A Semantic Approach to XML-Based Data Integration

Patricia Rodríguez-Gianolli and John Mylopoulos

Department of Computer Science, University of Toronto,
6 King's College Road, Toronto, Canada M5S 3H5
{prg,jm}@cs.toronto.edu

Abstract. The paper describes a prototype tool, named DIXSE, which supports the integration of XML Document Type Definitions (DTDs) into a common conceptual schema. The mapping from each individual DTD into the common schema is used to automatically generate wrappers for XML documents, which conform to a given DTD. These wrappers are used to populate the common conceptual schema thereby achieving data integration for XML documents.

1 Introduction

Integrating data from multiple heterogeneous data sources has been a major focus of database research for more than two decades. Heterogeneity, in both conventional and semistructured databases, arises from the adoption of different data models and/or different schemas by two data sources. With the widespread acceptance of the Web as the primary vehicle for data interchange, interest in data integration has been renewed, with a focus on semistructured data. However, little has been proposed yet for data integration of XML documents. XML, as a standard for representing both structured text documents and data on the Web, facilitates data publishing and interchange. This is accomplished through a simple syntax which, unlike HTML, is intended for both human browsing and computer consumption. Among the many advantages of XML over HTML we note that XML separates cleanly information content from presentation details. Moreover, XML tags are user-defined and can therefore be used to describe what data mean as opposed to how they should look. Finally, XML documents can be validated against grammar-like specifications known as Document Type Definitions (DTDs). It must be emphasized, however, that XML is intended as a language for describing the syntactic structure of a document, not its meaning. This makes the data integration task a difficult one for XML data. As pointed out in [23], the key to successful data integration is the identification of interschema relationships. The more expressive the underlying data model(s), the higher the chance of identifying interschema relationships and hence the easier the task of data integration.

The DIXSE[1] system presented in this paper addresses in a semi-automatic fashion the integration problem for XML data. Unlike most approaches, we address the problem from a conceptual modeling perspective. DIXSE is capable of semi-

[1]DIXSE stands for "**D**ata **I**ntegration for **X**ML based on **S**chematic **K**nowledge".

automatically deriving a common semantic description (in the form of a conceptual schema) from a set of input DTDs and allows the user to enrich and fine-tune this description with additional domain expertise. Given the mapping from input DTDs to the common conceptual schema, DIXSE automatically generates wrappers for XML documents that conform to these DTDs and populates the conceptual schema. Full details about the DIXSE implementation and the case study that has been used to validate the approach can be found in [21].

Approaches to the problem of data integration have generally adopted a traditional schema integration approach for heterogeneous structured databases, or a (more recent) semistructured data integration approach.

In traditional schema integration research, the identification of interschema relationships can be done at different levels of abstraction. In their comprehensive schema integration survey, Ram and Ramesh [19] indicate that the abstraction level at which interschema relationship identification techniques operate defines the nature of available semantic knowledge. Approaches at the conceptual schema level can deduce relationships among objects [14,25,1]. Other approaches employ the semantics conveyed by integrity constraints [20] or data values [12] to support data integration. Regardless of the approach, the derived relationships can be modified or confirmed by human integrators. This external input can be viewed as adding domain knowledge that was not captured in the original data sources. The generation of the integrated schema is then driven by the expressiveness of the data model (relational, semantic, object-oriented or logic-based model) chosen to describe the input schemas. The DIXSE system we propose inherits and extends this approach.

Interschema relationship identification is done differently in data integration systems for semistructured data [5,16,4,13,6]. The lack of a schema in semistructured data sources makes the conceptual schema approach previously described inappropriate. The data integration approach, on the other hand, relies mostly on the query language provided by these systems. The query language supports special constructs for accessing and integrating data sources, such as query primitives for dealing with type or structure mismatches and data restructuring. A problem with semistructured data models is that they do assume a schema, but provide very few modeling abstractions (essentially, only labeled graphs) to capture semantics. Given that the main purpose of XML data is to facilitate data exchange with structure[2], we favor a semi-automatic schema integration approach rather than a data-centered one. The DIXSE system exploits the schematic information provided by XML DTDs to derive a conceptual schema of the information represented by XML data.

Related work with similar motivations has recently been presented in [15,2]. In [15], the authors also propose a semantic data model as the basis for integrating XML data sources; we differ from them by using a richer data model and a semi-automatic mechanism to derive the target schemas. In [2], pieces of information contained in XML fragments are mapped to domain specific ontologies. Unlike this work, DIXSE combines the conceptual schema definition and mapping creation into a single step. In addition, it supports the notions of user-defined keys and intradocument and interdocument links, which play a central role when performing object identification and object fusion during the data integration phase.

[2] The schema is mostly needed for interoperability.

DIXSE blends techniques from conventional and semistructured data integration systems into a framework specifically designed for XML data. A metamodeling language, Telos [17,18], is used to represent both the DIXSE data model and the derived conceptual schemas. Telos supports attribution, classification and generalization; it also offers a novel treatment of attributes, which can be exploited to define any conceptual model. Another research project that uses Telos as the target language for mapping DTDs is [7]. We differ from them by providing a user-customized mapping with an emphasis on data integration.

The rest of the paper is structured as follows. Section 2 presents a quick overview of XML and Telos. Section 3 describes the DIXSE framework for mapping DTDs to a Telos conceptual schema, while section 4 outlines the mapping language through which the user can define directives on how the mapping is to be done. Section 5 presents an overview of a case study that involves several XML documents and DTDs for SIGMOD Record publications. Section 6 describes the DIXSE architecture and implementation, while section 7 summarizes the contributions of the paper and suggests directions for further research.

