[MoVES] Meeting - May 25th, Namur
The Computer Science Faculty of Namur

- The University of Namur
  - member of the Académie Louvain
  - 5,000 students
  - 6 faculties

- The Computer Science Faculty
  - 16 full time academics
  - 50 researchers
  - Research fields:
    - Information system engineering
    - Database engineering
    - Human-Computer interfaces
    - Software quality
    - Multimedia
    - Organization theory
    - Programming
    - Decision support systems
    - Mobile networks
    - ICT and Society
    - Security
    - etc.
The PRECISE group

- Precise is REsearch Center in Information Systems Engineering

- Research Interests
  - modelling and methods
  - database engineering
  - requirements engineering
  - Interoperability, reengineering and evolution
  - quality and measures
  - CASE and meta-CASE tools

- Team
  - 6 full time academics
    - Vincent ENGLEBERT
    - Naji HABRA
    - Jean-Luc HAINAUT
    - Patrick HEYMANS
    - Michael PETIT
    - Pierre-Yves SCHOBbens
  - 27 Researchers and collaborators
The PRECISE group

• **Goal:** to develop models (formalisms, languages), techniques, methodologies and tools for the engineering of Information systems

• *Information systems:* wide-scope, large-scale and multi-paradigm applications considered in their organizational context; software and data are major components of IS.

• Six transversal research axes

• Each axis is inter-disciplinary

• Each project contributes to one or several axes

• PReCISE is open and adaptive
The PRECISE group

- Modelling and methods
  
  *the common resources on which the other axes are based*

- Database engineering
  
  *addressing the specific data component of IS*

- Requirements engineering
  
  *the early stage of any design process*

- Interoperability, reengineering and evolution
  
  *coping with application domain evolution and its impact of all the IS components*

- Quality and measures
  
  *focusing on a major dimension of engineering artefacts and processes*

- CASE and meta-CASE tools
  
  *indispensable support of all engineering activities*
Partner

- IMRU Research Group (Louvain School of Management/University of Namur)

- Theme: **Web Service Composition**
  - Web Service Communities
  - Context-based and multi-type policy approach for Web service composition
  - Modelling and Developing Self-Healing Web Services Using Aspects
  - Policies for Context-driven Transactional Web Services
  - Dynamic Web Service Composition within a Service-Oriented Architecture
  - Dynamic Requirements Specification for Adaptable and Open Service-Oriented Systems

- Team
  - Prof. Stéphane Faulkner
  - Prof. Philippe Thiran
  - Dr Sattanathan Subramanian
  - Ivan Jureta (PhD Student)
  - Stéphane Dehousse (PhD Student)

1. Model-based Engineering of Evolving Data-intensive Systems

2. Model-based Engineering of Software Product Lines

3. A Flexible MetaCase Environment to support Domain-Specific Visual Languages

4. Measuring the Quality of Programs and Models to Support their Evolution

5. Business IT Alignment
Model-based Engineering of Evolving Data-intensive Systems
1. Introduction
2. GER and specializations
3. Transformations
4. Main DB engineering processes
5. CASE tools
6. Evolution & coevolution
7. Evolution of relational database applications
8. Legacy database migration
9. LIBD contribution to MoVES
1. Introduction
1. Introduction

Presentation in two parts

- **transversal resources**: GER, transformations, DB life cycle and CASE tools (J-L Hainaut)

- **contribution to MoVES**: database and database application evolution (A. Clève)
1. Introduction

Is DB engineering MDE compliant?

Database engineering basically is model-driven (MDE) and transformational from its early days:

- abstract vs physical levels: *Information Algebra*, CODASYL, CACM, 1962

Is DB engineering that important?

- DB are at the core of Information systems (e.g., enterprise/management applications), that form 70% of the world software portfolio.
- DB and application programs most often are strongly linked: even slightest changes in the DB schema may imply changes in the application code
- Badly designed and maintained DB induce severe problems in application program design and maintenance
1. Introduction

**Contribution of PReCISE in DB engineering**

- LIBD (Laboratory of DB application engineering) a member of PReCISE
- models, techniques, methodologies and tools for DBE
- emphasis of complex processes: reverse engineering, evolution, maintenance, migration, integration
- multi-paradigm approaches (genericity)
- main recent results
  - generic data structure model (GER), transformational framework
  - methodologies for complex engineering processes (reverse engineering, migration, evolution, integration)
  - DB-MAIN CASE tool,
  - ReveR: spin-off of the LIBD (marketing and evolution of the methodologies and tools)
  - . . . + 5 PhD theses
1. Introduction

