Toward Web-Based Application Management Systems

Avigdor Gal, *Member*, *IEEE Computer Society*, and John Mylopoulos

Abstract—As Web technology spreads, the number, variety, and sophistication of Web-based information services is literally exploding. While some effort has been put into managing a single, centrally controlled Web site, current Web technologies offer little help for managing Web-based applications in-the-large. This is partly due to the distributed, heterogeneous, and open nature of such applications. This paper proposes a generic framework for managing Web-based applications which addresses both semantic and managerial issues. Semantic issues are addressed through the inclusion of a domain model component in the framework which describes the kinds of information that are available. Management issues are treated through a framework which includes formally-defined notions for an information model, information base consistency, transactions, and concurrency control. Thus, the proposed management system provides a semantically robust environment for Web-based information services while allowing for Web source independence.

Index Terms—Semantic interoperability, domain model, change propagation, concurrency control.

1 Introduction

The number, variety, and sophistication of Web-based information services is literally exploding, as Web technology is increasingly adopted [20]. This technology is founded on generic facilities for exchanging and accessing data available in distributed computing environments without any central control over the use of tools and terminology. Whereas some effort has been put into managing a single, centrally controlled Web site, current Web technologies have little to offer in terms of concepts, mechanisms, or facilities for managing Web-based applications in-the-large. In particular, those technologies in use offer little to support the integration of the data being interchanged, either for classification or for retrieval purposes. Nor have the technologies dealt with management issues by ensuring that Web-based applications function reliably and efficiently over long periods, much like database applications, despite the autonomous nature of the Web resources.

Consider, for example, the delivery of information about employees within an organization through the integration of several independent Web-based information sources, from the departments of human resources, accounts payable, and the like. The use of such information presupposes that there is an understanding of what a *salary* amount means in each information source, e.g., amount to be paid by the employer according to an employment contract? ... actual amount paid?... for the current fiscal year, or the previous one? ... Without such an understanding, use of the information can lead to misunderstandings, errors, and failures. Moreover, each of the departments may change its information sources, both at the level of data (e.g., the salary amount) and metadata (e.g., the interpretation of the salary attribute), at any time, without consulting or informing other information resource managers.

Not surprisingly, researchers and practitioners alike are coming to realize that any generic technology for managing Web-based information services has to tackle the problem of semantics head-on (see, for example, the idea of subject-based search engines in [21]). For our research, as with that of many others, semantic issues are addressed through the use of a domain model (or, global conceptual schema) which represents and organizes information about the application and describes the contents of each Web site relative to this domain model. For example, for the employees information services alluded to earlier, the domain model describes employees, projects, tasks, departments, compensations (including salaries), and other relevant information. Each information source is related to this domain model by the information it contains and its local representation. The idea of using a domain model to capture the semantics of information sources is well-known and well-accepted. The new challenge that arises for Web-based applications is to offer facilities so that this domain model is built bottom up through analysis of relevant information sources and to allow for its continuous evolution as new information sources become available and relevant to the information service being delivered.

In a nutshell, a *Web-based application management system* allows a user to access independent Web resources indirectly, through a semantic layer, whose role is to integrate several information resources about the same or similar domains. This domain model can be conveniently queried while avoiding the major pitfalls of Web resources, including unavailability (either temporary or permanent) as well as frequently and autonomously modified resources.

- *A. Gal is with the Department of MSIS, Rutgers University, 94 Rockafeller Road, Piscataway, NJ 08854-8054. E-mail: avigal@rci.rutgers.edu.*
- *J. Mylopoulos is with the Department of Computer Science, University of Toronto, 6 King's College Road, Toronto, ONT M5S 3H5, Canada. E-mail: jm@cs.toronto.edu.*

Manuscript received 20 July 1999; accepted 16 Aug. 1999.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 110129.

Such a system can be interfaced by either using a Web-based tool, for example, dynamically generated HTML (Hypertext Markup Language) files, or by an add-on application that simulates existing tools in the organization, such as a database-like user interface. To accommodate this mission, a Web-based application management system should provide semantic support for heterogeneous Web resources change propagation from Web resources to the semantic layer. Moreover, unlike a database management system, a Web-based application management system is restricted in its control over the managed Web resources and, therefore, should react to changes as soon as they become visible without having the "luxury" of applying retractions to the independent Web resources. In this sense, a Web-based application management system is considerably more complex than a DBMS, as it needs to maintain the consistency of a domain model without having any guarantees regarding the consistency of the Web resources themselves.

The Web-based application management system proposed in this work aims at providing semantic support for Web-based information services through the combined use of strong abstraction mechanisms and active database technology. The former provides the required semantic infrastructure for integrating heterogeneous, yet related, Web resources, while the latter allows for a semiautomatic maintenance of the domain model, once constructed. Database technology, in general, and active database technology, in particular, is founded on a well-defined set of concepts, including those of a data model, query processing, database integrity, transactions, and concurrency control which together offer highly optimized, robust, and reliable management services for database systems. Contrast this state of affairs with current Web technologies which so far have little to offer in terms of concepts, mechanisms, or facilities for managing Web-based applications. Of course, there are good reasons why such management is difficult, including the distributed, heterogeneous (in computing platform, data format, and data semantics), autonomous, and open nature of such applications. The proposed Web-based application management system aims at tackling these difficulties while providing a semantically robust environment for Web-based information services on the one hand and a Web source's independence on the other.

The main contribution of the paper is the proposed generic framework for Web-based applications which addresses both semantic and managerial issues. Semantic coherency is obtained through the inclusion of a domain model component in the framework which describes the kinds of information that are available. Management issues are treated through a formal framework that precisely defines the different components of Web-based applications, offers tools for performing basic operations to a domain model, and establishes the properties of the framework through theoretical analysis.

As a concrete case study, we consider the design of a semantic layer on top of intranet applications. Web resources made available through intranets have more structure in their design than public Web pages since they share common subject matters. Therefore, one can exploit this added structure in creating domain models. In this paper, we discuss an architecture for a specific domain, involving a university Web site. We examine the public Web information that a university provides and design an information base that offers a reorganization of that information on a semantic basis. The collection of Web sites at a university resembles a corporate intranet both in scale and on 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.

The remainder of the paper is structured as follows: The information model is described in Section 2, followed by an account of the consistency maintenance mechanism in Section 3. Concurrency control methods, which ensure the correctness of the domain model with respect to the underlying Web site, are presented and discussed in Section 4. Section 5 surveys the literature and compares it with the proposed management system. Finally, Section 6 summarizes the contributions of the paper and offers directions for future work.

2 The Information Model

This section presents an information model for defining the subject matter for a set of related Web artifacts. The definitions are included in an *information base* which contains the domain model alluded to earlier. All modifications to the Web artifacts are propagated (either automatically or semiautomatically) to the information base in a transparent fashion. As demonstrated in this paper, the information base is consistent with the underlying set of Web artifacts, while at the same time it can prevent problems common to all Web environments, such as broken links and inaccessible Web resources. These problems are solved at the semantic level, without enforcing any restrictions on the independent Web information resources. We start with an overview of various Web information resources in Section 2.1. Web artifacts and domain concepts are presented in Section 2.2, followed by a definition of the information base (Section 2.3). To illustrate the use of the information base, Section 2.4 provides two query examples.

2.1 Background: Web Information Resources

It is evident that the Web has become a leading tool in designing applications. According to a study prepared by ActivMedia Inc., a market research firm, spending on Web site technology and services will reach about 3.6 billion dollars in 1999 and will ascend to 24 billion by 2002 (ComputerWorld, July 1998). Also, the companies that will spend the most on Web development are those that put legacy data on line. This legacy data is one of the major resources of information services.

The main Web resource consists of HTML files. According to a survey prepared by Strategic Focus Inc. HTML is, and will probably continue to be for years to come, the language of choice for European Web-based applications (ComputerWorld, June 1998). Even in the United States, where Java-based Web applications (including Java RMI—Remote Method Invocation) are predicted to increase

138 percent until the turn of the century, their share of the total Web applications will only reach 34.8 percent. Observing the efforts put into HTML tools by the manufacturers of the two leading Web browsers (Netscape and Microsoft), as well as many other vendors, further support this claim. Also, the ever growing software crisis will result in far less Java-proficient employees than HTML/CGI-proficient employees. Therefore, a typical scenario of a Web application consists of a backbone of HTML files, assisted by other Web resources, including Java applets, embedded CGI scripts (with DOM—Document Object Model—[8] as a standard way to interact with documents, including naming elements and associating event handlers with elements), stream media, 3D graphics, and clickable maps.

XML (eXtensible Markup Language) [37] is a simplified subset of SGML that serves as an open-ended extension to HTML. Based on the recently finalized XML proposed specification, it seems that while HTML will continue to be the standard for presentation and document publishing, XML will become the preferred standard for displaying database-driven data on a Web page.[1] However, the status of XML depends largely on its broad acceptance by Web programmers. Its benefits to Web surfers is not apparent since "it won't show up on the Web pages, so what have you gained?" according to John Berry, a Webmaster at Broadcast Electronics, Inc., a radio transmitter and receiver manufacturer (ComputerWorld, July 20, 1998). Therefore, similarly to program documentation, XML should be enforced by organizational standards to have an impact on intranet development. Once enforced, a Web-based application management system can use the added metadata to afford the end-user a better semantic model. It is worth noting that the benefits of XML do not relieve the semantic heterogeneity problem. Rather, the additional capability of defining new metadata elements may add to it. For example, if one Web master uses the tag "price" while another uses "product price" or "cost" for the same information element, XML-based data exchanges grind to a halt. Nevertheless, it allows developers a mechanism for negotiating new ontologies by exchanging XML definitions as a standard (yet syntactic) mechanism. It is worth noting that the introduction of XML induced interest in the Web community in the development of standards for metadata representation, e.g., Resource Description Framework (RDF) [32] and Meta Content Framework (MCF) [23], which provide metadata tagging to documents.

For backward compatibility purposes, the main Web protocol (HTTP—Hypertext Transfer Protocol) was enhanced with existing Internet protocols, e.g., FTP. FTP (File Transfer Protocol) provides an archive of files, organized into a directory hierarchy. The organization of the files strongly corresponds to the FTP machine's disc directory and, therefore, provides little semantics that can be utilized in extracting a domain model.

1. This assessment is shared by many professionals, including Walter Crosby, a consultant on Web and legacy systems integration (ComputerWorld, March 9, 1998).

2.2 Web Artifacts and Domain Concepts

The information base for Web applications consists of two types of information, namely, *Web artifacts* and *domain concepts*. The Web artifacts are purely syntactic entities like those found in HTML or XML files, and HTTP or FTP requests. Web artifacts contain unstructured, hyperlinked information about the application. The domain concepts relate to the semantic content provided by the application, and serve as the semantic skeleton of the domain. The combination of domain concepts and Web artifacts allow a user to navigate through the information contained in Web sites while using a firm structured domain model. The user navigates from concept to concept with the ability to browse associated Web resources upon request.

Section 2.1 described the Web main resources, including HTML and XML files (including RDF and MCF schemata represented using XML tagging), Java applets, and stream media. HTML/XML files differ from Java applets and stream media. While Java applets and stream media feed a local application with preprocessed input and, therefore, are considered to be a black box on the client's end,[2] HTML/XML files are semistructured data files, whose content can be analyzed and utilized by the client. In what follows, we shall refer to Web artifacts whose content can be analyzed as *documents*. Graphics provide an alternative approach towards data representation and is easily embedded within this framework. However, its characteristics (e.g., captured objects and colors) are more difficult to extract than HTML characteristics and, therefore, we restrict the treatment of graphics files in this work. The embodiment of tools to analyze graphic files (e.g., [29]) would enable some further refinement of these Web artifacts. In particular, clickable maps that are a combination of graphics and CGI scripts can be analyzed further by using the form list fields. XML files enhance the capabilities of Web-based application management systems since the predefined tags serve as an extension to the application ontology and become useful in grouping parts of XML files within a single semantic concept.

Hyperlinks containing FTP requests are treated similarly to those containing HTTP requests. However, due to the difficulty in extracting meaning by analyzing FTP sites, such analysis is reduced to a minimum.

Domain concepts are initially generated by examining the hyperlink graph topology and content of each Web site. Each Web artifact in a site initially generates a corresponding domain concept. The HTML/XML title of a Web document (or the file name for other Web artifacts) is utilized in generating the name of its corresponding concept.

A domain concept is associated with other concepts by using domain attributes. The hyperlinks in a document generate the attributes of the concept, which are labeled links to other concepts. For example, the concept "University of Toronto" is associated with the concept "UofT Admissions" through the former concept's attribute

2. Sophisticated software engineering tools [4] may be applied to uncover Java applets capabilities and present them to the user. However, this requires that the Java code itself is available to the client, which is rarely the case.

$admissions$. A domain concept also has one or more associated Web artifacts which are bound to the concept object by an attribute $WebArtifacts$.

