Int J Digit Libr (2004) 4: 52–55 / Digital Object Identifier (DOI) 10.1007/s00799-003-0076-2

Mapping objects

Fragkiskos Pentaris, Yannis E. Ioannidis

University of Athens, Dept. of Informatics and Telecommunications, Panepistemiopolis, 157 71, Athens, Greece;
E-mail: {frank,yannis}@di.uoa.gr

Published online: 27 July 2004 – © Springer-Verlag 2004

Abstract. We present a technique that is based on volatile mapping objects and enables wrappers-based mediation architectures to describe bi-directional (read–write) interschema mappings of multiple, disparate data sources. We describe the structure of these mapping objects, explain how they work, and compare them to other traditional techniques used for describing schema mappings in data-mediation systems.

1 Introduction

The natural replacement of aged information technologies with more recent ones, and the rather large number of different metadata ‘standards’ that emerged in the last decade, forces digital libraries (DLs) to use many autonomous, heterogeneous, and usually incompatible systems to store their metadata. The de facto adoption of XML language has reduced the syntactic-related heterogeneity problems, yet it has been of no help in attacking the structure- and semantics-related compatibility problems of digital object collections. At the time of writing, almost every major DL stakeholder is somehow involved in creating new (meta)metadata standards, such as the Metadata Encoding and Transmission Standard of the Digital Library Federation [9] or the drafts of the numerous Dublin Core Metadata Working Groups [8]. This constant revision of standards intensifies existing interoperability problems in DL federations and raises the need for the design of systems that can quickly and easily adapt to metadata standards modifications.

System designers have developed several different approaches for maintaining interoperability [7]. The use of mediation/middleware techniques has gained considerable support, as it provides good results without compromising the autonomy of existing systems. Examples of such DL systems and architectures include the MARIAN system [2], the MIX project [1], and the Alexandria Digital Library architecture [3]. 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 projects.

An important property of all these systems is the mechanism used for describing the mapping between the internal/mediation schema(s) and the external (mediated) ones. In this short paper, we present a technique that describes bi-directional (read–write) interschema mappings between multiple, disparate data sources with the help of volatile mapping objects. Our technique is not only comparable to other declarative ways of describing mappings, such as mapping languages and, more recently, metamapping languages, but also allows for easy querying, sharing, reusing, and updating of the local or remote mapping information.

The rest of the paper is organized as follows: in Sect. 2 we briefly describe the architecture assumed in this paper. In Sect. 3, we describe our schema mapping technique. In Sect. 4 we compare our approach with relevant techniques and conclude the paper.

2 Mediation environment

In this paper, we assume that a wrapper-based mediation architecture, like the one in Fig. 1, is used to provide a single point of access to data stored in multiple disparate data sources. This is accomplished through the use of wrappers that are capable of accessing in a native, bi-

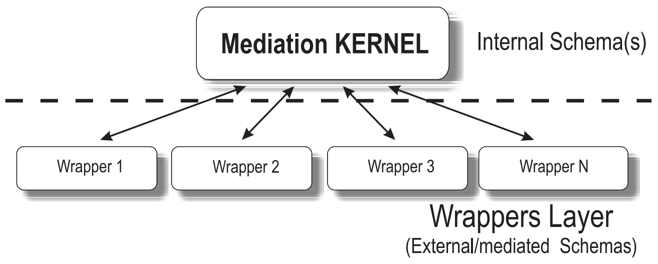

Fig. 1. Mediation architecture

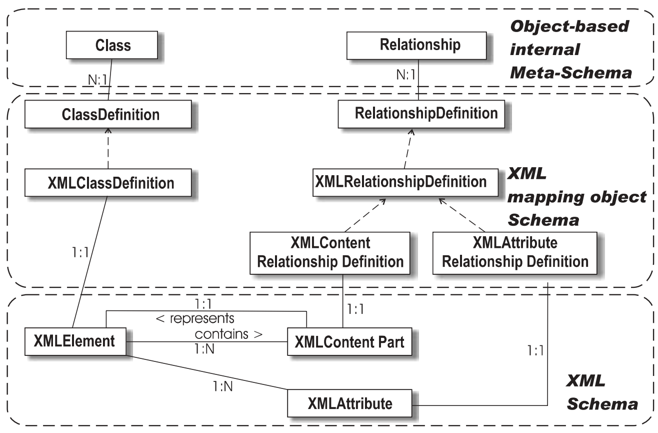

Fig. 2. Mapping objects for XML data sources

directional way, different data sources, such as XML files, relational databases, or even data that are the output of a different software application. Each wrapper converts multiple external sources of the same type into a single common data model used by the mediation kernel. This paper will use an object-based model, though our technique is model-neutral and can be used with any data model such as the relational one or an XML-structure-like one.

