Data exchange between distributed spectral databases

A. Huenia, T Malthusb, M. Kneubuehlera, M. Schaepmana

a Remote Sensing Laboratories, University of Zurich, Winterthurerstrasse 190, 8057 Zurich, Switzerland
b Environmental Earth Observation, CSIRO Land and Water, Clunies Ross Street, Black Mountain ACT 2601, Australia

Abstract

Spectral databases constitute one of the components of a complete observing system, storing in situ spectroscopic measurements plus associated metadata and providing data for the validation, calibration, and simulation of imaging spectrometer products. Such databases may be employed by physically or organisationally separate entities. Consequently, methods for data exchange between distributed spectral databases are required, allowing the transfer of defined subsets of spectral data including their full metadata context from a source to a target system. The data exchange comprises generic approaches to the sequential steps of ordered table row export, relational storage in XML files, and nonconflicting import into the target database. The SPECCHIO spectral database system was used as a test bed for the data exchange between databases of identical schemata and according import/export functionality has been added to the SPECCHIO application. Import and export speeds were assessed using test cases of different metadata space densities, a score for the density with which associated metadata are detailed, and the potential utility as a quantitative rating for quality. Future spectral databases should allow the exchange between heterogeneous systems, ideally implementing a common subset of metadata parameters and thus supporting the long-term usability and data sharing between research partners.

1. Introduction

1.1. Complete observing systems in support of Earth system sciences

Since the birth of our planet some 4.54 billion years ago (Dalrymple, 2001), change has been a constant factor (ESA, 2006). Natural parameters and forces such as the geometry of the Earth’s orbit, solar irradiation, and plate tectonics have driven this change. However, these natural sources of change have been gradually supplemented by the anthropogenic influence, which has become a new factor to be reckoned with on a global scale. There is mounting evidence that human activities in the last 250 years have had a profound influence on the changes of the Earth System (ESA, 2006; IPCC, 2007). These changes not only threaten to change finely tuned ecosystems but also jeopardise the functioning of human societies (Stern, 2007). Consequently, policymakers are obliged to react to these threats by implementing useful mitigations. Such actions must be based on informed decisions, which in turn must be delivered by science. For these reasons, a thorough understanding of the Earth system and the changes induced by anthropogenic and natural causes is of high importance and represents one of the biggest current scientific challenges (National Research Council, 2007). The issue of global change is to be tackled by addressing objectives defined during the first Earth Observation Summit in 2003 (GEO, 2005). These objectives are put into action by the Group on Earth Observations (GEO) by continuously monitoring the state of the Earth, increasing the knowledge about the dynamic Earth processes, and enhancing the prediction of the Earth system (GEO, 2005).

The global, technical strategy to put the above objectives into action is to build a “Global Earth Observation System of Systems” (GEOSS). GEOSS will comprise components and processes needed to generate information from signals collected by space-based, airborne, and in situ sensors (GEO, 2005). Systems like GEOSS, comprised of sensors at various scales, storage, and processing systems, have also been termed “complete observing systems” (National Research Council, 2007). Within a complete observing system, in situ spectral data play a crucial role by providing a baseline for satellite and airborne measurements (GCOS, 2009). The remainder of this section describes the structure of complete observing systems and the role spectral databases play within such systems.

As proposed in this paper, a complete observing system may be defined as a system including all primary sensors, secondary
sensors, storage and processing capacity, and auxiliary data needed to describe complex Earth systems as entirely as possible. In this definition, primary sensors are dependent on the scale of the phenomena to be observed, e.g., satellite observations for global scales and airborne systems for regional scales, while secondary sensors deliver observations supporting the primary data. Thus, the structure of a complete observing system may vary from small, well-contained processing and archiving systems designed for specific primary sensor systems (Hueni et al., 2009a) to large networks with spatially and organisationally distributed entities including many sensors and data archives (Bernard et al., 2005; Bernholdt et al., 2005; Latham et al., 2009; Lawrence et al., 2009; Muchoney, 2008). The latter have lately been based on grid architectures, employing metadata catalogues and vocabulary services to present users with a homogeneous interface to data stored in a network of heterogeneous computing systems (Lawrence et al., 2009). Such system architectures support the dissemination, exchange, and sharing of products, eventually allowing the generation of new information based on existing products (Christian, 2008; Durbha et al., 2008; Pearlman et al., 2008). Independent of the size of the complete observing system, its main feature is the assimilation of observations at various scales including ground-based data, enabling the calibration and validation of data and products (Cao et al., 2008; Liang et al., 2005).

Spectroscopic point observations acquired by field spectroradiometers are one form of in situ data (Milton et al., 2009). In the remote sensing context, spectroradiometers are used for the collection of spectral data for calibration, validation, and simulation of imaging spectrometers and derived products covering all domains of the Earth system (Schaeppman et al., 2009). As such, field spectroradiometer data constitute an important in situ part in a complete observing system and must be stored in a manner enabling efficient retrieval and independent, comprehensive assessment regarding their usefulness and quality. We propose that spectral databases are the tool of choice for the storage of spectroscopic point observations within complete observing systems.

1.2. Spectral databases and data exchange

Spectral databases are systems for the storage of spectral data acquired by spectroradiometers under both field and laboratory conditions, augmented with associated auxiliary data. From a technical point of view, spectral databases include systems based on relational or object-oriented databases, but do not include collections of spectral data held in any semistructured or static way, such as files and folders on servers or spectral library files. Metadata play a key role in spectral database systems, as they define the context of each spectrum and allow the retrieval of spectra via metadata subspace projections (Hüni et al., 2007). In fact, one may argue that the metadata are more important than the primary record, as they are paramount to broad and long-term use and interpretation of scientific data (Michener, 2000).

