Interoperability via Mapping Objects

Fragkiskos Pentaris and Yannis Ioannidis
University of Athens, Dept. of Informatics and Telecommunications
Panepistemioupolis, 157 84, Athens, Hellas (Greece)
{frank, yannis}@di.uoa.gr

1. Introduction

In order to provide end-users with services of better quality, Digital Libraries (DLs) need to be constantly adopting new technologies that are emerging in various related fields, e.g., new metadata models and languages, new information retrieval algorithms, and new storage means. These force DLs to constantly alter their basic structure, generating significant interoperability problems between the new and the old systems. Solving the various semantic, syntactic and structural interoperability issues in large federations of DLs, remains a major problem [8].

Over the years, system designers have developed several different approaches for interoperability [12], including the use of (families of) standards, external mediation, specification-based interaction, and mobile functionality. The use of mediation or middleware techniques has gained much support, as it provides good results without compromising the autonomy of existing systems. Recent examples of such DL systems and architectures include MARIAN [3], MIX [1], and the Alexandria Digital Library [4]. Some other relevant, yet more general, systems include Pegasus, Infomaster, TSIMMIS, GARLIC, HERMES, MOCHA, MOMIS, OPM, SIMS/ARIADNE, Information Manifold, Clio, IRO-DB, and MIRO-Web.

In this short paper, we give an overview of the HORSE II distributed mediated system. This system, follows the traditional mediator-wrapper model, combined with several techniques to provide bi-directional (read-write) access to multiple, disparate data sources. Transformations between different forms of data and queries are expressed with the help of volatile *mapping objects*. Mapping objects offer most of the functionality of other declarative approaches to transformation, e.g., through mapping languages or, more recently, meta-mapping languages, and in addition allow for easy querying, sharing, reusing, and updating of transformation information.

In the next section, we give an overview of the HORSE II architecture. Subsequently, we present the way mapping objects are created, stored, and used by the system. After comparing mapping objects with other relevant approaches, we conclude with the status of the implementation.

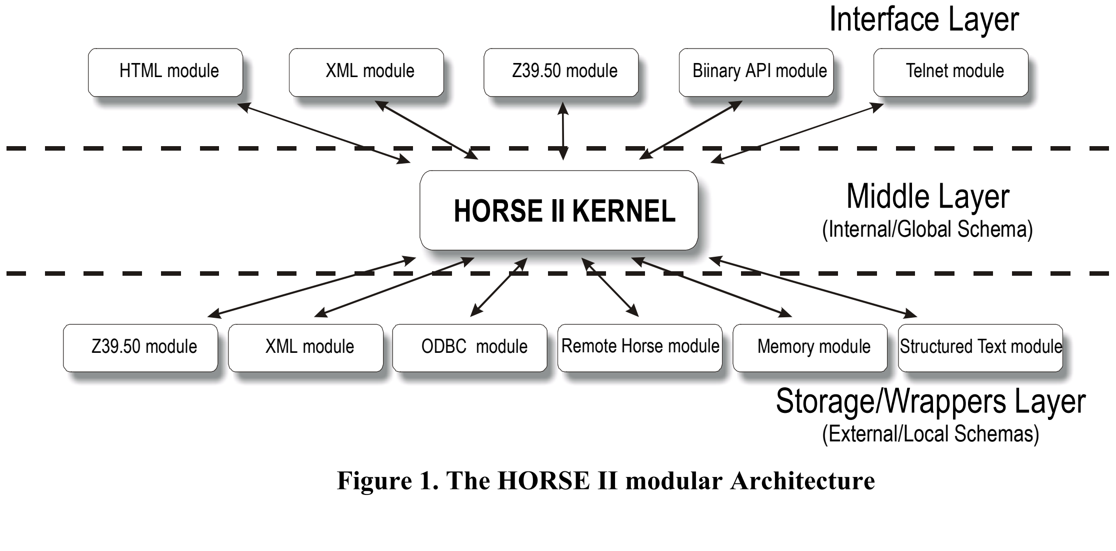

Figure 1. The HORSE II modular Architecture

2. The Horse II Architecture

HORSE II follows a multi-layered, component-based architecture (Figure 1). The top layer contains *interface* modules that provide the means to end-users and third-party applications to interact with the system. It performs two tasks: (a) It accepts and parses user queries to convert them into an internal binary-tree format and (b) it converts the answers of these queries back to the “native” format of each external interface.

