An Ontology-Based Approach for Semantic Conflict Resolution in Database Integration
Qiango Liu\textsuperscript{1,2}, Tao Huang\textsuperscript{1}, Shao-Hua Liu\textsuperscript{1}, Hua Zhong\textsuperscript{1,2}
\textsuperscript{1}Institute of Software, Chinese Academy of Sciences
\textsuperscript{2}Graduate School of the Chinese Academy of Sciences
Beijing 100080, P.R. China

Corresponding author E-mail: liuq@otcaix.iscas.ac.cn

Abstract
An important task in database integration is to resolve data conflicts, both on schema-level and on semantic-level. Especially difficult the latter is. Some existing approaches to semantic interoperability have been criticized for their lack of semantic richness and domain generality. To overcome these limitations, this paper introduces a novel approach to detect and resolve various semantic conflicts in heterogeneous databases. Our approach includes two important parts: a semantic conflict representation model based on our classification framework of semantic conflicts, and a method for detecting and resolving semantic conflicts based on this model. The system has been developed, experimental evaluations on which indicate that this approach can resolve the semantic conflicts effectively, without learning about domain and integration patterns.

Introduction

With the constantly increasing reliance on database systems to store and process data, comes the additional problem of ensuring interoperability between these systems. The fundamental problem in interoperability is that of identifying objects in different databases that are semantically related, and then resolving the data conflicts both on schema-level and on semantic-level. Another important problem is how to construct an integrated view over many heterogeneous data sources\cite{1}. According to Alon Halevy\cite{2}, achieving semantic interoperability has been the first challenge facing information integration. In this paper, we are interested in the former question, and focus on the solutions to relational databases(RDBs).

Previous researches in interoperability can resolve data conflicts on schema-level effectively by schema translation and intermediary mechanisms. For example, the federated schema approach attempts to construct a global schema and establish mappings between the global schema and the participating local schemas. However, because of the lack of semantic richness and flexibility, it provides little or no support for semantic conflict resolution.

It is becoming increasingly clear that the emerging technology concerned with the development and application of ontologies will play a central role in overcoming the problems of semantic heterogeneity. An ontology is a formal, explicit specification of a shared conceptualization\cite{3}. Compared with database schemas, ontologies allow more complete and more precise domain models\cite{4}. Recently, some ontology-based approaches to semantic integration have been developed. They can be classified into two categories: domain-ontology-based approach and ontology-mapping-based approach. The former approach attempts to construct ontologies for specific domains, which express the shared knowledge of concepts, attributes, and rules. Local schemas participated in integration are built using the concepts defined in domain ontologies, so that mappings between #
semantic related information sources can be constructed. Even though such an approach may be theoretically valid, the application of it is practically infeasible due to the complexities of the domain knowledge. Hence, it is lack of domain generality. The latter approach attempts to transform the problem of schema matching to ontology mapping after the explicit expression of local schemas. Ontology mappings can be created semi-automatically using heuristic-based and machine-learning algorithms, which utilize lexical and structural characteristics to find correspondences. Although many techniques have produced good results, automatic mapping between ontologies is still beyond our grasp at the moment.

The design of an ontology-based semantically interoperable system environment that manages various semantic conflicts is a daunting task. Semantic conflicts should be expressed explicitly and resolved effectively. At the same time, the approach should be flexible enough to keep independent of domains and integration patterns. The work described in this paper provides a solution to this challenge. We introduce a semantic conflict representation model (SCM) based on our classification framework of semantic conflicts, which includes two important parts, an ontology (SCO) describing the classification of semantic conflicts and an abstract semantic model (ESM) extending RDB schema. They are described by RDF graph[5] and OWL language[6] respectively. Also we propose the method based on this model to detect and resolve semantic conflicts, which is accomplished by some semantic mediators. The system has been developed, experimental evaluations on which indicate that this approach can resolve the semantic conflicts effectively.