2 XML and Telos

We provide a brief overview of XML and Telos. For more comprehensive information about XML and Telos, the reader is referred to [10] and [17,18], respectively.

XML is used to markup documents for purposes of presentation (like HTML) or further processing. Marked-up documents are called *XML documents*. The basic component of an XML document is the *element*, that is, a piece of text bounded by matching tags such as `<article>` and `</article>`. The content of an element can be raw text, other elements or a combination of the two. The term *subelement* is used to describe the relationship between an element and its component elements. In addition, elements may contain *attributes*. Attributes are "name-value" pairs specified in the start tag of an element. The structure of an XML document can be described by a *Document Type Definition* (DTD). A DTD provides a list of elements and attributes contained in a document and the relationships between them. *Element type declarations* describe the name of the tag being declared (e.g. `article`) and the allowed contents of that tag, usually referred to as *content specification*. Attributes are declared for specific element types using *attribute-list declarations*.

Telos provides facilities for constructing, querying and updating structured *knowledge bases*. A Telos knowledge base consists of structured objects built out of two kinds of primitive units: *individuals* and *attributes*. The first ones are intended to represent entities, while the second ones represent binary relationships between entities or other relationships. Individuals and attributes are treated uniformly by the mechanisms of structuring a knowledge base. The term *proposition* is used to denote either an *individual* or an *attribute*. Propositions are organized along three dimensions: *attribution*, *classification* and *generalization*. The first relates a proposition to the values of all its attributes (by default, attributes in Telos are multivalued). The classification dimension calls for each proposition to be an instance

of one or more generic propositions or *classes*. Propositions (both individuals and attributes) are classified into *tokens* (propositions having no instances and representing concrete objects in the modeled domain), *simple classes* (propositions having tokens as instances; these represent generic concepts), *metaclasses* (propositions having simple classes as instances), *metametaclasses* and so on. In addition, *ω-classes* are propositions with instances from any level. Finally -- and orthogonal to the classification dimension -- class propositions can be organized in terms of generalization or ISA hierarchies. In general, there will be one such hierarchy for each classification level (i.e., for simple classes, metaclasses, etc.).

3 The DIXSE Framework

The DIXSE framework supports the derivation of a conceptual schema from several input DTDs. The data model used for the conceptual schema offers concepts such as entity, attribute, and mapping. Since Telos is a language for metamodeling, the data model can be extended with additional semantic primitives (e.g., activity, goal, agent) depending on the semantics of the information that is to be integrated. Figure 1 illustrates the structure of a Telos representation of the DIXSE XML model (at the MetaClass and ω levels), a DIXSE conceptual schema for an XML DTD (at the SimpleClass level) and some XML data (at the token or object level). Planes in the figure depict classification levels; gray links represent "instance-of" links between two consecutive classification levels or between a classification level and the ω-level.

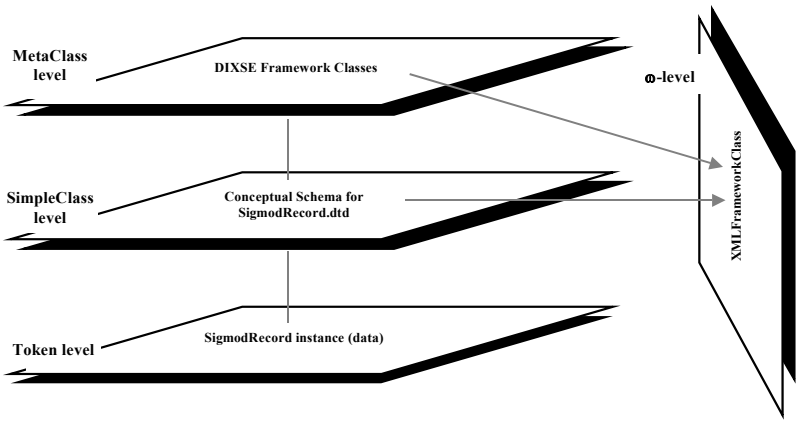

Fig. 1. Telos representation of the DIXSE data model

3.1 The Data Model

Conceptual schemas are represented in DIXSE as collections of entity types and their attributes. The model supports four main concepts: *entity class*, *entity attribute*,

mapping and *document type*. An *entity class* represents types of objects or concepts found in the input DTDs. Figure 2 shows a DTD that describes the structure of a hypothetical XML-based SIGMOD Record database. In this schema, there are at least four XML elements that represent different types of entities: `SigmodRecord`, `Issue`, `Article` and `Author`. The rest of the elements and attributes (such as `volume`, `title`, `contactAuthor`, etc) may be thought as *attributes* of these entity types. *Mappings* describe particular conceptual schemas of the information represented by DTDs. A mapping may be thought as a wrapper for a collection of interrelated entity classes. Finally, *document types* describe DTDs and the collection of mappings attached to them.

```
<!ELEMENT SigmodRecord (issue)+>
<!ELEMENT issue (volume,number,articles)>
<!ELEMENT volume (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT articles (article)* >
<!ELEMENT article (title,numberOfPages,fullText,contact,authors)>
<!ATTLIST contact (EMPTY)>
<!ATTLIST contact contactAuthor IDREF #IMPLIED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT numberOfPages (#PCDATA)>
<!ELEMENT fullText (size?)>
<!ATTLIST fullText xLink:type CDATA #FIXED 'simple'>
<!ATTLIST fullText xLink:href CDATA #IMPLIED>
<!ELEMENT size (#PCDATA)>
<!ELEMENT authors (author)*>
<!ELEMENT author (name,address)>
<!ATTLIST author organization CDATA #IMPLIED>
<!ATTLIST author degree (bachelor|master|phd) "phd")
<!ATTLIST author id ID #REQUIRED>
<!ELEMENT name (firstName?,lastName)>
<!ELEMENT firstName (#PCDATA)>
<!ELEMENT lastName (#PCDATA)>
<!ELEMENT address (home|office)>
<!ELEMENT home (#PCDATA)>
<!ELEMENT office (#PCDATA)>
```