Interface modules use a well-defined API to forward user queries to the *kernel* of HORSE II. These queries are expressed using the MOOSE object-based data model and the FOX query language [6]. HORSE II allows queries to reference multiple parts of the database schema that transparently reside in different external storage systems. For example, a single query can refer to data stored in an XML file, a text-formatted file, and a relational database. The HORSE II kernel is responsible for splitting user queries into subqueries for execution by the relevant storage modules (bottom layer) performing optimisation to the extent that its knowledge about the storage systems allow. In addition, the kernel provides basic join, merge, sort, and filter capabilities for building the final answer of a query from the partial answers of its subqueries.

The bottom layer of HORSE II contains the *storage/wrapper* modules. These enable HORSE to access data residing in external, heterogeneous, autonomous data sources, e.g., XML files, relational databases, data under remote instances of HORSE II, or even data produced by arbitrary software applications. Each module is responsible for converting (parts of) user queries and updates (whenever these are allowed [11]) from the FOX query language used internally by the kernel into the native external query language.

A critical question of mediator-based systems is how external data are mapped to the internal schema. Earlier systems implement mappings in the wrapper code. This typically has the best performance but also the least flexibility, as excessive source-code modifications may be required every time a mapping is changed. Even when mappings are expressed as declarative rule code, this may hamper the system’s expandability.

HORSE II follows a new declarative approach, which is based on the use of *mapping objects*. These are described in the next section.

3. Mapping Objects

In principle, external schemas are independent from the internal schema and vice versa. Furthermore, for each one there are multiple mappings onto schemas on the other side that are meaningful, and often several of them may need to be active simultaneously. That is, mappings are independent from the schemas as well, in the sense that they are not absolutely determined by them. Given such autonomy, we propose to express and store the information required for data conversion as *mapping objects* of a “mapping schema”, in exactly the same fashion that the internal and external schemas are stored as objects in the appropriate “meta schemas”.

More specifically, communication of HORSE II with any external system assumes the existence of three system-level schemas: First, there is the meta-schema of HORSE II, which is managed directly by the kernel and is unique. It will store the classes and relationships of the internal (MOOSE) schema. Second, there is the meta-schema for schemas specific to the external system. It will store the components of the external schema as expressed in the data model of the corresponding system. For example, to manage external data stored in XML files, this meta-schema should be able to capture the components of a DTD file. Third, there is the schema for capturing (possibly bi-directional) mappings between schemas of the internal and external types. This is also specific to the nature of the external system, as it must refer to parts of the second meta-schema above (as well as the first one), although some higher-level parts of it are generic.

Figure 2 shows an example of (parts of) the three system-level schemas for XML external data. At the top is (part of) the MOOSE meta-schema, consisting of classes CLASS and RELATIONSHIP, which can store information on any internal HORSE II schema. At the bottom is the XML meta-schema, consisting of classes XML ELEMENT, XML CONTENT PART, PSEUDO XML ELEMENT, and XML ATTRIBUTE. These classes represent our way of capturing the schema of an XML file (i.e., a DTD file). Finally, in the middle is the schema for XML mapping objects linking internal (MOOSE) schemas and external (XML file) schemas.

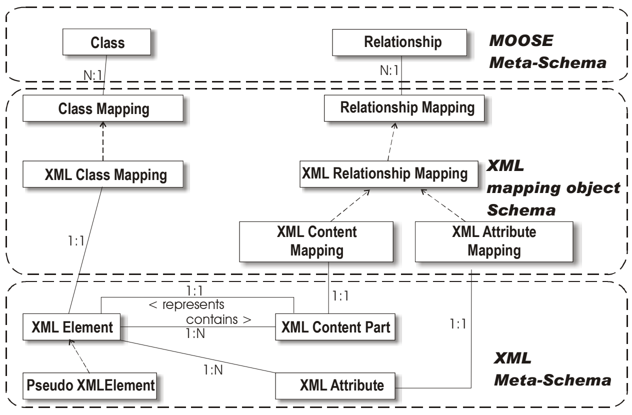

Figure 2. The schema of an XML mapping object.