The rest of this paper is organized as follows. Section 2 discusses the relevant researches in ontology-based semantic interoperability. Section 3 and section 4 present our semantic conflict representation model. In section 5, we introduce the methodology for the detection and resolution of semantic conflicts. Section 6 discusses a case study to give a proof of the proposed framework and algorithms. Section 7 shows the system implementation and the results of experimental evaluations. Section 8 summarizes this work and discusses some possible extensions.

Related Work

The domain-ontology-based semantic integration researches are mostly domain specific, such as Gene Ontology[7] and Unified Medical Language System[8] in the life science community. In these systems, ontologies can be used to define a common controlled vocabulary, similarly to applying XML to industry specifications. These ontologies often have thousands of concepts, and need to be developed iteratively. In practical applications, we don’t have this “luxury” using domain ontologies, because there are no absolute semantics that are valid for all potential users.

On the contrary, ontology-mapping-based approaches are domain independent. Some representative works include InfoSleuth[9][10], OBSERVER[11] and ONION[12]. InfoSleuth is an agent-based system, which supports construction of complex ontologies from smaller component ontologies so that tools tailored for one component ontology can be used in many application domains. Mappings are explicitly specified among these ontologies as relationships between terms in one ontology and related terms in other ontologies. OBSERVER uses a component-based
approach to ontology mapping. It provides brokering capabilities across domain ontologies to enhance distributed ontology querying, thus avoiding the need to have a global schema. ONION is an architecture based on a sound formalism to support a scalable framework for ontology integration that uses a graph-oriented model for the representation of the ontologies. Articulation rules are established between ontologies to enable knowledge interoperability. These systems substitute ontology mapping for schema matching by virtue of the explicit expression ability of ontologies. Types of mappings include class mappings, property mappings, axioms/rules mappings, and conditional mappings, the number of which that is supported determines the number of semantic conflicts that can be resolved. Although heuristic-based and machine-learning algorithms can help automate this process, they are far from being applied. Our SCM model differs from existing research on ontology-based semantic integration in the following ways.

1. Domain ontologies often have large size, and are difficult to maintain due to the complexity of domain knowledge. While the SCO ontology we proposed has the moderate scale. During iterative development, it tends to be stabilized.

2. Ontologies in ontology-mapping-based approaches are equivalent to local schemas, and ontologies in domain-ontology-based approaches express the shared domain knowledge. While the SCO ontology has the characteristics of semantic richness and domain generality.

3. The domain-ontology-based approach can be viewed as a global view, and the ontology-mapping-based approach is similar to a local view. While the SCM model can be used in both integration patterns.

The most related research to our work is SCROL[13]. It is developed to encode extensible knowledge on commonly found semantic conflicts, which provides a systematic way of detecting and resolving various semantic conflicts found in heterogeneous databases. In addition, all schema components captured by a common semantic data model, called USM, are mapped to this ontology. On the basis of this model, the software environment called CREAM[14] is implemented. It can be used to access heterogeneous databases via a graphical user interface.

Compared with SCROL, our approach has some distinct characteristics as follows.

1. SCROL is designed mainly for the resolution of conflicts among heterogeneous geographic data sources, so the classification framework of conflicts is based on geographic database systems. However, our work concerns relational databases.

2. SCROL arranges the ontology hierarchically, like a kind of taxonomy. Therefore, it can’t describe the horizontal relationships among the “tree nodes” and the conversion relationships among conflict instances. While our SCO ontology is defined by RDF graphs, which can easily express these relationships and be processed by many ontology editors.

3. In order to resolve semantic conflicts, SCROL stores mapping information between the classification ontology and underlying schema components. Consequently different applications need different conflict classification ontologies. However, our SCO ontology does not need to store this mapping knowledge. Concepts defined in the SCO ontology are used when we extend RDB schemas, through which the correspondences are built implicitly.

