc. of the Nineth International World Wide Web Conference (WWW9), Elsevier, 2000 585
8. De Bra, P., Houben, G. J.: Automatic Hypermedia Generation for ad hoc Queries on Semi-Structured Data. Proc. of the Fifth ACM Conference on Digital Libraries, ACM, 2000 585
9. Frasincar, F., Houben, G. J., Vdovjak, R.: An RMM-Based Methodology for Hypermedia Presentation Design. Proc. of the Fifth East-European Conference on Advances in Databases and Information Systems (ADBIS '01), Springer-Verlag, 2001 585, 586, 587
10. De Bra, P., Houben, G. J., Wu, H.: AHAM: A Dexter-based Reference Model for Adaptive Hypermedia. Proc. of the 10th ACM Conference on Hypertext and Hypermedia (Hypertext '99), ACM, 1999 587
11. Clark, J.: XSL Transformations (XSLT) Version 1.0. W3C Recommandation, 1999 `http://www.w3.org/TR/xslt` 587
12. Ludscher, B., Papakonstantinou, Y., Velikhov, P.: A Framework for Navigation-Driven Lazy Mediators. Proc. of ACM Workshop on the Web and Databases, ACM,1999 588
13. Dublin Core Metadata Element Set, Version 1.1: Reference Description. DCMI, 1999 `http://dublincore.org/documents/1999/07/02/dces/` 588

14. Klyne, G., Ohto, H., Reynolds, F., Woodrow, C.: Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies. W3C Working Draft, 2001 http://www.w3c.org/TR/CCPP-struct-vocab/ 588
15. Beged-Dov, G., Brickley, D., Dornfest, R., Davis, I., Dodds, L., Eisenzopf, J., Galbraith, D., Guha, R. V., MacLeod, K., Miller, E., Aaron Swartz, A., van der Vlist, E.: RDF Site Summary (RSS) 1.0. RSS-DEV, 2001 http://purl.org/rss/1.0/spec 588
16. Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., Stein, L. A.: DAML+OIL (March 2001) Reference Description. W3C Note, 2001 http://www.w3.org/TR/daml+oil-reference 588
17. Hunter, J., Lagoze, C.: Combining RDF and XML Schemas to Enhance Interoperability Between Metadata Application Profiles. Proc. of the Tenth International World Wide Web Conference (WWW10), ACM, 2001 588
18. Biron, P. V., Malhotra, A.: XML Schema Part 2: Datatypes. W3C Recommandation, 2001 http://www.w3.org/TR/xmlschema-2 588
19. Kashyap, V., Sheth, A.: Schema Correspondences between Objects with Semantic Proximity. Technical report DCS-TR-301, Department of Computer Science Rutgers University, 1993, October 591
20. Mark, A. M., Noy, N. F.: Anchor-PROMPT: Using Non-Local Context for Semantic Matching. Proc. of the Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence, 2001 594
21. Maruyama, H., Tamura, K., Tamuran, K., Uramoto, N.: In: XML and Java, LMX: Sample Nontrivial Application. Addison-Wesley, 1999, 97–142 594
22. Wiederhold, G.: Mediators in the Architecture of Future Information Systems. Computer Magazine of the Computer Group News of the IEEE Computer Group Society, Vol. 25. IEEE, 1992, 38–49 595
23. Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D.: The RDFSuite: Managing Voluminous RDF Description Bases. Proc. of the Second International Workshop on the Semantic Web (SemWeb '01),WWW10, 2001 597
24. Duschka, O. M., Genesereth, M. R., Levy, A. Y.: Recursive Query Plans for Data Integration. Journal of Logic Programming, Vol. 43, No. 1, 2000, 49–73 597
25. Elkan, C. P., Monge, A. E.: The field matching problem: Algorithms and applications. Proc. of the Second International Conference on Knowledge Discovery and Data Mining, AAAI Press, 1996