Tools for Data Translation and Integration *

Serge Abiteboul† Sophie Cluet† Tova Milo‡ Pini Mogilevsky‡ Jerome Siméon†
Sagit Zohar‡

Abstract

A broad spectrum of data is available on the Web in distinct heterogeneous sources, stored under different formats. As the number of systems that utilize this data grows, the importance of data conversion mechanisms increases greatly. We present here an overview of a French-Israeli research project aimed at developing tools to simplify the intricate task of data translation. The solution is based on a middleware data model *to which various data sources are mapped, and a* declarative language *for specifying translations within the middleware model. A complementary schema-based mechanism is used to automate some of the translation. Some particular aspects of the solution are detailed in [3, 7, 10].*

1 Introduction

Data integration and translation is a problem facing many organizations that wish to utilize Web data. A broad spectrum of data is available on the Web in distinct heterogeneous sources, stored under different formats: a specific database vendor format, XML or Latex (documents), DX formats (scientific data), Step (CAD/CAM data), etc. Their integration is a very active field of research and development, (see for instance, for a very small sample, [4, 5, 9, 6]). A key observation is that, often, the application programs used by organizations can only handle data of a specific format. (e.g. Web browsers, like Netscape, expect files in HTML format, and relational databases expect relations). To enable a specific tool to manipulate data coming from various sources (e.g. use, in a relational system, data stored on the Web in HTML format), a translation phase must take place – the data (in the source format) needs to be mapped to the format expected by the application.

The naive way to translate data from one format to another is writing a specific program for each translation task. Examples are the Latex to HTML and HTML to text translators [8, 1]. Writing such a program is typically a non trivial task, often complicated by numerous technical aspects of the specific data sources that are not really relevant to the translation process (e.g. HTML/XML parsing, or specific DB access protocol). A sound solution for a data integration task requires a clean abstraction of the different formats in which data are stored, and means for specifying the correspondences between data in different worlds and for translating data from one world to another. For that, we introduced a *middleware* data model that serves as a basis for the integration task, and *declarative rule languages* for specifying the integration. Translation from source to target formats is achieved by (1) importing data from the sources to the middleware model, (2) translating it to another middleware representation that better fits the target structure, and then (3) exporting the translated data to the target system.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

* Work partially supported by AFIRST and the Israeli Ministry of Science.
† I.N.R.I.A, Rocquencourt, France Email: <firstname>.<lastname>@inria.fr
‡ Tel Aviv University, Israel, Email: {milo,pinim,sagit}@math.tau.ac.il

A first attempt: Our first attempt (to be refined later on) was to propose a very simple middleware model where data is represented by ordered labeled trees. The model is very similar to the common OEM model for semistructured data. A particularity here is that we allow an order to be defined on the children of nodes. Order is an inherent component of some data structures, e.g. ordered tuples and lists. Similarly, textual data can either be described as a sequence of characters or words, or on a higher level by a certain parse tree; in both cases, the order of data element is important. Supporting order as part of the data model enables a natural representation of data coming from such sources [3]. Together with the data model, we introduced a declarative rule-based language for describing correspondences between data in different worlds [3]. The language has several useful properties: it provides constructs to search and build complex trees that represent data; the semantics of rules take into consideration the fact that nodes may represent collections with specific properties (e.g., sets are insensitive to order and duplicates); the complexity of correspondence computation was proved to be PTIME in many practical cases. Of course, deriving correspondences within existing data is only one issue in a heterogeneous context. One would also want to translate data from one representation to another. Interestingly, we showed that in many practical cases, translation rules can automatically be derived from the correspondence rules. Thus, a complete integration task (derivation of correspondences, transformation of data from one world to the other, incremental integration of a new bulk of data, etc.) can be specified using a *single* set of declarative rules. This was an important result. It simplifies the specification task and helps in preventing inconsistencies in specifications.

To experiment with the above translation scheme we built a prototype system, W3TRANS [11], and used it to define mapping and translations among various formats including SGML, HTML, relational and OODBs. We soon realized that although this simplified greatly the translation specification, still, very often, a considerable programming effort was required when a new translation was to be defined. Our goal was thus to design additional mechanisms to further simplify the translation. The solution was based on two main observations.