Among the researches of traditional database integration, there are also many works related to resolving semantic conflicts, such as X-Specs[15] and COIN[16].
They often use lexical systems or global dictionaries to define the global schema, and use some extensible metadata models (such as deductive object-oriented data model) to express data semantics. But in our opinion, they are primarily explorations in the approach of ontology-based semantic integration.

The Ontology for Semantic Conflict Classification in Database Integration

In this section, we firstly present our classification framework of semantic conflicts, which is the foundation of our research. Then this classification framework is defined as an ontology, which is represented as a RDF graph. So it can be clearly comprehended and easily processed.

The Classification Framework of Semantic Conflicts

According to Won Kim[17], since a database is defined by its schema and data, one can classify conflicts at the highest level as either schema or data conflicts. Schema conflicts result from the use of different schema definitions in different databases. While data conflicts are due to inconsistent data in the absence of schema conflicts.

Here we develop a complete framework for enumerating and classifying those conflict types which can't be resolved in the RDB model. So it is only a subset of data conflicts that are classified in [16], [17], and [18]. The RDB schema can deal with "weak semantics", such as data scale and data length. They can't represent the implied data semantics. Therefore, we can't detect these "deep level" conflicts, to say nothing of resolving them according to some predefined transformation rules.

The classification framework we proposed is as follows.

1. **Data Type Conflicts.** These conflicts occur when two attributes have different data types. For example, "Student.id" can be defined as an 8 character string or an 8 digit integer. Some kinds of data type conflicts can be resolved by database itself, while most of them rely on the explicit expression of transformation rules.

2. **Data Format Conflicts.** These conflicts occur when two attributes have identical data type but different data formats. For instance, the "date" type can be represented as "DD/MM/YY" and "YYYY/MM/DD" two different formats. There is a one-to-one mapping relationship between them.

3. **Data Unit Conflicts.** These conflicts occur when two attributes are represented using different units or measures. For example, the "length unit" might be "meter", "kilometer", and "feet" etc. There are one-to-one mapping relationships between them, and these relationships are transitive.

4. **Data Precision Conflicts.** These conflicts occur when two attributes have identical data type but different data precisions. The mapping relationships between them might be one-to-one or many-to-one. For example, the "area" field can be represented using 10^5 and 10^7 two different scales. It is a one-to-one mapping. Considering the representation of "exam grade", one can use...
digits from 0 to 100 or characters from “A” to “E”. There exists a many-to-one mapping between them.

5. Default Value Conflicts. This type of conflict depends on the definition of the domain of concerned attributes. The default value of an attribute is that value defined in the absence of more information about the real world. For instance, the default value for “age” of an adult might be defined as 18 in one database or 21 in another.

6. Attribute Integrity Constraint Conflicts. These conflicts occur when two attributes are restricted by different constraints. Some of them might be consistent with each other. For example, the constraint for “age” of an adult might be defined as “≥16” or “≥18”. Because there is intersection between them, it can be resolved by a filter. While other conflicts may not be resolved.

This classification framework is based on previous researches and our analysis of data from practical database integration applications. It is the foundation of the following ontology definition.

The Definition of the SCO Ontology

Firstly we define the SCO ontology for the classification framework of semantic conflicts introduced above. Then we use RDF graph to represent it. This is mostly because that RDF is intended for situations in which the information needs to be processed by applications, rather than being only displayed to people, and there exist many RDF parsers and processing tools.

The formal definition of the SCO ontology is as follows.

Definition 1: An ontology for the classification of semantic conflicts is a tuple \((h, cv, CI, RC)\). It is a set of many concepts, instances and their interrelationships, where \(h, cv, CI\) and \(RC\) are defined as below. The RDF graph of the SCO ontology is depicted in Fig. 1.

![RDF graph of the SCO ontology](#)
Definition 2: There is exactly one element in a graph. It can be considered as the home vertex in RDF specification, a blank node can be assigned an identifier prefixed with "_:". Therefore, we use "_:H" to represent in Fig. 1.

