MODEL-DRIVEN APPLICATION DEVELOPMENT ENABLING INFORMATION INTEGRATION

Georgios Voulalas and Georgios Evangelidis
Department of Applied Informatics, University of Macedonia, 156 Egnatia St., Thessaloniki, Greece

Keywords: Interoperability, Enterprise information integration, Extract – transform - load, Application integration, Relational database, Schema matching, Java, Database metadata, Meta-model.

Abstract: Interoperability is the capability of different software systems to exchange data via a common set of exchange formats. Interoperability between two products is often developed post-facto, due to insufficient adherence to standardization during the software design process. In this paper we present a mechanism that enables the dynamic communication of different software systems at database level, based on the principles of the Enterprise Information Integration architectural framework. The mechanism is built on the top of a database schema (meta-model) and extends the framework we elaborate on for the dynamic development and deployment of web-based applications.

1 INTRODUCTION

Systems interoperability and online-data integration represent some of the most significant challenges facing the information technology community in the last years. Traditionally, data models are designed for specific applications without regard to integration. The semantics conveyed in those models often represent an informal agreement between the developer and the end-users in a task-specific environment. As a result, interoperability between two products is often developed post-facto.

In this paper we present a mechanism that enables dynamic communication of different software systems at database level, based on the Enterprise Information Integration (EII) architectural framework. The mechanism extends the framework that we have previously presented (Voulalas and Evangelidis, 2007), (Voulalas and Evangelidis, 2008a), (Voulalas and Evangelidis, 2008b), (Voulalas and Evangelidis, 2009a), (Voulalas and Evangelidis, 2009b) for the dynamic development and deployment of web-based applications.

For achieving data integration it is essential that the data sources are mapped efficiently. Previous research papers (Li and Clifton, 1994), (Madhavan and Bernstein, 2001), (Milo and Zohar, 1998), (Castano and De Antonellis, 2001) have proposed many techniques for automating the mapping operation for specific application domains. Taken as granted that the mapping operation cannot be supported in a fully automatic way and that the human intervention is always required, we are working on a mechanism that, given the schema mappings, releases the users from the technical difficulties of the integration and allows them to focus solely on the schema mapping process.

The remainder of the paper is structured as follows: a review of the related literature is given in Section 2. In Section 3 we introduce a running example that will facilitate us in presenting the interoperability problem in practice and applying the proposed data integration mechanism. In Section 4 we introduce the core elements of the proposed mechanism, and in Section 5 we demonstrate the way the new mechanism extends our previous work in the field of dynamic development and deployment of web-based applications. This extended version of our framework is essentially the final outcome of this paper. We conclude the paper with future steps.

2 RELATED WORK

A variety of architectural approaches is used for information integration (Bernstein and Haas, 2008). Schema matching is a common requirement in almost all architectural approaches.

Voulalas G. and Evangelidis G..
MODEL-DRIVEN APPLICATION DEVELOPMENT ENABLING INFORMATION INTEGRATION.
DOI: 10.5220/0003336401410146
In *Proceedings of the 7th International Conference on Web Information Systems and Technologies* (WEBIST-2011), pages 141-146
ISBN: 978-989-8425-51-5

2.1 Architectural Approaches for Integration

In this section we summarize the three most important architectural approaches.

2.1.1 Extract, Transform and Load (ETL)

Extraction, transformation, and loading (ETL) processes run in the background of a data warehouse architecture. An ETL process involves (Vassiliadis and Simitsis, 2009) extracting data from external sources, transforming them to fit operational requirements and loading them into the end target (database or data warehouse). In a typical data warehouse configuration, the ETL process periodically refreshes the data warehouse during idle or low-load periods of its operation.

2.1.2 Enterprise Information Integration (EII)

Enterprise Information Integration offers uniform access to different data sources, ranging from database systems and legacy systems to forms on the web, web services and flat files, without having to first load all the data into a central warehouse (Halevy and Ashish, 2005). In EII, a designer creates a mediated schema that covers the desired subject and maps the data source schemas to the mediated schema. The data sources are integrated without materializing the integrated view. Users pose queries to the mediated schema, and those queries are reformulated to separate queries to the different data sources. Data are transformed during query processing.

2.1.3 Enterprise Application Integration (EAI)

Enterprise Application Integration (EAI) is concerned with making applications that operate on heterogeneous platforms to work together seamlessly, by sharing data and functions, with the use of common middleware. EAI minimizes the dependencies between applications by using the design principle of loose coupling (Wong, 2009). This allows applications to evolve independently and if one application is modified or an exception occurs, impact to other applications is limited.

