AN ARCHITECTURE FOR MANAGING SCHEMA EVOLUTION IN A FEDERATED SPATIAL DATABASE SYSTEM

Xiaoying Wu 1; Jianhong (Cecilia) Xia 1; Geoff West 1; Lesley Arnold 2; Bert Veenendaal 1

1. Department of Spatial Sciences, Curtin University
Kent St, Bentley, WA, 6102, Australia
xiaoying.wu@student.curtin.edu.au;{c.xia, g.west, b.veenendaal}@curtin.edu.au
2. Landgate, Midland Square, Midland WA 6056, Australia
lesley.arnold@landgate.wa.gov.au

ABSTRACT

A Federated Spatial Database System (FSDBS) is the integration of multiple spatial data sources and the realisation of effective spatial data sharing. FSDBS environments are becoming increasingly popular as more and more spatial and non-spatial datasets are integrated, especially those across a number of independent organisations. However, in a FSDBS environment, database schemas are subject to change and the management of these changes is complex and inefficient. This is because schema changes in one local database will affect or invalidate not only applications built against this local schema, but also applications built against the global schema.

In this research, a Schema Element Dependency Meta-model and a set of Schema Change Templates which incorporates view generation, view rewriting and query rewriting have been developed as the solution in managing schema evolution in a FSDBS. This paper focuses on the architecture of Automatic Schema Evolution that includes a schema element dependency tool, a schema mapping tool and a view rewriting tool. These tools are embedded with the developed methods will assist in effective management of schema evolution in a FSDBS and engendering collaboration and interoperability between data custodians and application developers and users.

1. INTRODUCTION

This research proposed methodologies to deal with schema changes that can occur in a Federated Spatial Database environment in which queries are built on one or more databases that may not be under the control of the federated database administrator. A spatial database system is distinguished from a non-spatial database system through the inclusion of [1]:

- spatial data objects such as points, lines and polygons;
- methods to process spatial data such as spatial indexes, extended spatial query languages and effective algorithms for spatial operations;
as well as the normal data types and methods for non-spatial data. A Spatial Database Management System (SDBMS) manages and manipulates a spatial database [2].

In a traditional database, a view is defined as a stored query, usually as SQL statements. In many cases, a spatial view extends the traditional view by adding a spatial field. Such spatial views are created in the same way as traditional non-spatial views and stored as spatial queries that contain the spatial attribute and spatial operations such as spatial selection and spatial joins.

A Federated Spatial Database System (FSDBS) integrates various geographically distributed spatial databases and sources and provides a unified data access mechanism in order to facilitate spatial data sharing [1]. This has been achieved through advancements in networks and communications, distributed computing technologies, and conformance with standards and policies. FSDBSs have been the subject of active research for dealing with database heterogeneity [3] [4] and are seen as an important part of any proposed Spatial Data Infrastructure (SDI).

A traditional database can be described by schemas that define relations, attributes etc. A spatial database schema is an extension of the traditional schema to include spatial descriptions and behaviours required for a spatial database [1]. Spatial database schemas, like non-spatial schemas, are subject to change or evolution due to changes in perception of reality and application requirements.

Schema changes often result in applications built against the schemas being affected or even invalidated. In a FSDBS, the federated schema is dependent on local schemas so when a local schema changes, applications built on local schemas and federated schemas and applications built on the federated schemas will be affected. Managing schema evolution overcomes the mismatch between applications and the evolved schemas, and maintains consistency of the correspondences between schemas after schema changes.

Currently schema management is performed manually and there is a need to develop methods that can be used to automatically or semi-automatically deal with schema evolution. The Automatic Schema Evolution (ASE) architecture proposed in this paper defines the components and functionalities of each component. With the ASE, schema evolution in a FSDBS can be managed in a semi-automatic and, in the future, in an automatic manner.

2. QUERY AND VIEW REWRITING

Query rewriting is the main method used to deal with schema evolution. It relies on schema mapping that specifies the correspondences between schemas [5]. In a FSDBS, a query imposed on the global schema is typically decomposed into a number of sub-queries imposed on local schemas. The decomposition here depends on mapping information typically stored as metadata.

In a FSDBS, schema mapping consists of (i) mapping between local database schemas and the federated global schema, and (ii) mapping between different schema versions of each system.

