Querying Multiple Databases Dynamically on the World Wide Web

J. Cardiff*, T. Catarci*, M. Passeri*, G. Santucci*,
*Dipartimento di Informatica e Sistemistica Università di Roma "La Sapienza", Rome, ITALY
*Institute of Technology Tallaght, Dublin, IRELAND

Abstract

The management and retrieval of Web data has recently received significant attention. Among the various approaches, systems have been proposed whose main goal is to provide a framework to integrate different and heterogeneous information sources into a common domain model. The Web-At-A-Glance (WAG) system falls into this category, its key characteristic being that instead of requiring an explicit description of the sources, it attempts to semi-automatically classify the information gathered from various sites based on the conceptual model of the domain of interest. The initial WAG prototype dealt exclusively with “standard” Web pages, which typically are HTML or XML documents and present non- or semi-structured information, and they vary widely in their means of presenting information to the user. In this paper, we present the extension of the WAG system to deal with form pages. In particular, we describe how the system semi-automatically first extracts a conceptual schema for the form page, and then fills and submits the form in consequence of a user query expressed on the domain conceptual schema.

1 Introduction

The management and retrieval of Web data, i.e., data residing on the Web and accessible through Web browsers, has recently achieved growing attention from the scientific community in general and the database community in particular because of its impact on the way information is stored, accessed, and distributed to almost everybody. Several results have been recently published mainly related with the tasks of modelling, extracting, and integrating information (see [5, 9, 17, 19] for survey papers), which constitute the research background of this paper. Our work is particularly related with the so-called global information management systems (GIMSs) [9], whose main goal is to provide a framework to integrate different and heterogeneous information sources into a common domain model. In the WAG (Web-At-a-Glance) project [6, 7], a database conceptual model (namely the Graph Model [14]) and its environment to interact with the user [8] are coupled with the Classic knowledge representation system [3]. This gives rise to a system which semi-automatically extracts information from various Web sites, integrates them in terms of database views, and allows the user to query such views.

WAG differs from the other GIMSs in two major aspects. First of all, WAG exploits the advantages of both knowledge representation and database techniques, since it follows a database approach at the external (user-oriented) level, and a knowledge representation approach at the internal level. Indeed, WAG allows the user to access the Web data by issuing a visual query on the conceptual schema of a database (thus, s/he never encounters the well-known problems of disorientation and cognitive overhead in finding the data of interest on the Web), but, in order to build such a database, the system relies on sophisticated knowledge representation techniques. Second, WAG, instead of requiring an explicit description of the sources, attempts to semi-automatically classify the information gathered from various sites based on the conceptual model of the domain of interest.

The initial WAG prototype dealt exclusively with “standard” Web pages, i.e., those having a static design in which users will navigate their way through a hierarchy of pages to locate information of interest. Standard pages typically are HTML or XML documents and present non- or semi-structured information, and they vary widely in their means of presenting information to the user. Significantly, both the underlying schema of the represented information, and the information itself are “blended” on pages of this type, and WAG’s task is to identify such a hidden schema, match it against a suitable domain schema, extract the information from the pages in form of database instances, and populate the corresponding database view (which can be queried subsequently) with such instances.
In this paper, we present the extension of the WAG system to deal with form pages. In particular, we describe how the system semi-automatically first extracts a conceptual schema from the form page, and then fills and submits the form in consequence of a user query expressed on the domain conceptual schema. This is an important extension to the system for several reasons. As interaction through forms is analogous to the query/result interaction in databases, the data returned by the Web database can be expected to be more structured and voluminous than a standard Web page, and therefore more likely to be of interest to a WAG user. This benefits both the organisation by way of reduced network traffic, and the user, who does not have to scan through long lists in order to locate the information of interest and we believe that organisations wishing to publish large volumes of information on the Internet will increasingly use forms interaction mechanisms. In addition, interaction with forms pages overcomes the data currency problem experienced with standard Web pages (once the local database is populated, the data immediately becomes historical), as WAG queries which access forms pages do so dynamically.