**Contribution of PReCISE / LIDB to MoVES**

- model-driven processes
- transformational approach
- database evolution / migration
- coevolution of DB/programs
2. GER and specializations
2. GER and specializations

Problem: *dealing with multiple models*
2. GER and specializations

Problem: *dealing with multiple models*

Considering all the inter-model and intra-model conversions, the organization requires $N \times N$ different mappings (= 16).
2. GER and specializations

Problem: *dealing with multiple models*

The usual answer: *introducing a pivot model.*
Considering all the inter-model and intra-model conversions, the organization requires \(2 \times N + 1\) different mappings (= 9).
2. GER and specializations

Problem: *dealing with multiple models*

*Our answer:* pivot model = Generic Entity-Relationship model (GER)

- abstract union of all operational (practically used) information/database models
- Encompasses several paradigms: ER, UML, SQL, CODASYL, IMS, file structures, XML, etc.
- Encompasses several levels of abstraction: conceptual, logical, physical, external
- formal semantics (based on NF\(^2\) relational theories)
- sound basis for building transformational frameworks (all inter-model transformations becomes intra-model transformations)
- operational models defined by specialization rules
  - selection
  - renaming
  - assembly rules (structural constraints)
2. GER and specializations

Problem: *dealing with multiple models*

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>
2. GER and specializations

Problem: *dealing with multiple models*

Example: *relational logical design.*
3. Transformations
3. Transformations

A transformation $T$ replaces a construct $C$ in a schema $S_1$ with another construct $C'$, leading to schema $S_2$.

$$T$$

$schemas$

$S_1$

$C$

$S_2$

$C'$

$data$
A transformation $\Sigma$ is defined by two mappings $T$ and $t$

$$\Sigma = <T, t>$$

$T$: structural mapping = syntax of $\Sigma$

t: instance mapping = semantics of $\Sigma$
3. Transformations

Structural mapping $T$ can be specified with two predicates:

- $P$: minimal pre-condition on the source constructs
- $Q$: maximal post-condition on the target constructs

$$\Sigma = <T,t> = <P,Q,t>$$
3. Transformations

The structural mapping of a transformation can be defined graphically:

\[
\begin{align*}
P & \quad \Rightarrow \quad Q \\
\text{CUSTOMER} & \quad \text{PHONE} \\
\text{Cust#} & \quad \text{Phone} \\
\text{Name} & \quad \text{id: Phone} \\
\text{Phone[0-5]} & \quad \text{id: Cust#} \\
\text{id: Cust#} & \quad \text{has} \\
0-5 & \quad 1-N
\end{align*}
\]
3. Transformations

Semantics-preserving transformations

- both constructs/schemas have the same information contents
- *examples*: mutation transformations
- property formally defined
- possible to prove that an arbitrary transformation is SP

Non semantics-preserving transformations

- loss of semantics or addition of (spurious) semantics
- *example*: set attribute $\Rightarrow$ array attribute
- often empirical
- often used in refinement (logical/physical design)
3. Transformations

Elementary semantics-preserving transformations

- mutation (*trans-gender*) transformations (attribute ↔ entity type, entity type ↔ relationship type, etc.)
- simplification transformation (ISA hierarchies, non-set constructs, etc.)
- others

Complex semantics-preserving transformations

- compound transformations: $\Sigma_{12} = \Sigma_2 \circ \Sigma_1$
- predicate-driven transformations: $\Sigma (p)$
- model-driven transformations: $\text{schema}_{M_1} \Rightarrow \text{schema}_{M_2}$
4. Main database engineering processes
4. Main database engineering processes (life cycle)

- Forward engineering
- Reverse engineering
- Reengineering
- Migration
- Integration
- Evolution
4. Main database engineering processes (life cycle)

Forward engineering
Design and implementation of a new database from users requirements
4. Main database engineering processes (life cycle)

Reverse engineering

Recovering the technical and conceptual specifications of an existing database.

Observations: only a subset of the conceptual specifications has been implemented in the DDL code, frequent idiosyncrasies, poor design, dead parts, redundant constructs, etc.

⇒ analysis of various information sources: DDL code, data dictionaries, application programs code, database contents, user interfaces, reports, program execution, etc., to get additional information of the actual constructs.

