Using YAT to Build a Web Server*

Jérôme Siméon and Sophie Cluet

INRIA, BP 105, 78153 Le Chesnay, France
tel: (33) 1 39 63 51 46, fax: (33) 1 39 63 56 74
email: {Jerome.Simeon,Sophie.Cluet}@inria.fr

Abstract

Integration of heterogeneous data sources in a Web environment has become a major concern of the database community. Architectures, data models and query languages have been proposed but the complementary problem of data conversion has been less studied. The YAT system provides a means to build software components based on data conversion, such as wrappers or mediators, in a simple and declarative way. We show that the YAT system can also be used to create integrated Web views over heterogeneous data sources very easily. Only minor changes were required for YAT to provide data integration (as opposed to data conversion) in a Web environment. Finaly, we report on our experience while building the Verso Web site using YAT.

1 Introduction

Many semistructured data models (e.g. [11, 5, 8, 1]) and appropriate query languages (e.g. [2, 7, 4]) have been proposed in the last few years. All of them have been designed as a means to integrate heterogeneous, more or less regular data. We pursued a complementary approach and focused on a system to support data conversion (as opposed to integration). The main motivations for this work were: (i) data conversion is an important part of the global integration problem and has been kept hidden in other proposals (usually managed by hand-made wrappers), (ii) existing semistructured models do not consider ordered collections which is crucial to many existing formats, especially in documents (HTML, SGML or XML), (iii) conversion languages for semistructured data require some typing capacities that are currently lacking in other proposals, and finally, (iv) query languages do not provide the rich restructuring capabilities required to perform data conversions.

*This work is partially supported by the OPAL project (Esprit IV project number 20377) and the AFIRST association (Association Franco-Israélienne pour la Recherche Scientifique et Technique).

In this paper, we describe how the YAT system [6], that we built for data conversion, can also be used for data integration in a Web environment. Essential features such as a query processing, persistence or view maintenance are still missing. Yet, we show that, as it is, the YAT system is very well suited to develop quickly, in a simple and declarative way, sophisticated Web views over heterogeneous data sources.

The YAT system relies on a data model allowing a rich and uniform representation of data and a rule based language. The YAT system provides a graphical interface, a library of basic import/export wrappers and standard conversion programs, an interpretor for the language, some type-checking facilities, as well as various mechanisms to combine or compose programs. Its originality, as compared to other similar approaches, can be summarized as follows:

YAT vs other semistructured based systems: As previous semistructured data models (e.g., [11, 5, 8, 1]), we rely on a simple representation of graphs to capture data coming from heterogeneous sources. But YAT also provides (i) the possibility to order the outgoing edges from a vertex (in the style of [1]), and most importantly (ii) the ability to capture various levels of representation via a sophisticated type mechanism (as will be shown in Section 3 on Web-oriented examples).

The YAT language (YATL) is a declarative, rule based conversion/integration language. It is not a query language and does not provide query facilities such as generalized path expressions [2]. On the other hand, it offers more powerful restructuration primitives and explicit Skolem functions, as in [10, 8], to manipulate identifiers and create complex graphs. Its main originality is in the management of non-set collections, allowing to manage ordering and grouping in collections with duplicates (e.g., list, bag). Both these features are important for most usual formats and essential in HTML.

As opposed to integration systems based on a semistructured model, YAT does not yet support persistence, view maintenance, etc. However, we believe that such systems could benefit from the YAT model and conversion language which provide richer modeling and restructuring capacities.

YAT vs other Web targeted systems: Several systems designed for the management of Web sites over legacy heterogeneous sources already exists. Some proposals rely on semistructured data models (e.g., [8]), others on a document driven approach (e.g., [13]). Finally, [3] focuses on the maintenance of large sites leaving aside the problems related to the heterogeneity of the input sources.

Because they lack the ability to deal with ordered collections, [8] needs two languages: one to describe the integration part, another to specify the generated HTML documents. YATL supports both tasks.

