OntoWebber: Model-Driven Ontology-Based Web Site Management

Yuhui Jin, Stefan Decker, Gio Wiederhold
Stanford University
{yhjin, stefan, gio}@db.stanford.edu

Abstract. Building data-intensive Web sites and especially Web portals is a costly task, which requires considerable effort for data integration and maintenance and usually does not result in many reusable components. This is mainly because most of the design is hard-coded in static or dynamic Web pages. In this paper we integrate three different approaches to create a comprehensive solution to Web site and Web portal creation dubbed OntoWebber. OntoWebber integrates (1) the explicit modeling of different aspects of Web sites, (2) the use of ontologies as the foundation for Web portal design and (3) semi-structured data technology for data integration and Web site modeling. The resulting system and methodology supports the creation of reusable specifications of Web sites. OntoWebber is the basis for creating the Semantic Web Community Portal as part of the OntoAgents project, which will help the Semantic Web research community (distributed on the Web) to exchange and share knowledge conveniently and effectively.

1 Motivation

Building *data-intensive Web sites* is a high-effort task, which usually does not result in many reusable components, mainly because nowadays most of the design is hard-coded in HTML and executable code like CGI scripts, Active Server Pages (ASP) or Java Server Pages (JSP).

Web portals is a special kind of data-intensive Web sites, which presents a large collection of information related to specific topics and are often organized by hierarchical directories. Examples of Web Portals are Yahoo!, and company portals, which present available resources inside and outside the company to their employees to facilitate cooperation. Knowledge management and dynamic personalization are key features of these Web portals, which make the management of these portals even more demanding than that of an ordinary Web site.

Building and maintaining a portal requires considerable effort for data integration and maintenance, since quite often available information (e.g. inside a large corporation) is heterogeneous, distributed and constantly changing. Therefore it is highly desirable to automate the data integration and maintenance tasks as much as possible.

In software engineering area, design patterns [6], declarative specification approaches and modeling of software artifacts (using e.g. UML) help to generate reusable components and models – this is already partially used for modeling Web sites by approaches like WebML [3].

In the database area, semi-structured data has proven to be very successful as a means to integrate heterogeneous data sources [7]. The usefulness of semi-structured data approaches for modeling Web sites was demonstrated by Strudel Web site management system [5].

Finally, AI-centric approaches have suggested ontologies as a means to organize and present Web Portals [9] [15].

OntoWebber brings efforts from all these different areas together into a coherent system and methodology. It adopts a model-driven, ontology-based approach for declarative Web site management and data integration, and offers support throughout the life cycle of a Web site, including design, generation, personalization and maintenance. The fundamental idea behind OntoWebber is the use of ontologies as the basis for constructing different models necessary for creating a Web site.

As a demonstration of our idea, we are building the Semantic Web Community Portal (SWCP) as part of the OntoAgents project[*], which will help the Semantic Web research community (distributed on the Web) to exchange and share knowledge conveniently and effectively. The reference ontology used to structure the design of the Web site is the Semantic Web Research Community Ontology[†].

The rest of the paper is organized as follows: the next section presents the architecture and different layers of the OntoWebber system. Section 3 defines the different ontologies necessary to specify a Web site, and describes the modeling of the SWCP as a running example. Section 4 discusses the Web site generation process. Finally Section 5 and 6 present related work, conclusion and future work.

2 The OntoWebber Web Site Management System

In this section, we first describe the system architecture, with all the important software components and how they offer support throughout a Web site's life cycle. Then we introduce the Web site design methodology, which is based on an ontology-based declarative modeling approach.

2.1 OntoWebber Architecture

The architecture of OntoWebber system is shown in Figure 1, which can be decomposed into four layers:

Integration layer. The integration layer resolves *syntactic differences* between different distributed heterogeneous data sources. We have adapted approaches for integration of heterogeneous information sources (e.g. the TSIMMIS approach [7]) by establishing a joint data format over all information sources. As for the semi-structured data format, we have chosen RDF (Resource Description Framework) since it is essentially identical to the OEM (Object Exchange Model) format used in the TSIMMIS project. The key point of our approach for information integration is we convert all types of data into RDF data using the reference ontology, and only perform queries locally to the resulting data stored in the central repository, without going to the data sources.

We currently support three kinds of source data in this layer: RDF data can be directly passed to the articulation layer. Data and ontologies in the UML/XMI format are rewritten by the Data Translator using the InterDataWorking approach described in [12], before passed to the articulation layer. Data sources based on HTML are wrapped and written as RDF data using the reference ontology, thus the resulting RDF data needs no articulation and is directly stored into the repository. Please note that instance data and ontologies are handled uniformly – ontologies are just another kind of data.

* The work is supported by the Defense Advanced Research Projects Agency through the Air Force Research Laboratory.

