Universal Symbolic Translator for Procedural Language over SQL

CALIN-ADRIAN COMES¹, LUCIAN-DOREL SAVU²
IOAN OVIDIU SPATACEAN¹, BEATRICE STEFAN¹, and ANCUTA AVRÁM¹
Petru Maior University¹, Nicolae Iorga 1, 540088 Tg-Mureș
Dimitrie Cantemir University², Bodoni Sandor 3-5, 540055 Tg-Mureș
ROMANIA

Abstract: Schema mapping represent the specification that describes how data structured from source schema S will be transformed into data structured on target schema T. Data exchange represent the process that transfers data structured from one schema S the source schema into data structured under schema T, the target schema. Schemata S and T without triggers and/or stored procedures(functions and procedures) are statical. We propose a Universal Symbolic Schema Translator graphical specification that describes the conversion of a Schema Model from Platform-Independent Model to Platform-Specific Model, Universal Symbolic Data Translator graphical specification that describes the transfer of a Data from Platform-Independent Model to Platform-Specific Model, Universal Symbolic Data Translator graphical specifica-

1 Introduction

Etymological according to [17], the grapheme for word data (with the phonemes \’dæ-ta\’; \’də-tə\’) represent the plural form for the Latin singular dat\’um (with the phonemes \’dæt-am\’; \’dət-am\’), which means something given, in different languages we have Czech: údaje, informace; Danish: data, materia; Dutch: gegevens, data; Estonian: andmed; Finnish: tieto, tiedot; French: données; German: Daten(plural); Greek: στοιχεία, δεδομένα; Icelandic: staðreynir, upplýsingar, gögn; Indonesian: data; Italian: dati Italian: faktus, datī, informācijas; Lithuanian: duomenys; Norwegian: data, informasjon, materiale; Polish: dane; Portuguese (Brazil): dados; Portuguese(Portugal): dados; Romanian: date(plural); Slovak: údaje, pokyny; Slovenian: podatki; Spanish: datos; Swedish: data; Turkish: veri, bilgi; for non-latin graphemes see Figure 1 from [20]. The grapheme for word schema [16] comes from the Greek word σχήμα(with the phoneme \’skĕ-ma\’), which means plan, shape or project, I cite “design several thinks are brought in one view”; the Greek plural is σχήματα (with the phoneme \’skĕ-ma-ta\’) from one point of view and from the Latin word schema, a diagrammatic representation, an outline or a model [18] in different languages we have Czech: plán, projekt; Danish: plan; Dutch: plan; Estonian: kava; Finnish: suunnitelma; French: plan; German: das Schema; Hungarian: terv(ezet); Icelandic: aætlun; Indonesian: rencana; Italian: piano, progetto; Latvian: plāns, projekts; Lithuanian: planas, projektas, būdas; Norwegian: plan, ordning, system; Polish: system, plan, projekt; Portuguese (Brazil): projeto; Portuguese (Portugal): esquema; Romanian: plan, schemă; Slovak: plán, projekt; Slovenian: načrt; Spanish: plan, programa, proyecto; Swedish: plan, förslag; Turkish: plân; for non-latin graphemes see Figure 2 according to [20]. Data exchange are used in many tasks in theoretical studies research [26], [11, 12, 13, 14, 15], [21], [23], [24], and Commercial Off The Shelf Tools - COST: SQL Assist ©from IBM DB2, ER/Studio ©from Embarcadero Technologies, Power Builder ©from Sybase Inc, OracleSQL and SQLJ ©from Oracle Inc., and MySQL Migration Toolkit ©from MySQL AB . We present a new graphical framework that translate procedural languages between different RDBMS’s flavors.

2 Related work

Data exchange are used in many tasks in theoretical studies research and practical in software products. In early stage 1977, in (Shu1977) with their EXPRESS, data exchange system with main functionality conversion data between hierarchical schemata the data exchange was in the top research topics.
UPDATE DEPT
SET EmpN = EmpN + 1
WHERE
EMPLOYEE.EmployeeID=EN
END;

CREATE TRIGGER NEWHIRED
AFTER INSERT ON EMPLOYEE
BEGIN
UPDATE DEPT
SET EmpN = EmpN + 1
WHERE
EMPLOYEE.EmployeeID=EN
END;

A translator operates in the following phases:

1. **Lexical analysis**: This phase involves parsing the input program into tokens, which are meaningful words or symbols.
2. **Syntax analysis**: This phase involves checking the grammatical correctness of the input program. It checks whether the tokens form a valid sentence according to the grammar of the language.
3. **Semantic analysis**: This phase involves checking the meaning of the input program. It ensures that the program is semantically well-formed and makes decisions about the program’s behavior.
4. **Code generation**: This phase involves converting the semantic representation of the input program into executable code in a target language.

