“Write Once, Deploy N”

a Performance Oriented MDA Case Study

Pieter Van Gorp and Dirk Janssens
Formal Techniques in Software Engineering
Pieter.VanGorp@ua.ac.be, Dirk.Janssens@ua.ac.be
http://www.fots.ua.ac.be/

Tracy Gardner (IBM)
tgardner@uk.ibm.com
http://www.hursley.ibm.com/
Presentation Roadmap

1. Research Context: Problem and proposed solution in a nutshell

2. MDA Today

3. The “WODN” MDA Case Study
   - Case study: why?
     ✓ MDA is young, needs more (and more complex) case studies
     ✓ Basis for assessing quality of tools and QVT submissions

4. Conclusions & Future Work
Part I:
Problem and solution in a nutshell
Problem Statement

Optimizing the performance of distributed database applications is hard to combine with middleware vendor independence since cache, transaction and cluster configuration is database and application server specific.

= Part of this decade’s integration challenge as explained by keynote of Prof. Douglas C. Schmidt
Platform Independence?
Component Model Essentials

- **Distributed Server Components**
  - Memory Management, Persistence, Caching, Connection Management, Transaction Management, Object Distribution, … as *container services*

- **Example: Enterprise JavaBeans**
  - Java sources inherit / implement certain interfaces
    - Remote Home Intf, Local Home Intf
    - Remote Bean Intf, Local Bean Intf
    - Bean Class (focus on business logic)
  - Deployment Attributes for code generators / compilers
    - XML Deployment Descriptors (tuning!!!)
  - Final component accesses server specific API
    - Callbacks to Bean Class

- **Popular alternative: Hibernate**
  - Same principles (interfaces and descriptors)
Proposed Solution

1. **Model** platform independent business components
   - ✔ PIMs

2. **Generate** platform specific implementations
   - ✔ May be interactive wizard (e.g. point to database)
   - ✔ PSMs, “PSC”

3. **Generate** platform independent “wrapper” code
   - ✔ Generate “PIC”

4. **Write** applications using these components
   - ✔ Plain Java (depends on your high-level API)
   - ✔ or visual behavioral language (e.g., SDM)

5. **Analyze** access scenario of such applications
   - ✔ Model analysis

6. **Generate** delegation code (component discovery)

Optimizing performance = tuning code generator ➔ Requires good transformation language!
Part II:
Solution Space:
MDA
✓ Mainstream and
✓ next-generation tools
» Standards
Existing MDA Tools

Input Model for “contracts” package (PIM)

---

Generated Source for ContractBean.java (PSC)

```java
package org.andromda.samples.carrental.contracts;

import java.lang.Object;
import javax.ejb.EntityBean;

public abstract class ContractBean extends java.lang.Object implements javax.ejb.EntityBean {
    public abstract org.andromda.samples.carrental.customers.Driver getDriver();
}
```
How to realize research goals?

- Performance optimization
- Platform independence

» Adapt the code generator!
package $packagename;

import javax.ejb.EntityContext;
import javax.ejb.RemoveException;

public abstract class ${entityname}BeanImpl extends ${entityname}Bean {
    private EntityContext context;

    public void setEntityContext(EntityContext ctx)
    {
        //Log.trace("${class.name}Bean.setEntityContext...");
        context = ctx;
    }

    public void unsetEntityContext()
    {
        //Log.trace("${class.name}Bean.unsetEntityContext...");
        context = null;
    }

    public void ejbRemove() throws RemoveException
    {
        //Log.trace(
        //      "$\{class.name\}Bean.ejbRemove...");
    }
}
#foreach ( $op in $class.operations )
#if ($transform.getStereotype($op) == "FinderMethod")
  * @ejb.finder signature="${transform.findFullyQualifiedName($op.getType())}
  ${transform.getOperationSignature($op)}"
#set($viewtype = "")
#set($viewtype = $transform.findTagValue($op.taggedValues, "@andromda.ejb.viewType"))
#if($viewtype == "local" || $viewtype == "remote" || $viewtype == "both")
  * view-type="$viewtype"
#end
#set($querystring = "")
#set($querystring = $transform.findTagValue($op.taggedValues, "@andromda.ejb.query"))
#if($querystring == "")
#set($querystring = "SELECT DISTINCT OBJECT(c) FROM $class.name AS c")
#if($op.parameters.size() >0 )
#set($querystring = "$\{querystring\} WHERE")
#foreach($prm in $op.parameters)
#set($querystring="$\{querystring\} c.$prm.name = ?$velocityCount")
#if($velocityCount != $op.parameters.size())
#set($querystring = "$\{querystring\} AND")
#end
#end
#end
#end
* query="$querystring"
Evolving today’s MDA tools