In dealing with form pages, our system also partially addresses the problem of query rewriting. In data integration, data warehousing, and query optimisation, the problem of query rewriting using views is receiving much attention [26]: Given a query \( Q \) and \( k \) queries \( Q_1 \ldots Q_k \) associated to the symbols \( q_1 \ldots q_k \), respectively, generate a new query \( Q' \) over the alphabet \( q_1 \ldots q_k \) such that, first interpreting each \( q_i \) as the result of \( Q_i \), and then evaluating \( Q' \) on the basis of such interpretation, provides the answer to \( Q \). Several papers investigate this problem for the case of conjunctive queries (with or without arithmetic comparisons) [20, 23], queries with aggregates [11, 24], recursive queries [15], disjunctive views [2, 16], non-recursive queries and views for semi-structured data [22], and queries expressed in Description Logics [4]. Rewriting techniques for query optimisation are described, for example, in [1, 10, 25], and in [18, 21] for the case of path queries in semi-structured data.

In our approach, the problem of query rewriting shows up when the system tries to use form pages when getting the answer to a user query. Indeed, a form page is defined as a conjunctive view (with arithmetic comparison) over the global schema. When the system processes a user query expressed over the global schema, the query rewriting algorithm is in charge to compute the appropriate query (i.e. to fill in the form fields) for the form page, to send the computed query to the page, and to suitably use the obtained result.

The paper is organised as follows: Section 2 describes the WAG Engine, and Section 3 describes the phase of conceptualisation of form pages, i.e., the discovery of the underlying conceptual schema and the filling and submitting of a form following a user query and schema population. Finally, Section 4 draws some conclusions.

2 The WAG Engine

The WAG Engine represents the core of the whole system and it allows us to realise a materialised view of the Web. Particularly it is able to produce a conceptual schema simply by analysing a site proposed by the user and to use that schema for populating the WAG Database with the data extracted from the site. The real abilities of the WAG Engine derive from the cooperation of different modules, as shown in Figure 1.

Figure 1: The Internal Structure of the WAG Engine

The most important modules composing the WAG Engine are as follows:

User Interface. This module represents the access point to the WAG system and it provides a series of graphic tools that allows the user to easily interact with the system. It is comprised of two modules: the Presentation Manager (PM) which manages the graphic environment, and the Interaction Manager (IM), which deals with the
different user activities carried out in the analysis phase, in which the user is required to validate the system choices, and may be asked to guide the conceptualisation process.

**Page Classifier.** This module analyses the structure of the pages and classifies them as belonging to certain predefined categories. The categories are differentiated considering the contributions they can give to the subsequent conceptualisation phase, e.g., the home page of an individual becomes an instance of some class, while the index page of an organisation is transformed into a conceptual subschema; a page containing a form may provide a sketch of the underlying relational database, etc.

**Robor.** This module unloads HTML pages from a particular site and stores them, by using a specific criteria, into the system. This activity is necessary to guarantee an efficient execution of the subsequent analysis methods.

**Conceptualiser.** The conceptualiser is the core of the system. It receives inputs from the modules above (including the user suggestions), builds a conceptual schema from the HTML pages of a certain site, and then populates the schema with different kinds of instances (e.g., URL, tuples, multimedia objects, etc.) extracted from the site. The conceptualiser relies on a conceptual data model, namely the Graph Model [12, 14], which provides a conceptual view of the information of interest and acts as the formal basis for the various visual representations and interaction modalities of the WAG query environment. The Conceptualiser is in charge of merging the various site schemas exploiting additional pieces of information coming from the knowledge base, and producing a partial integrated schema of the subject of interest. The basic ideas for the schema integration process (even if in a different context) have been already presented in [13].

### 3 Conceptualisation of Forms Pages

Since conceptualisation for standard pages is completed prior to a user issuing a query, and conceptualisation for form pages is performed as part of executing a user query, we refer to these as static and dynamic conceptualisation respectively. Conceptualisation of standard pages has been described in detail in [6], and we will focus in this paper on the dynamic conceptualisation.

When a page is identified by the Page Classifier as being a Form Page, its fields will be used either to provide information to the website administrator, or to identify the characteristics of the information requested by the user. Only the latter case is of interest to us, and forms of this type provide a method of allowing a user to query objects belonging to a specific class. It is essentially an interface to allow the user to pose a query to a database through a Web front-end. Since each form field specifies a restriction on the set of values to be returned, it is reasonable to assume that the form represents the schema of a class, and therefore the initial objective of the Dynamic Conceptualisation process is to identify the structure of the information that would be returned, were the form fields to be completed and the form submitted. It must then determine how to automatically complete a form, for any given user query, and marshall the returned information into the Result Database.

