HePToX: Marrying XML and Heterogeneity in Your P2P Databases

Angela Bonifati
Icar CNR, Italy
bonifati@icar.cnr.it

Elaine Qing Chang **Terence Ho**
Laks V.S. Lakshmanan **Rachel Pottinger**
University of British Columbia, Canada
{echang, terenho, laks, rap}@cs.ubc.ca

Abstract

We present HePToX, a full-fledged peer-to-peer database system that efficiently handles XML data heterogeneity. In a highly dynamic P2P network, it is unrealistic for a peer entering the network to be forced to agree on a global mediated schema, or to perform heavyweight operations for mapping its schema to neighboring schemas. In our demo, we show that to enter the HePToX network a peer user is only asked to draw a simple set of visual annotations to a few other schemas. We show how the mapping rules are then automatically generated and how efficient query translation is performed on top of these mappings.

1 Introduction

Peer-to-peer (P2P) systems allow data sources to share their data with other sources. These sources become peers in a P2P network, where each peer maintains its own data, but gains the ability to read the other peers' data. As in a data integration system, the pre-existing usage of the sources' data causes peers to wish to maintain their own schema; this causes a problem of *schema heterogeneity*. In a data integration system, schema heterogeneity is resolved by a mediated global schema. However, the lack of a centralized authority and the ad-hoc membership of the P2P network renders this infeasibility in a P2P system. Instead, each peer that enters the P2P network provides a *mapping* between itself and some subset of the peers already in the network.

Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

In this demo, we present HePToX (pronounced Hep Talk), a heterogeneous XML P2P database system which uses lightweight Datalog-like schema mappings exhibiting a data-exchange semantics. In HePToX, we only expect a peer user to draw a simple set of *arrows and boxes* and the system supplies the rest. HePToX uses a novel algorithm which automatically infers Datalog-like mappings between arbitrary XML database schemas, from the correspondences.

Previous research has focused either on frameworks which are either too lightweight to express complex XML mappings, or require heavyweight mapping creation from the user. For example, value mappings, such as those used in [10], are simple and lightweight, but insufficient to express complex mappings between XML DTDs. HePToX permits easy creation of sophisticated mappings without requiring heavyweight view definitions unlike previous works such as [1, 2, 4, 7, 11].

To give a flavor of the visual annotations, Figure 1 graphically depicts schemas of two distinct hospital peers. This example is adapted from [9]. For the moment, we ignore the arrows. The first DTD is for `Montreal (General) Hospital` database. Every `Patient` is assigned a unique hospital `ID` and has a unique Medicare number (`MedCr#`). The `Patient` elements contain sub-trees corresponding to their history of health problems (`Hist`) and treatments (`Treat`), while `Admission` data is maintained separately and is captured via the ID/IDREF link `@PatRef` → `@ID` (shown as a solid grey arrow in Figure 1(a)).

If a patient is transferred from Montreal to Boston, that patient's data must then be able to be queried by the `Boston Hospital` schema. The Boston Hospital schema is organized quite differently; it is grouped into an `Admission` complaint (e.g., pulmonary, or coronary) and `Progress`. The usual patient details are stored under the `Admission` complaints. The `Progress` sub-tree records the patients' `Symptoms` and the `Treatments`, and is connected to the patients via the ID/IDREF link `@PatRef` → `@ID`.

Patients often make such transfers, which requires

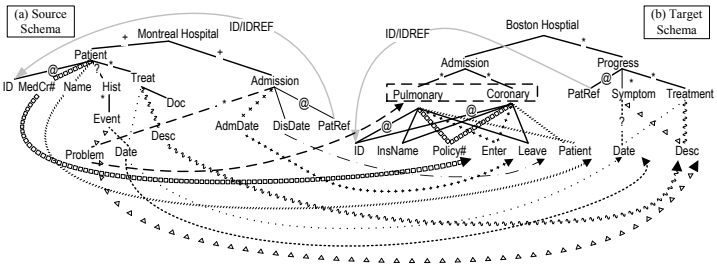

Figure 1: Mapping two Heterogeneous Peer DTDs. Every unlabeled edge is labeled '1' by default.

their data to travel with them. Given that the transfers do not always occur between a well-defined set of hospitals, and there is no hospital or other source that could be relied upon to create a mediated schema, a *P2P database system* for health care is a natural choice. A P2P database system is a collection of autonomous database systems connected by a P2P network, and peers may enter or leave the network at will.