Fig. 2. XML DTD SigmodRecord.dtd

Figure 3 offers a complete Telos description of the DIXSE data model in semantic network notation. The figure illustrates the different relationships among DIXSE concepts and how Telos has been used to model them. To improve readability, we have used several drawing conventions. Ellipses represent classes; gray links represent "instance-of" links while black links represent attributes.

In addition to the distinction between entity classes and attributes, the data model classifies attributes along two orthogonal dimensions. The first facility distinguishes three categories of attributes, namely **components**, **properties** and **links**. An attribute is a **component** when it represents one component (of the structure) of an entity (e.g. XML element `name`). An attribute is a **property** when it represents information about the content of an entity (e.g. `organization` CDATA attribute). Finally, a **link** attribute represents intradocument or interdocument information (e.g. `contactAuthor` IDREF and `href` XLINK attributes).

The distinction between components and properties in our data model is inspired by the difference between XML elements and attributes, according to the XML 1.0 Recommendation [10]. In addition, we have chosen to model IDREF and simple

XLINK attributes as special **link** attributes because they represent special relationships among XML data. Through this distinction, the DIXSE model recognizes the different roles that each entity attribute plays at the time of data integration.

Attribute categories are represented in Telos using three attribute metaclasses: "hasComponents", "hasProperties" and "hasLinks". Instances of metaclass XmlPropertyClass (i.e. XmlCDATAProperty, XmlIDProperty and XmlENUMProperty) model different types of XML properties. Each of these attribute metaclasses contains as instances particular entity attributes (i.e. components, properties or links).

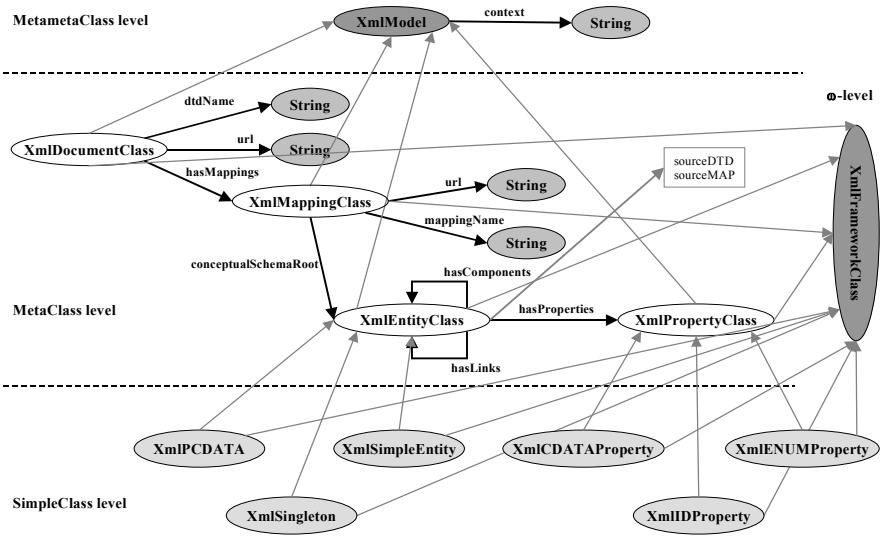

Fig. 3. Telos definition of the DIXSE data model

The second dimension for classifying attributes models frequently used constraints or characteristics of attribute values, regardless of their category. These constraints are inspired by the constraints that XML itself imposes on elements and attributes, namely: the occurrence and choice indicators for XML elements (i.e. "?", "*", "+" and "|") and the attribute modifiers for XML attributes (i.e. "#FIXED", "#REQUIRED" and "#IMPLIED").

Attribute constraints are modeled in Telos using nine attribute metaclasses at the ω-level. Attribute metaclasses "**exactlyOne**", "**atMostOne**", "**zeroOrMore**" and "**oneOrMore**" are used to model mandatory or optional single-valued attributes (the first two) or multi-valued attributes (the second two). The attribute metaclass "**union**" is used to restrict an attribute defined in a given class to be exclusive with respect to other "union" attributes (i.e. only one has a value). The attribute metaclass "**fixed**" is used to restrict an attribute to have a fixed (user-supplied) value. Attribute metaclasses "**idRef**" and "**xLink**" restrict an attribute to be the recipient of an

intradocument or interdocument reference, respectively. Last, the attribute metaclass "**key**" restricts an attribute to have exactly one single and unique value. Instances of class XMLFrameworkClass can have a key composed of more than one attribute.

A *mapping* consists of a conceptual schema which models the information represented by a given DTD, typically authored for a given context. Mappings attempt to capture the meaning, interpretation or intended use of the data sources to be integrated. Different perspectives or views of the data may lead to different models of relationships among objects. The application context in which relationships are stated is fundamental for capturing the real-world semantics that will drive the data integration process (e.g. a bibliographical library context versus a scientific e-mail directory one).