Two main problems

- recovering implicit constructs
- interpreting physical constructs
4. Main database engineering processes (life cycle)

Reverse engineering

Recovering the technical and conceptual specifications of an existing database

- Code DDL
- Code PRG

Database Reverse Engineering

- Conceptual schema

Reverse Engineering

- Parsing
  - Raw physical schema
- Refinement
  - Physical schema
- Cleaning
  - Logical schema
- Conceptualization
4. Main database engineering processes (life cycle)

Reengineering

Rebuilding (*rearchitecturing*) an existing database without changing its conceptual schema.
4. Main database engineering processes (life cycle)

Migration

Porting an existing database to another (generally more modern) technical platform

*Bad approach:* physical migration (straightforward, easy, cheap, unreadable, difficult to maintain)
4. Main database engineering processes (life cycle)

Migration

Porting an existing database to another (generally more modern) technical platform. Special case of evolution.

*Good approach*: conceptual migration (complex, expensive, clean result, easy to maintain)
4. Main database engineering processes (life cycle)

**Integration**

Deriving a new (physical or virtual) database from a collection of independent, autonomous, and heterogeneous databases

*Physical database:*

![Diagram showing integration process with BD1, BD2, and Integrator leading to BD 1+2](Diagram here)
4. Main database engineering processes (life cycle)

Integration

Deriving a new (physical or virtual) database from a collection of independent, autonomous, and heterogeneous databases

Virtual database:
4. Main database engineering processes (life cycle)

Integration

Engineering processes
4. Main database engineering processes (life cycle)

Evolution

Adapting an existing database to new functional or non-functional requirements

Example: evolution of functional requirements

Diagram:

- Conceptual schema 1
  - Database Reverse Engineering
  - code DDL1
- modification
- Conceptual schema 2
  - Database design
  - code DDL2
4. Main database engineering processes (life cycle)

Database Engineering Processes

- Each process basically is an inter-model compound transformation
- Thanks to the GER, database engineering processes mainly are intra-model transformation
- Most processes are semantics preserving transformations
- Formal approaches can be developed to model, develop and control Database Engineering processes

Consequences

- Properties can be proved (e.g., semantics preservation)
- Processed can be supported by tools
5. CASE tools
5. CASE tools

Observation

No methodologies can succeed without tools that support them

Requirements

- user-friendly
- performance: from small to very large complex schemas (100.000+ objects)
- modularity: basic platform + specific components
- wide spectrum: multiparadigm, multi-level
- extendable
- transformational
- DB-oriented input/output
5. CASE tools

DB-Main CASE environment

- user-friendly: various graphical/hypertext view
- performance: written in C++
- modularity: basic platform + specific components
- wide spectrum: GER
- extendable: repository + programming languages (Java, Voyager 2)
- transformational: transformation toolbox + transformation composers
- DB-oriented input/output: parsers, generators
6. Evolution & Coevolution
6. Evolution & Coevolution

• Context:
  • Multi-paradigm environment (e.g., database + programs + HMI)
  • Engineering processes based on a hierarchy of abstraction levels
  • Each abstraction level comprises several coordinated artefacts, each belonging to a definite paradigm
  • Artefacts of two successive levels are linked through mapping deriving from engineering (transformational) processes
  • Each artefact has multiple (evolvable) facets
    • Intrinsic properties: name, type, structure, constraints,…
    • Additional properties: annotations, views, instance samples, statistics,…
6. Evolution & Coevolution

• Evolution $\rightarrow$ modification of an artefact at a given level of abstraction

• Research Methodology:
  • Identification of artefacts modification primitives
  • Modification propagation based on inter-artefact mapping
  • Study of compositionality of modification propagation

• Horizontal (intra-level) co-evolution
  • Schema change $\rightarrow$ sub-schema change
  • Schema change $\rightarrow$ process change

• Vertical (inter-level) co-evolution
  • Conceptual schema change $\rightarrow$ logical schema change
  • Abstract process change $\rightarrow$ code change
6. Evolution & Coevolution

Database oriented Coevolution Scenarios

• The conceptual schema is modified due to a change in user requirements
  • What about the logical schema?
  • What about the data instances?
  • What about the client programs?
• The database platform is replaced due to technological modernization
  • What about the logical schema?
  • What about the data instances?
  • What about the client programs?
• The data resources of companies are merged due to corporate merging
  • How to restructure their databases?
  • How to adapt legacy programs?
  • How to develop new programs?
• The data instances change due to application domain instances evolution
  • How to record this evolution in the database?
  • How to rebuild the history of the application domain?
6. Evolution & Coevolution

