Integration of Heterogeneous Data Sources

Harsh Sharma

Last Updated: May 23, 2014
List of Figures

1 TSIMMIS Components ........................................ 1
2 Object Exchange Model ........................................ 4
3 Answer for Query ............................................... 6
4 Home Page: MOBIE ............................................. 7
5 Fetch Result page ............................................... 7
6 Fetch with Title ................................................ 8
## Contents

1. **Introduction: TSIMMIS**  
   1.1 Goal of TSIMMIS  
   1.2 Philosophy  
   1.2.1 Translators and Common Model  
   1.3 Mediators  
   1.4 System and User Interface  
   1.5 Labels and Mediator Processing  
   1.6 Constraint Management  
   1.7 Classification and Extraction  
   1.8 Difference  

2. **OEM**  
   2.1 structure  
   2.2 OEM-QL  
   2.3 Implementation  

3. **Object browsing**
1 Introduction: TSIMMIS

TSIMMIS stands for "The Stanford-IBM Manager of Multiple Information Sources".

In Yiddish, tsimmis represents stew with heterogeneous fruits and vegetables integrated into a surprisingly tasty whole.

A common problem facing many organizations today is that of multiple, disparate information sources and repositories, including databases, object stores, knowledge bases, file systems, digital libraries, information retrieval systems, and electronic mail systems.

The information obtained can be inconsistent and contradictory.

1.1 Goal of TSIMMIS

- To provide tools for accessing, in an integrated fashion, multiple information sources, and to ensure that the information obtained is consistent

- To develop tools that facilitate the rapid integration of heterogeneous information sources that may include both structured and unstructured data.

1.2 Philosophy

1.2.1 Translators and Common Model

Above each source is a translator (or wrapper) that logically converts the underlying data objects to a common information model.

For performing logical translation, the translator converts queries over information in the common model into requests that the source can execute, and it converts the data returned by the source into the common model.

Figure 1: TSIMMIS Components
TSIMMIS has a simple self-describing (or tagged) object model. The common model used is called the **Object Exchange Model**, or OEM. OEM allows simple nesting of objects. The fundamental idea is that **all objects, and their subobjects, have labels that describe their meaning**. For example, the following object represents a Fahrenheit temperature of 80 degrees:

\[(\text{temp-in-Fahrenheit}, \text{int}, 80)\]

where the string "temp-in-Fahrenheit" is a human-readable label, "int" indicates an integer value, and "80" is the value itself.

In case of Complex Object:

For example, an object representing a set of two temperatures may look like:

\[(\text{set-of-temps}, \text{set}, \{\text{cmp 1}, \text{cmp 2}\})\]

\[\text{cmp 1} : (\text{temp-in-Fahrenheit}, \text{int}, 80)\]
\[\text{cmp 2} : (\text{temp-in-Celsius}, \text{int}, 20)\]

For requesting OEM objects a query language, OEM-QL, SQL-like language has been developed. It can deal with labels and object nesting.

### 1.3 Mediators

A mediator is a system that refines in some way information from one or more sources. A mediator embeds the knowledge that is necessary for processing a specific type of information.

When mediator is queried, it knows which sources are relevant for the query.

The mediator may also process answers before forwarding them to the user, say by converting dates to a common format, or by eliminating articles that duplicate information.

While the task of converting dates is probably straightforward, the task of eliminating duplicate information could be very complex, figuring out that two articles written by different authors say the same thing" requires real intelligence. In Tsimmis, a relatively simple mediator has been developed.

Goal of Tsimmis project is to automatically or semi-automatically generate mediators from high level descriptions of the information processing they need to do. This is illustrated by the **mediator generator box**.

Also there is **translator generator** that can generate OEM translators based on a description of the conversions that need to take place for queries received and results returned. This component significantly facilitates the task of implementing a new translator.

### 1.4 System and User Interface

- Mediators export an interface to their clients that is identical to that of translators.

- Both translators and mediators take as input OEM-QL queries and return OEM objects.
End users and mediators can obtain their information either from translators and/or other mediators.

End users (top of Figure 1) can access information either by writing applications that request OEM objects, or by using one of the generic browsing tools we have developed. Mosaic or World Wide Web.

The user writes a query on an interactive world-wide-web page, or selects a query from a menu.

The answer is received as a hypertext document.

The root of this document shows one or more levels of the answer object, with hypertext links available to take the user to portions of the answer that did not appear on the root document.

1.5 Labels and Mediator Processing

It is important to note that there is no global database schema, and that mediators can work independently.

When mediator passes subobjects to its client, it can append source name to attribute so that client can understand query better.

