

# iRM: A MOF-based Repository System with Querying Capabilities

Ilia Petrov, Marc Holze, Gabor Nemes, Markus Schneider, Stefan Jablonski

<http://www6.informatik.uni-erlangen.de/research/projects/irm/>

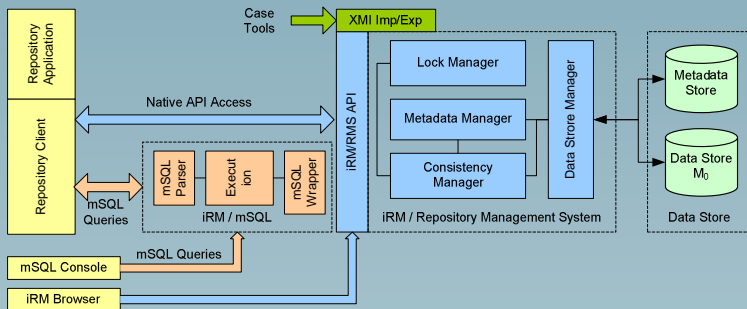
[ilia.petrov@informatik.uni-erlangen.de](mailto:ilia.petrov@informatik.uni-erlangen.de)

## Repository Systems

- Database Systems and metadata
- Repositories - Data stores with custom defined system catalogue
  - Layered metadata architecture →  $M_0, M_1, M_2, M_3$
  - Custom-defined system catalogue
- Main Contributions
  - Declarative Query Language
  - Structural consistency - consistency across metadata layers
    - Repository transactions
- iRM – research prototype of a repository system

## Why is querying metadata repositories an issue?

- Retrieve the set of repository object meeting a **range of criteria**
  - Example: give me the all of  $M_1$  classes instances of  $M_2$ :Table and their attributes having data type VarChar2 (Poster B)
  - The application developer should provide code fragments for:
    - Iteration: enumerate all repository objects
    - Filtering: evaluate the concrete criteria against each object
  - **Issues in favour:** Usability, Code Maintenance, Complexity, Resource Consumption, Performance
- Requirements
  1. Dual/Uniform treatment of data and metadata
  2. Model independent querying and information discovery
  3. Easy to use and expressive



## Kinds of iRM / mSQL Variables and Variable Declaration

- Declarative power of QL → FROM clause variable declaration
- iRM/mSQL utilizes **several kinds of variables** (SQL has only one)
  - Query models, model elements and instances
  - *Metalevel::Package:Class.Attribute^DataType*
  - Use SchemaSQL's arrow notation to declare a new variable
  - Package Variable:  $M_2::->p$
  - Class Variable:  $M_2::RDBPackage2->c$  or  $p->c$
  - Instance Variables:  $M_2::RDBPackage2->c\ ci$  or  $c\ ci$  or  $p\ pi$
  - Structural feature variable:  $M_2::RDBPackage2:Table->a$  or  $c->a$

## Why is consistency an issue?

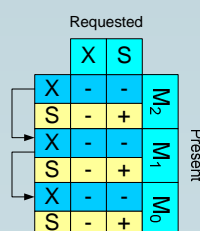
- Application1 deletes an attribute of a super-class of class C in an  $M_2$  model
- ⇒ Result 1: The system does not automatically delete attribute from the sub-classes, the direct instances and the sub-classes' instances
- Application2 instantiates one of the sub-classes of C providing value for the deleted attribute

## Atomicity

- Cooperative Atomicity – One logical operation comprises multiple iRM/RMS API operations. Example: Create a class in a package
- A demarcation mechanism needed → BeginRT / EndRT

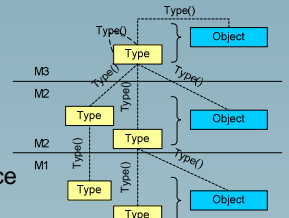
## Isolation

- Locking
  - Extension of the multi-granularity locks
  - Short S and long X locks, 2PL compatible
  - Instance lattice
- A lock on an object on  $M_n$  can be placed if:
  - No dependent object on any underlying  $M_{n-p}$
  - Is locked with an incompatible lock



## Querying Model Elements

- Packages, Classes, Attributes and their Instances
- Query generalization hierarchy: Sub(), Super(), ONLY()
- Type() and Object() to convert an instance data to type definitions



## iRM/mSQL Declarative Reflection, Introspection, Discovery

- Schema Exploration:  $M_2::->p, p->c, c->a^d$
- Schema Discovery:  $M_2::RDBPackage2->c\ ci, ci->ai^di, type(di)$

## iRM/mSQL Attributes and Attribute Types

- Normal Attributes
  - Give me all attributes of a certain class (Q12, poster B)
  - **Major issue:**
    - Classes are complex objects, i.e. attributes separate objects
    - How to distinguish classes from attributes → abstract model
- Pseudo Attributes
  - Virtual attributes of a repository object
  - Name, Level, MofID

## iRM/mSQL Sub-Queries

- Not-Correlated / Correlated / FROM clause sub-queries

## iRM/mSQL Query Results: JDBC-like API

- We use iRM/mSQL ResultSet similar to JDBC ResultSet

- Application3 counts the attributes of all  $M_1$  classes
- ⇒ Result 2: Inconsistent → non-serializability. Dirty Read, Unrep.Read
- Solution:** Enforce consistency within repository a transaction!

## Consistency

- Well-Formedness → Syntax conformity, Immediate constraints
- Structural consistency → Do objects conform to their type definition
  - Creation: do newly created objects conform to their meta-objects
    - Create new attribute in a class / Create new instance
  - Modification:
    - Can all instance objects be adapted to the modified meta-obj.
    - Change generalization or containment hierarchy
    - Change of an attribute's datatype / Change association end
  - Deletion
    - Delete an attribute / Delete a class

## Durability

- Rely on the transactional capabilities of the data store