(i) Instantiation: In many cases, the translations needed by the user are combinations or specialization of other translations. For example consider the following application. A research institute wants to build an intranet application. Among other things, the institute stores informations about its researchers in an OODB, and keeps their research papers in XML documents. Assume we want to provide an HTML interface to the system so that employees can view it on the Web. If the system happens to contain two generic programs providing an HTML view of OO and XML data resp., then the specific application can be obtained by customizing and combining the above programs. Following this idea we enhanced our model by a novel *instantiation* mechanism, extending both the data model and the rule-based translation language and providing a powerful mechanism for customization and synthesis of programs. Interestingly, both the model and the rules have a natural graphical representation which further simplifies the programming by allowing a convenient graphical interface via which the user can, with minimal programming, customize and combine existing programs.

(ii) Schema-based translation: Frequently, much of the structure of the source data is similar to that of the target translated data, and many of the structure modifications to be performed by the translation process are rather standard and result from various differences between the schemas of the source and target systems. We use here the general term *schema* to denote whatever way a data model chooses to model its data. For example, databases use schemas to model database instances; Structured documents often obey some grammar (e.g. Document Type Definition – DTD – in SGML and XML); In other models such a definition may be partial (e.g. in semistructured data [2]). The observation is that, in many translations, the schema of the target system is closely related to that of the source system – both schemas aim to represent the same data. This implies that a large part of the translation can be done *automatically*, relying on this (often standard) relationship, hence reducing the programming effort, and involving the user only in the specification of the "non standard" parts of the translation.

Based on the above observations we have developed two complementary prototype systems, YAT and TranScm. YAT focuses on the rule language and implements the instantiation and customization mechanisms. TranScm focuses on schema analysis and automatic generation of translations when possible. Both support a convenient graphical interface and together provide a sound solution to the data translation problem. In the rest of this

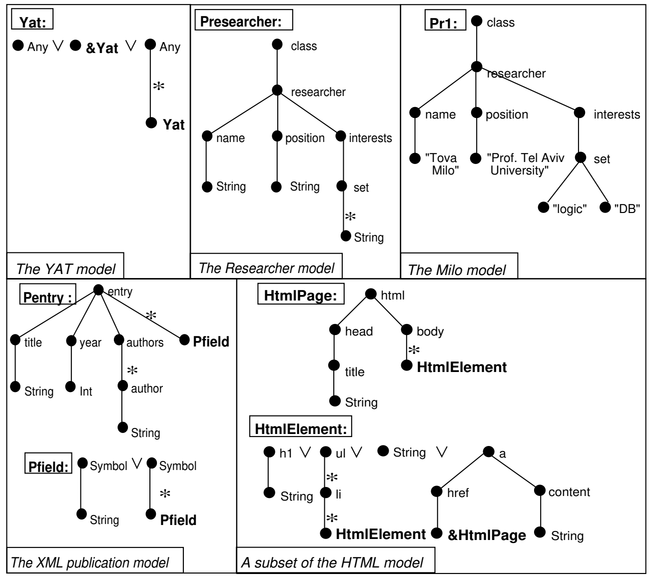

Figure 1: Some YAT Models

paper we briefly describe the main ideas underlying the two systems. For lack of space the presentation is rather informal and is mainly based on examples and intuitive explanations. For full details see [7, 10].

2 YAT

We explain below the main ideas using a running example - the research institute repository mentioned above.