To achieve this goal, there are three tasks to be accomplished.

1. **Form Structure Discovery** associates a form with a view over the domain schema. Note that we assume that the view may be expressed as a conjunctive query or at most as the union of conjunctive queries (see below). In addition, this phase filters out "uninteresting" forms which only serve to provide the administrator of the remote site with information.

2. **Form Activation** is the process of automatically completing the form fields and submitting the form, once the WAG user has issued a query, and

3. **Schema Population** extracts the relevant data from the results page into a relational database.

To illustrate the procedure of dynamic conceptualisation, we use a Real Estate example. The fragment of the Domain Schema which is relevant for this purpose is shown in Figure 2. Note that we assume the Domain Schema and associated information to be available in the knowledge base. However, we are presently working on the WAG conceptualization modality called "off-line" (see [6]) in which the existence of a domain schema is not required.

![Figure 1: Fragment of the Domain Schema](image-url)
Several sites provide forms based interfaces to Real Estate databases on the Web. In our example, we use a typical site\(^1\), which allows a user to search for properties matching certain criteria. The form page is shown in Figure 2.

![Form Page Screenshot](http://www.homescout.com)

**Figure 2: Example Form Page**

3.1 Form Structure Discovery

The Form Structure Discovery phase is initiated when the Page Classifier categorises a particular page as being a form. Its task is to deduce the structure of the information that will be returned to the user on the results page/s, should the form be completed and submitted. Typically, if the information is of interest to the user (rather than being of interest to the site administrators, for example), the form fields represent attributes of one or more classes and therefore we take the approach that a view schema can be extracted from the form page. Form Structure Discovery is subdivided into three steps: Class name analysis, Attribute analysis, and Domain analysis. Class name and attribute analysis collectively identify the view represented by the form. On completion of these phases, there is a user verification step which confirms their results, and augments them if necessary. The final step is Domain Analysis, which produces a mapping between the attribute domains from the class in the domain schema to the expected input value types on the form fields.

3.1.1 Class Name Analysis

Class name analysis attempts to identify the name of the primary class\(^2\) represented by the form page. It is based on the premise that the name of the class, or a string which would suggest the name of the class, is likely to appear towards the top of the form page, and in any case before the occurrence of the first form field. We therefore analyse this section of the form page, and identify possible class names by a matching process which compares class names (and their synonyms) from the knowledge base of conceptual schemas with keywords and phrases from the form page.

This step yields the candidate primary classes [Real Estate, Home, HomeScout]. Each of these is matched against the set of class names from the domain schema (see Figure 3) and a match is found with the Property class (identified as being a synonym for Real Estate).

![Form Diagram](http://www.homescout.com)

**Figure 2: View Representing the Form**

3.1.2 Attribute Analysis

The purpose of attribute analysis is to associate each form field name with a template restriction for queries posed to the domain schema (i.e., an attribute name and an operation). The majority of the attributes should belong to one of the candidate primary classes, and the other attributes should come from classes related with it. The matching algorithm in this case parses the HTML code for each form field and extracts its internal name, and also the text displayed beside the field on the screen. These values are then used to identify which attribute the field name may refer to.

---

\(^1\) [http://www.homescout.com](http://www.homescout.com)

\(^2\) We call "primary class" the class of the domain schema which is semantically closer to the form structure. For instance, the majority of the form attributes should come from such a primary class.
Collectively, we refer to the class and attribute name analysis steps as Class Discovery. There are a number of outcomes to Class Discovery.

- Firstly, a proposed mapping between a class name from a class in the knowledge base of conceptual schemas to the form, and a proposed mapping for each form field of the form to an attribute of the domain schema. There may also be a number of form fields for which no matching attribute could be identified.

- Secondly, it is possible that two or more candidate classes were identified by the class name analysis. The attribute analysis is repeated for each candidate class, and the lack of a match of attributes with form field would suggest that a candidate class could be discarded. However, a match could still be identified, causing the Class Discovery to suggest that the form primary class is a generalisation of two or more classes. In this case the view corresponding to the form is the union of conjunctive queries.