LIDB Experience in (co)Evolution

- Evolution of Relational Database Applications
- Temporal Databases Engineering
  - TimeStamp project (1997-2001)
- Database Reverse Engineering
- Heterogeneous Database Federation
  - InterDB project & Ph. Thiran PhD thesis (2003)
- Taxonomies Evolution
  - Quetelet.net project (2003-2007)
- Legacy Database Migration
  - RIStart project (2004-2008)
7. Evolution of Relational Database Applications
7. Evolution of Relational Database Applications

- Coevolutions scenarios (inter-level + intra-level)

<table>
<thead>
<tr>
<th>Conceptual Modifications</th>
<th>Logical Modifications</th>
<th>Physical Modifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS0 to CS1</td>
<td>CS0 to CS1</td>
<td>CS0 to CS1</td>
</tr>
<tr>
<td>LD0 to LD1</td>
<td>LD0 to LD1</td>
<td>LD0 to LD1</td>
</tr>
<tr>
<td>LS0 to LS1</td>
<td>LS0 to LS1</td>
<td>LS0 to LS1</td>
</tr>
<tr>
<td>PS0 to PS1</td>
<td>PS0 to PS1</td>
<td>PS0 to PS1</td>
</tr>
<tr>
<td>P0 to P1, D0 to D1</td>
<td>P0 to P1, D0 to D1</td>
<td>P0 to P1, D0 to D1</td>
</tr>
</tbody>
</table>

- Typology of schema transformations according to
  - the type of object involved: entity type, attribute, constraint …
  - the nature of the modification: creation, destruction, restructuring, renaming …
  - the (non-)preservation of different properties: type, semantics, …
7. Evolution of Relational Database Applications

\[ CE1 = \langle\text{work,EMPLOYEE}\rangle \langle-\text{modify-Role(work,EMPLOYEE,1-N)}\rangle \]

\[ LD0 + PD0 = \langle\text{EMPLOYEE,\{NumP\}}\rangle \langle-\text{RT-to-FK(work)}\rangle \]

\[ PE1 = \langle\text{WORK}\langle-\text{FK-to-ET(EMPLOYEE,\{NumP\})}\rangle\text{fkproject}\langle-\text{create-FK(WORK,\{NumP\},}\rangle\text{(PROJECT.NumP)})\rangle\text{fkemployee}\langle-\text{create-FK(WORK,\{NumE\},}\rangle\text{(EMPLOYEE.NumE),equal})\rangle \]

-- Check new constraint:
SELECT * FROM EMPLOYEE WHERE NumP is NULL;
-- 1. WORK <- FK-to-ET(EMPLOYEE,\{NumP\})
   -- Create table WORK :
   CREATE TABLE WORK (NumP NUMERIC(5) NOT NULL,NumE NUMERIC(5) NOT NULL,CONSTRAINT IDWORK PRIMARY KEY(NumP,NumE));
   CREATE UNIQUE INDEX IDXWORK ON WORK (NumP,NumE);
   CREATE INDEX IDXPROJECT ON WORK (NumP);
   -- Instance transfert:
   INSERT INTO WORK (NumP,NumE) (SELECT Nump,NumE FROM EMPLOYEE);
   -- Delete column NumP of table EMPLOYEE:
   ALTER TABLE EMPLOYEE DROP NumP;
-- 2. fkproject <- create-FK(WORK,\{NumP\},\{PROJECT.NumP\})
ALTER TABLE WORK ADD CONSTRAINT FKPROJECT FOREIGN KEY (NumP) REFERENCES PROJECT (NumP);
-- 3. fkemployee <- create-FK(WORK,\{NumE\},\{EMPLOYEE.NumE\},equal)
ALTER TABLE WORK ADD CONSTRAINT FKEMPLOYEE FOREIGN KEY (NumE) REFERENCES EMPLOYEE (NumE);
ALTER TABLE EMPLOYEE ADD CONSTRAINT EQWORK CHECK(EXISTS(SELECT * FROM WORK WHERE WORK.NumE = EMPLOYEE.NumE));
8. Legacy Database Migration
8. Legacy Database Migration

• Migration = particular case of evolution
  • Source and target schema convey the same information content
    ➔ Automated program conversion is easier
• Propagation at the program level to preserve
  • Structural consistency: use the target database structure
  • Language consistency: use the target data manipulation language
• RIStart approach: combination of generative & transformational techniques
  • Schema conversion through database reverse engineering (DBRE)
  • Wrapper-based program conversion
8. Legacy Database Migration

Schema Conversion through DBRE

- Recover the complete conceptual schema
  - enriched with implicit structures and constraints elicited through program and data analysis
- Design the target schema through standard methodology
8. Legacy Database Migration

![Diagram showing the process from conceptualization to design with refinement steps.]