Query rewriting is the process of converting query statements into different expressions while still keeping the logical structure of the query [6]. Equivalent query rewriting occurs when the result of a rewritten query is equivalent to the results of the original query. Query rewriting has been widely studied for different applications such as query optimisation, data integration, data exchange and schema evolution [7]. For example, rewriting queries with materialised views has attracted attention for performance improvement of a database system. Different query rewriting algorithms have been
developed to rewrite queries with views [8] [9] [10]. Rewriting queries against one schema into equivalent ones against another schema has been adopted as a solution to schema integration, schema transformation and schema evolution[6] [11] [12]. There are a number of different approaches that have been proposed for representing schema mapping: Global-As-View (GAV), Local-As-View (LAV) and Global-Local-As-View (GLAV) [13] [14]. GAV has been adopted in this research due to its simplicity for query rewriting and the fact that the combination of two GAV mappings is still a GAV mapping. Therefore, schema mapping in this research is restricted to this form of GAV mapping that consists of unfolding and substitution [15]. Unfolding is when a query defined as a combination of sub-queries is rewritten as a combination of the components of the sub-queries i.e. a form of factorisation that allows manipulation of the components to produce new queries. Substitution is where a new query is generated to replace an old query. Queries expressed against the global schema can be rewritten as equivalent ones against the source schemas. Similarly, queries expressed against old schemas can be rewritten against the new schemas.

3. ARCHITECTURE OF AUTOMATIC SCHEMA EVOLUTION

Figure 1 illustrates the architecture of the ASE. There are five main components included: spatial data repositories, a metadata repository, a view rewriting tool, a schema mapping tool and a schema element dependency tool.
3.1. Spatial Data Repositories
The Spatial Data Repositories include local spatial databases and the federated spatial database. Local spatial databases contain base tables that store spatial and non-spatial data and views. The federated spatial database consists of tables and views as a virtual database built on the distributed local databases. That is the federated spatial database doesn’t contain any data as this is stored in local spatial databases. Queries on the federated database, including new queries, are indicated as “legacy queries” in Figure 1.

3.2. Metadata Repository
Metadata is data about data [16]. In this research, metadata is treated as the first class for schema evolution management. The metadata repository stores various types of metadata and provides a consistent and united access mechanism to data to improve the effectiveness of information management. As shown in Figure 1, the three main processes are linked to the metadata repository. There are four types of metadata in the repository: (i) schema element dependency metadata; (ii) schema mapping metadata; (iii) schema change history metadata; and (iv) metadata from which the schema element dependencies are derived.

3.3. Schema Mapping Tool
The Schema Mapping Tool in Figure 1 imbeds two modules: Schema Change Templates (SCTs) and Schema Mapping Composition. The functionality of this tool is to generate and update schema mapping metadata which then can be used for view/query rewriting. SCTs represent all schema change scenarios that can occur in a spatial database environment and satisfy different schema changes. Based on mapping between the conceptual spatial schema and SQL schema (the implementation schema), a set of SCTs can be generalised in SQL schema as shown in Table 1.

<table>
<thead>
<tr>
<th>Conceptual Spatial Schema</th>
<th>SQL Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature Object Class</td>
<td>Table (one or more tables depending on the data types supported by the DBMS[17])</td>
</tr>
<tr>
<td>Non-spatial Object Class</td>
<td>Table</td>
</tr>
<tr>
<td>Spatial Attributes</td>
<td>Geometry Column or Geometry table with reference to the feature table [17]</td>
</tr>
<tr>
<td>Non-Spatial Attribute</td>
<td>Column</td>
</tr>
<tr>
<td>Relationship</td>
<td>Foreign Key and Primary Key (1:1 or 1:M)</td>
</tr>
<tr>
<td></td>
<td>Table with two Foreign Keys (M:N)</td>
</tr>
</tbody>
</table>

In this research, schema change is treated as schema mapping. So each schema change in the SCTs has corresponding schema mapping rules. Table 2 lists the mapping rules related to each SCT where A, B, C & D are sets of attributes, a, b, c & d are single attributes, and R, T & S are tables. The symbols are the normal relational algebra symbols: \(\pi\), \(\sigma\), \(\bowtie\), \(U\) and \(←\) mean project, select, join, decompose and rewrite respectively. When a schema change is specified by the DBA, schema mapping metadata between schema versions of before and after change are generated. The SCTs
also can advise whether or not a view should be generated. For example, after renaming a table, a view with the same name of the original table can be generated that references the renamed table.

