Structuring and Querying the Web through Graph-Oriented Languages

E. Damiani[1] L. Tanca[2]
edamiani@crema.unimi.it
tanca@biotech.sci.univr.it

[1] Università di Milano - Polo di Crema
[2] Università di Verona and Politecnico di Milano

Abstract. In order to pose effective queries to Web sites, some form of site data model must be implicitly or explicitly shared by users. Many approaches try to compensate for the lack of such a common model by considering the hypertextual structure of Web sites; unfortunately, this structure has usually little to do with data semantics. In this paper a different technique is proposed that allows for both navigational and logical/conceptual description of Web sites. The data model is based on *WG-log*, a query language based on the graph-oriented database model of *GOOD* [Gys94] and *G-log* [Par95], which allows the description of data manipulation primitives via (sets of) graph(s). The *WG-log* description of a Web site schema is lexically based on standard hypermedia design languages, thus allowing for easy schema generation by current hypermedia authoring environments. The use of *WG-log* for queries allows graphic query construction with respect to both the navigational and the logical parts of schemata. Site schemata are managed by *Schema Robots*, which assist clients in the process of *identification* and *retrieval* of a set of *candidate schemata*. On the basis of the set of candidate schemata, the client may then query individual Web sites; extensive data caching is used to avoid *flooding* resulting from an excessive number of candidates. A remote *Query Manager* process, running side by side with standard Web servers, manages query execution and handles the *presentation* of the results to the client. Our schema is particularly suited for Intranets, while allowing for a smooth migration of Internet Web sites as more and more of them are produced on the basis of hypermedia design and generation methodologies.

1 Introduction

The steady growth in the amount of data published via the World Wide Web (WWW) has led to a number of attempts to index Web's contents. Initially, these attempts only tried to collect and index title-like information about every reachable page of data on the WWW, and then build Boolean keyword searches into that database. Jonathan Fletcher's JumpStation, probably the first well-known system designed to index WWW information, collected only information marked by HTML `<TITLE>`-tags from pages it encountered. In the past few years, as the amount of WWW data continued to grow steadily, users grew unsatisfied with search engines based on pure keyword matching. Today, one could hardly find a Web search engine relying on term indexing alone. Some keyword-based indexes, like WebCrawler [Pin97] and Lycos [Lyc97] try to complement keyword indexing by taking into account the HTML document structure in order to make educated guesses about semantics. For example, Lycos summarizes the actual content of documents by taking advantage of human-tagged information (HTML headings), of often-appearing keywords and of the introductory text that generally is positioned at the beginning of a file. Besides exploiting Altavista [Alt97] for keyword indexing, the popular

Yahoo [Yah97] search engine relies on a broad hierarchical classification system of subjects, much similar to those used by the Library of Congress or by the ACM Classification of Computer Science Topics. Yahoo's success has spawned multiple similar tools, all based on the idea of providing large, monolithic servers holding indexes of site contents (Point [Lyc97-2], Magellan [McK97], ..). Recently, *meta-search* services (MetaCrawler [Sel95], WebCompass [Qua97], SavvySearch [Dre97] and others) have been built that are able to use monolithic indexes as subroutines, querying all available services in parallel and then aggregating the results. Although many of these search engines present a sophisticated query interface, the results they deliver are perceived by the user community as unsatisfactory. In our opinion, this situation is mainly due to the following three causes:

- **Noise effect**
  In the current systems, query language interfaces suggest precise semantics while the underlying keyword search mechanism remains mainly syntactical in nature and therefore prone to the well-known *noise* and *silence* side-effects. Sophisticated indexing techniques exploiting HTML tagging may relieve this problem, but by no means solve it completely.
- **Flatness of results**
  Currently, query results are flat lists of pages that do not capture the underlying structure of the searched Web sites. As a consequence, retrieved information neither is easily presented to the user nor can be efficiently reorganized.
- **Non-HTML objects indexing** The increasing presence of non-HTML objects [Hu96] attached to Web pages (such as multimedia clips or Java *applets*) jeopardizes automatic index construction and update.

The above considerations suggest to complement keyword-based *searching* with database-style support for *querying* the Web. Several research projects addressed this problem, and three main WWW query languages have been proposed so far: Web3QL [Kon95], WebSQL [Men96] and WebLog [Lak96]. These languages are modelled after standard SQL and Datalog languages used for DBMS, but give only a small fraction of their power. In fact, while no data model at all is defined for Web3QL and WebSQL, WebLog provides indeed a simple data model, but lacks an explicit concept of schema. This may be due to the fact that, while the advantages of schema-aware query formulation are widely recognized in the database context, this technique has been considered unfeasible on the WWW because no schema information is associated to Web sites. A complete, graphical description and retrieval system for hypermedia is described in [Luc96]. However, this system does not include scheme-based query facilities. In [Luk97] instead of a formal notion of a schema, a system is proposed to annotate HTML pages with semantic knowledge, allowing for structured queries instead of keyword-based searches. A related though distinct approach [Atz97] relies on the notion of *page schema* i.e a description of page structure automatically extracted from HTML code. In [Atz97] the schema of a Web site is defined as a collection of page schemas connected by links. As we shall see in the next Section, this paper adopts a different view, describing a full-fledged notion of schema and proposing to reuse site design artifacts to attain (semi)automatic construction of site schemata.

1.1 An outline of our approach

Our approach is based on the following two principles:

- The availability of a schema is a prerequisite for the development of an effective query mechanism, since the schema carries most of the semantic information needed for querying.

– If a well specified methodology is used to design a Web site, some notion of schema is present during the site design process

So, this paper proposes to reuse site design artefacts to attain (semi)automatic construction of schemata for Web sites. As outlined in Fig. 1, many popular authoring environments for Web sites already hint at the idea of some sort of navigational schema to be chosen by the user as the basis of automatic site generation. Moreover, several research prototypes of Web site generators [Fra97]) based on hypermedia design languages such as HDM [Gar95] (*Hypermedia Design Method*), *YOO* [Bal95], RMM [Isa95] and the like, are becoming available. This suggests the possibility of semi or fully automatic generation of schemata during the design process, allowing for smooth migration to a more organized Web as more and more sites are produced on the basis of standard design and generation methodologies. In our

