19 USING WG-LOG SCHEMATA TO REPRESENT SEMISTRUCTURED DATA

E. Damiani, B. Oliboni, L. Tanca, D. Veronese

Università di Verona

edamiani@crema.unimi.it barbara@romeo.sci.univr.it

tanca@sci.univr.it davvero@romeo.sci.univr.it

Abstract: In this paper we discuss the possibility to represent synthetically semistructured information via a loose notion of schema: we say that data are *semistructured* when, although some structure is present, it is not as strict, regular, or complete as the one required by the traditional database management systems. Our proposal is based on WG-Log, a graph based language for the representation of WWW site information. We show how information encoded in a typical semistructured information model, as OEM, can be represented and queried by means of the WG-Log language, and how the TSIMMIS and WG-Log Web Query System can be integrated to allow site content exploration and exploitation by means of WG-Log.

19.1 INTRODUCTION

We say that data are *semistructured* when, although some structure is present, it is not as strict, regular, or complete as the one required by the traditional database management systems (see [1] for a survey on semistructured data).

* This work has been partially supported by the INTERDATA project from the Italian Ministry of University and Scientific Research, 1997.

* The WG-Log/OEM experience has been also described in a paper published in SEBD 1998 under the title "Using WG-Log to represent semistructured data: the example of OEM".

Information is semistructured also when the structure of data varies w.r.t. time, rather than w.r.t. space: even if data is fairly well structured, such structure may evolve rapidly.

Traditional database systems force all data to adhere to an explicitly specified, rigid schema, which is far less dynamic than the data itself. For many new applications involving large amounts of semistructured data there can be two significant drawbacks to the database approach:

- Data may be irregular and thus not conform to a rigid schema. In relational systems, the presence of irregular data forces the use of null values, whose appropriate handling is still a research issue in the relational database community. While complex types, object identity and inheritance in object-oriented databases clearly enable more flexibility, it can still be difficult to design an appropriate object-oriented schema to accommodate irregular data.

- It may be difficult to decide in advance on a single, correct schema. If the structure of data is in constant evolution, data element types may change, or data not conforming to the previous structure may be added.

As a first example of semistructured information we can consider data integrated from multiple, heterogeneous data sources. When data is integrated in a naïve fashion from several heterogeneous sources, there may be discrepancies among the various data representations: some information may be missing in some sources, an attribute may be single-valued in one source and multi-valued in another, or the same entity may be represented by different types in different sources. Considerable effort is typically spent to ensure that the integrated data are well structured and conforms to a single, uniform schema. Additional effort is required if one or more information sources change, or when new sources are added.

As a second example, consider data stored on the World Wide Web. At a typical Web site, data is varied and irregular, and the overall structure of the site changes often. Today, very few Web sites store all their available information in a database system; it is clear, however, that Web users could take advantage of database support, e.g., by having the ability to pose queries involving logical data relationships (which usually are known by site's creators but not made explicit).

When querying semistructured data, one can't expect the user to be fully aware of their complete structure, especially if the structure evolves dynamically. Thus, it is important not to require full knowledge of the structure to express meaningful queries.

In most schema-based systems these features cause frequent schema modifications, and for this reason a synthetic representation of the data cannot be very rigid, but should be adaptable to changes. Because of these limitations, many applications involving semistructured data are forgoing the use of a database management system, despite the fact that many strengths of a DBMS (ad-hoc queries, efficient access, concurrency control, crash recovery,

security, etc.) would be very useful to those projects. Moreover, database schemata have proved to be a good mean to convey information about data semantics, and this is the reason why some kind of schema is essential in the process of query formulation. Thus, we should do as much as possible to keep this heritage from the database world.

In this paper we discuss the possibility to represent synthetically semistructured information via a loose notion of schema. As an example of application of a schema-based approach, we refer to the WG-Log project [6], a proposal that involves the use of a graph-based schema and query language for WWW information: the WG-Log proposal includes also an architecture for a Web Query System (WQS)[1].

We show how information encoded in a typical semistructured information model, as OEM [16], can be represented and queried by means of the WG-Log language, and how the TSIMMIS and WG-Log Web Query Systems can be integrated to allow site content exploration and exploitation by means of WG-Log.

In particular, we are referring to a scenario in which information about Web sites' contents represented in OEM (Object Exchange Model) is automatically translated to a WG-Log schema. Then a WG-Log user formulates a query in WG-Log; the system translates such query into LOREL [4] (an OQL-like language which queries OEM-represented information) and presents the answer to the WG-Log user. Thus, WG-Log schemata of OEM-represented sites can be kept in a schema repository, together with the other WG-Log schemata, to be queried in a transparent way by any WG-Log user.

The paper is organized as follows: next Section concerns the major research streams in this area, focusing on the TSIMMIS project [7]. After, recalling the WG-Log model and language, we present the translation of OEM representations into WG-Log schemata and the mapping of WG-Log queries on these schemata to the LOREL language. Then we describe the architecture of the WG-Log Web Query System and its integration with TSIMMIS. Finally we draw the conclusions.

