Using Correspondence Assertions for Specifying the Semantics of XML-Based Mediators

Vânia Maria Ponte Vidal
Bernadette Farias Lóscio
Ana Carolina Salgado

1Departamento de Computação - Universidade Federal do Ceará - vvidal@lia.ufc.br
2Centro de Informática - Universidade Federal de Pernambuco - {bfl,acs}@cin.ufpe.br

Abstract

Mediators are facilities that support an integrated view over multiple information sources, and allow for queries to be made against the integrated view. To facilitate an uniform and flexible representation of arbitrary source data, there is an increasing interest in using XML for data exchange and integration. In XML-based integration systems, instance information is represented by XML documents and schema information is represented by XML DTD.

In this work, we propose the use of correspondence assertions to formally specify the relationship between the mediator schema and the source databases schemas. In that way, the mediators correspondence assertions semantically specify how the mediator objects are synthesized from the sources’ objects. The advantage of our formalism is that it can manage the problem of semantic heterogeneity, the fact that mediator data is represented differently in the underlying information sources. In this paper we discuss how the semantic specification of mediators can be used to automate some aspects of data integration.

1 Introduction

In the recent years, the number of applications requiring integrated access to multiple information sources that can be autonomous and heterogeneous has increased immensely. The problem of information integration in databases has been strongly studied in the literature [16,17,18] and, more recently, in the Web environment. Several systems [1, 2, 3, 4] have been built with the goal of integrating information from multiple web sources.

Data integration architectures are usually based on mediators, which are facilities that support an integrated view over diverse information sources. A schema for the integrated view is available from the mediator, thus allowing queries and updates to be made against that schema. The mediated integrated view can be either virtual or materialized. In the virtual approach, the data remains in the local sources, and, therefore, queries submitted to the mediator are decomposed at run time into queries on the local sources. The results from these queries on the local sources are translated, filtered and merged, and then, the final answer is returned either to the user or to the application. In the materialized approach, on the other hand, information from each source database is extracted in advance and then, translated, filtered, merged and stored in a centralized repository. Thus, when a user’s query arrives, it can be evaluated directly at the repository, and, therefore, no access to the source database is required. In this work, we consider only virtual integrated views.

One of the difficulties in integrating information from multiple information sources is the heterogeneous structure of the data sources. The integration systems use a common data model for representing the sources’ content and structure. Due to the flexibility of XML to represent both structured and semi-structured information, and to the ease with which one can convert any data to XML[5], there is an increasing interest
in using it as a common data model for data exchange and integration, which would provide a uniform and flexible representation of data from an arbitrary data source.

In XML-based information integration systems, the mediated view is defined in a declarative language specifically designed for XML [13,14,15]. The mediated view definition consists of a sequence of rules or queries that describe, in a declarative manner, how queries against the mediator’s schema can be mapped onto queries against the underlying data sources. Specifying the mediated view definition usually requires a fair amount of knowledge about the concepts in the underlying data sources and about the correspondences between those concepts and the ones in the mediator’s schema.

In this work, we propose the use of correspondence assertions to formally specify the relationship between the mediator’s schema and the source databases’ schemas. In this way, the mediator’s correspondence assertions semantically specify how the mediator’s objects are synthesized from source objects. Thus, it is possible to have a better understanding of the semantics associated with the local schemas and with the mediator’s schema. We show in this paper that with our formalism one can handle the problem of semantic heterogeneity - the fact that the mediated view is represented differently in the underlying information sources [6]. We also show that our formalism is advantageous in automating some aspects of data integration in XML-based information integration systems. In this paper, we show how the correspondence assertions can be used for helping to generate the mediated view definition.

The remainder of this paper is organized as follows. In Section 2, we present the data model used to represent the schema of the mediated view and the source databases’ schemas. In Section 3, we present the process that we propose for building mediated views. In Section 4, we present our conclusions and future work.

2 Terminologies

In XML-based information integration systems, one represents instance information with XML documents and intentional information with XML DTD. A DTD serves both as a schema formalism for the data represented by XML documents, and as a grammar for the underlying XML documents. DTDs are unsatisfactory as schema formalisms and, for that reason, several XML schema languages have been proposed []. However, beyond the basic DTDs, there is no consensus, so far.