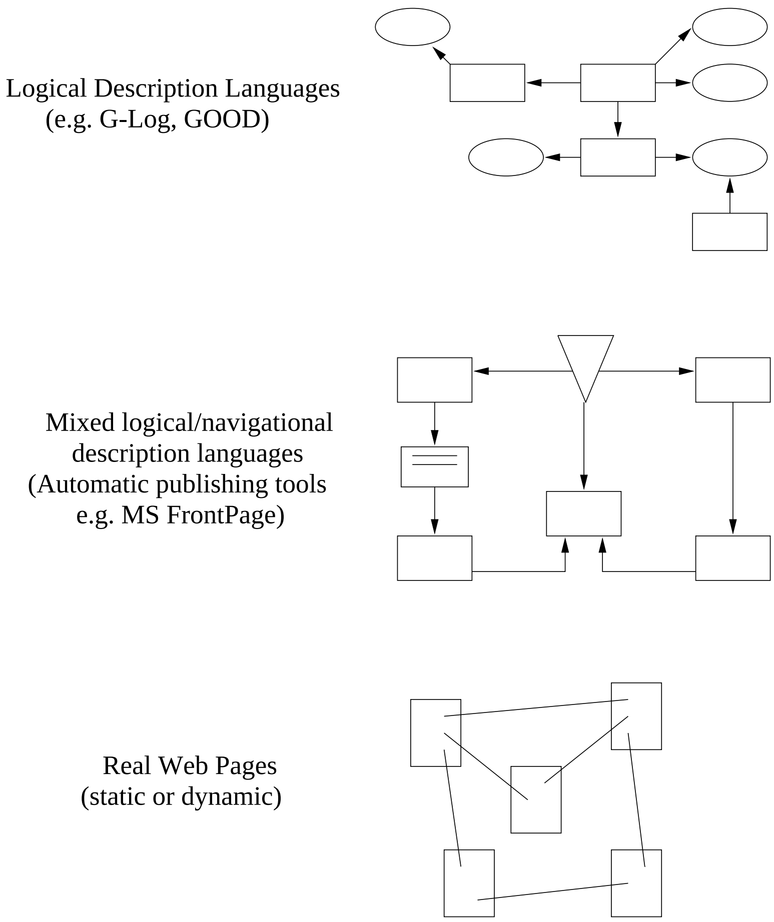

Fig. 1. Links between conceptual and hypermedia description languages

approach, Web site schemata form a *distributed hierarchy*, managed by server processes called *Schema Robots*. A *Schema Robot* is a server process which mantains and provides information about Web sites schemata, stored and presented in *WG-log* form, to the clients in execution over the Net. Although in this paper we will not elaborate on site classification issues, it is worthwhile to remark that Schema Robots need not be all equal; for instance, a set of *domain schema robots* might form a distributed partitioned repository of all known schemata on the Net, while *category schema robots* might allow for a subject-related search. In this perspective, Robots can also be regarded as browsable similarity-based hypertexts of WG-log schemata, providing semantically rich information about Web sites. This hypertextual structure should not be regarded simply as "another technique" for schema identification; the availability of large, searchable schema repositories may prove a significant contribution to the much needed Web restructuring via resource indexing systems [Yam96]. All queries are formulated w.r.t. a *site schema*, supposed to be

known in advance to the client formulating the query. In order to formulate and execute the query, the following steps are performed:

1. Schema identification
2. Schema retrieval and query formulation
3. Instance retrieval
4. Presentation of results

In the following, we shall describe each step in some detail.

- **Schema identification**
  To get the best results, queries to the Web must be formulated on the basis of a known schema. In order to help the user in identifying a suitable schema w.r.t. the planned query, facilities for schema browsing and keyword-searching, together with a Thesaurus are available at Robot sites. The Thesaurus is mainly intended to provide a browsable controlled vocabulary to help the user in getting acquainted with the schema data dictionary. Note that, as the number of stored schemata will grow, automatic Thesaurus inizialization and update [Dam95] will become crucial for the Robots' performance.
- **Schema retrieval**
  With the help of the keyword-based information provided by the user, the Schema Robot's search engine identifies candidate schemata to be proposed to the user. With the help of a graphical editor, the user constructs appropriate queries for the schemata identified by the Robot; therewith, on reception of the schema-based user queries, the Robot consults an *instance cache* to discard sites that surely do not contain the desired information. This cache holds, for each schema, the values of selected attributes of some entities; aging mechanisms are required to ensure its consistency. The result of this second step is a list of Web sites together with valid references to them, i.e. the network addresses of the associated query manager processes.
- **Instance retrieval**
  The query is now broadcast to all query managers running on candidate Web sites. If the candidate Web site is based on an underlying database, the Query Manager process can provide an interface to the existing DBMS. In general, however, the Query Manager will maintain a copy of the WG-log schema and some indexing information linking navigational entities to (a list of) URLs. The resulting instance is computed at the target site; efficiency is critical at this moment, both in terms of time of computation of the resulting instance and of amount of gathered data that has to be trasmitted to the originating site.
- **Presentation of Results**
  This last step performs gathering and presentation of the result pages, on the basis of the data structures mapping HTML pages and the navigational part of the schema. This is a very interesting problem, since the amount of information contained in the resulting instance may be huge and the user must be presented with a synthesis of the available information, organized in such a way that he/she can choose to browse the resulting instance according to different perspectives. The synthetical result submitted to the user must enable him/her to formulate

one or more appropriate goals that select exactly the needed information from the computed instance, to be transported to the originating site.

Fig.2 summarizes the system architecture. This paper is organized as follows: Sec-

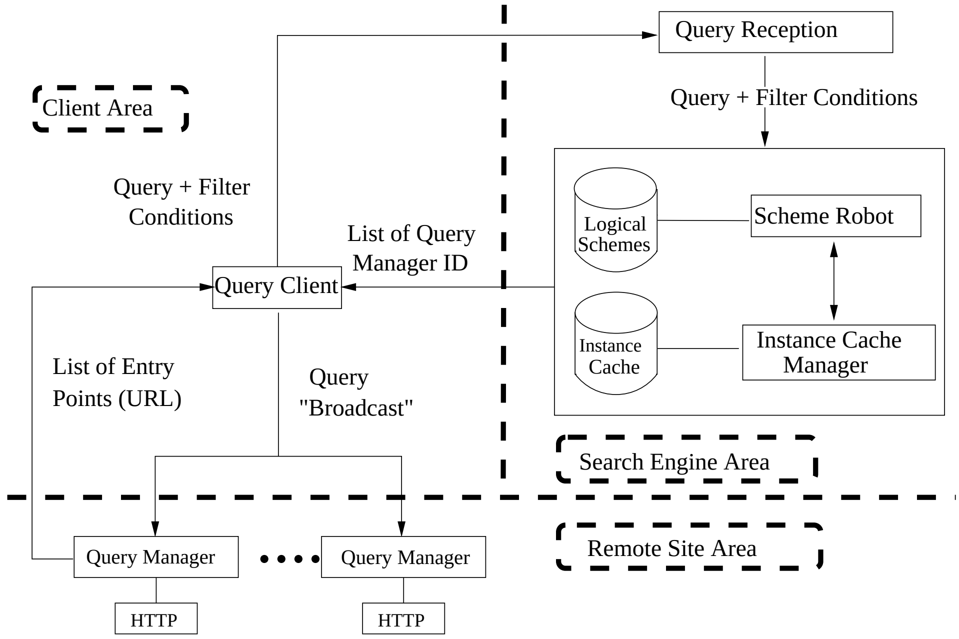

Fig. 2. System architecture

tions 2 and 3 describe the data model and language of WG-log; Section 4 presents the naive computation of WG-log queries, while Section 5 draws the conclusions and outlines future research on the same subject.