Finally, a *document type* describes a given DTD and a collection of mappings (i.e. conceptual schemas) attached to it. *Contexts* are represented in the data model as distinguishing attributes (string names) of document types, mappings and entity classes. At a higher level, contexts may be thought of as partitions of the semantic model. Each partition clusters information about a specific application domain, and provides the appropriate framework for the creation of unique entity classes and their instances (that is, two conceptual schemas using entity class `Author` refer to the same concept if they belong to the same context).

Mappings and document types are represented in Telos using two metaclasses: XmlMappingClass and XmlDocumentClass, respectively. To record the origin of entity attributes, we define two attributes metaclasses (sourceMap and sourceDTD) using Telos' *"attributes on attributes"* feature.

3.2 Default Mapping

The DIXSE framework supports the derivation of a default conceptual schema for a given DTD. This mapping is based on heuristic rules on what DTD constructs usually represent, and thus captures only partially the semantics conveyed by the data. The main value of this mapping mechanism is that it offers a starting point for a user-defined mapping for a given DTD. Additional domain expertise or contextual knowledge can be added into the default mapping through specifications written in DIXml (see Section 4).

The conceptual schema derivation process takes a single DTD as input and generates a DIXSE conceptual schema as output. Basically, the process analyzes the schematic information provided by the DTD and mines a conceptual representation of it by applying a set of *heuristic rules*. These rules were discovered by manually trying out derivations, and analyzing the results. As a whole, they aim at identifying a correspondence between elements and attributes of DTDs and entity classes and attributes of the DIXSE model. Since mapping directly all elements with structure into entity classes is likely to lead to excessive fragmentation of the mapping, they also intend to reduce the number of entity classes. Only the most specific rule is applied. Below we describe two mapping rules that drive the derivation process and illustrate their use with examples from "SigmodRecord.dtd".

The first default mapping rule (DR1) maps an XML element with complex content model (i.e. excluding #PCDATA, ANY and EMPTY types) into a DIXSE *entity*

class. In addition, it creates: a *component attribute* for each subelement, a *property attribute* for each XML CDATA, ID and ENUM attribute, and a *link attribute* for each XML IDREF or XLINK attribute. For example, this rule is applicable to the `author` element. The following Telos specification illustrates its mapping:

```
SimpleClass Author in XmlEntityClass,
                      XmlFrameworkClass with
  hasComponents,exactlyOne
    name : Name;
    address : Address
end
```

Another rule (DR3) maps the XML attributes of an EMPTY element into DIXSE *entity attributes*. The element's attributes are collapsed (or inlined) as either *property* or *link attributes* of immediate parent elements' entity classes. Each entity attribute is named with the result of concatenating the EMPTY element name, the string "_" and the proper XML attribute name. EMPTY elements without attributes are not mapped into DIXSE. This rule can be applied to the `contact` element; the following link attribute definition is included in the element `article`'s entity class:

```
SimpleClass Article in XmlEntityClass,
                       XmlFrameworkClass with
  hasLinks,atMostOne,idRef
    contact_contactAuthor : XmlSimpleEntity
end
```

The complete set of default mapping rules can be summarized as follows. The first two (DR1 and DR2) are concerned with identifying XML elements that represent entities. For us, elements with structure or atomic elements with distinguishing characteristics (like an ID attribute) embody the notion of a concept or an entity. Therefore, such elements are considered good candidates for DIXSE entity classes. Rules DR3 and DR4 describe how the remaining elements can be mapped into entity attributes. Rules DR5 and DR6 indicate how constraints on element values can be translated into additional knowledge (DIXSE attribute constraints). Last, rule DR7 recognizes the fact that some elements in the DTD (in particular those representing "*" or "+" lists, like the `articles` element) function more as delimiters (non-terminals in the grammar) than as meaningful entities. This rule helps to reduce the number of entity classes in the derived schema.

Figure 4 shows the default conceptual schema derived for "SigmodRecord.dtd". Please note that only some "instance-of" links with respect to DIXSE metaclasses are shown to keep the figure relatively uncluttered. Unless otherwise specified, all entity attributes in the figure are instances of the "hasComponents" attribute category. According to this mapping, there are seven entity classes. Interestingly, users with specific domain expertise may indicate that the entity classes `FullText`, `Address` and `Name` do not represent objects in the underlying data sources. Moreover, they could argue that the derived mapping does not convey much of the implied semantics. To overcome these deficiencies, DIXSE offers the possibility of customizing the default mapping.

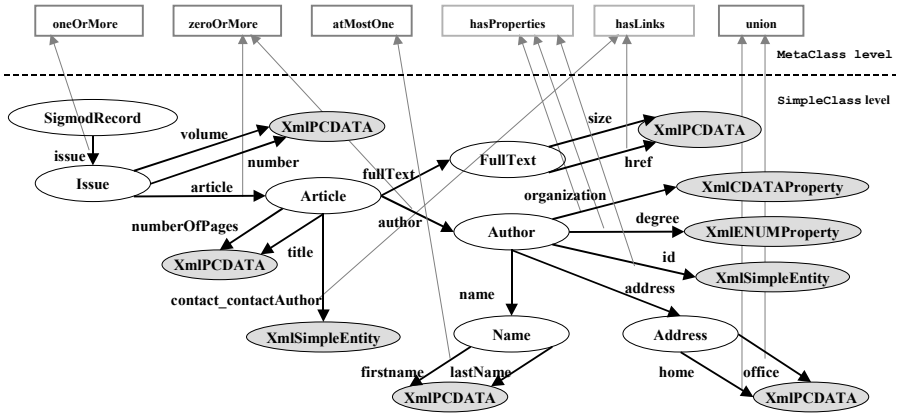

Fig. 4. Default Mapping for SigmodRecord.dtd

4 The Mapping Language

DIXml[3] is a declarative mapping language for specifying a DTD mapping to a conceptual schema. This specification annotates a DTD with simple instructions for generating entity classes from DTD element type declarations. In the same spirit of XSL stylesheets [11], DIXml specifications accompany DTDs to provide an extra level of information to XML data.