In this work, DTDs are translated to the YAT model [ ] in order to abstract out their intentional information. The YAT model, which is based on ordered trees with labeled nodes, nicely abstracts the modeling primitives supported in XML DTDs. To illustrate this model’s ideas, consider for example, the XML DTD presented in Figure 2.1. It contains information about patients and doctors from an emergency sector of a hospital.

Figure 2.2 illustrates the tree model of the DTD for the Emergency. Note that (i) bold fonts denote tree identifiers, (ii) the & symbol denotes references and (iii) the *
symbol denotes multiple occurrences of an element. A tree model consists of a set of trees whose nodes are labeled with data variables. In addition, leaves may also be labeled with tree names (e.g. doctor) or references to trees (e.g. &doctor). The later resembles references to objects in a standard object model. The tree model in Figure 2.2 specifies that an emergency element (the root element) contains one doctors element and one patients element. A doctors element contains one or more doctor element which contains name and phone elements. A doctor element also contains an id_doc attribute, which allows a unique identification to be associated with each doctor element. A patients element contains one or more patient element, each containing name, phone and id_pat elements. A patient element also contains a doc attribute, which is a reference to a doctor element.

![Tree model of the DTD for the Emergency Sector](image)

Figure 2.2 – Tree model of the DTD for the Emergency Sector

We also use the YAT model to abstract out the information content of XML documents. As an example, consider the XML document presented in Figure 2.3 that conforms to the DTD for the Emergency Sector. Figure 2.4 illustrates the tree structure of this XML document. The leaves of the tree contain atomic values (e.g. “Steve”) while internal nodes (e.g. name) correspond to XML elements or attributes, and capture the document structure.

We use the notion of a path expression [ ] to navigate following a path on a data tree to retrieve that path’s children. We view a path expression as a simple query whose result, for a given data tree, is a set of nodes (i.e. objects). For example, the result of the path expression emergency.patients.patient* in Figure 2.4 contains the nodes p₁ and p₂. We can also navigate through the inverse of a path expression. For example, we can use the inverse of the path expression patient*.doc to obtain the patients of a given doctor. Suppose, for example, that “d” is a doctor element with id=”d12” in emergency.doctors.doctor*. The result of d.(patient*.doc)¹ contains the set of all nodes “p” in emergency.patients.patient* such that p.doc = “d12”.

![XML Data for the Emergency Sector](image)

Figure 2.3 – XML Data for the Emergency Sector
XML can hold a wide variety of data, ranging from irregularly structured data and semi-structured data to highly structured data such as those from relational databases. Consider, for example, the relational database presented in Figure 2.5 that contains data about patients and doctors from the Intensive Care sector of a hospital. We can represent such database directly in XML as follows. The Intensive Care database is represented by an intensivecare element. For each relation, we create a sub-element (doctors, patients, ...). Next, we represent each relation as an element. For example, the doctors relation is represented by a doctors element. For each tuple of the relation doctors, we create a doctor sub-element. Thus, doctors can hold an arbitrary number of doctor sub-elements. Finally, we represent each attribute (name, phone, ...) as sub-elements of doctor with the corresponding name. This database structure can be described as the XML DTD presented in Figure 2.6. The tree model of the DTD for the Intensive Care Sector is presented in Figure 2.7.

3 Building Mediated Views

Building a mediated view can be complicated and time-consuming, therefore, a lot of effort has been made recently to developing tools for automation of the mediator generation process. In our approach, the mediated view generation process consists of 3 steps:

(i) Mediated view modeling – Analyzes the user requirements and specifies the mediated view schema using a high-level data model. In this work, we use YAT’s data model to represent the mediated view schema.
Mediated view integration – Integrates the mediated view schema with the local schemas in order to identify the correspondence assertions that formally specify the relationships between the mediated view schema and the local schemas. To accomplish this, the mediated view schema and the local schemas should all be expressed in the same data model, the so-called “common” data model. Therefore, a translation of the local schemas to the YAT model is necessary and is dealt with as a separate task.

Mediated view definition – Generates the mediated view definition based on the mediated view schema and the mediator’s correspondence assertions. The mediated view definition consists of a sequence of rules or queries, which describe declaratively how queries against the mediator schema can be mapped onto queries against the underlying data sources.