A context-free grammar is 4-tuple:

\[ G = (V, \Sigma, R, S) \]

where:

- \( G \) represents a finite set of non-terminal characters or variables;
- \( \Sigma \) represents a set of terminals, disjoint with \( V \);
• **R** - represents a finite set of **rules**.

• **S** - represents the start variable, used to represent the or program;

**Definition 2.2.2** Let $\sum_1$ and $\sum_2$ be two alphabets, named source alphabet respective target alphabet and two languages $L_1 \subset \sum_1^*$, $L_2 \subset \sum_2^*$. A **translator** from the language $L_1$ to the language $L_2$ is a relation $T$ from $\sum_1^*$ to $\sum_2^*$ when the domain of $T$ is $L_1$ and the image of $T$ is $L_2$.

$$T : \sum_1^* \rightarrow \sum_2^*$$

where

$$\text{dom}(T) = L_1 \text{ and } \text{img}(T) = L_2$$

![Figure 4: Phases of a translator](image)

In [24] Pranevicius H. present an approach in idea to use Z specification language for development aggregate formal specifications, because the use of Z schemata in aggregate model permits mathematically strictly define **data structures** used in system description. Pranevicius define a abstract data type with the following requirements:

• All operation, which are possible to perform with data of that type have to be defined in data type definition;

• User of abstract data type can operate with values of that type by using only the defined type operation.

The formal specification approach using both aggregate approach an Z specification language are useful for specification the dynamical behavior of distributed information system and the large and global relational database systems.

In [5] Andreica et al. show that the categorial theory developed in symbolic computation can offer tools for systematically tackling the basis of fairly complex problems which rise in software design. They proposed a model who aims at proving the consistency of such transformations, which are often used in software applications that process databases; a symbolic model for the transformations between the relational database form and its XML representation.

According to the relational database definition given in [3] by Abiteboul et al., Andreica et al. extend for the domain of databases in [5].

MOF-QVT (Meta Object Facility - Queries / Views / Transformations) represents a standard for model transformation defined by the Object Management Group. QVT [22] define three **Domain Specific Language**: two of them are declarative Relation(has graphical concrete syntax) and Core and Operational Mappings(imperative language) that extends both QVT/Relations and QVT/Core. On the other hand QVT language conformance is specified along two dimensions: the **language dimension** and the **interoperability dimension**. By the convention we have:

QVT-language level-interoperability level

![Figure 5: QVT components](image)

Immanuel Kant in [19] said that: “The schemata, therefore, are nothing but a priori determinations of time according to rules, and these, in regard to all possible objects, following the arrangement of the categories, relate to the series in time, the content in time, the order in time, and finally, to the complex or totality in time.”

### 3 Our Approach

Our algebraical approach to data exchange and schema mapping is to include the stored procedures in **schema mappings** and to snapshot the dynamical of the schemata content in time extending [11, 12, 13, 14, 14, 21], because they parse the **statical schema mapping** not a **dynamical schema mapping**. We propose the source schema $S(t) = \langle S_1(t), S_2(t), \ldots, S_n(t) \rangle$, where $S_i(t)$’s are the source relation symbols, the target schema $T(t) = \langle T_1(t), T_2(t), \ldots, T_m(t) \rangle$, where $T_i(t)$’s are the target relation symbols and the schema $\langle S(t), T(t) \rangle = \langle S_1(t), S_2(t), \ldots, S_n(t), T_1(t), T_2(t), \ldots, T_m(t) \rangle$.

All instances over the $S(t)$ represent source instances $l(t)$, while instances over $T(t)$ $j(t)$ are target instances. If $l(t)$ is a named source instance in $S(t)$ and $j(t)$ is a named target instance the $K = \langle l, j \rangle$ is the
named instance over the schema \((S(t), T(t))\). A dependency named source-to-target dependencies over \((S(t), T(t))\) of the form
\[
(\forall x(t))(\phi_S(t)(x(t)) \rightarrow \chi_T(t)(x(t)))
\]
where \(\phi_S(t)(x(t))\) is an expression(formula), with free variable \(x(t) = (x_1(t), x_2(t), \ldots, x_k(t))\) of logical formalism over \(S(t)\) and \(\chi_T(t)(x(t))\) is an expression(formula) with free variable \(x(t) = (x_1(t), x_2(t), \ldots, x_l(t))\) of logical formalism over \(T(t)\). A dependency named target dependencies over the target schema \(T(t)\) (the target dependencies are different from those use for the source-to-target dependencies).

