The TSIMMIS Project: Integration of Heterogeneous Information Sources*

Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, Jennifer Widom

Department of Computer Science
Stanford University
Stanford, CA 94305-2140

last-name@cs.stanford.edu

Abstract

The goal of the Tsimmis Project is to develop tools that facilitate the rapid integration of heterogeneous information sources that may include both structured and unstructured data. This paper gives an overview of the project, describing components that extract properties from unstructured objects, that translate information into a common object model, that combine information from several sources, that allow browsing of information, and that manage constraints across heterogeneous sites. Tsimmis is a joint project between Stanford and the IBM Almaden Research Center.

1 Overview

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. Decision makers often need information from multiple sources, but are unable to get and fuse the required information in a timely fashion due to the difficulties of accessing the different systems, and due to the fact that the information obtained can be inconsistent and contradictory.

*Research sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Material Command, USAF, under Grant Number F33615-93-1-1339. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of Wright Laboratory or the US Government. This work was also supported by the Reid and Polly Anderson Faculty Scholar Fund, the Center for Integrated Systems at Stanford University, and by Equipment Grants from Digital Equipment Corporation and IBM Corporation.

The goal of the TSIMMIS[1] project is to provide tools for accessing, in an integrated fashion, multiple information sources, and to ensure that the information obtained is consistent. Numerous other recent projects have similar goals, of course. Before describing the differences between Tsimmis and other data integration projects, let us give an overview of the Tsimmis architecture, describing the functions of the various components and the philosophy of our approach. Refer to Figure 1.

1.1 Translators and Common Model

Figure 1 shows a collection of (disk-shaped) heterogeneous information sources. Above each source is a *translator* (or *wrapper*) that logically converts the underlying data objects to a common information model. To do this 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.

For the Tsimmis project we have adopted a simple *self-describing* (or *tagged*) object model. Similar models have been in use for years; we call our version the *Object Exchange Model*, or *OEM*. OEM allows simple nesting of objects, and a complete specification is given in Section 2. 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:

$\langle$temp-in-Fahrenheit, int, 80$\rangle$

where the string "temp-in-Fahrenheit" is a human-readable label, "int" indicates an integer value, and "80" is the value itself. If we wish to represent a complex object, then each component of the object has its own label. For example, an object representing a set of two temperatures may look like:

[1] As an acronym, TSIMMIS stands for "The Stanford-IBM Manager of Multiple Information Sources." In addition, Tsimmis is a Yiddish word for a stew with "heterogeneous" fruits and vegetables integrated into a surprisingly tasty whole.

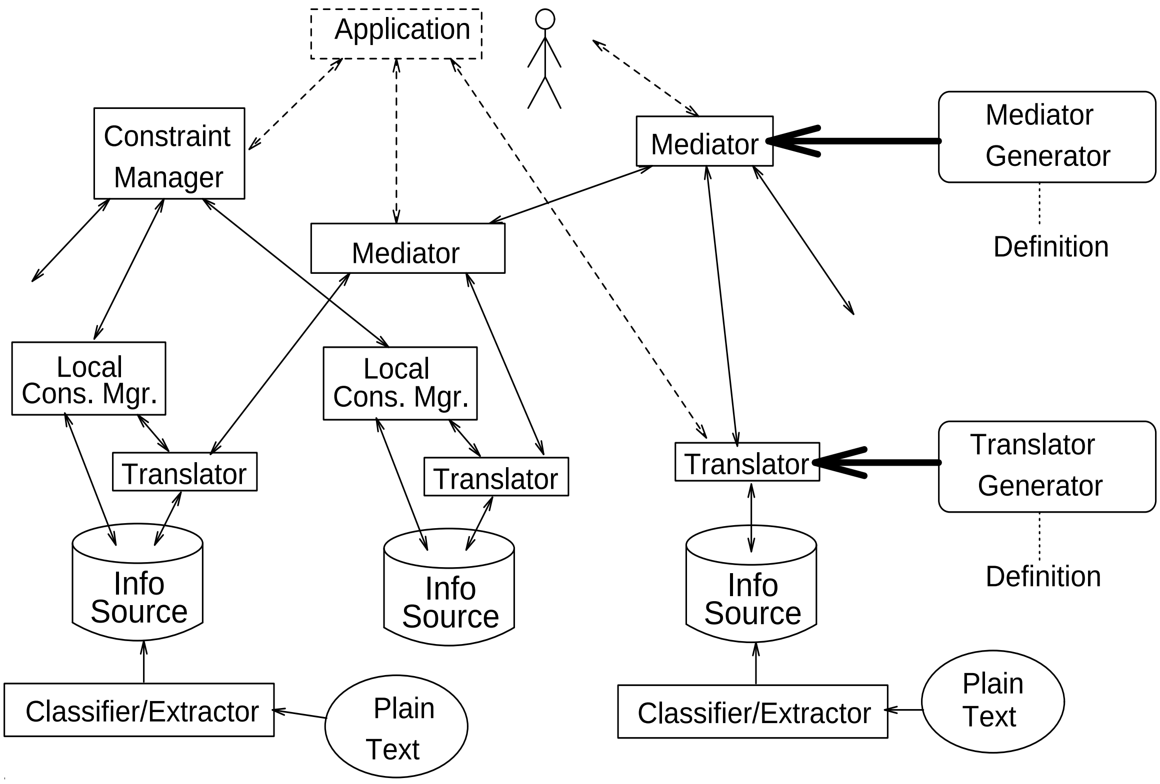

Figure 1: Tsimmis Architecture

⟨set-of-temps, set, {cmp_1, cmp_2}⟩
cmp_1: ⟨temp-in-Fahrenheit, int, 80⟩
cmp_2: ⟨temp-in-Celsius, int, 20⟩

OEM is very simple, while providing the expressive power and flexibility needed for integrating information from disparate sources. We also have developed a query language, *OEM-QL*, for requesting OEM objects. OEM-QL is an SQL-like language extended to deal with labels and object nesting; see Section 2.