2 A Graph Description Language for Web sites: the Data Model

The use of design methodologies for hypermedia applications is currently well estabilished and widely employed to develop multimedia hypertextual applications. Besides allowing conventional or object-oriented design elements, such as E/R-like entities or OMT-like classes, nearly all modern hypermedia specification languages are associated to a presentation and navigation semantics, clearly indicating how entities are to be presented to the user. This approach is justified by the fact that no query support is generally offered to hypermedia products, whose fruition is based on free user navigation. In our opinion, an effective Description and Manipulation Language for Web sites should be able to complement the hypermedia model with database-like querying capabilities. In this Section we describe *WG-log*, a graph-oriented language supporting representation of both data model and structural entities. Graphs have indeed been an integral part of the database design process, and ever more so after the introduction of object-oriented data models; moreover they are naturally connected to graphical interfaces. WG-log has its formal basis in the graph-oriented language G-log [Par95]. G-log, being designed as an object database language, only includes two node types (representing abstract and concrete objects) and one link type (representing logical relationships, generally aggregations);

WG-log extends G-log by including some standard hypermedia design notations that allow for linking conceptual entities and relationships to navigational concepts such as WWW pages and links. As a result, WG-log schemata cleanly denote both *logical* and *structural* concepts. In addition, there is the possibility to specify some hypertext features like index pages or entry points, which are essentially related to the hypertext presentation style. In this section we informally present the data model and the language of WG-Log; more formal definitions and examples of G-Log can be found in [Par95,Par97]. References can also be found in [Gar95], where the HDM hypermedia design language is presented. In WG-Log, *directed labeled graphs* are used as the formalism to specify and represent Web site schemata, instances, views (also called *access structures*) , and queries. The *nodes* of the graphs stand for objects and the *edges* indicate relationships between objects. In WG-log schemata, instances and queries we distinguish four kinds of nodes:

- *slots* (also called *concrete nodes*), depicted as ellipses, indicate objects with a representable value; instances of slots are strings, texts, pictures, sound tracks, numbers, movies or movie frames (depending on the desired granularity of representation);
- *entities*, depicted as rectangles, indicate abstract objects such as monuments, professors, or cities; note that an abstract object can be chosen to correspond to one or more Web pages, possibly linked to each other in different ways: it is for the designer to decide which level of granularity the schema is meant to convey;
- *collections*, represented by a rectangle containing a set of horizontal lines, indicate collections or aggregates of objects, generally of the two types above; an instance of such a node is the index of all painters in a certain gallery (in this case we say that the collection is *homogeneous*) ;
- *entry points*, depicted as triangles, represent the unique page that gives access to a portion of the site (or to an alternative view of the site), for instance the site home page. To each entry point type corresponds only one node in the site instance. It is worth noticing that entry points and collection nodes can be used in queries for creating new *access structures* for providing alternative presentations of Web portions.

We also distinguish four kinds of graph edges:

- *structural edges*, representing navigational links between pages; such an edge may stand for the link between a collection node representing the painter index and the entity of type painter;
- *logical edges*, representing logical relationships between objects; such an edge might connect painters to their paintings. The presence of a logical relationship does not necessarily imply the presence of a navigational link between the two entities at the instance level;
- *double edges*, representing a navigational link coupled with a logical link; such an edge might connect a painter to his/her paintings, also indicating that there is a navigational link that allows paintings to be reached from their author;
- *Is_a edges*, representing the inheritance relationship between objects; such an edge might connect painters and artists (as their generalization). Note that Is_a edges are only allowed in a WG-log schema, while do not make any sense at the instance and query level.

As an example of use of these lexical elements, Fig. 4 shows the WG-log schema while Fig. 5 contains an istance, namely the experimental WWW site whose URL is `http://romeo.sci.univr.it/vrtour`. This WG-log description was easily obtained as a part of the design process of the site; an `HTML` page is shown in Fig. 6. It should be also noted that all entities in a schema are marked by a unique code that will be exploited during query execution.