**Definition 3.1** A data exchange represent a 4-tuple \(DE(t) = (S(t), T(t), \sum st(t), \sum t(t))\) with a source schema \(S(t)\), a target schema \(T(t)\), a set \(\sum st(t)\) of source-to-target dependencies and set \(\sum t(t)\) of target dependencies.

For practical purposes each source-to-target dependency \(\sum st(t)\) represents a tuple-generating-dependency(tgd) of the form
\[
(\forall x(t))(\phi_S(t)(x(t)) \rightarrow \chi_T(t)(x(t), y(t)))
\]
where \(\phi_S(t)(x(t))\) represents a conjunction of atomic expression(formulas) over \(S(t)\) and \(\chi_T(t)(x(t), y(t))\) represents a conjunction of atomic expression(formulas) over \(T(t)\). A stored procedure named stored-procedure-s over \(S(t)\), of the form
\[
(\forall x(t))(\alpha_S(t)(x(t)) \rightarrow \alpha_S(t)(x(t)))
\]
where \(\alpha_S(t)(x(t))\) is a stored procedure over \(S(t)\) and a stored procedure named stored-procedure-t over \(T(t)\), of the form
\[
(\forall x(t))(\beta_S(t)(x(t)) \rightarrow \beta_S(t)(x(t)))
\]
where \(\beta_S(t)(x(t))\) is a stored procedure over \(T(t)\).

**Definition 3.2** A data exchange metamodel represent a 6-tuple \(DE(t) = (S(t), \sum \alpha_S(t), T(t), \sum \beta_T(t), \sum st(t), \sum t(t))\) with a source schema \(S(t)\), all stored procedures over \(S(t)\) \(\sum \alpha_S(t)\), a target schema \(T(t)\), all stored procedures over \(T(t)\) \(\sum \beta_T(t)\), a set \(\sum st(t)\) of source-to-target dependencies and set \(\sum t(t)\) of target dependencies.

For practical purposes each source-to-target dependency \(\sum st(t)\) represents a tuple-generating-dependency(tgd) of the form
\[
(\forall x(t))(\phi_S(t)(x(t)) \rightarrow \chi_T(t)(x(t), y(t)))
\]

where \(\phi_S(t)(x(t))\) represents a conjunction of atomic expression(formulas) over \(S(t)\) and \(\chi_T(t)(x(t), y(t))\) represents a conjunction of atomic expression(formulas) over \(T(t)\).

Our approach on symbolic modeling of data exchange and schema mapping are:

**Definition 3.3**

\[
DB(t) := \bigcup \{db(t)|is - database(db(t))\}
\]

where \(db(t)\) is a database.

Given a set of attributes \(Attr(t)\) and a set containing sets of attribute values \(D(t)\), we define a column as a function mapping an attribute into the set containing its corresponding values:

\[
ValColumn(t) : Attr(t) \rightarrow D(t),
\]

\[
ValColumn(a(t)) := \{d(t)|d(t) \in D(t)\}
\]

where \(d(t)\) is a value for attribute \(a(t)\).

**Definition 3.4**

Given a set of attributes \(Attr_i(t), i = 1, \ldots, n\) the table \(T(t)\) from database is defined by:

\[
is - Table(T_n(t), Attr_i(t), i=1,\ldots,n, D_i(t), i=1,\ldots,n) \iff T(t) \in \bigcup_{i=1}^{n} \{Attr(t), ValColumn(Attr_i(t))\}
\]

, where \(Card(ValColumn(Attr_i(t))) = nrw(t) = NoRows(T(t))\)

the number of lines in table \(T(t)\), \(i = 1, \ldots, n\), \(n(t) = NoCol(T(t))\) the number of columns in the table \(T(t)\).

In practice is possible to have \(S=T\) but \(S(t) \neq T(t)\) that case is named by us data exchange for copy schema mapping because all stored procedures over \(S(t)\) \(\sum \alpha_S(t)\), and all stored procedures over \(T(t)\) \(\sum \beta_T(t)\) have the same semantic but different syntax in SQL and Procedural Languages / SQL on different RDBMS.

We consider the following sub diagram with schema \(S=(EMPLOYEE, DEPARTMENT)\) with EMPLOYEE (#EmpoyeeID, FName, LName, CompanyID), DEPT (#DeptID, EmpN, Location) see the Database Diagram for schema \(S\), see Figure 7. In our case \(S=T=(EMPLOYEE, DEPARTMENT)\). A trigger that increments the number of employees each time a new person is hired, that is, each time a new row is inserted into the table EMPLOYEE has the same semantic in \(S\) and \(T\) but different syntax in different Procedural Language over different SQL flavors.
Table 1: The triggers when a new person is hired.