1.2 Mediators

Above the translators in Figure 1 lie the *mediators*. A mediator is a system that refines in some way information from one or more sources [31]. A mediator embeds the knowledge that is necessary for processing a specific type of information. For example, a mediator for "current events" might know that relevant information sources are the AP Newswire and the New York Times database. When the mediator receives a query, say for articles on "Bosnia," it will know to forward the query to those sources. 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 we are focusing on relatively simple mediators based on patterns or rules. Still, even simple mediators can perform very useful information processing and merging tasks.

Implementing a mediator can be complicated and time-consuming, but we believe that much of the coding involved in mediators can be automated. Hence, one important goal of the 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 on the right side of Figure 1. Similarly, we provide a *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, also illustrated in Figure 1, significantly facilitates the task of implementing a new translator.

1.3 System and User Interfaces

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. Hence, end users and mediators can obtain their information either from translators and/or other mediators. This approach allows new sources to become useful as soon as a translator is supplied, it allows mediators to access new sources transparently, and it allows mediators to be "stacked," performing more and more processing and refinement of the relevant information.

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. Our most recent browsing tool provides access through *Mosaic* or other *World-Wide-Web* viewers [4,29]. 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. This tool provides a mechanism for exploring heterogeneous information sources that is easy to interact with and that is based on a commonly used interface. The browser is described in more detail in Section 3.

1.4 Labels and Mediator Processing

It is important to note that there is no global database schema, and that mediators can work independently. For instance, to build a mediator it is only necessary to understand the sources that the mediator will use. In fact, it is not even necessary to fully understand the sources used. For example, returning to our "current events" mediator, suppose one source exports objects with subobjects labeled by *title*, *date*, *author*, and *country*. The mediator might always pass the *author* and *country* subobjects to its client with no additional processing. Now suppose a second source provides *topic* and *date* subobjects. The mediator might convert the dates from both sources into a common format, and it will know how to convert a mediator query about the subject of an article into the appropriate *topic* or *title* queries to be sent to the sources.

When a mediator simply passes subobjects to its clients (as in *author* and *country* above), it might append the source name to the labels so that the client can interpret the objects correctly. For example, a mediator subobject might have label *NYTimes.author*, indicating that this author is from the New York Times source and follows its conventions for authors. Another object might have the label *AP.author*. (Of course, the mediator could also make the formats consistent and export subobjects with label *author*, but here we are illustrating a simple mediator that does not do such processing.)

The key points are that a mediator does not need to understand all of the data it handles, and no person or software component needs to have a global view of all the information handled by the system.

1.5 Constraint Management

Another important component in the Tsimmis architecture is constraint management, illustrated in Figure 1 by a *Constraint Manager* and two *Local Constraint Managers*. *Integrity constraints* specify semantic consistency requirements over stored information; such constraints arise even when the information resides in loosely coupled, heterogeneous systems. For example, a construction company keeps data about a building under construction. This data must be consistent with the architect's design (e.g., walls must be in the same places), which may be stored in an entirely different system. Constraint management in the distributed, heterogeneous environments addressed by Tsimmis is a much more difficult and complex problem than constraint management in centralized systems: Transactions across multiple information sources usually are not provided, and each information source may support different capabilities for accessing and monitoring the data involved in a constraint.

In current environments, constraints across heterogeneous information sources usually are monitored or enforced by humans, in an ad-hoc fashion (or, frequently, not checked at all). For example, an architect may freeze the building design and send the latest specifications to the construction company so that consistency is "guaranteed." Of course, it is clear that these ad-hoc mechanisms do not work well in general; in our example, it is likely that the building may eventually not meet its specifications.

Since in a loosely coupled environment it is generally not possible to guarantee that every user or application sees consistent data every time it interacts with the system, the Tsimmis constraint manager enforces constraints with weaker guarantees than what a centralized system may provide. Tsimmis makes "relaxed" guarantees, e.g., a constraint is true from 8am to 5pm every day, or a constraint is true if some "Flag" is set. Ensuring relaxed consistency is especially challenging because one now has to deal with the timing of actions and of guarantees. However, the advantages of being able to handle relaxed guarantees in heterogeneous systems are significant; knowing precisely what holds and what does not hold, and when, will clearly lead to more trustworthy systems.

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 in Figure 1 are responsible for describing and supporting interfaces, while the Constraint Manager processes constraints and executes strategies. Note that the Constraint Manager actually is not centralized as illustrated in Figure 1, but rather is a set of distributed components that jointly manage constraints. Constraint management is described in more detail in Section 4.

1.6 Classification and Extraction

The final component of the Tsimmis architecture consists of the *Classifier/Extractors* shown at the bottom of Figure 1. Many important information sources are completely unstructured, consisting of plain files or incoming bit strings (e.g., from a newswire). Often it is possible to automatically classify the objects in such sources (e.g., is the file an email message, a text file, or a gif image?), and to extract key properties (e.g., creation date, author). The Classifier/Extractor performs this task, based on identifying simple patterns in the objects. 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. The Classifier/Extractor component is based on the *Rufus* system developed at the IBM Almaden Research Center [25] and is not discussed further in this paper.

1.7 Related Work

There are a number of differences between integration of information sources in the Tsimmis project and other database integration efforts (e.g. [2,13,18,28] and many others):

- Tsimmis focuses on providing integrated access to very diverse and dynamic information. The information may be unstructured or semi-structured, often having no regular schema to describe it. The components of objects may vary in unpredictable ways (e.g., some pictures may be color, others black and white, others missing, some with captions and some without). Furthermore, the available sources, their contents, and the meaning of their contents may change frequently.

- Tsimmis assumes that information access and integration are intertwined. In a traditional integration scenario, there are two phases: an integration phase where data models and schemas (or parts thereof) are merged, and an access phase where data is fetched. In our environment, it may not be clear how information is merged until samples are viewed, and the integration strategy may change if certain unexpected data is encountered.