- Thirdly, it is possible that no candidate class names may emerge from class name analysis. In this case, the attribute analysis is applied against every set of attributes of each class from each conceptual schema in the knowledge base. The output may be zero or more proposed class to form mappings. In the case of no proposed mappings emerging, it is possible that the form represents a class not currently present in the domain schema, in which case a proposed structure of the class is presented.

As there is only one candidate class, Property, a single iteration of attribute analysis is performed. The following form field to attribute mappings are identified:

<table>
<thead>
<tr>
<th>Form field</th>
<th>Attribute</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>City</td>
<td>City</td>
<td>=</td>
</tr>
<tr>
<td>State/Province</td>
<td>State</td>
<td>=</td>
</tr>
<tr>
<td>Include cities within</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Property Type(s)</td>
<td>Type</td>
<td>=</td>
</tr>
<tr>
<td>Maximum Price</td>
<td>Price</td>
<td>&lt;</td>
</tr>
<tr>
<td>Minimum Price</td>
<td>Price</td>
<td>&gt;</td>
</tr>
<tr>
<td>Minimum Bedrooms</td>
<td>Beds</td>
<td>=</td>
</tr>
<tr>
<td>-</td>
<td>Country</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Garage</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>No Baths</td>
<td></td>
</tr>
</tbody>
</table>

Due to the clear naming schema used on the site (which, from our research, we believe can be expected in general), most mappings are performed correctly. These mappings must be verified by the user and corrected where necessary. The default operation is equality and if the form query is applying another operation to any field value, it must be possible to infer this from the form's source code. In this case, there are two restrictions corresponding to the attribute Price from the domain schema, and in each case, the inequality is inferred from the respective label.

An equally important task of this analysis is to identify classes and forms pages that can be classified as "uninteresting" to the user. By this we mean that they will not yield any useful information to the user, should they be completed. Forms of this nature include input forms, through which the host site may try to extract information about the user such as name, address, city etc., to be used for their own purposes. This is efficiently done by including in the Knowledge Base the notion of a "discardable" candidate class. Such forms pages are easily identifiable by their form field names.

3.1.3 Domain Analysis

Once the schema has been verified, it is necessary to establish a correspondence between the domains of each attribute in the class from the domain schema, and the range of valid input values to be entered on the form. This information is required in order to be able to automatically fill out and submit the form. There are two categories we need to consider, firstly, cases in which a correspondence exists between the attribute and a form field, and secondly, where there is no correspondence.

Case 1: Correspondence Exists

There are two main methods of entering data on a form. A field may be completed using free text, in a text box, or the user may select from a number of options, using pull down menus or check boxes. In the former case, the domain is defined to be the same as that of the corresponding attribute. This may be altered by the user during verification based on, for example, instructions on the form page indicating the format the input should take. In cases where the user is constrained in the values they can enter through menus or check boxes, this amounts to a definition of the domain of the form field. In these cases, the system extracts the individual domain values from the HTML coding and attempts to perform a matching between each value of the domain of the attribute, and the form field domain. The domain matching algorithm is again based on inexact pattern matching and heuristics. A complication in aligning the domains is that the degree of precision may differ, e.g. the domain of attribute "Price" may be the Natural Numbers, indicating a dollar amount, whereas the domain of the corresponding form field may be a discrete set of price ranges. This is a variant of the data integration problem that occurs in multidatabase integration, and is dealt with in detail in [13]. The approach we take is summarised as follows:
If the form field domain is more precise than the attribute domain, we map the latter to the corresponding subset of the former’s domain, e.g. if an attribute domain is a set of cities, and the domain of the corresponding form field is a set of boroughs within cities, then each city is mapped to the set of boroughs comprising the city.

- If the form field domain is less precise than the attribute domain (which will more often be the case), each attribute domain value is mapped to a corresponding form field domain value in such a way that a user query will return at least all results satisfying the query, e.g. a house price of $210,000 would be mapped to a price range of $200,000 - $250,000.