Domain concepts are defined and refined into a domain model using the conceptual modeling language Telos [27]. The model supports multivalued attributes, multiple classification (i.e., an object can be an instance of multiple classes), and multiple specialization (i.e., a class can be a specialization of several other classes). Moreover, attributes are objects as well, classified under attribute classes, and there is strong support for multiple levels of metaclasses. The model is available in several implementations, including ConceptBase [16].

The presence of metadata tagging as in XML and RDF becomes handy in describing domain concepts. However, different Web masters may manage the information in their respective Web sites quite differently. Even within the domain of a single organization (such as public information about a university, or private/semiprivate information in a corporate intranet), the different Web sites may be developed and maintained independently. Thus, integrating multiple Web sites into a single application with common domain model is a challenge. This problem will most likely become even more acute as more and more XML files replace HTML files. With XML files, the concepts that are available through the additional tagging should be scrutinized to ensure semantic compatibility across Web sites. Therefore, the designer of the application is responsible for integrating the information on the Web into a common domain model. To alleviate this problem, we have developed a methodology and a prototype system called MIRROR that facilitates the creation of the domain model and its subsequent population. MIRROR (Metamodel Integration for Reflective, Reactive, Observational Repositories) [18] is an extension of a Java-based implementation of Telos. This repository (called "jTelos") allows Telos objects (metaclasses, simple classes, and tokens) to be created and stored, modified, and destroyed by using Java routines. MIRROR employs the concept of an active metamodel to provide a systematic approach for organizing data access methods and to coordinate their control. A metamodel is active in that, for each attribute of an object, there is an executable routine to compute the value of that attribute. Therefore, MIRROR extends the jTelos implementation by adding executable extensions to Telos objects. These extensions consist of a Java method for each attribute. For example, the user can declare a Java class `WebDocument` and corresponding Telos simple classes are then automatically generated, checked for consistency with Telos semantics, and added to the repository. For example, if the University home page includes a new section (say Faculty and Staff) a new class will be generated automatically. Similarly, the designer can include a Java method in $WebDocument$ called $hyperlinks()$ corresponding to the Telos attribute of the same name. This method can return an enumeration of Web artifact instances. Thus, it can populate its attribute and is called a "populator" method.

2.3 The Information Base

Let $V_W = \{v_w^1, v_w^2, ..., v_w^n\}$ be a set of Web artifacts and $V_C = \{v_c^1, v_c^2, ..., v_c^m\}$ be a set of concepts. In what follows, we use the following notations. For $\{v_i, v_j\} \subseteq V_W$, $v_i \rightsquigarrow v_j$ indicates that a hyperlink to v_j exists in v_i. For $v_i \in V_C$ and $v_j \in V_W \cup V_C$, $v_i \mapsto_x v_j$ suggests that v_j is semantically associated with v_i. The semantic relationship can be either by attribution (where x is the attribute name), classification (where $x = IN$) or specialization (where $x = ISA$). In the case where $v_j \in V_W$, v_j is related through attribution to v_i, such that $x = WebArtifacts$. It is worth noting that we use a single Web artifact as an atomic unit from the viewpoint of change management, as changes in the Web environment are identified through file modifications. This does not contradict the ability of the model to support access to a named fragment of HTML. Therefore, $v_i \rightsquigarrow v_j$ may very well identify a hyperlink between two named fragments of the same file (i.e., $v_i = v_j$).

Formally speaking, an information base $\mathcal{IB} = (V, E)$ is a directed graph with a set of vertices V and a set of edges E over V such that:

- $V = V_W \cup V_C$.
- $E = \{\langle v_i, name_{ij}, v_j\rangle | \{v_i, v_j\} \subseteq V\}$
  $\forall \{v_i, v_j\} \subseteq V_W, \langle v_i, \emptyset, v_j\rangle \in E \Longleftrightarrow v_i \rightsquigarrow v_j$
  $\forall \{v_i, v_j\} \subseteq V_C, \langle v_i, x, v_j\rangle \in E \Longleftrightarrow v_i \mapsto_x v_j$
  $\forall v_i \in V_C, v_j \in V_W, \langle v_i, \emptyset, v_j\rangle \in$
  $E \Longleftrightarrow v_i \mapsto_{WebArtifacts} v_j$.

It is worth noting that relationships among domain concepts are named (annotated as $name_{ij}$ and can be an attribute name, *IN*, or *ISA*), while relationships that involve Web artifacts are not (represented as $\emptyset$). This difference stems from the separation of semantic concepts from purely syntactic relationships, which are captured by hyperlinks.

Let $\mathcal{T} \cong \mathrm{N}$ be a discrete temporal domain, isomorphic to the natural numbers. $t \in \mathcal{T}$ is a nondecomposable unit of time, whose granularity is application-dependent. $\mathcal{TIB} : \mathcal{T} \rightarrow P(\mathcal{IB})$ is a time varying function, where P is the power set of $\mathcal{IB}$. We can extend the relationships $\rightsquigarrow$ and $\mapsto_x$ to be time varying, where $\rightsquigarrow^t$ stands for "a hyperlink to v_j exists in v_i at time t" and $\mapsto_x^t$ stands for "v_j is semantically related to v_i at time t through x." Therefore,

$$\forall \{v_i, v_j\} \subseteq V_W(t), \langle v_i, \emptyset, v_j\rangle \in E(t) \Longleftrightarrow v_i \rightsquigarrow^t v_j,$$
$$\forall \{v_i, v_j\} \subseteq V_C(t), \langle v_i, x, v_j\rangle \in E(t) \Longleftrightarrow v_i \mapsto_x^t v_j, \text{ and}$$
$$\forall v_i \in V_C(t), v_j \in V_W(t), \langle v_i, \emptyset, v_j\rangle$$
$$\in E(t) \Longleftrightarrow v_i \mapsto_{WebArtifacts}^t v_j.$$

For simplicity's sake, the time varying notation is dropped whenever possible. Should the unique identification of time becomes necessary, the distinction will be made explicit.

Before studying the properties of $\mathcal{IB}$, we first provide a short discussion on Web information accessibility. Accessibility is a major concern for managing Web information. A Web artifact can be temporarily unavailable due to a server daemon problem, or can be permanently unavailable due to the deletion of the file. Therefore, a set of Web artifacts, $V_W(t)$, at a given time $t \in \mathcal{T}$ can be further refined into $V_A(t) \cup V_I(t)$, such that $V_A(t)$ is the set of all accessible Web artifacts at time t and $V_I(t)$ is the set of all inaccessible Web artifacts at time t. $V_I(t)$, in turn, can be further refined into two categories, based on the root of the artifact's inaccessibility problem. Therefore, we can specify the following two categories $V_I(t) = V_P(t) \cup V_T(t)$. A Web artifact $v_w \in V_T(t)$ is temporarily inaccessible due to a network or a server daemon problem, identified by the message, "There was no response. The server could be down or is not responding." It can also be temporarily inaccessible due to a DNS problem, identified by the message, "The server does not have a DNS entry" (assuming the problem relates to a client's mapping problem), or due to other problems such as server overload or a stall condition. A Web artifact $v_w \in V_P(t)$ is permanently inaccessible due to accessibility problems that relate to the artifact itself. For example, a Web artifact can be inaccessible due to a naming problem ("File not found" or "404 not found") or due to security violation ("Forbidden"). We consider the former category ($V_T(t)$) as representing transient problems either on the server side or on the client side. The latter category ($V_P(t)$) is considered permanent, in the sense that it represents an ongoing accessibility problem with respect to the Web artifact.

Based on these definitions, we will say that an information base $\mathcal{IB}$ is *weakly accessible* at time t if for all $v \in V_W(t)$, $v \in V_A(t) \cup V_T(t)$. $\mathcal{IB}$ is said to be *strongly accessible* at time t if for all $v \in V_W(t)$, $v \in V_A(t)$. It is readily seen that an information base that is strongly accessible at time t is also weakly accessible at t. Therefore, as long as an information base is weakly accessible, we can assume that in the long run, any of the Web artifacts in V_W will become accessible and, therefore, the existence of all of the Web artifacts as part of the information base is justified.

Definition 1 (Information base accuracy). *An information base $\mathcal{IB}$ is accurate at time t iff*

1 *$\mathcal{IB}$ is weakly accessible at time t and*
2 $\forall\{v_i, v_j\} \subseteq V_C(t), \langle v_i, name_{ij}, v_j\rangle \in E(t) \Longrightarrow$
$\exists\{v'_i, v'_j\} \subseteq V_W(t) | v_i \mapsto^t_{WebArtifacts} v'_i$
$\wedge\, v_j \mapsto^t_{WebArtifacts} v'_j \wedge v'_i \rightsquigarrow^t v'_j$.

Based on Definition 1, an information base is considered to be accurate iff it is weakly accessible and a relationship between any two semantic concepts can be matched by a hyperlink between the concepts' related Web artifacts.

2.4 Querying the Information Model

We now provide two examples of querying the information model, which demonstrate the typical usage of the architecture. The first (Section 2.4.1) is a query posed by a designer, while the second (Section 2.4.2) is an example of a query posed by an end-user.

2.4.1 Designer Queries

Suppose that a designer is designing a domain model of the University of Toronto Web site, for internal use by the staff. Initially, she requests a display of all of the concepts that can be generated from the University of Toronto home page and its hyperlinks. 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 links the URL with the tentative domain concept. Also, for each labeled link in the page, the system adds an attribute to the domain concept, where the new attribute's label is initially the same as the label on the hyperlink. The system then generates a concept for the target Web artifact of the hyperlink (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
- University of Toronto Home $\rightarrow_{Welcome}$ University of Toronto: Welcome
- University of Toronto Home $\rightarrow_{AboutUofT}$ University of Toronto: About UofT
- University of Toronto Home $\rightarrow_{Directories}$ University of Toronto: Contacts and Directories
- University of Toronto Home $\rightarrow_{News,Events,Weather}$ University of Toronto: News, events, weather
- ...
- University of Toronto Home $\rightarrow_{AcademicPrograms\&Divisions}$ 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.[3] Assume that 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, and edits attribute names generated from hyperlink labels, as desired.

Each new concept is placed within a schema requiring "abstract concepts" (classes) for each of the "concrete concepts" (instances) generated and refined as discussed above. In Telos, the abstract concepts are *metaclasses* and the concrete concepts are *simple classes*. Therefore, the simple class "University of Toronto" instantiates the "University" metaclass, "UofT Admission" instantiates the "Admission" metaclass, etc. Telos terminology is concealed from the designer, adding "abstract concepts" as deemed necessary. The class level of the metamodel becomes particularly handy when associated concepts from different university Web sites are integrated into a unified domain model, using various fusion techniques.

2.4.2 End-User Queries

For the second example, suppose the designer, alluded to above, first prepares a domain model which captures the semantics of a corresponding set of related Web artifacts.

3. In order to reduce the designer intervention in the initial design process, additional techniques may be applied. For example, a systematic polling of the state of Web documents over time divulges transient links and removes them from the domain model without the need for a human intervention.

Following this step, an end-user (in our example, a university staff member) can browse either the conceptual graph or the Web artifacts associated with any concept, returning to the conceptual graph at any time.

The end-user is initially presented with the domain model'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 an end-user's query. The same target object is retrieved from the repository as for the designer, but it only displays information that is relevant to an end-user.

The response to the query provides links that were deemed relevant by the designer. These links include hyperlinks from the concept to the associated Web artifact(s) it was derived from. Also, a link for each attribute of "University of Toronto" is provided. These links correspond to a reorganization of the links originally presented to the designer for the University of Toronto home page.

3 Consistency 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 Web resources. A traditional design assumes the infrequent changes of metadata with respect to data modifications and often leads to a rigid selection of data structures and programs that are heavily dependent on a specific metadata structure [11]. However, in order to provide complete maintenance of Web-based information services, the following requirements need to be fulfilled. First, the domain model, consisting of the conceptual schema component of the information base, should be amenable to evolution without the need to modify the internals of browsing and querying tools. Second, any changes performed to the information base or any of the information sources should be propagated to other components of the overall system to ensure global consistency. For example, if a Web artifact is removed, so should its information base counterpart (and any reference to it).

The former requirement is fulfilled through the adoption of the Telos object model, as discussed in Section 2. Use of this model allows for changes to the conceptual schema of the information base without any need for recompilation. This section focuses on an architecture and supporting mechanisms for the latter fundamental requirement, namely, change propagation. The adopted architecture is that of CoopWARE (Cooperation With Active Relationship Enforcement) [13], a generic integration architecture based on active database technology. Section 3.1 describes the CoopWARE architecture and how it can be used to integrate information from disparate information sources, as well as to manage changes. Section 3.2 focuses on change management for Web-based applications and how it can be addressed within the CoopWARE framework.