The common use cases of spectral databases include:
(a) storage and retrieval of spectral data using a centralised server, which may be part of a complete observing system, with permanent network connection and intranet/internet accessibility,
(b) incremental storage and documentation of ongoing field campaigns on computers not necessarily connected to a network,
(c) maintenance of several databases with differing contents for project-specific or educational purposes with varying data access rights, and
(d) building of specialised, centralised databases in research networks or complete observing systems by copying spectral data collections between database servers. Most of these use cases require the exchange of data between separate database entities at some point.

In general, information transfer between systems is carried out for various reasons: enhanced storage redundancy, disaster recovery or increased access speed by data replication (Chen et al., 2007), shared data access to collections of data resources (Bernholdt et al., 2005; Pouchard et al., 2003), and consolidation of data in central storage systems, e.g., for auditing purposes (Chen et al., 2007). For standard situations, data replication techniques between database systems are well established and allow live replications using lock mechanisms to prevent data inconsistencies. However, full database access rights and simultaneous online connections to the involved schemata are required to carry out such data exchanges. In the case of spectral databases, possible ontologies range from stand-alone machines to computers being part of a network. This implies that standard database replication techniques based on, e.g., authentication services (Chervenak et al., 2005), may not be used. The following section describes the specific requirements for data exchange between distributed spectral databases of identical schemata, covering the described use cases. It must be noted that many, if not most, of the problems treated within this study are not necessarily specific to spectral databases but occur with relational databases in general. In this respect, spectral databases may be seen as a case example. However, within the field of remote sensing and geographic information in general, the notion of spectral databases is fairly new and only a few implementations exist. As a result, there are no standards for the data exchange between spectral databases and the existing protocols available for sensor information (Cox, 2007; Na and Priest, 2006) or geographic information (Di, 2003) seem ill fitted to the particular nature of spectral point data collections.

1.3. Definition of the partial database import/export problem

As defined in the preceding section, the use cases of spectral databases necessitate methods for the data exchange. In particular, a defined spectral dataset including its full metadata context is required to be transferred between two relational spectral database systems. We refer to this requirement as the partial database import/export. The partial nature of the problem results from the requirement of exchanging specific spectral datasets only rather than the whole database content. The data are to be imported into the target database without causing any conflicts, producing an exact copy of the original dataset (Barcel, 2009). This copy process is similar to the initial copy applied during the setup of database replications. However, in contrast to the common replication, which defines the set of tables to be replicated (Chen et al., 2007), the partial database import/export requires the replication of a data subset contained in several tables. For these subsets, both the required tables and their involved content are defined by the metadata context of the primary resource. It is due to this context dependency that no “off the shelf” solutions seem to exist, despite the fact that data exchange is an old and common data management problem (Fagin et al., 2005). Consequently, the development of specialised code and interfaces is required. The remainder of this section describes the implications of the relational, normalised storage on the data exchange and the requirements for the data export and import operations and associated schema and access-related constraints.

Generally, relational databases store data in a normalised form, meaning that data are represented naturally and completely in simplest, least redundant form (McFadden and Hoffer, 1988; Yannakakis, 1996). The normalised form avoids anomalies during
insertions, updates, and deletions and lowers the required storage size by minimising data redundancies (Codd, 1990). The relational approach is vastly superior to flat records when it comes to query speed, data integrity, and storage size but incurs a higher complexity due to data being spread over a multitude of tables. It is therefore beneficial to retain the relational, normalised structure during data exchange for two main reasons. First, an export to a flat file structure will introduce redundancy and hence considerably increase the data size during transfer. Second, restructuring the data to create relational table entries during import on the target database is not trivial and may not achieve exact reproductions of the original relations (Florescu and Kossman, 1999; Shanmugasundaram et al., 2001).

The main functionality of the export operation is the extraction of a spectral data subset and its storage in a transferrable and relational form. As already alluded to above, the tables and tuples (table rows) involved in an export operation are defined by the metadata context of the primary resource. The export must therefore retrieve the metadata context of the spectral data subset in question. To do this, knowledge about the ontology of the schema is required. Information about the tables and their associations may be extracted from the schema, a process commonly referred to as entity relationship extraction (Premerlani and Blaha, 1994). This extraction utilises the foreign key information contained by the schema. Foreign keys are referential constraints between tables, enforcing 1:N relationships, i.e., they provide the means by which one tuple can refer to another tuple (Buneman et al., 2001). Essentially, the export therefore needs to extract the entity relationship information of the given schema. This information is also of importance regarding the import into the target system.

The insert of relational data into a schema requires a certain order of insert statements to avoid inconsistencies. These are caused by foreign key violations, which happen if a tuple tries to reference another tuple that has not yet been inserted. In other words, a table row can only be inserted when all referenced tuples are inserted beforehand. Thus, the correct order of the inserts is essential for consistent inserts. This order may already be provided by the export operation, as it possesses the ontology information about the schema.

The nature of relational databases gives rise to a number of constraints regarding the import of data into a target system. These further complicate the data exchange and are introduced in the following paragraphs.

Every table within relational schemata requires a primary key in order to uniquely identify each row within a table. Importing data from a different database leads to primary key conflicts if identical primary key values already exist in the target database. The data import must therefore avoid the creation of such conflicts by assigning unique key values to the imported table rows.