- Integration in our environment requires more human participation. In the extreme case, integration is performed manually by the end user. For example, a stock broker may read a report saying that IBM has named a new CEO, then retrieve recent IBM stock prices from a database to deduce that stock prices will rise. In other cases, integration may be automated by a mediator, but only after a human studies samples of the data, determines the procedure to follow, and develops an appropriate specification for the mediator generator.

In summary, the Tsimmis goal is *not* to perform fully automated information integration that hides all diversity from the user, but rather to provide a framework and tools to assist humans (end users and/or humans programming integration software) in their information processing and integration activities.

Regarding the constraint management aspects, there has been substantial prior work on database constraints, focusing on centralized databases (e.g., [14]), tightly-coupled homogeneous distributed databases (e.g., [12, 26]), or loosely-coupled heterogeneous databases with special constraint enforcement capabilities (e.g., [8,24]). The multidatabase transaction approach weakens the traditional notion of correctness of schedules (e.g., [5, 10]), but this approach cannot handle a situation in which different databases support different capabilities. In its modeling of time, our work has some similarity to work in temporal databases [27] and temporal logic programming [1], although our approach is closer to the event-based specification language in RAPIDE [19].

1.8 Remainder of Paper

In the rest of this paper we provide additional details on some of the Tsimmis components. In Section 2 we describe the OEM object model and its query language. In Section 3 we present the Tsimmis/Mosaic object browser. In Section 4 we outline the main components of the constraint management toolkit. In Section 5 we conclude, describe the status of the Tsimmis prototype, and discuss future directions of our work.

2 Object Exchange

As described in Section 1.1, our Object Exchange Model (OEM) is used as the unifying object model for information processed by Tsimmis components. Note that 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.

Each object in OEM has the following structure:

Label	Type	Value	Object-ID

where the four fields are:

- **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 Λ (for `null`). The use of this field is described below.

In denoting an object on paper, we often drop the Object-ID field, i.e. we write ⟨label,type,value⟩, as in the examples in Section 1.1.

Suppose an object representing an employee has label *employee* and a set value. The set consists of three subobjects, a *name*, an *office*, and a *photo*. All four objects are exported by an information source IS through a translator, and they are being examined by a client C. The only way C can retrieve the employee object is by posing a query that returns the object as an answer.

Assume for the moment that the employee object is fetched into C's memory along with its three subobjects. The value field of the employee object will be a set of *object references*, say $\{o_1, o_2, o_3\}$. Reference o_1 will be the memory location for the name subobject, o_2 for the office, and o_3 for the photo. Thus, on the client side, the retrieved object will look like:

⟨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"⟩

On the information source side, the employee object may map to a real object of the same structure, or it may be an "illusion" created by the translator from other information. If IS is an object database, and the employee object is stored as four objects with object identifiers id_0 (employee), id_1 (name), id_2 (office), and id_3 (photo), then the retrieved object on the client side would have id_0 in the Object-ID field for the employee object, id_1 in the Object-ID field for the name object, and so on. The non-null Object-ID fields tell client C that the objects it has correspond to identifiable objects at IS. Suppose instead that IS is a relational database, and that the employee "object" is actually a tuple. Then, the name, office, and photo objects (attributes of the tuple) will not have object identifiers, and their Object-ID fields at the client side will be Λ (null).

So far we have assumed that the client retrieves the employee object and all of its subobjects. However, for performance reasons, the translator may prefer not to copy all subobjects. For example, if the photo subobject is a large bitmap with a unique identifier, it may be preferable to retrieve the name and office subobjects in their entirety, but retrieve only a "placeholder" for the photo object. In this case, the value field for the employee object at the client will contain $\{o_1, o_2, id_3\}$. 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.

Note that, regardless of the representation used in set and list values, the translator always gives the client the illusion of an object repository. Thus, we can think of our employee object as:

⟨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"⟩

where each cmp_i is some mnemonic identifier for the subobject. We use this generic notation for examples throughout the remainder of this section.

As mentioned in Section 1, self-describing models have been used in many systems, including file systems [30], Lotus Notes [20], by Teknekron Software Systems [21], and for electronic mail. In many of these systems, nesting of objects is not allowed, so OEM can be viewed as a generalization of these models. OEM is simpler than conventional object models, but it does support the two key features required by object models [6]: *object nesting* and *object identity*.

Our primary reason for choosing a very simple model is to facilitate integration. As pointed out in [3], simple data models have an advantage over complex models when used for integration, since the operations to transform and merge data will be correspondingly simpler. Meanwhile, a simple model can still be very powerful: advanced features can be "emulated" when they are necessary. For example, if we wish to model

[2] We assume that identifiers are unique for each information source. Uniqueness across information sources can be achieved by, e.g., prepending each object identifier with a unique ID for the information source.

an employee class with subclasses *active* and *retired*, we can add a subobject to each employee object with label *subclass* and value "active" or "retired." Of course this is not identical to having classes and subclasses, since OEM does not force objects to conform to the rules for a class. While some may view this as a weakness of OEM, we view it as an advantage, since it lets us cope with the heterogeneity we expect to find in real-world information sources.[3]

2.1 Query Language and Examples

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 (e.g., [15, 16, 17, 23]) to OEM. Here we will give two examples to illustrate the "flavor" of OEM-QL; additional details and examples can be found in [22].

For the examples, suppose that we are accessing a bibliographic information source called `Biblio` with the object structure shown in Figure 2. (Note that we are using mnemonic object references.) Although much of this object structure is regular—components have the same labels and types—there are some irregularities. For example, the call number format is different for each document shown, and the n^{th} document uses a different structure for author information.