3.1 CoopWARE

CoopWARE is a generic integration architecture which provides a flexible and easy to use mechanism for supporting cooperative information systems through asynchronous communication and has been successfully tested in a reverse engineering environment [28]. Fig. 1 demonstrates the general structure of the information services architecture of CoopWARE. It consists of several conceptual *components* (a conceptual component can extend beyond the boundaries of a single machine) that communicate through a coordinator. Each component contains zero or more structured, unstructured, or semistructured data sources, and has an interface which defines a set of *services* that can be executed by the component associated with the interface and a set of *events*: compact reliable occurrences that enable the flow of information regarding the state of the domain model. A coordinator contains an information schema, a rule mechanism (referred to as "Rule Set" and "Rule Engine" in Fig. 1), and an interface.

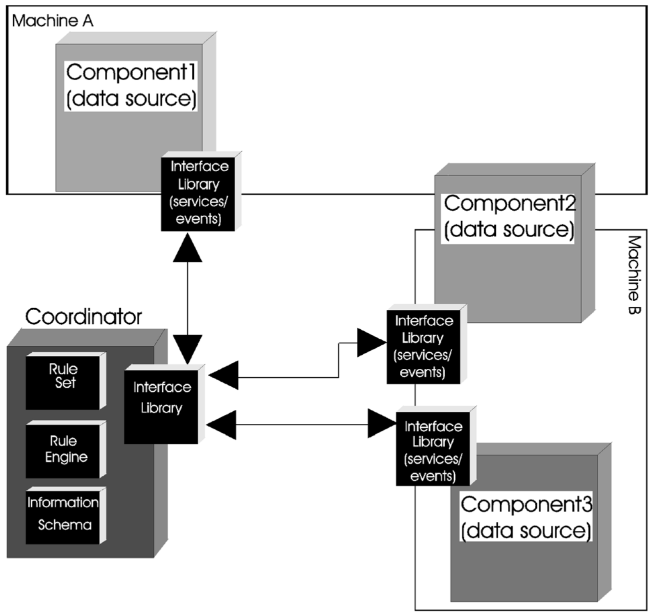


Fig. 1. The CoopWARE architecture.

An application in CoopWARE is built in four layers, as follows:

- **MetaClass:** The metaclass layer describes the basic primitives of CoopWARE. This layer provides the semantic relationships between components (defined as a $ComponentClass$), services (defined as a $ServiceClass$), events (defined as an $EventClass$) and rules (defined as a $RuleClass$). Also, it consists of some additional useful features, e.g., the workflow of subservices within a service and the classification of events.
- **Designer toolkit:** The metaclasses are instantiated into a designer toolkit, which provides a set of classes to be inherited by the application domain definition. For example, services can be classified along two orthogonal dimensions: They can either be subservices or services, where in the latter case they are not utilized in the construction of other services. Also, services can either provide a termination status or not. If a service is of the latter type, the coordinator cannot rely on receiving a response from the service and, therefore, it cannot serve as a

trigger for a rule. The designer toolkit provides all four combinations (services with termination status, services without termination status, subservices with termination status, and subservices without a termination status) by using multiple classification, as provided by Telos.

- **Application domain:** At this layer, the designer provides the application domain, i.e., specific components, services, events, and rules by specializing classes that are available at the designer toolkit.
- **Application state:** This layer supports the application state, e.g., specific components that currently run, execution of services, notification of events, etc., through tokens (instances), by instantiating domain classes from the application domain layer.

There are two types of communication in this framework. The access to the Web is handled through Web protocols (e.g., HTTP) from within a Web querying tool (in this case WebSQL) or a browser. The local communication between various components of a CoopWARE application and the communication between users' client processes and the system are handled more generally. Non-Java components communicate through sockets. In CoopWARE, the communication between the coordinator and a component is handled using a request/response protocol, provided by a TMB (Telos Message Bus), an extendible message server implemented in C++, on top of *mbus*, a public domain software that uses UNIX sockets. The Java components support the full range of network computing communication options (TCP/IP, CORBA, sockets, HTTP, RMI) since there are Java libraries or IDL support for all of these. A Java-based implementation of CoopWARE is being developed (using the jTelos API) which runs in a Java Virtual Machine. This process can be extended to support the range of communications options listed above. The HTTP support can also appear in a Java servlet version of CoopWARE which can be run inside any commercial Web server supporting servlets.

The following section discusses, in detail, the use of the CoopWARE architecture in handling change management of Web applications.

3.2 Change Management of Web Application

Table 1 provides a partial definition of the active maintenance component, written in the TELOS language, using the designer's toolkit of CoopWARE. $WebMonitor$ is a component of the architecture that monitors a set of URLs given to it through the services $AddToScope$ and $RemoveFromScope$. $URLExpression$ can be a single URL or a compact definition of a set of URLs (e.g., http://www.toronto.edu/*). The $WebMonitor$ detects and informs the coordinator of an insertion, modification, removal, or relocation of artifacts within its scope. The detection process can be done by using a periodic polling or by having a proxy that detects these changes automatically, e.g., BackWeb [3]. Generally speaking, the former is more costly than the latter in terms of network traffic, yet when the scope of the search is localized (as is the case with an intranet), the benefits of the latter are less obvious. Also, push technologies, as the one used in BackWeb, necessitate a proxy at the server's end to avoid network congestion. This is not always feasible, as the Web-based application management system has no control over the Web resources. In the case of a periodic polling, the polling rate determines the time granularity of the information base function. For example, if a polling is performed every 10 seconds, the nondecomposable time unit of the information base is 10 seconds. Therefore, if a Web artifact v_w is accessible at a polling done at 10:00:00 am then it is considered to be accessible throughout 10:00:10 am. In addition, the $WebMonitor$ verifies whether a Web artifact is part of the scope, extracts a state of a document given to it as a parameter, and identifies whether a Web artifact is accessible or not, and the cause of inaccessibility.

The designer herself is treated as another component of the architecture, which offers the services of both suggesting a domain concept and verifying an existing domain concept. These services result in the signalling of the $DomainConceptIdentified$ and $DomainConceptVerified$ events, respectively. The communication with the designer is asynchronous and can be handled either via email, or through the use of a worklist which prompts the designer whenever such a request is pending.

The $ConceptEditor$ provides the capability of modifying the domain model as stored at the information base. As an example, Table 1 details the information regarding the $IdentifyRelationships$ service which is provided by the $ConceptEditor$. $IdentifyRelationships$ signals the $ArtifactInsertion$ event whenever it comes across a linked artifact which is not reported at the information base.

Table 2 formalizes the services of the $ConceptEditor$, given in Table 1, by identifying their side-effects with respect to the information base. One uses v_w and v_c to identify a single Web artifact and a single domain concept, respectively. $Associate$ adds an edge to $\mathcal{IB}$ to represent $v_c \mapsto_{WebArtifacts} v_w$. $Disassociate$ removes the edge that represents $v_c \mapsto_{WebArtifacts} v_w$. $IdentifyRelationships$ adds an edge for $v_c \mapsto_{name} v_c'$. $RemoveRelationships$ disconnects any two concepts that are related solely through the hyperlink of the deleted artifact. $UpdateRelationships$ deletes relationships that are no longer valid in the current state of the information base.

The detection of updates is accomplished through events. In this context, we support four major types of events, namely, *artifact insertion* (given as an example in Table 1), *document modification*, *artifact removal*, and *artifact relocation*. These events are basically the result of new information that arrives at the $WebMonitor$ from the Web artifacts within its scope. For example, the $WebMonitor$ signals an artifact removal of an artifact v_w as soon as v_w becomes permanently inaccessible (i.e., $\exists t \in \mathcal{T} | \forall t' \geq t, v_w \in V_P(t')$). In addition, artifact insertion and artifact removal events can be the result of extending or curtailing the $WebMonitor$'s scope. An event transfers parameters to be used by services. The first three events pass a single parameter, the artifact's URL. The fourth event passes two parameters which correspond to the old and new URLs. It is worth noting that the modification of an artifact which is not a document (i.e., an HTML file or an XML file) is not informed as an event, as no further processing can be performed to it.

TABLE 1
A Partial Definition of the Active Model Maintenance Component

TELL CLASS WebMonitor
ISA Component
WITH
offers
: AddToScope;
: RemoveFromScope;
: InScope;
: ExtractDocumentState;
: Accessible
generates
: ArtifactInsertion;
: DocumentModification;
: ArtifactRemoval;
: ArtifactRelocation
END WebMonitor
	TELL CLASS Designer
	ISA Component
	WITH
	offers
	: SuggestDomainConcept;
	: VerifyDomainConcept
	generates
	: DomainConceptIdentified;
	: DomainConceptVerified
	END Designer
		TELL CLASS ConceptEditor
		ISA Component
		WITH
		offers
		: AddArtifact;
		: RemoveArtifact;
		: NewConcept;
		: AddConcept;
		: RemoveConcept;
		: FindConcept;
		: Associate;
		: Disassociate;
		: IdentifyRelationships;
		: RemoveRelationships;
		: UpdateRelationships;
		: RelocateRelationships;
		: RecordDocumentState;
		: RetrieveDocumentState;
		: RetrieveArtifactTitle;
		: RemoveDocumentState;
		: Ingoing
		generates
		: ArtifactInsertion
		END ConceptEditor

TELL CLASS AddToScope ISA TerminationStatusAvailableService WITH parameters : URLExpression generates : ArtifactInsertion END AddToScope	TELL CLASS SuggestDomainConcept ISA TerminationStatusAvailableService WITH parameters : URL; : URL.Title generates : DomainConceptIdentified END SuggestDomainConcept	
TELL CLASS ArtifactInsertion ISA InternalEvent WITH parameters : URL triggerRules : r1 END ArtifactInsertion	TELL CLASS DomainConceptIdentified ISA InternalEvent WITH parameters : URL; : Concept triggerRules : r2; : r3; : r4; : r5 END DomainConceptIdentified	TELL CLASS IdentifyRelationships ISA TerminationStatusAvailableService WITH parameters : URL generates : artifactInsertion END IdentifyRelationships

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) [36]. The semantics of a rule are as follows: When an event ev occurs, conditions $co_1, co_2, ..., co_n$ are independently evaluated, wherein 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. CoopWARE has a well-defined rule language (see [14] for the language's BNF), where an event consists of components' events, a condition is defined as a Boolean function of the system's state, and an action is a sequence of services. At runtime, a rule is instantiated as a result of any of its events' instantiation.[4] The rule is executed if its condition is evaluated to true, which results in service instantiations. SR and EV annotate the sets of service instances and events instances, respectively, such that an instance is a pair $\langle id$, $type\rangle$, where id is a unique system-wide identifier and $type$ associates an instance with the appropriate service class or event class. CN is a set of condition instances of rules, such that an instance is a triplet $\langle id$, $rule$, $outcome\rangle$, where id is a unique system-wide identifier, $rule$ associates the condition with the appropriate rule class, and $outcome$ is the result of the condition evaluation (either $true$ or $false$). Whenever relating to a set of services, events, or conditions, we use $id(SR)$, $id(EV)$, and $id(CN)$, respectively, to refer to their unique identifiers.

Whenever an artifact insertion event occurs, facts about the artifact's state are recorded in the information base. In case the artifact is a document, these facts include the

4. The CoopWARE rule language supports either simple events or disjunctive events.

TABLE 2
The Impact of Services on the Knowledge Model

Service	Side-effect on $\mathcal{IB}$
AddArtifact(v_w)	$V_W := V_W \cup \{v_w\}$
RemoveArtifact(v_w)	$V_W := V_W - \{v_w\}$
NewConcept(v_c)	none
AddConcept(v_c)	$V_C := V_C \cup \{v_c\}$
RemoveConcept(v_c)	$V_C := V_C - \{v_c\}$
FindConcept(v_w)	none
Associate(v_w, v_c)	$E := E \cup \{\langle v_c, \emptyset, v_w\rangle\}$
Disassociate(v_w, v_c)	$E := E - \{\langle v_c, \emptyset, v_w\rangle\}$
IdentifyRelationships(v_w, v_c)	$E := E \cup \{\langle v_c, \text{name}, v_c'\rangle \mid \exists v_w' \in V_W \wedge v_w \rightsquigarrow v_w' \wedge v_C' \mapsto_{\text{WebAtrifacts}} v_w' \wedge v_c \mapsto_{\text{name}} v_C'\}$ $\cup \{\langle v_c', \text{name}, v_c\rangle \mid \exists v_w' \in V_W \wedge v_w' \rightsquigarrow v_w \wedge v_C' \mapsto_{\text{WebAtrifacts}} v_w \wedge v_c \mapsto_{\text{name}} v_C'\}$
RemoveRelationships(v_w, v_c)	$E := E - \{\langle v_c, \text{name}, v_c'\rangle \mid \exists v_w' \in V_W \wedge v_w \rightsquigarrow v_w' \wedge$ $(\forall v_w'' \in V_W, (v_C' \mapsto_{\text{WebAtrifacts}} v_w') \rightarrow (v_w' = v_w''))\}$
UpdateRelationships(v_w, v_w', v_c)	$E := E \cup \{\langle v_c, \text{name}, v_c'\rangle \mid \exists v_w'' \in V_W \wedge v_w' \rightsquigarrow v_w'' \wedge v_C' \mapsto_{\text{WebAtrifacts}} v_w'' \wedge v_c \mapsto_{\text{name}} v_c'\}$ $E := E - \{\langle v_c, \text{name}, v_c'\rangle \mid \forall v_w'' \in V_W, (v_C' \mapsto_{\text{WebAtrifacts}} v_w'') \rightarrow (v_w \not\rightsquigarrow v_w'')\}$
RelocateRelationships(v_w, v_w')	$E := E \cup \{\langle v, \emptyset, v'\rangle \mid (v = v_w' \wedge \langle v_w, \emptyset, v'\rangle \in E) \vee (v' = v_w \wedge \langle v, \emptyset, v_w\rangle \in E)\}$ $E := E - \{\langle v, \emptyset, v'\rangle \mid v = v_w \vee v' = v_w\}$
RecordDocumentState(v_w)	$E := E - (\{\langle v_w, \emptyset, v_w'\rangle \mid v_w' \in V_W\} \cup \{\langle v_w', \emptyset, v_w\rangle \mid v_w' \in V_W\}) \cup$ $\{\langle v_w, \emptyset, v_w'\rangle \mid v_w' \in V_W \wedge (v_w \rightsquigarrow v_w')\} \cup \{\langle v_w', \emptyset, v_w\rangle \mid v_w' \in V_W \wedge (v_w' \rightsquigarrow v_w)\}$
RetrieveDocumentState(v_w)	none
RetrieveArtifactTitle(v_w)	none
RemoveDocumentState(v_w)	$E := E - (\{\langle v_w, \emptyset, v_w'\rangle \mid v_w' \in V_W\} \cup \{\langle v_w', \emptyset, v_w\rangle \mid v_w' \in V_W\})$
Ingoing(v_c)	none