The tables of a database may be categorised into user tables and system tables. Normal database users can modify user tables while system tables can be read by all users but only changed by system administrators. This, for example, serves to protect the integrity of categorical variables that have a defined range of possible values. Inserting system table information into a database requires the corresponding rights, i.e., the role of system administrator. For the given use cases of distributed spectral databases, this causes a problem as different administrators are often involved. A live connection between two databases for system table information transfer is only possible if the exporting user has administrator rights on the target database. A live transaction is therefore generally not feasible for the given scenario. Consequently, the transfer of data must be arranged by the means of a file that can be sent to the administrator of the target system for offline import. The file format must allow for the storage of alphanumeric and binary data, the latter enabling the transfer of imagery or data vectors encoded in a binary format.

The aim of this paper is thus to present methods for the partial data exchange between distributed spectral databases where neither constant database availability nor common administrator access can be assumed.

2. Methods

The solution of the partial database import/export problem requires a number of concepts described in this section, which cover the retrieval of the database structure and categorisation of tables, the definition of the sequence required for an ordered table export/import, the definition of a suitable data exchange file format, import strategies that avoid the occurrence of conflicts, and a corresponding, object-oriented software design. The concepts were finally implemented as a new functionality of the SPECCHIO database system (Hueni et al., 2009b). SPECCHIO serves as a repository for field and laboratory spectroscopy data and related metadata and is based on a client-server architecture with data stored in a relational MySQL database (MySQL AB, 2007) with end-user access provided via a platform-independent Java application (Sun Microsystems Inc (JavaTM 2 Platform Standard Edition, 2006)). The SPECCHIO schema implements a comprehensive data model, allowing the nonredundant definition and storage of metadata. A subset of the SPECCHIO schema comprising system and user tables is used to illustrate the concepts presented in this paper. The according Entity Relationship Diagram (ERD) is provided in Fig. 2.

The final implementation was tested using a number of test cases, characterised by a new metric termed Metadata Space Density (MSD), which is introduced within this section.

2.1. Retrieval of the relational structure

Knowledge about the topology of relations is required for an automated information extraction by the export process. It allows for traversing of the network defined by relations and associations, enabling the retrieval of the full metadata context of a primary resource.

The relational structure of a schema can be retrieved from a database in a generic fashion, given that the associations were properly defined via foreign keys during implementation and that the Relational Database Management System (RDBMS) offers access to this information. In the case of MySQL databases, the required information is contained in the information_schema (MySQL AB, 2007).

First, all table names of the schema are extracted. The results are then used to retrieve information about all fields per table by a generic SQL query:

```sql
SELECT column_name, data_type, column_key
FROM information_schema.columns
WHERE table_name = '<table name>'
AND table_schema = '<schema name>'
```

The above query allows assigning a defined data-type and field category (primary key, foreign key, or normal field) to all fields. Associations between relations are retrieved from the key_column_usage table. This compiled field information is required to resolve associations during data export and to carry out key exchanges during data import, as will be detailed further on.

2.2. Table categories

Tables need to be assigned to categories for the reason of being treated differently during import and export. From a user point of
Author's personal copy

view, tables belong to two main categories: system and user tables. The latter can be modified by all database users while the former can only be edited by system administrators. As a rule, user tables can reference system tables but not vice versa. System tables thus store data shared by multiple user table rows and require special treatment during import and export, as will be demonstrated in the sections below.

A generic identification of table categories is possible based on schema information. In the schema provided in Fig. 1, all user tables reference the specchio_user table. This association is needed for the implementation of the multiuser concept and can thus be used systematically for system table determination. Consequently, the system tables are identical to the set of tables not being user tables

\[ \text{sys\_tables} = \text{all\_tables} \setminus \text{user\_tables} \]

System tables may further be categorised into nodes and end-nodes. The importance of this differentiation will be detailed in the export concept. System table end-nodes encompass all system tables that are nodes at the edge of the entity relationship network; they have no foreign keys and are thus only referenced by other tables. Identification of the end-nodes can be achieved by projecting the system tables to a subspace using a “no foreign keys” constraint. Applying this categorisation to the entities shown in Fig. 1 results in the groups listed in Table 1.

All tables may optionally belong to further, special table types: intersection (cross reference) and recursive tables (Table 2). Both types need special attention during the design of the export algorithm. They can be identified using the structure information of the schema.

---

**Fig. 1.** Entity relationship diagram showing user and system tables and their associations.

---

**Table 1**

<table>
<thead>
<tr>
<th>User tables</th>
<th>System tables nodes</th>
<th>System table end-nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>campaign</td>
<td>institute</td>
<td>country</td>
</tr>
<tr>
<td>picture</td>
<td>instrument</td>
<td>measurement_unit</td>
</tr>
<tr>
<td>spectrum</td>
<td>sensor</td>
<td>sensor_element_type</td>
</tr>
<tr>
<td>spectrum_x_picture</td>
<td>sensor_element</td>
<td>specchio_user</td>
</tr>
<tr>
<td>spectrum_x_target_type</td>
<td>target_type</td>
<td></td>
</tr>
</tbody>
</table>

---

**Table 2**

<table>
<thead>
<tr>
<th>Table type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersection</td>
<td>A table that is used to resolve N:M relationships between two tables by storing primary key values of both tables in foreign key fields. Intersection tables are introduced during the normalisation of the database schema.</td>
</tr>
<tr>
<td>Recursive</td>
<td>Recursive tables are defined by introducing foreign keys that reference the same table (recursive associations). This allows, for example, for the storage of hierarchical structures in a single table.</td>
</tr>
</tbody>
</table>

---

2.3. Ordered table export

Tables are to be exported in a defined order, allowing insertion in the target system without causing foreign key constraints to
fail. In the following, we assume that data are exported in a campaign context; i.e., all spectral data and associated metadata being part of a spectral measurement campaign will be exported (nonetheless, the export mechanism would work in a similar fashion on spectral datasets belonging to one or more campaigns). The export virtually navigates through the entity relationship network by resolving all associations and thus relies on the relational structure retrieved as described above.