† http://www.semanticweb.org/ontologies/

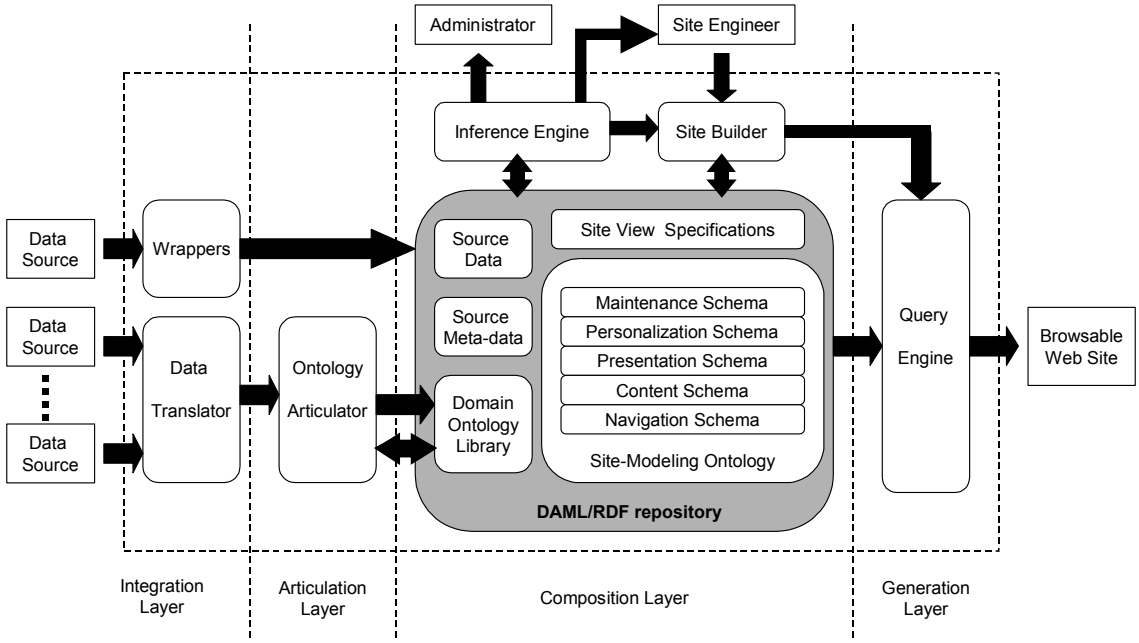


Figure 1. OntoWebber System Architecture

Articulation layer. The articulation layer resolves the *semantic differences* between the different data sources. Even if all source data have been converted into the RDF format. To be able to use the data for the site generation we need to relate the incoming data to the reference ontology of OntoWebber. Since different data providers may use different vocabularies (domain ontologies) to annotate their data, ontology articulation [13] bridges the semantic gap by establishing mapping rules between the concepts and relationships described in source ontologies to those in the reference ontology. Then the data can be queried based on the reference ontology of the OntoWebber system.

Composition layer. At this stage, the reference ontology and RDF data are available, together with articulation rules that relate the source data to the reference ontology. The ontologies for site modeling are a set of predefined schemas using DAML+OIL, available in the central repository as well. Thus, a particular *site view* consisting a set of Web pages can be created from the underlying data. A *site view specification* is a set of site models describing different aspects of a site view based on the site modeling ontologies (see Section 3). Later the site view specification is exported to the query engine (located in the next layer) to be instantiated as Web pages in the desired format. Site models can be constructed using provided software components and are materialized in DAML+OIL. Initially, a default site view is instantiated from a predefined site view specification for general public access. Other site views can be created for specific user or user group by defining their own site view specifications. Furthermore, by declaratively modeling personalization and maintenance of a site, we can achieve these tasks after the site generation phase.

Generation layer. A browsable Web site can be generated by instantiating the corresponding site view with data in the repository. This is done by the query engine, which queries the site view specification for the specific site view to be generated, at the same time queries the data to produce Web pages in desired format. There is a continuum of possibility

of query compilation to materialize Web pages for the site view. By declaratively specifying models, we can achieve all possibilities, i.e. full compilation, partial compilation, or interpretation of Web pages, and choose the optimal compilation strategy depending on various factors such as site requirements, user characteristics, etc.. Details are discussed in Section 4.2.

2.2 Web Site Design Methodology

The design of a Web site is an iterative process [3][14]. Each cycle goes through the following steps:

(1) Requirements analysis. This involves the detailed analysis of objectives of the site, data characteristics, user requirements, etc.. These aspects are the foundation for the site modeling process.

(2) Domain ontology design. The site modeling process starts from designing the default domain ontology, which also serves as the reference ontology for ontology articulation. Analysis of the data helps to extract the common elements to be included in the default domain ontology. The objectives and usage of the Web site will also influence the scope and complexity of the ontology.

(3) Site view design. A *site-view graph* is a graphical representation of three aspects of a site view, i.e., navigation, content, and presentation. The design of the site-view graph is determined by factors like characteristics, preferences, and requirements for targeted users of the site.

(4) Personalization design. Based on user analysis, different personalization elements need to be defined, including categorical information about the user, such as age, browser type, etc., and user requirements such as what operations are expected when changes occur on certain data. If some data elements of interest are missing from the site-view graph designed in step 3, we also need to go back and refine the graph.

(5) Maintenance design. Here we will not dealing with functionality maintenance, which relates to software engineering issues, such as debugging and empowering the software. We only focus on the data maintenance aspect of a Web site. Data maintenance involves manipulating data when certain data changes. Therefore, we need to find out all the anticipated changes of the data, and the corresponding actions to be performed.

3 Modeling of Web Site

From a data management perspective, a Web site here can be considered as a collection of data, including site modeling schemas, site models (i.e., instance of modeling schemas), and source data (i.e., instance of site models). Site model is a notion we use to define all the models we used in the site modeling process, each represents a different aspect of the Web site. There are altogether six types of site models which are shown in Figure 2. To facilitate the processing of these models for Web site management, these six types of site models can further be classified into two categories, site-specific and site-view-specific. If a site model for a particular Web site is *site-specific*, that means there is only one of this type of models for the Web site. Domain model, personalization model and maintenance model all belong to this category. For instance, a personalization model captures all the information about users of a Web site, therefore only one personalization model is needed for any Web site. On the other hand, there can be multiple *site-view-specific* models for a particular Web site. These models

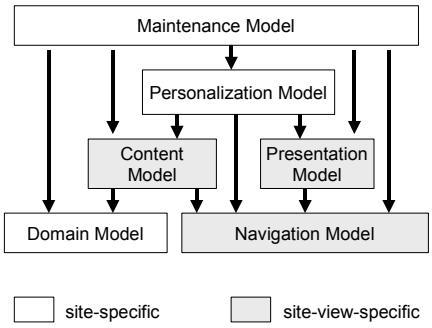


Figure 2. Site Models and Their Relationship

are only specific to a particular site view, tailored for a particular user. Navigation model, content model and presentation model belong to this category. Take content model as an example, for a particular Web site, there could be many content models, though each is associated with a specific site view. Put it another way, a particular site view specification contains a navigation model, a content model, and a presentation model. And a Web site contains multiple site view specifications.

The relationship of the site models is also shown in Figure 2. There is an arrow between two models if the source model refers to the destination model as part of its operational data. Specific to a particular site view, the navigation model specifies the navigational structure of the site view without concerning what content will be associated with primitive elements of the structure. Based on the domain model, content model then relates concepts in the domain to the primitives in the navigation model. The primitives in the navigation model can also be associated with appropriate presentation styles by the presentation model. Specific to the Web site, the domain model defines all the concepts and their properties and relationships in the domain. The personalization model handles the update of individual-dependent data according to user preferences over navigation, content and presentation aspects of their own site views. All these models are part of the operational data for the maintenance model, which not only manages source data, but the other models as well.