To illustrate our approach, suppose that, first, a user specifies the mediated view schema med presented in Figure 3.1, which integrates information from the Emergency schema (Figure 2.2) and the Intensive Care schema (Figure 2.7).

The next step in the process of building the mediated view med is the integration of the med schema with the two local schemas as described in the following:

1) The integration process starts with the specification of the Correspondence Assertions (CA) for the root element med, which contains a set of patient elements that are semantically equivalent\(^1\) to elements in \(E_{\text{patients.patient}}\) and/or elements in \(IC_{\text{patients.patient}}\) as specified by the CA

\[
\Psi_1: med_{\text{patient}} = E_{\text{patients.patient}} \cup IC_{\text{patients.patient}},
\]

which specifies that \(med_{\text{patient}}\) contains the “outer union” of the set of elements in \(E_{\text{patients.patient}}\) and the set of elements in \(IC_{\text{patients.patient}}\). More formally, \(\Psi_1\) specifies that: (i) for each element \(p_1\) in \(E_{\text{patients.patient}}\) there is one element \(p\) in \(med_{\text{patient}}\) such that \(p \equiv p_1\) and (ii) for each element \(p_2\) in \(IC_{\text{patients.patient}}\) there is one element \(p'\) in \(med_{\text{patient}}\) such that \(p' \equiv p_2\). This is an “outer union” because, when an element \(p_1\) in \(E_{\text{patients.patient}}\) and an element \(p_2\) in \(IC_{\text{patients.patient}}\) represent

\(^1\) Two elements \(e_1\) and \(e_2\) are semantically equivalent \((e_1 = e_2)\) if \(e_1\) and \(e_2\) represent the same real world object.
the same real world objects, then there will be only one corresponding element in med.patient* which is obtained by merging $p_1$ and $p_2$.

2) Next, we have to specify the correspondence assertions for the nested elements in med.patient*.

2.1) Since an element in med.patient* can be semantically equivalent to an element in E.patients.patient*, we have to specify the correspondences of the nested elements in med.patient* with the nested elements in E.patients.patient*. These correspondences are expressed by the CAs

$$\Psi_2: E.\text{patients}.\text{patient*}.\text{name} \equiv \text{med.patient*}.\text{name},$$
$$\Psi_3: E.\text{patients}.\text{patient*}.\text{id_pat} \equiv \text{med.patient*}.\text{id},$$
$$\Psi_4: E.\text{patients}.\text{patient*}.\text{doc} \in \text{med.patient*}.\text{doctor*},$$

which specify that given an element $p$ in med.patient*, if there is an element $p_1$ in E.patients.patient*, such that $p \equiv p_1$, then: (i) $p$ contains an element name such that $p$\_name $= p_1$\_name (from $\Psi_2$); (ii) $p$ contains an element id such that $p$\_id $= p_1$\_id_pat (from $\Psi_3$), and (iii) there exist $d$ in p.doctor*, such that $d \equiv p_1$\_doc (from $\Psi_4$).

The correspondences of the nested elements in med.patient*.doctor* with the nested elements in E.doctors.doctor* are specified by the CAs

$$\Psi_5: E.\text{doctors}.\text{doctor*}.\text{name} \equiv \text{med.patient*}.\text{doctor*}.\text{name} \quad \text{(similar to } \Psi_2 \text{)},$$
$$\Psi_6: E.\text{doctors}.\text{doctor*}.\text{id_doc} \equiv \text{med.patient*}.\text{doctor*}.\text{id} \quad \text{(similar to } \Psi_3 \text{)}.$$

2.2) Since an element in med.patient* can also be semantically equivalent to an element in IC.patients.patient*, we have to specify the correspondences of the nested elements in med.patient* with the nested elements in IC.patients.patient*. These correspondences are expressed by the CAs

$$\Psi_7: IC.\text{patients}.\text{patient*}.\text{name} \equiv \text{med.patient*}.\text{name} \quad \text{(similar to } \Psi_2 \text{)},$$
$$\Psi_8: IC.\text{patients}.\text{patient*}.\text{id_pat} \equiv \text{med.patient*}.\text{id} \quad \text{(similar to } \Psi_3 \text{)},$$
$$\Psi_9: IC.\text{patients}.\text{patient*}.(\text{docpat.pat})^\dagger_\text{doc} \subset \text{med.patient*}.\text{doctor*}.$$