document's HTML title and the links contained in the file (both their labels and target URLs). The designer is notified and a provisional domain concept corresponding to the HTML title of the document (or the artifact's name) is suggested to the designer by the system. This concept can be edited by the designer. For example, the initial concept assigned with "http://www.toronto.edu/index.html" 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 artifact contents in response to a user query.

New semantic relationships are added to the domain model based on the hyperlinks in a new document. Therefore, the concept associated with the new Web document is related with any concept that represents a target Web artifact of a hyperlink. It is possible that one or more of the target artifacts are new artifacts as well, in which case an artifact insertion event is signalled for each of the new artifacts. All new artifacts are queued for initial examination by the designer. It is noteworthy that in lieu of adding a new concept, the designer may choose to supplement an already existing concept with information from the new Web artifact by performing a fusion operation [2]. The new extracted information is added to the existing concept and modifications in the multiple artifacts associated with the concept are monitored with changes propagating to the unifying concept. It is also worth noting that hyperlinks link a single document to a single target Web artifact. Thus, by default, concepts are related through 1-1 relationships. 1-n relationships are suggested by the system when a document is parsed and contains an HTML list of hyperlinks. 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 document modification event occurs, a new set of Web artifacts associated with the document is generated and compared to the existing one. Whenever possible, changes in the Web documents are propagated to concepts automatically. While some changes require manual intervention, 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 document's title changes or there is a substantial change in the document's content, the designer is prompted to examine the previous and current state of the document to decide on the appropriate manual intervention.

TABLE 3
Rule R_1: Insertion Identification

Event:	ArtifactInsertion(URL)
Condition:	InScope(URL)
Action:	AddArtifact(URL)
	State:=ExtractArtifactState(URL)
	RecordArtifactState(State)
	SuggestDomainConcept(URL, URL.title)
	AddConcept(URL.title)
	Associate(URL, URL.Title)

A Web artifact removal event results in removing this artifact from its associated domain concept in the information base. Therefore, there may be domain concepts for which there is no longer an associated artifact. Any recorded hyperlink in other documents that point to the removed artifact are considered to be broken links and, therefore, they are not reflected in the semantic domain model.

Finally, an artifact relocation results in updating the information base to include the new URL for each instance where the obsolescent URL was mentioned.

Appendix A provides a design of the change management component using CoopWARE. We next provide an example of three rules that support the artifact insertion scenario. The following rule sends a request to identify a domain concept. The request includes a default concept generated from the Web artifact (see Table 3).

The execution of the rule is conditioned upon the Web artifact being part of the scope. This mechanism prevents an exponential processing of Web artifacts, as a result of adding artifacts to the information base due to hyperlinks. The action part of the rule consists of a sequence of services, whose impact on the information base is given in Table 2.

The designer may choose to adopt the suggested concept or to provide an alternative concept. When such a concept is identified (see the two rules above), a new concept is added if needed, and the relationship information is added to the information base. The *IdentifyRelationships* service seeks the concept in the information base and is possibly suspended until the concept is added (see Table 4).

Fig. 2 provides a visual representation of the three rules. The graph is automatically generated from the information base and the rule specification is by using a modified version of GraphDraw [9]. This tool is used for the analysis of the execution flow of set of rules [12]. Events are depicted as triangles, conditions as diamonds, and services are depicted as circles. A single circled node represents an event or a service of the $WebMonitor$, a double circled node represents an event or a service of the $Designer$, and a triple circled node represents a condition or a service of the $ConceptEditor$. It is worth noting that the graph contains a cycle, in which the first rule and the third rule are being executed alternately. A careful analysis shows that the number of iterations is limited to the size of the $WebMonitor$ scope and, therefore, the set of rules guarantees to terminate.

TABLE 4
Rule R_2 and Rule R_3: Concept Identification

Event:	DomainConceptIdentified(URL, Concept)
Condition:	NewConcept(Concept)
Action:	AddConcept(Concept)
Event:	DomainConceptIdentified(URL, Concept)
Action:	IdentifyRelationships(URL, Concept)

As a final note, it is worth noting that the discussion in this section highlighted the capabilities of CoopWARE in maintaining the semantic consistency of the information base. CoopWARE's rules may also serve in maintaining syntactic consistency. For example, the concept model can reflect the weak accessibility of the information base, using the following mechanism. Whenever the $WebMonitor$ detects a temporarily inaccessible Web artifact (i.e., for some $t \in \mathcal{T}$, $v_w \in V_T(t)$), it can signal an event $Inaccessible$ with the Web artifact as a parameter. The CoopWARE coordinator, upon receiving the information, would request the $ConceptEditor$ to prevent a user from attempting to access the Web artifact as long as it is inaccessible. This can be materialized by marking the Web artifact as inaccessible in the information base. Therefore, while the conceptual model remains intact, some of the browsing capabilities are restricted. An $Accessible$ event, signaled by the $WebMonitor$, would result in reversing the constraint imposed on the related Web artifact.

4 The Transaction Model

A transaction is a logical unit of work [6]. A transaction model in a database provides a framework for concurrent processing of retrieval and update operations. A conventional database transaction model ensures the following properties (ACID):

- **Atomicity**. Either all the operations of a transaction are properly reflected in the database or none are.
- **Consistency**. Execution of a transaction in isolation preserves the consistency of the database.
- **Isolation**. Each transaction assumes that it is executed alone. Thus, no intermediate transaction results are available to other concurrently executed transactions.
- **Durability**. The values changed by the transaction persist after the transaction is successfully completed.

In this section, we devise a transaction model for the support of the proposed architecture. We start by examining six transaction types that are available in the architecture (Section 4.1). We discuss the applicability of ACID transactions to the architecture in Section 4.2. Finally, Section 4.3 provides the concurrency control mechanism which supports a correct execution of concurrent transactions in this framework.

It is not the intention of this section to introduce yet another transaction model but rather to show how to adopt existing concepts to our architecture. The main characteristic which prohibits the use of a conventional transaction

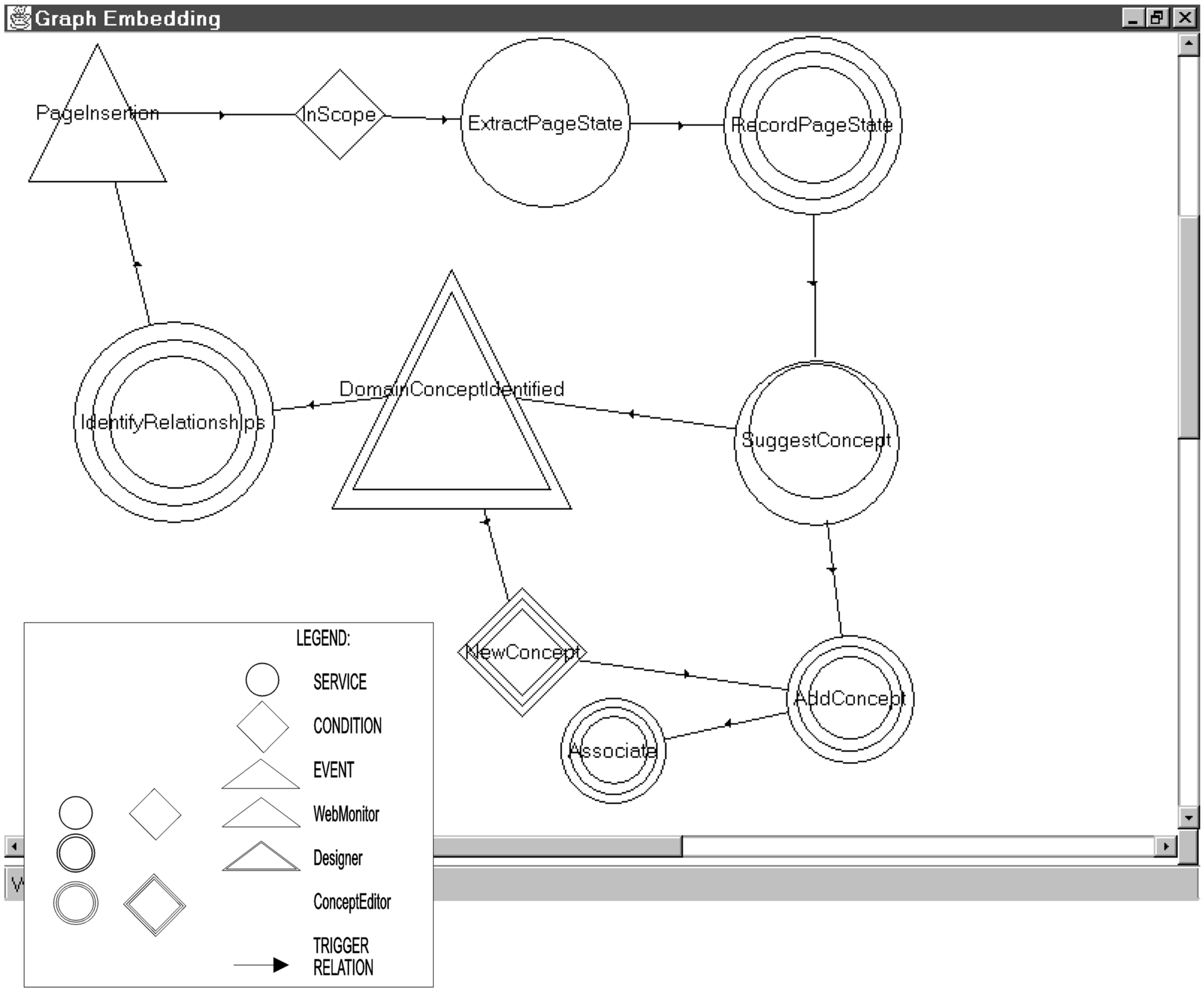


Fig. 2. Visual representation of rules r_1-r_3.

model in this architecture is the existence of an independent system (the Web) which cannot be put under the auspices of the information base transaction manager, yet its independent activities influence the information base through the semiautomatic propagation of Web artifact modifications to the domain model. Therefore, we use a combination of a pessimistic approach (i.e., locking), which is most commonly utilized in centralized databases, and an optimistic approach (i.e., timestamp ordering [30]), which has been suggested for distributed databases. The former is useful in preventing redundant aborts during the update process of the information base. The latter enforces a retrieval query to abort whenever modifications to conflicting Web artifacts are identified (see Section 4.3 for details). Another difficulty, which is a direct consequence of the characteristic noted above, is the incapability of recovering the information base by restoring its latest persistent state. This difficulty, not handled in the literature to the best of our knowledge, is the result of the ever-changing environment of the Web which is reflected in the information base. Any change to Web artifacts in the $WebMonitor$ scope makes a persistent state inaccurate, thus removing the conventional database "safety net" of resorting to the latest persistent state. This problem is handled in Section 4.2.1.

The use of rules and the involvement of the designer as the initiator of events indicate the existence of long running activities [7]. Advanced transaction models, e.g., Nested transactions [26] and Split transactions [31], provide various mechanisms for minimizing the impact of vast and prolonged lockings in databases; therefore, these transaction models allow for the increase in the parallel execution of long running activities. Unfortunately, these models become far less effective whenever the persistent database state is no longer valid and, therefore, when any access to that data cannot benefit the user. In the proposed architecture, the rules serve as an alternative mechanism for enhancing concurrent execution. For example, rules r_2-r_5 (see Appendix A) are processed in parallel as soon as the designer identifies the concept to associate with the inserted Web artifact. To eliminate redundant repetition of activities, which might result from the lack of concurrency control among rules, we introduce the notion of a transaction log in the next section.

