FINDER: A Mediator System for Structured and Semi-Structured Data Integration

M. Álvarez, A. Pan, J. Raposo, Fidel Cacheda, A. Viña
University of A Coruña. {mad,jrs,fidel,avc}@udc.es
Denodo Technologies {apan}@denodo.com

Abstract

Mediator systems aim to provide an unified vision over heterogeneous, distributed, structured and semi-structured data sources. We present FINDER, a mediator system which has been used for building several real-world data integration applications, both in the Internet and Intranet domains. In this paper we provide an overview of the system and remark some of its distinctive features.

1. Introduction

Mediator systems, which were first proposed in [1], address the problem of structured and semi-structured data integration. In this approach, the data remains in the sources and the mediator is responsible for providing users with the “illusion” of being querying a single and coherent global schema.

This paper describes FINDER, a system following such architecture, which allows the fast yet powerful extraction and combination of information from various heterogeneous, structured or semi-structured sources, to create an unified global schema, for said information. This system has been used for the construction of various industrial data integration applications integrating information from more than 700 different sources, such as Web sites, databases, spreadsheets, structured files, etc.

The following sections provide an overview of some key issues of the system: source model, wrapper generation and definition of the global schema relations. Some conclusions arising from our experience using the system are also shown. More detail can be found in the extended version of this paper [11].

2. Source Model

In our model each source exports a combination of relations, called base relations.

Each base relation is composed of a combination of attributes, each one belonging to a particular data type. Besides of the usual atomic data-types (strings, integers, money, dates, etc.), structs and arrays are also supported.

Each relation will represent its query capability through what we term as search-methods. Each search-method is comprised of three elements:

1) Negative capabilities: indicate the restrictions which must be satisfied by a query sent to the relation in order to be answerable through this search-method.
2) Positive capabilities: Indicate the query capabilities offered by the relation, once the negative capabilities has been satisfied.
3) Output of attributes: Set of attributes which appear in the response of the queries sent against the relation.

Both negative and positive capabilities are represented through a set of 4-uples with the format: (attribute, operator, multiplicity, possible_values)

where:

1) attribute is an attribute of the relation.
2) operator is an operator valid for the attribute data type.
3) multiplicity indicates how many query conditions for the given pair attribute/operator can execute the relation in the same query. ‘+’ indicates a number of values above 0 but with no upper limit.
4) Possible_values is the list of values the attribute can be consulted by. If it contains the value ‘Any’ it means that the range is not limited (in the range associated to the attribute data type).

Example: We consider the example of an on-line bookshop whose search form is shown in figure 1. The form forces the user to specify a value for the TITLE attribute (i.e. it is a mandatory field) and optionally the user can specify a value for the AUTHOR attribute, for the FORMAT attribute (restricted to a list of possible values) and a range of values for the PRICE attribute.
Queries by title and author are searches by keyword (operator Contains). An arbitrary number of words can be used to fill in the boxes and the source will perform the logical AND operation between them. For instance, filling the title box with the words ‘java’ and ‘xml’ would return all the books containing both words in its title.

In addition to the former attributes, we assumed that the shop also returns a PUBLISHER attribute.

![Figure 1. Bookshop search form](image)

We model this source as the following relation:

\[
R = \{ \text{TITLE} : \text{String}, \text{AUTHOR} : \text{String}, \text{FORMAT} : \text{Enumerated} (\text{Hardcover}, \text{Paperback}, \text{eBooks}), \text{PUBLISHER} : \text{String}, \text{PRICE} : \text{Money} \} \]

which capabilities can be modeled with the following search-method:

\[
\{ \\
\quad \text{NEGATIVE}\{ \\
\quad \quad \text{(TITLE, Contains, 1, ANY)} \\
\quad \}\ \\
\quad \text{POSITIVE}\{ \\
\quad \quad \text{(TITLE, Contains, +, ANY)} \\
\quad \quad \text{(AUTHOR, Contains, +, ANY)} \\
\quad \quad \text{(FORMAT, =, 1, \{‘Hardcover’, ‘Paperback’, ‘eBooks’\})} \\
\quad \quad \text{(PRICE, <=, 1, ANY)} \\
\quad \quad \text{(PRICE, >=, 1, ANY)} \\
\quad \}\ \\
\quad \text{OUTPUT\{TITLE, AUTHOR, FORMAT, PUBLISHER, PRICE\}} \\
\}
\]

