Information Services for the Web: Building and Maintaining Domain Models

Scott Kerr
Department of Computer Science
University of Toronto
10 King's College Road
Toronto, ONT M5S 3H5, Canada
skerr@cs.toronto.edu

Avigdor Gal
Department of MSIS
Rutgers University
94 Rockafellar Road
Piscataway, NJ 08854-8054
avigal@rci.rutgers.edu

John Mylopoulos
Department of Computer Science
University of Toronto
10 King's College Road
Toronto, ONT M5S 3H5, Canada
jm@cs.toronto.edu

Abstract

The World Wide Web is serving as a leading vehicle for information dissemination by offering information services, such as product information, group interactions, or sales transactions. Three major factors affect the performance and reliability of information services for the Web: the distribution of information which has resulted from the globalization of information systems, the heterogeneity of information sources, and the sources' instability caused by their autonomous evolution. This paper focuses on integrating existing information sources, available via the Web, in the delivery of information services. The primary objective of the paper is to provide mechanisms for structuring and maintaining a domain model for Web applications. These mechanisms are based on conceptual modeling techniques, where concepts are being defined and refined within a meta-data repository through the use of instantiation, generalization and attribution. Also, active databases techniques are exploited to provide robust mechanisms for maintaining a consistent domain model in a rapidly evolving environment, such as the Web.

1 Introduction

The research field of Cooperative Information Systems intends to develop concepts, techniques and methodologies for building information systems which integrate legacy software systems and align them with current business processes and organizational objectives [8]. This paper focuses on one aspect of this integration, namely the integration of existing information sources in the delivery of information services. The information sources may include databases, formatted or ASCII files and other computer-based data. These sources may be multiple, distributed and heterogeneous. They may also contain legacy data in that their original designers are long-gone and their semantics are only partially understood.

Delivery of information services under such circumstances is hampered by a host of technical issues which range from communication protocols and syntactic interoperability among different legacy systems, to semantic issues which deal with the meaning of the data that are being integrated. Researchers and practitioners alike are coming to realize that there can be no solution to the delivery of information services unless one tackles head-on the latter problem. For our research, as with that of many others, semantic issues in the delivery of information services are addressed through the use of a domain model which represents information about the application and describes the contents of each information source relative to this domain model.

The idea of using a domain model as a means towards integrating several heterogeneous databases is well known and well accepted. The new challenges that arise for cooperative information systems in general, and the World Wide Web in particular, is that this domain model needs to be built bottom up through analysis of the information sources. Moreover, it needs to continuously evolve as new information sources become available and relevant to the information service being delivered, while existing information sources are being rapidly modified. Therefore, the primary objective of the paper is to provide mechanisms for structuring and maintaining a domain model for World Wide Web (hereafter Web) applications. These mechanisms are based on conceptual modeling techniques, where concepts are being defined and refined within a metadata repository through the use of instantiation, generalization and attribution. Also, active databases techniques are exploited to provide robust mechanisms for maintaining a consistent domain model in a rapidly evolving environment, such as the Web.

Several works in this area, e.g. WebSQL [7], have proposed using the *structure* of the Web for generating information services, yet none of the these models provide the essential mechanisms to overcome the gap between structural (syntactic) interoperability and semantic interoperability. Some attempts in related areas (e.g. [1]) to provide "semantic coating" to flat files bypass the maintenance problem by generating a virtual database schema that is materialized only at retrieval time. However, the need for a designer intervention, and the shaky availability of Web pages cannot guarantee real-time processing to provide the required semantic model. Several commercial products, e.g. HotSuite [5], provide the capability of mapping a Web site and generating a Web site map upon request. These tools do not provide semantic capabilities and therefore lack the domain model aspect of the proposed architecture. Also, once constructed, the map sites remain static unless a request for remapping arrives from the user, while our model provides a reactive propagation of Web modifications to the semantic level. WAG [2] allows the user to query the Web, rather than browsing it, and therefore attempts to provide a transparent layer for having Web-based information services. Nevertheless, WAG uses a database which is far less expressive than a repository (which is used in this work). Also, WAG is passive and does not provide any mechanism for propagating Web modifications to the domain model, and therefore it falls into the same category as the commercial site-map generators mentioned above.

The main contribution of the paper lies in the provision of an architecture for semi-automatic generation and maintenance of user-oriented, semantic-based domain models that describe distributed heterogeneous information sources. The rest of the paper is organized as follows. The proposed mechanisms and implementation issues are described in Sections 2 and 3 respectively. Section 4 summarizes the contributions of the paper and offers directions for future work.