4.1 Transaction Management

Before discussing the applicability of ACID transactions to the proposed architecture, the various activities within the architecture are examined more closely. It is readily seen that sets of services are related through generated events and executed rules, which provide a natural division of the activities. We first define a transaction log, followed by a division of the architecture's activities into six logical units of work.

Definition 2 (A transaction log). *Let tr be a transaction. A* transaction log $LG(tr) = (V, E)$ *is a directed acyclic graph such that:* $V = id(SR) \cup id(EV) \cup id(CN)$. $v \in LG(tr)$ *iff* v *is executed/triggered as part of tr.* $\langle v_i, v_j \rangle \in E$ *iff:*

1. $v_i \in id(SR), v_j \in id(EV)$, *and* v_i *generated* v_j,
2. $v_i \in id(EV), v_j \in id(CN)$, *and* v_i *triggered the rule which* v_j *is part of, or*
3. $v_i \in id(CN), v_j \in id(SR)$, *and* v_i*'s evaluation resulted in requesting* v_j.

A transaction log documents the activities within a single transaction. It assists the transaction manager in identifying the termination of transaction activities and in reducing the number of redundant activities. In particular, due to the parallel execution of rules, it is possible that multiple events will be signaled as a result of a single occurrence and, therefore, should not be processed independently. For example, as a result of the $IdentifyRelationships$ service, an $ArtifactInsertion$ of a Web artifact v_w event is possibly generated whenever the service cannot locate v_w in the repository. The $ArtifactInsertion$ event can be repeated several times for v_w whenever several instances of $IdentifyRelationships$ are executed concurrently. Once the transaction manager identifies such a repetition (using the transaction log), it makes sure that only a single instance of rule r_1 is generated for v_w. The transaction log would reflect this scenario by having multiple ingoing edges from services instances into the same event instance. Also, if $v_i \in id(CN)$ and a condition $c_i = \langle v_i, rule, false \rangle$, then v_i is a leaf in the transaction graph. This information, among other things, is utilized by the transaction manager to identify the termination of a transaction. It is worth noting that the transaction log cannot assist in restoring a consistent state since it cannot be utilized to retract any modifications to the Web artifacts which are beyond the control of the transaction manager.

We now provide six logical units of work within the architecture. Each of these logical units can be defined as a single transaction. The analysis of the architecture is based on the interrelationships among the activities described in Section 3.2. The interrelationships are easily identified through a visual representation of the rule set. For example, the interrelationships among the three rules in Fig. 2 suggest that they are part of a single transaction, as demonstrated below.

Artifact(s) addition involves the required activities in adding a new artifact to the scope of the $WebMonitor$, including the identification of its associated semantic concept and the relationships it is involved in. This unit of work involves the $AddToScope$ service followed by the processing of rules r_1-r_5 (see Appendix A). Each new artifact in the extended scope results in activating rule r_1 followed by a single $SuggestDomainConcept$ service request. An instance of any of the rules r_2-r_5 is triggered in response to the $DomainConceptIdentified$ event. Therefore, if the parameter of the $AddToScope$ service consists of n Web artifacts, out of which m are new to the scope, there will be m instances of each of the rules r_1-r_5 in the transaction log. The transaction manager is informed whenever $AddToScope$ is successfully completed and, therefore, the number of new Web artifacts is available to it[5] at that time. The transaction is completed once each of the $5m$ rules' instances is either fully executed or cannot be executed since its precondition is not satisfied.

Artifact(s) removal involves the required activities in removing an artifact from the scope of the $WebMonitor$, including disassociating it from semantic concepts and removing it from all relationships it is currently involved in. This unit of work involves the $RemoveFromScope$ service followed by the processing of rule r_9. Each removed artifact results in activating rule r_9. Therefore, if the parameter of the $RemoveFromScope$ service consists of n Web artifacts, there are n instances of rule r_9 as part of the transaction. The transaction manager is informed whenever $RemoveFromScope$ is successfully completed and, therefore, the number of r_9 instances available to it at that time. The transaction is completed once each of the n rule's instances is fully executed.

Artifact insertion involves the required activities that result from the detection of an artifact insertion to the scope of the $WebMonitor$. The artifact insertion unit of work is similar in many respects to the artifact addition, yet its origin is different. While the latter is triggered by expanding the scope of the $WebMonitor$, the former is triggered by the addition of an artifact to the already existing scope. This unit of work involves the triggering of rules r_1-r_5. As part of the processing of rule r_3, additional $ArtifactInsertion$ events can be generated. However, by the time all of the $IdentifyRelationships$ involved in the transaction are terminated, the number of instances of rule r_1 and, therefore, the number of $DomainConceptIdentified$ events, is known to the transaction manager. Once all of the rules triggered by $DomainConceptIdentified$ events have completed processing, the transaction is completed.

Document modification involves the required activities that result from the detection of a document modification, including the verification of the existing association with a semantic concept and the verification of existing relationships in which the artifact is involved. This unit of work involves a single instance of each of the rules r_6 and r_7 (although only one of them would be executed, due to the mutually exclusive conditions). In case r_6 is the rule to be executed, an instance of each of the rules r_4 and r_5 is also part of the transaction. Whenever a service

5. In the CoopWARE architecture, events and messages are being processed in order of arrival and, therefore, after the termination notification of $AddToConcept$ there will be no more $ArtifactInsertion$ events which are part of the transaction.

encounters a nonexisting Web artifact, the process of inserting this artifact is executed, given that the new artifact is indeed part of the scope. By the time the $UpdateRelationships$ service is terminated, the number of instances of rule r_1, and, therefore, the number of $DomainConceptIdentified$ events, is known to the transaction manager. Once all of the rules triggered by the $DomainConceptIdentified$ events have completed processing, as well as the instances of r_4-r_7, the transaction is completed.

Artifact deletion involves the required activities resulting from the detection of an artifact's removal, including its disassociation from semantic concepts and its removal from all relationships in which it is currently involved. The artifact deletion unit of work is similar in many respects to the artifact removal, yet its origin is different. While the latter is triggered by curtailing the scope of the $WebMonitor$, the former is triggered by the removal of an artifact from the already existing scope. This unit of work involves a single instance of rule r_9. Therefore, the transaction is completed once the services of r_9 are completed.

Artifact relocation involves the required activities as a result of the detection of an artifact relocation, including updating all the references to the artifact. This unit of work involves a single instance of rule r_{10}. Therefore, the transaction is completed once the $Disassociate$ service is completed.

We partition the six transaction types into two sets: The first includes the artifact(s) addition and the artifact(s) removal transactions and the second consists of the other four transactions. The former set is termed *scope transactions of Web artifacts* $v_1, v_2, ..., v_k$ and the latter set is termed *repository update transactions of Web artifacts* v_1, v_2 (v_2 is required only in the case of artifact relocation). A third set of transactions, termed *user retrieval transactions of Web artifacts* $v_1, v_2, ..., v_m$, involves the retrieval requests made by a user. It is worth noting that due to autonomous updates of Web artifacts, the updates cannot be defined as part of any scope transaction. Therefore, we assume the existence of a fourth transaction set, *artifact update transactions of a Web artifact* v, which involves the update of a single Web artifact v.

The rule's execution is associated with a specific transaction, based on a unique identification of messages among the components and the coordinator. For example, using the TMB protocol the request message requests a service and the response message provides information about the conclusion of the service.[6]. Otherwise, a Monologue message is being sent to the component. The request and the response messages are matched based on a TMB unique identifier which allows the coordinator to associate services and events with a specific transaction. This information, in turn, is available to the transaction manager in building the transaction log.

4.2 ACID Transactions

The next two sections discuss the applicability of ACID transactions to our architecture. In what follows, we discuss atomicity and consistency, while isolation is discussed in the next section. We refrain from discussing recovery issues and durability in this paper.

4.2.1 Atomicity

In Section 4.1, transactions were partitioned into four types, namely, scope transactions, repository update transactions, user retrieval transactions, and artifact update transactions based on their origin. Scope transactions and user retrieval transactions can be handled as conventional atomic database transactions and, therefore, a failure of the transaction restores the previous system's state to maintain atomicity. Artifact update transactions are not managed by the architecture transaction management and their impact is visible only through the events as provided by the $WebMonitor$. A repository update transaction cannot abort in the conventional database sense since the Web artifact modification is not within the system's control and cannot be reversed. Therefore, a failure of any activity in a repository update transaction cannot result in the restoration of the previous permanent state since some of the relationships and concepts may not be represented correctly once the Web artifact has changed. Therefore, there are two possible protocols of handling a failure of a repository update transaction as follows:

Redo protocol. The *redo* protocol first undoes any updates performed by the failing transaction and prevents, to the extent possible, any retrieval of concepts and artifacts within the scope of the transaction (by locking, if such a mechanism is enforced). The transaction is issued again at a later time. The redo protocol is useful in situations where the failure of the transaction is due to transient inaccessibility problem (i.e., V_T type). According to the redo protocol, the transaction manager, through the $WebMonitor$, polls the accessibility of the relevant Web artifacts and issues the transaction whenever accessibility is restored.

Void protocol. The *void* protocol considers the transaction to be nonexecutable and, therefore, issues a voiding transaction to void the impact of the involved Web artifact on the information base altogether. The void protocol performs an information base "dissection" to remove the minimal number of inconsistent artifacts and relationships in order to restore accuracy. Let $\mathcal{IB} = (V, E)$ be an information base and $tr(v_w)$ be a transaction applied to $\mathcal{IB}$, which involves the Web artifact $v_w \in V_W$. $tr_{void}(v_w)$ is a voiding transaction which results in the information base $\mathcal{IB}' = (V', E')$ such that $V'_W = V_W - \{v_w\}$,

$$\begin{aligned} E' = E - (&\{\langle v_w, \emptyset, v'_w\rangle\} \cup \{\langle v'_w, \emptyset, v_w\rangle\} \cup \langle v_c, \emptyset, v_w\rangle \\ &\cup \{\langle v_c, \mathrm{name}, v'_c\rangle | \exists v'_w \in V_W \wedge v_w \rightsquigarrow v'_w \\ &\wedge (\forall v''_w \in V_W, (v'_c \mapsto_{WebArtifacts} v'_w) \rightarrow (v'_w = v''_w))\}). \end{aligned}$$

Therefore, the modified information base voids any reference to the Web artifact. It is worth noting that while $tr_{void}(v_w)$ basically performs a $RemoveFromScope$ service, its failure may result in an inaccurate information base, and therefore should be attempted (using the redo protocol) until success.

6. This protocol is utilized only if the service is of type $TerminationStatusAvailable$. Otherwise, a Monologue message is being sent to the component.

A hybrid of both protocols is possible, where based on the inaccessibility type or by using a time-out mechanism, a decision whether a redo protocol should be replaced by a void protocol is taken.

4.2.2 Consistency

As a metric of consistency, we use the information base accuracy, as defined in Section 2. The following theorem ensures the consistency property in our architecture.

Theorem 1. *Let $\mathcal{IB} = (V, E)$ be an accurate information base and tr be a transaction applied to $\mathcal{IB}$. The execution of tr in isolation results in an accurate information base $\mathcal{IB}' = (V', E')$.*

Theorem 1 ensures that a transaction performed on an accurate information base results in an accurate information base. Therefore, most user queries (with the exception of a median transaction, see Section 4.3) are performed on an accurate information base, i.e., all Web artifacts are weakly accessible and the semantic model is supported by the underlying organization of the Web site.

The proof of Theorem 1 is simple, yet tedious. Therefore, we defer it to the appendix. The proof considers each of the transaction types to ensure that the impact on the information base (as provided in Table 2) restores accuracy as a response to the corresponding events.

It is worth noting that the information base accuracy is defined with respect to the underlying Web site. However, in some cases, the designer wishes to extend the semantic model by adding, for example, links that are not supported by the Web site hyperlinks (as is done in Microcosm [25] for example). This extension to our model can be easily attained by allowing user-defined semantic links, which will not be considered as part of the accuracy checking mechanism, that would allow the user to extend the semantic model based on her own preferences.

4.3 Concurrency Control

In this section, we present a concurrency control mechanism for accessing the information base. We define five types of primitive operations:

- $\mathrm{R}(v)$: A read operation for either a concept or a Web artifact.
- $\mathrm{W}(v)$: A write operation to either a concept or a Web artifact.
- $\mathrm{E}(v)$: An event signal regarding a Web artifact. $\mathrm{E}(v) \Longrightarrow \mathrm{W}(v)$.
- $\mathrm{R}(\langle v_i, v_j \rangle)$: A read operation for either a hyperlink (i.e., traversing the hyperlink to another artifact) or a repository property. $\mathrm{R}(\langle v_i, v_j \rangle) \Longrightarrow \mathrm{R}(v_i)\mathrm{R}(v_j)$.
- $\mathrm{W}(\langle v_i, v_j \rangle)$: A write operation of either a hyperlink or a repository property. $\mathrm{W}(\langle v_i, v_j \rangle) \Longrightarrow \mathrm{W}(v_i)$.