Definition 3: CV is a distinct set of virtual concepts. Each element of CV is called a virtual concept. It has subconcepts and is a superconcept, but is not allowed to have instances. In RDF graph depicted in Fig. 1, cv represents an element of CV. Some properties of virtual concepts are defined as follows:

1. Name is a term that represents a virtual concept.
2. SubClassOf is a mandatory property of all concepts except h. It contains its superconcept. Concepts can have exactly one superconcept, that is to say, multiple inheritance is not allowed.
3. SubClass is the property contains a list of subconcepts belonging to it.
4. Map Func is a property that represents the mapping function's name. This function is used to transform between objects of one and those of another's subsequent sibling. It is only a character string, not a function or a class. The map Func value of the last element among ci's siblings is "null".

According to these definitions, only elements in CI represent the concepts of semantic conflicts. So we discuss CI further. In the classification framework of semantic conflicts described in section 3.1, we noted that mapping relationships between conflict instances are various. What makes this problem even more complex is that conflict instances are peer to peer, which have no sequences. For the sake of simplicity and clarity, we put some assumptions on the elements of CI.

1. Many-to-many mappings can be converted to one-to-one mappings. Suppose we can achieve this goal by virtue of "concept generalization". For example, "exam grade" can be represented by digits from 0 to 100 or characters from "A" to "E". It is a many-to-one mapping between them when we consider 101 digits and 5 characters as conflict instances. From a different point of view, if only two conflict instances of "centesimal grade" and "five ranks system" are taken into account, one-to-one mapping can be built to shield users from understanding complex relationships.

2. Elements in CI are ordertly arranged. Objects of one element can only be converted to the objects of its subsequent element. Under this assumption, objects of two arbitrary elements should be converted using a list of mapping functions. For instance, f1, f2, and f3 are the mapping functions of object a, object b, and object c, respectively. One can convert object a to object b by executing f1(a). Similarly, the compound function f2(f1(a)) denotes the conversion from object a to object c.

3. Elements in CI can be arranged to ensure the "information lossless" during conversion. For example, the date format can be represented as follows: 1#
“YYYY-MM-DD”, 2. “MM-DD-YY”, 3. “DD-MM-YYYY”. According to assumption 2, the conversion processes are executed orderly. If we arrange them as 123, the information about “year” in 3 will lose.

We illustrate the difference between CV and Cl with examples. “Length” is an element in CV, it is a virtual concept. Some of its subconcepts are “meter”, “kilometer”, and “feet”, which directly express the length unit. So they are instanceable concepts. Conversions can only occur between objects of these elements.

In the RDF graph, we use RDF collections to denote the sequence relationships between elements of Cl. The instance of Cl is the mapping function’s name. For example, “meterToKilo”, “kiloToFeet”, and “null” are the mapping function names of element “meter”, “kilometer”, and “feet” respectively, which can be seen from Fig. 2. To convert the objects of “meter” to those of “kilometer”, the “meterToKilo” function should be executed.

Definition 5: RC refers to the sibling relationships on CV and Cl. The relationship among elements of CV is disjoint. It means: elements of CV which inherit from the identical superconcept have different data semantics. However, elements of Cl have peer relationships. They have similar data semantics, so that equivalent conversion can be performed among them. For instance, “unit” and “precision” are elements of CV, “Centigrade” and “Fahrenheit” are elements of Cl. They have different sibling relationships.

Fig. 2 depicts a fragment of the SCO ontology. We take “sco” as the QName of those concepts in the classification framework. Another two well known QNames “rdf” and “rdfs” are also used. In the following section, concepts defined in the SCO ontology (e.g., “unit” and “length”) are used to define the abstract model which extends RDB schema semantically. In this way, mapping relationships between the SCO ontology and RDB data sources that participate in integration can be built impliedly.