19.2 RELATED WORK

To date, two main research streams can be recognized on the querying and management of WWW data:

- *program oriented:* the focus of these approaches is on the generation of *wrappers*, programs that facilitate database-like querying of semi-structured data retrieved directly from sources. The wrapper accepts queries (in a standard query language) about information in the source, fetches relevant information (hypertextual documents in the case of the WWW) and returns the results.

[1]By Web Query System we mean a set of integrated tools for the efficient and effective retrieval of information in the World Wide Web.

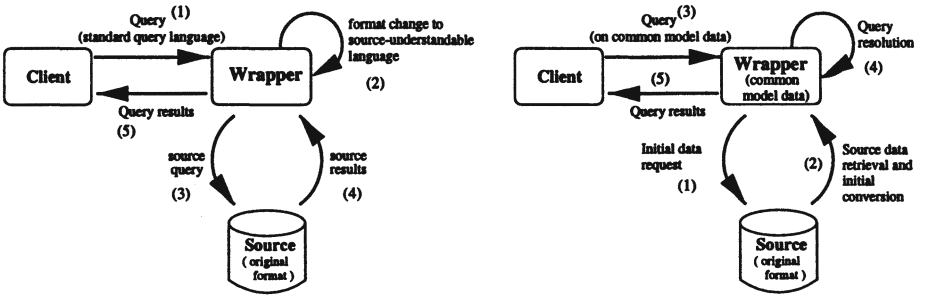

Figure 19.1: The program based approach (left) and the data-model approach (right)

An example of this approach can be found in the **SIMS** project [2], where wrappers are generated in order to provide database-like querying for semistructured WWW sources.

Another example is the **Infosleuth** project [5], developed at MCC, which uses wrappers to translate standard SQL queries to Information Retrieval expressions, with the support of a classical Entity-Relationship schema which represents the semantics of the different data sources indipendently of their actual data representations. Note that the data format remains the one internal of each data source: the E-R schema is only a conceptual representation to ease query formulation.

- *data-model oriented:* here the grounding idea is to convert semi-structured data to a representation based on a common information model, which is stored into a repository in order to be later queried via a (more or less) standard query language; this repository is kept indipendently of the original semistructured data. Queries are initially formulated over information coded into the common model; wrappers are used also here, but this time to convert (*una tantum*) the source data into the common data model, and then to answer the user queries.

  This approach to data integration is adopted in systems as Araneus [3], TSIMMIS, STRUDEL [10].

The **Araneus** project uses a representation of semantics based on a standard relational schema, where Web site crawling is employed to induce schemata of Web pages. These fine grained page schemata are later to be combined into a site-wide schema, and a special-purpose language, Ulixes, is used to build relational views over it. Resulting relational views can be queried using standard SQL, or trasformed into autonomous Web sites using a second special-purpose language, Penelope.

The **TSIMMIS** project was born at Stanford University, aimed at the development of tools that ease the rapid integration of heterogeneous information sources which may include both structured and unstructured data.

Each source of interest is endowed with a wrapper having two main functions:

- to convert the underlying data to a common information model known as OEM (Object Exchange Model). This process consists of two phases: first of all the user defines the OEM classes to be used for data representation; afterwards, an extraction technique based on a textual filter is applied (see [13]), initializing objects from the site data. OEM is a graph-structured data model where the basic idea is that each object has a label that describes its meaning. The label is used to extract information about objects that represent the underlying data.

- The second task of a TSIMMIS wrapper is to convert queries specified in a language for the common data-model, called LOREL into requests that the source can execute.

  The OEM objects obtained from the extraction process have no fixed schema; to provide usual benefits of a schema, DataGuides are introduced [12]. DataGuides are a sort of dynamic schemata generated from the OEM representation of the information sources, and are employed in order to make indexing and querying easier.

It is interesting to note that the TSIMMIS project is inspired by an instance-based representation of source data semantics, but in fact, DataGuides must be used to index and query semistructured data, since they provide the necessary facilities that are usually in charge of schemata in the database context.

STRUDEL provides a single graph data model similar to OEM in which all data sources are uniformly modeled, and a query language for data integration.

At the bottom-most level, data is stored in STRUDEL's own graph data repository or in external sources which are also viewed as graphs. The communication with the data sources is done through a set of wrappers. A wrapper maps an external source's data representation into STRUDEL collections and objects and translates STRUDEL queries into queries or operations understood by the source. This approach defines a virtual loose schema and maps on it the contents of the data sources.

Under this same category we can loosely classify also two more approaches: in the Resource Description Framework (**RDF**) [14] a data model based on directed labeled graphs is proposed for representing Web sites' metadata.

RDF is a formalism that has been proposed to the W3C consortium for the Web standards, to facilitate a synthetic representation of Web sites' contents. This is interesting, since it indicates that also the industrial community has reached the conviction that Web site semistructured data must be represented in some sort of schematic way, in order to become fully accessible and understandable.