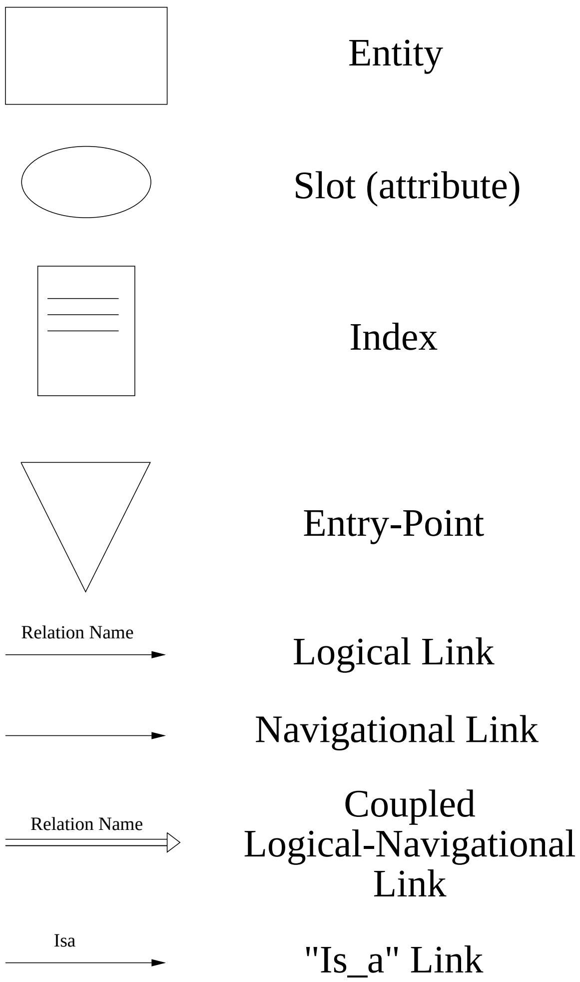

Fig. 3. WG-log lexicon

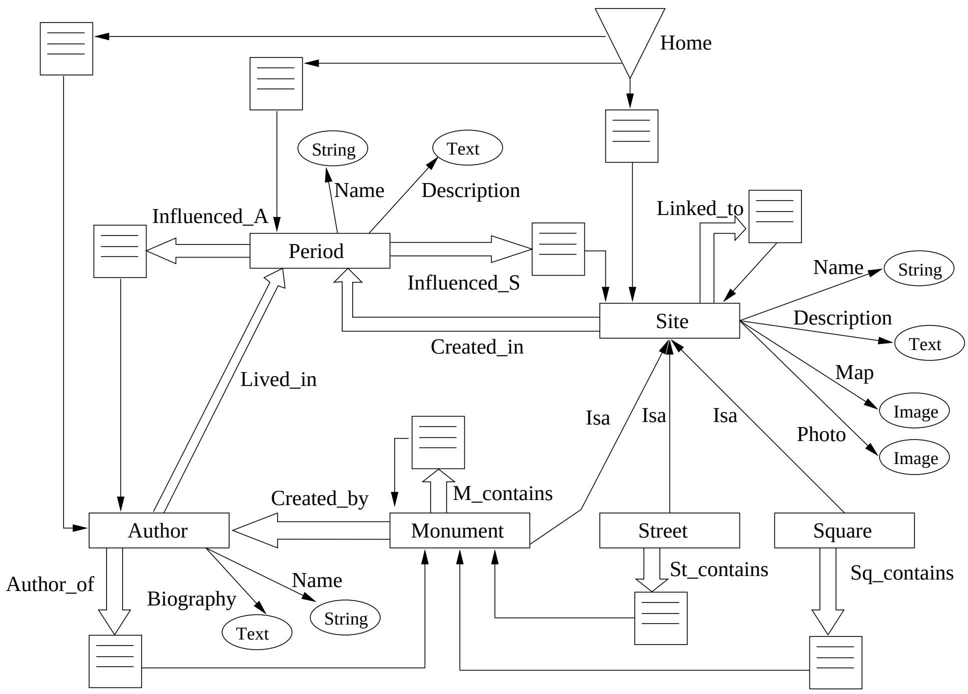

Fig. 4. The WG-log schema

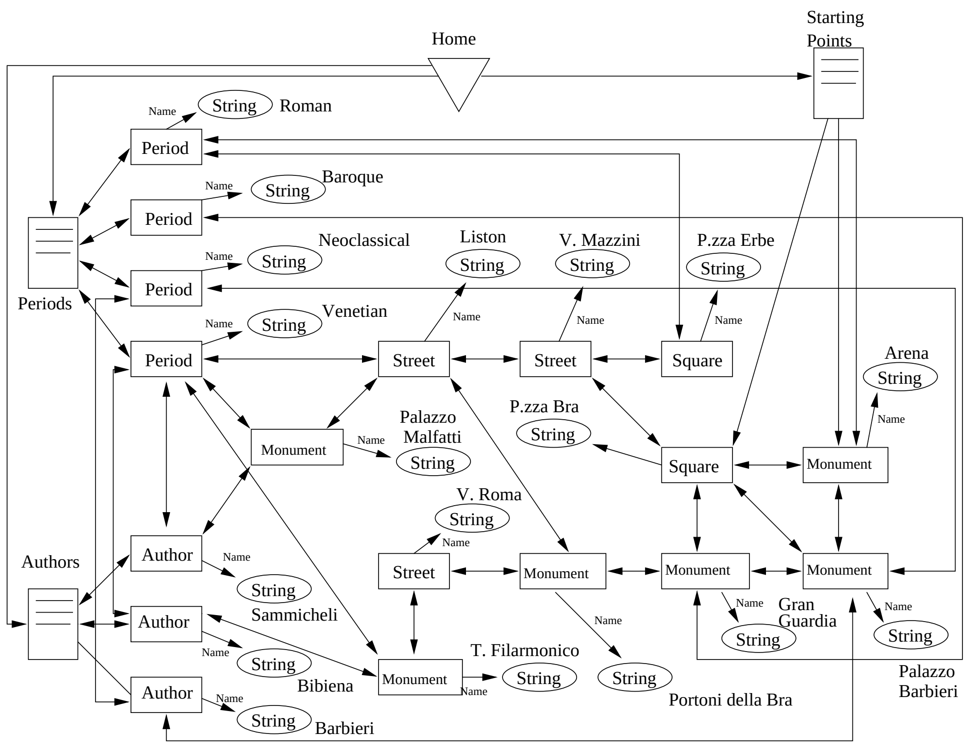

Fig. 5. The site instance

Fig. 6. A page of our trial site

2.1 WG-log schemata

A (site) *schema* contains information about the structure of the Web site. This includes the (types of) objects that are allowed in the Web site, how they can be related and what values they can take. Logical as well as navigational (structural) elements can be included into a site schema, thus allowing for flexibility in the choice of the level of detail. In fact, index and entry point nodes, mainly related to the hypertext presentation style, may be used by designers who want to emphasize the hypertextual aspects of their design, while such elements can be dispensed with by those designers who are more interested in the conceptual contents of the site schema. As far as node granularity is concerned, an entity can be chosen to represent one or more site pages. For instance in a University site an entity Professor in the schema might be mapped in the instance to one page or several pages for each professor; in the latter case, pages referring to the same professor may be differently linked to each other, the only constraint on different implementations being the availability at query time of the information about instance references to their schema entities. Formally, a schema contains the following elements:

- a set of *Entry Point labels* EP,
- a set SL of *concrete object (or Slot) Labels*,
- a set ENL of *ENtity Labels*, containing the special label *dummy*,
- a set COL of *COllection Labels*,
- a set LEL of *Logical Edge Labels*,
- one *Structural Edge Label* SEL (which in practice is omitted),
- a set DEL of *Double Edge Labels*,
- one *Is_a edge label* ISA,
- and a set $\mathcal{P}$ of *productions*[1].

The productions dictate the structure of WG-Log instances (which are the actual sites); the productions are triples representing the types of the edges in the instance graph. The first component of a production always belongs to $ENL - \{dummy\} \cup COL \cup EP$, since only non-concrete objects can be related to other objects. The second component is an edge label and the third component is an object label of any type. If the edge label is ISA, the two node labels must both belong to ENL. A Web site schema can be represented as a directed labelled graph, by taking all the objects as nodes and all the productions as edges. Note that two nodes might be connected by more than one edge. If multiple logical edges connect two nodes, they represent different relationships between those objects; the presence of a structural edge between two nodes represents the (possible) presence of a navigational direct link between the corresponding pages in the site instance: no two nodes, however, can be connected by more than one structural edge or by more than one ISA edge, since this would be meaningless. Finally, we assume a function π associating to each slot label a set of *constants*, which is its domain; for instance, the domain of a slot of type *image* might be the set of all *jpeg* files.

2.2 WG-log instances

A (Web site) *instance* over a schema $\mathcal{S}$ contains the actual information that is stored in the Web site pages. It is a directed labeled graph $I = (N, E)$. N is a set of labeled nodes. Each node represents an object whose type is specified by its label. The label $\lambda(n)$ of a node n of N belongs to $EP \cup SL \cup ENL - \{dummy\} \cup COL$[2].

[1] We require that the productions form a *set* because we do not allow duplicate productions in a schema.

[2] Note that no dummy node is allowed in schemata and instances

If $\lambda(n)$ is in EP, then n is an entry point node, and it is the only one with label $\lambda(n)$; if $\lambda(n)$ is in SL, then n is a concrete node (or a slot); if $\lambda(n)$ is in ENL, then n is an abstract object, that can coincide with one or more or a part of site page; otherwise n is a collection node, which means that it contains an aggregation of homogeneous or eterogeneous objects. If n is concrete, it has an additional label $print(n)$, called the *print label*, which must be a constant in $\pi(\lambda(n))$. E is a set of directed labeled edges. An edge e of E going from node n to n' is denoted (n, α, n'). α is the label of e and belongs to $LEL \cup \{SEL\} \cup DEL$[3]. The edges must also conform to the productions of the schema, so $(\lambda(n), \alpha, \lambda(n'))$ must belong to $\mathcal{P}$. Besides these edges, we also assume an implicit equality edge (a logical edge with an equality sign as label) going from each node of the instance to itself. Figure 5 contains an instance over the schema of Figure 4.

3 WG-log Rules and Programs

A WG-Log query is a (set of) graph(s) whose nodes can belong to all four node types used in schemata, and whose edges can be logical, double or structural. This allows for pure database-like and pure navigational queries; w.r.t. our experimental site, a query could select all the monuments which are the work of a given author as opposed to another listing all the pages in the site linked to more than five distincts nodes. It is interesting to remark that this technique also opens the interesting possibility of mixed queries, e.g. listing the works index page of all authors. In all three cases, in fact, the query results in a *transformation* performed on the instance. WG-Log *rules, programs* and *goals* can be used to deduce, query and restructure information from the information contained in the Web site pages. Rules are themselves graphs, which can be arranged in programs in such a way that new views (or perspectives) of (parts of) the Web site be available for the user. Like Horn clauses, rules in WG-Log represent implications. To distinguish the body of the rule from the head in the graph P representing the rule, the part of P that corresponds to the body is colored red, and the part that corresponds to the head is green. Since this paper is in black and white, we use thin lines for red nodes and edges and thick lines for green ones. Figure 7 contains a WG-Log rule over the Web site schema of Figure 4. It expresses the query: *Find all the monuments whose author is Bibiena.* Note the use of the **Result** collection node in the rule: it will build an *access structure* in the resulting instance. The application of a rule r

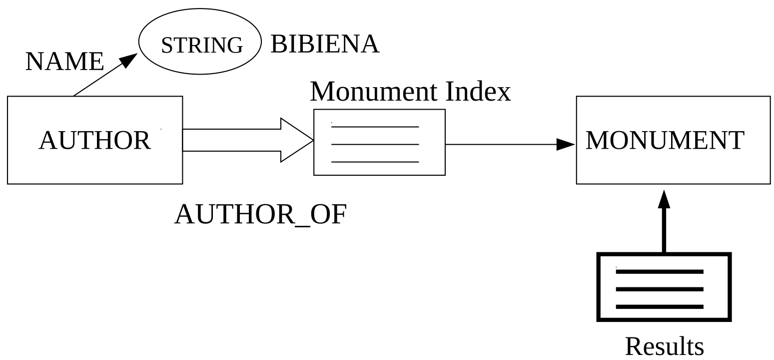

Fig. 7. A WG-Log solid rule.