The DIXml specification model revolves around the idea of directive rules. A *directive rule*, hereinafter called directive, is the main mapping construct offered by the language. A DIXSE mapping can be defined by a collection of directives (at most one per DTD element declaration type). Each directive describes particular preferences for generating a conceptual representation of an XML element. These preferences are combined with the default mapping rules (presented in Section 3.2) to produce a conceptual schema.

There are five different *directive actions*, namely: **default**, **create-class**, **create-attribute**, **inline**, and **ignore**. The **default** directive indicates that the default mapping rules (DR1 to DR7) should be applied to the target element. Its inclusion or omission in the mapping specification does not affect the generated conceptual schema. The **create-class** directive says that the target element should be mapped into an entity class, while the **create-attribute** directive indicates that the target element should be mapped into a component attribute of immediate parent entity classes, without creating an entity class for it. On the other hand, the **inline** and **ignore** directives indicate that neither an entity class nor a component attribute should be created for the target element. In the first case, the meaning of the **inline** directive is to collapse the

[3] The acronym DIXml stands for "**D**ata **I**ntegration for **X**ML **m**apping **l**anguage".

target element's content (both XML subelements and attributes) into entity attributes of immediate parent entity classes. This directive works as a grammar re-writing rule that can be applied to XML elements with content models made of atomic or empty subelements[4]. Last, the **ignore** directive indicates that neither the target element nor its content should be included in the derived mapping, unless otherwise specified (e.g. **keep** component directive for a particular subelement).

In addition, the user can provide supplementary information with each of these directives. For example: a different entity class name or attribute label (`AS` and `label` clauses); classification or generalization relationships with respect to previously created entity classes (`IN` and `ISA` clauses); an indication for mapping XML attributes into entity properties (`properties` clause); explicit identification of interschema relationships (`xLink` clause); etc.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- DTD File: DIXSEmapping.dtd  -->

<!-- DIXSE-MAPPING specification -->

<!ELEMENT DIXSEmapping (directive*)>
<!-- Attributes for DIXSEmapping -->
<!ATTLIST DIXSEmapping name CDATA #REQUIRED>
<!ATTLIST DIXSEmapping dtd CDATA #REQUIRED>
<!ATTLIST DIXSEmapping context CDATA #IMPLIED>

<!ELEMENT directive ( default
                    | create-class
                    | create-attr
                    | inline
                    | ignore ) >
<!-- Attributes for directive -->
<!ATTLIST directive elem CDATA #REQUIRED>

<!-- DEFAULT directive -->

<!ELEMENT default EMPTY>

<!-- CREATE-CLASS directive -->

<!ELEMENT create-class (component*)>
<!-- Attributes for create-class -->
<!ATTLIST create-class AS CDATA #IMPLIED>
<!ATTLIST create-class IN CDATA #IMPLIED>
<!ATTLIST create-class ISA CDATA #IMPLIED>
<!ATTLIST create-class properties (on|off) 'on'>
<!ATTLIST create-class xlink CDATA #IMPLIED>
```

```
<!ELEMENT component ( default
                    | ignore
                    | keep
                    | subclassing
                    | union )>
<!-- Attributes for component -->
<!ATTLIST component num CDATA #REQUIRED>
<!ATTLIST component label CDATA #IMPLIED>
<!ATTLIST component AT-class CDATA #IMPLIED>
<!ATTLIST component key (yes|no) 'no'>
<!ATTLIST component WITH-class  CDATA #IMPLIED>

<!-- CREATE-ATTRIBUTE directive -->

<!ELEMENT create-attr EMPTY>
<!-- Attributes for create-attr element -->
<!ATTLIST create-attr WITH-type CDATA #IMPLIED>
<!ATTLIST create-attr properties (on|off) 'on'>
<!ATTLIST create-attr xlink CDATA #IMPLIED>

<!-- INLINE directive -->

<!ELEMENT inline (component)*>
<!-- List of attributes for inline -->
<!ATTLIST inline naming (on|off) 'off'>
<!ATTLIST inline properties (on|off) 'on'>
<!ATTLIST inline xlink CDATA #IMPLIED>