Fig. 2 A fragment of the SCO ontology

© IBIS+ Issue 2(2), 2006
Extending RDB Schema Semantically Using the SCO Ontology

We aim to use ontologies, which express data semantics explicitly, to resolve those conflicts that the RDB model can’t handle. In section 3, the ontology for the classification of semantic conflicts is introduced. But where are those conflict concepts in the SCO ontology from? We don’t expect to find them in RDB schemas, because they can only present “weak semantics”. In order to resolve the semantic conflicts in RDBs, we must extend RDB schema semantically.

According to Tim Berners-Lee[19], RDF model is great as a basis for ER-modeling. It is very directly connected with the RDB schema and can be easily mapped onto. The OWL language is designed to facilitate greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. What’s more, there exist many editors and process tools for both of them. Thus, we propose the abstract model ESM which extends RDB schema with OWL language.

Due to space limitation, we only present the most important parts of ESM in Fig.3. In the OWL ontology, we describe the RDB schema abstractly, which includes classes like Table, Column, and Primary Key. Additionally we specify possible relationships among them by OWL:ObjectProperty classes. Columns of table are defined with OWL:DatatypeProperty classes, where all the properties required are specified. Note that the SCO ontology is imported (see row 3). By using concepts defined in the SCO ontology, we can easily extend properties of Column explicitly (e.g., Properties of "unit" and "precision" are defined in row 13 and row 14). Compared with SCROL, we don’t need to store the mapping knowledge in the SCO ontology, which ensures generality of our approach.

Fig.3 The ESM abstract model described with OWL language
Detecting and Resolving Semantic Conflicts Based on SCM

Our conflict detection and resolution method is based on the concept of "mediator" proposed by [20]. Fig. 4 illustrates the software environment, in which the data, control, and resources are inherently distributed.

The data components in Fig. 4 are as follows.

1. RDB Data Source, which denotes the resource that participates in integration. According to the data flow direction, it can be distinguished as "original data source" and "destined data source".

2. esm, which is a concrete ontology defined following the ESM abstract model. It is built by an equivalent transformation of the RDB schema and then an extension of some possible conflict attributes. An esm only describes the schema of a RDB data source, in which no data instances are stored.

3. SCO, which is the ontology defined in section 3.

4. Conversion Knowledge, which denotes the concrete knowledge for the conversion of conflict concepts. In the definition 4 of the SCO ontology, the mapFunc property refers to this knowledge.

5. Reference Knowledge, which denotes dynamic knowledge compared to the conversion knowledge. Static knowledge is based on current human knowledge or strong belief that has been true for a long period of time. Examples are "length conversion ratio" and "time range conversion ratio". Dynamic knowledge, on the other hand, changes regularly. An example is the "currency exchange rate".

Note that there are mapping relationships between esms and the SCO ontology, because esms use some conflict concepts in the SCO ontology when we extend RDB data sources. There are also mapping relationships between the SCO ontology and the conversion Knowledge, which are built by the mapFunc property in the SCO ontology.

On the basis of these semantic mediators, the semantic integration process can be accomplished in the following six steps.

1. Design the SCO ontology manually or using ontology editors.
2. Build conversion knowledge and reference knowledge;
3. Use modeling component to extend RDB data sources to esms based on the ESM abstract model and the SCO ontology;
4. According to the possible semantic conflict concepts that RDB data sources brought, extend the SCO ontology, the conversion knowledge, and the reference knowledge incrementally;
5. Use the detector component to detect semantic conflicts when integration process occurs;
6. Use the resolver component to resolve the semantic conflicts detected, and load transformed data to destined data sources.

Because the detection and resolution of semantic conflicts are two complex processes, we describe the algorithms in Fig. 5 and Fig. 6.