The export of a row \( R \) of table \( T \) consists generally of three main steps for which corresponding operations are defined (Table 3).

The above steps hold true for the user tables. However, system tables require some more rules, as will be demonstrated by the following description of an export for the schema shown in Fig. 1.

The export starts at the campaign table with a user-defined row that specifies the campaign to be exported. The first operation is therefore: ref(campaign). The campaign has one foreign key, referring to the specchio_user table; thus export is called on that table. By resolving the associations, a cascade of operations evolves as shown below.

\[
\text{ref(campaign)} \\
\text{ref(specchio_user)} \\
\text{ref(institute)} \\
\text{ref(country)} \\
\text{exp(country)} \\
\text{iref(country)} \\
\text{exp(institute)} \\
\text{iref(institute)} \\
\text{exp(specchio_user)} \\
\text{iref(specchio_user)} \\
\text{exp(campaign)} \\
\text{iref(campaign)} \\
\text{ref(spectrum)} \\
\text{...}
\]

In the above cascade of operations, skipped iref() operations are printed in italics and stroked through; these represent special cases:

1. The iref(country) operation would lead to undesirable consequences; all institutes of this country would be exported, thus violating the requirement that only the metadata context relevant for the chosen campaign shall be exported. Generally speaking, the iref operation must not be applied to system tables without foreign keys; i.e., these are tables that are only referenced by other tables. In fact, this is exactly the definition of the system table end-nodes.

2. The iref(institute) operation is undesirable as neither all instruments nor all users belonging to this institute shall be exported. In this respect, it is similar to the first special case above. However, the institute is not a system table end-node and therefore further rules are required. The choice whether a referencing table must be exported depends on answers to the following question: "What further tables are needed to define the current table?" The sensor system table serves as an example for this case. The sensor_element table is needed for the full definition of the sensor as it holds the band characteristics. A corresponding rule, answering this question, cannot be created from the information scheme; it requires know how about the business logic and must therefore be defined by the system developer.

3. The iref(specchio_user) operation must not be called, as this would trigger the export of all data this user has ever entered into the system. Therefore, the iref operation of system tables must not consider user tables.

The consolidated rules regarding the iref operation for system tables are summarised in Table 4. The special exceptions for the presented example (Fig. 1) are (a) institute: no iref at all and (b) sensor: iref for sensor_element only.

The export operations ref, exp, and iref combined with the above iref-rules for system tables suffice to export a spectroradiometer campaign including the full metadata context. Multiple exports of the same table row are avoided by keeping a list of already exported rows per table.

### 2.4. File format

The nature of the distributed databases considered in this paper requires an electronic file for the exchange of data between systems. The file format should be able to store all data types occurring in the exported schema, i.e., alphanumeric and binary. Furthermore, it should be human readable for easy interpretation without the need for special software and contain information allowing consistency checks during import.

The Extensible Markup Language (XML) is a widespread file format that meets these requirements, although the transfer of binary data needs special attention. XML is based on Standard Generalised Markup Language (SGML) (ISO, 1986; Needleman, 1999). For these reasons, XML was chosen as file format for the data exchange in SPECCHIO.

The file format for the relational data exchange of spectral campaigns can be described using Extended Backus Naur Form (EBNF) (ISO/IEC, 1996) as follows:

\[
\text{spectral\_campaign\_exchange\_file} = \langle \text{campaign} >, \langle /\text{campaign} >; \\
\text{table} = \langle \text{table} >, \text{field}, \{\text{field}\}, \langle /\text{table} >; \\
\text{field} = \langle \text{field\_name=}'', \text{field\_name=}, '' >', \text{field_value}, \langle \text{field}\rangle >; \\
\text{Examples of the representation of table rows in XML can be found in Fig. 2.}
\]

Including binary data in a text file requires suitable encoding. Hexadecimal representation of byte values allows such storage of binary vector or image data as hex strings in text files and was selected as a suitable solution.
2.5. Import

Importing a spectral campaign into a new database system requires the insert of new rows into the required tables. Due to the ordered table export, the XML file already contains the tables in the correct order ready for insert. However, three issues remain and are discussed further: (a) the insertion of tables with new primary keys to avoid conflicts with already existing rows, (b) the exchange of foreign key field values with the new primary key values, and (c) avoiding duplication of existing system table entries.

2.6. Primary and foreign key exchange

Primary keys act as unique identifiers for table rows and are quite commonly artificially generated keys; i.e., they have no relation with the rest of the content of the row. In any case, inserting rows originating from a different system leads to inconsistencies if the key values already exist in the target system. Primary keys of new table rows must therefore be newly created during the insert to ensure the uniqueness of keys. Creating new primary keys also implies that all foreign key values referencing the old primary key must be updated to refer to the new key value. For this study, we rely on the fact that all tables use system-generated primary key values, automatically assigned to the primary key field on insert.

The insert of tables is a sequential process: (1) the table fields are read from the XML input file, (2) an insert statement is created, and (3) the insert is executed. The creation of insert statements requires the following steps:

1. Removal of the primary key field from the field list of the table row to be inserted.
2. Exchange of the values of all foreign key fields with the new primary key values on the target system.

The foreign key exchange requires continuously updated lists of old/new primary key pairs for all tables during import. Every insert of a table row generates a new primary key, which is stored in a list along with the original key value.

This mechanism is illustrated in Fig. 2: the XML table data shown on the left are transformed to SQL insert statements. The old primary keys, shown in bold in the XML definitions, and the new primary keys are stored in table-specific lookup tables (LUT). Foreign key values are replaced by the new values, for example, when building the insert statement for the campaign, the user_id value 37 of the campaign row is swapped with the new specchio_user primary key value of 58.