Example 2.1 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:

$\langle$answer, set, $\{o_1, o_2\}\rangle$
    o_1: $\langle$topic, str, "Databases"$\rangle$
    o_2: $\langle$topic, str, "Algorithms"$\rangle$ □

[3] Note that some proposed interchange standards, e.g. CORBA's Object Request Broker [11], tend to be significantly more complex than OEM. We expect that if such standards are adopted, OEM could be used to provide a simpler, more "client-friendly" front end. Other proposed standards, such as ODMG's Object Database Standard [7], are directed towards interoperability and portability of object-oriented database systems, rather than towards facilitating object exchange in highly heterogeneous environments.

$\langle$bib, set, $\{doc_1, doc_2, \ldots, doc_n\}\rangle$
    doc_1: $\langle$doc, set, $\{au_1, top_1, cn_1\}\rangle$
        au_1: $\langle$authors, set, $\{au_1^1\}\rangle$
            au_1^1: $\langle$author-ln, str, "Ullman"$\rangle$
        top_1: $\langle$topic, str, "Databases"$\rangle$
        cn_1: $\langle$local-call#, integer, 25$\rangle$
    doc_2: $\langle$doc, set, $\{au_2, top_2, cn_2\}\rangle$
        au_2: $\langle$authors, set, $\{au_2^1, au_2^2, au_2^3\}\rangle$
            au_2^1: $\langle$author-ln, str, "Aho"$\rangle$
            au_2^2: $\langle$author-ln, str, "Hopcroft"$\rangle$
            au_2^3: $\langle$author-ln, str, "Ullman"$\rangle$
        top_2: $\langle$topic, str, "Algorithms"$\rangle$
        cn_2: $\langle$dewey-decimal, str, "BR273"$\rangle$
    $\vdots$
    doc_n: $\langle$doc, set, $\{au_n, top_n, cn_n\}\rangle$
        au_n: $\langle$one-author, str, "Michael Crichton"$\rangle$
        top_n: $\langle$topic, str, "Dinosaurs"$\rangle$
        cn_n: $\langle$fiction-call#, int, 95$\rangle$

Figure 2: Object structure for example queries

Example 2.2 Our next example illustrates how variables are used to specify different paths with the same label sequence. This query retrieves each document for which both "Aho" and "Hopcroft" are authors:

```
SELECT bib.doc
FROM Biblio
WHERE bib.doc.authors.author-ln(a1) = "Aho"
  AND bib.doc.authors.author-ln(a2) = "Hopcroft"
```

Here, the query's `WHERE` clause 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. Hence, for the portion of the object structure shown in Figure 2 the query returns:

$\langle$answer, set, $\{o\}\rangle$
    o: $\langle$doc, set, $\{au_2, top_2, cn_2\}\rangle$
        au_2: $\langle$authors, set, $\{au_2^1, au_2^2, au_2^3\}\rangle$
            au_2^1: $\langle$author-ln, str, "Aho"$\rangle$
            au_2^2: $\langle$author-ln, str, "Hopcroft"$\rangle$
            au_2^3: $\langle$author-ln, str, "Ullman"$\rangle$
        top_2: $\langle$topic, str, "Algorithms"$\rangle$
        cn_2: $\langle$dewey-decimal, str, "BR273"$\rangle$ □

2.2 Implementation

We have argued that OEM and its query language are designed to facilitate integrated access to heterogeneous data sources. To support this claim we have used the OEM model and language to integrate a variety of bibliographic information sources, including a conventional library retrieval system, a relational database holding structured bibliographic records, and a file system with unstructured bibliographic entries. Using our OEM-based system, these sources are accessible through the Tsimmis browser (Section 3), allowing evaluation of queries and object exploration.

As an example, consider one of our operational translators that accesses the Stanford University *Folio* System. Folio provides access to over 40 repositories, including a catalog of the holdings of Stanford's libraries, and several commercial sources such as INSPEC that contain entries for Computer Science and other published articles. Folio is the most difficult of our information sources, partly because the translator must emulate an interactive terminal. The translator initially must establish a connection with Folio, giving the necessary account and access information. 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. The Folio translator is written in C and runs as a server process on Unix BSD4.3 systems. Translators for the other bibliographic sources have involved substantially less coding because the underlying sources (e.g., a relational database) are much easier to use.

We also have implemented mediators that fuse information from multiple bibliographic sources. For example, one mediator provides a simple "union" of the sources, making the information appear as if it all comes from one source. Another mediator performs a "join" of two sources, combining entries that refer to the same document into a single entry that contains all information on the document available from either source.

Finally, we also have implemented *OEM Support Libraries* to facilitate the creation of future translators, mediators, and end-user interfaces. These libraries contain procedures that implement the exchange of OEM objects between a server (either a translator or a mediator) and a client (either a mediator, an application, or an interactive end-user). The Support Libraries handle all TCP/IP communications, transmission of large objects, timeouts, and many other practical issues. A Unix BSD4.3 and a Windows version of the package have been implemented and demonstrated. The Support Libraries are described in [22].

3 Object Browsing

The goal of the object browsing component of Tsimmis is to provide a platform-independent tool for displaying and exploring the OEM objects that are returned as a result of OEM-QL queries. Due to the nested structure of OEM objects, it is necessary to provide mechanisms that let end users navigate easily through the answer space, much like they would navigate through a tree structure. We have implemented *MOBIE* (MOsaic Based Information Explorer), a graphical browsing tool based on Mosaic and the World-Wide-Web [4, 29] for submitting Tsimmis queries and exploring the results. MOBIE lets end users connect to mediators or translators and specify queries using OEM-QL. An important advantage of using Mosaic as the basis for our user interface is its widespread use and popularity. (Mosaic currently operates on Unix workstations, on Macintosh computers, and on many PC's.) Hence, ultimately anyone on the internet should be able to use Tsimmis and MOBIE to explore any information source on the net, provided there is an appropriate translator or mediator available for it.