2.2 Schema Matching Techniques

Substantial effort in the area of data interoperability focuses on automatic schema matching techniques (Li and Clifton, 1994), (Madhavan and Bernstein, 2001), (Milo and Zohar, 1998), (Castano and De Antonellis, 2001). In (Bernstein, 2001) an interesting taxonomy is presented that covers many of the existing approaches. In general, it is not possible to determine fully automatically all matches between two schemas, primarily because most schemas have some semantics that affect the matching criteria but are not formally expressed or often even documented. Determining match candidates, which the user can accept, reject or change, and allowing the user to specify additional matches seems to be the most adequate approach.

3 RUNNING EXAMPLE

In this section, we present a real world data integration problem. We draw our running example from the real estate industry. Listing properties on as many portals as possible has become an essential part of real estate marketing. The problem is how to find the time to list them on multiple sites and keep them up to date (removing sold properties, modifying prices, etc.). Ideally, big real estate portals should provide mechanisms that allow the collection of structured data from small sites operated by individual agents. Suppose that Database A (dbA) that hypothetically belongs to a real estate portal is a normalized data model, while Database B (dbB) that hypothetically belongs to an individual real estate site is a simpler database schema. The following statements outline the common business rules of the two systems along with the differences in the way those business rules are modelled at database level:

1. In both databases there exists a table named Properties. For each property there exists a record in this table.

2. Three types of properties exist: ‘Residencies’, ‘Business Properties’, and ‘Land’. Properties are further categorized according to predefined property subtypes (flats, detached houses, offices, plots, etc.). For both types and subtypes a similar approach with a look-up table (PropertyTypes / PropertyCategories) has been followed in the two databases. Additionally, dbA includes a separate table for each property type connected with an identifying 1:1 relationship with the Properties table. So, the Properties table carries the type-independent fields of the property and the tables Residencies, Business Properties and Land carry the type-dependent fields. In dbB, all fields have been put in the Properties table.

3. Each property is located in a geographical area. Again, a similar approach with a look-up table (Locations / Areas) has been followed in both databases.

4. A property can be available for sale or rent, or for both sale and rent (transaction types). In dbA, there exists a look-up table named TransactionTypes. This table is associated with a M:N relationship with the Properties table that is implemented with the use of the Property_TransactionType table. In dbB, the data designer has used two nullable fields in the Properties table: priceForSale and priceForRent.

5. Photos of the property are optionally attached to the listing. In dbA, there exists a separate table (Images) associated with an 1:N non-identifying relationship with the Properties table. In dbB, each record in the Properties table carries itself five (nullable) photos.

4 THE PROPOSED MECHANISM

First we have first to decide on the type of integration we are looking for. Are we are looking for real-time integration? Yes, because it is very important for the Real Estate Portal to deliver up to date information (ads) to its end-users. Is there a need to integrate the two systems at functional level? No, because the two systems do not implement business processes that are somehow interrelated.

The EII approach seems to best fit to our case, since it offers real-time access to different data sources. Queries posed to the "mediated" database, i.e. the database of the Real Estate Portal (dbA), will be then transformed to queries to the source database, i.e. the database of the Real Estate Site (dbB). This requires the mapping of the schema fields.

4.1 Analyzing the Database Schemas

First, we have to decode the structure of the two databases, and analyze it in tables, columns and associations between the tables. Java provides a set of useful methods implemented by the java.sql.DatabaseMetadata class: method getTables retrieves the tables that are included in a given database schema, method getColumns retrieves the columns of a given table, method getPrimaryKeys retrieves the primary keys of a given table, method getImportedKeys retrieves all primary key columns that are imported by a given table) and method getExportedKeys retrieves all foreign key columns exported by a table. Those methods will enable us to identify the entities, their fields and the way the entities are associated (relationships). Additionally, by examining the data we will be able to define the cardinality of the relationships. This requires the implementation of a custom method that will be generic in order to work for every schema: the method will take as arguments the information returned by the java.sql.DatabaseMetadata methods (e.g. table name, primary key name, foreign key name) and by searching in the database will identify whether a record in a table can be related with 1 or more records in the associated table.

4.2 Matching the Database Schemas

After analyzing the structure of the two databases we have to map each field of the source database to a field of the target database. By utilizing one of the existing semi-automated schema mapping techniques and with some extra human intervention mapping could be successfully completed. Ideally, the user should be presented with all matching candidates and enabled through a friendly user interface to accept, reject or modify the suggestions, and define new mappings.