The distinct separation of these site models facilitates the conceptual modeling process. Designers can focus on each aspect of the site design at a time without bothering with detailed dependencies on different aspects other than those explicitly specified in the model. Models can also be reused easily, such as the reuse of favorite presentation style with different content and navigation models. The declarative specification of these models also makes it much easier to change any aspect of the site, simply by defining rewriting rules for the models.

The vocabulary (ontologies) for describing site models is a set of pre-defined site modeling schemas using DAML+OIL. Table 1 shows the relationship between models, the schemas used to define them, and meta-schemas (schemas used to define the modeling schemas).

Table 1. Relationship between models and schemas

Site model	Site modeling schema	Meta-schema
Domain model	DAML+OIL	DAML+OIL
Navigation model	Navigation schema	DAML+OIL
Content model	Content schema (and upper ontology)	DAML+OIL
Presentation model	Presentation schema	DAML+OIL
Personalization model	Personalization schema (and upper ontology)	DAML+OIL
Maintenance model	Maintenance schema (and upper ontology)	DAML+OIL

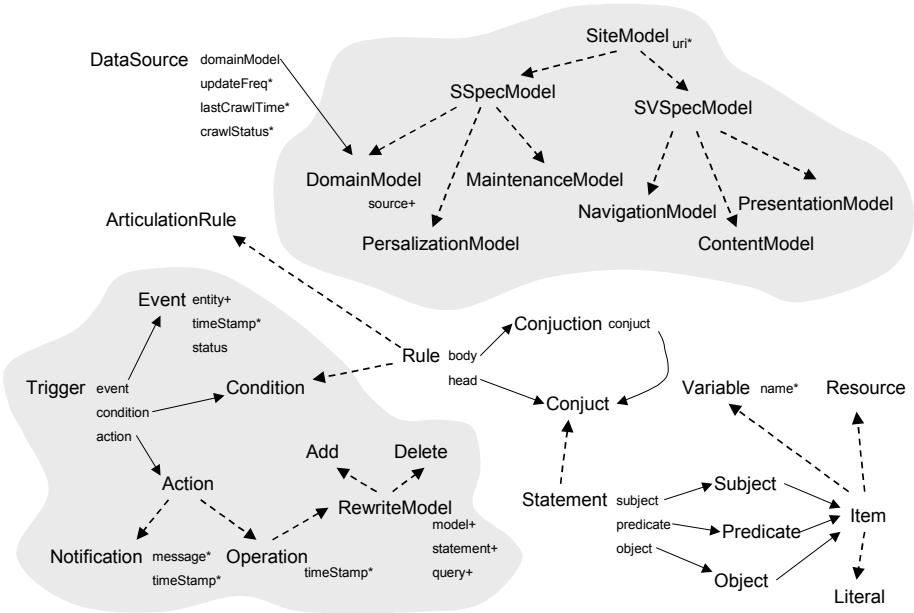


Figure 3. Upper Ontology for Site Modeling

(Note that in this paper, we will use this graphical representation to describe schemas and models, just for illustration purpose. The corresponding serialization in DAML+OIL should be straightforward. Nodes in the graph stands for classes in the ontology, the properties of the class are listed beside it. Solid arrows are used for specifying property values as instances of the pointed class, and the dashed arrows represent sub-class relationship. Asterisk sign (*) indicates the value of the property is a literal, and plus sign (+) means the property value is an instance of a certain class but solid arrow to that class is omitted for readability of the graph.)

To illustrate the process of modeling a Web site, we will present all site modeling schemas and a set of site models described using corresponding schemas for an example site view. The site view can be further instantiated with the data collected from research communities distributed on the Web, and serves as a simplified version of the SWCP.

3.1 Upper Ontology

Before we discuss all site modeling schemas and example site models, we need to define an upper ontology which contains all the necessary concepts either not captured by any of the schemas (e.g., data sources), or will be shared among multiple schemas (e.g., triggers). Figure 3 shows the graphical representation of the upper ontology.

As can be seen in the upper ontology, we define the six types of *site models* as first class objects. This makes models describing the Web site part of the processible data. Management of Web site can thus be reduced to the management of site models. These site models belong to two distinct categories, SSpecModel (*site-specific* model), and SVSpecModel (*site-view-specific* model), as we have discussed before. In the upper ontology, we also explicitly define data sources, rules, and triggers, which will be used later in defining schemas for individual site models.