➢ Problem
  Input Metamodel (UML) is too General Purpose
  (Too abstract for code generation)

➢ Solution:
  1. Code Templates on very concrete Metamodels
     ✓ Refactor input metamodel to stack of metamodels
  2. Solution: Stepwise refinement
     ✓ Refactor code templates to model transformations

➢ This requires “vertical exogenous” model to model transformations
MDA Standards

➢ Definition
  - Model-Driven Engineering with UML and MOF
    ✓ UML: Widely known notations for diagrams (Visualize your models)
    ✓ MOF: Repository standard (Store your models)
    ✓ QVT: Model transformation standard
    ✓ M2T: Code template standard

➢ Goal
  - Standard MDD
    ✓ Tools
    ✓ Education, ...
  - Managing Evolution of Framework Standards
    ✓ Beyond J2EE
      • Best practices as first class programming artifacts before in standard
      • Your *company standards* for what IBM, BEA, SUN, ... should not standardize
    ✓ Beyond MOF (!!)
Model Transformation Today...

- Submissions to QVT RFP
  - See T. Gardner et al. and K. Czarnecki et al.

- Research Languages and Tools
  - Triskell: MTL
  - Lina: ATL
  - ISIS: GReAT (GME)
  - Segravis: SDM (Fujaba)
  - BME–MIT: VIATRA
  - ...

- QVT:
  - What do they have in common?
  - How can vendors compete?
Declarative VS Imperative

[ T. Gardner et al. ]

- **Declarative**
  - No explicit execution step specification
  - Relationships between variables
  - Engine infers result from the relation

- **Imperative**
  - Explicit state manipulation

- **Hybrid**
  - Combination of declarative and imperative
Problem with QVT RFP

- No *concrete* use cases
  - Therefore, no example of
    “a relation between variables”
  - Unclear
    ✓ what “declarative” means
    ✓ why it is relevant

- Solution: case studies
  - More (horizontal *and* vertical transformations)
  - More realistic (e.g. performance-oriented)
Part III:

The “WODN”

MDA Case Study
Write Once, Deploy N

- Online Data Access Scenario’s
  - 85% Read for Display:
    - Lazy Loading
    - Invalidations from Writers
    - No Transactions
  - 10% Read–Write:
    - Aggressive Loading
    - Transaction Support
  - 5% Batch Update:
    - Lazy Loading
    - Transaction Support
WODN as and MDA Case Study

- Server Component Deployment
  - A “Deploy 1” Component would need conservative deployment attributes and waste resources!
    ✓ Manual Implementation is tedious
    ✓ Attributes vary per vendor

- MDE Approach:
  - Generate the 3 deployments
  - Hide clients from this optimization
Part III: Suggested solution to the case study
Domain Specific Metamodels

- Metamodel for business analysis (PIM)
- Metamodel for transactional caching
- Metamodel for object-to-relational mapping
- Metamodel for WL EJB (PSM1)
- Metamodel for JBoss EJB (PSM2)
Model Transformation Process
Metamodels for “WODN” (I/III)

Note MM Reuse
Metamodules for “WODN” (II/III)
Metamodels for “WODN” (III/III)
An interpretation of “Declarative”

Most *declarative* transformation description
= the consistency constraint that it establishes !!!

```java
Transformation Data2TransactionCaching {
  // models as structural features (attributes)
  Data d; // metamodel as static type of model
  TransactionalCaching tc; // model as container of model elements (8)
  // rules as behavioral features (operations)
  Rule Entity2RO_Component () {
    postcondition: // a specific invariant is established
    d.Entity.allInstances->forAll(e |
      tc.Component.allInstances->exists(c |
        e.Classifier = c.Classifier and
        c.lockingStrategy = LockingStrategy::noLock and
        c.transactionDemarcation = tc.TransactionDemarcation::optional
      )
    )
  }
  ...
}
```

✔ Role of the transformation language *engine*...
✔ Framework needs to be *hybrid*
Therefore, a transformation engine should be integrated with a consistency checker

✓ Review of ArcStyler tool
  • User experience: OK
  • Developer experience: Problematic
What about “Imperative” languages?