Finally, our data-model oriented approach for structuring and querying WWW data is based on the **WG-Log** language, briefly described in the following section. WG-Log is a graph-oriented description and query language specifically designed for the needs of Web sites. Based on the graph query language G-Log [17], WG-Log was extended to include some standard hypermedia design notations, thus allowing to express model entities and relationships as well as navigational concepts. The last two approaches do not require a total conversion of the semistructured data to the common data model, rather, they complement the site contents with information about their semantics.

The choice of using a graph oriented approach is not new, as we have seen observing also OEM and RDF; however, the idea of the WG-Log project is to use graphs as the formalism for a visual language unifying navigational and logical aspects of Web sites.

19.3 THE WG-LOG MODEL AND LANGUAGE

The main purpose of the WG-Log language is to support database-like querying over Web sites; differently from many of the other projects, its approach to the representation of semistructured data is *explicitly* schema-based. Indeed, a WG-Log schema can be easily derived from the design phase of a Web site, and later used for posing and answering queries. Site dinamicity is not a real issue here; as long as the schema remains the same, querying based on this schema will always give the appropriate results. However, a periodical refreshing of the schemata kept in the repository will guarantee a tolerable amount of precision in case of site evolution that affects the structure (i.e., the schema) of the site. The point here is that answers to WWW queries are themselves portions of WWW sites. Even if at some point the schema does not reflect exactly the site structure, the answer to the query will contain an access point to some of the site information, wherefrom the user can navigate through the real pages. The approach is not proposed as an alternative to the classical searching and browsing of Web sites, thus the kind of information that would be available to a normal browser is still there.

WG-Log schemata, instances and queries are depicted as directed labeled graphs, whose nodes represent objects, while the edges indicate relationships between objects. The details of the WG-Log language can be found in [9]; for our purposes we only recall that two main node types exists, indicating simple objects, or *slots* (those with an atomic, perceivable value as strings, numbers) and abstract objects, or *entities* (the ones whose properties are described by means of aggregates of simple objects, e.g. a car, a person etc). Moreover, there are other kinds of nodes to describe indexes and entry points, useful for the representation of the hypermedia structure.

Graph edges can indicate both logical and navigational relationships, the former having a label indicating the relationship name. WG-Log rules, programs and goals can be used to deduce, query and restructure 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 the Web

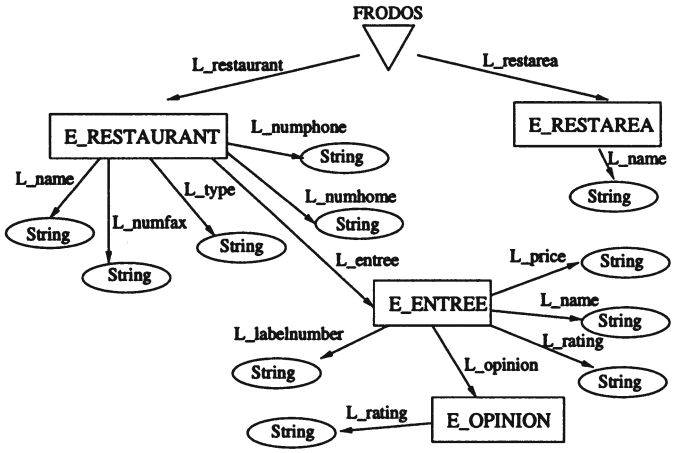

Figure 19.2: A sample WG-Log schema

site are available to the user. Goals can be applied to isolate the desired information and to arrange it in different presentation styles. 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 (respectively thin and bold black in this paper). Queries can include *dummy* nodes, which match any type of node of the schema. This yields a great flexibility in query expression, by allowing the expression of "wildcard nodes". As an example we include a WG-Log schema representing the Frodos restaurant chain (Fig. 19.2);
the main entities are depicted as rectangles (complex objects) and indicate restaurants, rest areas and restaurant entrees. The attributes (slots) of each entity are depicted as ellipses; the Node 'Frodos', depicted as a triangle is a special WG-Log node representing pages with singleton semantics, as for example the Home Page of a site.

In Fig. 19.3 we outline a sample query aimed at selecting all the entrees of the restaurant "Blues_by_the_bay"; the green node 'Result' is used to indicate the access point to the query result. This means that the query results will be presented as a list of the selected Entrees.

