Towards a General Consistency Management Framework in the Context of Database Application Evolution

Anthony Cleve
10th December 2007
Joint work with
Ragnhild Van Der Straeten (VUB)
Jean-Luc Hainaut
Plan

• Introduction
• What does consistency mean to Database Applications?
• Inconsistency classification for MDE revisited
• Consistency management framework for program-db co-evolution
  • Co-evolution scenarios
  • Consistency checking
  • Consistency preservation/reconstruction
• Tool support and demo
• Conclusions and perspectives
Introduction

• Software-intensive systems made of
  • inter-dependent artefacts
  • of different natures
  • at different levels of abstraction

• Consistency management
  • crucial issue in the context of software evolution

• Our focus
  • evolution of database applications
Introduction

- Research objective
  - A general framework for consistency management including
    - a classification of possible inconsistencies
    - a classification of (co-)evolution scenarios
    - A generic approach
      - for consistency checking (detecting inconsistencies)
      - for consistency preservation (preventing or resolving inconsistencies)

- Research questions (still open)
  1. How to classify inconsistencies arising in a database evolution context?
  2. How is such a classification related to the existing classifications in an MDE context?
  3. Can consistency checkers be derived from inter-model mappings?
  4. How to derive consistency reconstruction from detected inconsistencies?
What does consistency mean to database applications?
What does consistency mean to database applications?

- DML syntax
- Data model
- DDL syntax

- Programs
  - source code
  - execution

- Conceptual schema
- Logical schema
- Physical schema
- DDL code

Data instances

Must be consistent with
Nature of consistency relationships

Logical Schema VS Conceptual schema
- Semantic compatibility (↔ equivalence)
- Potential semantic gap during logical design

Logical Schema VS Data Model
- Meta-model compliance

Program source code VS DML syntax
- Syntactical compliance of DB queries

Program source code VS Logical schema
- Structural compliance of DB queries

Program execution VS Data instances
- Behavioural compatibility
- Because of the potential gap between CS and LS
- Programs may introduce inconsistent data instances

Physical Schema VS Logical schema
- Semantic compatibility/equivalence

DDL code VS Physical schema
- High-fidelity translation

DDL code VS DDL syntax
- Syntactical compliance

Data instances VS DDL code
- Format compliance
Is the following query consistent?

```
insert into ORDERS (ORDNUM, DATE, CUSNUM) values ('1', '10-12-2007', '2')
```
Is the following query consistent?

```
isert into ORDERS (ORDNUM, DATE, CUSNUM) values (1, '10-12-2007', 2)
```
Inconsistency classification for MDE revisited

- In the MDE context…
  - Inconsistencies can be classified according to 2 dimensions [Van Der Straeten 2005]

<table>
<thead>
<tr>
<th>Specification</th>
<th>Behavioural</th>
<th>Structural</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>invocation interaction inconsistency</td>
<td>dangling type reference</td>
</tr>
<tr>
<td></td>
<td>observation interaction inconsistency</td>
<td>inherited cyclic composition connector specification missing</td>
</tr>
<tr>
<td>Specification-Instance</td>
<td>specification incompatibility</td>
<td>instance specification missing</td>
</tr>
<tr>
<td>Instance</td>
<td>specification behaviour incompatibility</td>
<td></td>
</tr>
<tr>
<td></td>
<td>invocation behaviour inconsistency</td>
<td></td>
</tr>
<tr>
<td></td>
<td>observation behaviour inconsistency</td>
<td></td>
</tr>
<tr>
<td>Instance</td>
<td>invocation inheritance inconsistency</td>
<td>disconnected model</td>
</tr>
<tr>
<td></td>
<td>observation inheritance inconsistency</td>
<td></td>
</tr>
<tr>
<td></td>
<td>instance behaviour incompatibility</td>
<td></td>
</tr>
</tbody>
</table>

- In the DB context… difficult to distinguish between
  - structural and semantic/behavioural consistency
    - Notion of *semantic compatibility* between two database schemas
    - A database query has a syntax, a structure but also a behaviour/semantics at run time

  - specification and instance levels
    - conceptual, logical and physical schemas are of distinct levels of abstraction (all at the specification level)
    - a DB query relates to the specification level, but can be seen as an instance of DML syntax rules
    - a particular execution of a DB query rather relates to the instance level
Classification of Co-evolution Scenarios

- *Database first* evolution
- 3 dimensions
  - *Structural dimension*: is the database structure modified?
  - *Semantic dimension*: does the data meaning evolve?
  - *Platform dimension*: are the DDL/DML replaced?
- Typical co-evolution scenarios

<table>
<thead>
<tr>
<th></th>
<th>structural</th>
<th>semantic</th>
<th>platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>database migration</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>database restructuring</td>
<td>Yes</td>
<td>?</td>
<td>No</td>
</tr>
<tr>
<td>database integration</td>
<td>Yes</td>
<td>Yes</td>
<td>?</td>
</tr>
</tbody>
</table>
Example: Migration