- Why Imperative?
  - For complex automatic reconciliation
  - Less needed as engines mature

- What language can be used?
  - Fujaba’s SDM,
  - GME’s GReAT,
  - ATL,
  - Java,
  - ...

A QVT Standard within months…?

- Initially,
  - QVT standard should NOT standardize the imperative language
  - MDA tool vendors can compete on transformation engines that automatically satisfy complex consistency constraints!

- After a couple of years,
  - OMG can standardize
    ✓ The most popular imperative transformation language
    ✓ Popular constraint predicates (like “isIsoMorphicTo”)
Transformation Language Requirement

- **Integration with code templates**
  - No separate “model to text” RFP! (ArcStyler review…)
  - WODN code generator requires polymorphism (AndroMDA review…)

- **Advantage:** smooth visualization in model transformation process
Concrete integration example from WODN

```java
abstract Transformation Components2xDoclet {
    TransactionalCaching tc;
    ORDB o;
    Text code;
    Rule Generate() {
        postcondition:
        tc.Component.allInstances->forAll(
            c | o.Entity.allInstances->forAll(
                e | (c.Classifier = e.Classifier)
                implies (
                    code.File.allInstances->exists(
                        // ignore directories in this example
                        f | f.name= (c.name + ".java") and
                            f.content= Join2JavaFile(c, e)
                    )
                )
            )
        )
    }
}```
Polymorphism in Code Template Rules: Platform Specialization without Code Duplication

Polymorphic execution = Declarative: engine executes concrete transformation rule from dynamic type of transformation

abstract Transformation Components2xDoclet {
  ...
  Rule Join2JavaFile (TransactionalCaching::Component c, ORDB::Entity e): Text::String {
    // code template fragment for Java imports
    // code template fragment for conventional Javadoc
    #call Join2ClassTags(c,e);
    ...
    // code template fragment for iterating over methods
    ...
    #call Join2MethodTags(c,e);
    ...
  }
}

abstract Rule Join2ClassTags (TransactionalCaching::Component c, ORDB::Entity e): Text::String {
  // code template for vendor independent xDoclet class
  // tags like @ejb.finder, @ejb.home, @ejb.interface,
  // ...
  }
}
Concrete rule in subtype transfo

abstract Transformation Components2WL
inherits Components2xDoclet {

    // Join2JavaFile inherited, not overridden

    Rule Join2ClassTags
        ( TransactionalCaching::Component c,
          ORDB::Entity e): Text::String {
        #call super.Join2ClassTags(c,e);
        // code template for WebLogic specific
        // xDoclet class tags like
        // @weblogic.persistence, @weblogic.cache,
        // @weblogic.invalidation-target, ...
    }

    ...
}

Polymorphic execution= Declarative: engine executes concrete transformation rule from dynamic type of transformation
Part IV:
Conclusions & Future Work
WODN: Lessons learned for QVT

Solution to declarative<>imperative dilemma:
- Most **declarative description** of a transformation = consistency constraint
- Standardize framework for **automatic constraint solvers** instead of one unproven language
  - OCL for consistency constraints
  - Proprietary languages for **imperative transformation fragments**
    - JMI could already be used as a standard language!
- **Vendors can compete** on power of transformation engines
- Future versions of spec. can standardize most popular imperative language

Integrate QVT and M2T from the beginning
- Today’s middleware code is text! MDA emerged there!
- Most popular MDA tools have mature code template support
WODN: Modeling transformations

- Model transformation process
  - Use activity diagrams

- Transformation structure and Variability of code generator
  - Use class diagrams
Future Work

- Implement reference framework with maximal reuse of existing software (migration effort assessment)
  - Approach 1:
    - Combine OCL software with transformation software
    - E.g. OCL Toolkit (or AndroMDA) combined with JCMTG (or VIATRA)
  - Approach 2:
    - Use software supporting OCL and transformations
    - ATL or GReAT

- Add model transformations from EJB model elements to plain object model elements
  - Enables low-level style checking
  - Parser => Bidirectionality

- Add more platforms support to code generator lib!
- Investigate polymorphism of consistency rules