A *schedule* $S = \langle o_1, o_2 ... \rangle$ is defined as a sequence of primitive operations. In what follows, we use the following notation: $o_i < o_j$ stands for "o_i was executed before o_j." We assume a single thread transaction processing, therefore, $<$ constitutes a complete order. It is worth noting that, for a Web artifact v_w, $\mathrm{W}(v_w)$ can be identified from the corresponding file's time stamp (*Last Updated* in the server's header), as long as the translation of v_w's server time to the local time is available. Let tr be a transaction; $\{tr\} \subseteq S$ iff S consists of tr's operations. If S is a serial schedule and tr_1, tr_2 are two transactions such that $\{tr_1, tr_2\} \subseteq S$, then the statement $tr_1 < tr_2$ stands for "all operations of tr_1 were executed before all operations of tr_2." Finally, given a serial schedule S and two transactions tr_1, tr_2 such that $\{tr_1, tr_2\} \subseteq S$, a transaction tr_1 is the *closest successor* of type x of tr_2 iff $tr_1 < tr_2$ and there is no transaction $\{tr_3\} \subseteq S$, such that $tr_1 < tr_3 < tr_2$ and tr_3 is of type x. x stands for a scope transaction, a repository update transaction, a user retrieval transaction, or an artifact update transaction with some node v. The notion of the closest successor can be extended to any schedule (not necessarily serial) where $tr_1 < tr_2$ is interpreted as "tr_1 begins before tr_2."

Definition 3. (Schedule correctness). *A schedule S is* correct *iff:*

1. *S is serializable and*
2. *there exists a serial schedule S' such that:*
   - *S and S' are equivalent and*
   - *$\nexists \{tr_i, tr_j, tr_k\} \subseteq S$ such that:*
     - *$tr_i < tr_j < tr_k$,*
     - *tr_i is an artifact update transaction with v,*
     - *tr_j is tr_i's closest successor of type user retrieval transaction with v, and*
     - *tr_k is tr_j's closest successor of type repository update transaction with v.*

Definition 3 provides the conditions for a schedule correctness. In addition to the serializability condition, a schedule in this framework requires that no retrieval of a Web artifact v will be performed as long as the information base is not accurate with respect to v.

From the second condition in Definition 3, it becomes evident that conventional mechanisms for ensuring the correctness of schedules may fail in doing so in this framework. Indeed, 2PL does not necessarily guarantee a correct schedule. For example, consider a conventional locking mechanism in which a Read lock is shareable and a Write lock is exclusive. Further consider three transactions, tr_1, tr_2, and tr_3, such that tr_1 is an artifact update transaction with a parameter v_w, tr_2 is a user retrieval transaction of a concept v_c, and tr_3 is a repository update transaction triggered by an event $ArtifactModification$ (v_w). Finally, assume that $v_c \mapsto_{WebArtifacts} v_w \in E$. Let H be the history shown in Table 5. It is worth noting that transaction tr_1 is performed outside the architecture scope; therefore, it is not committed to the architecture locking mechanism. Transactions tr_2 and tr_3 both obey the 2PL protocol and the compatibility rules of the Read/Write locks. The only equivalent serialized schedule is $tr_1 tr_2 tr_3$. However, this schedule contradicts the second condition of correctness of Definition 3.

The correctness of this schedule can be resolved by using a strict 2PL, where a Write lock, issued by tr_3, will force tr_2 to only execute once tr_3 is done updating the repository. However, a conservative 2PL cannot be practically enforced in this framework since not all of the accessed items are

TABLE 5
A Nonserializable Schedule

tr_1	tr_2	tr_3
$\mathrm{W}(v_w)$		
		$\mathrm{E}(v_w)$
		$\mathrm{RLock}(v_w)$
	$\mathrm{RLock}(v_w)$	
		$\mathrm{R}(v_w)$
	$\mathrm{R}(v_w)$	
	$\mathrm{ULock}(v_w)$	
		$\mathrm{WLock}(v_w)$
		$\mathrm{W}(v_w)$
		$\mathrm{ULock}(v_w)$

TABLE 6
A Schedule with a Median Transaction

tr_1	tr_2	tr_3
$\mathrm{W}(v_w)$		
	$\mathrm{RLock}(v_w)$	
		$\mathrm{E}(v_w)$
		$\mathrm{RLock}(v_w)$
	$\mathrm{R}(v_w)$	
	$\mathrm{ULock}(v_w)$	
		$\mathrm{WLock}(v_w)$
		$\mathrm{R}(v_w)$
		$\mathrm{W}(v_w)$
		$\mathrm{ULock}(v_w)$

known in advance. For example, as a result of an $ArtifactInsertion$ event, the designer is requested to provide a concept to be associated with the newly inserted Web artifact. This concept is not known in advance and, therefore, cannot be locked at the beginning of the transaction. Also, even conservative 2PL cannot prevent the establishment of a Read lock by tr_2, as shown in the history example (Table 6). In this example, tr_2 establishes a Read lock, before the $WebMonitor$ traces the modifications made to the Web artifact v_w. In order to establish the appropriate mechanism for ensuring correct concurrency control, we now provide two more definitions.

Definition 4 (A median transaction). *Let S be a schedule. A transaction $\{tr\} \subseteq S$ is a* median transaction *w.r.t. a Web artifact v iff:*

1. *tr is a user retrieval transaction with v and*
2. *$\exists \{tr_i, tr_k\} \subseteq S$ such that:*
    - *tr_i is an artifact update transaction with v,*
    - *tr_k is tr_i's closest successor repository update transaction with v, and*
    - *tr_k acquires a lock on v after tr released a lock on v.*

Definition 5 (A weakly correct schedule). *A schedule S is* weakly correct *iff:*

1. *S is serializable and*
2. *exists a serial schedule S' such that:*
    - *S and S' are equivalent and*
    - *$\nexists \{tr_i, tr_j, tr_k\} \subseteq S'$ such that:*
        - *$tr_i < tr_j < tr_k$,*
        - *tr_i is an artifact update transaction with v,*
        - *tr_j is tr_i's closest successor user retrieval transaction with v,*
        - *tr_j is not a median transaction, and*
        - *tr_k is tr_j's closest repository update transaction with v.*

Definition 4 identifies a user retrieval transaction which accesses a conflicting artifact between the time a change has occurred to a Web artifact and the time it was notified by the $WebMonitor$. The weak correctness, as provided in Definition 5, relaxes the correctness criteria, as defined previously in Definition 3, to exclude median transactions. It is noteworthy that, while median transactions access an inconsistent information base, they cannot be prevented altogether since update transactions are performed independently of any local concurrency control mechanism.

We extend the common two locking categories to consist of three categories, namely, shareable, exclusive, and abortive. A shareable lock $L(v)$ is a lock that allows for other locks of v to be acquired before $L(v)$ is unlocked. An exclusive lock $L(v)$ is a lock that prohibits the acquisition of other locks for v before $L(v)$ is unlocked. An abortive lock is a lock that causes the failure of existing transactions that already hold a lock of v. The compatibility table (Table 7) partitions the read and write locks of both repository update and user retrieval transactions based on the above categorization.

Transaction tr_1 issues a lock before transaction tr_2. The relationships between two repository update transactions correspond to the conventional interpretation of read/write locking. However, a read lock of a repository update is considered exclusive for any read lock of a user retrieval transaction. Also, a read lock acquired by a user retrieval transaction results in aborting tr_1 should a repository

TABLE 7
Compatibility Table

tr_1	tr_2	Repository update		User retrieval
		$\mathrm{RLock}(v_w)$	$\mathrm{WLock}(v_w)$	$\mathrm{RLock}(v_w)$
Repository update	$\mathrm{RLock}(v_w)$	shareable	exclusive	exclusive
	$\mathrm{WLock}(v_w)$	exclusive	exclusive	exclusive
User retrieval	$\mathrm{RLock}(v_w)$	abortive	abortive	shareable

update transaction acquire a conflicting lock. In practical terms, aborting a user retrieval transaction entails a notification to the user that the requested portion of the concept model is being revised and would become available once it is aligned with the underlying Web site.

The concurrency control protocol follows the categorization of locks as provided above along with the following two concurrency rules:

- **2PL**. Each transaction (except artifact update transactions) follows the 2PL protocol.
- **RL of user retrieval transactions.** Read locks for repository update transactions are acquired immediately before the relevant read operation.

Any activity besides a read lock of a repository update transaction, e.g., write lock of a repository update transaction or the locking of a scope transaction, is constrained only by the 2PL protocol. The following theorem ensures that any legal schedule, i.e., a schedule that follows the concurrency control protocol is a weakly correct schedule.

Theorem 2. *Let S be a schedule. If all transactions in S follow the concurrency control protocol then S is weakly correct.*

The proof of the theorem is deferred to the appendix. It is based on the assumption devised in the categorization table that if a user retrieval transaction is not a median transaction, then it would abort as soon as a repository update transaction requests a conflicting lock. It is worth noting that in order for a schedule to be weakly correct, the enforcement of the locks categorization and the 2PL protocol are sufficient conditions. The latter rule is added to prevent redundant abort operations. Its aim is to allow any information of an artifact's update to delay a user retrieval transaction execution (by allowing a repository update transaction to acquire a read/write lock) rather than aborting it (in the case where a repository update transaction acquires a conflicting read/write lock).

We conclude this section with a discussion of the isolation property of the transactions in our architecture. The isolation property of an ACID transaction states that each transaction assumes it is executed alone. Any intermediate transaction results are not available to other concurrently executed transactions. The conventional interpretation of this property is that as long as the transaction is not committed to the database, any retrieval operation would not reflect the transaction's activity. A straightforward implementation of this property in our architecture, e.g., by using serializability as the sole correctness criteria, may result in an inaccurate retrieval result. For example, assume that, while an artifact deletion transaction regarding a Web artifact v_w is being executed, a user retrieval query which involves the relationship $\langle v_c, \text{name}, v_w \rangle$ is submitted. The user should not be directed to the Web artifact v_w, which no longer exists. Consequently, as soon as a repository update transaction identifies a modified page, all relationships that involve the page in question should become unavailable for retrieval until the transaction is completed (using either exclusive or abortive locks).

A final word on scope transactions: They do not interact with the user retrieval transaction the same way as repository update transactions do. The reason for that being the fact that scope transactions are initiated by the designer without any previous Web site updates. Therefore, they can be forced to wait until a user retrieval transaction has released a lock for the Web artifact in question, thus resulting in serializing the user retrieval transaction before the scope transaction.

5 Related Work

Several works in this area, e.g., WebSQL [24] and Microcosm [25], utilize the **structure** of the Web for generating information services. WebSQL is an SQL-like query language that allows the use of Web concepts (e.g., hyperlinks and URLs) as part of its syntax. Microcosm is a link manager that allows the user to add links among Web artifacts without modifying the Web artifacts themselves. Neither model provides the essential mechanisms to overcome the gap between structural (syntactic) interoperability and semantic interoperability. The latter cannot be attained by embedding Web concepts in a query language or by extending the flexibility of structural mechanisms, but rather it should consist of a domain model that provides a transparent manipulation of the Web. In addition, link managers lack the active capabilities of the proposed architecture where a semiautomatic mechanism maintains the additional link layer consistent with the content of the underlying Web resource.

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 resources[7] cannot guarantee realtime processing to provide the required semantic model.

Several research and commercial products, e.g., Ptolomaeus [30] and HotSuite [15], provide the capability of generating a Web site map upon request. These tools do not provide semantic capabilities and, therefore, lack the domain aspect of the proposed architecture. Also, once constructed, the site-map remains static unless a remapping is sought by the user, while the proposed model provides a reactive propagation of Web modifications to the semantic level. Moreover, the management tools (e.g., concurrency control) provided in this paper reduce the occurrence of errors common to the Web environment, e.g., the attempt of browsing a nonaccessible page, without the need for control over the Web resources. WAG [5], [34] allows the user to query the Web, rather than to browse it; therefore, this approach provides a transparent layer for Web-based information services. Nevertheless, WAG is passive and does not provide any mechanism for propagating Web changes to the domain model and, therefore, it falls into the same category as site-map generators.

Database transaction models may be applied on the server's end by using databases, either as the data source itself, or as a mechanism to manage the Web resource (a similar purpose with different means, as DOM and XML).