A document-drivent approach is followed in [13]. It proposes to generate Web sites starting from an editor which allows to generate HTML documents with embedded queries over heterogeneous sources. However, this system does not provide integration capabilities but only mapping to HTML using a rather procedural language.

The Araneus Project [3] proposes a conceptual and a logical model to design Web sites. The Penelope language then allows to specify the generation of HTML pages from some databases. As opposed to this system, YAT offers a quick and easy-to-use approach which allows to integrate heterogeneous sources but does not consider the maintenance of the generated server.

The paper is organized as follows. In the remainder of the paper, we will use the Verso Group publications Web site as an example to demonstrate the YAT system abilities[1]. In Section 2, we explain how to install a Web server architecture using the YAT system. The YAT model is presented in Section 3, Section 4 shows a sample of the program that is used to build the Web server and how this program is applied. Section 5 concludes the paper.

[1]See `http://www-rocq.inria.fr/verso/publications/` for a demonstration.

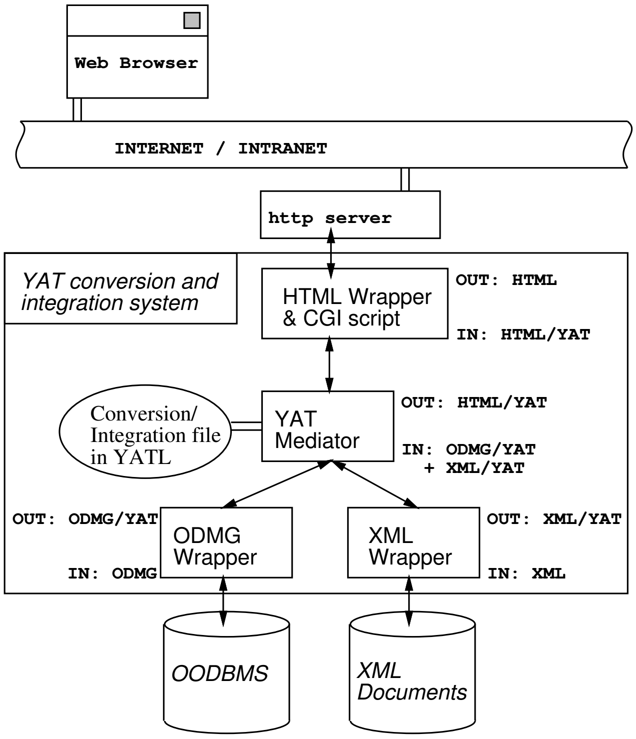

Figure 1: Architecture of a Web server with YAT

2 Building a Web server with YAT

Building a Web server with YAT can be done quickly and easily. To demonstrate this, we use the Verso group publication site as an example. This server is built on top of two heterogeneous sources: (i) an O_2 database management system stores information about the group researchers, (ii) an XML file contains an annotated list of their publications.

Figure 1 describes the architecture of the Web site. It is a very standard mediator/wrapper architecture. Wrappers for ODMG, XML (resp. HTML) import (resp. export) information about the connected sources to (resp. from) the YAT middleware model. As will be illustrated in the next section, they implement a straightforward translation from/to YAT. The YAT mediator is used to (i) integrate the input sources and (ii) to specify the conversion from these sources to an HTML-compliant YAT representation.

The Web server is built in four steps:

1. *Determine the relevant data sources*

   In our application, we need to access the O_2 DBMS and some XML files.

2. *Connect these sources to YAT supplied wrappers and the wrappers to a YAT mediator.*

The YAT system provides a library of standard wrappers[2]. Each wrapper supports three kinds of services:

- Connection/disconnection to/from a source or a mediator,
- Extraction of meta-information about the connected source (e.g. which model or format, what are available query primitives, etc.),
- Import/Export facilities. The wrapper provides information from a source at different levels of details. For instance, the ODMG wrapper provides YAT representation of: (i) the ODMG model, (ii) a specific schema or class (for a given database), (iii) actual data (for a given query or object identifier). As will be explained in Section 3, the YAT model imposes no restriction on what the wrappers may generate. For instance, if needed, one could easily import a partially instantiated schema.