3. Wrapper Generation

The following stage in the creation phase is the construction of wrappers. Each wrapper must provide access to the base relations of a source in a way that, when faced by the mediator, it behaves according to the model shown in the previous section. In FINDER, the wrapper generation process for Web sources, JDBC databases and structured or semi-structured text files is performed with the assistance of a semi-automatic generation tool which enables them to be created and maintained in a quick and simple way.

Due to its importance, we have paid an special attention to web sources. The following sub-section addresses that particular problem.

3.1 Wrapper Generation for Web Sources

Web scraping entails two main tasks: 1) how to access pages containing the required data, 2) how to extract the data.

In our system, to specify how to access the pages containing the required data, users can generate complex web flows by simply navigating with a web browser, without needing to worry about the complexities of handling Javascript, Dynamic HTML or session maintenance mechanisms.

For parsing the required data, we provide users with a supervised interactive tool (integrated into his/her web browser) in order to enable them to generate complex extraction patterns by simply highlighting relevant data from very few example pages, and answering some simple questions. The system relies on two wrapper programming languages: Navigation SEQuence Language (NSEQL) for specifying navigation sequences and Data EXtraction Language (DEXTL) for specifying extraction patterns. An overview of these two languages follows. See the extended version of this paper [11] for a description of the graphical tools supporting them.

NSEQL is a navigation sequence specification language which works at a “browser-level” instead of at a “http-level” (as most previous approaches do).

NSEQL lets users define “macros” directly over a web browser interface, thus making an access to a source in our system identical to the process carried out by a user who connects to this source using MSIE. This enables the wrapper to be completely independent of the source session maintenance mechanisms (which can be highly complex in commercial sources), Javascript code, dynamic HTML, etc. It also allows wrapper navigation sequences to be generated “by means of examples”, i.e. simply by navigating.

Example: Let us suppose we want to model a web bookshop(e.g. Barnes & Noble) as a relation R which attributes are TITLE, AUTHORS, PUBLISHER, PRICE and DISCOUNT. The “advanced search” form lets the user perform queries by filling an author field and/or a title field.

The following sequence of commands is able to perform a search in such a form.
The `goto` command makes the browser navigate to the given URL. Its effects are equivalent to that of a human user typing the URL on his/her browser bar.

The `FindFormByAction` command looks for the first HTML form in the page with the given ‘action’ attribute value.

Then, the `SetInputValue(fieldName, value)` commands are used to assign values to the form fields. The ‘value’ parameter might be a string (closed by "") or a variable(prefixed by '@'). Variables are replaced by its value at execution time.

The command Submit(), submits the form and loads the result page. Note that the form is submitted by generating a ‘submit’ event over the object representing the form in the browser DOM tree. We do not construct any HTTP request; instead, we delegate this task to the browser. This way, we do not have to deal with common and usually very complex issues as hidden session numbers, Javascript forms, and so on. These are transparent to our system, exactly in the same way that they are transparent to a human user navigating with his/her browser.

Once the wrapper navigation sequence has been constructed (and its mapping with base relation search methods realized), a specification must be written in the DEXTL language to state how to extract the tuples from the HTML pages returned by the source. Consider the following simple example.