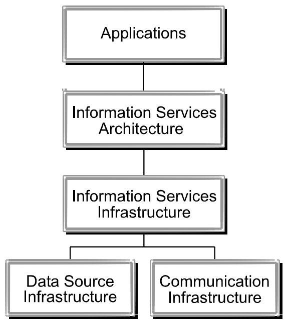
Applications
Information Services Architecture
Information Services Infrastructure
Data Source Infrastructure
Communication Infrastructure

Figure 1. The layered approach towards distributed heterogeneous information services

2 Distributed heterogeneous information services architecture for the Web

As a conceptual framework we adopt a layered approach for the design and maintenance of distributed heterogeneous information services (hereafter DHIS). Figure 1 illustrates such an approach where each layer is founded on the infrastructure provided by the lower layers. A short description of each of the layers follows:

- The data source infrastructure consists of various methods for managing data quality of a single data source using either syntactic (e.g. data mining) or semantic (e.g. metadata) tools.
- The communication infrastructure consists of the seven layers of the OSI reference model [10].
- An Infrastructure for DHIS assumes the existence of communication and data source tools and provides tools for handling information from heterogeneous distributed data sources.
- An architecture for DHIS provides an overall architecture to handle the semantic issues involved in DHIS.
- The top layer, the application layer, consists of particular services, such as data warehouses and intranet applications.

The layered structure is useful in delineating research issues and in offering a means for integrating technologies. In particular, Web technologies can be accounted for in terms of the layered structure as follows. At the data source infrastructure we include the interpretation of the semi-structured data of markup languages (e.g. HTML). The communication infrastructure consists of various interoperable and Internet-related protocols, e.g. TCP/IP and sockets. A distributed heterogeneous information services infrastructure consists of tools that utilize the Web structure to provide information services. For example, WebSQL [7] is a query language that allows the user to make use of the Web structure (e.g. hyper-links) in constructing queries by using a Java class library which responds to queries in the language. This paper focuses on a DHIS architecture for Web applications that uses CoopWARE [4] and MIRROR [6].

At the application layer we consider the design of semantic structures of intranet applications. HTML pages made available through intranets have more structure in their design than public Web pages since they share common subject domains. Therefore, we can exploit this added structure in creating domain models. In addition, the heterogeneous nature of Web sites on intranets calls for the use of a DHIS architecture. As a concrete case study we introduce an architecture for a specific domain, involving a university Web site. We examine the public Web information that a university provides, and design a DHIS that offers a reorganization of that information on a semantic basis. The collection of Web sites in a university resembles a corporate intranet both in scale and in the level of heterogeneity from one department to another. Hence, results found in applying our system to university Web sites also apply to other domains, such as corporate intranets.

2.1 The knowledge model

There are three types of information that are stored in the repository: Web artifacts, domain concepts and change propagation rules for maintaining semantic consistency as the site changes. Web artifacts are purely syntactic entities found in HTML (Hypertext Markup Language) and HTTP (Hypertext Transfer Protocol) requests. These artifacts are automatically generated and regenerated by a Web extraction tool which is supplied by a third party and integrated into our architecture. Such tools are generally syntactically based, i.e. they do not reflect the subject matter (domain) the Web pages are concerned with.

Domain concepts relate to the semantic content provided by several Web sites. The concepts vary independently between sites with the concerns and practices of each Web site manager. Therefore, even within the domain of a single organization (e.g. a university or a corporate intranet), integrating multiple Web sites into a common domain model is a challenge. The domain concepts in the repository are used by clients of the DHIS to navigate through the information contained in the Web sites. The client navigates through a semantic network from one concept to another with the ability to browse a Web page that is associated with a concept. The designer of the DHIS is responsible for "making sense" of the Web pages for the clients through an integrated domain metamodel (the semantic network). This task is supported by the system, as described in sections 2.2 and 2.3.

Domain concepts are structured as follows: A concept is associated with other concepts through domain attributes. For instance, the concept "University of Toronto" is associated with the concept "UofT Admissions" through the former concept's attribute "admissions." Also, a domain concept has one or more associated Web pages which are bound to the concept by an attribute "webPages."

Domain concepts are initially generated by examining the graph topology and content of each Web site. Each page in a site initially generates a corresponding domain concept, where its name is taken from the HTML title of the Web page. The hyper-links in the page generate the attributes of the concept, which are labeled links to other concepts.

2.2 Querying the DHIS

How does one retrieve information from a DHIS architecture? This section considers the problem for two types of users.

2.2.1 Designer queries

A designer is instructed to design a DHIS for internal use by the staff at the University of Toronto. Initially she looks at the university's home page asking the system to display all of the concepts that can be generated from the page and its hyper-links. The repository processes this as a single query.