The 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 real-world data (OODB, 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 1 shows several such models. The three models on the upper part of the figure may be used to represent an OODB 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 an OODB 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., $\boldsymbol{Yat}$) or (v) reference to a pattern name (e.g., $\boldsymbol{\&Yat}$). The edges of a YAT tree may be annotated 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:

$\boldsymbol{HtmlPageAll}()$ **:**
$html\langle \rightarrow head \rightarrow title \rightarrow$ "Verso Researchers",
$\rightarrow body\langle \rightarrow ul \xrightarrow{\star} li \rightarrow a$
$\langle \rightarrow href \rightarrow \&\boldsymbol{HtmlPage}(N),$
$\rightarrow content \rightarrow N\rangle\rangle\rangle$
$\Longleftarrow$
$\boldsymbol{Presearcher}$ **:**
$class \rightarrow researcher$
$\langle \rightarrow name \rightarrow N,$
$\rightarrow position \rightarrow P,$
$\rightarrow interests \rightarrow set \xrightarrow{\star} I\rangle$

Figure 2: Rule **All Researchers**

$\boldsymbol{HtmlPage}(N)$ **:**
$html\langle \rightarrow head \rightarrow title \rightarrow H1,$
$\rightarrow body\langle \rightarrow h1 \rightarrow H1,$
$\rightarrow$ "Position is ",
$\rightarrow P$
$\rightarrow ul \xrightarrow{[]_T} li\langle \rightarrow$ "title : ",
$\rightarrow T,$
$\rightarrow$ "published in ",
$\rightarrow Y\rangle\rangle\rangle$
$\Longleftarrow$
$\boldsymbol{Pentry}$ **:**
$entry\langle \rightarrow title \rightarrow T,$
$\rightarrow year \rightarrow Y,$
$\rightarrow authors \xrightarrow{\star} set \rightarrow N,$
$\xrightarrow{\star} \boldsymbol{Pfield}\rangle,$
$\boldsymbol{Presearcher}$ **:**
$class \rightarrow researcher$
$\langle \rightarrow name \rightarrow N,$
$\rightarrow position \rightarrow P,$
$\rightarrow interests \rightarrow set \xrightarrow{\star} I\rangle,$
$H1$ `is` concat("Homepage of ", N)

Figure 3: Rule **Single Researcher**

- A constant/symbol node can be instantiated by nodes with an identical label (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 $\boldsymbol{Yat}$ 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 $\&\boldsymbol{HtmlPage}$ labeled node is instance of the $\&\boldsymbol{Yat}$ labeled node.

Defining translations: Now that we are able to describe the structural input (OODB and XML) and output (HTML) of our application, we need a way to map the former to the later. This is done using the translation language YATL (either directly or through its graphical interface) that allows to describe complex conversions on graphs. It is rule-based, each rule describing a small part of the data conversion. For space reasons we will not give the complete program allowing to map researcher objects 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 already existing in 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 [7]. We now illustrate the YATL language on two simplified rules of the example application.

Consider the rule **All Researchers** on Figure 2. 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 $\boldsymbol{Presearcher}$ 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: $\boldsymbol{HtmlPageAll}()$.

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 ($\&\boldsymbol{HtmlPage}(N)$). Note also that we could have used the researcher's identity as a parameter, replacing N by $\boldsymbol{Presearcher}$ 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 Researcher** given in Figure 3.

This rule computes the page associated to one researcher identified by its name. It integrates data from the OODB and the XML files. Its input is a set of patterns, instances of the $\boldsymbol{Pentry}$ and the $\boldsymbol{Presearcher}$ patterns. Its output is a new HTML page identified by the $\boldsymbol{HtmlPage}(N)$ Skolem function. Remark that Variable N is used in both $\boldsymbol{Pentry}$ and $\boldsymbol{Presearcher}$ 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 $\stackrel{[]_T}{\rightarrow}$ 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 can be used for example to give several views of a given list of publications (ordered by title, by publication date, by authors, etc).

Just like the data model, translation rules can also be instantiated and customized for specific needs. The programmer can start with a general program that can be instantiated into one that corresponds to the given input. Customization can follow. For example, through successive instantiations/customizations, we can find programs generating HTML pages from (i) any input, (ii) an arbitrary OODB (iii) data corresponding to a specific OODB schema or (iv) an object of that schema. It is also possible to compose programs and efficiently build complex translations out of simple, previously defined blocks. We omit this here for lack of space.