Wrappers must be flexible and expandable with regard to the type and schema of the mediated data sources. This depends on the way wrappers map external data to the internal schema(s). Old systems hard-coded the mapping between those two schemas, which severely affected the flexibility and expandability of these systems. Our approach, which we describe in detail in the next section, overcomes these problems by using mapping objects.

3 Mapping objects

Consider an external data source wrapped by a mediation/middleware system. The schema of this source and the internal schema of the middleware are completely independent of each other, since external sources are autonomous and can (simultaneously) be mapped into several different internal ones, simply by changing the mapping algorithm. Modifications of the external schema do not necessarily propagate to the internal one. Thus, external and internal schemas are two separate entities, simply bound together through some mapping algorithm.

To ensure flexibility and expandability, our mapping technique respects the autonomy of the two types of schemas and keeps information concerning the internal schema, the external schema, and their mapping separately. More specifically, connecting the internal and external schemas involves three separate actions:

- Definition of the internal schema (i.e. the classes and relationships of the common object-based schema), which is used by the mediation. The corresponding information is stored in the private mediation metaschema and is validated and managed directly by the mediation kernel.
- Identify the external schema, followed by the physical data sources. The corresponding information is wrapper-specific and is, therefore, stored in a wrapper-specific metaschema. For example, if the data is stored in an XML file, then the DTD of the file will have to be declaratively specified.
- Specification of the mapping between the two schemas. The corresponding information is also stored in a wrapper-specific metaschema, since it cross references objects specified in the first two steps.

The first two actions may occur in either order, while the third must naturally follow the other two. Particular instances in the three inter-related metaschemas structurally compose a single mapping object.

Figure 2 shows an example of an XML mapping object. In this figure, three sub-schemas are displayed. The first is the internal object-based metaschema, consisting of classes CLASS and RELATIONSHIP, which store information on the internal mediation schema. The second sub-schema is the XML metaschema consisting of classes XMLElement, XMLContentPart, and XMLAttribute. These XML wrapper-specific classes are sufficient to store information on the schema of an XML file (i.e. the DTD file). Finally, the remaining six classes of the XML mapping object store the information that links the internal schema with the schema of the XML file. For example, instances of the XMLContentRelationshipDefinition class map attributes of the classes of the internal schema to the contents of the elements of the XML files. Note that the XMLClassDefinition and XMLRelationshipDefinition classes inherit from the ClassDefinition and RelationshipDefinition classes respectively, which hold mapping information common to every external source type.

As an example of how our technique works in practice, Fig. 3 shows the statements (mostly in SQL form) executed when a very simple mapping from an XML file with books having titles and authors to an appropriate object-based schema is declared. The schema and mapping declaration statements are between the `Begin schema` and `End schema` statements. The `Register class Book` statement adds to the internal schema a new class *Book* with

```
> Begin schema;

> Register class Book // Alter internal schema
  { String Title, // definition
     String Author }
// Alter external XML schema definition
> Insert into XMLElement(ElementName,URL)
    instance('Book',
               'http://www.di.uoa.gr/books.xml')
                    as BookObject;
> Insert into XMLAttribute
                   (AttributeName,xmlElement)
    instance('Title',BookObject)
                    as TitleObject;
> Insert into XMLAttribute
                   (AttributeName,xmlElement)
    instance('Author',BookObject)
                    as AuthorObject;

// Now create the mapping algorithm. Note that
// this is similar to creating a normal object.
> Insert into XMLClassDefinition
                   (className, xmlElement)
    instance ('Book', BookObject)
> Insert into XMLAttributeRelationshipDefinition
               (classAttributeName, xmlAttribute)
    instance ('Title', TitleObject);
> Insert into XMLAttributeRelationshipDefinition
               (classAttributeName, xmlAttribute)
    instance ('Author', AuthorObject);

> End schema;
```

Fig. 3. Schema definition using mapping objects

two attributes (*Title* and *Author*) (updating the CLASS and RELATIONSHIP classes of Fig. 2). The first three SQL `Insert` statements record the features of the external XML file by constructing one instance of the XMLElement class and two instances of the XMLAttribute classes. The last three `Insert` statements define the mapping between the internal and the external schemas.