Algorithm DetectConflicts(IntegrateStatement, ConflictSet)
{
    Input: IntegrateStatement // e.g. “Select ColumnA From TableA, Insert into TableB(ColumnB) values(ValueOfColumnA)”
    // We call ColumnA and ColumnB a ColumnPair that has the mapping relationship
    // ColumnA is the column of original data source ColumnB is the column of destined data source
    Output: ConflictSet // a set of conflict objects
    ColumnPairs = Parse(IntegrateStatement); // parse the integration statement to get ColumnPairs
    For (each pair<ColumnA, ColumnB> of ColumnPairs)
    {
        OrigianlDataAttributeSet = GetDataAttributes(ColumnA); //get data attributes of ColumnA from esm1
        DestinedDataAttributeSet = GetDataAttributes(ColumnB); //get data attributes of ColumnB from esm2
        //data attributes includes type, length, unit, and precision etc.
        For (each OriginalDataAttribute in OriginalDataAttributeSet){ //for each data attribute of ColumnA
            For (each DestinedDataAttribute in DestinedDataAttributeSet){ //compare with the data attribute of ColumnB
                If (OriginalDataAttribute and DestinedDataAttribute have same name but different ranges){
                    co = New ConflictObject(ColumnA, ColumnB, OriginalDataAttribute Name, OriginDataAttribute.Range, DestinedDataAttribute.Range);
                    //create a new ConflictObject, each conflict object contains column of original data source, column of destined data source, data attribute name and their different ranges
                    Add(ConflictSet, co); //add the new ConflictObject to ConflictSet
                    break; //exit inner loop for next data attribute of original data source
                }
            }
        }
    }
}

Fig. 5 The algorithm for conflicts detection

Algorithm ResolveConflicts(ConflictSet)
{
    Input: ConflictSet
    For (each ConflictObject in ConflictSet){
        MapFunctionSet = GetMapFunctions(ConflictObject.DataAttributeName, ConflictObject.OriginalDataAttributeRange, ConflictObject.DestinedDataAttributeRange);
        //according to the name and different ranges of the data attribute, find MapFunctions in the SCO ontology
        ColumnValues = CaptureDataSets(ConflictObject.OriginalColumnName); //capture data from original data source
        For (each MapFunction in MapFunctionSet){ //execute a series of map functions to transform data
            Update(ColumnValues, MapFunction);
        }
        Load(ConflictObject.DestinedColumnOriginalName, ColumnValues); //load transformed data to destined data source
    }
}

Fig. 6 The algorithm for conflicts resolution
Case Study

To clarify how to detect and resolve semantic conflicts based on our framework and algorithms, we discussed a case study in this section.

The case study comprises an original data source and a destined data source. We aim to add data from the former to the latter. The main parts of two schemas and some example data are illustrated in Fig.7.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelID</td>
<td>integer</td>
<td>ChannelID</td>
<td>string</td>
</tr>
<tr>
<td>ShowTime</td>
<td>datetime</td>
<td>PlayTime</td>
<td>date</td>
</tr>
<tr>
<td>Duration</td>
<td>integer</td>
<td>ContinueTime</td>
<td>integer</td>
</tr>
</tbody>
</table>

**Program – Original Schema**

**TVShow – Destined Schema**

<table>
<thead>
<tr>
<th>ChannelID</th>
<th>ShowTime</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>322</td>
<td>03:05:01_08:05:00</td>
<td>30</td>
</tr>
<tr>
<td>323</td>
<td>03:05:01_08:35:00</td>
<td>45</td>
</tr>
<tr>
<td>324</td>
<td>03:05:01_08:20:00</td>
<td>78</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ChannelID</th>
<th>PlayTime</th>
<th>ContinueTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;196&quot;</td>
<td>2003/04/25 03:10:00</td>
<td>900</td>
</tr>
<tr>
<td>&quot;197&quot;</td>
<td>2003/04/25 03:15:00</td>
<td>2700</td>
</tr>
<tr>
<td>&quot;198&quot;</td>
<td>2003/04/25 04:00:00</td>
<td>480</td>
</tr>
</tbody>
</table>