- In cases where the domains of the attribute is numeric, a mapping function converts the value in terms of the user’s domain to those expected by the form field. This will typically be necessary for currency, area or distance values. Unit mismatches are identified by the domain matching algorithm using a combination of recognised symbols or abbreviations (e.g. $, Km) and knowledge of the country in which the Web site is based (identifiable by the URL).

**Case 2: No Correspondence Exists**

Where there is no corresponding form field for an attribute, either the attribute may simply not be represented as a searchable field on the form, or the attribute’s value may be implicitly represented. The former case can be ignored, but any implicitly completed fields must be identified, particularly if the attribute is classified as being “significant”. A significant attribute is one which is likely to be important in restricting the query result. In the Property class, attributes Country, Type, and possibly State are significant. For each significant attribute having no corresponding form field, we analyse the form, attempting to identify implicit values. The techniques we use to perform this task include analysing the URL to identify the country, and comparing domain values from significant attributes with text in the form header.

### 3.1.4 Form Query Expressiveness

The purpose of this phase is to characterise the kind of queries which may be expressed by using the fixed query patterns provided by the form. Establishing the expressiveness of a form query is a difficult task since forms are designed by a variety of informal and ad hoc approaches. Assumptions are made by each designer about the ability of an individual user to interpret the form in a manner consistent with its usage, and equivalent entries on two independently designed forms may have very different interpretations. Nevertheless because the common goal of all sites providing forms interfaces is to minimise the amount of information returned to the user (a goal shared by the user), we can identify certain characteristics of form queries that allow us to make general statements about their expressiveness.

The most common case is where the form is a conjunct of selection conditions over combinations of fields. The simplest condition is equality ($A = a$, where $A$ is a field name and $a$ is a value of the attribute domain). More complex conditions involve predefined operators such as range operators and inequality operator. Also, single conditions can be composed by using AND, OR, and NOT boolean operators. It is also worth noting that forms accepting blank fields (meaning that the query is satisfied by any value) implicitly provides the user with the SQL “ANY” operator. Other special operators are string operators, including wild cards, string containment, pattern match, etc. Finally, using as input to a query the output of another one (on the same form) one could express union, self-join (sort of recursion), but cannot express generic joins. We establish the expressiveness of the form query by considering it as a set of pairs $<L_i, V_i>$, $1 \leq i \leq n$, where $L_i$ is a tag that can be mapped to an attribute $A_i$, and $V_i$ is a parameter to which $A_i$ may be instantiated when the form is submitted. Each $<A_i, V_i>$ pair is therefore a restriction in the form query, and in the absence of further information, the operator implicit in each restriction is equality. Furthermore, in accordance with the stated goal of minimising result size, the restrictions form a conjunct.

Operators which involve operations other than equality occur commonly in form queries (e.g. price ranges). However the user seldom has the option to select the operator itself, and there must therefore be some means by which the use of the alternative operator is communicated to the user. This is typically achieved by a label which may either be part of the form tag, or located close to it. We can use a series of heuristics based on keywords and pattern matching to determine when a non equality operator is to be applied. For example to allow the user to enter a value range the following are common patterns in the form:

- the name and data type of the attribute (e.g. must be integer or real; certain attribute names such as price, value)
- the existence of two input fields for one attribute suggests a range
- keywords can determine the precise operator (e.g. maximum, from...to, greater than)

The inequality operator is used less frequently but can be identified using similar techniques. Keywords to identify
inequality include NOT, "excludes", etc. Furthermore in cases where inequality is available, it is often selected as an option by the user, and therefore the presence of a check box or similar boolean input suggests the possibility of this operator. Where the form offers the possibility to perform string comparisons, the tag must contain an instruction to the user as to how such comparisons are to be constructed. Typically, the pattern entered by the user is to be a subset of the text to be matched (and can be identified as such by the corresponding text), but some forms offer the user more precision, using special characters for wildcards. In the latter case, instructions must be explicitly stated to the user about the special characters and their interpretations, which can be deciphered from an analysis of the text.

In the event that the user enters no value for a field, there are two possible scenarios:

- All values are to be returned, i.e. the restriction is to be removed from the form query;
- If the field is designated as being a mandatory field, then submission of the form query with no value entered would result in an error message (i.e. an empty result).