### Table 2. Schema Change & Corresponding Schema Mapping

<table>
<thead>
<tr>
<th>Schema Change</th>
<th>Input Schema</th>
<th>Output Schema</th>
<th>Schema Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a column</td>
<td>R(A)</td>
<td>R(A,b)</td>
<td>π R(A) ← π R(A,b)</td>
</tr>
<tr>
<td>Add a table</td>
<td></td>
<td>T(A)</td>
<td></td>
</tr>
<tr>
<td>Rename a table</td>
<td>R(A)</td>
<td>T(A)</td>
<td>R(A) ← T(A)</td>
</tr>
<tr>
<td>Merge tables</td>
<td>R(A), S(A)</td>
<td>T(A,b)</td>
<td>R(A) ← σ (T(A,b)), S(A) ← σ (T(A,b))</td>
</tr>
<tr>
<td>Join tables</td>
<td>R(a,B), S(a,C)</td>
<td>T(a,B,C)</td>
<td>R(a,B)← π T(a,B,C) S(a,C)← π T(a,B,C)</td>
</tr>
<tr>
<td>Split a table</td>
<td>R(a,B,C)</td>
<td>S(a,B), T(a,C)</td>
<td>R(a,B,C)← S(a,B) T(a,C)</td>
</tr>
<tr>
<td>Decompose a table</td>
<td>R(A)</td>
<td>S(A), T(A)</td>
<td>R(A) ← S(A) ∪ T(A)</td>
</tr>
<tr>
<td>Move a column</td>
<td>R(a,B,c), S(a,D)</td>
<td>R(a,B), S(a,c,D)</td>
<td>R(a,B,c) ← π σ (R(a,B) ~ S(a,c,D)); S(a,D) ← π S(a,c,D)</td>
</tr>
<tr>
<td>Split a column</td>
<td>R(a,B)</td>
<td>R(c,d,B)</td>
<td>R(a) ← F(R(c,d))</td>
</tr>
<tr>
<td>Merge columns</td>
<td>R(a,b,C)</td>
<td>R(d,C)</td>
<td>R(a) ← F₁(R(d)), R(b) ← F₂(R(d))</td>
</tr>
</tbody>
</table>

*In order to be able represent mapping in GAV, a column is added as the condition when two tables are merged;*

*To avoid data loss, when join tables, outer join is needed.*

Schema Mapping Composition is used to combine schema mappings and schema evolution can result in the existing schema mapping being invalid. In order to ensure the consistency of schema mapping, schema mapping combines two schema mappings into one [18]. This approach can be applied to vertical and horizontal mapping. When a data source schema changes, a new schema mapping between the global schemas and local schemas can be derived from the original mapping and the change schema mapping. Likewise, for the local schemas, schema mapping between older versions and the current one can be derived from the schema changes and the old schema mapping.

### 3.4. Schema Element Dependency Tool

The Schema Element Dependency (SED) tool in Figure 1 is used to generate column level SED metadata for a newly created view, and to update the SED metadata when a view is rewritten. The SQL Parser module embedded with this tool is based on the SED metamodel developed in this research. According to the SED metadata, the impact of schema changes can be analysed, and, affected views can be detected for view rewriting.

### 3.5. View Rewriting Tool

The Query Rewriting module, embedded in the View Rewriting Tool in Figure 1 is used to rewrite queries against one schema into queries against another schema according to the schema mapping. Since a view is a stored query, view rewriting can be regarded as being similar to query rewriting. The View Rewriting Tool identifies views (spatial views and non-spatial views) according to SED metadata and then rewrites and recompiles them according to the schema mapping metadata. However, there is a difference between view rewriting and query rewriting. View rewriting is a one-off operation as it only occurs when schema changes occur. Query rewriting happens every time the query is processed. This is because any affected view can be detected from the SEDs while the queries from the application can only be detected when they are processed.
3.6. Example of Schema Updating

Figure 2 shows an example road spatial database schema S and two proposed schema changes for S. The first one is to merge ExistingMainRoads and ProposedMainRoads into MainRoads, and the second one is to split MainRoads into MainRoads and RoadNames. In addition a spatial view Highways is built against schema S₁ to display all the highways.