[3] Note that no ISA edge is allowed in instances.

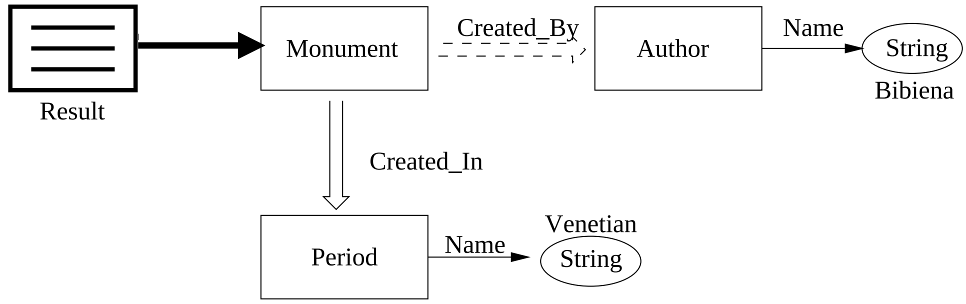

Fig. 8. A WG-Log rule involving negation.

to a site instance I produces a minimal superinstance of I that *satisfies* r. We say that an instance satisfies a rule if every matching of the red part of the rule in the instance can be extended to a matching of whole rule in the instance. The matchings of (parts of) rules in instances are called embeddings. For example, the instance I of Figure 5 does not satisfy the rule r of Figure 7. In fact, there is one possible embedding i of the red part of r in I, hence, the **monument** -nodes pertaining to Bibiena of I must be connected to a **Result**-node and this is not the case. Because I does not satisfy r, I is extended in a minimal way such that it satisfies r. In this case, the effect of rule application is that a **Result**-node is created and is linked to all the appropriate **monument**-nodes by an **SEL** -edge. Now the instance satisfies the rule, and no smaller superinstance of I does, so this is the result of the query specified by the rule. Note that the new instance, obtained from the query, contains a new *access structure* (the node RESULT and its links to all Bibiena's monuments), which allows the retrieval of the nodes in an alternative way, that was not possible in the initial instance. We will also see in the sequel that WG-log also allows the expression of *goals*, in order to filter out non-interesting information from the instance obtained from the query. Rules in WG-Log can also contain negation in the body; we use solid lines to represent positive information and dashed lines to represent negative information. So a WG-Log rule can contain three colors: red solid (RS), red dashed (RD), and green solid (GS). The rule of Figure 8 expresses the query *find all monuments of the venetian period whose author is not Bibiena*. The instance I of Figure 5 also does not satisfy this rule. The two possible embeddings of the RS part of r in I are applicable since they cannot be extended to embeddings of the RS and the RD part of r in I. Hence, the **monument**-nodes of I should be connected to a **Result**-node (to extend the found embeddings to embedding also of the GS part of r in I), and this is not the case in I.

3.1 WG-Log Rules and Goals

We now formally define what WG-Log rules are and when an instance satisfies such a rule, or a set of such rules. As in G-log, WG-Log rules are constructed from patterns. A *pattern* over a Web site schema is similar to an instance over that schema. There are three differences: 1) in a pattern equality edges may occur between different nodes, having the same label, 2) in a pattern concrete nodes may have no print label, and 3) a pattern may contain entity nodes with the *dummy* label, used to refer to a generic instance node. A pattern denotes a graph that has to be embedded in an instance, i.e. matched to a part of that instance. An equality edge between two different nodes indicates that they must be mapped to the same node of an instance. A *colored pattern* over a schema is a pattern of which every node and edge is assigned one of the colors RS, RD, or GS. If P is a colored pattern, we indicate by P_{RS} the red solid part of P, by $P_{RS,RD}$ the whole red part of P, and by $P_{RS,GS}$

the solid part of P. In a generic colored pattern, these parts will not be patterns themselves, since they can contain dangling edges. However, for P to be a WG-Log rule, we require that these subparts of P be patterns. Formally, a *WG-Log rule* r consists of two schemata $\mathcal{S}_1$ and $\mathcal{S}_2$, and a graph P. $\mathcal{S}_1$ is called the *source* (schema) of r. $\mathcal{S}_2$ is a superschema[4] of $\mathcal{S}_1$, and is referred to as the *target* (schema) of r. P must be a colored pattern over $\mathcal{S}_2$ such that P_{RS}, $P_{RS,RD}$ and $P_{RS,GS}$ are patterns over $\mathcal{S}_2$. Figure 7 contains the colored pattern P of a rule, that has as source the schema of Figure 4, and as target the same schema, to which a **Result**-node and an **in**-edge are added. To define when an instance satisfies a rule, we need the notion of embedding. An *embedding* i of a pattern $P = (N_P, E_P)$ in an instance $I = (N_I, E_I)$ is a total mapping $i : N_P \to N_I$, such that for every node n in N_P holds that

- either $\lambda(i(n)) = \lambda(n)$ or
- There is a production $(\lambda(n), ISA, \lambda(i(n)))$ in the target scheme;

moreover, if n has a print label, then $print(i(n)) = print(n)$. Also, if (n, α, n') is an edge in E_P, then $(i(n), \alpha, i(n'))$ must be an edge in E_I. Let $P = (N, E)$ be a subpattern of the pattern P' and let I be an instance. An embedding j of P' in I is an *extension* of an embedding i of P in I if $i = j|N$. An embedding i of P in I is *constrained* by P' if P' equals P or if there is no possible extension of i to an embedding of P' in I. We use the notion of "constrained" to express negation: an embedding is constrained by a pattern if it *cannot* be extended to an embedding of that pattern. Let r be a WG-Log rule with colored pattern P and target $\mathcal{S}_2$. An instance I over $\mathcal{S}_2$ *satisfies* r if every embedding P_{RS} in I that is constrained by $P_{RS,RD}$, can be extended to an embedding $P_{RS,GS}$ in I. As we informally mentioned before, the instance of Figure 5 does not satisfy the rule of Figure 8. The only embedding i of P_{RS} in I is constrained by $P_{RS,RD}$ (because it cannot be extended to an embedding of $P_{RS,RD}$ in I), and cannot be extended to an embedding of $P_{RS,GS}$ in I. To express complex queries in WG-Log, we can combine several rules that have the same source $\mathcal{S}_1$ and target $\mathcal{S}_2$ in one *WG-Log set*. So, a WG-Log set A is a finite set of WG-Log rules that work on the same schemata. $\mathcal{S}_1$ is called the *source* (schema) of A and $\mathcal{S}_2$ is its *target* (schema). The generalization of satisfaction to the case of WG-Log rule sets is straightforward. Let A be a WG-Log set with target $\mathcal{S}$. An instance I over $\mathcal{S}$ *satisfies* A if I satisfies every rule of A. In WG-Log is also possible to use goals. A *goal* over a schema $\mathcal{S}$ is a subschema of $\mathcal{S}$, and is used to select information of the Web site. Normally, a goal is combined with a query to remove uninteresting information from the resulting instance. The effect of applying a goal G over a schema S to an instance I over S is called I *restricted to* G (notation: $I|G$) and is the maximal subinstance of I that is an instance over G. The definition of satisfaction of a WG-Log set is easily extended to sets with goals. If A is a WG-Log set with target $\mathcal{S}_2$, then an instance I over G *satisfies* A *with goal* G if there exists an instance I' over $\mathcal{S}_2$ such that I' satisfies A and $I'|G = I$.