Any peer entering the network can establish schema relationships with other peers. The peers chosen by the entering peer are technically called *acquaintances* [1]. Coming back to our example, the `Montreal Hospital` peer may choose `Boston Hospital` as an acquaintance and provide correspondences between the concepts in its schema and those in the `Boston Hospital` schema.

We require the peer database administrator or the final user to supply only very simple correspondences. The correspondences are specified in the form of arrows and boxes, illustrated in various dashed lines, in Figure 1, and will be explained in Section 2. We use the correspondences as a basis for automatically inferring a *mapping expression*, expressed in the form of Datalog-like rules. The peer DBA can examine the rules and make any necessary adjustments. These correspondences can either be created by hand or through some (semi-)automatic mapping discovery algorithm (see [5] for a survey of such approaches).

There may be considerable differences in the way the peers organize their data, including differences in data representations (e.g., stock names instead of ticker symbols, different units, etc.), as well as differences in underlying schemas (e.g., group treatments under the patients receiving them as opposed to maintaining separate lists of patients and treatments and linking them with key-foreign key links).

Our further goal is to permit users and applications of any peer database to access data of interest by simply posing a query to their peer, regardless of the location of the data items or the schema under which they are organized. In other words, the existence of numerous peers and their schemas should all be transparent to the user/application posing the query. E.g., for answering the query "what are the treatments administered to patients admitted with a coronary illness?", we want to manipulate data from all peers "visible" to the original peer, containing logically relevant information. Clearly, queries posed to a peer need to be translated appropriately so as to run on other peers.

2 A Motivating Example

We now explore the example discussed in the introduction (Figure 1) in more depth. The arrows provide informal correspondences between the two hospital DTDs, and are illustrated by dashed lines to differentiate them from the structure of the DTDs. Henceforth we refer to the Montreal Hospital DTD as Montreal and the Boston Hospital DTD as Boston. The arrows capture simple 1-1 correspondences between terms such as "`MedCr#` in the first DTD to `Policy#` in the second" and "`Name` in the first to `Patient` in the second"; we do not consider 1-n or m-n correspondences in this paper.

Consider the correspondence between `Desc` in the two DTDs. `Desc` has a unique parent in first DTD while it has two parents in the second. For disambiguation, we use the same line style for the edges `Treat/Desc` in Montreal, `Treatment/Desc` in Boston, and the arrow connecting them. Other arrows can be understood similarly. In the only more complicated mapping in this figure, consider the dashed box enclosing the `Pulmonary` and `Coronary` nodes in (b). Boxes are used to group together tags of nodes in a DTD that correspond to instances of a tag in another DTD, as part of the correspondence specification. For example, the dashed arrow matching `Admission/Problem` to this box says that 'Pulmonary' and 'Coronary' correspond to values of the element `Admission/Problem` in the first database. In effect, illnesses, which may be instances of `Admission/Problem` in the first database, correspond to tags in the second database. However, this correspondence makes no assumption that the set of illnesses occurring in the two databases are the same or even overlap.

Arrows and boxes in and of themselves do not tell us how a database that conforms to a DTD may be transformed to one that conforms to the other DTD. This is relevant because it is closely tied to the semantics of query answering. HePToX lets the user perform the whole process, from the creation of the peer-to-peer mappings to the translating of queries. In [3] we describe *both* how to infer mapping expressions from

the correspondences specified using the boxes and arrows *and* how queries are translated over the inferred mappings. We also show this in the demo. For space reasons, here we just informally explain the mapping semantics by examining the two DTDs in Figure 1.

As reasoned above, HePToX's mapping expressions define the data exchange semantics of heterogeneous data transformation. Even in this simple example, the structure of the two schemas is quite different. For example, in the Montreal DTD, patients' history and treatments are both nested under the patient. All admission information is maintained separately and linked to the appropriate patient via the patient's ID. In the Boston database, treatment and history information (symptoms) is separated out from patients and linked to them via their ID. Additionally, patients are represented along with the rest of the admission data, but this data is classified based on the type of problem/illness identified at the time of admission.