Additional human intervention is required for mapping the values in the Lookup tables. This is an issue that has not been addressed in the previous research efforts. Candidate mappings could be discovered automatically and the user should be able to manage them through a friendly user interface.

4.3 Transferring Data

Since the two databases have been designed in a completely different way, the mechanism that will retrieve the data from the source database and transfer them in the mediated database should address the following issues:

- Each property in dbB, associated with the property category 'Residencies', is equal to a record in the Properties table joined with a record in the Residencies table in dbA.
- If the Properties.priceForSale field in dbB is not NULL, this is equal with a joined record in the Property_TransactionType table in dbA.
- If any of the image1,…, image5 fields in the Properties table in dbB do not contain a NULL value, this is equal with a joined record in the Images table in dbA.
- Each value in the PropertyCategories look-up ta-

ble in dbB should be mapped to explicitly one value in the PropertyTypes look-up table in dbA.

- Each value in the Areas look-up table in dbB should be mapped to explicitly one value in the Locations look-up table in dbA.

The following SQL statement posed to the mediated table returns all data related to a residency:

```
SELECT P.PROPERTYID, P.CODE,
P.SIZE, R.NUMBER_OF_FLOORS,
R.FLOOR,R.NUMBER_OF_BEDROOMS,
R.NUMBER_OF_LIVING_ROOMS,
R.NUMBER_OF_KITCHENS, R.HEATING,
R.NUMBER_OF_BATHROOMS,
R.NUMBER_OF_WCS, L.NAME,
PT.NAME AS _TYPE,
PST.NAME AS SUBTYPE
FROM PROPERTIES P
LEFT JOIN RESIDENCIES R
ON P.PROPERTYID = R.RESIDENCYID
INNER JOIN LOCATIONS L
ON P.LOCATIONID = L.LOCATIONID
INNER JOIN PROPERTYTYPES PT
ON P.PROPERTYTYPEID =
PT.PROPERTYTYPEID
INNER JOIN PROPERTYTYPES PST ON
P.PROPERTYSUBTYPEID =
PST.PROPERTYTYPEID
WHERE P.PROPERTYID = ?
```

The statement should be transformed as follows in order to retrieve the required data from the source database:

```
SELECT P.PROPERTYID, P.CODE, P.SIZE,
P.FLOORS, P.FLOOR, P.BEDROOMS,
P.LIVINGROOMS, P.KITCHENS, P.HEATING
P.BATHROOMS, P.WCS, A.NAME,
PC.NAME AS _TYPE,
PSC.NAME AS SUBTYPE
FROM PROPERTIES P
INNER JOIN AREAS A ON
P.AREAID = A.AREAID
INNER JOIN PROPERTYCATEGORIES PC ON
P. PROPERTYCATEGORYID =
PC.PROPERTYCATEGORYID
INNER JOIN PROPERTYCATEGORIES PSC ON
P.PROPERTYSUBCATEGORYID =
PSC.PROPERTYCATEGORYID
WHERE PROPERTYID = ?
```

If we use all the information extracted during the two previous steps (i.e. by analyzing and mapping the two database schemas) then we will be able to manually transform the SQL queries posed to dbA to queries to be executed on dbB. However, the challenging point is to automate the whole process in a way that is completely independent of the involved schema details. We can achieve this by storing all information in a database model. This data model will be essentially a meta-model since it will model the structure of the application data models. An initial version of this meta-model is presented in Figure 1.

The meta-model includes the following entities:

- _Databases: One record in this table is created for the mediated database and one for each source database. The table holds connection information (server, port, username, password, etc.).
- Connections: For each connection established between two databases (mediated and source) a record in this table is created.
- Relationships: This table holds all relationships between the tables of a database schema.
- Entity_Relationship: For each entity participating in a relationship a record in this table should exist. The parent entity is flagged appropriately. The cardinality (1 or n) is also included. Optional participation is also declared (0..1 or 0..n).
- Columns: This table holds information about the columns of the tables.
- ColumnMappings: This table holds the column mappings between the two schemas (mediated and source).
- LookupValues: This table holds information about the values of the lookup tables. Note that for each value we store the integer identifier and the value in a string field (the real data type can be derived from the association with the Columns table).
- LookupValueMappings: This table holds the lookup value mappings between the two schemas (mediated and source).