It is easy to see, from this example, that the presence of a WG-Log schema does not necessarily impose the restrictions cited in Section 1: the schema

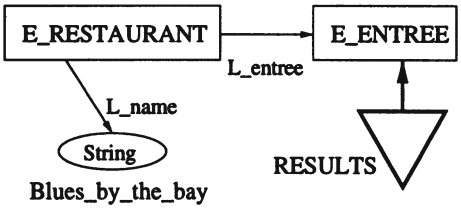

Figure 19.3: A sample WG-Log query

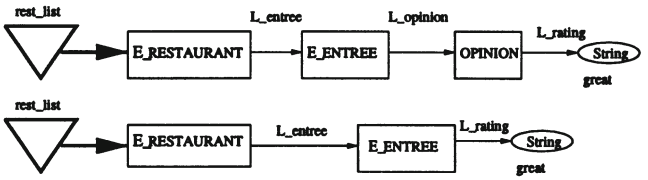

Figure 19.4: Another WG-Log query

of Fig.19.2 was deduced from an OEM data source, where irregular data are represented. Notice for example that the "Entree Rating" is expressed both as an atomic value and as a complex object ("Opinion"). In the schema this situation is depicted by introducing both a slot directly linked to the entity E_Entree via a link L_Rating and a slot linked to the entity E_Opinion which, in turn, is linked to E_Entree. Redundancy does not represent a problem here, since it is related to the schema and not to the instance.

The query of Fig. 19.4 aims at selecting all the restaurants which have an Entree rated 'Great'. Two rules are needed to capture both the situations: a query node 'Rest_list' is linked to both kinds of E_Restaurant entities, thus meaning that a unique result list is requested. Instead, if we were to select only one of the possible rating formats, we should use only the rule pertaining to it.

Note also that most of the WG-Log rules can be composed, like in this case, in a very simple way, by cutting and pasting parts of the schema with the help of a syntax-driven graphical *Query Editor*.

Our schema based approach is particularly apt for representing data which have a well-defined structure because with this method we obtain a compact data representation. However, this approach applies also to very loosely structured data or, in the worst case, to totally unstructured data: for this sort of data the schema and the instance tend to become identical. Our aim is to exploit the structure of data whenever it is possible, without losing the capability of describing unstructured data sources.

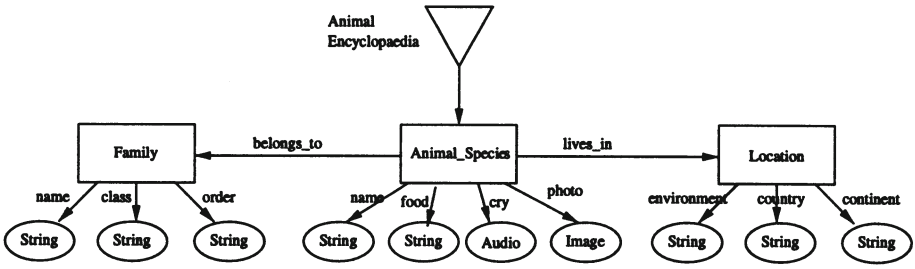

Figure 19.5: A sample WG-Log schema for the animals encyclopaedia

It is important to underline that, though WG-Log has initially been developed for the Web, it can be used also for generic semi-structured data. Consider for example the case of a CD-ROM, containing a flat file system, whose files represent data of an encyclopaedia regarding animals. The relevant information about the different species is available as text files; photos of the animals are available as image files and their cries as audio files. In this example there is a well-defined data domain (animal breeds); this knowledge has to be exploited to create a WG-Log schema (as the one in Fig. 19.5) representing the large amount of information provided in the CD-ROM.

The presence of a schema allows to refer to the data in a precise and compact way; in fact, on the instance graph representation can be added, where all multimedia files are considered just by means of a reference (file name). A software module can then take advantage of this representation to answer the queries submitted by the user, which are based on the abstract (schema) representation.

In the next section we will outline our proposal for the translation of OEM-represented data sources and DataGuides into WG-Log schemata.

19.4 EXPRESSING OEM IN WG-LOG

As seen in the first Section, the idea of representing data as a graph has inspired many projects that deal with (re)structuring and querying semistructured data. The TSIMMIS project adopts a data-model oriented approach based on OEM, whose graph based data model is currently used in other related projects, as LORE and C3 [8]. An example of OEM graph representation is shown in Fig. 19.6

The proposed standard has three main characteristics:

- *object orientation:* each OEM object has an *Object Identifier (OID)* and its representation is composed of three elements:
  - a label that describes what the object represents;

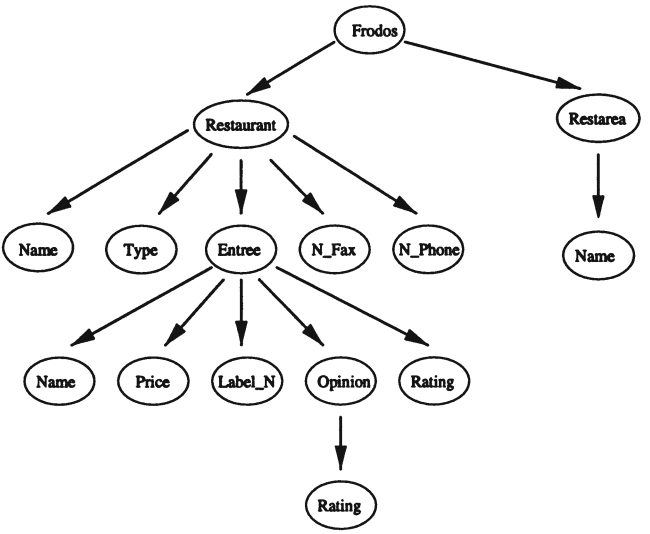

Figure 19.6: A sample OEM graph

- the data type of the object value. Any object may be atomic or complex. In the first case the type may be integer, real, string or any other data considered indivisible. The type is "set" if the object is complex.
- a value that is either the variable length value for atomic objects or, if the object is complex, a collection of one or more OEM subobjects each linked to the parent via a "descriptive textual label" .