**Conceptualization**
- Customer (CUS)
  - C-CODE
  - C-DESCR
  - NAME
  - ADDR
  - FUNCT
  - REC-DATE
  - id: C-CODE
- Product (PROD)
  - P-CODE
  - P-NAME
  - P-LEVEL
  - id: P-CODE
- Orders (ORD)
  - O-CODE
  - O-CUST
  - O-DETAIL
  - id: O-CODE
- Detail (DETAIL)
  - PROD_CODE
  - ORD_CODE
  - QUANTITY
  - id: CODE

**Refinement**
- Customer (CUS)
  - C-CODE
  - C-DESCR
  - NAME
  - ADDR
  - FUNCT
  - REC-DATE
  - id: C-CODE
- Product (PROD)
  - P-CODE
  - P-NAME
  - P-LEVEL
  - id: P-CODE
- Orders (ORD)
  - O-CODE
  - O-CUST
  - O-DETAIL
  - id: O-CODE
- Detail (DETAIL)
  - PROD_CODE
  - ORD_CODE
  - QUANTITY
  - id: CODE

**Design**
- Customer (CUS)
  - C-CODE
  - C-DESCR
  - NAME
  - ADDR
  - FUNCT
  - REC-DATE
  - id: C-CODE
- Product (PROD)
  - P-CODE
  - P-NAME
  - P-LEVEL
  - id: P-CODE
- Orders (ORD)
  - O-CODE
  - O-CUST
  - O-DETAIL
  - id: O-CODE
- Detail (DETAIL)
  - PROD_CODE
  - ORD_CODE
  - QUANTITY
  - id: CODE

Legacy Database Migration
8. Legacy Database Migration

Wrapper-based Program Adaptation

• Wrapper
  • simulates the legacy data management system (COBOL, CODASYL)
  • on top of the new database (Oracle, DB2,...)

• Wrapper generated from
  • source-to-target schema mappings
  • source-to-target DML translation rules
    • COBOL-to-SQL & CODASYL-to-SQL
Industrial Application

- Migration project by Rever s.a.
- Legacy system running on a Bull Mainframe (GCOS8)
- 2300 COBOL programs, over 2 million LOC
- Goal =
  - migrate a legacy database (CODASYL-IDS/II) to a relational platform (DB2)
  - while preserving the functionalities of the application programs
- Methodology
  - DBRE-based schema conversion
  - Wrapper-based program conversion
8. Legacy Database Migration

Industrial Application

- DBRE-based program conversion

<table>
<thead>
<tr>
<th></th>
<th>Physical</th>
<th>Refined</th>
<th>Conceptual</th>
<th>Relational</th>
</tr>
</thead>
<tbody>
<tr>
<td># ent. types</td>
<td>231</td>
<td>231</td>
<td>156</td>
<td>171</td>
</tr>
<tr>
<td># rel. types</td>
<td>213</td>
<td>213</td>
<td>90</td>
<td>0</td>
</tr>
<tr>
<td># attributes</td>
<td>648</td>
<td>11157</td>
<td>2176</td>
<td>2118</td>
</tr>
<tr>
<td>max # att./ent.</td>
<td>8</td>
<td>104</td>
<td>61</td>
<td>94</td>
</tr>
</tbody>
</table>

- Close to fully-automated program conversion
  - 98% of source code files automatically converted
  - 92% of IDS/II primitives automatically converted

<table>
<thead>
<tr>
<th></th>
<th>Migrated</th>
<th>Manually transformed</th>
</tr>
</thead>
<tbody>
<tr>
<td># programs</td>
<td>669</td>
<td>17</td>
</tr>
<tr>
<td># copybooks</td>
<td>3917</td>
<td>68</td>
</tr>
<tr>
<td># IDS/II verbs</td>
<td>5314</td>
<td>420</td>
</tr>
</tbody>
</table>
9. LIBD contribution to MoVES
9. LIBD contribution to MoVES

• WP2: Modelling languages and Restructuring
  • Modelling language
    • Multiple models, multiple abstraction levels
  • Transformational/MDE approach to restructuring

• WP4: Consistency checking & Coevolution
  • Typology of consistency relationships
  • Classification of coevolution scenarios
  • Impact analysis in a coevolution context
  • Classification of propagation strategies
  • Automated evolution propagation
    • Generative & transformational techniques

+ possible contributions to other WPs