Having modelled the structure of both databases and the way they are mapped we can create generic methods that transform the SQL queries posed to the mediated schema to queries to be executed on the source schemas. The methods will work independently of the database schemas since they will be based on the meta-model that is common for all databases. Handling all complex mapping cases presented above and many others that may exist in real databases requires complex business logic. The human intervention should then be solely focused on the schema mapping task. Note that the proposed mechanism should be incorporated by any database that communicates with many, diverse databases. In our example the Real Estate Portal, by utilizing the mechanism, will be able to communicate with any individual Real Estate site.

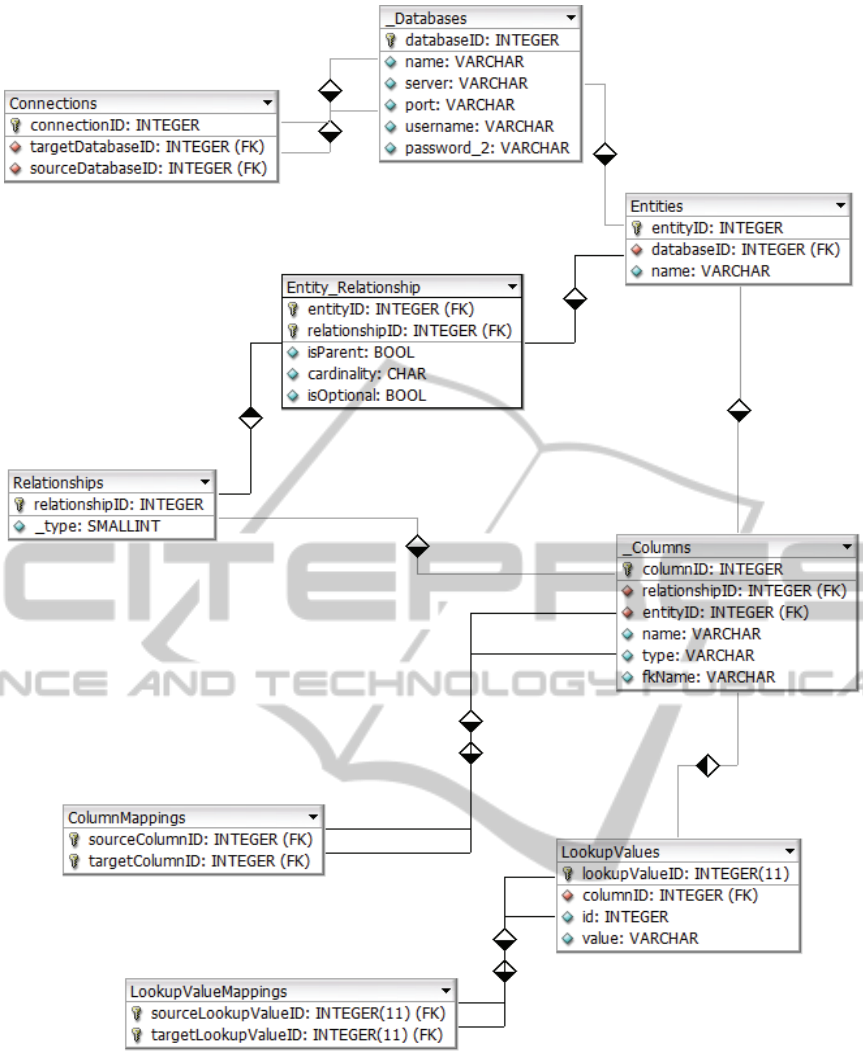

Figure 1: The proposed Meta-Model.

5 EXTENDING OUR PREVIOUS WORK

In (Voulalas and Evangelidis, 2007), (Voulalas and Evangelidis, 2008a), (Voulalas and Evangelidis, 2008b), (Voulalas and Evangelidis, 2009a), (Voulalas and Evangelidis, 2009b) we elaborated on a development and deployment framework that targets to web-based business applications and supports runtime adaptations. The framework hosts multiple applications within a single installation. The running application constitutes of generic components and application-specific components that are produced by the runtime compilation of the application-specific source code. There always exists one deployed application, independently of the actual number of running applications. The framework is structured upon a universal database model (meta-model), divided into three regions.