Assume that the home page of the University of Toronto is under consideration, then the query generates the concept "University of Toronto Home" (from the page's title). It also places the URL for the home page in a attribute labeled "Provisional concept was generated from Web page" linking the domain concept with its page. Also, for each labeled link in the page, the system adds an attribute to the first domain concept. The new attribute's label is initially the same as the label on the hyper-link. The system then generates a concept for the target Web page of the hyper-link (again using the page's title as the concept name). This concept becomes the value of the attribute. The URL of the target concept's page is stored the same way as the source page's URL.
In this example the state of "The University of Toronto Home" page is initially generated as follows:

- "University of Toronto Home"
- "Welcome" → "University of Toronto: Welcome Temp"
- "About UofT" → "University of Toronto: About UofT"
- "Directories" → "University of Toronto: Contacts and Directories"
- "News, Events, Weather" → "University of Toronto: News, events, weather"

- ...
- "Academic Programs & Divisions" → "University of Toronto: Academic Programs & Divisions"

The designer deletes the superfluous concepts, e.g. links to pages that are only of transient interest, rather than ongoing concerns. The designer wishes to eliminate all attributes but "Directories," "Academic Programs & Divisions," "Policies, Documents, Committees," and "Departments". Each of these attributes points to a corresponding concept: "University of Toronto: Contacts and Directories," "University of Toronto: Academic Programs & Divisions," "University of Toronto: Policies, Documents, Committees," and "University of Toronto: Departments."

Next, the designer manually renames and refines concepts, deletes irrelevant attributes, creates new associations between concepts and artifacts and edits attribute names generated from hyper-link labels, as desired. These activities are supported by an HTML-based concept editor using CGI forms that present the concepts, attributes and Web artifacts in a unified metamodel. This concept editor runs in the designers Web browser and can thus both edit and show Web sites being conceptually "digested."

In the repository, each of the new concepts is placed within a schema requiring "abstract concepts" for each of the "concrete concepts" generated from pages. In Telos, the abstract concepts are *metaclasses* and the concrete concepts are *simple classes.* Therefore, the token "University of Toronto" instantiates the "University" simple class, "UofT Admissions" instantiates the "Admissions" simple class, etc. Telos terminology is concealed from the designer who manually elaborates the model, adding "abstract" concepts as requested. The class level of the metamodel becomes useful if associated concepts from different university Web sites are integrated into a unified semantic model.

2.2.2 End user queries

As described in Section 2.2.1, the designer first prepares a conceptual graph that is a "digest" of a corresponding set of related Web sites. Following this step, an end user (in our example, a university staff member) can browse either the conceptual graph or the Web pages associated with any concept, returning to the conceptual graph at any time.

The client is initially presented with introductory pages about using the system. One of the pages contains a hyperlink pointing to the conceptual graph's root. In our example, this corresponds to the concept "University of Toronto." The URL for this page contains a query for the corresponding object. The query also indicates that it is a client's query. The same target object is retrieved from the repository as for a designer, but it only displays information which is relevant to a client.

The response to the query is a dynamically generated HTML page showing links for the concept that is deemed relevant by the designer. These links include hyper-links from the concept page to the associated Web page(s) it was derived from. In our example these links correspond to a reorganization of the links originally presented to the designer for the University of Toronto home page. The conceptual graph can be navigated and when the end user wishes to browse the actual Web pages she can follow the attribute link (contained in the page for each concept) for the page(s) associated with it.

2.3 Model maintenance

The ever-changing environment of the Web poses serious challenges for repositories aimed at defining a "global schema" of information over autonomous, distributed and heterogeneous information sources. A typical design in such frameworks assumes the infrequent changes of metadata with respect to data modifications (e.g. [3]) and often leads to a rigid selection of data structures and programs that are heavily dependent on a specific metadata structure. Hence, in order to provide swift maintenance of Web related information services, the following two requirements should be fulfilled:

Domain model independence: A flexible data structure provides the capability of modifying the domain model structure without modifying the internals of the browsing and querying tools.

Automatic propagation: Modifications to the model are detected and propagated through the use of an automatic mechanism, to allow a continuously consistent model.

The former requirement is fulfilled through the use of a rich conceptual model as discussed in Section 2.2. In this section we provide a mechanism to ensure the latter requirement.

The detection of model modifications is done by using events. These are occurrences such as the beginning or end of an operation, which enable the flow of information regarding the state of the domain model. In this context, we support four major types of events, namely *page insertion*, *page modification*, *page removal*, and *page relocation*. An event transfers parameters to be used by services. The first three events pass a single parameter, the page's URL. The fourth event passes two parameters which correspond to the old and new URLs.

The consequences of an event detection are captured in rules. A rule is a programming mechanism which is constructed of three segments, namely an event, a condition, and an action (ECA) [12]. The semantics of a rule are as follows: when an event ev occurs, conditions $co_1, co_2, ..., co_n$

are independently evaluated, where condition co_i $(1 \leq i \leq n)$ is part of a rule r_i such that $ev_i = ev$. If co_i $(1 \leq i \leq n)$ evaluates to true, then ac_i is activated.