- *representation of semantics*: the label component in OEM structure captures the semantics of the object.
- *flexibility*: the OEM structure is flexible enough to encompass all types of information.

An OEM object may be translated into a standard textual format [11]. This format keeps the information about semantics and contents and can be easily parsed by a program. An example of this format is the text in Fig. 19.7 (left).

This is the textual representation of an OEM database which contains complex (Restaurant, Entree, etc) and atomic (Name, Type, etc) objects. As in all semistructured data, in this OEM database there is no fixed schema in advance. In place of a standard schema, the LORE project uses an alternative tool to represent the time-invariant structure of an OEM database: an OEM object, called DataGuide, that is straightforwardly generated from the OEM database and dynamically maintained in order to provide several of the functions normally provided by means of a schema.

It is interesting to remark that, from the same OEM database, many DataGuides may be extracted, each of them representing a summary of the structure of the database. We are interested, in particular, to the so-called *strong DataGuide* [12], that induces a straightforward one-to-one correspondence between source target sets and objects in the DataGuide.

Like a WG-Log schema, a DataGuide contains no atomic values and it is essential for both programs and users to explore the OEM database and formulate queries. Since a DataGuide is an OEM object it is possible to translate it into the standard textual format: the graphical representation is depicted in Fig. 19.6.

Representing semistructured data in a synthetic way is also the basic aim of WG-Log schemata; exactly in the same straightforward way as a DataGuide is extracted, we can derive a WG-Log graph based on the OEM source data representation. Unfortunately the OEM syntax does not allow the expression of navigational and presentation-related concepts, which are very useful in the representation of Web sites: think for instance of queries involving some kind of "reachability" relationship between Web pages: this cannot be expressed over OEM or DataGuide representations.

However, even though OEM-based data lack the explicit navigational information contained in a WG-Log schema, any object in the database is either complex or atomic, both types that exist in WG-Log, so we can exploit this information to logically connect the objects.

As we outlined in Section 19.3, WG-Log schemata contain two types of nodes: *"non-printable"* representing complex object classes and *"printable"* representing atomic object classes; it is therefore possible to represent complex OEM objects via WG-Log non-printable nodes with label "E_<*complexObjectLabel*>" and atomic OEM objects by means of printable WG-Log nodes (slots) with label "string".

In an OEM representation, the binding between complex objects and their subobjects is the relationship "part_of".

This scenario is representable in WG-Log by means of logical links from the parent object to subobjects with textual label "L_<*subObjectLabel*>". If the subobject is itself complex it will have a set of labeled links to its component objects. The translation algorithm is specified in Fig. 19.8

Thus, OEM objects correspond to WG-Log objects; as an example, consider the OEM DataGuide object and its WG-Log representation in Fig. 19.9.

The textual format also provides the notion of *SymOid* (Symbolic Object Identifier) [11] to identify objects included as children of multiple complex objects. In particular, it is possible to specify that a SymOid is persistent: in an OEM database at least one persistent SymOid is required to serve as entry point. Considering this persistent SymOid as the entry point of the corresponding WG-Log schema, we can recursively apply the previous method starting from this identifier, in order to obtain a WG-Log representation of an entire OEM data source.

```
<Frodos::Frodos{
        < Restaurant {
                <Name "Blues_by_the_bay">
                <Type "Vegetarian">
                < Entree {
                        <Name "black_bean_soup">
                        <Price 10>
                } >
                < Entree {
                        <Name "asparagus_timbale">
                        <Price 2.04 >
                        < Label_Number 2 >
                } >
        } >
        < Restaurant  {
                <Name "Thai_city">
                <Number_Phone "497-4845">
                <Number_Fax "497-0000">
                <Number_Home "333-3333">
                <Type  "Thai">
                < Entree {
                        < Name "Route_9Red_curry">
                        < Opinion {
                                < Rating "Great">
                              } >
                } >
                < Entree {
                        <Name "green_curry" >
                        <Price 7.95 >
                        <Rating "Great">
                }>
         }>
        < Restarea {
                <Name "Route_9">
        } >
} >
```