- Region A holds the functional specifications of the modeled application and includes the following entities: Classes, Attributes, Methods, Arguments, Associations and Imports (class dependencies).
- For each table of Region A, a companion table using "_versions" as suffix is included in Region A´. This enables us to keep all different versions of the modeled applications.
- Region B holds data produced by the applications and consists of the following tables: Objects, AssociationInstances and AttributeValues. Those tables are structured independently of the actual data structure of the applications. Thus,

changing the database structure of a modeled application (e.g. adding a new field in an existing table or creating a new table) does not affect Region B.

It is clear that the database model presented in Figure 1 and the Region A of this database, model the same concepts under different naming convetions. In the first, the naming conventions have been derived from the OO terminology (Entities, Attributes and Associations), while in the latter the naming conventions used are taken from the ER terminology (Tables, Columns and Relationships). Thus, we can easily incorporate the mechanism introduced in the previous section to our framework in order to cover data integration issues between applications developed and deployed applications within our framework or between an application developed and deployed within our framework and an application developed and deployed externally.

6 FURTHER RESEARCH

Information integration is a 'fragile' process, since modifying the structure of the involved data sources requires integration redesign (Bernstein and Haas, 2008). Although the problem of schema evolution has received much research attention, the way it influences the integration process is not adequately addressed. Our framework supports runtime evolution of the applications that are developed and deployed on it by retrieving the data and functional specifications from the database. Since the operation of the proposed integration process is based on the same concept, the extended framework could support at runtime changes in the integration process that are caused by changes in the data structure of the integrated schemas, at least for the applications that are developed and deployed on it.

REFERENCES

Bernstein, P., Haas, L., 2008. *Information integration in the enterprise.* Commun. ACM.

Castano, S., De Antonellis V., De Capitani diVemercati, S., 2001. Global Viewing of Heterogeneous Data Sources. In J. IEEE Trans. Knowl. Data Eng.

Halevy, A., 2009. *Information Integration.* Encyclopedia of Database Systems.

Halevy, A., Ashish, N., Bitton, D., Carey, M., Draper, D., Pollock, J., Rosenthal, A., Sikka, V., 2005. Enterprise information integration: successes, challenges and controversies. In *SIGMOD'05, 24th International Conference on Management of Data / Principles of Database Systems.* ACM.

Li, W., Clifton, C., 1994. Semantic integration in heterogeneous databases using neural networks. In *VLDB '94, 20th International Conference on Very Large Data Bases*. Morgan Kaufmann.

Madhavan, J., Bernstein, P., Rahm, E., 2001. Generic schema matching with Cupid. In *VLDB '01, 27th International Conference on Very Large Data Bases.* Morgan Kaufmann.

Milo, T., Zohar, S., 1998. Using schema matching to simplify heterogeneous data translation. In *VLDB '01, 24th International Conference on Very Large Data Bases.* Morgan Kaufmann.

Rahm, E., Bernstein, P., 2001. A survey of approaches to automatic schema matching. *In the VLDB Journal, Volume 10.*

Scotney, B., McClean, S., Zhang, S., 2006. Interoperability and Integration of Independent Heterogeneous Distributes Databases over the Internet. In *Lecture Notes In Computer Science 2006, Volume 4042*. Springer, Heidelberg

Wong, J., 2009. Enterprise Application Integration. In *Encyclopedia of Database Systems.*

Vassiliadis, P., Simitsis, A., 2009. Extraction, Transformation, and Loading. In *Encyclopedia of Database Systems.*

Voulalas, G. and Evangelidis, G., 2007. A framework for the development and deployment of evolving applications: The domain model. In *ICSOFT '07, 2nd International Conference on Software and Data Technologies.* INSTICC Press.

Voulalas, G., Evangelidis, G., 2008(a). Introducing a Change-Resistant Framework for the Development and Deployment of Evolving Applications. In *Communications in Computer and Information Science, 1, Volume 10.* Springer Berlin Heidelberg.

Voulalas, G., Evangelidis, G., 2008(b). Developing and deploying dynamic applications: An architectural prototype. In *ICSOFT '08, 3rd International Conference on Software and Data Technologies.* INSTICC Press.

Voulalas, G., Evangelidis, G., 2009(a). Evaluating a Framework for the Development and Deployment of Evolving Applications as a Software Maintenance Tool. In *ICSOFT '09, 4th International Conference on Software and Data Technologie.* INSTICC Press.

Voulalas, G., Evangelidis, G., 2009(b). Application Versioning, Selective Class Recompilation and Management of Active Instances in a Framework for Dynamic Applications In *WEBIST '09, 5th International Conference on Web Information Systems and Technologies.* INSTICC Press.