Whenever a page insertion event occurs, facts about the page's state are recorded in the repository. These facts include the page's HTML title and the links contained in the file (both their labels and target URLs). When a new page is inserted, the designer is notified and a provisional domain concept corresponding to the HTML title of the page is suggested to the designer by the system. This concept can be edited by the designer. For example, the initial concept assigned with "www.utorotno.ca" is "University of Toronto Home Page," and is modified to be "University of Toronto." The concept is associated with the URL of the page to enable the retrieval of the page contents as a response to a user query.

New semantic relationships are added to the semantic domain model based on the hyper-links in the new page. Therefore, the concept associated with the new Web page is related with any concept that represents a target page of a hyper-link. It is possible that one or more of these target pages are new pages as well, in which case a page insertion event is signalled for each of the new pages. All of these new pages are queued for initial examination by the designer. It is noteworthy that instead of adding a new concept, the designer may choose to supplement an already existing concept with the information from the new Web page, performing a fusion operation.[1] The new extracted information is added to the existing concept and modifications in the multiple pages associated with the concept are monitored with changes propagating to the unifying concept.

It is also worth noting that hyper-links associate a single page to a single target page. Thus by default, concepts are related through 1-1 relationships. 1-n relationships are suggested by the system when a page is parsed and contains a HTML list of hyper-links. By recognizing the "list" HTML markup, the system automatically suggests candidate 1-n relationships and an appropriate title is chosen for the new relationship. The list can be further partitioned into sublists and items can be deleted by the designer.

When a page modification event occurs, a new set of Web artifacts associated with the page is generated and compared to the existing ones. Whenever possible, changes in the Web artifacts are propagated to concepts automatically. Some changes require manual intervention, yet this intervention is kept to a minimum. Therefore, previous design choices are reinforced. For example, links that are deleted can be propagated as 1-1 relationships that are removed from the corresponding concept, or values removed from 1-n relationships. If the page's title changes or there is a substantial

[1]The fusion operation was also introduced in the LOREL system [11]. However, as opposed to the approach taken in LOREL, the fusion in our architecture fuses two object through the use of multiple inheritance.

Rule r_1:	
Event:	PageInsertion(URL)
Condition:	InScope(URL)
Action:	AddPage(URL)
	State:=ExtractPageState(URL)
	RecordPageState(State)
	SuggestDomainConcept(URL, URL.title)
	AddConcept(URL.title)
	Associate(URL, URL.Title)
Rule r_2:	
Event:	DomainConceptIdentified(URL, Concept)
Condition:	NewConcept(Concept)
Action:	AddConcept(Concept)
Rule r_3:	
Event:	DomainConceptIdentified(URL, Concept)
Action:	IdentifyRelationships(URL, Concept)
Rule r_4:	
Event:	DomainConceptIdentified(URL, Concept)
Condition:	URL.title$\neq$Concept
Action:	Disassociate(URL, URL.title)
	Associate(URL, Concept)
Rule r_5:	
Event:	DomainConceptIdentified(URL, Concept)
Condition:	Ingoing(URL.title)$\leq$1 and URL.title$\neq$Concept
Action:	Disassociate(URL, URL.title)
	RemoveConcept(URL.title)

Table 1. Domain model maintenance rules (partial)

change in the page's content, the designer is prompted to examine the previous and current state of the page to decide on the appropriate manual intervention.

When a link that points to a new page is added, the page insertion event is signalled. Links to new pages generate a queue of new pages to be examined by the designer. To prevent multiple page insertions for the same page, the queuing mechanism uses the pages' URLs as a unique identification key to verify that two occurrences of page insertion events of the same page are not attempted. This mechanism circumvents the problem of lack of concurrency control in the rule system.

A page removal event results in removing a page from its associated domain concept in the repository. Therefore, there may be domain concepts for which there is no longer an associated page. Any recorded hyper-links in other pages that point to the removed page are considered to be "broken" links, and therefore they are not reflected in the semantic domain model. It is worth noting that the domain concept is not deleted from the repository, even though its associated page is, since the domain model is independent of its associated

Web artifacts. Rather, the concept becomes purely conceptual, i.e. having no correspondents in the Web pages.

Finally, a page relocation results in updating the repository to include the new URL for each instance where the obsolescent URL was mentioned.

Table 1 provides a partial set of rules for maintaining the domain model (as represented in the repository) consistent with respect to the Web site. Rules r_1-r_5 handle the page insertion scenario. Rule r_1 sends a request to identify a domain concept. The request includes a default concept generated from the Web page. The designer may choose to adopt the suggested concept or to provide an alternative concept. When such a concept is identified (see the event of r_2-r_5), a new concept is added if needed, the concept is associated with the URL and the relationship information generated from the Web page and edited by the designer is added to the repository. The Associate service and the IdentifyRelationships service seek the concept in the repository and is possibly suspended until the concept is added to the repository. It is worth noting that the execution of rule r_1 is conditioned upon the Web page being part of the scope. This mechanism prevents unbounded processing of Web pages, as a result of adding pages to the repository due to hyperlinks.