3. *Write the integration/conversion program using YATL or its graphical interface.*

   This obviously is the hardest part. The Web designer first asks the input/output wrappers for the structure of the data he plans to export/import. This structure can be more or less precise according to his needs (see the wrappers Import/Export facilities above). Then he writes an integration/conversion program that fits these data structures. In the example, the program input combines ODMG and XML data structures and the output is HTML compliant.

   YAT provides a number of facilities that considerably help the programmer in this task:

   - A library of standard conversion programs (e.g., a generic ODMG to HTML-compliant conversion *à la* O_2Web [9]).
   - A customization mechanism by instantiation of YAT programs that allows, for instance, to customize an ODMG to HTML program in order to display a given schema, class or object in a special way. This is done as follows: (i) a generic program is loaded, (ii) a more specific YAT representation of the schema/class/object is imported through the wrapper or given explicitly, (iii) the program is instantiated automatically to fit the given representation. The generated rules can then be modified either manually or through another round of instantiation. For instance, we may instantiate the program again on the output side, using this time a new HTML structure imported from a WYSIWYG editor.
   - A mechanism that allows to combine two programs into one. For instance, one may combine the ODMG to HTML and XML to HTML programs to form a program whose input combines ODMG and XML compliant data.

   Using these mechanisms, writing a program is, most of the time, only a matter of slightly modifying another existing set of rules. Finally, YAT supports a type checking module that can be used to check the coherence of the modified program or trace potential specification deficiencies at run time (i.e., some part of the input data does not match any rule of the program). Because of space limitation, we will not describe all these mechanisms any further, but the reader can consult [6] for a more complete presentation.

4. *Describe the data on which the program is to be executed.*

   One last information is missing to execute the YATL program, namely the set of actual data on which it will run. Originally, YAT was designed as a conversion system. Thus, a conversion program was called by a user on some given input data. This is not appropriate for a Web system where HTML data has to be generated on the fly according to what is requested. The current version of YAT, relies on a pragmatic solution to this problem: the user specifies the queries that have to be executed on the sources and their relationship with the program rules (see Section 4 for an example). A more elegant solution would be to describe this kind of mapping at the wrapper level in the style of [12]. This solution requires additional work related to query processing in mediator/wrappers architectures that we are currently considering.

Once these four steps are achieved, the programmer has a Web server that is accessible through a YAT provided CGI script linked to the HTML output wrapper. To give an idea of the work involved, the specification for the Verso site contains 7 different conversion programs and required a couple of hours

[2] Current list of wrappers covers HTML and XML documents, Object-oriented databases and BibTeX. Consult the YAT system homepage for an updated list of existing wrappers (`http://www-rocq.inria.fr/verso/Jerome.Simeon/YAT/`).

work. Some final remarks should be stressed: (i) the YATL language is interpreted and then, allows to easily test program modifications or to quickly change the layout of pages (no compilation is required for the mediator that remains generic), (ii) since HTML pages are generated on the fly from the actual data sources, any update is automatically and immediately visible on the Web server.

In the next section, we show how outside information (either model/format, schema, data, ...) are represented in YAT in an homogeneous fashion. Section 4 introduces two of the YATL rules that are used in the example Web server.

3 The YAT data model

The YAT data model consists of named ordered labeled trees that may be unioned (with the $\vee$ symbol) to form a pattern. A set of patterns form a model, which is used to represent external data (ODMG, XML and HTML in our example).

The most interesting feature of YAT is that it allows to represent data at various levels of detail. For instance, Figure 2 shows several models as exported by wrappers. The three models on the upper part of the figure may be used to represent the O_2 object corresponding to the researcher whose name is Tova Milo. Going from left to right, the models get more and more precise. The first one allows to represent any YAT compliant data. The *Researcher* model represents the O_2 class *researcher* but also all objects belonging to that class. The last model is an actual representation of the object itself like in usual semistructured data models. Each of these three models is an instance of its predecessor(s).