In this demo, we show how to produce the mapping expressions shown in Figure 2 given only the simple correspondences shown in Figure 1, and then how to translate queries from one DTD to the other. The mappings are expressed as Datalog-like rules, (⟨rule head⟩ ⟵ ⟨rule body⟩), adapted for tree structured data and explained next.

1. $\texttt{Boston} \rightarrow f1(\$Montreal)[\texttt{Admission} \rightarrow f2(\$Montreal)$
$[\$AP/text() \rightarrow f3(\$AP/text(), \$ID, \$M, \$AD, \$DD, \$N)$
$[\texttt{@ID} \rightarrow \$ID, \texttt{Policy\#} \rightarrow \$M,$
$\texttt{Enter} \rightarrow \$AD, \texttt{Leave} \rightarrow \$DD, \texttt{Patient} \rightarrow \$N]]]$
$\longleftarrow$
$\texttt{Montreal} \rightarrow \$Montreal[\texttt{Patient} \rightarrow \$P$
$[\texttt{@ID} \rightarrow \$ID, \texttt{MedCr\#} \rightarrow \$M, \texttt{Name} \rightarrow \$N],$
$\texttt{Admission} \rightarrow \$A[\texttt{Problem} \rightarrow \$AP, \texttt{AdmDate} \rightarrow \$AD,$
$\texttt{DisDate} \rightarrow \$DD, \texttt{@PatRef} \rightarrow \$PR]], \$PR = \$ID.$

2. $\texttt{Boston} \rightarrow f1(\$Montreal)[\texttt{Progress} \rightarrow f2(\$PR)$
$[\texttt{Symptom} \rightarrow f3(\$EP, \$ED)[\texttt{Date} \rightarrow \$ED, \texttt{Desc} \rightarrow \$EP]]]$
$\longleftarrow$
$\texttt{Montreal} \rightarrow \$Montreal[\texttt{Patient} \rightarrow \$P$
$[\texttt{@ID} \rightarrow \$ID, \texttt{MedCr\#} \rightarrow \$M, \texttt{Name} \rightarrow \$N, \texttt{Hist} \rightarrow \$H$
$[\texttt{Event} \rightarrow \$E[\texttt{Problem} \rightarrow \$EP, \texttt{Date} \rightarrow \$ED]]]$
$\texttt{Admission} \rightarrow \$A[\texttt{Problem} \rightarrow \$AP, \texttt{AdmDate} \rightarrow \$AD,$
$\texttt{DisDate} \rightarrow \$DD, \texttt{@PatRef} \rightarrow \$PR]], \$PR = \$ID.$

3. $\texttt{Boston} \rightarrow f1(\$Montreal)[\texttt{Progress} \rightarrow f2(\$PR)$
$[\texttt{Treatment} \rightarrow f3(\$TDate, \$TDesc)$
$[\texttt{Date} \rightarrow \$TDate, \texttt{Desc} \rightarrow \$TDesc]]]$
$\longleftarrow$
$\texttt{Montreal} \rightarrow \$Montreal[\texttt{Patient} \rightarrow \$P$
$[\texttt{@ID} \rightarrow \$ID, \texttt{MedCr\#} \rightarrow \$M, \texttt{Name} \rightarrow \$N,$
$\texttt{Treat} \rightarrow \$T[\texttt{Date} \rightarrow \$TDate, \texttt{Desc} \rightarrow \$TDesc]],$
$\texttt{Admission} \rightarrow \$A[\texttt{Problem} \rightarrow \$AP, \texttt{AdmDate} \rightarrow \$AD,$
$\texttt{DisDate} \rightarrow \$DD, \texttt{@PatRef} \rightarrow \$PR]], \$PR = \$ID.$

4. $\texttt{Boston} \rightarrow f1(\$Montreal)[\texttt{Progress} \rightarrow f2(\$PR)[\texttt{@PatRef} \rightarrow \$PR]]$
$\longleftarrow$
$\texttt{Montreal} \rightarrow \$Montreal[\texttt{Admission} \rightarrow \$A[\texttt{Problem} \rightarrow \$AP,$
$\texttt{AdmDate} \rightarrow \$AD, \texttt{DisDate} \rightarrow \$DD, \texttt{@PatRef} \rightarrow \$PR]$

Figure 2: Mapping Rules between schemas in Figure 1.