Figure 19.7: OEM instance textual representation

```
WGGraph wgraph;
// symoid is the OEM database entry point's identifier
OemObject obj = *symoid;
// translate an OEM object into a WG-Log graph using
//persistent SymOid as entry point.
procedure OemToWGLog(obj) {
  while(obj.hasOtherChildren()) {
    OemObject child = obj.NextChild();
    if(obj.isComplex()) {
      // in an OEM object it's possible to create
      //some cycles using SymOid
      if(!child.alreadyVisited()) {
        // add Oem complex objects as WG-Log node
        wgraph.addNode(obj, child, child.label);
        OemToWGLog(child); // recursive procedure for
                           //complex object
      }
      // add a WG-Log link to a previously visited object
      else wgraph.addLink(obj, child);
    }
    //atomic OEM object is translated into a WG-Log slot
    else wgraph.addSlot(obj, child, child.value);
  } }
```

Figure 19.8: Algorithm to translate an OEM object into a WG-Log graph

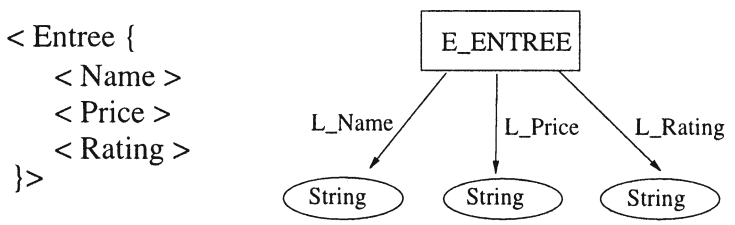

Figure 19.9: The OEM DataGuide textual format (left) and the translation in WG-Log graph

Note that Figure 19.2 depicts the WG-Log schema derived from the Data Guide textual representation of Fig. 19.7, where *Frodos* is the (unique) entry point.

The translation process outlined in this section can be applied to any OEM object and produces a WG-Log graph. DataGuides are themselves OEM objects, therefore the translation of a DataGuide will become a WG-Log schema, while the translation of a generic OEM representation will become a WG-Log instance. Consider now the translation of WG-Log queries into Lorel queries.

The query translation algorithm has to consider several particular conditions that increase its complexity and its readability. In this paper we just describe a simplified version of the algorithm showing how it works on two sample queries.

Let us consider the query depicted in Fig. 19.3, asking for a list of all the Entrees of the "Blues_by_the_bay" restaurant. The target of the operation is represented in WG-Log with a green node; therefore, in the considered query, the result is a list of Entrees. The target information has to be included in the SELECT clause of the Lorel query. The FROM clause is set by default to "root", and it is different only when more complex queries are translated.

The constraints that the result set must satisfy are expressed in the WHERE clause: in the sample WG-Log query, the only condition is that the restaurant name must be the one written in the "name" slot (i.e. "Blues_by_the_bay"). It is necessary to visit the query graph nodes to obtain the proper path expression.

The translation algorithm is specified in Fig.19.10
The Lorel translation of such a query is then:

```
SELECT Restaurant.Entree
FROM root
WHERE *.Restaurant.Name = "Blues_by_the_bay"
```

The "*" character is used here as a wildcard; in this way Lorel can indicate a variable length path starting from the root (specified in the FROM clause). Note that this is not necessary in WG-Log, where no language constraint imposes to navigate starting from the entry point.

Similarly the query depicted in Fig. 19.4 is translated into the following Lorel expression:

```
SELECT Restaurant
FROM root
WHERE *.Restaurant.Entree.Rating = "great"
OR *.Restaurant.Entree.Opinion.Rating = "great"
```

It is worth noticing that in this query we added the boolean operator OR; in this way we can express costraints on disjoint "path-expressions". This translates the WG-Log double-rule query.

19.5 INTERACTION WITH OEM DATASOURCES USING WG-LOG

Let us now use the translations we have defined. The WG-Log project is based on a distributed architecture (Fig. 19.11) whose main focus is the description

```
string WGtoLorel(WGGraph query, WGGraph schema){
   string Lorelquery;
   WGNode result = query.entry();
   // returns node to insert in SELECT
   WGNode root = schema.getroot();
   string select = result.getlabel(); //returns label
   Lorelquery.concat(select);
   string from = schema.getEntry(result);
   //returns path of the entry-node of the query
   Lorelquery.concat(from);
   WGNode node;
   string where;
   while (node = query.NextNode()){
   //returns NULL if all nodes are visited
      for i = 0 to node.numberSlot(){
         string constraint = node.getConstraint();
         //Node.link = value
         where.concat(constraint);
      }
   }
   LorelQuery.concat(where);
   return Lorelquery;
}
```