We illustrate MOBIE's operation by walking through a particular interaction. The first step in accessing information through MOBIE is to select a translator or mediator (henceforth referred to as TM) and connect to it. Figure 3 shows of MOBIE's home page[4] with a list of currently available TMs. The user may select any of the TMs on the list, enter its name in the provided box, and click on the `Connect` button. (Information shown below the `CONNECT` button is used to "fine-tune" the communication between the source and the client, and can generally be left in its default configuration.)

After the connection is established, a *Query Request* page (not shown) is displayed and the system is ready to accept an OEM-QL query. In the current version of MOBIE, queries must be entered by hand, meaning that the user must fill in the boxes provided on the screen (one box for the SELECT clause, one for the FROM clause, and one for the WHERE clause). However, future extensions will include the ability to select parameterized "frequently asked queries" by clicking on menus.

If a submitted query is valid and successfully executed by the TM, the answer object is returned to MOBIE and displayed on a *Query Result* page. Except for very small objects, to see the complete result the user will move through the structure of the answer object using MOBIE's navigational capabilities. This is best understood by thinking of the answer object as a tree (or a graph, in the most general case), where the atom objects are the leaves, and the set objects are the internal nodes. Initially, only the root of the answer object and its immediate subobjects are displayed on the Query Result page (not shown). For our bibliographic data, the root is typically a set containing a set of documents (labelled `doc`). The user can move from the current level in the object structure to a lower level by clicking on the `FETCH`

[4] Mosaic displays information through a series of text screens or *pages*, the first of which is always called the *home page*.