$\Psi_9$ specifies that given an element $p$ in med.patient*, if there is an element $p_1$ in IC.patients.patient*, such that $p \equiv p_1$, then for each $d$ in $p_1.(\text{docpat.pat})^\dagger_\text{doc}$ there exist $d'$ in p.doctor* such that $d \equiv d'$. Note that $p_1.(\text{docpat.pat})^\dagger_\text{doc}$ returns all the doctors of $p_1$.

The correspondences of the nested elements in med.patient*.doctor* with the nested elements in IC.doctors.doctor* are specified by the CAs

$$\Psi_{10}: IC.\text{doctors}.\text{doctor*}.\text{name} \equiv \text{med.patient*}.\text{doctor*}.\text{name} \quad \text{(similar to } \Psi_2 \text{)},$$
$$\Psi_{11}: IC.\text{doctors}.\text{doctor*}.\text{id_doc} \equiv \text{med.patient*}.\text{doctor*}.\text{id} \quad \text{(similar to } \Psi_3 \text{)}.$$

The final step in the process of building a mediated view consists of the generation of the mediated view definition. In our approach, the mediated view is defined based on the mediated view schema and the mediator’s correspondence assertions. In this work, we use XML-QL [7] for the definition of the mediated view. XML-QL is a query language specifically designed for XML and allows XML data to be queried, translated and integrated. XML-QL queries consist of a WHERE clause, specifying what to select, and a CONSTRUCT clause, specifying what to return. It is important to note that our approach can be used to define mediated view definitions in any other languages.
Figure 3.2 shows the view definition for med. As we can see, the view definition consists of two WHERE-CONSTRUCT main blocks. The first WHERE-CONSTRUCT block extracts information about patient elements from the local source “emergency.xml” and the second block extracts information about patient elements from the local source “intensivecare.xml”. When the same patient occurs in both sources, the two elements are combined. As we can observe, these two blocks correctly implement the outer join as specified by the correspondence assertion $\Psi_1$.

In the first block, the information that must be selected in the WHERE clause and the information that must be returned in the CONSTRUCT clause is defined based on the correspondence assertions that specify the correspondences of the nested elements in med.patient with the nested elements in E.patients.patient*. For example, lines 5 and 18 are defined based on the CA $\Psi_2$ and lines 6 and 19 are defined based on the CA $\Psi_3$. The correspondences of the nested elements in med.patient*.doctor* with the nested elements in E.doctors.doctor* are used to generate the lines 12 and 19 (from $\Psi_6$) and the lines 13 and 21 (from $\Psi_5$). Analogously, the WHERE and CONSTRUCT clauses of the second block are defined based on the CAs that specify the correspondences of the nested elements in med.patient with the nested elements in IC.patients.patient*.

4 Conclusions

In this work, we propose the use of correspondence assertions for specifying the semantics of XML-based mediators. The advantages of having a high-level specification of the mediator are the language-independence and the facility that it provides for one to have a better understanding of the semantics associated with the mediator.

We have shown how the correspondence assertions can be used for helping to generate the mediated view definition. With our formalism we are able to prove, in a formal way, that a given mediated view definition correctly implements the mediators’ specification. As shown by the example used in the paper, our formalism can also handle the problem of semantic heterogeneity.

The semantic specification of the mediator can also be used to automate other aspects of data integration, such as, for example, the Maintenance of the mediated view definition. In dynamic environments, such as the Web, individual information sources may change not only their data but also their capabilities. As a result, whenever a local schema changes, the mediated view definition needs to be updated to reflect the modifications. We intend to use the mediators’ Correspondence Assertions to help in re-writing a mediator view definition in response to local schemas’ changes.


[4] EVE

[5] XML


[12] Linguagens de esquemas

[13] Linguagem declarativa

[14] Linguagem declarativa

[15] Linguagem declarativa

[16] problema de integração de bd

[17] problema de integração de bd

[18] problema de integração de bd