3.2 WG-Log Programs and Semantics

There is a strong connection between G-Log and first order predicate calculus. In [Par95] it is shown that for every formula on a binary many sorted first order language there is an effective procedure that transforms it into an "equivalent" set of G-Log rules and a goal; the converse is trivially true. Hence, G-Log can be seen as a graphical counterpart of logic. WG-log is only a syntactic variant of G-log, whose semantics we want to retain in order to keep its expressive power and representation

[4] Sub- and superschema, sub- and superinstance, and sub- and superpattern are defined with respect to set inclusion.

capability; thus the same correspondence holds for WG-log. Consider for instance the rule of Figure 8. This may be expressed in First Order Logic as follows:

$$\forall m \forall p \forall a \exists result \ : \{created\_in(m,p) \wedge period(p, \text{“}Venetian\text{”}) \wedge$$

$$\wedge name(a, \text{“}Bibiena\text{”}) \wedge \neg created\_by(m,a)\} \Rightarrow SEL(result, m)$$

Note that simpler languages like Datalog do not capture the whole expressive power of G-log: a Datalog rule is expressed in G-log by a simple rule containing red solid nodes and edges, and only one green edge. Thus, it is not possible to express the semantics of WG-log by translating it in Datalog. In the previous section we defined when an instance satisfies a WG-Log rule set; by examining the logical counterpart of WG-log, we get an intuition of the meaning of a WG-log rule; however, in order to use WG-Log as a query language we need to define its *effect*, i.e. the way it acts on instances to produce other instances; only in this way we will be able to isolate, among the infinity of instances that satisfy a certain rule, the one we choose as the rule's result. The *semantics* of a WG-Log set A with source $\mathcal{S}_1$ and target $\mathcal{S}_2$ is thus a binary relation over instances defined by:

$$\begin{aligned} Sem(A) = \{(I,J) \mid\ & 1.\ I \text{ is an instance over } \mathcal{S}_1 \text{ and } J \text{ is an instance over } \mathcal{S}_2,\\ & 2.\ J \text{ satisfies } A,\\ & 3.\ J|\mathcal{S}_1 = I,\\ & 4.\ \text{No subinstance of } J \text{ satisfies conditions 1. to 3.} \end{aligned}$$

Item 3 expresses the requirement that in WG-Log we only allow *queries*, and no updates. If a WG-Log rule contains a red dashed and a green solid part, then it can be satisfied either by adding the red dashed part to an instance or by adding the green solid part. Because of item 3, the source schema can be chosen is such a way that only one (or even none) of the two extensions is allowed. In this way the semantics of the rule also depends on its source schema. Item 4 expresses minimality. In general there will be more than one minimal result of applying a WG-Log set to an instance, which corresponds to the fact that WG-Log is non-deterministic and Sem is a relation and not a function. In WG-Log, it is allowed to sequence sets of rules. A *WG-Log program* P is a finite list of WG-Log sets such that the target schema of each set of P equals the source schema of the next set in the program. The source schema of the first set is the *source* (schema) of P, and the target schema of the last set is the *target* (schema) of P. The *semantics* $Sem(P)$ of a WG-Log program $P = \langle A_1, \ldots, A_n \rangle$ is the set of pairs of instances (I_1, I_{n+1}), such that there is a chain of instances $I_2, \ldots, I_n$ for which (I_j, I_{j+1}) belongs to $Sem(A_j)$, for all j. If a number of WG-Log rules are put in sequence instead of in one set, then, because minimization is applied after each rule, fewer minimal models are allowed. In fact, sequencing can be used to make a non-deterministic set of rules deterministic. Finally, a goal can be used in conjunction with a program. If $\mathcal{S}_2$ is the target of P and G is a goal over $\mathcal{S}_2$, then the *semantics* of P with *goal* G is: $Sem(P,G) = \{\ (I,J) \mid \exists (I,J') \in Sem(P)$ such that $J'|G = J\ \}$. There are 3 complexity levels of constructions to express queries in WG-Log: rules, sets and programs, which all three can be used in conjunction with a goal. This results in the six cases stated in the table of Figure 9. The use of all the three complexity levels guarantees that WG-log is computationally complete [Par95], i.e., it can produce any desired superinstance of a given instance. Normally, one or two rules, together with a goal, are sufficient to express most of the interesting queries we can pose to a Web site; however, some important queries do require the full language complexity. As an example, suppose we want to find all the pairs of nodes that are unreachable from each other by navigation; in other words, we want all the pairs that are not in the transitive closure of the relationship expressed by label SEL. An easy and

	without goal	with goal
rule	WG-Log rule	WG-Log rule + goal
set of rules	WG-Log set	WG-Log set + goal
sequence of sets of rules	WG-Log program	WG-Log program + goal

Fig. 9. The complexity levels of WG-Log queries.

natural way to solve this query is to compute the transitive closure *stc* of SEL, and then take the complement *ctc* of that relation. The WG-Log program of Figure 10 solves this problem. It is a sequence of two sets of rules. The first set, which consists of two rules, adds **stc**-edges (logical) between all nodes that are linked by a SEL-path. The second set has only one rule and takes the complement of the transitive closure by adding a **ctc**-edge if there is no **stc**-edge. Eventually, a goal can be added to select only the node pairs that are linked by the **ctc** (logical) relationship.

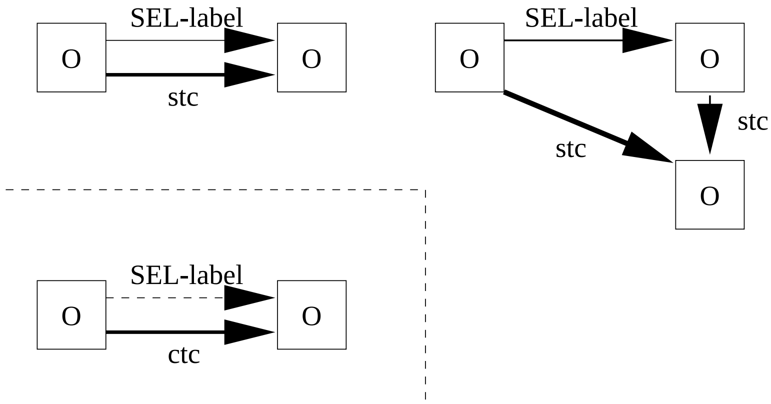

Fig. 10. The complement of the navigational transitive closure.