2.1 HePToX Mapping Expression Language

HePToX uses a mapping expression language in which the mapping rules between peer schemas that it generates are specified. Each rule is made up of *atoms* of the form $\texttt{Tag} \rightarrow id$, where Tag is a tag or a tag variable and id is the id associated with a node with this tag. Here, id may be a variable or any term of the form $f(\$v_1, ..., \$v_n)$, for some variables $\$v_i$ and some Skolem function f. Skolem functions are used not only for creating new node id's (as has often been done in the literature), but also for expressing group-bys, as we explain shortly. Atoms can be nested inside other atoms, thus expressing nesting, while a comma-separated list of atoms expresses the sub-elements of a given element. Attributes are preceded with a '@'.

Before we explain the meaning of the rules, it's important to bear in mind that the rules and the mapping do *not physically* transform data from one source's schema to another. As in [1], these mapping rules rather express the semantics of data exchange – if data were to be exchanged from source 1 to 2, how it would correspond to the schema of source 2.

Atoms can be nested to form *tree expressions*. Tree expressions are either atoms ($t \rightarrow i$) or are of the form $t \rightarrow i[TE_1, ..., TE_k]$, where $t \rightarrow i$ is an atom and TE_i are tree expressions, as shown in Figure 2.

2.2 Creating the Rules

One of the major contributions of HePToX is to automatically generate the rules from the correspondences. In this section we give an intuitive notion of how the rules are created. In the demo, we show the actual algorithm that HePToX uses to create these rules.

Rule 1 says corresponding to the (unique) root of the Montreal source, there is a (unique) root in Boston. The uniqueness of the latter follows from applying the Skolem function $f1$ to the node id associated with the former's root, which is unique. Similarly, there is a unique Admission node in Boston, and we have used again the root of Montreal as the argument of the Skolem function $f2$ for capturing this uniqueness. The rule body binds the variable $\$AP$ to the problem of a patient at admission time. $\$AP/text()$ extracts the text value associated with node $\$AP$. This value (which may have a value from the same domain as "Pulmonary' and 'Coronary'[1]) is used to form the tag of a new node, whose id is $f3(\$AP/text(), \$ID, \$M, \$AD, \$DD, \$N)$, i.e., it is a function of the patient's admission time problem ($\$AP/text()$), id ($\$ID$), insurance policy (or medicare) number ($\$M$), admission and discharge dates ($\$AD, \$DD$), and name ($\$N$). *Note that the arguments of the Skolem function $f3$ are exactly the single-valued sub-elements of the* Pulmonary *and* Coronary *elements in Boston.* HePToX does not assume any knowledge of keys, and can still work correctly in the absence of such knowledge.

As expected, patient id, policy number, admission date, discharge date, and name are all matched to their counterparts in Boston.

Rule 2 maps the patient history consisting of

[1]The value need *not* be one of these.

`Problem`s and their `Date`s of occurrence (nested in Montreal through `Hist/Event`) to `Symptom/Desc` and `Symptom/Date` in Boston. Note that in Boston, the `Symptom` elements are nested inside a `Progress` element, which has as its id a function of the patient ID (via `@PatRef`), i.e., $f2(\$PR), \$PR = \$ID$. Thus, there is one `Progress` element per patient. Consequently, `Symptom`s are grouped by patient ID. The node ID $f3(\$EP, \$ED)$ used for `Symptom` elements shows that for each occurrence of a problem for a given patient, a separate `Symptom` element is created.

Rule 3 maps treatment information from Montreal to Boston. `Progress` elements are created with id $f2(\$PR)$ just as they are in rule 2. Note that the use of the node id $f3(\$TDate, \$TDesc)$ for `Treatment` ensures that for every treatment on any date administered to a given patient, the corresponding `Treatment` element is nested inside the `Progress` element associated with the patient.