Another interesting example is the XML model in the lower left hand side of the figure. It mixes precise with non-specific informations. An XML entry is described as being a sequence of fields. The first fields are precisely described (i.e., *title*, *year*, *authors*). Then, the pattern just indicates that they are followed by zero or more fields. Each field has a label of type *Symbol* and may be composed of a simple string or of a sequence of other fields. Finally, the last model (partially) represents any HTML data (internal nodes, e.g. *h1*, *ul*, *a*, represent HTML tags).

Let us now see how these models are constructed and give an intuition of the instantiation mechanism. The nodes of a YAT tree may be labeled with a (i) constant (e.g., "Tova Milo"), (ii) symbol (e.g., *class*, *researcher*, *set*), (iii) type (e.g., *Any*, *String*, *Symbol*), (iv) pattern name (e.g., ***Yat***) or (v) reference to a pattern name (e.g., **&*Yat***). The edges of a

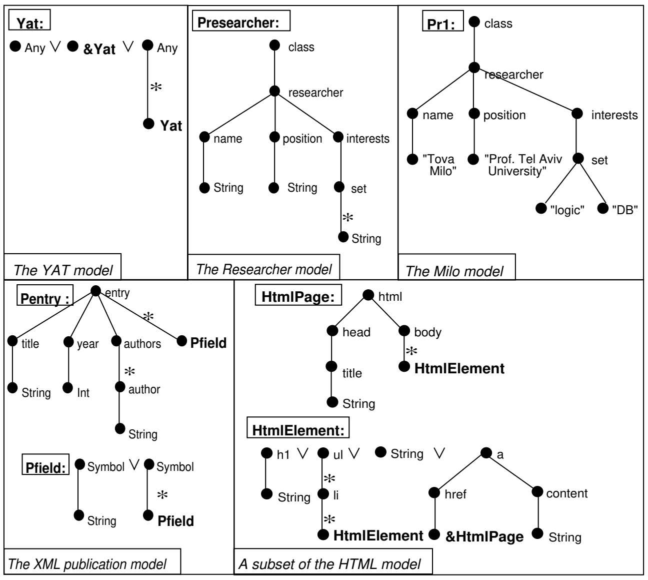

Figure 2: Some YAT Models

YAT tree may be annoted by an occurrence symbol. In the example, there is only one such symbol: $\star$. It indicates the possibility to replace the annotated edge by an arbitrary number of edges (e.g., the two *set* rooted subtrees).

A model is an instance of another, if one can find a mapping between their edges and nodes. An edge is instantiated according to its occurrence symbol. A node is instantiated according to the following rules:

- A constant or symbol node can be instantiated by a node whose label is identical (e.g., the nodes labeled *class*).
- A type node can be instantiated by a node whose label is a subtype or an instance of that type. E.g., the node labeled *Any* can be instantiated by the nodes labeled *String*, *Symbol*, "Tova Milo" or *class*.
- A pattern node can be instantiated by a tree, instance of that pattern. E.g., the two trees whose roots are labeled *researcher* are instances of the ***Yat*** labeled node.
- A referenced pattern node can be instantiated by a node whose label is a reference to an instance of that pattern. E.g. the **&*HtmlPage*** labeled node is instance of the **&*Yat*** labeled node.

4 Integration using YATL

Now that we are able to describe the structural input (ODMG and XML) and output (HTML) of our

$$
\begin{array}{l}
\boldsymbol{HtmlPageAll}() : \\
html\langle \to head \to title \to \text{"Verso Researchers"}, \\
\qquad \to body\langle \to ul \xrightarrow{\star} li \to a \\
\qquad\qquad\qquad \langle \to href \to \&\boldsymbol{HtmlPage}(N), \\
\qquad\qquad\qquad\quad \to content \to N\rangle\rangle\rangle \\
\quad \Longleftarrow \\
\qquad\quad \boldsymbol{Presearcher} : \\
\qquad\quad class \to researcher \\
\qquad\qquad\qquad \langle \to name \to N, \\
\qquad\qquad\qquad\quad \to position \to P, \\
\qquad\qquad\qquad\quad \to interests \to set \xrightarrow{\star} I\rangle
\end{array}
$$