2.7. System tables

System tables are designed to hold general information, which is referenced by entries in the user tables. Changes to the system tables can have grave consequences for all data stored in user tables and special care must be taken to ensure the integrity of system table information. The duplication of system table entries on import of a spectral campaign must be avoided, as it would lead to inconsistencies in the database. For this reason, checks for already existing table rows must be carried out. Existing system table entries are identified by building a query containing all fields apart from the primary key. Foreign key values in such queries must be replaced with the values of the current system in a manner identical to the foreign key exchange during insert. The primary keys of already existing rows are entered along with the old primary keys into the key lookup table. This ensures that other tables can carry out foreign key value exchanges before insert or perform existence checks. Consequently, system table rows are only ever inserted if no existing, matching table entry is found.

2.8. Software design

One goal of this study was the actual implementation of data exchange functionality as a new feature of the SPECCHIO Java application to support the consolidation of spectral data collections stored in various SPECCHIO database instances. The
implementation of the introduced import/export concepts therefore required an object-oriented software design, as will be detailed in this section.

The generic and recursive nature of the partial database import/export allows a streamlined object-oriented design approach, illustrated by the Unified Modelling Language (UML) diagram in Fig. 3 (Booch et al., 2000). The design consists of classes which model the structure of the database in a generic way; i.e., table structures are not preprogrammed but created during runtime.

The DbTable class models the table entities. A DbTable instance is instantiated with a specific table name and contains methods to retrieve the table structure autonomously. The structure is stored in dynamic, dedicated lists, holding all fields of the respective table, primary/foreign keys, exported row ids per primary key, and key LUTs. The DbTable contains further methods to export a row of this table to XML, insert data as new row, effect foreign key constraints, and check on the existence of identical rows to avoid duplication of system tables.

Database table fields are modelled as two classes, TableField for normal fields and FkTableField for foreign key fields, where the latter is a subclass of the former and contains additional information about the referenced table. The TableField class is again very generic with the actual value of a field stored in a subclass of the abstract class FieldValue. There are FieldValue subclasses for all types of fields used in the SPECCHIO schema, such as Integer, Boolean, DateTime, Varchar, and Blob (binary large object). The FieldValue class holds methods to read the actual value from either SQL result sets or strings when parsing XML files and to write the value to a string for XML file export. The conversion to a string representation of a value depends on the data type, for example, binary values are transformed into their hexadecimal form.

On creation of a TableField, the required instance of a FieldValue is instantiated by calling the FieldValueFactory. This class utilises both Singleton and Factory patterns (Gamma et al., 1997); i.e., it may only exist as one instance with a global access point. The Factory pattern is used to create FieldValue instances, e.g., the getInstance() method returns an instance of a class corresponding to the given name, and the FieldValueFactory::getFieldValue() method returns the FieldValue instance of the given FieldValue field.

In the context of relational databases, the MSD for a table T is defined as

\[
MSD_T(id) = \sum_{i=1}^{n} \delta_{i} + \sum_{i=1}^{u} MSD_{ref_table}(val(u_{fk_col}, id)) + \sum_{i=1}^{s} \delta_{i} + MSD_{ref_table}(rt_{id})
\]

where, id is the primary key value identifying a row in table T; n is the number of non-key fields of T; col is the non-key column; u_{fk_col} is the foreign key column of T referencing a user table; sys_{fk_col} is the foreign key column referencing a system table; u is the number of user tables referenced via foreign key fields of T; s is the number of system tables referenced via foreign key fields of T; r if T \in A: number of tables referencing T, else: 0 < A: a table that needs to resolve indirect referencing for MSD calculation; rt_{id} is the primary key of a table referencing the row identified by id in T; not_null(col, id) is the function returning 1 if the supplied column is not empty; ref_table() is the function returning the table name of the table referenced by fk_col of T; iref_table() is the function returning the table name of the table referencing T; val(col, id) is the value of the column col in the row of table T identified by id.

Note that the last term of the summation will only be called for selected tables to restrict the density calculation to metadata related to the current primary resource only. In the case of the SPECCHIO schema, only the spectrum table needs to resolve indirect references, thus: \(A = \{t: \text{spectrum}\}\).

The recursive nature of the function definition ensures that the relational structure is traversed automatically. In the case of spectra referring to other spectra, e.g., a target spectrum referencing a reference panel spectrum, the MSD can reach higher values than expected, as the MSD of the referenced spectrum is also taken into account. While logically true, such a measurement may lead to false perceptions about the density. Therefore, the MSD is restricted to the metadata space of just one primary resource.

Multiple calls of MSD on the same row due to foreign key referencing must be avoided by keeping a list of ids already handled during the current MSD operation.

2.9. Metadata space density

The Metadata Space Density (MSD) is a metric for the quantitative information content of the metadata space of a resource. In the context of this study the MSD serves to characterise the nature of the test cases used to assess the performance of the data-exchange implementation. We define the MSD as the total count of values in all dimensions of the metadata space, where the metadata space comprises all user-definable parameters. The metric is fairly simple but its retrieval from data scattered over the relational tables requires some special attention as detailed below.

In the context of relational databases, the MSD for a table T is defined as

\[
MSD_T(id) = \sum_{i=1}^{n} \delta_{i} + \sum_{i=1}^{u} MSD_{ref_table}(val(u_{fk_col}, id)) + \sum_{i=1}^{s} \delta_{i} + MSD_{ref_table}(rt_{id})
\]

where, id is the primary key value identifying a row in table T; n is the number of non-key fields of T; col is the non-key column; u_{fk_col} is the foreign key column of T referencing a user table; sys_{fk_col} is the foreign key column referencing a system table; u is the number of user tables referenced via foreign key fields of T; s is the number of system tables referenced via foreign key fields of T; r if T \in A: number of tables referencing T, else: 0 < A: a table that needs to resolve indirect referencing for MSD calculation; rt_{id} is the primary key of a table referencing the row identified by id in T; not_null(col, id) is the function returning 1 if the supplied column is not empty; ref_table() is the function returning the table name of the table referenced by fk_col of T; iref_table() is the function returning the table name of the table referencing T; val(col, id) is the value of the column col in the row of table T identified by id.