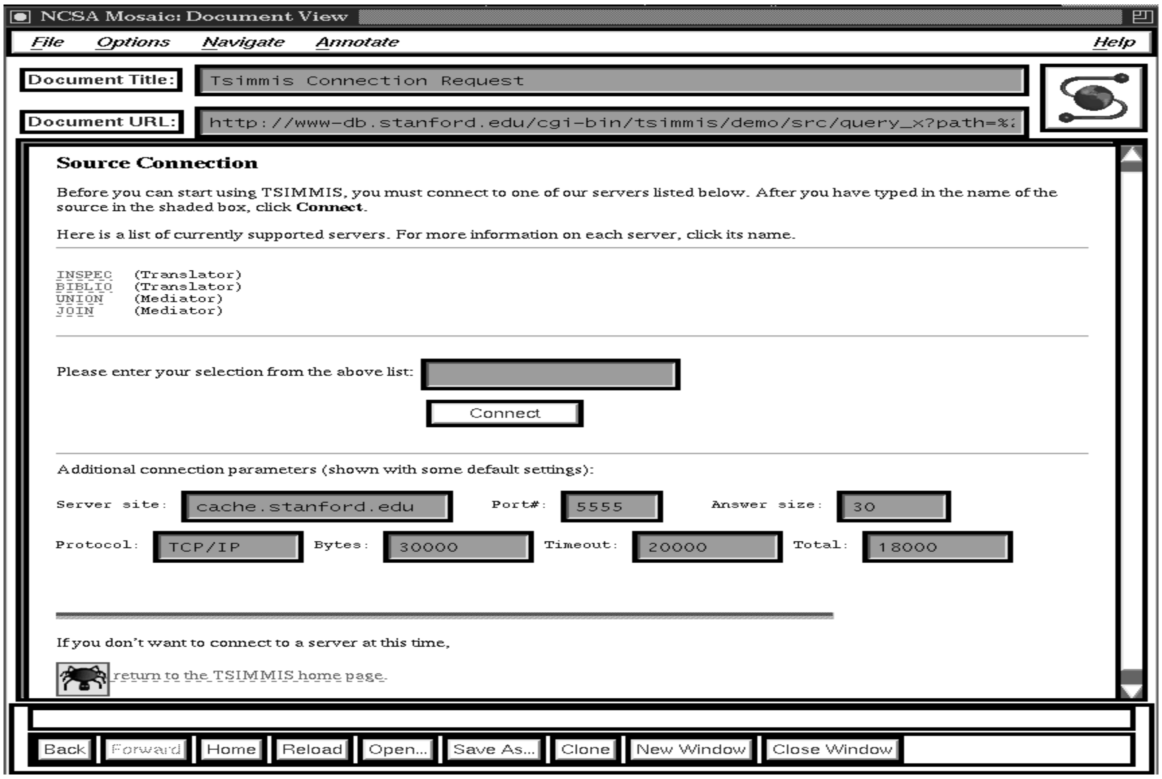

Figure 3: MOBIE's home page

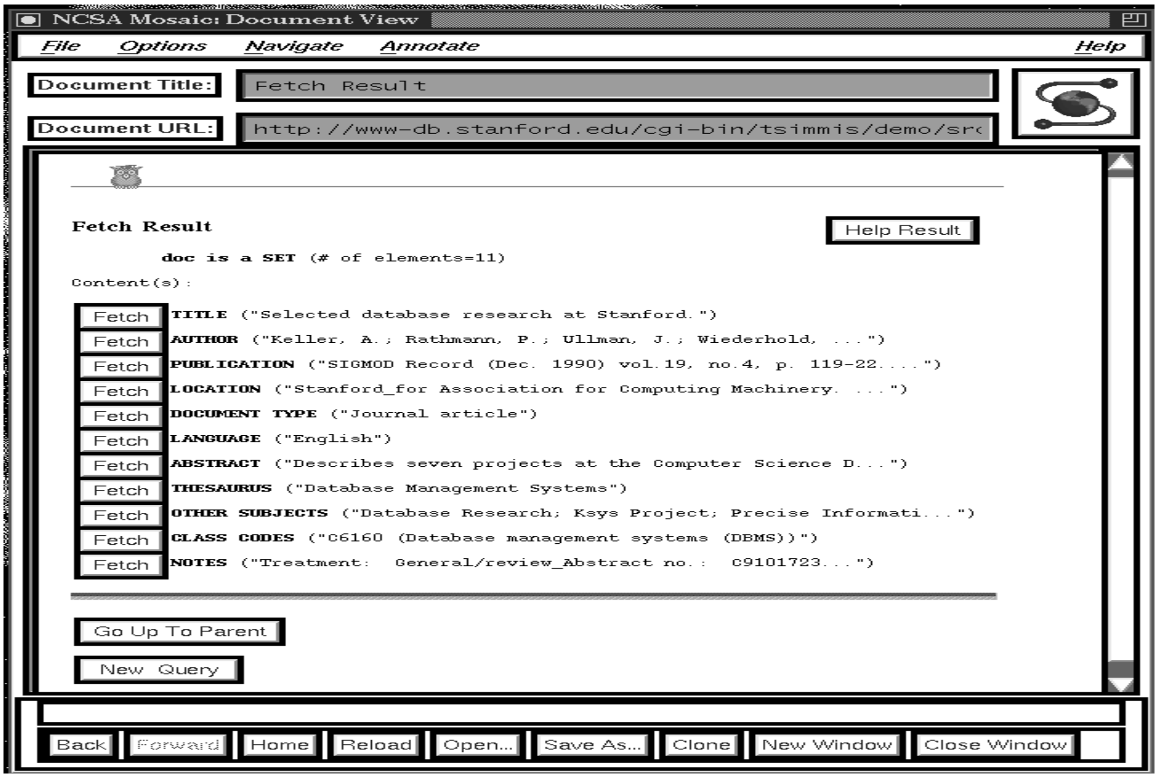

Figure 4: *Fetch Result* page displaying a selected document

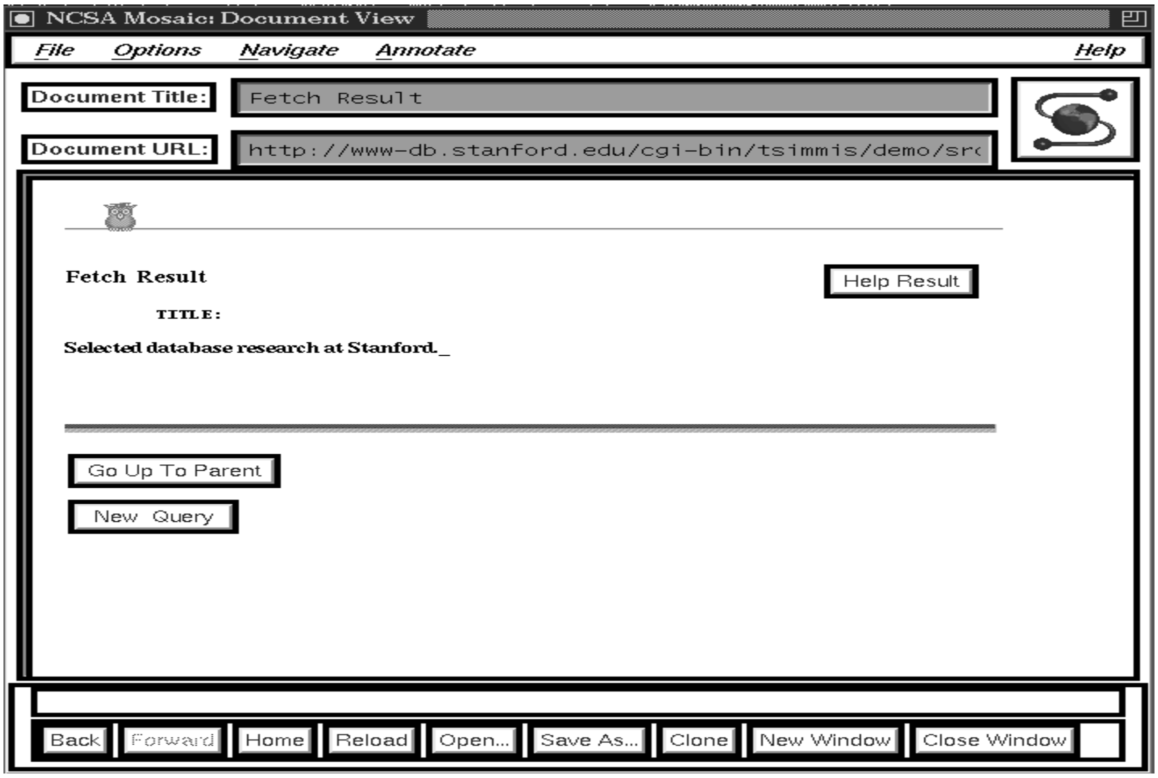

Figure 5: *Fetch Result* page showing the title of a document

buttons preceding each subobject. The result of clicking `FETCH` for one document in the intial query page is shown in Figure 4. The subobjects of this document are labelled `Title`, `Author`, etc., and their values can again be fetched by clicking on the `FETCH` buttons. For example, the result after clicking the `TITLE` button is shown in Figure 5. At this point we have a reached a leaf (or atom) in the answer space and cannot descend any further. The user can either backtrack to one of the parent objects by clicking on the `Go Up To Parent` button, or enter a new query by selecting `New Query`.

At any point in the session, the user can ask for help by selecting the `Help Result` button, which displays text on the meaning of a particular result object. As discussed in Section 2, each TM provides capabilities for describing (in English) the meaning of a label, and how to interpret the value of objects with that label. As an example, the help entry for the *author* label as returned by the translator would explain that author objects consist of a last name followed by a first name or initials. A MOBIE session is ended by selecting the `Close Session` button on the Query Request page.

4 Constraint Management

The Tsimmis Constraint Manager is based on a general formal framework we have developed for constraints in heterogeneous systems. Each information source (recall Figure 1) chooses an *interface* it can offer to the local constraint manager (LCM) for each of its data items involved in a multi-source constraint. The interface specifies how the data item may be read, written, and/or monitored by the LCM. Applications inform the constraint manager (CM) of constraints that need to be monitored or enforced. Based on the constraint and the interfaces available for the items involved in the constraint, the CM decides on the constraint management *strategy* it executes. This strategy monitors or enforces the constraint as well as possible using the interfaces offered by the information sources. The degree to which each constraint is monitored or enforced is formally specified by the *guarantee*. We briefly describe interfaces, strategies, and guarantees next. Complete formal specifications of each can be found in [9].