3 Implementation issues

In this section we describe in more detail some of the implementation issues involved in designing an architecture for a DHIS. Sections 3.1 and 3.2 discusses data and tools integration, respectively, while the manipulation of Web artifacts is discussed in Section 3.3.

3.1 Data integration

One of the problems faced in creating a DHIS is the fact that it must integrate preexisting, independently developed information sources executing under autonomous, asynchronous control. In traditional data integration, data sources are databases, so integration consists of developing a global schema that incorporates the component schemata. For the DHIS, information sources are the Web-sites themselves which are queried using a "Web extraction tool." However, within the DHIS, there is another integration process needed, since it is made up of several modeled components as well. Each has its own data model and domain model. The challenge is to combine these in a way that places them under a single metamodel. These modeled systems include:

1. One or more Web extraction tools. Each such tool supplies the DHIS with data on Web artifacts.
2. A HTML/CGI form-based concept editor. This tool displays the concepts generated from the Web artifacts and allows them to be manually edited by the DHIS designer.
3. An ECA rule engine. The control logic for this is the CoopWARE rule-base (described in Section 2.3) and related objects (CoopWARE's components, services, and actions).

Generally speaking, data models of any modeled system may vary if, for example, one system uses an object-oriented data model while another uses a relational data model. In a DHIS, the independently developed Web extraction tool has its own data model while its domain model consists of Web artifacts. Also, the domain models built in the concept editor vary with the subject domains of the Web-sites examined. The editor's data model is that of the conceptual modeling language Telos [9]. The data model of CoopWARE is also that of Telos and the domain model of CoopWARE is a generic ontology of events, rules, conditions and actions that applies to distributed cooperative systems in general. The Web DHIS-specific application of the CoopWARE model includes subclasses that are specific to the application domain.

In order to integrate these variations, we use the rich data model of Telos, since it can express a relational data model, an object-oriented data model and even richer modeling frameworks. It supports classification, generalization and aggregation, multiple levels of instantiation, multiple instantiation, type narrowing/widening, attribute categories and relationships, etc. This richness is essential for data integration since the heterogeneity of tools' models requires flexibility and precise control of data relationships.

For integration of executing tools, we also use a methodological framework called MIRROR (Metamodel Integration for Reflective, Reactive, Observational Repositories) [6]. In addition to modeling systems in Telos, MIRROR employs the concept of an active metamodel to provide a systematic approach for organizing tools' data access methods and coordination of their control. The metamodels are active in that, for each attribute of an object in the conceptual model of a tool, there is an executable routine to populate it. Also, dependencies in the flow of data between tools are also included in the metamodel in a declarative way, using ECA rules, so that objects in the model that depend on data from another tool communicate with objects in that tool's metamodel.

In the MIRROR framework, each tool is modeled independently of other tools. This produces a "wrapper" metamodel which consists of sets of metaclasses, classes, and instances to represent objects in the tools' domain. Objects in one wrapper do not reference objects in another wrapper, although wrappers may inherit simple classes from others wrappers. A mediator metamodel integrates these wrappers into a global model, using inheritance and instantiation to fuse objects from disparate domains. The isolation of dependencies between tool models in wrappers is advantageous in that the wrappers can then be reused in other integrated applications.

3.2 Tool integration

The layered approach discussed in Section 2 isolates the data source and network infrastructures from the DHIS infrastructure. The DHIS is implemented as a Java servlet since this offers multithreaded support for network-centric operations. MIRROR is similarly implemented in Java binding Telos attributes to Java instance methods. This allows CoopWARE services to similarly be represented as methods, accepting parameters at run-time.

Figure 2 shows how the DHIS servlet is designed, using the layered approach as a conceptual framework. In our system, HTML parsing and high-level network interactions are delegated to the Web extraction tool chosen, namely WebSQL [7]. WebSQL resides (as a Java class library) in the DHIS infrastructure layer, and therefore this functionality can then be isolated from a particular DHIS application which simply supplies queries and parses returning WebSQL tuples. Similarly the query processor resides at this layer receiving queries from clients (which simply retrieve repository objects directly), or designers (which are sent to the HTML/CGI-based editor). Also, the DHIS application layer which is domain-specific is isolated from the DHIS architecture layer, consisting of the generic MIRROR and CoopWARE implementations – both of which remain invariant and isolated from the domain-specific data elaborated by the designer and the Web artifacts objects in the repository created from WebSQL.