Note that the last term of the summation will only be called for selected tables to restrict the density calculation to metadata related to the current primary resource only. In the case of the SPECCHIO schema, only the spectrum table needs to resolve indirect references, thus: \(A = \{t: \text{spectrum}\}\).

The recursive nature of the function definition ensures that the relational structure is traversed automatically. In the case of spectra referring to other spectra, e.g., a target spectrum referencing a reference panel spectrum, the MSD can reach higher values than expected, as the MSD of the referenced spectrum is also taken into account. While logically true, such a measurement may lead to false perceptions about the density. Therefore, the MSD is restricted to the metadata space of just one primary resource.
3. Results

The results of the implemented concepts are presented in the form of speed and data size metrics hereafter. Tests were carried out on a machine equipped with a 2.2 GHz Intel Core 2 Duo processor and 2 GB RAM at a clock speed of 667 MHz, with the database server and the Java application running on the same machine.

Speed test results are based on the logged system time per insert/export operation, resampled to rows per second in 0.1 s steps. The sampling interval of 0.1 s was chosen to document the short-term fluctuations in performance present in a multitasking system.

Four test cases were created to assess the impact of the number of spectral bands, Metadata Space Density (MSD), and number of spectra on the import/export speed (Table 5). The MSD is given as mean (μ) and standard deviation (σ). The spectral data of the test cases were acquired with two makes of spectroradiometers: the ASD FSFR (Analytical Spectral Devices Inc., 2007) and the GER 3700 (Spectra Vista Co., 2005).

3.1. Export speed

The export speed was measured as the total of exported rows versus system time and as exported Rows Per Second (RPS). The results for the ASD SPARSE and GER SPARSE test cases are shown in Fig. 4. In a first phase, both exports start with a high number of RPS till about 2160 rows for ASD SPARSE and 700 rows for GER SPARSE, and then the speed drops to mean values of 119 RPS (ASD) and 216 RPS (GER). The high RPS at the beginning of the exports are associated with the extraction of sensor band specifications. Therefore, the ASD test case features a longer period of high RPS due to the higher number of sensor bands compared to the GER (see also Table 6). The difference in RPS between ASD and GER during phase 1 is not readily explained. This effect is presumably caused by the caching mechanisms of the database server.

In a second phase, the lower export speeds following the sensor export are related to the amount of data per spectrum. This data volume per spectrum is mainly governed by the size of the spectral data vector; i.e., it is a function of the number of bands of the sampling instrument. The dependency of spectral table size on the number of bands of the sampling instrument is presented in Table 6.

The observed mean export speeds during the spectrum export partly reflect the difference in data volume of a factor of about 1:3. However, as the spectral table contains mainly metadata and spectral data are contained in one field only, the drop in speed is not a direct function of the number of bands; i.e., the effective export speed for ASD spectra is about half the export speed of GER spectra. The undulating RPS curves are most likely caused by the varying data flow between database server and Java application; however, the real causes of these short stalls are difficult to assess and beyond the scope of this paper.

Fig. 5 shows the export speed measurements for the ASD DENSE test case. The observable, inverse exponential drop in RPS is the result of combined effects caused by the characteristics of the ASD DENSE test case, which contains a lot of metadata held by table rows of relatively small data volumes when compared to the spectrum entity. The export of spectral vectors is interspersed with metadata and, therefore, no sharp drop of RPS after exporting the sensor band specifications can be observed. The gradual drop in RPS is a penalty caused by the increasing time needed to check the lookup tables for already exported rows.

The GONIO test case is a real world example of a sampling campaign, comprising data of two different sensors (ASD FSFR and GER 3700) and a host of metadata, although not as excessive as the

Table 5
Test cases for speed and data size measurements.

<table>
<thead>
<tr>
<th>Test case</th>
<th>No. of spectra</th>
<th>MSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASD SPARSE</td>
<td>1920</td>
<td>μ: 13 σ: 0</td>
<td>ASD FSFR spectra with minimal metadata description</td>
</tr>
<tr>
<td>GER SPARSE</td>
<td>1920</td>
<td>μ: 14 σ: 1</td>
<td>GER 3700 spectra with minimal metadata description</td>
</tr>
<tr>
<td>ASD DENSE</td>
<td>1920</td>
<td>μ: 63 σ: 0</td>
<td>AFSR spectra with a rich metadata description</td>
</tr>
<tr>
<td>GONIO</td>
<td>3300</td>
<td>μ: 33.8 σ: 1.5</td>
<td>A real dual-view FGOS (Schopfer et al., 2008) goniometer campaign containing GER 3700 and ASD FSFR spectral data. Metadata include spatial position, illumination and sampling geometry, pictures, target type for all spectra, and reference panel spectrum links for GER target spectra.</td>
</tr>
</tbody>
</table>

Table 6
Spectral table sizes in relation to number of bands for the SPARSE test cases.