Interfaces are specified using a notation based on *events* and *rules*. As an example, we illustrate a simple "write interface" for a data item X. With this interface, the information source promises to write any requested value to X within five seconds. The interface is expressed as the rule:

$$WR(X, b) \rightarrow W(X, b);\ B \leq 5.$$

Here, $WR(X, b)$ represents a *write-request* event requesting operation $X := b$. The rule says that whenever such a write-request event occurs, a *write* event, $W(X, b)$, occurs within 5 time units. We assume that the interfaces for the data items involved in constraints are specified by a "constraint administrator"[5] at each site, based on the level of access and performance that can be provided to the CM for the data item. Currently, we rely on the users of our framework to verify that the interfaces specified do faithfully represent the actual systems.

The strategy for a constraint describes the algorithm used by the CM to monitor or enforce the constraint. Like interfaces, strategies are specified using a notation based on events and rules. In addition to performing operations on the data items involved in a constraint, strategies may evaluate predicates over the values of data items (obtained through read operations) and over private data maintained by the CM. As a simple example, consider the strategy description below, which issues a write request to Y within 7 seconds whenever a *notify* event is received from X. (A notify event represents the source notifying the CM of a write to a data item. Thus, e.g., $N(X, 5)$ represents the notification that a write $X := 5$ occurred.) This strategy might be used to maintain the constraint $X = Y$.

$$N(X, b) \rightarrow WR(Y, b);\ B \leq 7.$$

Rule-based strategy specifications are implemented using the host language of the CM. The translation from rules to host language is usually straightforward, and it may be achieved using a rule processing engine.

The guarantee for a constraint specifies the level of global consistency that can be ensured by the CM when a certain strategy for that constraint is implemented. Typically a guarantee is *conditional*, e.g., a guarantee might state that if no updates have been performed recently then the constraint holds, or if the value of a CM data item is true then the constraint holds. Guarantees are specified using predicates over values of data items and occurrences of certain events. For example, consider the following guarantee for a constraint $X = Y$:

$$(\text{Flag} = \text{true})@t \Rightarrow (X = Y)@@[t - \alpha, t - \beta].$$

This guarantee states that if the Boolean data item Flag (maintained by the CM) is true at time t, then $X = Y$ holds at all times during the interval $[t - \alpha, t - \beta]$. Note that this guarantee is weaker than a guarantee that $X = Y$ always holds, which is a very difficult guarantee to make in the heterogeneous, autonomous environments we are considering.

4.1 A Constraint Management Toolkit

As part of the Tsimmis prototype, we have built a toolkit that permits constraint management across heterogeneous and autonomous information systems. This toolkit allows us to enforce, for example, a copy constraint spanning data stored in a Sybase relational database and a file system, or an inequality constraint between a *whois*-like database and an object-oriented database.

Figure 6 depicts the architecture of our constraint management toolkit, which is based on the formal framework described above and interfaces with the Tsimmis architecture depicted in Figure 1. The Raw Information Sources (RIS) are what exist already at each site (for example, a relational database, a file system, or a news feed). The RISI is the source-specific interface offered by each RIS to its users and applications. For example, for a Sybase database, the RISI is based on a particular dialect of SQL, and includes details on how to connect to the server.

The CM-Translator is the module that implements the interfaces for each of its data items. The CM-Translator is specified by a configuration file called a CM-RID (for Raw Interface Description), which includes: (1) which interfaces (selected from a menu of interface types) are supported by the CM-Translator, and (2) how these interfaces are implemented using the underlying RISI.[6] The CM-Shells cooperate to execute the constraint management strategies. The CM-Shells are distributed rule engines that are configured by a Strategy Specification file.

We now describe how constraint administrators would use our toolkit to set up constraint management across multiple sources. The administrators at each site first decide on the CM-Interfaces they are willing to offer, selected from menu of predetermined interfaces provided by the toolkit. For example, if the underlying RIS provides triggers, then a notify interface may be offered; if not, perhaps a read/write interface can be offered. The choice also depends on the actions the administrator wants to allow. For instance, even if the RIS allows updates to the source, the administrator may disallow a write interface that lets the CM make changes to the local data. Each CM-RID file records the interfaces supported, as well as the specification of the RIS objects to which the interface applies.

[5] The constraint administrator is an individual who is familiar with the structure and behavior of a given information source, much like a database administrator.

[6] Note that the CM-Translator is responsible for translating between rule-based interface specifications (as described earlier) and source-specific operations. For translation of data and queries, a Tsimmis translator can be used. Hence, the CM-Translator together with the CM-RID comprise the Local Constraint Manager illustrated in Figure 1.