**Example:** The following figure shows a fragment of a response from the e-shop, Barnes & Noble.

![Image of a response from the e-shop, Barnes & Noble.](image_url)

- **Beginning Java 2 - Jdk 1.3 Version**
  - In Stock: Ships within 24 hours.
  - Ivor Horton / Paperback / Wrox Press, Inc. / March 2000
  - Our Price: $39.99, You Save 20%

- **The Complete Java 2 Certification Study Guide**
  - In Stock: Ships within 24 hours.
  - Simon Roberts, Philip Heller, Michael Ernest / Hardcover / Sybex, Inc / September 2000
  - Our Price: $39.99

Let us suppose again we want to model this source as a relation R which attributes are: TITLE, AUTHORS, PUBLISHER, PRICE and DISCOUNT. A valid specification would be:

```plaintext
PATTERN R {
    :TITLE ENDOFLINE
    IRRELEVANT ENDOFLINE
    :AUTHORS "/" IRRELEVANT "/" PUBLISHER "/" IRRELEVANT ENDOFLINE
    "Our Price:" :PRICE ",You Save" :DISCOUNT ENDOFLINE?
}
```

The basic idea is to construct a pattern in which the system will search for matches in the whole document. A pattern will consist mainly of text attributes to extract (here TITLE, AUTHORS, PUBLISHER, PRICE and DISCOUNT), text portions not to be extracted (denoted as IRRELEVANT) and separators.

Separator between attributes can be text separators (",", "Our Price.", etc.) or tag-type separators. A tag-type separator(ENDOFLINE) represents a regular expression concerning document format information(commonly HTML tags).

The tag-type separators can be defined to suit and reused in different specifications. ENDOFLINE, for example, could be defined through a regular expression of the style ("<br>|"<p>|"<tr>|"</td></tr>"), and could be used in any specification for HTML sources. Normally this kind of separators are defined by a user who is something more of an expert and grouped into reusable sets. Then, these sets are used by less qualified wrapper generators to build pattern specifications for many sources.

Our experience shows that most applications need a very reduced set of tag-type separators. Nevertheless, the ability to define any tag-type separator needed guarantees the system will be able to deal with any situation.

To avoid pattern ambiguity, several constructions are provided to limit the search space for a pattern, identifying relevant regions in the page. The basic constructions are FROM and UNTIL which identify start and end patterns to a relevant region.

The system also allows for the easy handling of nested subpatterns inside the page(which are mappable to array and record-type attributes of a relation). Another feature lets the user include navigation sequences inside the pattern in order to handle situations as tuples whose attributes are distributed in different pages or multi-page search results.

## 4. Definition of the Global Schema Relations

Once the base relations have been defined and their wrappers constructed, each relation of the global schema is defined by a query concerning the base relations, in a similar way to the definition of views in a conventional database. This approach is known in mediator literature as Global As View. The query is expressed in a SQL-like language and can include selections, projections, unions, joins, orderings and aggregations. It should also be pointed out that a view, like base relations, can also be defined by previously defined intermediate views.

**Example:** Consider the three base relations:
A ={TITLE:String,AUTHOR:String, PRICE:Money}
B={TITLE:String,AUTHOR:String, PRICE:Money}
C={TITLE:String,AUTHOR:String, RELEVANCE:Integer}

A and B represent two Internet book merchant sites. C represents a web source in which its users assess the books’ relevance, and the system enables searches for the average relevance of a certain book. Consider that we want to obtain a global schema relation:

R={TITLE,AUTHOR,PRICE,RELEVANCE}

which contains all the books from A and B (only one tuple for each book), together with their average relevance according to C, and the minimum price for each book found in the two e-shops. The definition of R in our system would be:

SELECT  TITLE, AUTHOR, RELEVANCE, 
MIN(PRICE)
FROM (I SELECT TITLE, AUTHOR,PRICE 
FROM A
UNION
SELECT TITLE,AUTHOR,PRICE 
FROM B),C
WHERE  I.TITLE=C.TITLE AND
I.AUTHOR=C.AUTHOR
GROUPBY  TITLE,AUTHOR,RELEVANCE

As has already been mentioned, the base relations can be limited in terms of their query capability. When the global schema relations are defined, FINDER is capable of computing their query capabilities. This allows FINDER to know in advance if a certain query is going to be answered and also makes it possible for a mediator to be a source for other mediators. The query capability propagation mechanism from the base relations to the global schema relations is described in [9].