When multiple sources are involved, the mediator might convert the attributes from both sources into a common format.

Example:
NYTimes.author
AP.author

1.6 Constraint Management

Integrity constraints specify semantic consistency requirements over stored information such constraints arise even when the information resides in loosely coupled, heterogeneous systems.

Currently constraints monitored/enforced by human in adhoc way.

TSIMMIS Constraint Manager enforces constraints with weaker guarantees than what a centralized system may provide. Tsimmis makes "relaxed" guarantees.

The Tsimmis constraint manager supports the definition of the interfaces that a source supports for the information involved in a constraint (e.g., can a trigger be set on a data item?), specification of the desired constraint (e.g., two items should have the same value), and specification of the strategy that is to be followed for enforcing the constraint or for detecting violations.

The Local Constraint Managers are responsible for describing and supporting interfaces, while the Constraint Manager processes constraints and executes strategies.
1.7 Classification and Extraction

Many important information sources are completely unstructured, consisting of plain files or incoming bit strings.

Often it is possible to automatically classify the objects in such sources. The information collected by the Classifier/Extractor can then be exported (via a translator, if necessary) to the rest of the Tsimmis system, together with the raw data.

1.8 Difference

TSIMMIS assumes that information access and integration are intertwined.
Integration in this environment requires more human participation.

2 OEM

Object Exchange Model (OEM) is used as the unifying object model for information processed by Tsimmis components.

Information need not actually be stored using OEM, rather OEM is used for the processing of logical queries, and for providing results to the user.

2.1 structure

![Object Exchange Model]

Figure 2: Object Exchange Model

**Label**: A variable-length character string describing what the object represents. For each label a translator or mediator exports, it should provide a "help" page that describes (to a human) the meaning and use of the label. These help pages can be very useful during exploration of information sources, and for deciding how to integrate information.

**Type**: The data type of the object's value. Each type is either an atom (or basic) type (such as integer, string, real number, etc.), or the type set or list. The possible atom types are not fixed and may vary from information source to information source.

**Value**: A variable-length value for the object.

**Object-ID**: A unique 2 variable-length identifier for the object or (for null). The use of this eld is described below.

**Client Side**

(employee, set, { o 1 , o 2 , o 3 } )

- o 1 : location of ( name, str, ”some name” )
- o 2 : location of ( office, str, ”some office” )
- o 3 : location of ( photo, bitmap, ”some bits” )

However, for performance reasons, the translator may prefer not to copy all subobjects.
This indicates that the name and office subobjects can be found at memory locations o_1 and o_2, but the photo subobject must be explicitly retrieved using id 3.

(employee, set, { cmp_1, cmp_2, cmp_3 })

cmp_1 : ( name, str, "some name" )
cmp_2 : ( office, str, "some office" )
cmp_3 : ( photo, bits, "some bits" )

2.2 OEM-QL

To request OEM objects from an information source, a client issues queries in a language we refer to as OEM-QL. OEM-QL adapts existing SQL-like languages for object-oriented models to OEM.

Our first example retrieves the topic of each document for which "Ullman" is one of the authors:

```
SELECT bib.doc.topic
FROM Biblio
WHERE bib.doc.authors.author-ln = "Ullman";
```

Intuitively, the query’s WHERE clause finds all paths through the subobject structure with the sequence of labels bib, doc, authors, author-ln such that the object at the end of the path has value "Ullman." For each such path, the SELECT clause specifies that one component of the answer object is the object obtained by traversing the same path, except ending with label topic instead of labels authors, author-ln. Hence, for the portion of the object structure shown in Figure 2 the query returns:

```
(answer, set, o_1, o_2)
o_1 : ( topic, str, "Databases" )
o_2 : ( topic, str, "Algorithms" )
```
3 OBJECT BROWSING

SELECT bib.doc
FROM Biblio
WHERE bib.doc.authors.author-ln(a1) = ”Aho”
AND bib.doc.authors.author-ln(a2) = ”Hopcroft”
Finds all paths through the subobject structure with the sequence of labels [ bib, doc, authors ], and with two distinct path completions with label author-ln and with values ”Aho” and ”Hopcroft” respectively. The answer object contains one doc component for each such path.

2.3 Implementation

When the translator receives an OEM-QL query to evaluate, it converts the query into Folio’s Boolean retrieval language. Then it extracts the relevant information from the incoming screens and exports the information as an OEM answer object.

Implemented mediators that fuse information from multiple bibliographic sources.

OEM Support Libraries to facilitate the creation of future translators, mediators, and end-user interface.

3 Object browsing
Figure 4: Home Page: MOBIE

Figure 5: Fetch Result page
References