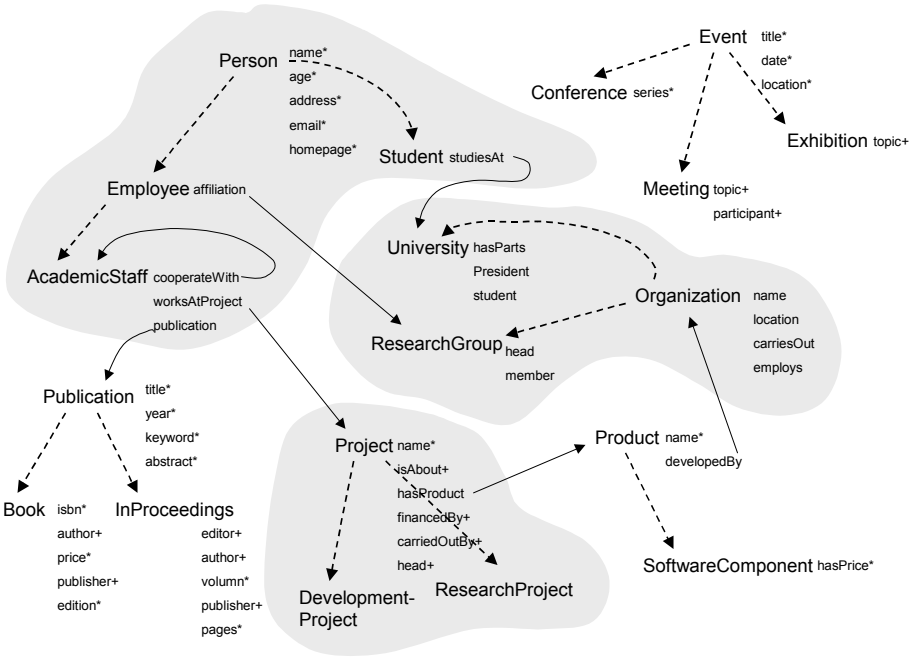


Figure 4. The domain model for SWCP

3.2 Domain Modeling

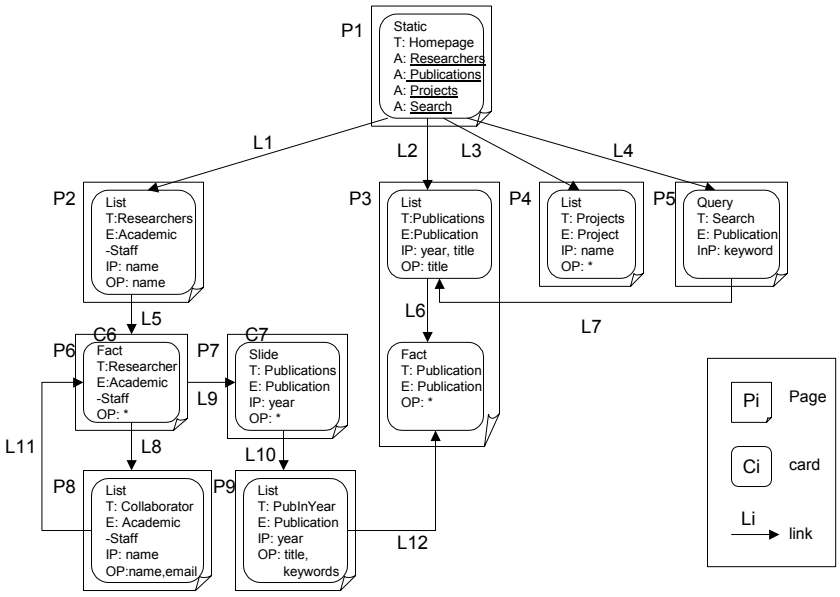
The domain model is actually an ontology constructed by analyzing the collected data in the domain, and extracting common concepts, their properties and relationships. It is used as the reference ontology in ontology articulation for mapping source ontologies to it, and as a foundation for modeling other aspects of a particular site view. The schema for domain modeling is DAML+OIL. The domain model for the example site view of SWCP is shown in Figure 4 (Note some classes and properties such as constraints are omitted due to space limit).

3.3 Site View Modeling

To facilitate the process of modeling navigation, content, and presentation of a particular site view, we have designed a graphical representation called a *site-view graph* to incorporate these three aspects of a site view. By designing a site-view graph, three models of the site view specification can be generated based on the graph, and later guide the instantiation of Web pages in desired format from the underlying data.

3.3.1 Site-view Graph

The site-view graph is a simplified conceptual model to describe hypertext on the Web. It contains a minimal set of design primitives for composing basic information structures in a typical Web site.



(T: title E: entity IP: Indexed Properties OP: Output Properties InP: Input Properties * : all properties)

Figure 5. An example of a site-view graph

Design primitives. The basic elements of a site-view graph are cards, pages and links. A *card* is the minimal unit of a site-view graph. A *page* contains one or more cards and corresponds to a physical Web page. *Links* are used to connect cards to form the navigational structure of the site-view graph. Pages are connected only through the links attached to their cards. The semantics of these design primitives are defined in the schemas for navigation, content and presentation. Each schema categorizes and attaches necessary properties to these design primitives. For instance, a card is attached with properties about coming and outgoing links in navigation schema, entity property in content schema, and font property in presentation schema.

To see how these design primitives fit together to form a complete site-view graph, an example of a site-view graph for the SWCP is presented in Figure 5. The details about the graph will be explained in the rest of the section when we describing each model in site view specification.

Web information structures. The typical information structures on the Web can be categorized into three basic types [8]:

- *Sequential*. A linear form of information flow, the simplest and most common structure. The linkage from page P6 to P7 in Figure 5 is an example.
- *Hierarchical*. A hierarchical structure involves having a page linking to lower level pages of detail. An example could be the root page P1 links to next level of pages P2 to P5 in Figure 5.
- *Associative*. This structure involves nonlinear navigation, fundamentally any structure that is not sequential or hierarchical. L7 in figure 5 is an example where the retrieval of a list of publications as search result from database helps to form the linkage between the two pages.

The information structure of a site view is usually a combination of the above structures. The provided design primitives are able to form all these basic information structures, which are the building blocks of a site view.

Minimalist Approach. Web sites that are too complicated for their intended user will likely frustrate the user and make site maintenance a nightmare. Sophistication could add value if applied appropriately, but still it depends on the nature of the Web site, and the intended user. When modeling a site view, we explicitly offered a set of guidelines, in favor of a predictable and consistent user interaction and ease of maintenance.

A minimalist design includes the following ingredients. A site-view graph always starts from a default root page, with links to the first level of pages. Content information is categorized and aggregated by cards and pages. Only one type of content is contained in each card. Navigation is made possible only through links. Moreover, a site map is generated out of the site-view graph, which can be presented in a separate page or as part of the root page. This site map makes the structure of the site visible to users, and gives the location information so they know where they are and where they can go.

3.3.2 Navigation modeling

The navigation model of a site view is a description of the site-view graph with respect to how the cards and pages are connected through links, without concerning what semantics will be associated with these primitives. The schema for navigation modeling is shown in Figure 6.

We classified cards into two categories, dynamic cards and static cards. Dynamic card contains content that depends on the changes of source data, i.e., the query used to generate the content needs to be reevaluated if source data changes. A further classification of dynamic card is the following four types of card. Each represents a typical way to structure information within a card:

- *Fact Card*. Only one instance of the entity will be shown with specified output properties in the card.
- *List Card.* A list of instances of the entity will be shown, with indexes on key properties (i.e., some literal properties of the class). And each instance will be shown with specified output properties.