3 TranScm

To further simplify the programming, we observe that in many cases the schema of the data in the source system is very similar to that of the target system. In such cases, much of the translation work can be done automatically, based on the schemas similarity. This can save a lot of effort for the user, limiting the amount of programming needed. The TranScm system implements the above idea. Given the schemas for the source and target data, the system examines the schemas and tries to find similarities/differences between them. This is done using a rule-based method, where each rule (1) defines a possible common matching between two schema components, and (2) provides means for translating an instance of the first to an instance of the second. The system has a set of build-in rules that handles most common cases, and that can be extended/adjusted/overridden by the user during the translation process. The system uses the rules and tries to find for each component of the source schema a unique "best matching" component in the target schema, or determine that the component should not be represented in the target. This is called the *matching process*. If the process succeeds, the data translation can be performed automatically using the translation facilities of the matching rules. There are two cases where the process may fail: (i) a component of the source schema cannot be matched with a target one using the current set of rules, (and the matching process can neither derive that the component should be just ignored), or (ii) a component of the source schema matches several components in the target schema, and the system cannot automatically determine which is the "best" match. For (i) the user can add rules to the system to handle the special component and describe the translation to be applied to it. For (ii) the user is asked to determine the best match. Then, based on the user's input, the matching process is completed and the translation is enabled.

Handling schemas of different sources requires a common framework in which they can be presented and compared. Note however that the middleware model presented above, with its instantiation mechanism, allows to present not only data but also schema, hence can serve as such common framework. (In the actual implementation of the system we used a slightly more refined model, but the differences are irrelevant for the purpose of the current paper). Similarly, the match&translate rules used by the system could be defined in the style of YATL. (In fact, the system provides a generic interface for rules, independent of the specific language used to specify them. This enabled us for example to use in the prototype Java as the rule definition language.)

A typical scenario of the system's work is as follows. It receives as input two schemas, one of the data source and the other of the target. The two schemas are imported into the system and presented in the common schema model. The next step is matching. The system tries to find for every component in the source schema a corresponding component in the target schema (or determine that the component should not be represented in the target), using the rule-based process mentioned above. Once the matching is completed (perhaps with the user's assistance), a data translation is possible. To perform the translation, a data instance of the source schema is imported to the common data model, and is "typed", i.e. every element in the data is attached a corresponding schema element as a type. Now, the system uses the match between the schema components, achieved in the previous step, to translate the data. Recall, from above, that every rule in the system has two components, the first defines a possible common matching between two components of schemas, and the second provides means for translating an instance of the first to an instance of the second. Every element of the source data is translated, using the translation function of the rule that matched its type with a type (component) of the target schema, to an instance of the target type. The resulting elements are "glued" together to form a valid instance of the target schema. Finally, the translated data is exported to the target application.

To conclude, note that the schema-based translation method is not aimed at *replacing* the YATL programming language, but rather to *complement* it - rather than writing a translation program for all the data, much of the translation will be done automatically by the system, based on the schema matching, and the programmer needs to supply only some minimal additional code handling the data components not covered by the system.

References

[1] The html2text translator, 1996. Available by from `http://www.scanline.com/html2txt/`.

[2] S. Abiteboul. Querying semi-structured data. In *Proc. ICDT 97*, pages 1–18, 1997.

[3] S. Abiteboul, S. Cluet, and T. Milo. Correspondence and translation for heterogeneous data. In *Proc. ICDT 97*, pages 351–363, 1997.

[4] P. Buneman, S. 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, 1996.

[5] M.J. Carey et al. Towards heterogeneous multimedia information systems : The Garlic approach. Technical Report RJ 9911, IBM Almaden Research Center, 1994.

[6] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Proceedings of IPSJ Conference*, pages 7–18, Tokyo, Japan, October 1994.

[7] S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your mediators need data conversion! In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 177–188, 1998.

[8] Nikos Drakos. The latex2html translator, 1996.
Available from `cbl.leeds.ac.uk/nikos/tex2html/doc/latex2html.ps`.

[9] A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 251–262, 1996.

[10] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 122–133, 1998.

[11] Pini Mogilevsky. W3trans translation system, 1996.
Available from `www.math.tau.ac.il/pinim/w3trans-home.html`.