7. For example, an average lifetime of a URL is estimated as 44 days [17].

As an example of the former, one may consider Web-based APIs to a database (e.g., the w3-msql CGI script for interfacing mSQL databases [35]). Several examples of the latter also exist, including Hyper-G and Strudel. Hyper-G, along with its authoring tool Hyperwave [22], is a framework for collaborative development of Web applications. It allows a cleaner design and implementation of Web applications through the use of a database that maintains consistency among various Web resources. A similar approach, although more database-oriented, was taken in the Strudel research project [10]. Strudel supports the declarative specification of a Web site's content and structure and automatically generates a browsable Web site from a specification. However, the service of either Hyper-G or Strudel is rather limited, as some stringent requirements should be fulfilled for their successful deployment. These requirements include the collaboration of the participating applications and, in the case of Hyper-G, the use of it for all system components. Obviously, this approach is highly limited in scope under the Web's current state of affairs. In contrast, by virtue of its nature, our proposed architecture allows for the Web resources independence while supporting consistency where possible. Therefore, the proposed architecture is not limited to a predefined subset of the Web resources and, thus, allows flexible extension and retraction of the scope in question. Moreover, while Hyper-G provides a transaction mechanism for performing activities on the server's end, the transaction model does not respect transaction-less Web resources.

A final note is in order regarding concurrency control in this framework. The main characteristic which precludes the use of a conventional transaction model in this architecture is the existence of an independent system (the Web) which cannot be put under the auspices of the information base transaction manager, yet its independent activities influence the information base through the semiautomatic propagation of Web artifact modifications to the semantic model. Accounts of such systems cannot be found within the existing database research and, therefore, we provide an alternative transaction model (Section 4) which takes into account this characteristic.

6 Conclusion

This paper proposes a Web-based application management system as a generic architecture offering specific functionalities for managing Web-based applications and coping with semantic diversity among its information sources. These functionalities include facilities for building, in a bottom-up fashion, a domain model of the information contained in the application's Web sites, basic syntactic integrity enforcement mechanisms, as well as a transaction model specifically tailored to Web-based transaction processing. A Web-based application management system, unlike a DBMS, assumes no control over the underlying data sources, nor can it assume the availability of any transaction capabilities (e.g., ready-to-commit) in these data sources. Therefore, its control concentrates on the domain model, keeping it consistent to the degree possible, with the underlying Web application.

An ongoing collaboration at the University of Toronto and at Rutgers University focuses on the implementation of the various components of the system as a proof of concept. For this implementation, CoopWARE serves as the coordination tool, MIRROR [18] as the concept editor, and WebSQL [24] as the basis for the `WebMonitor`. A primitive Web-based navigation tool allows queries of the type demonstrated in Section 2.4; however, a full fledged GUI interface is yet to be built.

This work constitutes only a beginning in establishing a framework for Web-based application management. Efficient algorithms need to be devised for syntactic consistency enforcement. Also, performance analysis is required to study the parameters which affect the performance of Web-based applications. Lastly, our proposal needs to be evaluated with real Web-based applications to study the amount of effort required in order to develop these applications. After all, generic tools of any sort are useless unless they substantially reduce the amount of effort required to build and manage software.

Appendix A

Application Maintenance Using CoopWARE Rules

Table 8 provides the set of rules for maintaining the accuracy of the information model. Rules r_1-r_5 handle the artifact insertion scenario. Rule r_1 sends a request to identify a domain concept. The request includes a default concept generated from the Web artifact. 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 artifact (if it is a document) 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.

Rules r_4-r_8 provide the required activities in case of a document modification. Rule r_6 handles a situation where a major modification to a document (either in the document's title or in terms of a distance metric) is detected. In this case, the designer is prompted to identify the concept the modified document is now associated with. Therefore, the concept related with the modified document is retrieved (using the $FindConcept$ service) followed by the disassociation of the concept from the URL and a request for the designer intervention to decide on the accuracy of the association with respect to the modified document. Rule r_7 is executed whenever a minor modification to a document is detected. In this case, the new state of the document is recorded and relationships are updated based on a comparison of the previous state with the current one. Rule r_8 is executed once a domain concept is verified and its execution flow is similar to that of r_7. Rule r_1 is triggered in case a new artifact insertion is detected during the modification process.

TABLE 8
Domain Model Maintenance Rules

Rule r_1:		Rule r_6:	
Event:	ArtifactInsertion(URL)	Event:	DocumentModification(URL)
Condition:	InScope(URL)	Condition:	RetrieveArtifactTitle(URL)!=URL.Title
Action:	AddArtifact(URL)		or Delta(RetrieveDocumentState,
	State:=ExtractArtifactState(URL)		ExtractDocumentState(URL))>K
	RecordArtifactState(State)	Action:	State:=RetrieveDocumentState(URL)
	SuggestDomainConcept(URL, URL.title)		Concept:=FindConcept(URL)
	AddConcept(URL.title)		Disassociate(URL, Concept)
	Associate(URL, URL.Title)		AddConcept(URL.title)
			Associate(URL, URL.Title)
			VerifyDomainConcept(URL, Concept, URL.Title)
Rule r_2:		Rule r_7:	
Event:	DomainConceptIdentified(URL, Concept)	Event:	DocumentModification(URL)
Condition:	NewConcept(Concept)	Condition:	RetrieveArtifactTitle(URL)=URL.Title and
Action:	AddConcept(Concept)		Delta(RetrieveDocumentState,
			ExtractDocumentState(URL))<=K
		Action:	State1:=RetrieveDocumentState(URL)
			State2:=ExtractDocumentState(URL)
			Concept:=FindConcept(URL)
			RecordDocumentState(State2)
			UpdateRelationships(State1, State2, Concept)
Rule r_3:		Rule r_8:	
Event:	DomainConceptIdentified(URL, Concept)	Event:	DomainConceptVerified(URL, Concept)
Action:	IdentifyRelationships(URL, Concept)	Action:	State1:=RetrieveDocumentState(URL)
			State2:=ExtractDocumentState(URL)
			RecordDocumentState(State2)
			UpdateRelationships(State1, State2, Concept)
Rule r_4:		Rule r_9:	
Event:	DomainConceptIdentified(URL, Concept)	Event:	ArtifactRemoval(URL)
	or DomainConceptVerified(URL, Concept)	Action:	Concept:=FindConcept(URL)
Condition:	URL.title≠Concept		Disassociate(URL, Concept)
Action:	Disassociate(URL, URL.title)		RemoveRelationships(URL, Concept)
	Associate(URL, Concept)		RemoveDocumentState(URL)
			RemoveArtifact(URL)
Rule r_5:		Rule r_{10}:	
Event:	DomainConceptIdentified(URL, Concept) or	Event:	ArtifactRelocation(OldURL, NewURL)
	DomainConceptVerified(URL, Concept)	Action:	Concept:=FindConcept(OldURL)
Condition:	Ingoing(URL.title)≤1 and URL.title≠Concept		RelocateRelationships(OldURL, NewURL)
Action:	Disassociate(URL, URL.title)		Disassociate(OldURL, Concept)
	RemoveConcept(URL.title)		Associate(NewURL, Concept)

Artifact removal is handled in Rule r_9. The associated concept is retrieved and disassociated from the Web artifact, along with all relationships in which the removed artifact is involved. Finally, Rule r_{10} handles an artifact relocation by updating the association and the relationships to be consistent with the current artifact location.

APPENDIX B

PROOFS OF THEOREMS

Theorem 1. *Let S be a schedule. If all transactions in S follow the concurrency control protocol then S is weakly correct.*

Proof (partial). Let $\mathcal{IB} = (V, E)$ be an accurate information base
Definition 1 $\Longrightarrow$

1. $\mathcal{IB}\ (t)$ is weakly accessible and
2. $\forall\{v_i, v_j\} \subseteq V_C(t), \langle v_i, \text{name}_{ij}, v_j\rangle \in E(t) \Longrightarrow$
$\exists\{v_i', v_j'\} \subseteq V_W(t) | v_i \mapsto^t_{WebArtifacts} v_i'$
$\wedge v_j \mapsto^t_{WebArtifacts} v_j' \wedge v_i' \rightsquigarrow {}^t v_j'$.

Let $tr(v_w)$ be a transaction applied to $\mathcal{IB}$. The proof relates to each of the six transaction types separately. Due to space limitations, we introduce the proof for two of the six types. The proof for the other four types is similar:

Artifact deletion: Let v_w be the deleted Web artifact.
Table 2 and Rule r_9 in Table 8 $\Longrightarrow v_w$ is the only affected Web artifact.
Assume that the resulting information base $\mathcal{IB}' = (V', E')$ is not accurate.
$\Longrightarrow \mathcal{IB}'(t)$ is not weakly accessible **(1)** or
$\exists\{v_i, v_j\} \subseteq V_C', \langle v_i, \text{name}_{ij}, v_j\rangle \in E' \wedge \forall\{v_i', v_j'\} \subseteq V_W'$,
$\neg(v_i \mapsto_{WebArtifacts} v_i') \vee$ **(2)**
$\neg(v_j \mapsto_{WebArtifacts} v_j') \vee$ **(3)**
$\neg(v_i' \rightsquigarrow v_j')$.**(4)**
$\forall v \in V_W' \Longrightarrow v \in V_W$.
From statement **(1)** $\Longrightarrow \exists v \in V_W'(t) | v \notin V_A'(t) \cup V_T'(t)$.
$\Longrightarrow \exists v \in V_W' | v \in V_P'(t)$.
Since the impact of $RemoveArtifact\ implies$
$V_W' := V_W - \{v_w\} \Longrightarrow \exists v \in V_W - \{v_w\} | v \in V_P(t) - \{v_w\}$.
$\Longrightarrow \exists v \in V_W | v \in V_P(t)$.
$\Longrightarrow \mathcal{IB}$ is not accurate, which is a contradiction.
As for statement **(2)**, Since the impact of $Disassociate$ implies $E' := E - \{\langle v_c, \emptyset, v_w\rangle\} \Longrightarrow v_i' \neq v_w$
$\Longrightarrow \exists v_i \in V_C | \langle v_i, \text{name}_{ij}, v_j\rangle \in E \wedge \forall v_i' \in V_W$,
$\neg(v_i \mapsto_{WebArtifacts} v_i')$
$\Longrightarrow \mathcal{IB}$ is not accurate, which is a contradiction.
As for statement **(3)**, Since the impact of $Disassociate$ implies $E' := E - \{\langle v_c, \emptyset, v_w\rangle\} \Longrightarrow v_j' \neq v_w$

$\Longrightarrow \exists v_j \in V_C | \langle v_i, \mathrm{name}_{ij}, v_j \rangle \in E \wedge \forall v'_j \in V_W,$
$\neg(v_j \mapsto_{WebArtifacts} v'_j)$
$\Longrightarrow \mathcal{IB}$ is not accurate, which is a contradiction.
As for statement **(4)**, Since the impact of $RemoveDocumentState$ implies
$E := E - (\{\langle v_w, \emptyset, v'_w \rangle | v'_w \in V_W\}$
$\cup \{\langle v'_w, \emptyset, v_w \rangle | v'_w \in V_W\}) \Longrightarrow v'_i \neq v_w \wedge v'_j \neq v_w.$
Since statements **(2)** and **(3)** above lead to contradiction, assuming **(4)** we get $\exists \{v_i, v_j\} \subseteq V_C, \langle v_i, \mathrm{name}_{ij}, v_j \rangle \in E \wedge \forall \{v'_i, v'_j\} \subseteq V_W - \{v_w\}, (v_i \mapsto_{WebArtifacts} v'_i$
$\wedge v_j \mapsto_{WebArtifacts} v'_j \rightarrow \neg(v'_i \rightsquigarrow v'_j))$
$\Longrightarrow \exists \{v_i, v_j\} \subseteq V_C, \langle v_i, \mathrm{name}_{ij}, v_j \rangle \in E \wedge \forall \{v'_i, v'_j\} \subseteq V_W,$
$(v_i \mapsto_{WebArtifacts} v'_i \wedge v_j \mapsto_{WebArtifacts} v'_j \rightarrow \neg(v'_i \rightsquigarrow v'_j))$
$\Longrightarrow \mathcal{IB}$ is not accurate, which is a contradiction.
$\Longrightarrow \mathcal{IB}'$ which results from an artifact deletion transaction is accurate.