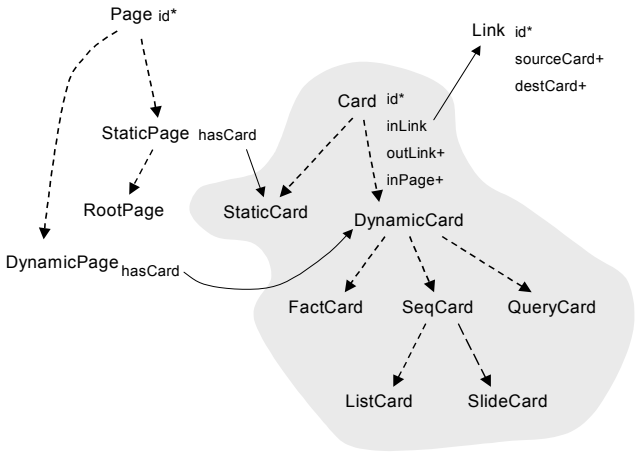


Figure 6. Navigation Schema

- *Slide Card.* A sequence of instances of the entity will be shown with specified output properties in the card, one instance at a time, and hyperlinks are created to browse nearby instances (in the order given by specified key properties) in the sequence.
- *Query Card.* A set of input properties needs to be filled out to search for entities satisfying these criteria.

The list-card and slide-card are both a type of sequence-card, which means the content of these cards is a sequence of instances. Details about how content is generated and presented in the card are discussed in content modeling (see Section 3.3.3). Static card contains content which is source data independent, such as static text, and images. A common example is the root page of a site view, which is always a static card, with predefined anchor texts leading to the next level pages. Pages are also classified into dynamic and static types according to the types of cards they contain.

Navigation model of a site view can be defined using the given schema. As an example, a portion of the navigation model for the example site view is presented in Figure 7.

3.3.3 Content Modeling

The content model associates meanings to design primitives in the site-view graph. The schema used for content modeling is shown in Figure 8. It basically specifies two aspects of content modeling. One is how to present the content in a rendered card (part of a Web page), the other is how to generate the content for a specific card.

How to present the content in a card can be explained by meanings of the attached properties to different card classes in the content schema. Each card has a property 'title' that can be used when rendering the card in Web pages. For a static card, we define all types of static elements that can be contained in the card, i.e., text, image, and anchor.

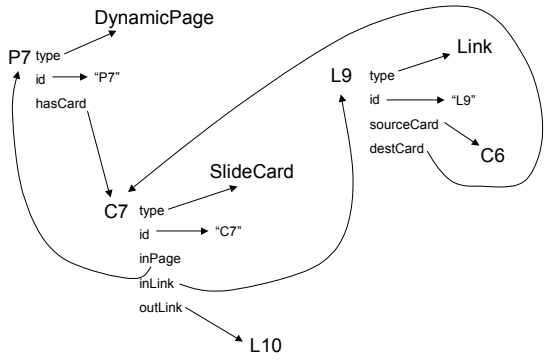


Figure 7. Navigation model for example site view

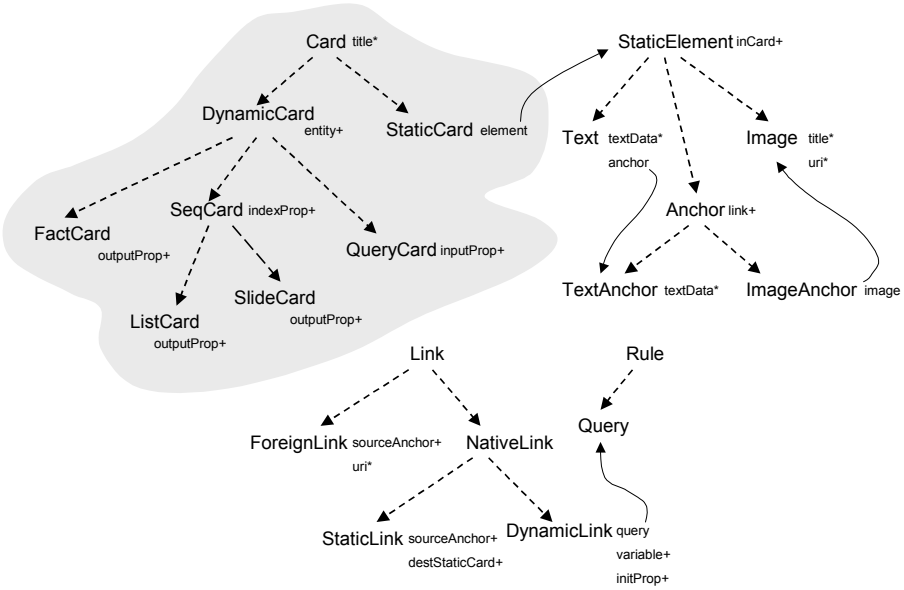


Figure 8. Content Schema

For a dynamic card, a property 'entity' takes a value as a class in the domain model. The content of the card is instantiated with instances of this class. The 'outputProp' property of a card specifies what properties of each instance are listed on the card. For sequence-card, the 'indexProp' property specifies the set of indexed properties used in ordering the listed instances. A constraint on this property (which is omitted in Figure 8) is it must take values that are literals (i.e., can not be type of 'resource'). Consider an example in Figure 5, card C7 is a slide-card, which is a type of sequence-card, it has the following listed properties: title 'Publications', which can be the name of the card when it is rendered in a Web page; entity 'Publication', which means only instances of class Publication can be contained in the card; indexed property 'year' (note we can have more properties listed to form a multi-key index), so the order of listed publication instances has to be determined by values of their 'year' property; output property '*', this means values of all the properties of publication instances are shown in the card.

The generation of content (i.e., instances of entity property of a card) can be realized by attaching certain properties to link classes. Links can be either foreign (linking to a page outside the current Web site) or native (linking to a Web page inside the current Web site). Native links can be either static or dynamic. Static links connect cards without any information flow, while dynamic links always connect dynamic cards, where the content in the destination card is determined by information passed from the source card. To instantiate the destination card with desired content, we associate three properties to a link: a *query* property, which can be assigned with a query the execution of which produces the content of the card; a *binding-variable* property, which indicates variables in the query which will be instantiated with data values passed from the source card; and *initiating-property* property, which helps to create a hyperlink in the source card. The query is a type of 'Rule' class, and the binding-variable is a type of 'Variable' class. A query will first be rewritten with binding variables replaced with data values passed from source card, then executed to produce the instances.