![Figure 2. Schema Changes in S](image)

When a schema changes, the DBA specifies the schema change using the schema mapping tool. Then the required schema mappings M₁₂ and M₂₃ are generated according to the schema mapping rules defined in the SCTs. In addition, schema mapping M₁₃ is also generated by combining M₁₂ and M₂₃. Table 3 lists the schema mappings generated by the schema changes and schema composition (in the form of GAV).

<table>
<thead>
<tr>
<th>Mapping Name</th>
<th>Schema Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>M₁₂ (mapping between S₁ and S₂)</td>
<td>S₁.ExistingMainRoads (n, g, lc, ru, sl, rtID) ← σ rs = ‘Existing’ (S₂.MainRoads (n, g, lc, ru, sl, rtID))</td>
</tr>
<tr>
<td></td>
<td>S₁. ProposedMainRoads (n, g, lc, ru, sl, rtID) ← σ rs = ‘Proposed’ (S₂.MainRoads (n, g, lc, ru, sl, rtID))</td>
</tr>
<tr>
<td>M₂₃ (mapping between S₂ and S₃)</td>
<td>S₂.MainRoads (n, g, lc, ru, sl, rtID) ← S₂.MainRoads (nID, g, lc, ru, sl, rtID) ≡ S₂.RoadNames (nID, rn)</td>
</tr>
<tr>
<td>M₁₃ (mapping between S₁ and S₃)</td>
<td>S₁. ExistingMainRoads (n, g, lc, ru, sl, rtID) ← σ rs = ‘Existing’ (S₃.MainRoads (nID, g, lc, ru, sl, rtID) ≡ S₃.RoadNames (nID, rn))</td>
</tr>
<tr>
<td></td>
<td>S₁. ProposedMainRoads (n, g, lc, ru, sl, rtID) ← σ rs = ‘Proposed’ (S₃.MainRoads (nID, g, lc, ru, sl, rtID) ≡ S₃.RoadNames (nID, rn))</td>
</tr>
</tbody>
</table>

Where n, g, lc, ru, sl, rtID, nID and rn denote Name, Geometry, LaneCount, RoadUsage, SpeedLimit, RoadStatus, RoadTypeID, NameID and RoadName respectively.

When schema S changes from S₁ to S₂, Highways built against S₁ will be invalid. The view rewriting tool rewrites and recompiles Highways to accommodate the schema change. Table 4 shows the rewriting of Highways (expressed in relational algebra) after schema S changes from S₁ to S₂. More comprehensive examples can be found in [19].
Table 4. Rewriting of Spatial View - Highways

<table>
<thead>
<tr>
<th>Before Rewriting (against $S_1$)</th>
<th>After Rewriting (against $S_2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma \text{tn} = \text{'highway'}((S_1.\text{ExistingMainRoads}(n, g, lc, sl, rtID)) \cup S_1.\text{ProposedMainRoads}(n, g, lc, sl, rtID)) \bowtie S_1.\text{RoadType}(tID, tn))$</td>
<td>$\sigma \text{tn} = \text{'highway'}((\sigma \text{rs} = \text{'Existing'} \cup \sigma \text{rs} = \text{'Proposed'}(S_2.\text{MainRoads}(n, g, lc, ru, sl, rs, rtID)) \bowtie S_2.\text{RoadType}(tID, tn))$</td>
</tr>
</tbody>
</table>

4. SUMMARY

Schema evolution is essential in a FSDBS because in many implementations, it is dependent on databases owned, maintained, and importantly are modified by other organisations. Managing schema evolution in such an environment is a significant challenge. This paper proposes methodologies to enable semi-automatic management of schema evolution in a FSDBS environment so that schema changes will be transparent to applications and users. To achieve this, SEDs across databases are determined, and, a set of SCTs proposed to define a rich bounded set of schema change scenarios in a spatial database environment, and provide the mapping rules for each change. View generation/rewriting and query rewriting are proposed as the solution to allow changes to the various databases to be accommodated.

Effective management of schema evolution in a FSDBS is an integral part of an SDI. It ensures discovery and access to spatial data and services is continuous even though schema changes have occurred. It also provides schema evolution transparency to end-users and minimises modification of applications thus saving time and money. Effective management of schema evolution also increases the reusability of spatial data; ensures the longevity of spatial data and services well into the future; improves spatial data and service management; and engenders collaboration and interoperability between data custodians and application developers and users.

ACKNOWLEDGEMENT

The authors would like to thank Landgate and Curtin University for providing funds to support this PhD research.

REFERENCES