Figure 3: Rule **All Researchers**

$$
\begin{array}{l}
\boldsymbol{HtmlPage}(N) : \\
html\langle \to head \to title \to H1, \\
\qquad \to body\langle \to h1 \to H1, \\
\qquad\qquad\qquad \to \text{"Position is "}, \\
\qquad\qquad\qquad \to P \\
\qquad\qquad\qquad \to ul \xrightarrow{[]_T} li\langle \to \text{"title : "}, \\
\qquad\qquad\qquad\qquad\qquad \to T, \\
\qquad\qquad\qquad\qquad\qquad \to \text{"published in "}, \\
\qquad\qquad\qquad\qquad\qquad \to Y\rangle\rangle\rangle \\
\quad \Longleftarrow \\
\qquad\quad \boldsymbol{Pentry} : \\
\qquad\quad entry\langle \to title \to T, \\
\qquad\qquad\quad \to year \to Y, \\
\qquad\qquad\quad \to authors \xrightarrow{\star} set \to N, \\
\qquad\qquad\quad \xrightarrow{\star} \boldsymbol{Pfield}\rangle, \\
\qquad\quad \boldsymbol{Presearcher} : \\
\qquad\quad class \to researcher \\
\qquad\qquad\quad \langle \to name \to N, \\
\qquad\qquad\quad\; \to position \to P, \\
\qquad\qquad\quad\; \to interests \to set \xrightarrow{\star} I\rangle, \\
\qquad\quad H1 \texttt{ is } \text{concat("Homepage of "}, N)
\end{array}
$$

Figure 4: Rule **Single Reasearcher**

application, we need a way to map the former to the later. This is done using YATL, either directly or through its graphical interface. YATL allows to describe complex conversions on graphs. It is rule-based, each rule describing a small part of the data conversion. Because of space limitations we will not give the complete program allowing to map object researchers and XML documents to HTML but two simplified, yet significant, rules. Although, the syntax may look somewhat complicated, note that: (i) most of the time, the programmer only has to modify some rules proposed by the system and, from our experience, can master the syntax of the rules rather fast; (ii) The programmer has the option to use the graphical interface to modify such rules [6]. We now illustrate the YATL language on two simplified rules, taken from our web site.

Consider the rule **All Researchers** on Figure 3. This rule creates a single Web page that contains a list of pointers to all researchers in the database. On the lower part of the rule (i.e., after the $\Longleftarrow$ symbol), the pattern ***Presearcher*** is used to filter the input data and retrieve the required information through variable bindings (in the example, the researchers names are bound to Variable N). On the upper part, the output pattern describes how the HTML structure is generated. The output of the rule is identified by a Skolem function: ***HtmlPageAll***(). The fact that this Skolem function has no parameter indicates that only one pattern will be generated, including the transformation of all the input filtered data. The head of the rule contains a reference to another Skolem function, this time parameterized by the researchers names (**&*HtmlPage***(N)). Note that the program does not contain URLs. These will be automatically generated from the Skolem functions by the HTML wrapper/CGI script. Note also that we could have used the researcher's identity as a parameter, replacing N by ***Presearcher*** in the Skolem function. As it is, there will be one reference per researcher name. The HTML page associated to this reference is described by another rule **Single Reasearcher** given in Figure 4.

This rule computes the page associated to one researcher identified by its name. It integrates data from the O_2 database and the XML files. Its input is a set of patterns, instances of the ***Pentry*** and the ***Presearcher*** patterns. Its output is a new HTML page identified by the ***HtmlPage***(N) Skolem function. Remark that Variable N is used in both ***Pentry*** and ***Presearcher*** patterns, making the correspondence between a given researcher in the object database and its publications in the XML file. The page title $H1$ is generated using a function that concatenates two string. Among other things, the page contains a list of the researcher publications (ul), which is ordered by title. This is specified by the $\xrightarrow{[]_T}$ edge which states that a new child is created for each distinct title T (grouping) and that the children must be ordered according to the value of T (ordering). The ordering possibilities of YATL are mandatory to manage Web documents, and are widely used in the Verso publication site to give several views of a given list of publications (ordered by title, by publication date, by authors, etc).