The interactions between the domain concepts, the Web artifacts and the generic MIRROR and CoopWARE implementations are elaborated in Figure 3. A CoopWARE component called WebMonitor is an asynchronous Java thread executing in the servlet and responding to changes detected in WebSQL tuples. Figure 3 shows how data streams flow through the MIRROR metamodel integration process. The mediator metamodel integrates all the Telos models of the DHIS, while keeping the respective wrapper metamodels isolated so that all direct dependencies between models occur in the mediator. The generic CoopWARE metamodel is the CoopWARE metaclasses that are domain-independent. These are instantiated as application-specific Telos simple classes in the mediator. The WebSQL and "Domain Concepts" wrappers do not reference objects beyond their partitions; only subclasses do this. This allows the superclasses to be used in other applications with clean separation.

WebSQL queries are requested asynchronously by WebMonitor and these queries result in data about artifacts represented as tuples made up of strings. These data items are used by WebMonitor to instantiate corresponding Web-oriented Telos simple classes ("WebDocument," "HyperLink", etc.) in the WebSQL wrapper metamodel. When a new page instance is generated, a corresponding CoopWARE "PageInsertion" event is instantiated in the mediator metamodel and sent to the CoopWARE rule engine. Rule r_1 (as described in section 2.3) is applied and a "SuggestDomainConcept" action is generated. This causes a corresponding domain concept instance to be generated automatically in the "Domain Concepts" metamodel, as described above. These concepts are then presented and edited in the concept editor by the designer.

3.3 Web artifacts and domain concepts

This section elaborates on some of the issues related with Web artifacts and domain concepts. In particular, we discuss the representation of Web artifacts in an active wrapper metamodel in Section 3.3.1 and domain concepts in Section 3.3.2.

3.3.1 Representing Web artifacts in an active wrapper metamodel

This section provides implementation details regarding the association of Web artifacts with domain concepts. Depending on the Web extraction tool chosen, the MIRROR metamodel classes may include Web artifacts for the following items:

Web documents (e.g. HTML pages), URLs: uniform resource locators pointing to pages, HTML titles, Web Document "last modified" dates, Web Document anchors: HTML tags containing pointers (e.g. to embedded images), hyper-links in Web Documents (consisting of URLs and their labels), hyper-link labels, file names (as part of URLs), file formats/MIME types (e.g. plain text, HTML, postscript), Web sites (corresponding to the "host" portion of a URL), and finally "name" anchors within pages (e.g. <a Name="Introduction"></a>).

WebSQL either recognizes these items or they can be interpolated from relevant strings (e.g. URLs) that are immediately available from the query context. The WebSQL tool consists of a Java class library with a corresponding API. In it, among other services, a class WebSQLServer uses query strings and returns sets of tuples according to WebSQL syntax. These tuples contain separated string fields. In the servlet, the strings are used to create instances of MIRROR classes that correspond to the listed artifacts above. MIRROR classes are Telos classes with executable routines bound to each of their attributes (Java methods in our implementation). By writing the appropriate set of MIRROR classes for a tool, we can wrap it in an active, self-populating

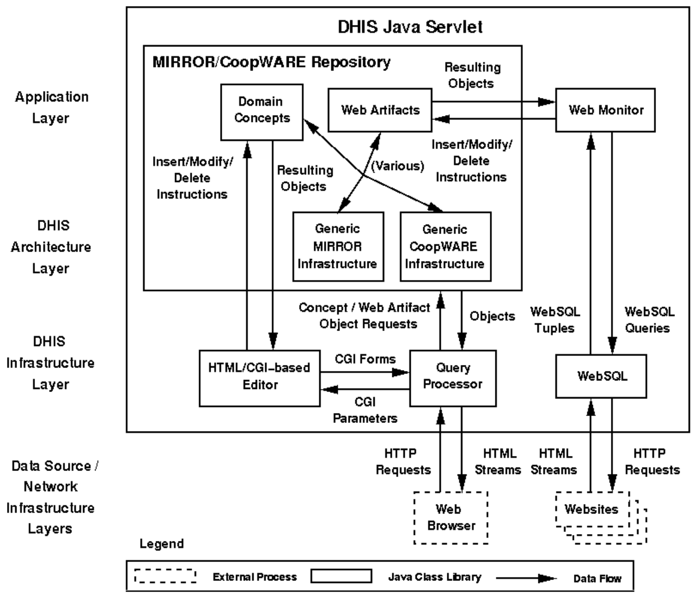

Figure 2. Query processing and repository maintenance for a Web DHIS

TELL CLASS WebDocument	TELL CLASS HyperLink
IN WebArtifactClass	IN WebArtifactClass
WITH	WITH
WebAttribute	WebAttribute
url : URL;	sourceDocument : WebDocument;
fileName : string;	label : string;
WebSite : WebSite;	url : URL;
protocaolAccessedBy : Protocol;	targetDocument : WebDocument
title : string;	conceptualAttribute
links : HyperLink;	domainAttribute : DomainAttribute
anchors : NonLinkAnchor;	END HyperLink
lastModified : string	
conceptualAttribute	
concepts : Concept	
END WebDocument	

Table 2. MIRROR wrapping classes

metamodel that can then be integrated with MIRROR metamodels for other tools and information sources. Table 2 illustrates some of MIRROR's wrapping classes as they appear in the mediator metamodel. Here, in addition to attributes from WebSQL, there are additional attributes pointing to domain concepts.