<!-- IGNORE directive -->

<!ELEMENT ignore EMPTY>
```

Fig. 5. DIXml syntax

A **create-class** or **inline** directive body can have zero or more component elements, where each one describes local mapping considerations for a subelement. For example, this allows choosing where the attribute definition should take place or indicating if the component should be part of the entity class key. The system supports a multivalued key per entity class. A component attribute can be part of the entity class key only if its value is single and mandatory. There are five possible component directives: *default*, *ignore*, *keep*, *subclassing* and *union*. The first one indicates that the default mapping rules should be applied to the component. The *ignore* and *keep* component directives allow us to explicitly discard or consider the

[4] This constraint prevents recursive collapsing of XML elements.

element as a candidate component attribute. These directives serve to override any global directive stated for this particular XML element (i.e. **ignore** directive action). Last, the *subclassing* and *union* directives provide explicit instructions on how to map simple alternative components.

DIXml Specifications.
Mapping specifications are written in XML. DIXml, as a markup language in its own right, provides a vocabulary to describe DIXSE mappings. The main two elements in this vocabulary are `directive` and `DIXSEmapping`. The first one represents a DIXml *directive rule*, while the second one represents the mapping itself by encompassing the collection of specified *directive rules*. Figure 5 shows the "end-user" syntax of DIXml, given as a DTD grammar.

5 The Case Study

We illustrate DIXSE's data integration approach through a case study. The case study we have adopted is based on the XML version of the ACM SIGMOD Record database [27]. The repository, built by the Araneus Group [3], contains a collection of approximately 1,000 XML documents. The documents are classified according to four different DTDs: `HomePage.dtd`, `ProceedingsPage.dtd`, `OrdinaryIssuePage.dtd` and `IndexTermsPage.dtd`. The `HomePage` DTD describes XML documents that represent the SIGMOD Record database as a collection of issues. Issues are firstly organized per year of edition, and then per number. Each issue contains some information about its volume, number and conference details. The `ProceedingsPage` and `OrdinaryIssuePage` DTDs describe two different types of issues: conference proceedings and ordinary issues, respectively. Regardless of whether an issue represents a conference proceedings or an ordinary issue, it is characterized by volume, number and date information, along with a collection of sections and articles. Each article has a title and a list of authors, among other relevant information. Last, the `IndexTermsPage` DTD describes specific indexing information for a SIGMOD Record article (such as its title, abstract, and a list of index terms).

Using the same application context, we process the four DTDs through the DIXSE system. The result of the process is an integrated conceptual schema in the DIXSE repository that encompasses four default mappings (one per each DTD). Figure 6 illustrates the configuration resulting from this schema integration process.

The integrated schema includes fourteen entities. The shared entity classes (i.e. classes whose name is used in more than one conceptual schema definition) represent the points where integration will occur when XML documents are uploaded. In particular, integration will occur for instances of `SectionListTuple`, `ArticlesTuple`, `Author` and `FullText`. For example, after loading several XML documents representing SIGMOD Record issues, the instances of the `Author` entity class will be all author objects described in these documents (that is, one object per XML element `author`). This would allow us to uniformly query the DIXSE

repository and, for example, retrieve at the conceptual schema level all the author names that have written an article regardless of where the article was published (i.e. Proceedings or Ordinary issue).

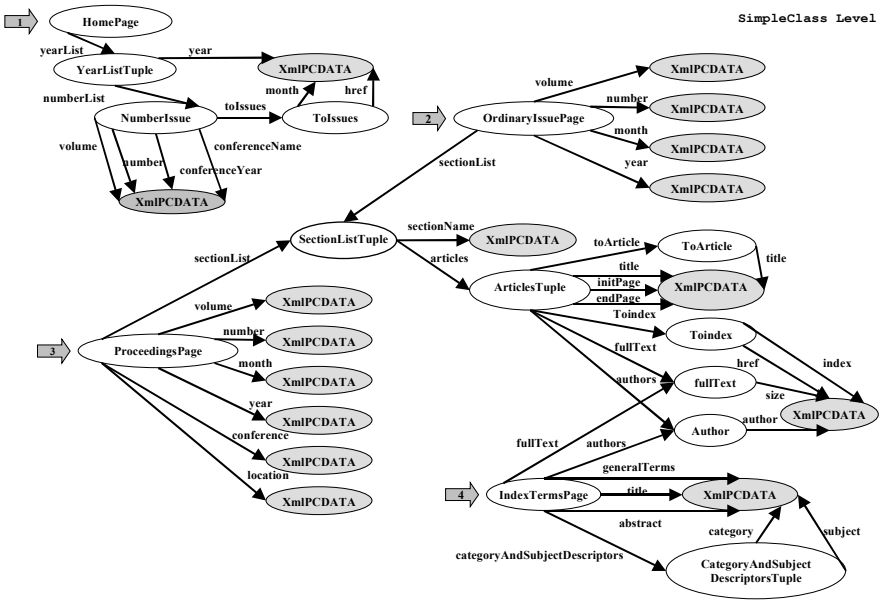

Fig. 6. Default Integrated Conceptual Schema

In spite of the above results, the integrated schema does not capture much of the implied semantics of the data. First, it does not model the relationship between the `HomePage` DTD and the remaining three DTDs. As shown in Figure 6 (see block arrow number 1), the `HomePage` default conceptual schema is completely isolated from the rest. Moreover, the relationships between Proceedings and Ordinary issues, on one hand, and Proceedings and Index Terms, on the other, are only stated in terms of the sharing of entity classes (see block arrows number 2, 3 and 4). Second, the integrated schema does not provide information on how to uniquely identify objects from the XML descriptions. Having the same author name in several XML documents will not be considered as the same object. Therefore, a new `Author` token will be created every time DIXSE uploads an XML document with an `author` element.