Mapping objects have several advantages compared to traditional ways of describing mappings between internal and external schemas. That is:

- Mapping objects make the definition of internal and external schemas completely independent tasks. Thus, it is possible to map the same external schema into two or more internal ones and vice versa.
- In distributed systems, mapping objects can easily be shared just like any common objects can be shared.
- Since mapping objects are normal objects, they can easily be queried or updated using the traditional DML statements, such as select, update, and delete. Obviously, updating a mapping object effectively alters the mapping between the internal and the external schemas.
- It is easy to reuse mapping objects of even pieces of them. Describing the mappings between two large schemas contains many repetitions and mapping objects provide a simple way of reusing the mapping information.
- Systems using mapping objects are easily expandable to use new wrappers. This is because, apart from building the new wrapper, nothing else needs to be changed. This is in contrast, for example, with systems using mapping languages to specify the mapping information. In these systems, the mapping languages may also need to be altered.

Mapping objects represent a declarative way of defining the unidirectional or bi-directional correspondence between the internal and external schemas. A bi-directional mapping is important whenever the user needs something more than a read-only object view. Information in digital libraries does not always remain fixed and it is important for the mediation to be able to provide a centralized system that can view and update data stored in multiple disparate systems in a uniform way.

Creating a mapping object is not always a trivial task. Depending on the complexity of the mapped information, a mapping object may consist of hundreds or even thousands of sub-parts. To assist the user, it is possible to create semi-automatic tools that create an initial version of the mapping object using the semantic information available in referential constraints of the original schema.

4 Comparison with existing systems

There is a large number of mediation systems such as TSIMMIS, GARLIC, HERMES, OPM, IRO-DB, and SIMS/ARIADNE. The recent mediation systems overcome the problem of hard-wiring the mappings in the wrappers by using some mapping language. Examples of such systems and languages are the MARIAN system, which uses a digital library description language called 5S [2], the TSIMMIS Mediation Specification Language (MSL) [5], and the BRIITY (bridging heterogeneity) mapping language [4]. The use of a mapping language resembles our approach, though it can only work on storage modules that were already planned during the design of the mapping language. That is, the expressive power of these languages is limited and, therefore, they cannot describe any arbitrary mapping. Mapping objects do not share this problem, since each wrapper uses a different mapping metaschema. Integrating a new wrapper into the mediation system automatically extends its internal metaschema to handle the new type of mapping objects.

Finally, in a recent article [6], an attempt was made to overcome the limitations of mapping languages by defining a language for defining mapping languages, i.e. a metalanguage. This approach shares many of the advantages of mapping objects. However, mapping objects may present better opportunities of reuse and sharing.

5 Conclusion

We have presented a technique that allows for declarative specification of interschema mappings. This can be used in wrapper-based digital library mediation systems to enable them to support bi-directional access to disparate data sources.

References

1. Baru C, Chu V, Gupta A, Ludscher B, Marciano R, Papakonstantinou Y, Velikhov P (1999) Xml-based information mediation for digital libraries. In: Proceedings of the ACM conference on digital libraries, Berkeley
2. Concalves MA, France RK, Fox AA, Doszkocs TE (2000) MARIAN searching and querying across heterogeneous federated digital libraries. In: 1st DELOS network of excellence workshop on information seeking, searching and querying in DL
3. Frew J, Freeston M, Freitas N, Hill L, Jane G, Lovette K, Nideffer R, Smith T, Zheng Q (2000) The Alexandria digital library architecture. Int J Digital Libr 2:259–268
4. Härder T, Sauter G, Thomas J (1999) The intrinsic problems of structural heterogeneity and an approach to their solution. VLDB J 8:25–43
5. Li C, Yerneni R, Vassalos V, Garcia-Molina H, Papakonstantinou Y, Ullman J, Valiveti M (1998) Capability based mediation in TSIMMIS. In: Proceedings of the ACM international conference on management of data (SIG-MOD)
6. Melnik S, Garcia-Molina H, Paepcke A (2000) A mediation infrastructure for digital library services. In: Proceedings of the fifth ACM conference on digital libraries, June 2–7, 2000. San Antonio, TX, USA. pp 123–132
7. Paepcke A, Chang C-CK, Garcia-Molina H, Winograd T (1998) Interoperability for digital libraries world-wide. Commun ACM 41(4):33–43
8. http://dublincore.org/
9. http://www.loc.gov/standards/mets/