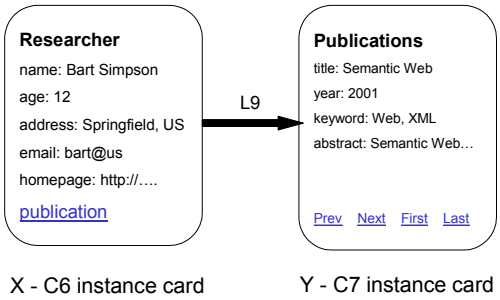


Figure 9. An example of content instantiation

An example of content instantiation is shown in Figure 9, where we extracted the portion of site-view graph (see Figure 5) containing card C6, C7 and link L9. In this example, we name the C6 instance card as X, and similarly, the C7 instance card as Y. Suppose X is already instantiated with an instance of class 'AcademicStaff', whose URI is 'Bart' (we use first name of the instance as the URI only for illustration purpose). And now we need to instantiate Y, which is intended to have a slide show of all instance publications of 'Bart'.

The way the instantiation is done is by specifying appropriate values to the query and binding-variable properties of link L9. The query[‡] attached with link L9 is:

```
FORALL A,P <-
      A[type->AcademicStaff] and A[publication->P]
```

The binding-variable value of link L9 is 'A'. To add a hyperlink in X, the initiating property takes the value as 'publication'. Then the query is evaluated after replaced the variable 'A' with the URI 'Bart'. Given destination card is a slide-card, a set of Web pages is instantiated with each instance in the query result (assume we are instantiating Web pages statically). These Web pages are linked together by pre-defined hyperlinks in a slide-card, such as 'Prev' and 'Next', and ordered by the indexed properties of the card C7. Finally, a static card containing an anchor which links to the first of these Web pages is created and added into X. The name of the anchor is given by the initiating property, which is 'publication'.

Another different type of query is that for link L7 in Figure 5. Because the source card is a query-card, the instances to be contained in the destination card cannot be compiled statically. The variable binding has to take place at run-time when user specifies the search criteria. The query for L7 is

```
FORALL P,K,T <-
      P[type->Publication] and P[keyword->K] and
      P[title-> T]
```

[‡] Queries and rules in this paper are written in TRIPPLE notation. TRIPPLE is an inference engine we are developing for the OntoAgents project. Note that O[P->V] stands for a statement in RDF (O,P,V).

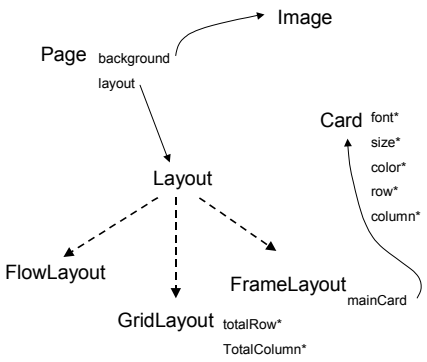


Figure 10. Presentation schema

The binding variable is 'K'. And initiating property is NULL, since no need to have hyperlink in the search page. After replacing variable K with values entered by the user, the query will return titles of all the publication instances with the specified keywords. An instance of the destination card will then be instantiated dynamically as a list of titles of publications.

3.3.4 Presentation Modeling

The third aspect of a site view is the presentation model. By associating presentation elements to design primitives in the site-view graph, the presentation model specifies the look-and-feel of the Web pages generated from the site view. The schema for creating presentation models is shown in Figure 10.

The background of the page can be chosen as an instance of images. Card and page both have style elements like font, color, etc., with the elements of card, if present, overriding those of the containing page. The layout of cards in a page can be one of the three types. The *flow layout* (default layout) arranges all the cards in a row, the *grid layout* maps these cards to certain position on the screen, and the *frame layout* places one card, denoted by mainCard property, in the static frame, and other cards in the dynamic one.

3.4 Personalization Modeling

Personalization in our approach includes providing personalized content collection, navigational experience, and presentation style through adapting the site view to the needs of users. This is accomplished by manipulating all three models of the site view. The schema for personalization modeling is shown in Figure 11. Users are explicitly modeled by three properties, i.e., capacity, interest and request.

Capacity property describes basic information about the user, such as age, preferred browser type, connection speed, etc.. The capacity of a user can be used to assign user to certain predefined groups, and adjust presentation styles for better online experience. *Interest* aspect of the user includes the three models of the site view of the user. These models specify the user's site view and can be rewritten to specify a new site view. And *request* property defines triggers which will be fired if certain conditions are satisfied, the actions of the trigger is either update the site view by model rewriting, or notify user by messages.

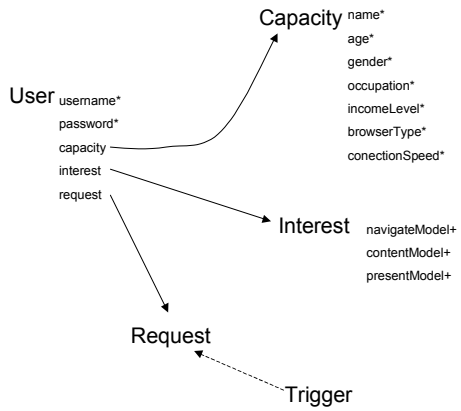


Figure 11. Personalization schema

Basically, two types of personalization can be provided by the system. The *fine-grained personalization* is achieved by defining the personalization model using the above schema. A *coarse-grained personalization* can also be used, by assigning user to specific user group. For each user group, a particular site view and personalization model is constructed. This can be modeled by defining similar properties for user groups as for users in the above schema and add relationship between user and user group. In the course-grained personalization, the site view of the user will not be updated as often as in the fine-grained personalization, since it only changes when group view changes. This helps to reduce workload of the system considerably.

3.5 Site Maintenance Modeling

Maintenance of a Web site typically falls into two categories, content maintenance and functionality maintenance. The later can be further classified into corrective, adaptive, and perfective maintenance [14]. Here we will focus on the content maintenance aspect, since the functionality part is more of a software-engineering issue, while what we are interested is the data management of a Web site.

From data management point of view, Web site maintenance can be regarded as a manipulation of data when certain data changes. Therefore, we come to a simple schema for maintenance modeling, which is shown in Figure 12.

Administrator is the target object of maintenance rules, and will update the source data, meta-data, and site view specifications according to the fired triggers. There are basically two types of maintenance rules.

User-oriented rules. Administrator is a super user, who has the authority to initiate actions that influence users and user groups with certain properties. It can be achieved by rewriting the personalization model. An example of these rules could be “if any instance of Book about Semantic Web (e.g. title or keyword contains the phrase) has been published, re-compute the site views of users who are working on a project about DAML.