A node id plays a key role: for instance, `Progress` elements are created by *both* rules 2 and 3. HePToX ensures that no duplicate `Progress` elements are created and that both of the sub-elements created by rules 2 and 3 are included by matching up the node ids. This is achieved as follows: whenever the id of a `Progress` node created by rule 2 matches the one created by rule 3, they refer to one and the same node. For instance, suppose $'p5'$ is the ID value of a patient, then the subtree rooted at the `Progress` node $f2(p5)$ created by rule 2 and the subtree rooted at the `Progress` node $f2(p5)$ created by rule 3 are both glued at the node $f2(p5)$. More generally, whenever subtrees are created by applications of the same or different rules, conceptually all these subtrees are glued together at nodes having a common node id. This ensures that the pieces "computed" by rules are correctly glued together.

Finally, rule 4 maps `@PatRef` attribute in Montreal to `@PatRef` attribute in Boston.

2.3 Translating queries

Even once the mapping has been created as shown in 2.2, translating the queries themselves can be difficult if not impossible [8, 6]. In the demo we show how to translate queries simply and efficiently. Rather than using a view-based rewriting scheme, HePToX uses a different mechanism which allows it to be very efficient over a simple but substantial subset of XQuery.

3 HePToX Demonstration Specifics

We demonstrate how a set of heterogeneous schemas and the corresponding data can be managed by the HePToX system. A set of physical peers which have variants of XML schemas of some real and synthetic data sets are simulated by Emulab, a P2P network emulation testbed. Our GUI lets a generic peer user draw a set of annotations between its schema and any other schema on other peers. We then enable the generation of mapping rules from the visual annotations. In particular, we show how different variants of the schemas above can be mapped to one another by leveraging the convenience of the HePToX end-to-end mapping tool.

Furthermore, once correspondences are created among these schemas, we let the peer user pose queries against that peer's own schema and let these queries be translated across the correspondences. Here, we highlight the correctness and efficiency of the translation process for a simple yet substantial XQuery fragment.

About the Demonstration Scenarios. In order to show the effectiveness of our system on Health Care datasets, such as those illustrated in Figure 1, we envision, among the others, the following list of users:

- `a patient` who moves from one hospital to another willing to transmit her own data to the destination hospital, *whatever the latter is*;
- `a doctor` who transmits his patient records to *a set of hospitals* in the network, for instance in order to ask suggestions to her colleagues on a difficult surgery;
- `the insurance company` which is interested to the policy of *a set of patients*, willing to query heterogeneous hospital databases;
- `the health care institute director`, who wants to know how *many hospitals* were in the patient history before she joined her institute.

All the scenarios above require schema transformations across a network of peers, and the latter need to be flexible and lightweight to accomodate new users and new schemas, representing indeed new hospitals joining the network.

References

[1] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R.J. Miller, and J. Mylopoulos. The hyperion project: From data integration to data coordination. *Sigmod Record*, 32(3), 2003.

[2] A.Y.Halevy, Z.G.Ives, D.Suciu, and I.Tatarinov. Schema Mediation in Peer Data Management Systems . In *ICDE*, 2003.

[3] A. Bonifati, E.Q. Chang, T. Ho, and L.V.S. Lakshmanan. HEPTOX: Heterogeneous Peer to Peer XML Databases. In *CoRR cs.DB/0506002*, 2005.

[4] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Logical Foundations of Peer-To-Peer Data Integration. In *Proc. of ACM PODS*, 2004.

[5] E.Rahm and P.A.Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[6] R. Fagin, P.G. Kolaitis, L. Popa, and W. Tan. Composing Schema Mappings: Second-Order Dependencies to the Rescue. In *PODS*, 2004.

[7] A. Kementsietsidis, M. Arenas, and R.J. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *SIGMOD*, 2003.

[8] J. Madhavan and A.Y. Halevy. Composing Mappings Among Data Sources. In *VLDB*, 2003.

[9] P.Bernstein, F.Giunchiglia, A.Kementsietsidis, J.Mylopoulos, L.Serafini, and I.Zaihrayeu. Data Management for Peer-to-Peer Computing: A Vision. In *WebDB*, 2002.

[10] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating Web Data. In *VLDB*, 2002.

[11] I. Tatarinov and A.Y. Halevy. Efficient Query Reformulation in Peer-Data Management Systems. In *SIGMOD*, 2004.