- Initial goal: changing the DB platform

![Diagram of database migration](image)

- DML syntax
- Data model
- DDL syntax
- Conceptual schema
- Logical schema
- Physical schema
- DDL code
- Data instances

Must be consistent with

- Programs
  - source code
  - execution

University of Namur, Belgium
Consistency Management

- Includes
  - Consistency checking
  - Consistency preservation/reconstruction
- Focus on program – schema consistency
Consistency Management

• Generic approach
  • Use of a generic data model (GER)
  • Use of an abstract, wide-spectrum data manipulation language (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/reexpression
Consistency Management

- Generic approach
  - Use of a generic data model (GER)
  - Use of an abstract, wide-spectrum data manipulation language (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/reexpression
The GER Model

- abstract union of all operational (practically used) database models
- encompasses several paradigms:
  - ER, UML, SQL, CODASYL, IMS, file structures like COBOL, XML, etc.
- encompasses several levels of abstraction:
  - conceptual, logical, physical, external
- formal semantics based on NF2 relational theories
- sound basis for building transformational frameworks (all inter-model transformations become intra-model transformations)
- operational models defined by specialisation 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
• To serve as a pivot, wide-spectrum DML
• 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.lun';
CUSTOMER : cus;
string : city;
begin
  input(city);
  for cus := CUSTOMER(:CITY = city) do
    if (cus.ACCOUNT > 0) then
      print(cus.NAME)
    else
      delete cus
    endif;
  endfor
end
```
Consistency Checking

- A small example

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

+ Type compatibility: det, ord, 5, 123
Consistency Checking

Notations

\[ S \quad \text{GER schema} \]
\[ E \quad \text{set of entity types of } S \]
\[ R \quad \text{set of relationship-types of } S \]
\[ A \quad \text{set of attributes of } S \]
\[ att(o) \quad \text{set of attributes of object } o \ (o \in E \cup R \cup A) \]
\[ rel(e) \quad \text{set of relationship types in which entity type } e \text{ plays a role} \]
\[ type(v) \quad \text{type of variable } v \]

Rule 1. \( \forall \) expression \( e \) of the form \( \text{EntityType Conditions} \)

1. \( \text{EntityType} \in E \)
2. \( \forall \) condition \( c \in \text{Conditions} \) of the form \( \text{Attribute op ValueExpression} \)
   - \( \text{Attribute} \in \text{att(EntityType)} \)
   - \( \text{Attribute} \) and \( \text{ValueExpression} \) are of compatible types
3. \( \forall \) condition \( c \in \text{Conditions} \) of the form \( \text{RelType : EntityType}_2 \text{ Conditions}_2 \)
   - \( \text{RelType} \in \text{rel(EntityType)} \cap \text{rel(EntityType}_2) \)
4. \( \forall \) condition \( c \in \text{Conditions} \) of the form \( \text{RelType : Variable} \)
   - \( \text{type(Variable)} = \text{EntityType}_2 \in E \)
   - \( \text{RelType} \in \text{rel(EntityType)} \cap \text{rel(EntityType}_2) \)
Consistency Preservation

• Idea
  • co-transformational approach inspired by [Laemmel 2004]
  • associate propagation rules to GER-to-GER schema transformations
  • these propagation rules
    • are defined on top of the LDA syntax
    • aim at reconstructing possibly broken consistency
Consistency Preservation

\[ T_1 = \text{Transformation of a multi-valued, compound attribute } A \text{ into an entity type } EA \]

\[
\begin{align*}
create \ e & := E((:A_1 = a_1) \\
& \quad \text{and} (:A_2 = a_2) \\
& \quad \text{and} (:A[1].B_1 = b_{11}) \\
& \quad \quad \ldots \\
& \quad \text{and} (:A[1].B_n = b_{1n}) \\
& \quad \quad \ldots \\
& \quad \text{and} (:A[N].B_1 = b_{N1}) \\
& \quad \quad \ldots \\