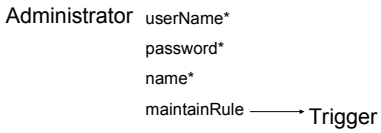


Figure 12. Maintenance Schema

Site-oriented rules. Administrators can also perform operations on meta-data, which provides basic information about data sources (the frequency of updates, crawl status, etc.) and about the Web site itself (number of users and user groups, different versions of ontologies, etc.). This is basically handled by rewriting the maintenance model. For instance, a rule of this type could be "if source A has changes weekly, and today is six days after the most recent crawling of the source site, then schedule the crawling for today".

4 Web Site Generation

The generation of a browsable Web site is an instantiation of a particular site view. It can be described as a two-phase process. First, integrity constraints are verified over the site view specification. Second, Web pages are materialized by querying the source data based on the specified site models.

4.1 Constraint Verification

Constraint verification on a traditional Web site is a difficult task. For example, the checking of whether each page is reachable from the root page are usually performed by manually following each link in all the pages, which takes much effort especially when the Web site contains a large number of pages. The constraints of the Web site generated using our approach can be easily verified. Since ontologies (i.e., site schemas and models) are explicitly specified using DAML+OIL, constraint verification becomes a direct application of semantics of the ontologies. On the other hand, it is also part of the reasoning facility provided by the inference engine, as we define rules and verify them against the ontologies. Note that a complete formalization of the ontologies is undesirable as it takes considerable amount of effort, and offers no obvious benefit.

There are mainly three types of integrity constraints to be verified against each of the three models in the site view specification.

Structural constraints. They dictate all the legal patterns of navigation in the site-view graph. These are verified against the navigational aspect of the site view. Examples of the constraints could be expressed as the following rules:

(a) Every dynamic card has at least one incoming link.

```
<- FORALL C C[type->DynamicCard] ->
   EXISTS L L[type->Link] and C[inLink->L] and L[destCard->C]
```

(b) Every card is reachable from the root page, this is expressed by defining a property called 'reachable' using recursion, and using this property to check the constraint.

```
FORALL X,Y X[reachable->Y] <-
      X[type->Card] and Y[type->Card] and
      EXISTS L X[inLink->L] and Y[outLink->L]
FORALL X,Y,Z X[reachable->Z] <-
      X[type->Card] and Y[type->Card] and Z[type->Card]
      and X[reachable->Y] and Y[reachable->Z]
<- FORALL P,X P[type->Rootpage] and X[type->Card]
      -> EXISTS Y Y[type->Card] and P[hasCard->Y] and
      X[reachable->Y]
```

Semantic constraints. The content model of a site view is validated based on semantic information. Example constraints could be:

(a) The entity associated with a query-card must match the entity associated with its destination card, since the search result should relate to the same entity.

```
<- FORALL X,Y,L,E1,E2 X[type->QueryCard] and Y[type->Card] and
      L[sourceCard->X] and L[destCard->Y] and X[entity->E1] and
      Y[entity->E2] -> E1 = E2
```

(b) The value of 'initProp' property of a dynamic link should be one of the properties of the entity associated with the destination card (see Section 3.3.3).

```
<- FORALL L,I,D,E L[type->DynamicLink] and L[initProp->I]
      and L[destCard->D] and D[entity->E] -> I[domain->E]
```

Presentational constraints. The look-and-feel of Web pages can be different on different platform, browser, and even depends on the connection speed. An example of how these presentation elements influence the site view could be: do not allow usage of background image and frame layout in the site view given a low connection speed.

```
<- FORALL P,U,C,S U[type->User] and U[capacity->C] and
   C[connetionSpeed->'Low'] and P[type->Page] ->
   NOT EXISTS X P[background->X] and
   NOT EXISTS Y P[Layout->Y] and Y[type->FrameLayout]
```

Constraints involving multiple site models are also possible. An example is to verify that queries produce a full text version of a site view. This requires defining rules to check both the static elements in the content model and page elements in the presentation model.

4.2 Site View Instantiation

After integrity constraints verification has been done, models in the site view specification will be compiled into Web pages to produce a browsable Web site. A typical instantiation is carried out by the query engine which generates HTML pages with data from the repository, based on the navigation and content model, and produce CSS style-sheets based on the presentation model.

Because all models are specified declaratively, there exists a continuum of possibilities in page compilation, which did not exist in any prior system. The possibilities range from full pre-generation of HTML pages at compile time (full compilation), to partial compilation (e.g.

a Java Servlet builds the HTML page, and instantiate pages at runtime out of data stored in database), to full interpretation (the models are interpreted by an interpreter, which creates the all HTML output at runtime).

Different solutions have different application areas. Pre-generation of the HTML code is desirable if the load of the Web site is extremely high and it is too expensive to access a database when a request comes in. Unfortunately pre-generation is also the most inflexible solution – a change to the source data may require a large number of the site views to be re-instantiated. An example of a full pre-generation system is Strudel. Partial compilation is desirable, if the load of the Web site is in balance with the need to reflect updates of the database quickly in the generated HTML code. Torri is an example of a Web site management system realizing partial compilation. The models defined for the Web site are used to generate JSP scripts. Finally, full interpretation of the models is the most inefficient, but also the most flexible way of Web site creation. The MyYahoo.com Web site [10] can be regarded as an interpretative system. Changes to the layout and the selection of modules are directly reflected in the personalized view of the Web site.

5 Related Work

Given the amount and complexity of the Web content, research has been conducted in the large context of extending database techniques for data on the Web, particularly with the goal of facilitating the creation and maintenance of data intensive Web sites. A few systems, such as ARANEUS[11], AutoWeb[4], Torri[2], have been developed using a model driven approach, which is adapted from classical database and hypermedia design methodologies. These systems have their own data models, query languages, and sets of CASE tools to facilitate the process of wrapping, modeling, generation and querying. But the common theme is a high-level description of a Web site by distinct orthogonal dimensions. Those dimensions include the modeling of information content, page composition, navigation, and presentation. Personalization by means of user modeling and business rule management has also been added in later systems. However, none of the approaches deal with integration of heterogeneous data sources, which is what OntoWebber is explicitly designed for. Since in OntoWebber ontologies and site models for different aspects of the site are both expressed in RDF, they can be rewritten and queried statically or dynamically, which is another feature not present in these systems, because they do not have a unified data model like RDF and do not construct ontologies for every aspects of site modeling.