Figure 19.10: Algorithm to translate a WG-Log query into a LOREL query

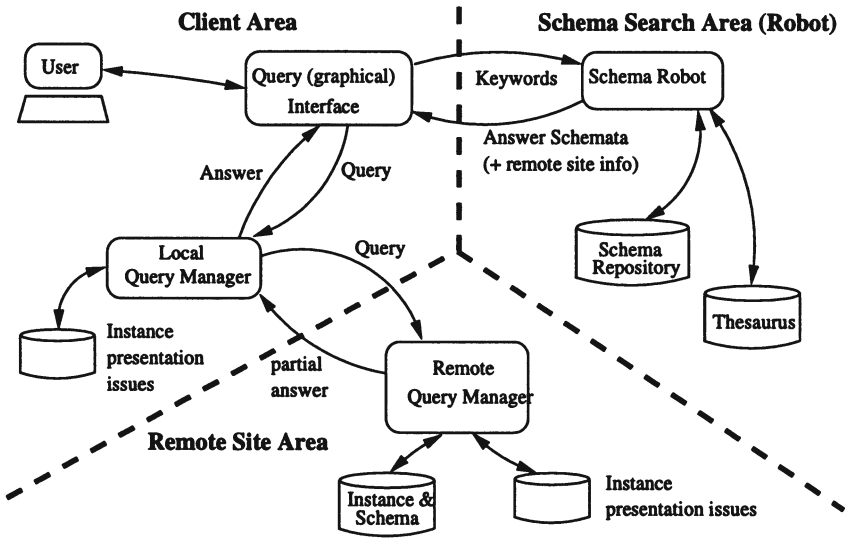

Figure 19.11: The WG-Log architecture

of Web sites by means of schemata. Schemata are made available to the user through a specialized module (Schema Robot) that keeps a repository of the known schemata and helps clients in posing the queries to the data sources more apt to the user needs, based on the information contained in the schemata.

Query execution is performed by the *Remote Query Manager* running at the target Web site, working on the internal representation of the site data. Depending on the site organization, such representation might have been extracted from the HTML pages or might exist from the beginning of the site life.

To integrate WG-Log with other Web Query Systems, the Schema Robot is replaced by a new, more general module, called the Trader, which makes WG-Log schemata describing both WG-Log and foreign data sources available to the user.

WG-Log schemata describing both WG-Log and foreign datasources are made available to the user through a specialized module, the *Trader* module also stored Translator modules as downloadable software components. Integration of OEM datasources managed by a TSIMMIS system (Fig. 19.12) in our general framework is indeed not difficult because on the TSIMMIS provides both the logical structure description (DataGuide), and the knowledge of the instance information. Since a DataGuide contains most information typical of a schema, and since it is possible to translate it from OEM to WG-Log (as described in Section 19.4), we can straightforwardly obtain the WG-Log description of OEM sites to be stored in our Trader repository. Moreover, we

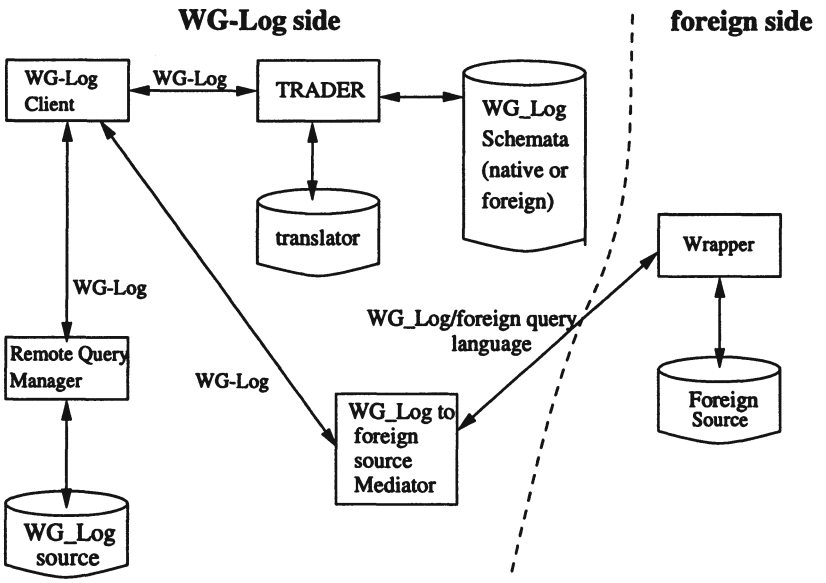

Figure 19.12: An interaction proposal

take advantage of the fact that the TSIMMIS system includes a *Mediator* component whose aim is to receive Lorel queries posed by clients and to forward them to a datasource *Wrapper*, which computes query results in terms of OEM objects that are sent back to the Mediator.

Whenever the user interacts with our Trader module, two basic usage patterns may occur.

The user selects a native WG-Log datasource, receives its schema and composes a WG-Log query. In this case, the query is sent to the remote datasource and execution is performed by the *Remote Query Manager* running at the target Web site, working on WG-Log's own internal representation of the site data.

Otherwise, the user chooses a TSIMMIS-derived schema. Then, a Translator component is tranferred to the user's machine together with the schema and the WG-Log query formulated by the user is converted to a Lorel query.

After the translation phase, the Lorel query is sent to the site Wrapper that will execute it exactly in the same way as if a TSIMMIS Mediator had sent the query. The query result produced by the Wrapper is an OEM object that is sent back to the WG-Log client and translated into a WG-Log result instance.