Assume that a WebDocument is to be instantiated in Telos for the "University of Toronto Home" page. In MIRROR, in addition to the above Telos declaration for WebDocument (a metaclass), there is a Java class with a constructor and attribute methods called WebDocumentSimpleClass. Java instances of this class represent Telos simple classes (instances) of WebDocument. When necessary, attributes of this metaclass are populated by a query in WebSQL (sent to WebSQLServer) that corresponds to each attribute. The returning string is either stored, or used to instantiate the appropriate Java class (and Telos metaclass).

For example, if the value of the attribute "links" is requested for the home page simple class, then the query to WebSQL returns a set of tuples that are used to instantiate simple classes for the Telos metaclass HyperLink. The URL and label values are passed to the constructor of the Java class HyperLinkSimpleClass. When the "targetDocument" attribute of one of these links is requested, the URL strin is passed to a constructor for WebDocumentSimpleClass and the process of growing the instance graph in memory incrementally continues. Duplicate instances of the same simple classes are not produced since the Java constructors check a registry to see if it has already been called. If it has, it returns the first (and only) instance of the desired Telos simple class.

3.3.2 Domain concepts

Domain concepts vary from site to site (even within a single university). Reconciliation of emphasis, paradigm differences, and other "impedance mismatches" can be done in a "politically expedient" or "best fit" manner by the domain expert (the designer or someone working with her).

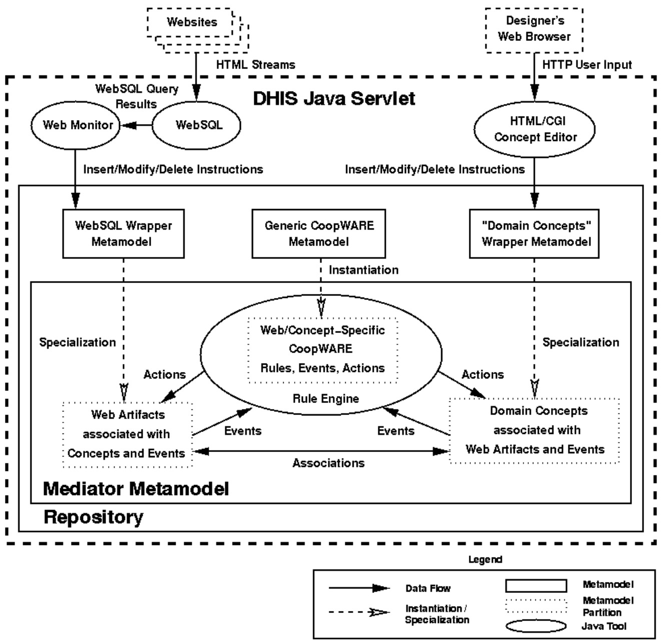

Figure 3. Web DHIS data integration using MIRROR and CoopWARE

The concept editor assists with this by allowing the designer to manipulate the domain graph. Moreover, having a conceptual modeling tool (a Telos repository) to support the HTML/forms concept editor assists the designer, in that semantic consistency checks, and related data operations can be done at a sophisticated level that the designer need not be aware of. This is another example of layering that separates the application being developed (in the application layer) from its supporting infrastructure (the DHIS architecture layer).

In conceptual modeling languages, classification is used to separate abstract concepts (say "University") from concrete instances ("University of Toronto"), as demonstrated in Figure 4. The abstract concepts and artifacts are Telos metaclasses. The concrete ones are Telos simple classes. "Instance of" arrows in the diagram indicate instantiation between these levels. Orthogonal to this, we can specialize abstract concepts, e.g. "Web Page" to "Home Page" (not shown in the diagram). These concepts can then be instantiated, e.g. "University of Toronto Home Page." In the DHIS implementation we associate concepts ("University") with Web pages ("Web Page"). This exists at both the abstract and concrete levels. Therefore, the attribute "pages" has a source "University" and a target "Web Page." At the concrete level, an instance of University ("University of Toront") has an instance of the metaclass WebPage ("U of T Home Page") as its attribute value for "pages." The concept editor allows the designer to manipulate the nodes in both the concrete and abstract graphs simultaneously. The end user only sees a simplified version of this graph, consisting of concrete nodes. The designer could then reuse the abstract graph in examining another University. Undoubtedly this university would have an only partially overlapping set of concepts, but a convergent model could be created which is instantiated by the two sets of university Web sites.