Consequently for forms where no value is entered on any mandatory field, the result is computed as null. Determining which interpretation applies can be determined by an analysis of the form source code identifying any headings which indicate that certain fields are mandatory, and by examining the results and messages returned by submitting the form.

3.2 Form Activation

When the user submits a query, the WAG engine needs to consult both the local database containing results downloaded in offline mode from static HTML pages, and also each form page which may contain relevant information. The Knowledge Base contains for each class a set of associated form pages, and also the mappings established during the Form Structure Discovery phase. Any forms having implicit values for any form field may be excluded at this stage if it is semantically impossible for any result to satisfy both the restriction in the user’s query, and the restriction in the implicit value (e.g. a user could specify “country ≠ USA” in the query, which would exclude a form where the implicit value of country is Italy). In order to express the query by using all forms which may produce results, it is necessary to automatically complete them in the appropriate way. In order to do this the following algorithm is performed for each form:

**INPUT:** Query $Q = (A_1 \ op_1 a_1) \wedge \ldots \wedge (A_n \ op_n a_n)$

Uninstantiated form $F = (\langle a, \ V \rangle, \ldots, \langle a, \ V \rangle)$

A partition of $F$ into REQUIRED-FIELDS and OPTIONAL-FIELDS

**OUTPUT:** An instantiation of $F$

Let $A$ be the attributes specified in $Q$, i.e. $A_1, \ldots, A_n$

For each form field $F_i = \langle a, \ V \rangle$, $1 \leq i \leq n$

Let $A_i$ be the attribute corresponding to $F_i$

If $F_i \in$ REQUIRED-FIELDS \ $A_i \notin A$

Then $\exists$ a default value $d_i$ for $A_i$ in KB

Then $V_i \gets d_i$

Else $Q \gets \emptyset$; EXIT /* $A_i$ was not instantiated*/

If $\exists A_i \in A$

Then $V_i \gets f(a_i)$, where $f$ is a user defined function as described in section 3.1

Else If $\exists$ a default value $d_i$ for $A_i$ in KB

Then $V_i \gets d_i$

Else $V_i \gets \emptyset$

4 Conclusions and Future Work

It is our opinion that, in order to exploit the enormous amount of disparate information contained on the Web, there is the need to develop effective ways to retrieve, extract and model the information of interest. Our proposal, namely the WAG system, aims to address these issues by a suitable integration of ideas and techniques coming from both the database and the knowledge representation areas. The basic idea of WAG is to allow the user to access the Web data by simply issuing a visual query on the conceptual schema of a database, and to rely on sophisticated knowledge representation techniques in order to build such a database. Furthermore, instead of requiring an explicit description of the sources, WAG attempts to semi-automatically classify the information gathered from various sites based on the conceptual model of the domain of interest.

While the initial WAG system dealt exclusively with static Web pages, we have described in this paper the extension of the WAG system to deal with form pages. In particular, we described how the system semi-automatically first extracts a conceptual schema for the form page, and then fills and submits the form in consequence of a user query expressed on the domain conceptual schema. This is an important extension to the system for several reasons. As interaction through forms is analogous to the query/result interaction in databases, the data returned by the Web database can be expected to be more structured and voluminous than a standard Web page, and therefore more likely to be of interest to a WAG user. In addition, interaction with forms pages
overcomes the data currency problem experienced with standard Web pages (once the local database is populated, the data immediately becomes historical), as WAG queries which access forms pages do so dynamically.

The present prototype has been tested on the real estate domain, as described in the running example in this paper. WAG was been activated to analyse a number of real estate Web databases, through which the user interacted with forms pages. We got encouraging results, showing that in the domain under consideration the system can establish a correct mapping from the form fields to the domain schema. Clearly, the mappings produced during conceptualisation of the forms cannot be relied on without manual verification, but having a suggested mapping scheme between the form and the domain schema can greatly assist in the process.

The WAG system is currently being extended in a number of directions. We are currently working on the functionality of the system to deal more fully with the problem of query rewriting, and to provide the capabilities of performing joins across data extracted from disparate sites. Furthermore, the current version deals only with HTML forms – since these are by far the most frequently occurring – and we are extending its capabilities to process XML forms.

References