Artifact relocation: Let v_w and v'_w be the Web artifacts before and after the relocation, respectively.
Table 2 and Rule r10 in Table 8 $\Longrightarrow v_w$ and v'_w are the only affected Web artifacts.
Assume that the resulting information base $\mathcal{IB}' = (V', E')$ is not accurate.
$\Longrightarrow \mathcal{IB}'(t)$ is not weakly accessible **(5)** or
$\exists \{v_i, v_j\} \subseteq V'_C, \langle v_i, \mathrm{name}_{ij}, v_j \rangle \in E' \wedge \forall \{v'_i, v'_j\} \subseteq V'_W,$
$\neg(v_i \mapsto_{WebArtifacts} v'_i) \vee$ **(6)**
$\neg(v_j \mapsto_{WebArtifacts} v'_j) \vee$ **(7)**
$\neg(v'_i \rightsquigarrow v'_j)$.**(8)**
For any $v \neq v'_w$, statement **(5)** suggests that $\mathcal{IB}(t)$ is not weakly accessible $\Longrightarrow \mathcal{IB}$ is not accurate, which is a contradiction.
As for $v = v'_w$ since the only change to the artifact $v_w$was its URL, while its accessibility remained the same, statement **(5)** suggests that $\mathcal{IB}(t)$ is not weakly accessible $\Longrightarrow \mathcal{IB}$ is not accurate, which is a contradiction.
As for statement **(6)-(8)**, since the impact of $RelocateRelationships$ (the only service to modify relationships in rule r10) involves $E := E - \{\langle v, \emptyset, v' \rangle | v = v_w \vee v' = v_w\}$ and $\mathcal{IB}$ is accurate,
$v'_i \neq v_w \wedge v'_j \neq v_w$.
From statement **(6)** $\Longrightarrow \exists v_i \in V_C | \langle v_i, \mathrm{name}_{ij}, v_j \rangle$
$\in E \wedge \forall v'_i \in V_W, \neg(v_i \mapsto_{WebArtifacts} v'_i)$
$\Longrightarrow \mathcal{IB}$ is not accurate, which is a contradiction.
From statement **(7)** $\Longrightarrow \exists v_j \in V_C | \langle v_i, \mathrm{name}_{ij}, v_j \rangle$
$\in E \wedge \forall v'_j \in V_W, \neg(v_j \mapsto_{WebArtifacts} v'_j)$
$\Longrightarrow \mathcal{IB}$ is not accurate, which is a contradiction.
As for statement **(8)**, The following two cases should be considered:
$v'_i = v'_W : \neg(v'_w \rightsquigarrow v'_j) \Longrightarrow \neg(\langle v'_w, \emptyset, v'_j \rangle \in E')$
$\Longrightarrow \neg(\langle v_w, \emptyset, v'_j \rangle \in E) \Longrightarrow \neg(v_w \rightsquigarrow v'_j).$
Since $\mathcal{IB}$ is accurate and $\langle v_i, \mathrm{name}_{ij}, v_j \rangle \in E \Longrightarrow \exists v''_i$
$\in V_W | v''_i \neq v_w \wedge v_i \mapsto_{WebArtifacts} v''_i \wedge v_j \mapsto_{WebArtifacts} v'_j$
$\wedge v''_i \rightsquigarrow v'_j$
$\Longrightarrow \exists \{v''_i, v'_j\} \subseteq V'_C | v_i \mapsto_{WebArtifacts} v''_i \wedge v_j \mapsto_{WebArtifacts} v'_j$
$\wedge v''_i \rightsquigarrow v'_j$. Contradiction to **(8)**.
$v'_j = v'_W : \neg(v'_i \rightsquigarrow v'_w) \Longrightarrow \neg(\langle v'_i, \emptyset, v'_w \rangle \in E')$
$\Longrightarrow \neg(\langle v_i, \emptyset, v'_w \rangle \in E) \Longrightarrow \neg(v_i \rightsquigarrow v'_w).$
Since $\mathcal{IB}$ is accurate and $\langle v_i, \mathrm{name}_{ij}, v_j \rangle \in E \Longrightarrow \exists v''_j$
$\in V_W | v''_j \neq v_w \wedge v_i \mapsto_{WebArtifacts} v'_i \wedge v_j \mapsto_{WebArtifacts} v''_j$
$\wedge v'_i \rightsquigarrow v''_j$
$\Longrightarrow \exists \{v'_i, v''_j\} \subseteq V'_C | v_i \mapsto_{WebArtifacts} v'_i$
$\wedge v_j \mapsto_{WebArtifacts} v''_j \wedge v'_i \rightsquigarrow v''_j$. Contradiction to **(8)**.
$\Longrightarrow \mathcal{IB}'$ which results from an artifact relocation transaction is accurate.
$\Longrightarrow$ Execution of tr in isolation results in an accurate information base $\mathcal{IB}' = (V', E')$. □

Theorem 2. *Let S be a schedule. If all transactions in S follow the concurrency control protocol then S is weakly correct.*

Proof. Let S be a schedule. Assume that all transactions in S follow the concurrency control protocol yet S is not weakly correct.
Due to the 2PL rule, S is serializable. Therefore, for any equivalent schedule S' exists $\{tr_i, tr_j, tr_k\} \subseteq S'$ such that:

- $tr_i < tr_j < tr_k$,
- tr_i is a artifact update transaction with v,
- tr_j is tr_i's closest successor user retrieval transaction with v,
- tr_j is not a median transaction w.r.t. v, and
- tr_k is tr_j's closest repository update transaction with v.

tr_j is not a median transaction. Therefore, tr_k acquires a lock on v before tr_j released a lock on v. However, if this is case, then tr_j aborts and, therefore, S' is not equivalent to S, which is a contradiction.
$\Longrightarrow$ If all transactions in S follow the concurrency control protocol, then S is weakly correct. □

ACKNOWLEDGMENTS

The authors thank Scott S. Kerr, who contributed to the original setting of the architecture as illustrated in [19] and extended herein. The graph analysis tool was built, in part, with the assistance of Jun Du. The work was supported, in part, by the Rutgers Undergraduate Research Fellow Program, Rutgers Faculty of Management, and the Information Technology and Electronic Commerce Clinic (ITECC); also, by Canada's Natural Sciences and Engineering Research Council, as well as the Institute of Robotics and Intelligent Systems.

REFERENCES

[1] S. Abiteboul, S. Cluet, and T. Milo, "Querying and Updating the File," *Proc. 19th Int'l Conf. Very Large Data Bases,* pp. 73–84, 1993.
[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener, "The LOREL Query Language for Semistructured Data," *Int'l J. Digital Libraries,* vol. 1, no. 1, 1997.
[3] The BackWeb Home Page, http://www. backWeb. com.
[4] E. Buss, "Investigating Reverse Engineering Technologies for the CAS Program Understanding Project," *IBM Systems J.* vol. 33, no. 3, pp. 477–500, 1994.
[5] T. Catarci, S.-K. Chang, D. Nardi, G. Sanctucci, and M. Lenzrini, "WAG: Web-At-a-Glance," *Proc. 31st Ann. Hawaii Int'l Conf. System Sciences, Vol. VII,* pp. 344–353, Jan. 1998.
[6] C.J. Date, *An Introduction to Database Systems.* White Plains, N.Y.: Addison-Wesley, 1983.
[7] U. Dayal, M. Hsu, and R. Ladin, "A Transactional Model for Long-Running Activities," *Proc. 17th Very Large Data Bases,* pp. 113–122, Sept. 1991.
[8] "Document Object Management," Year. http://www.w3.org/DOM.

[9] U. Erlingsson and M. Krishnamoorthy, “Interactive Graph Drawing,” Year. http://www.cs.rpi.edu/projects/pb/graphdraw/.

[10] M.F. Fernandez, D. Florescu, J. Kang, A.Y. Levy, and D. Suciu, “Catching the Boat with Strudel: Experiences with a Web-Site Management System,” *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 414–425, June 1998.

[11] A. Gal, “Handling Constantly Changing Metadata,” *Proc. Second IEEE Metadata Conf.,* Sept. 1997, http://computer. org/conferen/proceed/meta97/.

[12] A. Gal, “Modelling Cooperation Among Information Systems Using Control Elements,” *Proc. Third Int'l Workshop Next Generation Information Technologies and Systems,* pp. 125–132, July 1997.

[13] A. Gal and J. Mylopoulos, “The CoopWARE Demo: Wrapping up a Legacy System,” 1997 http://www.cs.toronto.edu/~coopware.

[14] A. Gal and J. Mylopoulos, “Supporting Distributed Autonomous Information Services Using Coordination,” *Int'l J. Cooperative Information Systems,* vol. 9, no. 3, pp. 255–282, 2000.

[15] “Internet HotSuite,” 1998, www.documagix.com/products/hotsuite.htm.

[16] M. Jarke, R. Gallersdoerfer, M. Jeusfeld, M. Staudt, and S. Eherer, “A Deductive Object Base for Metadata Management,” *J. Intelligent Information Systems,* vol. 4, no. 2, pp. 167–192, 1995.

[17] B. Kahle, “Preserving the Internet,” Year, http://www.sciam.com/0397issue/0397kahle.html.

[18] S. Kerr, “Data Integration and Change Management Using Active Metamodels,” master's thesis, Univ. of Toronto, 1998.

[19] S. Kerr, A. Gal, and J. Mylopoulos, “Information Services for the Web: Building and Maintaining Domain Models,” *Proc. Third IFCIS Int'l Conf. Cooperative Information Systems (CoopIS '98),* pp. 4–13, Aug. 1998.

[20] S. Malaika, “Resistance is Futile: The Web Will Assimilate Your Database,” *Bulletin of the Technical Committee on Data Eng.,* vol. 21, no. 2, pp. 4–13, 1998.

[21] U. Manber and P. Bigot, “The Search Broker,” *Proc. First Usenix Symp. Internet Technologies and Systems,* pp. 231–240, Dec. 1997.

[22] H. Maurer, *Hyper-G Now Hyperwave: The Next Generation Web Solution.* Addison-Welsley, 1996.

[23] The MCF Home Page, http://www.w3.org/TR/NOTE-MCF-XML/MCF-tutorial.html.

[24] A.O. Mendelzon, G.A. Mihaila, and T. Milo, “Querying the World Wide Web,” *Proc. Conf. Parallel and Distributed Information Systems (PDIS),* pp. 80–91, 1996.

[25] “Microcosm Link Management,” Year, http://www.multicosm.com/microcosm.

[26] J.E.B. Moss, “Nested Transactions: An Approach to Reliable Distributed Computing,” PhD thesis, MIT, Cambridge, Mass., 1981.

[27] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis, “Telos: Representing Knowledge about Information Systems,” *ACM Trans. Information Systems,* vol. 8, no. 4, Oct. 1990.

[28] J. Mylopoulos, A. Gal, K. Kontogiannis, and M. Stanley, “A Generic Integration Architecture for Cooperative Information Systems,” *Proc. First IFCIS Int'l Conf. Cooperative Information Systems (CoopIS '96),* pp. 208–217, June 1996.

[29] A. Pentland, R.W. Picard, and S. Scarloff, “Photobook: Tools for Content-Based Manipulation of Image Databases,” *Int'l J. Computer Vision,* 1996.

[30] “PTOLOMAEUS, the Web Cartographer,” Year, http://www.inf.uniroma3.it/vernacot/ptolpage.htm.

[31] C. Pu, G. Kaiser, and N. Hutchinson, “Split-Transactions for Open-Ended Activities,” *Proc. 14th Int'l Conf. Very Large Databases (VLDB),* pp. 26–37, 1988.

[32] The RDF Home Page, http://www.w3.org/Metadata/.

[33] D.P. Reed, “Implementing Atomic Actions on Decentralized Data,” *ACM Trans. Computer Systems,* vol. 7, no. 5, pp. 3–23, Feb. 1983.

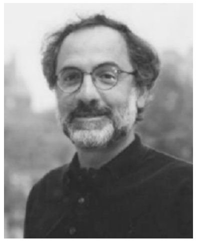
[34] M. De Rosa, T. Catarci, L. Iocchi, D. Nardi, and G. Santucci, “Materializing the Web,” *Proc. Third IFCIS Conf. Cooperative Information Systems (CoopIS '98),* pp. 24–31, Aug. 1998.

[35] Hughes Technologies, “mSQL Online Manual,”http://www.Hughes.com.au/library/msql2/manual/. 1997.

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

[37] “eXtensible Markup Language at the W3 Consortium,”http://www.w3.org/XML.

Avigdor Gal received the DSc degree from the Technion-Israel Institute of Technology in 1995 in the area of temporal active databases. He is a faculty member at the MSIS Department at Rutgers University. He has published 30 papers in journals (e.g., *IEEE Transactions on Knowledge and Data Engineering*), books (*Temporal Databases: Research and Practice*) and conferences (e.g., ER '95, CoopIS '98) on the topics of information systems architectures, active databases, and temporal databases. Together with Dr. John Mylopoulos, Avigdor has chaired the Distributed Heterogeneous Information Services workshop at HICSS '98 and was the guest editor of a special issue by the same name in the *International Journal of Cooperative Information Systems*. He served on the program committees of several international conferences and workshops, and he is both a member of the ACM and the IEEE computer society.



John Mylopoulos received the PhD degree from Princeton University in 1970, and is professor of computer science at the University of Toronto. His research interests include knowledge representation and conceptual modeling, covering languages, implementation techniques, and applications. His publication list includes more than 150 refereed journal and conference proceedings papers and six edited books. He is the recipient of the first ever Outstanding Services Award given out by the Canadian AI Society (CSCSI), a corecipient of the best paper award of the 1994 International Conference on Software Engineering (ICSE), a fellow of the American Association for AI (AAAI), and is currently serving a 4-year term as president of the VLDB Endowment.