4 Conclusion

The paper has proposed mechanisms for the semi-automatic creation of domain models which capture the semantics of disparate Web sites and facilitate their integration into coherent information services. In addition, the paper describes a change management facility which updates the domain model to reflect changes in the underlying Web sites. The contribution of the paper lies in the provision of an architecture for semi-automatic generation and maintenance of user-oriented, semantic-based domain models that uses distributed heterogeneous information sources.

The implementation components are integrated in the MIRROR framework which fuses Web artifacts supplied by WebSQL with manually authored domain concepts, while CoopWARE provides a change propagation mechanism to maintain semantic consistency over changing information sources.

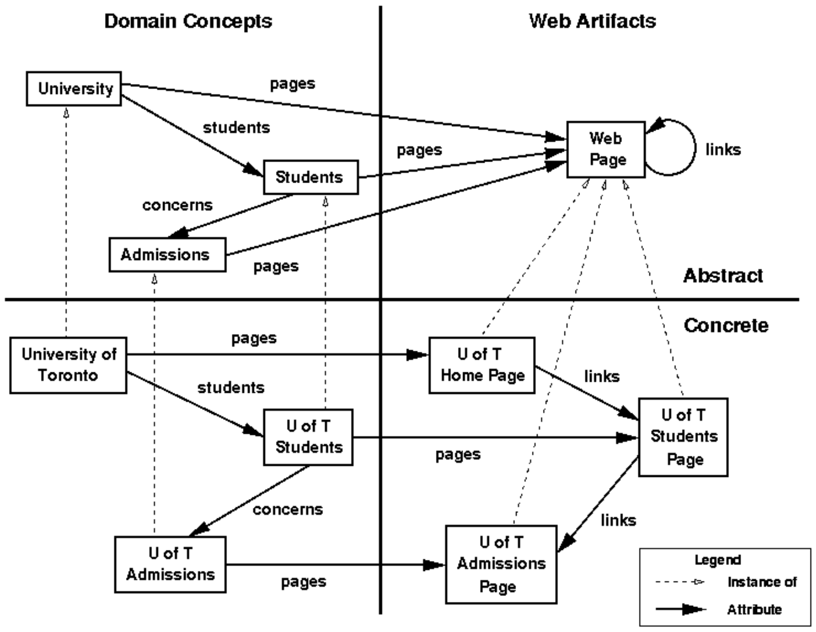

Figure 4. Domain concepts and Web artifacts

An obvious direction for further research is the application of the proposed framework to a real-world case study. Also, some theoretical aspects of the architecture, including concurrency control, should be considered. Finally, research for the support of unified access to several sets of Web sites which partially overlap semantically should be conducted, e.g. Web sites of several universities. This important research would address the problem of generating a unified schema for heterogeneous information services.

Acknowledgment

We would like to thank Michael Abloeser for his assistance in the prototype implementation. The material is based upon work supported (in part) by the Rutgers University Faculty of Management.

References

[1] S. Abiteboul, S. Cluet, and T. Milo. Querying and updating the file. In *Proceedings of the 19th International Conference on VLDB*, Dublin, Ireland, 1993.

[2] T. Catarci, S.-K. Chang, D. Nardi, G. Sanctucci, and M. Lenzrini. Wag: Web-At-a-Glance. In *Proceedings of the 31 Annual Hawaii International Conference on System Sciences, Volume VII*, pages 344–353, Jan. 1998.

[3] A. Gal and O. Etzion. A multi-agent update process in a database with temporal dependencies and schema versioning. to appear in the IEEE Transactions on Data and Knowledge Engineering (TKDE), 1998.

[4] A. Gal and J. Mylopoulos. The CoopWARE demo: wrapping up a legacy system. http://www.cs.toronto.edu/$\sim$coopware, 1997.

[5] Internet HotSuite. www.documagix.com/products/hotsuite.htm, 1998.

[6] S. Kerr. Data integration and change management using active metamodels. Master's thesis, University of Toronto, 1998.

[7] A. Mendelzon, G. Mihaila, and T. Milo. Querying the World Wide Web. In *Proceedings Conference on Parallel and Distributed Information Systems (PDIS)*, 1996.

[8] G. D. Michelis et al. Cooperative information systems: A manifesto. submitted for publication, 1996.

[9] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about information systems. *ACM Transactions on Information Systems*, Oct. 1990.

[10] I. S. Organization. The OSI reference model. http://www.iso.ch.

[11] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 132–141, New Orleans, Feb. 1996.

[12] J. Widom and S. Ceri, editors. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, San Francisco, CA, 1996.