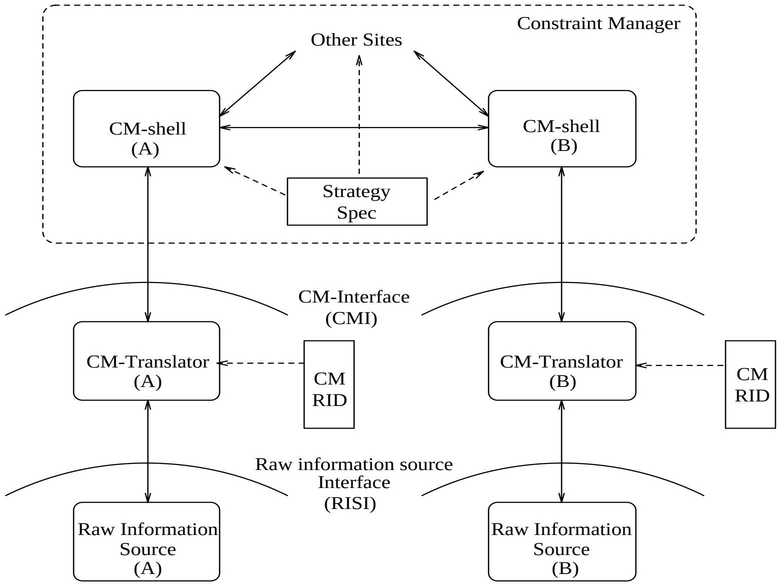

Figure 6: Constraint Management Toolkit Architecture

Next, the administrator uses a Strategy Design Tool (not shown in Figure 6) to develop the CM strategy. This tool takes as input the multi-source constraints; based on the available interfaces, it suggests strategies from its available repertoire. For each suggested strategy, the design tool can give the guarantee that would be offered. The result of this process is a Strategy Specification file, which is then used by the CM-Shells at run time. Note that knowledgeable administrators might choose to write their Strategy Specifications directly, bypassing the Design Tool.

5 Conclusion

In summary, the Tsimmis project is exploring technology for integrating heterogeneous information sources. Current efforts are focusing on translator and mediator generators, which should significantly reduce the effort required to access new sources and integrate information in different ways. We believe that the OEM model described here provides the right flexibility for handling unexpected heterogeneity.

Acknowledgements

We are grateful to Ed Chang and Jon Goldberg for their implementation efforts, to Ashish Gupta, Dallan Quass, and Anand Rajaraman for valuable comments, and to the entire Stanford Database Group for numerous fruitful discussions.

References

[1] Martin Abadi and Zohar Manna. Temporal logic programming. *Journal of Symbolic Computation*, 8(3):277–295, 1989.

[2] R. Ahmed et al. The Pegasus heterogeneous multidatabase system. *IEEE Computer*, 24:19–27, 1991.

[3] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18:323–364, 1986.

[4] T. J. Berners-Lee, R. Cailliau, and J.F. Groff. The world-wide web. *Computer Networks and ISDN Systems*, 25:454–459, 1992.

[5] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of multidatabase transaction management. *VLDBJ*, 1(2):181, October 1992.

[6] R. G. G. Cattell. *Object Data Management*. Addison-Wesley, 1991.

[7] R. G. G. Cattell. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, 1994.

[8] Stefano Ceri and Jennifer Widom. Managing semantic heterogeneity with production rules and persistent queues. In *VLDB*, pages 108–119, Dublin, Ireland, August 1993.

[9] Sudarshan S. Chawathe, Hector Garcia-Molina, and Jennifer Widom. Constraint management in loosely coupled distributed databases. Technical report, Computer Science Department, Stanford University, 1993. Available through anonymous ftp from host `db.stanford.edu` as `pub/chawathe/1993/cm-loosely-coupled-dbs.ps`.

[10] Ahmed Elmagarmid, editor. *Special Issue on Unconventional Transaction Management,* Data Engineering Bulletin 14(1), March 1991.

[11] Object Request Broker Task Force. The Common Object Request Broker: Architecture and Specification, December 1993. Revision 1.2, Draft 29.

[12] Paul Grefen. Combining theory and practice in integrity control: A declarative approach to the specification of a transaction modification subsystem. In *VLDB*, pages 581–591, Dublin, Ireland, August 1993.

[13] A. Gupta. *Integration of Information Systems: Bridging Heterogeneous Databases.* IEEE Press, 1989.

[14] M. Hammer and D.McLeod. A framework for database semantic integrity. In *Proceedings of the Second International Conference on Software Engineering*, pages 498–504, San Francisco, California, October 1976.

[15] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 59–68, San Diego, California, June 1992.

[16] W. Kim et al. On resolving schematic heterogeneity in multidatabase systems. *Distributed And Parallel Databases*, 1:251–279, 1993.

[17] H. F. Korth and M. A. Roth. Query languages for nested relational databases. In *Nested Relations and Complex Objects in Databases*, pages 190–204. Springer-Verlag, 1989.

[18] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22:267–293, 1990.

[19] David C. Luckham et al. Specification and analysis of system architecture using Rapide. *IEEE Transactions on Software Engineering*, 1994.

[20] D. S. Marshak. Lotus Notes release 3. *Workgroup Computing Report*, 16:3–28, 1993.

[21] B. Oki et al. The information bus—an architecture for extensible distributed systems. In *Proceedings of the Fourteenth ACM Symposium on Operating System Principles*, pages 58–68, Asheville, NC, December 1993.

[22] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object exchange across heterogeneous information sources. In *Proceedings Data Engineering Conference*, Taipei, Taiwan, March 1995.

[23] M. A. Roth, H. F. Korth, and A. Silberschatz. Extended algebra and calculus for nested relational databases. *ACM Transactions on Database Systems*, 13:389–417, 1988.

[24] Marek Rusinkiewicz, Amit Sheth, and George Karabatis. Specifying interdatabase dependencies in a multidatabase environment. *COMP*, 24(12):46–51, December 1991.

[25] K. Shoens, A. Luniewski, P. Schwarz, J. Stamos, and J. Thomas. The rufus system: Information organization for semi-structured data. In *VLDB*, pages 97–107, Dublin, Ireland, August 1993.

[26] Eric Simon and Patrick Valduriez. Integrity control in distributed database systems. In *Proceedings of the Nineteenth Annual Hawaii International Conference on System Sciences*, pages 621–632, 1986.

[27] Richard Snodgrass. Temporal databases. *COMP*, 19(9):35–42, September 1986.

[28] G. Thomas et al. Heterogeneous distributed database systems for production use. *ACM Computing Surveys*, 22:237–266, 1990.

[29] Steven J. Vaughan-Nichols. How to glue together mosaic (internet browser) (tutorial). *Government Computer News*, 13(15):33, July 1994.

[30] G. Wiederhold. *File Organization for Database Design.* McGraw Hill, New York, 1987.

[31] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25:38–49, 1992.