Now, that the creation of Web pages is specified, we only need to explain how to connect the rules to their actual input. This is done with the following instructions.

```
root HtmlPageAll() :-
    ODMG(AllResearchers);
HtmlPage(N) :-
    ODMG( select x
          from x in AllResearchers
          where x.name = $N),
    XML($N);
```

The first statement indicates that `HtmlPageAll()` generates the root page of our Web server (there could be several such rules). It also states how the input of the corresponding YATL rule is obtained (by the query `AllResearchers` on the ODMG wrapper). In a similar manner, the second rule specifies the input needed to generate the page corresponding to the Skolem function `HtmlPage(N)`. This time two queries are involved: one on the ODMG wrapper, the other on the XML wrapper.

Let us finally explain on this example how the components works together. The HTML wrapper asks the mediator for the identifier of the root page. It then generates the corresponding URL that looks like:

```
http://myserver/cgi-bin/cgiwrapper?HtmlPageAll()
```

By invoking that URL, a user activates the CGI script which in turn asks the mediator for this page. The mediator calls the corresponding query on the ODMG wrapper and converts the returned ODMG patterns according to the specification: one HTML pattern representing the root page with references to the researchers in the form of Skolem functions, e.g. ***HtmlPage***("Tova Milo"). The pattern is then given to the HTML wrapper which converts all Skolem functions into URLs and the global pattern into an HTML page. By clicking on any of these URLs, the user will activate the CGI script once more and similar action will be performed.

5 Conclusion

In this paper, we succinctly described how the YAT system, designed primarily for data conversion, can favorably be used to implement a Web interface over heterogeneous data sources. This approach has been used successfully for the Verso group Web site at INRIA. From this experience, we believe that the YAT system is a good step towards a complete data integration system. However, some issues should be addressed to achieve this goal (e.g. query processing, incremental view maintenance and optimization). We plan to work in this direction in the future.

References

[1] S. Abiteboul, S. Cluet, and T. Milo. Correspondence and translation for heterogeneous data. In *Proc. Int. Conf. on Database Theory (ICDT)*, Delphi, Greece, January 1997.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, April 1997.

[3] P. Atzeni, G. Mecca, and P. Merialdo. Design and maintenance of data-intensive web sites. Technical Report RT-DIA-25-1997, Università degli Studi id Roma Tre, Roma, Italy, June 1997.

[4] P. Buneman, S. B. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 505–516, Montreal, Canada, June 1996.

[5] P. Buneman, S. B. Davidson, and D. Suciu. Programming constructs for unstructured data. In *Proc. Int. Workshop on Database Programming Languages*, Gubbio, Italy, 1995.

[6] S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your mediators need data conversion! To appear in SIGMOD'98 International Conference on Management of Data, October 1997.

[7] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. System demonstration - strudel: A web-site management system. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, Tucson, Arizona, May 1997. Exhibition Program.

[8] M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3):4–11, September 1997.

[9] O_2 Technology, Versailles. *The O_2 Web Reference Manual version 4.6*, September 1996.

[10] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proc. International Conference on Very Large Data Bases*, pages 413–424, Bombay, India, September 1996.

[11] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proc. of IEEE Int. Conf. on Data Engineering (ICDE)*, pages 251–260, Taipei, Taiwan, March 1995.

[12] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J.Ullman. A query translation scheme for rapid implementation of wrappers. In *Proc. Int. Conf. on Deductive and Object-Oriented Databases (DOOD)*, pages 97–107, Singapore, December 1995.

[13] A. Vercoustre, J. Dell'Oro, and B. Hills. Reuse of information through virtual documents. In *Second Australian Document Computing Symposium*, pages 55–64, Melbourne, Australia, April 1997.