& \quad \text{and} (:A[N].B_n = b_{Nn}))
\end{align*}
\]

\[
\begin{align*}
t_{1c} \quad & \Rightarrow \\
\text{for } i \text{ in } 1..N \text{ do} \\
\quad & \text{create } ea := EA((:B_1 = b_{i1}) \\
& \quad \text{and} (:B_n = b_{in}) \\
& \quad \text{and} (R : e)) \\
\text{endfor}
\end{align*}
\]
Consistency Preservation

\[ T_2 = \text{Transformation of a relationship type R into a foreign key } R_{Id1} \ldots R_{Idn} \]

\[
\begin{align*}
\text{create } e2 & := E2\left(\text{ (: } B_1 = b_1 \right) \\
& \quad \text{ and } \left(\text{ : } B_2 = b_2 \right) \\
& \quad \text{ and } \left( R : e \right) \\
\Rightarrow & \quad t_{2c} \\
\text{create } e2 & := E2\left(\text{ (: } B_1 = b_1 \right) \\
& \quad \text{ and } \left(\text{ : } B_2 = b_2 \right) \\
& \quad \text{ and } \left( R_{Id1} = e.Id_1 \right) \\
& \quad \ldots \\
& \quad \text{ and } \left( R_{Idn} = e.Id_n \right)
\end{align*}
\]
\[ T_2 \circ T_1 = \text{Transformation of a multi-valued, compound attribute } A \text{ into an entity type } \text{EA} + \text{a foreign key } RA_1 \]

\[ create \ e := E((: A_1 = a_1) \text{ and } (: A_2 = a_2) \text{ and } (: A[1].B_1 = b_{11}) \text{ and } (: A[1].B_n = b_{1n}) \text{ and } (: A[2].B_1 = b_{21}) \text{ and } (: A[2].B_n = b_{2n}) \text{ and } \ldots) \]

\[ t_2c \circ t_1c \Rightarrow \]

\[ for \ i \ in \ 1..N \ do \]

\[ create\ ea := EA((: B_1 = b_{i1}) \text{ and } (: B_n = b_{in}) \text{ and } (: RA_1 = e.A_1)) \]

\[ endfor \]
Tool Support

- Combines
  - DB-MAIN (University of Namur, ReveR)
  - the ASF+SDF Meta-Environment (CWI, Amsterdam)
- ASF equations access to DB-MAIN repository via a dedicated library
Input history log = transformation signatures

(ORDERS, R1) <- ATT-to-ET-inst(CUSTOMER, ORDERS)
(DETAILS, R2) <- ATT-to-ET-inst(ORDERS, DETAILS)
Input history log = transformation signatures

\[(\text{ORDERS,} [\langle \text{CUS\_NAME, NAME} \rangle, \langle \text{CUS\_COMP, COMPANY} \rangle]) \leftarrow \text{RT-to-FK}(R)\]
Consistency Preservation : demo 3

Input history log

(CLIENT) <- Rename-ET(CUSTOMER)
**Input history log**

(CUSNUM) <- Rename-ATT(CUSTOMER, NUMBER)

(ORDNUM) <- Rename-ATT(ORDER, NUMBER)
Input history log

(DETAIL, [ORD_NUM, NUMBER]) <-> RT-to-FK(ORD-DET)
(DETAIL, [PRO_NUM, NUMBER]) <-> RT-to-FK(DET-PRO)
(ORDER, [CUS_NUM, NUMBER]) <-> RT-to-FK(CUS-ORD)

%% renaming

(NUM_PRO) <-> Rename-ATT(PRODUCT, NUMBER)
(NUM_CUS) <-> Rename-ATT(CUSTOMER, NUMBER)
(NUM_ORD) <-> Rename-ATT(ORDER, NUMBER)

%% translation

(PRO_NUMMER) <-> Rename-ATT(PRODUCT, NUM_PRO)
(NUM_COM) <-> Rename-ATT(ORDER, NUM_ORD)
(KLANT_NUM) <-> Rename-ATT(ORDER, CUS_NUM)
(KLANT) <-> Rename-ET(CUSTOMER)
(COMMANDE) <-> Rename-ET(ORDER)
(PRODUKT) <-> Rename-ET(PRODUCT)
(DATUM) <-> Rename-ATT(COMMANDE, DATE)
(QUANT_STOCK) <-> Rename-ATT(PRODUKT, STOCK_QTY)
(DESCRIPTIE) <-> Rename-ATT(PRODUKT, DESCRIPTION)
(KLANT_NUMMER) <-> Rename-ATT(KLANT, NUM_CUS)
(NAAM) <-> Rename-ATT(KLANT, NAME)
(ADRESSE) <-> Rename-ATT(KLANT, ADDRESS)
(STAD) <-> Rename-ATT(KLANT, CITY)
(CATEGORIE) <-> Rename-ATT(KLANT, CATEGORY)
(COMPTE) <-> Rename-ATT(KLANT, ACCOUNT)
(QUANTITE) <-> Rename-ATT(DETAIL, QUANTITY)
(COM_NUM) <-> Rename-ATT(DETAIL, ORD_NUM)
Conclusions and Perspectives

• Contributions
  • general approach to consistency management in the context of database applications evolution
  • pivot abstract DML: other applications
  • automated support for consistency checking and reconstruction

• Limitations
  • prototype define on a limited subset of
    • the GER
    • semantics-preserving GER transformations
    • LDA syntax

• Perspectives
  • towards a better classification of inconsistencies (MOVES collaboration!)
  • generating consistency checkers from inter-model mappings
  • automatically deriving hints about inconsistency resolution
  • generating operational source code from LDA programs