As an example of how HORSE II currently uses these schemas, Figure 3 shows some of the statements executed when a very simple mapping from an XML file with books and authors to an appropriate MOOSE schema is declared. The `load module` statement is given once to dynamically load and activate the XML storage module. The schema and mapping declaration statements are between a `Begin schema` and an `End schema` statement. The `Register tuple class Book` statement adds to the internal schema a new class `Book` with an attribute `Author` (updating the CLASS and RELATIONSHIP classes of Figure 2). The first two `Insert` statements record the features of the external XML file by constructing two instances of the XMLElement and XMLAttribute classes. The last `Insert` statement defines the mapping between the internal and the external schema. Finally, the `Handle class Book` statement binds this mapping with the internal class Book and the external XML file.

```
> Load module "xmlstorage.so"; // Dynamically load XML storage module.
> Begin schema;

> Register tuple class Book         // Alter Internal schema definition
   with haspart Author to string,
         .
         .
> Insert into XMLElement instance(...);  // Alter XML schema definition
> Insert into XMLAttribute instance(...);
         .
         .
> Insert into XML_ClassDefinition  // Create a mapping object. Note that this
     instance (...) as MapObject;  // is similar to creating a normal MOOSE object.

> Handle class Book
    using xmlstorage at "http://hephaestus.di.uoa.gr/dl/metadata.xml"
    mapped as MapObject;         // Bind internal class MetadataObject with the
                                 // external XML file using the
                                 // XML mapping object MapObject.
> End schema;
```

Figure 3. Creating a mapping between an internal and an external schema

Mapping objects can be used to express both unidirectional (in either direction) and bi-directional transformations of high complexity between the internal and the external schema. Thus, they can be used in conjunction with both the *global-as-view* and the *local-as-view* approaches [7], where the internal schema is a function over the external schemas or the opposite, respectively. Such flexibility is both important, as each approach is better than the other on queries or updates, and also necessary, as both forms of usage will be demanded of future DLs for specific data. For example, assume that the internal schema contains two classes named `Book` and `NewBook` (the latter feeding the former), which are mapped onto an XML file and a relational table respectively. Figure 4 shows how easy it is to perform the data transfer (with a single insert-select statement) having defined appropriate mapping objects in the two directions.

```
/* Select some Book objects and copy (insert)
   them into class NewBooks.    */
> insert into NewBooks
  (for e in Books
     select e where Author="Socrates");

// Commit changes
> Commit;
```

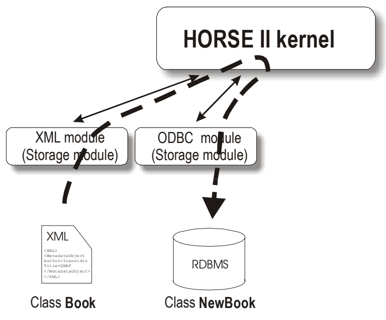

Figure 4. Moving objects from an XML file into a Relational

Mapping objects have several advantages compared to earlier approaches:

- Schemas and mappings are defined independently. Thus, in HORSE II, it is possible to map the same external schema onto two or more internal ones and vice versa.
- Mapping objects can be shared easily in distributed systems, just like all other objects.
- Mapping objects can be queried and updated through the regular query language. Such an update essentially alters the mapping between the internal and the external schema.
- Mapping objects (or their parts) can be shared and reused. This is useful for dealing with large schemas containing many repetitions and for incorporating parts of standard schemas (e.g., ontologies) into larger more specific ones.
- Mapping objects facilitate system expandability. To communicate with a brand new external system, only a new meta-schema and corresponding mapping schema need to be defined for data transformation.

4. Comparison with Existing Approaches

Most mediation systems overcome the problem of hard-wiring the mappings into the wrappers by using some specific mapping language, e.g., there is the BRIITY (bridging heterogeneity) mapping language [5], MARIAN uses 5S, a digital library description language [3], while TSIMMIS has MSL (Mediation Specification Language) [9]. The use of a mapping language shares some of the characteristics of our approach but not the advantages mentioned above. With respect to expandability, for example, interacting with a brand new system would require modification of the mapping language itself, as it can only express mappings to systems that are known at the time it was designed. Proposals have also been made to use logic and other Datalog-like notation as a mapping language (and for expressing other useful inter-schema information) [2, 13]. This offers superior expressive power but does not address all the points made above. Likewise, a recent proposal to use a meta-language for defining mapping languages seems to overcome some of the limitations of mapping languages [10], but it appears to be still inferior to mapping objects with respect to some aspects, e.g., reusability and sharing.