*Fig.7 The original source and destined source*

From the schemas of two data sources, we can easily find two type conflicts: the type of “ChannelID” in “Program” is integer, while the type of same field in “TVShow” is string; “ShowTime” in “Program” and “PlayTime” in “TVShow” also have different date types. But these kinds of conflicts can be resolved by DBMS itself. The semantic conflicts we intend to resolve are format conflicts of the second field and unit conflicts of the third field. As can be seen from the example data, the different date formats are “YY/MM/DD HH:MM:SS” and “YYYY/MM/DD HH:MM:SS”, while the different unit of field length are “minute” and “second”. To express the semantic conflicts and their transformation relationships explicitly, we extend the SCO ontology and Conversion Knowledge as Fig.8.
We also extend the two schemas to esms according to the SCO ontology, so that the detection of semantic conflicts can be done automatically. Two esms' fragments are shown in Fig.9, among which the boldfaced parts emphasize the semantic conflicts involved.
When the algorithms of conflicts detection and resolution are performed, we can get the following results as Fig.10. #

ConflictSet = 
{ (ShowTime, PlayTime, format, colonDiv, solidusDiv),
  (Duration, ContinueTime, unit, minute, second) }

<table>
<thead>
<tr>
<th>ChannelID</th>
<th>ShowTime</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>322</td>
<td>03:05:01_08:45:00</td>
<td>30</td>
</tr>
<tr>
<td>323</td>
<td>03:05:01_08:45:00</td>
<td>45</td>
</tr>
<tr>
<td>324</td>
<td>03:05:01_09:20:00</td>
<td>78</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ChannelID</th>
<th>PlayTime</th>
<th>Continue Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;196&quot;</td>
<td>2003/04/25 03:00:00</td>
<td>900</td>
</tr>
<tr>
<td>&quot;197&quot;</td>
<td>2003/04/25 03:15:00</td>
<td>2700</td>
</tr>
<tr>
<td>&quot;198&quot;</td>
<td>2003/04/25 04:00:00</td>
<td>480</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>&quot;322&quot;</td>
<td>2003/05/01 08:05:00</td>
<td>1800</td>
</tr>
<tr>
<td>&quot;323&quot;</td>
<td>2003/05/01 08:30:00</td>
<td>2700</td>
</tr>
<tr>
<td>&quot;324&quot;</td>
<td>2003/05/01 09:20:00</td>
<td>4680</td>
</tr>
</tbody>
</table>

Original Data

Destined Data

© IBIS-Issue(2)(2), 2006
Implementation and Evaluation#

System Implementation#

We have implemented this software environment in OnceDI[21], which is a data integration middleware we have developed in last several years. We use Protege[22] to edit the SCO ontology and esms, which is a free, open source ontology editor, and its ontologies can be exported into a variety of formats. The conflicts detection process can be executed immediately after the definition of data integration, while the conflicts resolution process are executed before loading data to destined data sources. In both processes, the algorithms of searching in esm and the SCO ontology affect directly the system performance.

There exist a set of open APIs for the secondary development by users. Additionally, we have developed a series of managers that can be directly used to accomplish any applications. A typical manager used to build mapping relationships between original and destined data sources is shown in Fig.11.

![Field Mapping Editor](image)

Fig.11: The field mapping dialog of the system#

Experimental Evaluations#

To evaluate this approach, we tested it with databases in a practical application. Additionally, we tested the CREAM platform with same cases as a comparison to the similar system. Note that although OnceDI is designed to transform data between heterogeneous databases, while CREAM aims to provide uniform and integrated access to multiple heterogeneous databases, both of them need the ability of resolving semantic conflicts.

To examine how well it can handle different types of semantic conflicts and to gain from the test results insights to guide our future research, we selected data from the ChinaEPG system, which is a data exchange system running in CCTV. In this system, a data center is built, which is responsible for data exchanges between program providers (e.g. ShanDong TV) and EPG (electronic program guide) customers. We selected data exchange scenario among two program providers and two EPG customers through the data center, so that data exchanges occurred six times.
times between these five nodes. Because some types of semantic conflicts rarely happen (e.g., data precision conflicts), we added some cases to contain all the different types of semantic conflicts described in section 3.