Another interesting query might ask all the nodes that are not reachable from a specific one, for instance the page of the artist *Bibiena*; in this case, the program must be complemented by the goal of Figure 11. Note typically, such goals can be used to optimize computation; however, this is outside the scope of this paper.

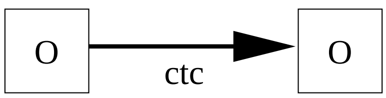

Fig. 11. A Goal on the complement of the navigational transitive closure.

4 Evaluation of WG-Log programs

In order to be able to express a rich set of queries, we have conceived WG-log as a language with a complex semantics; this gives rise to a computation algorithm that, in the general case, is very inefficient. However, in most cases the queries are expressed by only one or two rules, and possibly a goal which contributes to improving the efficiency of program computation. In the first subsection we present the computation algorithm in its most general form; later, we present an example of query computation, based on very simple data structures, which gives the flavour of the real complexity the system will have to tolerate without any improvements. In future work we will study appropriate data structures, and optimizations based on the goal structure, which will offer the possibility of increasing the efficiency of the naive approach presented here.

4.1 A general Computation Algorithm

We now present the *FastComp* algorithm, that computes the result of a generic WG-Log set by using a kind of backtracking fixpoint technique. Suppose we are given a set of rules $A = \{ r_1, \ldots, r_k \}$ with source $\mathcal{S}_1$ and target $\mathcal{S}_2$, and a finite instance I over $\mathcal{S}_1$. The procedure *FastComp* will try to extend I to an instance J, in such a way that J is finite and $(I, J) \in Sem(A)$. If this is impossible, it will print the message: "No solution". *FastComp* calls the function *Extend*, which recursively adds elements to J until J satisfies A, or until J cannot be extended anymore to satisfy A. In this last case, the function backtracks to points where it made a choice among a number of minimal extensions and continues with the next possible minimal choice. If the function backtracks to its first call, then there is no solution. In this sense, *FastComp* reminds the "backtracking fixpoint" procedure that computes stable models [Sac90].

Procedure *FastComp*$(I, A, \mathcal{S}_1, \mathcal{S}_2)$
$J = I$;
if $(Extend(J, A, \mathcal{S}_1, \mathcal{S}_2))$
{
  minimize(J);
  output(J);
}
else
  output("No solution");

Function *Extend*(**var** $J, A, \mathcal{S}_1, \mathcal{S}_2$))
for $(l = 1, \ldots, k)$ (* the rules are $r_1, \ldots r_k$ *)
  for (every embedding i of $P_{l,RS}$ in J)
    if (J does not satisfy r_l due to i)
    {
      $SetExt = \phi$;
      if $(P_{l,RD} \neq \phi)$
        for (every legal, minimal RD extension Ext of J)
          $SetExt = SetExt \cup \{ Ext \}$;
      for (every legal, minimal GS extension Ext of J)
        $SetExt = SetExt \cup \{ Ext \}$;
      while $(SetExt \neq \phi)$
      {
        select Ext from $SetExt$;
        add Ext to J;
        if $(Extend(J, A, \mathcal{S}_1, \mathcal{S}_2))$
          return (True);
        else
          remove Ext from J;
        $SetExt = SetExt \backslash \{ Ext \}$;

```
        }
        return (False);
    }
return (True);
```

The algorithm uses the notion of "legal, minimal extension" of an instance. By *legal*, we mean that the extension may only contain nodes and edges not belonging to $\mathcal{S}_1$. *Minimal* indicates that no subpart of the extension is already sufficient to make the embedding under consideration extendible. We denote by $FastComp(A)$ the set of all the pairs of instances (I, J), such that J is an output of the *FastComp* algorithm, for inputs I and A. In [Par97] we proved that the *FastComp* algorithm is sound and finitely complete:
$FastComp(A) = FSem(A)$, for every WG-Log set A. Note that the complexity

of *FastComp* is accounted for by the high expressive power of the language. The algorithm reduces to the standard fixpoint computation for those WG-Log programs that are the graphical counterpart of Datalog, i.e. sets of rules that consist of a red solid part and *one* green solid edge. Thus, efficiency of computation can easily be achieved for such programs, while optimization becomes more and more needed (and difficult) if more expressive queries are posed.

4.2 An example of Rule Evaluation

We shall now briefly comment on how *FastComp* can be used, at least in principle, to execute a WG-log query. Our sample query execution is based on three data structures:

- the (Typed) Adiacency Matrix *TAM* of the instance graph.
- the Instance Table *IT* linking schema entities and their instances
- the URL list *UL* linking instances to `HTML` pages or other network objects.

The role of the instance table is in many respects similar to that of the ontology introduced in [Luk97]. Each entry of the adiacency matrix lists the typed links (navigational, logical or coupled) connecting a pair of nodes. For the sake of conciseness, Fig. 12 does not show such a matrix, but a simpler, binary matrix where each entry i, j is 1 whenever instances i and j are linked by a navigational step, a logical relationships or both. Moreover, rows and columns pertaining to slots are not listed. Actually, we do not need to store slots in the TAM matrix; it is sufficient and surely less space-consuming to store them in auxiliary data structures pertaining to each single entity, in order to allow fast label matching. IT (Fig.13) associates the unique code of each schema entity to a list of unique numbers called *instance identifiers*; this allows the Query Manager to trace instances of schema-defined entities in the instance graph. Finally, the URL list associates each instance identifier to one or more HTTP URLs. This means that in our approach an istance of an entity is not necessarily a single page, though this will probably be the most frequent case. With respect to the sample query in Fig. 8, asking for the monuments of the Venetian period whose author is not Francesco Bibiena, we remark that the initial values of *FastComp* parameters are as follows: the whole instance of Fig. 5, the single rule of Fig. 8 and the source schema of Fig. 4. The target schema for this query can be easily deduced from the rule and will therefore be omitted.

To start with, *IT* is consulted to obtain the identifiers of the instance entities that match the rule entities. The following lists are obtained: Period = {1,2,3,4}, Monument = {8,10,13,16,17,18}. These lists are then used to extract from the TAM

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	0	1	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
3	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	1	1	0	1	0	1	0	1	0	0	0	0	0	1
5	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
7	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
8	1	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
10	1	0	0	0	1	0	1	0	0	1	0	0	0	0	1	0	0	0	0
11	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0
12	1	0	0	0	1	0	0	0	1	0	0	1	0	1	0	1	0	0	0
13	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0
14	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
15	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	1
16	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0
17	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
18	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0

Fig. 12. A simplified version of the adjacency matrix

Entity	**Instance**
A	0
B	1,2,3,4
C	5,6,7
D	E,F,G
E	8,10,13,16,17,18
F	9,11,12
G	14,15

Fig. 13. A sample instance table

the following possible adiacency information:

1	17	14
2	16	–
3	18	–
4	8	10

An equality test on the labels leaves only two possible embeddings for the red solid part of the rule: $4, 8$ and $4, 10$. Now, we are ready to follow the tree of recursive calls of Extend for our sample *FastComp* execution. Luckily, the recursion depth turns out to be only four in this case.

first call of Extend

1st for iteration; embedding 4,8 does not satisfy rule
Red dashed valid extensions: 5 (from TAM: the only instance of Venetian monument not related to Bibiena)
Green solid valid extensions: Result
$SetExt = \{5, Result\}$
1st while iteration
$J = J \cup \{5\}$

second call of Extend

1st for iteration; embedding 4,8 does not satisfy rule
Red dashed valid extensions: none
Green solid valid extensions: Result
$SetExt = \{Result\}$
1st while iteration
$J = J \cup \{Result\}$

3rd call of Extend

1st for iteration; embedding: 4,8 does satisfy rule
2nd for iteration; embedding: 4,10 does not satisfy rule
Red dashed valid extensions: none
Green solid valid extensions: Result
$SetExt = \{Result\}$
1st while iteration
$J = J \cup \{Result\}$

4th call of Extend

1st for iteration; embedding: 4,8 does satisfy rule
2nd for iteration; embedding: 4,10 does satisfy rule
Returns True (ends 4th call)

Returns True (ends 3rd call)

Returns True (ends 2nd call)

Returns True (ends 1st call)

The instance thus obtained is minimal, thus it is a solution to the query.