Note that a mapping-language approach can be easily combined with our approach, by putting code excerpts as part of an object. This may be necessary in particular when the expressive power of mapping objects fails to capture a particular complicated case.

5. Status of HORSE II

HORSE II is being implemented in C++ and is currently about 70K lines of code. The query processor is still incomplete, but mapping objects are already supported and can be used for updates to disparate external data sources. The current implementation focuses on global-as-view mapping objects but this will be generalized in the future supporting both kinds in a uniform way. With respect to mapping-object specification, which may still be nontrivial for complex schemas, the system currently offers tools to automatically generate MOOSE schemas from relational schemas and XML DTD files, respectively, using all semantic information available in them, e.g., referential constraints. These mappings can be modified by the user to fit to particular needs.

As a final, somewhat related note, with respect to transaction management, HORSE II makes a modest effort to support simultaneous querying or updating of multiple disparate data sources within the boundaries of a single global transaction. The system attempts to use two-phase commit and two-phase locking if the capabilities of the sources permit this. Otherwise, e.g., if a relational database and an XML file is referenced in the same transaction, the system falls back to *weaker* protocols (possibly even that of employing no concurrency control mechanism). In general, the current implementation does not try to maintain the properties of a proper global transaction, since this is not universally possible when dealing with arbitrary, truly autonomous data sources.

6. References

[1] Chaitanya Baru, Vincent Chu, Amarnath Gupta, B. Ludäscher, Richard Marciano, Yannis Papakonstantinou, and Pavel Velikhov, XML-Based Information Mediation for Digital Libraries, Proc. Of ACM Conf. on Digital Libraries, Berkley, August, 1999.

[2] Tiziana Catarci, and Maurizio Lenzerini, Interschema Knowledge in Cooperative Information Systems, CoopIS 1993, pp 55-62.

[3] Marcos André Conçalves, Robert K. France, Edward A. Fox, and Tamas E. Doszkocs, MARIAN Searching and Querying across Heterogeneous Federated Digital Libraries, 1st DELOS Network of Excellence Workshop on "Information Seeking, Searching and Querying in DL", 2000.

[4] J. Frew, M. Freeston, N. Freitas, L. Hill, G. Janée, K. Lovette, R. Nideffer, T. Smith, and Q. Zheng, The Alexandria Digital Library architecture, Int. Journal on Digital Libraries, 2:259-268, 2000.

[5] T. Härder, G. Sauter, and J. Thomas, The intrinsic problems of structural heterogeneity and an approach to their solution, The VLDB Journal, 8:25-43, 1999.

[6] Y. E. Ioannidis, M. Livny, S. Gupta, and N. Ponnekanti, ZOO : A Desktop Experiment Management Environment, Proceedings of 22nd Int. Conference on VLDB, 274-285, 1996.

[7] Alon Y. Levy, and Daniel S. Weld, "Intelligent Internet Systems: An introduction to the special Issue", Artificial Intelligtence, 2000.

[8] D. M. Levy, and C. C. Marshal, Going Digital: A Look at Assumptions Underlying Digital Libraries, Communications of the ACM, 38(4):77-84, April 1995.

[9] Chen Li, R. Yerneni, V. Vassalos, H. Garcia-Molina, Y. Papakonstantinou, J. Ullman and M. Valiveti, Capability Based Mediation in TSIMMIS, ACM Int. Conference on Management of Data, 1998.

[10] Sergey Melnik, Héctor Garcia-Molina, and Andreas Paepcke, A Mediation Infrastructure for Digital Library Services, Proc. of ACM Digital Libraries Conference, ACM Press, June 2000.

[11] R. J. Miller, Yannis E. Ioannidis, and R. Ramakrishnan, The Use of Information Capacity in Schema Integration and Translation, Proceedings of the 19th VLDB Conference, Dublin, Ireland, 1993.

[12] A. Paepcke, C-C K. Chang, H. Garcia-Molina, and T. Winograd, Interoperability for Digital Libraries Worldwide, Communications of ACM, 41(4):33-43, 1998.

[13] Jeffrey D. Ullman, Information Integration Using Logical Views, 6th Int. Conference on Database Theory (ICDT), 1997.