It is worth noticing that some kinds of WG-Log queries cannot be translated into Lorel, namely the navigational queries. However, such queries will never be issued against an OEM-represented site, since the WG-Log schema of such a

site, being derived from an OEM representation, will never contain any navigational information, thus the WG-Log queries will only involve logical concepts, and be translated into Lorel without difficulty.

19.6 CONCLUSIONS AND FUTURE WORK

In this paper we have presented a technique for the representation and querying of semistructured data, which can be effectively coupled with other approaches for querying the World Wide Web. In particular, we have shown the feasibility of this integration on the example of the Tsimmis/Lorel projects, which are based on the representation of site contents via the Object Exchange Model (OEM). This experience can be easily transferred to other data-model based Web Query Systems, like Araneus and Strudel, thus allowing the WG-Log environment to seamlessly manage information about site contents derived from different approaches.

Acknowledgments

The authors wish to thank A. Dovier, M. Baldi and F. Insaccanebbia for many useful discussions on the subject. Thanks are also due to H. Garcia-Molina and R. Goldman for their assistance with the TSIMMIS system.

References

[1] Serge Abiteboul, Querying Semi-Structured Data, ICDT'97, 6th International Conference on Database Theory, Vol. 1186, pp. 1-18, Springer, 8-10 Jan 1997.

[2] N. Ashish, C. Knoblock. Wrapper generator Semi-structured Internet Sources, Proceedings of the ACM SIGMOD International Conference on Management of Data.

[3] P. Atzeni, A. Masci, G. Mecca, P. Merialdo, and E. Tabet. ULIXES: Building relational views over the Web. In Proceedings of the 13th International Conference on Data Engineering (ICDE'97), pages 576–576, Washington - Brussels - Tokyo, April 1997. IEEE.

[4] S. Abiteboul, D. Quass, J. McHugh, J. Widom and J. Wiener. The Lorel query language for semistructured data. Journal of digital Libraries, November 1996.

[5] R. J. Bayardo, Jr. and W. Bohrer and R. Brice and A. Cichocki and J. Fowler and A. Helal and V. Kashyap and T. Ksiezyk and G. Martin and M. Nodine and M. Rashid and M. Rusinkiewicz and R. Shea and C. Unnikrishnan and A. Unruh and D. Woelk, InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments, Proceedings of the ACM SIGMOD International Conference on Management of Data, Vol. 26,2, pp. 195-206, ACM Press, May 13-15 1997.

[6] M. Baldi, E. Damiani, and F. Insaccanebbia. Structuring and querying the Web through graph-oriented languages. In *Proc. of SEBD 1997*, SEBD Conferences, Verona, Italy, June 1997.

[7] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, J. Widom. The TSIMMIS project: Integration of heterogenous information sources. In proceedings of IPSJ, Tokyo, Japan, October 1994.

[8] S. Chawathe and H. Garcia-Molina. Meaningful Change Detection in Structured Data.Proceedings of the ACM SIGMOD International Conference on Management of Data.Tuscon, Arizona, May 1997.

[9] E. Damiani, L. Tanca. Semantic Approach to Structuring and Querying the Web Sites. In *Proceedings of 7th IFIP Work. Conf. on Database Semantics (DS-97)*, 1997.

[10] M. Fernandez, D. Florescu, J. Kang, A. Levy, D. Suciu, STRUDEL: A Web Site Management System, Proceedings of the ACM SIGMOD International Conference on Management of Data, Vol. 26,2, pp. 549-552, ACM Press, May 13-15 1997.

[11] R. Goldman, S. Chawathe, A. Crespo, J. McHugh. A Standard Textual Interchange Format for the Object Exchange Model (OEM). Manuscript available from http://www-db.stanford.edu

[12] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases, VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, pp. 436-445, 1997.

[13] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, A. Crespo. Extracting Semistructured Information from the Web. Paper available at http://www-db.stanford.edu

[14] E. Miller, B. Schloss, O. Lassila, and R. Swick. Resource description framework model and syntax. Technical report, W3 Consortium, oct 1997. Revision 1,02, `http://www.w3.org/TR/WD-rdf-syntax/`.

[15] B.Oliboni, L.Tanca and D.Veronese. Using WG-Log to represent semistructured data: the example of OEM. In *Proceedings of SEBD 1998*, Ancona, Italy, Jun 1998.

[16] Y. Papakonstantinou and H. Garcia-Molina and J. Widom, Object Exchange Across Heterogeneous information Sources, Proceedings of the 11th International Conference on Data Engineering, pp.251-260, IEEE Computer Society Press, Mar 1995.

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

[18] D. Quass, J. Widom, R. Goldman, K. Haas, Q. Luo, J. McHugh, S. Nestorov, A. Rajaraman, H. Rivero, S. Abiteboul, J. D. Ullman, J. L. Wiener, LORE: A Lightweight Object REpository for Semistructured Data, Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, p. 549, 4-6 Jun 1996.