<table>
<thead>
<tr>
<th>RDBMS</th>
<th>STORED PROCEDURES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM DB2</td>
<td>CREATE TRIGGER NEWHIRED AFTER INSERT ON EMPLOYEE FOR EACH ROW MODE DB2SQL UPDATE DEPT SET EmpN = EmpN + 1</td>
</tr>
<tr>
<td>Oracle</td>
<td>CREATE TRIGGER NEWHIRED AFTER INSERT ON EMPLOYEE BEGIN UPDATE DEPT SET EmpN = EmpN + 1 WHERE EmployeeID=:New.EmployeeID END;</td>
</tr>
<tr>
<td>Sybase</td>
<td>CREATE TRIGGER &quot;NEWHIRED&quot; AFTER INSERT OF EmployeeID ON EMPLOYEE REFERENCING OLD AS EO NEW AS EN FOR EACH ROW BEGIN UPDATE DEPT SET DEPT.EmpN = DEPT.EmpN + 1 WHERE EMPLOYEE.EmployeeID=EN END</td>
</tr>
<tr>
<td>MySQL</td>
<td>CREATE TRIGGER NEWHIRED AFTER INSERT ON EMPLOYEE FOR EACH ROW UPDATE DEPT SET EmpN = EmpN + 1</td>
</tr>
<tr>
<td>Postgres</td>
<td>CREATE FUNCTION EmpA() BEGIN UPDATE FIRMA SET EmpN = EmpN + 1; END; LANGUAGE 'plpgsql' VOLATILE; CREATE TRIGGER NEWHIRED AFTER INSERT ON EMPLOYEE FOR EACH ROW EXECUTE PROCEDURE EmpA();</td>
</tr>
</tbody>
</table>

Figure 6: ER Diagram

Definition 3.5
We define a universal repository U, where S is the source schema, T the target schema, and X represent the XML schema and the following artifacts: ⊞, □, △, ⊙, ∇, ⊕, ⊖, ⊗, ⊠, ⊡ with the following quadratical functionality:

\[ U \xrightarrow{\text{repository}} X \]

\[ S \xrightarrow{\text{repository}} T \]

In Table 2 we show a sample:

Table 2: The triggers with repository.

<table>
<thead>
<tr>
<th>RDBMS</th>
<th>STORED PROCEDURES</th>
<th>REPOSITORY</th>
</tr>
</thead>
</table>
| DB2    | CREATE TRIGGER NEWHIRED AFTER INSERT ON EMPLOYEE FOR EACH ROW MODE DB2SQL UPDATE DEPT SET EmpN=EmpN+1 | EMPLOYEE ⊆ NEWHIRED ⊗ EMPLOYEE ⊕ EMPLOYEE ⊗ ⊕ EMPLOYEE ⊕ EMPLOYEE ⊗ ⊕ EMPLOYEE ⊕ EMPLOYEE ⊗ ⊕ EMPLOYEE ⊕ EMPLOYEE ⊗ ⊕ EMPLOYEE ⊕ EMPLOYEE ⊗ ⊕ EMPLOYEE ⊕ EMPLOYEE ⊗ ⊕ EMPLOYEE ⊕ EMPLOYEE ⊗ ⊕ EMPLOYEE ⊕ EMPLOYEE ⊗ ⊕ EMPLOYEE ⊕ EMPLOYEE ⊗ ⊕ EMPLOYEE ⊕ EMPLOYEE ⊗ ⊕ EMPLOYEE ⊕ EMPLOYEE ⊗ ⊕ EMPLOYEE ⊕ EMPLOYEE ⊗ ⊕ EMPLOYEE ⊕ EMPLOYEE ⊗ ⊕ EMPLOYEE ⊕ EMPLOYEE ⊗ ⊕ EMPLOYEE ⊕ EMPLOYEE ⊗ SUBTRACT  

4 Conclusion

A Universal Symbolic Schema Translator graphical specification is proposed that describes the conversion of a Schema Model from Platform-Independent Model to Platform-Specific Model, Universal Symbolic Data Translator graphical specification that
describes the transfer of a Data from Platform-Independent Model to Platform-Specific Model.

Acknowledgments: The research was supported by the Petru Maior University of Tg-Mureş, Faculty of Economics, Law, and Administrative Science.

References:


