Co-Evolution of Databases and Programs

Anthony Cleve
IPA Fall Days 2008
Nunspeet, The Netherlands
Plan

- Introduction & Motivation

- Part I: Database-programs co-Analysis
  - Database Reverse Engineering

- Part II: Database-programs co-Evolution
  - Database Schema Change
  - Database Migration

- Tool support

- Industrial project

- Conclusions and perspectives
Plan

- Introduction & Motivation
  - Part I: Database-programs co-Analysis
    - Database Reverse Engineering
  - Part II: Database-programs co-Evolution
    - Database Schema Change
    - Database Migration
- Tool support
- Industrial project
- Conclusions and perspectives
Introduction & Motivation

- Information systems (IS) made of
  - inter-dependent artefacts
  - of different nature
  - belonging to different levels of abstraction

- IS are subject of continuous evolution
  - due to ever-changing business requirements
  - complex and expensive process

- Our focus
  - evolution of *data-intensive* systems
  - comprising a *database* and associated *programs*
  - that must *co-evolve* such that global *consistency* is preserved
Introduction & Motivation (Part I)

- The initial phase of database evolution is (should be/has to be)
  - Database reverse engineering

- Database Reverse Engineering (DRE)
  = “a collection of methods and tools to help an organization determine the structure, function, and meaning of its data” [Chikofsky1996]

- DRE aims at retrieving implicit schema constructs:
  = data structures and constraints that are **not declared** in the DDL code but used/managed by the application programs

Examples:
  - finer-grained structure of entity types and attributes;
  - referential constraints (foreign keys);
  - exact cardinalities of attributes;
  - identifiers of multi-valued attributes.

**Program analysis techniques are needed**
Introduction & Motivation (Part II)

- Several kinds of database evolutions
  - Database schema change (refactoring or semantic evolution)
  - Database migration
  - Database integration

- Each kind of evolution
  - may have a huge impact on existing application programs

- Database-programs consistency relationships
  - Structural consistency: database schema
  - Language consistency: data manipulation language
  - Semantic consistency: database *implicit* constraints

**Program conversion techniques are needed**
What does consistency mean to data-intensive applications?
What does consistency mean to data-intensive applications?

Diagram:
- DML syntax
- Data model
- DDL syntax
- Conceptual schema
- Logical schema
- Physical schema
- DDL code
- Data instances
- Programs
- Source code
- Execution

Consistency relationship
## Nature of consistency relationships

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Logical Schema VS Conceptual schema</th>
<th>Logical Schema VS Data Model</th>
<th>Logical Schema VS Data Model</th>
<th>Program source code VS DML syntax</th>
<th>Program source code VS Logical schema</th>
<th>Program execution VS Data instances</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>➢ Semantic compatibility (↔ equivalence)</td>
<td>➢ Meta-model compliance</td>
<td></td>
<td>➢ Syntactical compliance (DB queries)</td>
<td>➢ Structural compliance of DB queries</td>
<td>➢ Behavioural compatibility</td>
</tr>
<tr>
<td></td>
<td>➢ Potential semantic gap during logical design</td>
<td></td>
<td></td>
<td></td>
<td>➢ Type compatibility (input/output host variables)</td>
<td>➢ Due to the potential gap between CS and LS</td>
</tr>
<tr>
<td>Logical Schema VS Data Model</td>
<td>➢ Meta-model compliance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>➢ Programs might introduce inconsistent data instances</td>
</tr>
<tr>
<td>Physical Schema VS Logical schema</td>
<td>➢ Semantic compatibility/equivalence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDL code VS Physical schema</td>
<td>➢ High-fidelity translation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDL code VS DDL syntax</td>
<td>➢ Syntactical compliance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data instances VS DDL code</td>
<td>➢ Format compliance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Plan

- Introduction & Motivation

- **Part I : Database-programs co-Analysis**
  - Database Reverse Engineering

- Part II : Database-programs co-Evolution
  - Database Schema Change
  - Database Migration

- Tool support

- Industrial project

- Conclusions and perspectives
**Database-Programs co-Analysis**

- **Principles**
  - Program analysis may be used to better understand/enrich the DB schema
  - Data analysis may help us to confirm/reject/discover hypotheses on schema structures and constraints, and to detect corresponding data inconsistencies
  - Schema analysis may help us to qualify discovered data dependencies
  - Re-analyzing programs against violated implicit constraints may help us to identify the possible causes of data inconsistencies

⇒ Database reverse engineering typically is an iterative, semi-automatic process, combining schema, program and data analysis techniques
General Approach

- Step 1: Retrieving implicit schema constructs
  - use of program analysis techniques
    - static analysis
    - dynamic analysis

- Step 2: Detecting data inconsistencies
  - data analysis process based on recovered implicit constructs

- Step 3: Improving data validation
  - at both database and program sides
  - based on violated implicit constructs
Step 1: Retrieving Implicit Constructs

- Static program analysis
  - based on system dependency graphs (SDG’s)
  - introduced by [HorwitzEtAl90]

- Its nodes represent
  - assignment statements, control predicates, procedure calls and parameters passed to and from procedures

- Its arcs represent dependencies among program components
  - A control dependency arc from node $v_1$ to node $v_2$
    - $v_2$ can be executed/evaluated only if $v_1$ has been executed/evaluated
  - A data dependency arc from node $v_1$ to node $v_2$
    - the state of objects used in $v_2$ can be affected by the evaluation of $v_1$
A sample program

begin
  S1: \texttt{read}(A);
  S2: \texttt{if}(A < 0)
      \texttt{then}
      S3: \texttt{B} := \texttt{f}_1(A);
      S4: \texttt{C} := \texttt{g}_1(A);
      \texttt{else}
      S5: \texttt{if}(A = 0)
          \texttt{then}
          S6: \texttt{B} := \texttt{f}_2(A);
          S7: \texttt{C} := \texttt{g}_2(A);
          \texttt{else}
          S8: \texttt{B} := \texttt{f}_3(A);
          S9: \texttt{C} := \texttt{g}_3(A);
      \texttt{endif};
  \texttt{endif};
  S10 \texttt{write}(B);
  S11 \texttt{write}(C);
end.

Corresponding dependency graph

* adapted from [Agrawal1990]
Program analysis based on SDG’s

- Many ways to query an SDG...
  - Program slicing
    - introduced by [Weiser84]
    - the *slice of a program* w.r.t. program point \( p \) and variable \( x \) consists of all the program statements and predicates that might affect the value of \( x \) at point \( p \)
    - used for debugging, reengineering
  - Dataflow analysis
    - useful in data reverse engineering
    - example: a data dependency between columns of distinct tables may suggest the existence of implicit foreign key...

Program slice for line S10, B

```plaintext
begin
S1:   read(A);
S2:   if(A < 0)
      then
S3    B := f_1(A);
S4    C := g_1(A);
      else
S5    if(A = 0)
      then
S6    B := f_2(A);
S7    C := g_2(A);
      else
S8    B := f_3(A);
S9    C := g_3(A);
      endif;
      endif;
S10   write(B);
S11   write(C);
end.
```
Intra-query dataflow analysis

```sql
select substring(Address from 61 for 30)
into :CITY
from CUSTOMER C, ORDERS O
where C.Num = O.Sender
and O.Num = :ORDID
```

<table>
<thead>
<tr>
<th>CUSTOMER</th>
<th>ORDERS</th>
<th>CUSTOMER</th>
<th>ORDERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: varchar (30)</td>
<td>Date: date (1)</td>
<td>Name: varchar (30)</td>
<td>Date: date (1)</td>
</tr>
<tr>
<td>Address: char (120)</td>
<td>Reference: char (12)</td>
<td>Address: compound (120)</td>
<td>Reference: char (12)</td>
</tr>
<tr>
<td>id: Num</td>
<td>Sender: num (8)</td>
<td>Data1: char (60)</td>
<td>Sender: char (12)</td>
</tr>
<tr>
<td></td>
<td>id: Num</td>
<td>City: char (30)</td>
<td>id: Num</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data2: char (30)</td>
<td>ref: Sender</td>
</tr>
</tbody>
</table>

dependency analysis

ORDERS.Sender

CUSTOMER.Num

CUSTOMER.Address (61, 30)

CITY

schema enrichment
exec sql
  select Sender
  into :CUST
  from ORDERS where Num = :ORDID
end-exec

exec sql
  select Name, substring(Address from 61 for 30)
  into :CNAME, :CITY
  from CUSTOMER where C.Num = :CUST
end-exec

ORDERS.Sender
  ↘
    └── CUSTOMER.Num

CUSTOMER.Address(61,30)
  ↘
    └── CITY

Schema enrichment
Building SDG’s in the presence of DB operations

• In the context of DRE
  • SDG’s are very useful results
  • used to derive implicit database structures and constraints …
  • … from SDG data dependencies

• But… in the presence of database operations
  • SDG potentially incomplete/imprecise
    • some data dependencies may be missed (silence)
    • some detected data dependencies may be uncorrect (noise)

• Cause of the problem
  • the interaction between program variables and database entity types and attributes must be taken into account
  • this interaction depends on the data manipulation language (DML)
**DML Categories**

- **Native DML** e.g. COBOL file handling primitives in COBOL programs
- **Built-in DML** e.g. CODASYL primitives in COBOL programs
- **Embedded DML** e.g. embedded SQL or IMS in COBOL programs
- **Call-based DML** when a dedicated program is invoked to access the DB

<table>
<thead>
<tr>
<th>Native</th>
<th>Built-in</th>
<th>Embedded</th>
<th>Call-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE 7 TO CUS-ID</td>
<td>MOVE 7 TO CUS-ID</td>
<td>MOVE 7 TO CUS-ID</td>
<td>MOVE 7 TO CUS-ID</td>
</tr>
<tr>
<td>READ CUSTOMER KEY IS CUS-ID</td>
<td>FIND CUSTOMER USING CUS-ID</td>
<td>EXEC SQL</td>
<td>MOVE “CUSTOMER” TO</td>
</tr>
<tr>
<td>DISPLAY CUS-NAME</td>
<td>DISPLAY CUS-NAME</td>
<td>SELECT NAME</td>
<td>REC-NAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTO :CUS-NAME</td>
<td>CUSTOMER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FROM CUSTOMER</td>
<td>STATUS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WHERE ID = :CUS-ID</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>END-EXEC</td>
<td>DISPLAY CUS-NAME</td>
</tr>
<tr>
<td>DISPLAY CUS-NAME</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Analyzing Embedded DML Code

```sql
23  exec sql
24    declare BYZIP cursor for
25    select NAME, PHONE
26    from CUSTOMER
27    where ZIP = :CUS-ZIP
28  end-exec.
... 
51  ACCEPT CUS-ZIP.
52  exec sql
53    open BYZIP
54  end-exec.
55  MOVE 0 TO END-SEQ.
56  PERFORM DISPLAY-CUS UNTIL END-SEQ = 1.
... 
78  DISPLAY-CUS.
79  exec sql
80    fetch BYZIP
81    into :CUS-NAME,
82    :CUS-PHONE
83  end-exec.
84  IF SQLCODE = 0
85    DISPLAY CUS-NAME CUS-PHONE
86 ELSE MOVE 1 TO END-SEQ.
```

How to retrieve the dataflow between lines 51 and 84?
Why Dynamic Analysis?

- Static analysis techniques might fall short
  - in the presence of dynamically generated queries

- An (exaggerated) example (Java/JDBC…)

```java
String query, SQLv, SQLa, SQLs, SQLc;
SQLv = currentAction;
SQLa = keyboard.readString();
SQLs = userDefaultTable;
SQLc = getFilter(currentDate, filterNumber);
Connection con = DriverManager.getConnection(url, login, pwd);
Statement stmt = con.createStatement();
query = SQLv + SQLa + SQLs + SQLc;
ResultSet rset = SQLstmt.executeQuery(query);
```

executed query string ???
Dynamic Analysis

• Two steps
  • Capturing query execution traces (tracing)
  • Analyzing query execution traces

• Capturing techniques
  • DBMS logs
  • Program instrumentation
  • API substitution
  • API overloading
  • Aspect-based tracing ➔ a lot of advantages !

Ask Ralf how to do it…
Aspect-based Tracing

• Aspect-based tracing consists in
  • specifying *separately* the tracing functionality
  • by means of *aspects*

• Advantages
  • no alteration of the source code
  • richness of the produced SQL trace
    • actual SQL query
    • query results
    • source code locations
Aspect-based tracing of SQL queries

- SQL query execution involves 4 main steps
  - Statement preparation
  - Value injection
  - Query execution
  - Result extraction

```java
String query = "select Num from ORDERS where Sender = "+custNum+"";
Statement stmt = con.createStatement();
ResultSet rset = stmt.executeQuery(query);
while (rset.next()){
    custNumber = rset.getString(1);
    // result extraction
}
```

```java
String query = "select Num from ORDERS where Sender = ?";
PreparedStatement stmt = con.prepareStatement(query);
stmt.setString(1, custNum);
ResultSet rset = stmt.executeQuery();
while (rset.next()){
    custNumber = rset.getString(1);
    // result extraction
}
```
Aspect-based tracing of SQL queries

- Each step is captured by the tracing aspect
  - Statement preparation
    ```
    pointcut statementPreparation(String query) :
      call (PreparedStatement Connection.prepareStatement(String)) && args(query);
    ```
  - Value injection
    ```
    pointcut valueInjection(PreparedStatement statement, int pos, Object val) :
      call (void PreparedStatement.set*(int, *)) && target(statement)
      && args(position, value);
    ```
  - Query execution
    ```
    pointcut queryExecution(String query) :
      call(ResultSet Statement.executeQuery(String)) && args(query);
    ```
    ```
    pointcut queryExecution(PreparedStatement statement) :
      call(ResultSet PreparedStatement.executeQuery()) && target(statement);
    ```
  - Result extraction
    ```
    pointcut resultExtraction(ResultSet rSet) :
      call(** ResultSet.get**(/*)) && target(rSet);
    ```
Example code fragment

```java
String query = "select Num from ORDERS where Sender = '"+custNum+"' ";
Statement stmt = con.createStatement();
ResultSet rset = stmt.executeQuery(query);
```

Tracing aspect

```java
public aspect SQLTracing { 

    private MySQLLog log = new MySQLLog();

    pointcut queryExecution(String query) :
            call(ResultSet Statement.executeQuery(String)) &&
            args(query);

    before(String query): queryExecution(query){
        String classFile = thisJoinPoint.getSourceLocation().getFileName();
        int lineOfCode = thisJoinPoint.getSourceLocation().getLine();
        Statement statement = (Statement) thisJoinPoint.getTarget();
        log.traceQuery(classFile, lineOfCode, statement.hashCode(), query);
    }

}
```

source code location statement # executed query string
SQL Trace Analysis for Data Reverse Engineering

- Detecting potential implicit foreign keys
  - SQL joins (FK usage)
  - output-input dependency (FK usage)
  - input-input dependency (FK validation)

Example trace

```
select Name, O.Num, Date from CUSTOMER C, ORDERS O where O.Sender = C.Num
```
SQL Trace Analysis for Data Reverse Engineering

- Detecting potential implicit foreign keys
  - SQL joins (FK usage)
  - output-input dependency (FK usage)
  - input-input dependency (FK validation)

Example trace

```sql
select Sender from ORDERS where Date = '2008-20-06'
ggetString(1) = C400
select Name, Address from CUSTOMERS where Num = 'C400'
ggetString(1) = B314
select Name, Address from CUSTOMERS where Num = 'B314'
ggetString(1) = C891
select Name, Address from CUSTOMERS where Num = 'C891'
```
SQL Trace Analysis for Data Reverse Engineering

- Detecting potential implicit foreign keys
  - SQL joins (FK usage)
  - output-input dependency (FK usage)
  - input-input dependency (FK validation)

Example trace

```sql
select count(*) from CUSTOMER where Num = 'C400'
getCode(1) = 1
insert into ORDERS(Num, Date, Reference, Sender) values (456,'2008-06-20','PA601','C400')
select count(*) from CUSTOMER where Num = 'C152'
getCode(1) = 0
select count(*) from CUSTOMER where Num = 'C251'
getCode(1) = 1
insert into ORDERS(Num, Date, Reference, Sender) values (457,'2008-06-20','ST014','C251')
```
Implicit Constraints Validation

- Detected dependencies
  - are only candidate constraints
  - must be validated through data analysis, user inputs, developers interviews
Step 2: Detecting data inconsistencies

- Automatic generation
  - of data analysis queries
  - from the enriched, annotated schema

```sql
select * from ORDERS
where Sender not in (select Num from CUSTOMER)
```
Step 3: Improving Data Validation

- **Goal:**
  - preventing new data inconsistencies to be inserted

- **Possible, non-exclusive strategies**
  - Improving database side validation
  - Improving program side validation
  - Transferring data validation from program side to database side

- **Improving database side validation**
  - Modifying the DDL
    - necessitates full data cleaning
    - may prove very difficult in practice
  - Our approach:
    - Generation of fault-tolerant triggers
    - temporarily tolerating consistency
Step 3: Improving Data Validation

- Improving program side validation
  - involves the detection of *unsafe data access paths*

- An unsafe data access path
  - executes a *critical database operation* w.r.t. related implicit constraints
  - without proper data validation

- Critical database operation w.r.t constraint C
  - may potentially cause the violation of C
  - ex. implicit FK ORDERS.Sender \( \rightarrow \) CUSTOMER.Num, critical operations include:
    - insert into ORDERS;
    - update ORDERS, where Sender value is modified;
    - update CUSTOMER, where the Num value is modified;
    - delete from CUSTOMER;
Example of safe data access paths

The following fragment makes sure that NewSender refers to an existing customer

```
exec sql
    select count(*) into :NBR
from CUSTOMER
where Num = :NewSender
end-exec
if (NBR = 0)
    display('unknown customer - insertion aborted !')
else
    exec sql
        insert into ORDERS(Num, Date, Reference, Sender)
        values(:NewNum, :NewDate, :NewRef, :NewSender)
end-exec
end-if
```

- Reactive validation
- Critical operation O preceded by a validation query V conditioning its execution
- Data dependency between V inputs and a subset of O inputs
- Control dependency between a control predicate C and O
- Data dependency between V output and C
Plan

- Introduction & Motivation

- Part I: Database-programs co-Analysis
  - Database Reverse Engineering

- Part II: Database-programs co-Evolution
  - Database Schema Change
  - Database Migration

- Tool support

- Industrial project

- Conclusions and perspectives
**Database Schema Change**

- A schema change can
  - **augment** the information contents of the schema

```
<table>
<thead>
<tr>
<th>CUSTOMER</th>
<th>CUSTOMER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cust#</td>
<td>Cust#</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Address</td>
<td>Address</td>
</tr>
<tr>
<td>Phone</td>
<td>Phone</td>
</tr>
</tbody>
</table>
```

- **decrease** the information contents of the schema

```
<table>
<thead>
<tr>
<th>CUSTOMER</th>
<th>CUSTOMER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cust#</td>
<td>Cust#</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Address</td>
<td>Address</td>
</tr>
</tbody>
</table>
```

- **preserve** the information contents of the schema (\(\approx\) **schema refactoring**)

```
<table>
<thead>
<tr>
<th>CUSTOMER</th>
<th>CUSTOMER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cust#</td>
<td>Cust#</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Phone</td>
<td>Phone</td>
</tr>
</tbody>
</table>
```

\[\rightarrow\] Our focus = schema refactoring
Potential Impact of Database Schema

Refactoring

- DML syntax
- Data model
- DDL syntax
- Programs
- Source code execution
- Consistency relationship
- Database Schema
- Data instances

Computer Science Faculty
University of Namur, Belgium
Focus: Program-Schema Consistency

- Generic approach
  - Use of a generic data model (GER)
  - Use of an abstract DML (LDA)
  - Consistency rules are based on the mapping between the GER meta-model and the LDA syntax
  - These rules hold for existing data models and DML via abstraction
The GER Model

- abstract union of all operational (practically used) database models
  - serves as a pivot model from a model to another
- encompasses several paradigms:
  - ER, UML, SQL, CODASYL, IMS, file structures like COBOL, XML, etc.
- encompasses several levels of abstraction:
  - conceptual, logical, physical
- sound basis for building transformational frameworks
  - all inter-model transformations become intra-model transformations
- operational models defined by specialization rules
  - selection
  - renaming
  - assembly rules (structural constraints)
The GER Model

- Example of an operational model: SQL2 relational model

<table>
<thead>
<tr>
<th>relational constructs</th>
<th>GER constructs</th>
<th>assembly rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>database schema</td>
<td>schema</td>
<td></td>
</tr>
<tr>
<td>table</td>
<td>entity type</td>
<td>an entity type includes at least one attribute</td>
</tr>
<tr>
<td>domain</td>
<td>simple domain</td>
<td></td>
</tr>
<tr>
<td>nullable column</td>
<td>single-valued and atomic attribute with cardinality [0-1]</td>
<td></td>
</tr>
<tr>
<td>not null column</td>
<td>single-valued and atomic attribute with cardinality [1-1]</td>
<td></td>
</tr>
<tr>
<td>primary key</td>
<td>primary identifier</td>
<td>a primary identifier comprises attributes with cardinality [1-1]</td>
</tr>
<tr>
<td>unique constraint</td>
<td>secondary identifier</td>
<td></td>
</tr>
<tr>
<td>foreign key</td>
<td>reference group</td>
<td>the composition of the reference group must be the same as that of the target identifier</td>
</tr>
<tr>
<td>SQL names</td>
<td>GER names</td>
<td>the GER names must follow the SQL syntax</td>
</tr>
</tbody>
</table>
The LDA Language

- semi-predicative, abstract language
- serving as a pivot DML language, defined on top of the GER
- adapted from a language defined by [Hainaut 1986]
- includes
  - DML primitives: create, delete, modify
  - Types: integer, string, booleans, and GER types
  - Conditional statements: If-then-else, For-loops, While-loops
  - I/O statements: input, print

```
program MySampleProgram.
schema 'sales';
cus : CUSTOMER;
begin
  for cus := CUSTOMER(:CITY = 'Namur') do
    if (cus.ACCOUNT > 0) then
      print(cus.NAME)
    else
      delete cus
    endif;
  endfor
end
```
LDA example: is the following create statement consistent with its DB schema?

create det := DETAIL(:QUANTITY = 5)
  and (ORD-DET : ord)
  and (DET-PRO : PRODUCT(:NUMBER = 123))

+ Type compatibility: det, ord, 5, 123
Consistency Checking/Impact Analysis

- Automatic generation of consistency checkers from annotated grammars
- Example: SQL

```
"SELECT" ColumnList "FROM" TableList WhereClause? ... -> SelectQuery
{Column "","}"+ -> ColumnList
{Table, "","}"+ -> TableList
```

```
constraint(SelectQuery, for each t:Table in TableList: isAnEntityOf(t))
constraint(SelectQuery, for each c:Column in ColumnList : exists t:Table in TableList : isAnAttributeOf(c,t))
constraint(SelectQuery, for each c:Column in WhereClause? : exists t:Table in TableList : isAnAttributeOf(c,t))
```

SQL queries  \[\text{generation}\]  SQL consistency checker  \[\text{Inconsistencies}\]  Schema facts
Consistency Preservation/Co-Evolution

- Schema refactorings are seen as schema transformations
- Idea
  - associate program transformations to GER-to-GER schema transformations
  - these propagation rules aim at reconstructing possibly broken consistency
Co-Transformations

- Database schema change involves
  1. Schema transformations
  2. Data transformations
  3. Program transformations

= Co-transformations

- Some definitions… **Thanks Ralf!**
  - “A co-transformation transforms mutually dependent software artifacts of different kinds simultaneously (…)” [Lämmel04a]
  - “(…) two or more artifacts of potentially different types are involved, while transformation at one end necessitates reconciling transformations at other ends such that global consistency is reestablished” [Lämmel04b]


Consistency Preservation by Co-Transformations

$T_1 = $ Transformation of a multi-valued, compound attribute $A$ into an entity type $EA$

create $e := E((: A_1 = a_1)$
and $(: A_2 = a_2)$
and $(: A[1].B_1 = b_{11})$
\[\ldots\]
and $(: A[1].B_n = b_{1n})$
\[\ldots\]
and $(: A[N].B_1 = b_{N1})$
\[\ldots\]
and $(: A[N].B_n = b_{Nn})$)

\[t_{1c} \Rightarrow \]

create $e := E((: A_1 = a_1)$
and $(: A_2 = a_2))$

for $i$ in $1..N$ do
create $ea := EA((: B_1 = b_{i1})$
\[\ldots\]
and $(: B_n = b_{in})$
and $(: R : e))$
endfor
Consistency Preservation by Co-Transformations

\[ T_2 = \text{Transformation of a relationship type } R \text{ into a foreign key } Rld_1 \ldots Rld_n \]

\[
\text{create } e2 := E2(\text{[: } B_1 = b_1) \\quad \text{and [: } B_2 = b_2) \\quad \text{and (} R : e)) \\
\text{t}_{2c} \quad \Rightarrow \quad \text{create } e2 := E2(\text{[: } B_1 = b_1) \\quad \text{and [: } B_2 = b_2) \\quad \text{and (} Rld_1 = e.Id_1 \\
\ldots \quad \Rightarrow \quad \text{and (} Rld_n = e.Id_n) \]
\[ T_2 \circ T_1 = \text{Transformation of a multi-valued, compound attribute } A \text{ into an entity type } EA + \text{a foreign key RA1} \]

\[
E \\
A_1 \\
A_2 \\
A[0-N] \\
B_1 \\
\ldots \\
B_n \\
id: A1
\]

\[
E \\
A_1 \\
A_2 \\
B_1 \\
\ldots \\
B_n \\
id: A1
\]

\[
E \\
RA1 \\
B_1 \\
\ldots \\
B_n \\
id: RA1 \\
ref: RA1
\]

**create e :=** \( E( (: A_1 = a_1) \) \\
\( \text{and} (: A_2 = a_2) \) \\
\( \text{and} (: A[1].B_1 = b_{11}) \) \\
\( \text{...} \) \\
\( \text{and} (: A[1].B_n = b_{1n}) \) \\
\( \text{...} \) \\
\( \text{and} (: A[N].B_1 = b_{N1}) \) \\
\( \text{...} \) \\
\( \text{and} (: A[N].B_n = b_{nn}) \) 

\[ t_{2c} \circ t_{1c} \Rightarrow \]

**create e :=** \( E( (: A_1 = a_1) \) \\
\( \text{and} (: A_2 = a_2) \) \\
\( \text{for i in 1..N do} \) \\
\( \text{create ea := } EA( (: B_1 = b_{i1}) \) \\
\( \text{...} \) \\
\( \text{and} (: B_n = b_{in}) \) \\
\( \text{and} (: RA_1 = e.A_1) \) \\
\( \text{endfor} \)
Plan

- Introduction & Motivation

- Part I: Database-programs co-Analysis
  - Database Reverse Engineering

- Part II: Database-programs co-Evolution
  - Database Schema Change
  - Database Migration

- Tool support

- Industrial project

- Conclusions and perspectives
Database Migration

- Database Migration = 3 steps

  1. Schema conversion
     - translation of the legacy schema into an equivalent schema in the new technology

  2. Data conversion
     - migration of the data instances from the legacy system to the new one
     - depends on the schema conversion step

  3. Program conversion
     - modification of the programs so that they access the new DB instead of the legacy one
     - programming language, user interface unchanged
     - depends on the schema conversion step
Wrapper-based Program Conversion

- **Backward wrappers**
  - simulate the legacy DMS API on top of the new database
  - allow legacy programs to be reused with minimal adaptation
  - our approach:
    - *automatic generation* of one wrapper per legacy record type
    - wrappers call each other to synchronize currency indicators (reading state)
Wrappers’ Objectives

- **Structure conversion**
  - Adaptation of the legacy database queries to the target structure
  - Based on the mapping between the source and target schemas

- **Language conversion**
  - High-fidelity translation of the legacy DML primitives
    - using target DML commands
    - according to structure conversion rules
  - Such a translation includes
    - Reading/writing sequence management
    - Currency indicators management (current state of reading sequences)
    - Error/exception handling
CODASYL to Relational Example

- **Structure conversion**
  - Each set type S is translated into a foreign key declared in the *member* table

- **Language conversion**
  - *set cursor* declaration:
    ```sql
    DECLARE S CURSOR FOR
    SELECT <columns>
    FROM ORDERS
    WHERE Owner = <current_customer>.ID
    ORDER BY <set_ordering_criteria>
    ```
  - SQL translation of *FIND NEXT WITHIN S* within the member wrapper

<table>
<thead>
<tr>
<th>Set current record</th>
<th>Set cursor state</th>
<th>Set cursor actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>member</td>
<td>open</td>
<td>fetch</td>
</tr>
<tr>
<td>member</td>
<td>closed</td>
<td>re-position to current, fetch</td>
</tr>
<tr>
<td>owner</td>
<td>closed</td>
<td>open, fetch</td>
</tr>
</tbody>
</table>
Legacy Programs Transformations

- **Goal:** interface the legacy programs with the generated wrappers
- **Several transformation steps**
  - replacing legacy database primitives with wrapper invocations
  - renaming some variables
  - adapting the type of some variables
  - adding new (generated) variable declarations
  - adding new (generated) code sections
  + pre-processing, post-processing and local pretty-printing

- **Example**

<table>
<thead>
<tr>
<th>Initial code fragment</th>
<th>Transformed code fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND NEXT WITHIN S</td>
<td>FIND NEXT WITHIN S</td>
</tr>
<tr>
<td>IF DB-STATUS = ZEROS</td>
<td>IF WR-STATUS = ZEROS</td>
</tr>
<tr>
<td>PERFORM GET-RECORD</td>
<td>PERFORM GET-RECORD</td>
</tr>
<tr>
<td>ELSE</td>
<td>ELSE</td>
</tr>
<tr>
<td>PERFORM DB-ERROR.</td>
<td>PERFORM DB-ERROR.</td>
</tr>
</tbody>
</table>
Plan

- Introduction & Motivation

- Part I: Database-programs co-Analysis
  - Database Reverse Engineering

- Part II: Database-programs co-Evolution
  - Database Schema Change
  - Database Migration

- Tool support

- Industrial project

- Conclusions and perspectives
Tool support

- DB-MAIN CASE tool (University of Namur, ReVeR)
  - DDL extraction
  - Schema storage, analysis and manipulation
  - Implicit constraint validation
  - Schema mapping management
  - Data analysis & migration
  - Wrapper generation (COBOL-to-SQL, CODASYL-to-SQL)
- ASF+SDF Meta-Environment (CWI, Amsterdam)
  - *Don’t miss the talk by Jurgen Vinju on Thursday !!!*
  - Program analysis & transformation -- *Thanks Ralf for the COBOL grammar !*
  - Consistency checker generation
- Grammar Deployment Kit (University of Amsterdam)
  - Program slicing & Dataflow analysis
  - COBOL/ESQL, COBOL/CODASYL, COBOL/IMS
Plan

• Introduction & Motivation

• Part I : Database-programs co-Analysis
  • Database Reverse Engineering

• Part II : Database-programs co-Evolution
  • Database Schema Change
  • Database Migration

• Tool support

• Industrial project

• Conclusions and perspectives
Industrial Migration Project (ReVeR)

- Customer: a Belgian ministry
- Goal of the project
  - Migrate a legacy COBOL application using an IDS/II database towards a relational platform (DB2)
  - Legacy programs running on Bull GCOS8 Mainframe
  - Remote access to the target DB2 database through a DBSP gateway
- Metrics
  - > 2 millions lines of COBOL code
  - # programs: 2,273
  - # call relationships: 9,527
  - # IDS/II verbs: 20,856
Two-phase project

- **Initial system**
  - Legacy programs
  - IDS/II DB

- **After refactoring**
  - Legacy programs*
  - procedures
  - IDS/II DB

- **After migration**
  - Legacy programs*
  - procedures*
  - DB2 DB
  - wrappers

*Indicates updated versions or refactoring
**Results**

- Implicit schema constructs discovered
  - 76 implicit foreign keys
  - finer-grained field decompositions and format constraints
    - Ex: String(8) of the form ‘YYYYMMDD’ → Date
    - more expressive record type/field names
- Data analysis results
  - Format Constraints
    - errors detected in more than 70% of the tables
    - 35 millions of invalid dates
      - example: ‘20070931’, ‘00000000’, ‘99999999’
      - mainly due to erroneous program behaviour
  - Foreign key constraints
    - 24 implicit foreign keys (up to 76) proved to be violated by the data!
Data Analysis Report

Total number of records: 4930158

<table>
<thead>
<tr>
<th>Column</th>
<th>Constraint</th>
<th>Expected type</th>
<th>Result</th>
<th>Nbr of records</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsom_stat</td>
<td>is a valid numeric</td>
<td>Numeric</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>rsom_stat</td>
<td>is not null (= 0.00)</td>
<td>Numeric</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>room_crt</td>
<td>is a valid date</td>
<td>Date</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>room_crt</td>
<td>is not null (= 00000000)</td>
<td>Date</td>
<td>X</td>
<td>7</td>
</tr>
<tr>
<td>room_crt</td>
<td>is not null (= 99999999)</td>
<td>Date</td>
<td>X</td>
<td>1596</td>
</tr>
<tr>
<td>rsom_prod</td>
<td>is a valid numeric</td>
<td>Numeric</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>rsom_prod</td>
<td>is not null (= 0.00)</td>
<td>Numeric</td>
<td>✓</td>
<td>4930158</td>
</tr>
<tr>
<td>rsom_cntr</td>
<td>is a valid numeric</td>
<td>Numeric</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>rsom_cntr</td>
<td>is not null (= 0.00)</td>
<td>Numeric</td>
<td>X</td>
<td>354301</td>
</tr>
<tr>
<td>room_dact</td>
<td>is a valid date</td>
<td>Date</td>
<td>X</td>
<td>1436</td>
</tr>
<tr>
<td>room_dact</td>
<td>is not null (= 00000000)</td>
<td>Date</td>
<td>X</td>
<td>607926</td>
</tr>
<tr>
<td>room_dact</td>
<td>is not null (= 99999999)</td>
<td>Date</td>
<td>X</td>
<td>180504</td>
</tr>
</tbody>
</table>
### System Conversion Results

- **Schema conversion results**

<table>
<thead>
<tr>
<th></th>
<th>IDS/II</th>
<th>Refined</th>
<th>Conceptual</th>
<th>DB2</th>
</tr>
</thead>
<tbody>
<tr>
<td># entity types</td>
<td>112</td>
<td>112</td>
<td>105</td>
<td>147</td>
</tr>
<tr>
<td># rel. types</td>
<td>73</td>
<td>118</td>
<td>110</td>
<td>144</td>
</tr>
<tr>
<td># attributes</td>
<td>1 249</td>
<td>1 509</td>
<td>1 210</td>
<td>1 841</td>
</tr>
</tbody>
</table>

- **Program transformation results**

<table>
<thead>
<tr>
<th></th>
<th>Transformed</th>
<th>Manually adapted</th>
</tr>
</thead>
<tbody>
<tr>
<td># programs</td>
<td>996</td>
<td>15</td>
</tr>
<tr>
<td># copybooks</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td># FIND</td>
<td>7 632</td>
<td>21</td>
</tr>
<tr>
<td># READY</td>
<td>5 891</td>
<td>0</td>
</tr>
<tr>
<td># GET</td>
<td>5 345</td>
<td>0</td>
</tr>
<tr>
<td># FINISH</td>
<td>1 144</td>
<td>0</td>
</tr>
<tr>
<td># MODIFY</td>
<td>1 008</td>
<td>0</td>
</tr>
<tr>
<td># STORE</td>
<td>942</td>
<td>0</td>
</tr>
<tr>
<td># ERASE</td>
<td>433</td>
<td>0</td>
</tr>
<tr>
<td># PRECALCULcATE</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>
what about correctness and performance?

- Correctness 😊
  - systematic testing process
  - in collaboration with IDS/II experts from the customer side

- Performance 😞
  - significant degradation observed (without hardware upgrade)
  - largely dependent on the program logic and the database state
  - but difficult to draw general conclusions yet

- Customer convinced by the results 😊
  - other similar migration projects are currently being conducted
Lessons learned from the project

- Database reverse engineering
  - difficult to find developers who really know the whole system
  - full automation clearly unrealistic

- Database design
  - a lot of unexpected manual (semi-automatic) work
    - the same structure not always transformed in the same way

- Wrapper generation
  - requires a very good knowledge of the legacy DMS
  - simulating the legacy DB primitives can be challenging
  - necessitates a long and systematic testing process

- Program transformation
  - high level of automation achievable
Plan

- Introduction & Motivation

- Part I: Database-programs co-Analysis
  - Database Reverse Engineering

- Part II: Database-programs co-Evolution
  - Database Schema Change
  - Database Migration

- Tool support

- Industrial project

- Conclusions and perspectives
Conclusions and Perspectives

- Database-programs co-analysis approach
  - combining schema analysis, data analysis, static & dynamic program analysis
- Database-programs co-evolution approach
  - combining transformational and generative techniques
- Co-transformations
  - form a sound basis for schema-program co-evolution
  - the abstract propagation rules can be re-used in multiple contexts
    - DB migration, schema refactoring, DB design, DB integration, etc...
- Current limitations
  - semantics-preserving schema transformations
  - Wrapper-based conversion is obviously a short-term solution (transition period)
- Future work
  - (Semi-)automatic adaptation of the data manipulation logic of a program to a new DB paradigm (dream = nested COBOL/CODASYL loops converted as a single SQL join)
Thank you for your attention!

Any questions?
The GER Model

- Conceptual schema fragment (1)
The GER Model

- Conceptual schema fragment (2)

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Example Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>multivalued attribute</td>
<td>Phone[0-5], Mobile[0-1]</td>
</tr>
<tr>
<td>optional attribute</td>
<td>Address</td>
</tr>
<tr>
<td>compound attribute</td>
<td>Street, City</td>
</tr>
</tbody>
</table>

N-ary relationship type

Diagram:
- SALESMAN
  - PID
  - Name
  - Phone[0-5]
  - Mobile[0-1]
  - Address
  - Street
  - City
- CUSTOMER
- PRODUCT
- sold
- Date
- Volume

Diagram shows the relationships and attributes in a conceptual schema fragment (2).
The GER Model

- Logical schema fragment

```
<table>
<thead>
<tr>
<th>record set / table</th>
<th>ORDER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ORD-ID</td>
</tr>
<tr>
<td></td>
<td>DATE_RECEIVED</td>
</tr>
<tr>
<td></td>
<td>ORIGIN</td>
</tr>
<tr>
<td></td>
<td>DETAIL[1:5] array</td>
</tr>
<tr>
<td></td>
<td>REFERENCE</td>
</tr>
<tr>
<td></td>
<td>QTY-ORD</td>
</tr>
<tr>
<td></td>
<td>id: ORD-ID</td>
</tr>
<tr>
<td></td>
<td>ref: ORIGIN</td>
</tr>
</tbody>
</table>

| array multivalued field |
|                        |
| CUSTOMER               |
|                        |
| id: CUSTOMER ID        |

foreign key

University of Namur, Belgium

Computer Science Faculty

University of Namur, Belgium
```
The GER Model

- Physical schema fragment

```
unique index

index

storage space

<table>
<thead>
<tr>
<th>PRODUCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRO_CODE</td>
</tr>
<tr>
<td>CATEGORY</td>
</tr>
<tr>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>UNIT_PRICE</td>
</tr>
</tbody>
</table>

id: PRO_CODE
acc
acc: CATEGORY

PRODUCT.DAT
PRODUCT
```