5 Concluding Remarks and Future Work

Experience with current WWW search engines has shown that the availability of a database-like schema is a prerequisite for any effective Web query mechanism. Though we are fully aware that the system described in this paper is only a preliminary step towards a satisfactory solution of the Web structuring and querying problem, we believe that its conceptual basis is sound and that its development may offer several interesting subjects for future research. For instance, a most important and promising issue is query execution itself, which must be both made more efficient and specialized to take into account the goal structure, schema information possibly available from a relational database underlying the site, and semantic properties of G-log, which enable the schema Robot to refuse *a priori* trivial or unsatisfiable queries. This is most needed since, as we have seen, *WG-log* retains the expressive power of the original *G-log* language: a carefully tuned execution mechanism is thus required to keep complexity in check (and to avoid "result explosion") when dealing with those queries that involve some kind of transitive closure. Another critical topic is the presentation of results: here not only efficiency considerations are involved, but also problems concerning the heterogeneous quality of the information stored: where text, images, sound tracks and similar pieces of information must be arranged to be shown to the user in a coherent and understandable way, architect's skills are needed, besides those of a Software designer. We plan to deal in the near future with querying *federate* Web sites. Namely, we plan to allow Web users to formulate queries on the basis of several site schemata at once, extending our query execution mechanism to take into account links between distinct Web sites. Finally, we plan to address at a later time more difficult problems like (semiautomatic) schema deduction on the basis of instance inspection; schema integration over unrelated sites; schema update at instance evolution; effective treatment of instance and schema graphs when these assume huge proportions.

Acknowledgments

The authors wish to thank Riccardo Torlone for putting forward the idea that gave rise to the present research. Thanks are also due to Franca Garzotto for many useful discussions on the subject. We also like to thank the students Michele Baldi and Fabio Insaccanebbia, from the University of Verona, for their contribution to the design and development of the Schema Robot as a part of their Master's Thesis, and the anonymous referees of this paper for their careful reading and useful comments on an earlier version.

References

[Alt97] Altavista, Inc."Altavista Search Index" `http://www.altavista.com`

[Atz97] Atzeni P., Mecca G., Merialdo P."Semistructured and Structured Data in the Web: Going Back and Forth". `http://www.research.att.com/šuciu/workshop-papers.html`

[Bal95] Balasubramanian V., Bang Min M., Joonhee M."Yoo: A Systematic Approach to Designing a WWW Application". CACM 38(8): 47-48 (1995)

[Dam95] Damiani E., Fugini M.G. " Automatic Thesaurus Construction Supporting Fuzzy Retrieval of Reusable Software". Proc. of ACM-SAC'95, Nashville 1995

[Dre97] Dreilinger, D. "SavySearch Home Page" `http://www.lycos.com`

[Fra97] Fraternali P., Paolini P. "Autoweb: Automatic Generation of Web Applications from Declarative Specifications", `http://www.ing.unico.it/Autoweb`

[Gar95] Garzotto, F., Mainetti, L., Paolini P. "Hypermedia Design, Analysis and Evaluation Issues" CACM 38(8) 74-86 (1995)

[Gys94] Gyssens M., Paredaens J., Van der Bussche J., Van Gucht D. " A Graph-oriented Object Database Model". IEEE Trans. on Knowledge and Data Eng., 6(4), (1994) 572:586

[Hu96] Hu J., Nicholson D., Mungall C., Hillyard A., Archibald A. "WebinTool: A Generic Web to Database Interface Building Tool". DEXA Workshop 1996: 285-290

[Isa95] Isakowitz T., Stohr Edward. A., Balasubramanian P. "RMM: A Methodology for Structured Hypermedia Design". Communications of the ACM, August 1995.

[Kon95] Konopnicki D., Shmueli O. "W3QL: A Query System for the World Wide Web", in Proc. of the 21th International Conf. on Very Large Databases, Zurich 1995 54-65

[Lak96] Lakshmanan L., Sadri F., Subramanian I. "A Declarative Language for Querying and Restructuring the WEB". RIDE-NDS IEEE Computer Soc. Press 1996 12-21

[Luc96] D.Lucarella, A.Zanzi (1996). A Visual Retrieval Environment for Hypermedia Information Systems. ACM Transactions on Information Systems 14(1), 3-29

[Luk97] Luke S., Spector L., Rager D., Hendler J. “Ontology-based Web Agents”, http://www.cs.umd.edu/projects/plus/SHOE

[Lyc97] Lycos Inc., “The Lycos Catalog of the Internet” http://www.lycos.com

[Lyc97-2] Lycos Inc. “Point” http://www.pointcom.com

[McB94] McBryan, O. A. “WWWW and GENVL: Tools for Taming the Web”, Proc. of the First Annual WWW Conference, Geneva 1994 58-67

[McK97] The McKinley Group, Inc., “Magellan Internet Guide” http://www.cs.colostate.edu/dreiling/smartform.html/

[Men96] A.O.Mendelzon, G.A. Mihaila, T. Milo, Querying the World Wide Web, in Proc. PIDA ’96, pages. 80-91, Dec. 1996.

[Par95] Paredaens, J., Peelman, P. Tanca, L. “G-log: A Declarative Graphical Query Language”. IEEE Trans. on Knowledge and Data Eng., vol.7 , 1995 436-453

[Par97] Paredaens, J., Peelman, P. Tanca, L.“Merging Graph-based and rule-based Computation: the language G-log”, accepted for publication of Data and Knowledge Engineering, 1997.

[Pin97] Pinkerton B. “Finding What People Want: Experiences with WebCrawler”, Proc. of the Second Annual WWW/Mosaic Conference

[Qua97] Quarterdeck Inc. “Web Compass Fact Sheet” http://www.arachnid.qdeck.com/qdeck/products/webcompass/

[Sac90] Sacca D, Zaniolo C. “Stable Models and Non-Determinism in Logic Programs with Negation.” Proc 1990 PODS

[Sel95] Selberg, E., Etzioni, O. “Multiservice Search and Comparison Using MetaCrawler”, Proc. of the Fourth International WWW Conference, 1995

[Yah97] Yahoo Inc. “Yahoo!” http://www.yahoo.com

[Yam96] Yamoto B., Dik Lun L. “WISE: A World Wide Web Resource Database System”. IEEE Trans. on Knowledge and Data Eng. 8(4) (1996) 548-554