5. Experience obtained in using the System

This system (in successive versions) has been used to construct various commercial applications which are currently being used in real operational environments.

Applications constructed so far can be classified in two groups:

- Search, comparison and aggregation applications on the Internet.
- E-business data integration applications.

In the first group of applications, the methods by which source data is combined are normally quite simple(i.e. mainly unions, projections and selections).

About the materialization scheme, we usually use the virtual approach along with a cache system. The cache stores results from recent queries and it is able of determine if a new query can be answered using any subset of the current cache content.

This simple schema turns to be very effective for most applications due to the critical mass provided by Internet users.

The greatest difficulties in Internet applications are encountered in the construction and maintenance of wrappers. In our experience, the most difficult problem involved in commercial web wrapper generation is not parsing (which has been the main issue addressed in literature), but creating the navigation sequences required to access the data, dealing with Javascript, Dynamic HTML, HTTPS, frames, or complex non-standard session maintenance mechanisms

Working at a “browser-level” instead of at a “HTTP-level”, makes all these complexities transparent for wrapper generators.

In the second group, typical application scenarios include CRM (Customer Relation Management), where the customer data is distributed across many heterogeneous data repositories and EIPs (Enterprise Information Portals), where our system is used to provide an unified view over the heterogeneous content to be delivered through the portal.

In these cases, FINDER normally operates on a mix of Web sources, relational databases, spread-sheet systems and semi-structured file systems. In addition, data is normally combined in a much more complex way than in the applications in the previous group, although there do tend to be fewer sources.

An important issue derives from the fact that limitations on sources query capabilities are very common in practice. Some techniques [10] have been proposed in order to address that kind of problems. Nevertheless, most research works do not compute the query capabilities of the global relations from the sources capabilities.

Our experience says that this is a must-have feature: it makes possible to use mediators as source for new ones, easily enabling incremental data integration processes. It is also very important that users know in advance the queries supported by the mediator. In our experience, industrial users do not find trial&error query processes acceptable. As far as we are aware, this important issue has been dealt with only in [7]. But the query capability description framework used is too simple to properly model most real sources: for instance, it is only able to properly represent sources having query interfaces which allow exclusively for queries involving an unique query condition per attribute, and where only the ‘=’ operator is permitted. To deal with this problem, we have created our
own algorithm for computing mediator query capabilities [9].

Our experience also shows clearly that the theoretical advantages of mediator–based data integration also become a reality in practice. Typical mediator-based data integration projects are much faster, cheaper and less intrusive than traditional approaches.

6. Related Work and Conclusions

In recent years, a significant amount of academic mediator systems has been developed, such as Tsimmis [2], Information Manifold [3], or Tukwila [4]. Various specific aspects in the construction of mediator systems have also been studied by the research community: wrapper generation for Web sources [5][6], query optimisation [4], reformulation mechanisms [8], calculation of mediator capability [7], etc.

Some distinctive features from our system follow:

1) Automatic calculation of the query capacity of the global schema relations using a simple but powerful capability description mechanism. Our experience says this is a “must-have” feature since it makes possible to use mediators as source for new ones, easily enabling incremental data integration processes and it enables users to know in advance the queries supported by the mediator.

2) Powerful wrapper generation tools to allow reduced maintenance cost for existing highly complex commercial web sites. Our experience suggests that most of the proposed approaches will fail in effectiveness when dealing with this kind of sources, mainly because not only parsing is needed: translating a query into a navigation sequence, and then performing that sequence, is often at least as complex.

7. References