To overcome these deficiencies, we need to provide additional domain knowledge through DIXSE mapping specifications. Figure 7 depicts the integrated conceptual schema obtained after a series of improvements based on our understanding of the semantics of the SIGMOD data (numbered block arrows indicate the entry points for each separate conceptual schema).

The main difference between the two schemas is that the latter models the intended interpretation and usage of the information represented by the SIGMOD Record XML documents, and thus serves as a better guide for data integration. This is demonstrated

by the following characteristics: use of meaningful entity class and attribute names (e.g. SigmodRecord instead of HomePage), distinction between two types of SIGMOD Record issues and their common and specialized information (illustrated by "isA" links in Figure 7), identification of key attributes (depicted by dashed-labeled attributes) and explicit assertion of interschema relationships (shown as thick-labeled attributes). As explained in Section 4, DIXSE seamlessly combines these additional knowledge sources with the schematic information provided by the DTDs.

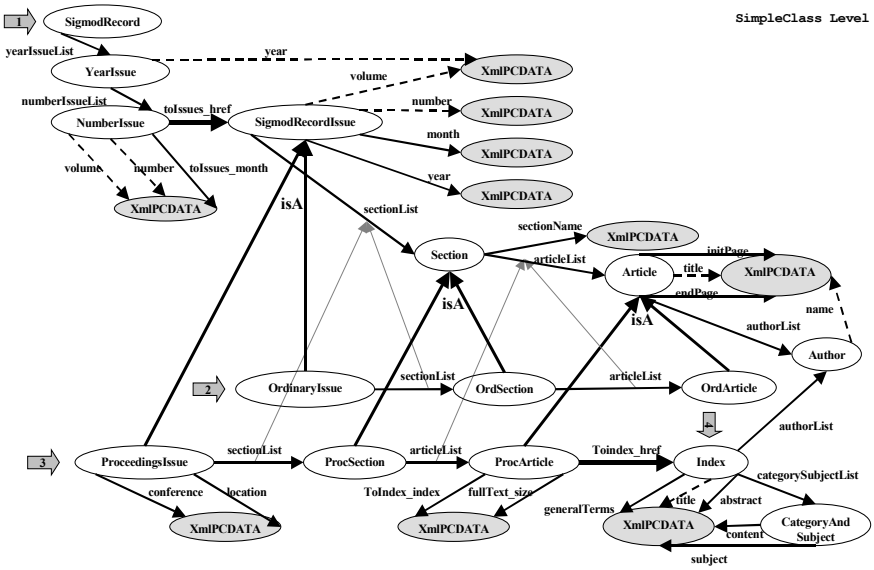

Fig. 7. User-customized Integrated Conceptual Schema

6 System Architecture

The DIXSE system is based on a data warehouse approach to data integration. The storage mechanism employed is ConceptBase [8], an object base management system that implements a version of the Telos language. The system, implemented in Java 2, comprises two main subsystems: the `Schema Engine` and the `Document Loader`. The first one allows the user to register XML DTDs into the repository, while the second one allows populating the repository with a collection of XML documents conforming to registered DTDs.

Figure 8 depicts the system architecture and the various interactions among its components. The `Schema Engine` subsystem includes five components: the DTD parser, the XML parser, the Schema Derivator, the Schema Generator and the XSL Wrapper Generator. On the other side, the `Document Loader` consists of the XSL

Processor and the Data Integrator. The communication between these two subsystems is accomplished through the Catalog Manager and the XSL Wrapper Repository.

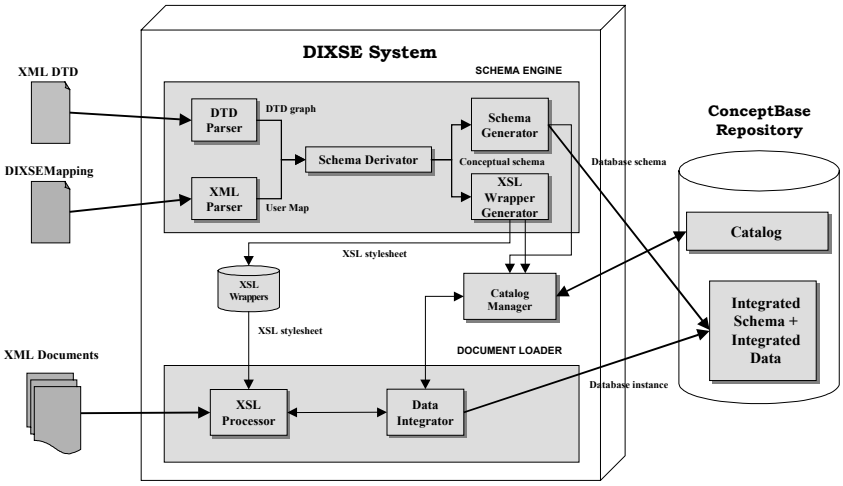

Fig. 8. System Architecture

We use the Xerces Java Parser and Xalan Java Processor as off-the-shelf components [24] (i.e. the *XML Parser* and the *XSL Processor*, respectively). Although Xerces is equipped with a *DTD Parser*, it doesn't provide external access to the grammar representation it builds. Thus, we have modified and extended the Xerces source code to provide both individual parsing of DTDs and access to their representations (DTD graphs).

The *Schema Derivator* applies a set of mapping directives to derive a conceptual schema from a DTD graph. These directives combine the knowledge embedded in the set of heuristic rules with any user-customized mapping information, provided in the form of a DIXSE mapping. The output conceptual schema is then used to generate the target schema definition and the corresponding wrapper (i.e. an XSL stylesheet), capable of converting XML documents that comply with the DTD into instances of the generated schema. Information about the generated database schema and wrapper is registered into the repository catalog.

At the time of loading an XML document into the ConceptBase repository, the loader retrieves the corresponding XSL stylesheet that will be used to transform the input document into a database instance. At several points, the *XSL Processor* interacts with the *Data Integrator*, which basically performs management of keys, object identification, and IDREF/XLINK value translation. These tasks are done using both conceptual schema and XML data information. The repository catalog is queried and updated during this process. Finally, the ConceptBase repository is populated with the newly created database instance.

7 Conclusions

This paper proposes a semantic framework for XML data integration. The framework, named DIXSE, has been implemented and evaluated with a case study. DIXSE offers a tool, which can be used semi-automatically to generate a conceptual schema from several DTDs. The tool can then parse XML documents to populate the conceptual schema.

Our approach differs from state-of-art data integration systems with respect to three main aspects. Firstly, and unlike [5,16,4,22,9], DIXSE employs conceptual modeling ideas to support data integration at a semantic level rather than at a logical level. Based on heuristics and user input, DIXSE extracts semantic details from DTDs to derive a conceptual schema. Secondly, the DIXSE approach is based on schema integration ideas like many conventional data integration systems [14,25,1]. But unlike them, DIXSE allows the user to enrich and fine-tune the default mapping derived from a set of DTDs. Finally, DIXSE employs a specialized object-based repository to store an integrated and semantically richer version of the data extracted from a collection of heterogeneous XML data sources. With the help of Telos, DIXSE provides a single framework for uniformly representing and querying conceptual schema information (metadata) and data. The paper includes elements of a detailed case study to illustrate the DIXSE approach.

We plan to try out DIXSE with another case study from the domain of knowledge management to illustrate its usefulness. We also propose to use XML Schema definitions [26] in place of DTDs, extend the implemented system so that it can produce an XML Schema definition from the derived conceptual schema, and generate a single XML document that integrates all the data included in the input XML documents. We expect that these extensions will be quite straightforward.

Acknowledgements.
This research was partly funded by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

1. Ahmed, R., De Smedt, P., Du, W., Kent, W., Ketbachi, M., Litwin, W., Raffi, A. and Shan, M.: The Pegasus heterogeneous multidatabase system. In IEEE Computer, 24(12):19-27 (1991)
2. Amann, B., Fundulaki, I., Scholl, M.: Mapping XML Fragments to Community Web Ontologies. In 4th WebDB Workshop (2001)
3. Atzeni, P., Mecca, G., Merialdo, P.: To Weave the Web. In 23rd VLDB Conference (1997)
4. Baru, C., Gupta, A., Ludäscher, B., Marciano, R., Papakonstantinou, Y., Velikhov, P., Chu, V.: *XML-Based Information Mediation with MIX*. In ACM SIGMOD International Conference (1999)
5. Chawathe, S., García-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J.: The TSIMMIS Project: Integration of heterogeneous information sources. In 16th Meeting of the IPSJ (1994)

6. Christophides, V., Cluet, S., Siméon, J.: On Wrapping Query Languages and Efficient XML Integration. In ACM SIGMOD International Conference (2000)
7. Christophides, V., Dörr, M., Fundulaki, I.: A Semantic Network Approach to Semi-Structured Documents Repositories. In 1st ECDL Conference (1997)
8. *ConceptBase.* www-i5.informatik.rwth-aachen.de/Cbdoc (1998)
9. Deutsch, A., Fernandez, M., Suciu, D.: Storing Semistructured Data with STORED. In ACM SIGMOD International Conference (1999)
10. Extensible Markup Language (XML) 1.0, W3C Recommendation. www.w3.org/TR/Rec-xml (2000)
11. Extensible Stylesheet Language (XSL) 1.0, W3C Recommendation. www.w3.org/TR/xsl (1999)
12. Holsheimer, M., Siebes, A.: Data Mining: The Search for Knowledge in Databases. Technical report CS-R9406, Amsterdam: CWI (1994)
13. Ives, Z., Florescu, D., Friedman, M., Levy, A., Weld, D.: An adaptive query execution system for data integration. In ACM SIGMOD International Conference (1999)
14. Larson, J., Navathe, S., El-Masri, R.: A theory for attribute equivalence and its applications to schema integration. In IEEE Transactions on Software Engineering, 15(4):449-463 (1989)
15. McBrien, P., Poulovassilis, A.: A Semantic Approach to Integrating XML and Structured Data Sources. In 13th CAiSE Conference (2001)
16. McHugh, J., Abiteboul, S., Goldman, R., Quass, D., Widom, J.: *Lore:* A database management system for semistructured data. In ACM SIGMOD Record, 26(3):54-66 (1997)
17. Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos: Representing Knowledge about Information Systems. In ACM Transactions on Information Systems, 8(4):325-362 (1990)
18. Mylopoulos, J.: Conceptual Modeling and Telos. In P. Loucopoulos and R. Zicari, editors, "Conceptual Modeling, Databases and Case", pages 49-68, Wiley (1992)
19. Ram, S., Ramesh, V.: Schema Integration: Past, Present and Future. In A. Emalgarmid, M. Rusinkiewicz, and A. Sheth, editors, "Management of Heterogeneous and Autonomous Database Systems", pages 119-155, Morgan Kaufmann (1999)
20. Ramesh, V., Ram, S.: Integrity constraint integration in heterogeneous databases: An enhanced methodology for schema integration. In Information Systems 22(8):423-446 (1997)
21. Rodríguez-Gianolli, P.: Data Integration for XML based on Schematic Knowledge. Master's Thesis, Department of Electrical and Computer Engineering, University of Toronto (2001)
22. Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., DeWitt, D., Naughton, J.: Relational Databases for Querying XML Documents: Limitations and Opportunities. In 25th VLDB Conference (1999)
23. Sheth, A., Gala, S.: Attribute relationships: An impediment in automating schema integration. In NSF Workshop in Heterogeneous Databases (1989)
24. The Apache Software Foundation. www.apache.org (1999)
25. Thieme, C., Siebes, S.: Schema integration in object-oriented databases. In 5th CAiSE Conference (1993)
26. XML Schema, W3C Recommendation. www.w3.org/TR/xmlschema-0 (2001)
27. XML version of the ACM SIGMOD Record database. www.dia.uniroma3.it/Araneus/Sigmod (1999)