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Number of bands</th>
<th>Exported table size for the spectrum entity [bytes]</th>
<th>Ratio of table size to number of bands [bytes]</th>
</tr>
</thead>
<tbody>
<tr>
<td>GER</td>
<td>647</td>
<td>6405</td>
<td>9.9</td>
</tr>
<tr>
<td>ASD</td>
<td>2151</td>
<td>18,425</td>
<td>8.6</td>
</tr>
</tbody>
</table>

Fig. 4. Export speeds for ASD SPARSE (left) and GER SPARSE (right) test cases.
one used for the ASD DENSE test case. The export speed measurement reveals a combination of two effects: (a) dependency on the sensor, i.e., the number of spectral bands are having a direct impact on the memory footprint of the signatures and hence influence to export speed; and (b) complexity of the metadata space resulting in increased time needed for row-id lookups (Fig. 6).

In summary, the export speed is governed by (a) number of bands, (b) number of spectra, and (c) complexity of the metadata space. For sparse metadata spaces, the export speed is a constant function dominated by the number of bands. For dense metadata spaces, the export speed is similar to an inverse exponential function, largely controlled by the number of spectra and the complexity of the metadata space.

3.2. File sizes

The file size metric refers to the number of bytes required to store an exported campaign in an XML file. The sizes of the XML files for all test cases are listed in Table 7. For the sparse metadata space density tests (ASD SPARSE and GER SPARSE), file sizes are directly related to the size of the spectral data vector; i.e., the ASD SPARSE XML file is about 3 times bigger than the GER SPARSE file. An increase of metadata density of factor 4.8 (ASD SPARSE to ASD DENSE) results in only a marginal increase of file size (factor 1.14). This effect is due to storing data in their relational, i.e., least redundant, form. In fact, many of the metadata of the ASD DENSE test case are shared data; e.g., all spectra refer to the same photos documenting the sampling setup. These shared data are exported

<table>
<thead>
<tr>
<th>Test case</th>
<th>No. of spectra</th>
<th>MSD (mean)</th>
<th>File size [MB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASD SPARSE</td>
<td>1920</td>
<td>13</td>
<td>34.4</td>
</tr>
<tr>
<td>GER SPARSE</td>
<td>1920</td>
<td>14</td>
<td>12.2</td>
</tr>
<tr>
<td>ASD DENSE</td>
<td>1920</td>
<td>63</td>
<td>39.4</td>
</tr>
<tr>
<td>GONIO</td>
<td>3300</td>
<td>33.8</td>
<td>34.1</td>
</tr>
</tbody>
</table>
only once and linked to the respective spectra by relational information. Obviously, the relative metadata overhead per spectrum is dependent on the number spectra in the file. It is also worth noting that the optimisation of the file size by preserving the relational form causes an increase in time required for the export (cf. 3.1).

The storage of data in XML format significantly increases the data volume compared to binary formats. For example, the input data size of the ASD SPARSE case totals to 22.8 MB while the XML file takes 34.4 MB. Thus, the increase in storage size for ASD SPARSE is considerable at 33%.

3.3. Import speed

The import speed quantifies the time needed to import a campaign stored in an XML file into a target database. The test campaigns exported during the export speed tests were imported again into the same database for import speed tests. The results for the four test cases are shown in Fig. 7. Generally, the following may be observed:

(a) The number of inserted rows is not equal to the number of exported rows. This is due to system table entries already existing in the target system.
(b) The insert always starts after some delay. This is due to the system table entries being checked regarding their existence and not being inserted. The delay is dependent on the sensor type and on the number of referenced system parameters.
(c) The import speed is sensor dependent, i.e., varies with the length of the spectral data vector. The overall speed is fairly linear and thus independent of the number of imported rows.
(d) Import speeds are always dropping at the beginning of the import and then stabilising. The number of rows per second is highly varying. The causes of both effects are presumably linked to the performance of the database server.
(e) The RPS increases for higher metadata densities. This may seem counterintuitive but is caused by, on average, smaller row sizes of the tables holding metadata. Note that the total time needed for the import of higher MSDs is not decreasing. This can be observed from the ASD DENSE test case where the time for the import is more than double than the time needed for ASD SPARSE. The higher RPS effect is only apparent during import but not during export. The RPS difference between import/export of campaigns with high MSD is caused by the checks needed during these operations. Importing data is a much simpler process and the shorter processing times for metadata tables consequently prevail over the administration overhead. Exporting data is rather more complex and the overhead dominates the time needed for the processing of metadata tables.

4. Discussion

This paper set out to present a solution to the partial data exchange between spectral databases of identical schemata. The following sections discuss the fundamental nature of database...
structure extraction, the rules governing ordered table export, speed, data volume, and data storage issues, the usability of the MSD as quantitative quality score, and the exchange between databases of dissimilar schemata.

4.1. Database structure extraction and order table export

The elaborated solution for the partial database import/export relies heavily on detailed knowledge about the database structure. Database structure extraction from existing schemata thus represents a central component of the database import/export algorithm. The extraction is simplified by the a priori knowledge about the schema and therefore the implementation is not complicated by functionality needed to accommodate generic schemata (Premierlani and Blaha, 1994). The strict use of foreign keys during the database implementation greatly facilitates the extraction of associations. One might argue that extracting a structure already known to the designer of the system would be a needless overhead. However, it ensures that changes to the data model have no impact on the extraction function. Furthermore, the algorithm should be generic enough to be applicable to other databases as long as a number of database design rules were adhered to.

The ordered table export relies on a set of simple rules, most of which can be gleaned automatically from the schema. However, for a few system tables, these generic rules do not apply when resolving indirect references. Our analysis suggests that these rules are based on business knowledge and thus cannot be generalised. In fact, this rule definition process is the only nongeneric part of the export.