Other systems like Strudel[5] and its variant Tiramisu[1], address the problem of data integration, and establish a separation over the Web site data management, content and structure specification, and visual presentation. The emphasis of their approach is the declarative specification of a Web site's structure and content. By defining a declarative query language, they showed how the generation of a Web site can be automated by querying the data graph to construct a site graph (i.e., site view), and how integrity constraints of the generated site can be enforced by reasoning over the site structure. However, they do not have fine-grained modeling hierarchies for each aspects of a Web site as in OntoWebber. Site view in these systems is simply a data graph containing all the navigation and content information. The presentation style is hard coded in the HTML templates. While in OntoWebber the site view is specified as three distinct models to separate the aspect for navigation, content, and presentation. The rewriting and reusing of these models eases the maintenance work and

enables flexible personalization. Since strudel does not concern the aspects of site maintenance and personalization, it is actually only an implementation tool, not a management system.

AI approaches like SEAL[9] focus on the presentation of portals based on a domain ontology. However, SEAL does not enable the creation of a site by modeling the site itself, and also offers no support for the integration of heterogeneous data sources. Although SEAL provides a set of tools (e.g. the Java based rule engine SiLRI), the underlying site still needs to be created and programmed in a conventional manner.

The fundamental difference between the OntoWebber approach and previous approaches is the integration of three different aspects into a coherent framework: (1) integration of heterogeneous data sources based on a formal model for semi-structured data, (2) explicit ontologies, which help to structure, create and generate the site, and (3) a rigorous modeling methodology, which helps to create reusable models.

6 Conclusion and Future Work

We proposed a model-driven ontology-based system architecture for creating data intensive Web sites and portals. Our approach combines the advantages of different technologies: Semi-structured data technology is used to integrate heterogeneous data sources; Declarative models help to define a Web site without hard-coding design into static or dynamic Web pages; Ontologies are used to provide access to the underlying data and guide the modeling process.

Future work includes the development and integration of all the software components of the system. We have already developed a number of tools, such as Web crawler, ontology articulation tools, data translation tool, ontology construction tool, and RDF query and storage facilities. However, the graphical Web site modeling tool and the inference engine are still under development. Furthermore, a number of problems to be considered could be:

Management of evolving ontologies. Different versions of ontologies (i.e., schemas) might be incompatible with each other. How to manage evolving ontologies and retain consistency and usability of ontologies is a necessity for the robustness of a Web site.

Optimization strategies for site generation. Evaluation and performance measurements of dynamic and static Web site generation needs to be investigated. And a set of optimization strategies can thus be defined to guide the cost-effective generation of Web site.

Adaptation to handle dynamic services. Currently approach of OntoWebber only deals with static information sources. With UDDI[§] and Microsofts.NET initiative, more and more dynamic Web services will be available, which will be integrated into portals and Web sites. OntoWebber needs to define appropriate ontologies to be able to handle dynamic services as well.

References:

[1] Corin R. Anderson, Alon Y. Levy, Daniel S. Weld: Declarative Web Site Management with Tiramisu. WebDB (Informal Proceedings) 1999: 19-24.

[2] Stefano Ceri, Piero Fraternali, Stefano Paraboschi: Data-Driven, One-To-One Web Site Generation for Data-Intensive Applications. VLDB 1999: 615-626.

[§] http://www.UDDI.org

[3] Stefano Ceri, Piero Fraternali, Aldo Bongio: Web Modeling Language (WebML): a modeling language for designing Web sites WWW9 Conference, Amsterdam, May 2000.

[4] Piero Fraternali, Paolo Paolini: A Conceptual Model and a Tool Environment for Developing More Scalable, Dynamic, and Customizable Web Applications. EDBT 1998: 421-435.

[5] Mary F. Fernandez, Daniela Florescu, Alon Y. Levy, Dan Suciu: Declarative Specification of Web Sites with Strudel. VLDB Journal 9(1): 38-55 (2000).

[6] Gamma, E., Helms, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[7] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and Jennifer Widom. "Integrating and Accessing Heterogeneous Information Sources in TSIMMIS". In *Proceedings of the AAAI Symposium on Information Gathering*, pp. 61-64, Stanford, California, March 1995.

[8] Charles J. Lyons, Essential Design for Web Professionals, Prentice Hall, 2000.

[9] A. Mädche, S. Staab, N. Stojanovic, R. Studer, Y. Sure. SEAL - A Framework for Developing SEmantic portALs. In: BNCOD 2001 - 18th British National Conference on Databases. Oxford, UK, 9th - 11th July 2001, LNCS, Springer Verlag, 2001.

[10] Udi Manber, Ash Patel, John Robison: Experience with Personalization on Yahoo! Communications of the ACM Vol. 43, No. 8 (August 2000), Pages 35-39.

[11] G. Mecca, P. Merialdo, P. Atzeni, V. Crescenzi The (Short) Araneus Guide to Web-Site Development - Second Intern. Workshop on the Web and Databases (WebDB'99) in conjunction with SIGMOD'99, May 1999.

[12] Sergey Melnik, Stefan Decker: A Layered Approach to Information Modeling and Interoperability on the Web. ECDL 2000 Workshop on the Semantic Web. 21 September 2000, Lisbon Portugal.

[13] Prasenjit Mitra, Gio Wiederhold, Martin L. Kersten: A Graph-Oriented Model for Articulation of Ontology Interdependencies. *Proceedings of the 7th International Conference on Extending Database Technology, EDBT 2000*, March 2000 Springer Verlag.

[14] Tomas A. Powell, David L. Jones, Dominique C. Cutts, Web Site Engineering Beyond Web Page Design, Prentice Hall, 1998.

[15] Steffen Staab, Jürgen Angele, Stefan Decker, Michael Erdmann, Andreas Hotho, Alexander Mädche, Hans-Peter Schnurr, Rudi Studer, York Sure. Semantic Community Web Portals. In: WWW9 / Computer Networks (Special Issue: WWW9 - Proceedings of the 9th International World Wide Web Conference, Amsterdam, The Netherlands, May, 15-19, 2000), 33(1-6): 473-491. Elsevier, 2000.