Summary of semantic conflicts resulting from the test cases are shown in Table 1.

<table>
<thead>
<tr>
<th>Conflict Type</th>
<th>Number of Conflicts</th>
<th>Number of Conflicts Found</th>
<th>Number of Conflicts Resolved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Type Conflicts</td>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>Data Format Conflicts</td>
<td>19</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Data Unit Conflicts</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Data Precision Conflicts</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Default Value Conflicts</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Attribute Integrity</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Constraint Conflicts</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Summary of semantic conflicts resulting from the test cases

From this test, we can conclude that:

1. After the SCoI ontology is built and the RDB data sources are extended to esms, all kinds of semantic conflicts can be automatically detected and resolved by the system without further human intervention. Building the SCoI ontology is a one-time task that should be performed by human experts.

2. The ability to detect and resolve semantic conflicts depends on the completeness of the SCoI ontology and esms. When we use same knowledge to build ontologies in OnceDI and CREAM, their abilities are equivalent. But the scale of ontologies is different, the size of our SCoI ontology is about 4/5 that of SCROLL. The reason is that SCROLL stores the mapping relationships with data sources additionally.

3. CREAM can't resolve some of the data precision conflicts, because SCROLL doesn’t support many-to-many mapping relationships between conflict concepts. Furthermore, CREAM can’t detect and resolve the default value conflicts. While our approach can overcome these shortcomings.

4. Some of the data format conflicts can’t be detected, because the SCoI ontology isn’t complete. Additionally, due to some schema conflicts exist, OnceDI can’t resolve some of the attribute integrity constraint conflicts, while CREAM has this ability.

Conclusion and Future Research

In this paper, we present a novel approach to detect and resolve semantic conflicts in database integration. This approach is based on a shared ontology for the classification of semantic conflicts and an abstract model extending RDB schema that can be used to express data semantics explicitly. Additionally, the software environment composed of semantic mediators based on this framework is implemented in our middleware system OnceDI. Results of our experimental evaluation have shown that this approach can resolve much of the semantic conflicts effectively, and keep independent of domains and integration patterns.

© IBIS+: Issue 11(2), 2006
There are a number of interesting extensions for our future research. One area is to improve automation of the whole process. Currently the SCO ontology is built manually. Developing algorithms to automatically generate the primary ontology by virtual reverse engineering approach may be a solution. Another problem we have to solve is that errors occur when the SCO ontology isn't complete enough to resolve all kinds of conflicts. This needs friendly wizards to guide users solving this problem smoothly. The third area is to integrate approaches for the resolution of semantic conflicts and schema conflicts effectively, so that all kinds of conflicts can be resolved in one solution.

Acknowledgment

This work is supported by the National Natural Science Foundation of China under Grant No. 60573126, the National Grand Fundamental Research 973 Program of China under Grant No. 2002CB312005.

References


IBIS: Interoperability in Business Information Systems

#

Bibliography

Qiang Liu: received his M.S. degree from Henan University in 2003. Currently he is a Ph.D. candidate in the Institute of Software, Chinese Academy of Sciences. His current research interests include distributed computing, database and knowledge-based systems and software engineering technology.

Tao Huang: received his M.S. and Ph.D. degrees in computer science, from the University of Science and Technology of China in 1991 and #1994 respectively. Currently he is a professor and Ph.D. supervisor in the Institute of Software, Chinese Academy of Sciences. He has published more than 40 research papers and two co-authoring books, and undertaken many research projects. His research interests include software engineering, distributed computing and middleware.

ShaoHua Liu: received his Ph.D. degree from the Institute of Software, Chinese Academy of Sciences in 2005. Currently he is an assistant professor. His research interests include distributed computing, workflow system and service collaboration.

HuaZhong: received his M.S. and Ph.D. degrees from the Institute of Software, Chinese Academy of Sciences in 1996 and 1999 respectively. Currently he is a professor and the director of Software Engineering Center. His research interests include distributed computing, information integration and software engineering technology.