4.2. Data volume and import/export speeds

The results of the speed and data size tests carried out in this study suggest that neither presents a real bottleneck in terms of computing time or data volume. It would thus be feasible to carry out data exchanges regularly. However, the current implementation is targeted at one-time transfers. Extension to regular data exchanges would imply changes to both the import/export functionality and the data model. In an ideal case, only delta information would be transferred. Identifying the delta would require information about the changes carried out over time (Chawathe et al., 1998).

4.3. XML as data exchange file format

An XML-type file format has been chosen for the transportation of data between systems. XML is now one of the main standards for information exchange (Fong et al., 2003) and is especially suited for the storage of metadata (Houlding, 2001). The main advantages are the ability to represent tree structures and the self-descriptive data format. However, a major deficiency of XML is its space efficiency as it increases file sizes considerably (Lawrence, 2004). Our current implementation uses hexadecimal coding for binary data, but this is not the most optimal form of encoding in terms of resulting file sizes. The most common approach in use today is the BASE64 algorithm, which is also used to encode email attachments, commonly known as the MIME format (Brås et al., 2008; Freed and Borenstein, 1996). Therefore, future SPECCHIO software versions may implement the BASE64 algorithm instead.

Recently, XML has been identified as a possible solution for the general exchange of spectroradiometer data and metadata (Malthus and Hueni, 2009). The conversion of data stored in RDBMS into XML documents and vice versa has been the focus of intense research (Fong et al., 2003; Shanmugasundaram et al., 2000). These efforts have been caused by XML being an emergent standard for data exchange while having deficiencies regarding efficient searches when stored as a file (Florescu and Kossmann, 1999). The transformation from a relational model to a hierarchical XML structure involves a denormalisation (Fong et al., 2003), which usually leads to the introduction of redundancies and the increase of data sizes. The inverse transformation requires the building of a relational model based on hierarchical structures (Fong, 1992; Min et al., 2008; Shanmugasundaram et al., 2001). Storing XML data in RDBMS has been applied for the sake of superior search functionality offered by relational databases. In the case of our presented export approach, the step of denormalisation is avoided and the data are stored in their relational structure. While this proves to be an advantageous concept for the problem of data exchange between identical schemata, it is clear that the XML structure used does not accord to standard XML rules, where the information is stored in a, usually redundant, tree structure. Introducing the notion of keys in the XML DTD (Document Type Definition) would allow proper representation of relational structures in XML (Arenas and Libkin, 2004). A possibility to comply with XML standards would be to use the ID and IDREF attribute types. These allow the definition of links within XML documents and should permit the use of generic XML tools to create Document Object Models (DOM).

4.4. Metadata space density

The implementation of the partial database import/export was tested using a number of test cases described by the newly introduced metadata space density metric. It has proven to be useful to quantitatively describe the amount of information contained in the metadata of spectra. While being a simple measure in the context of metadata spaces, its determination in a relational storage model is more complex but could be implemented using a generic approach. It is of interest to note that automated import of spectroradiometer files already creates an MSD of around 13. The effective number depends on the content of the input file and the data structure applied before loading. Metrics such as MSD can act as a quality indicator and are of importance for the automated estimation of data quality. In the case of MSD, it is an indication of the amount of metadata being available to judge the sampling context of the spectrum and consequently deduce some notion of data quality. The use of a weighted MSD taking into account the importance of the different metadata parameters might provide more realistic estimations of metadata quality. Such weighting will need further research, along with the definition of a minimal, common parameter set for spectral metadata.

The import and export speed metrics have shown that the metadata space density has an impact on the total time and the number of rows per second. Exporting metadata spaces of high density increases both the amount of data and the time needed to retrieve the data in the relational schema, as the latter must include checks to avoid multiple exports. The drop in export speed versus the exported number of rows might be addressed by using lookup tables with indexing for faster checks.

4.5. Exchange between heterogeneous database systems

The presented solution is targeted at data exchanges between identical schemata. The more general case of data exchange between heterogeneous systems would require a mapping of parameters between differing schemata (Gottlob, 2005; Libkin
and Sirangelo, 2008). Such a mapping would be eased by the definition of a minimal metadata set common to all systems. The definition of a common, minimal metadata set would need a consolidation of existing field and laboratory measurement protocols and techniques. Additionally, community-specific parameter sets complementing the common dataset would have to be established to cater for the requirements of the various field spectrometry subcommunities (e.g., vegetation, soils, geology, etc.).

The mapping between different schemata requires explicit knowledge about the schemata. Consequently, the XML file structure would have to be extended by field type and key information or, alternatively, the relational structure could be contained in an additional file. As with the identical schemata case, a denormalisation should be avoided to prevent redundancies. In general, the complications incurred by the exchange between heterogeneous systems have been a topic of increasing interest over the past years (Fagin et al., 2005) and remain an area of active research. It is recommended to base implementations of data exchange between heterogeneous spectral databases on the extensive knowledge of data exchange available in computer sciences.

5. Conclusions

With the advent of spectral databases for the storage of spectroradiometer data and associated metadata, efficient methods for the exchange of data between storage systems are getting ever more important. This paper introduces the concepts required for the partial export of spectral data from a relational database while preserving the full relational metadata context and the mechanisms for the seamless import into a target system. The solution to the partial database import/export problem presented has been implemented in the SPECHIO Java application from version 2.0 onward and enables the easy transfer of spectral data between heterogeneous systems including the full metadata context is a problem yet to be solved and may utilise XML files for data exchange as well.

We propose that data stored in relational databases should be exported in their relational form whenever possible to avoid redundancies and minimise the data volume. One of the main challenges connected with exchange between heterogeneous systems will be the mapping of metadata parameters between the schemata involved. The definition of a common, minimal metadata set describing spectroradiometer measurements and supported by all spectral database systems would be an important step toward the exchange, long-term use, and quality assessment of spectral data collections.

References


