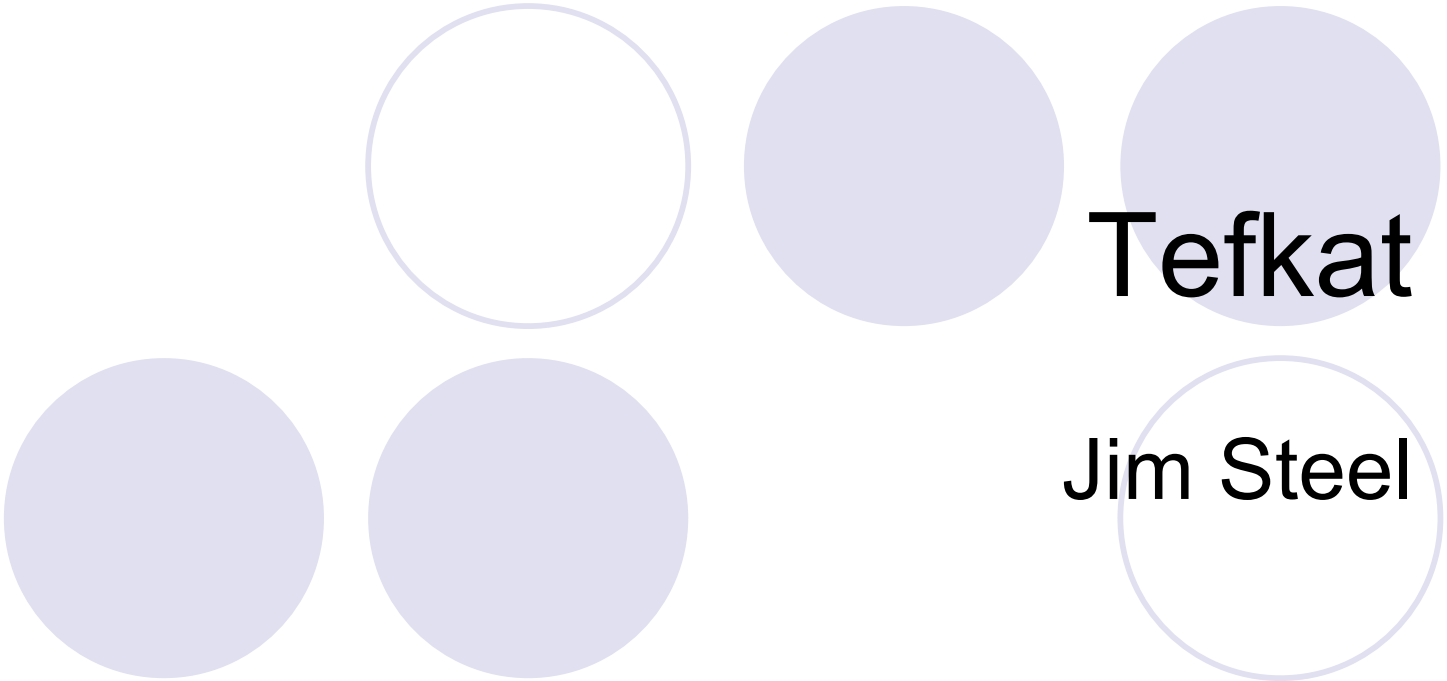

Tefkat

Jim Steel

The plan

- Introduction/background
- Language Features
- Mandatory Example
- Implementation highlights
- Status

Introduction/background

- 5 years of model transformations
  - GenGen, based on CWM, java black-boxes
  - MofLog, syntactic extension to F-Logic
  - QVT submission (DSTC/IBM/CBOP)
  - The Engine Formerly Known As Tarzan
- Model Transformation has specific issues
  - Recurring patterns and practices
  - Not a general programming problem

Language Overview (1)

- “Declarative” (in the parlance of our times)
  - Source and target constraints upon:
    - Existence of object instances in an extent
    - Type of objects
    - Value/s of object features
    - Relative order of values in an object’s feature
    - Relationship between values (traceability)

Language Overview (2)

- Notable:
  - No traversal order, no execution order
  - Constructive
  - Not designed for in-place updates
  - Change propagation can be treated using model-merge (Metke ‘05)
  - Separation of abstract & concrete syntax

Language Overview (3)

- 3 types of Extent
  - Source (match)
  - Target (constrain)
  - Tracking (match & constrain)
- Rules
  - A pair of constraints to match and constrain/enforce
  - No explicit invocation

Transformation

- Transformation
  - Name, parameters, imports

```
TRANSFORMATION c_to_r: cls -> rel

IMPORT http:///mtip05/class.ecore
IMPORT http:///mtip05/rdbms.ecore
```

Class Definitions

- For tracking relationships between source and target extents
- Defined inline or imported (e.g. for larger scale trace models)

Rules

- Action elements of the transformation
- 2 constraints - match & constrain - that share variables

```
RULE ClassAndTable(C, T)
  FORALL Class C {
            is_persistent: true;
            name: N; }
  MAKE Table T { name: N; }
  LINKING ClsToTbl WITH class = C, table = T;
```

Rules (2)

- We can also use rules to enforce preconditions/well-formedness rules, with a target constraint FALSE

```
RULE constraint_reflexive_non_persistent
  FORALL Class C
  WHERE C.is_persistent = false
    AND ClassHasReference(C, C, _)
    AND println("Found a non-persistent class in
  relation (by association or attribute) with
  itself: ", C)
  SET FALSE;
```

Patterns & Templates

- Named, parameterised, reusable constraints
  - Patterns for source, templates for target
  - Allows for recursion

```
PATTERN ClassHasSimpleAttr(Class, Attr, Name, IsKey)
  FORALL Class Class {
           attrs: Attribute Attr {
             type: PrimitiveDataType _PT;
             name: Name;
             is_primary: IsKey;
           };
         };
```

Trackings

- Track mapping relationships between source and target elements
- Allows for loose coupling of rules
- Allows for decoupling of rules that need a relationship from the rules that establishes it

FROM

- Injections to control the number of objects created
- Creates one unique object for each unique tuple given by the FROM
- If absent, there is an implicit injection:
  - Named for the rule and target variable
  - Parameters are the source variables
- Decoupling -> Maintainability, Reusability

The Example: Summary

- Tracking classes:
  - ClsToTbl
  - AttrToCol
- Constraint rules:
  - Only root classes may be persistent
  - No reflexive relations for non-persistent classes

The Example: Patterns

- Abstractions for related classes, attributes
- Find the root class
- Does a class “have” an attribute
  - Simple attributes
  - Included attributes
  - Attributes via subclasses

The example: Rules

- Create tables and trace from the class
- Create column, set pkey and trace from the attribute
- Make and link foreign keys

- About 100 lines of code

Notes: Spanning meta-levels

- Cases are few but very useful/important
- Reflection
  - Normal MOF reflection
  - Embedded expressions
    - Prefix $ allows the use of expressions where a literal is expected (variables, type names)
- Any Type: _
- Paper contains generic copy in 27 lines

Notes: Syntax

- Separate concrete & abstract syntax
  - SQL-inspired concrete syntax
- Object Literals
  - Syntactic sugar to replace constraints with object fragments
- Variable naming
  - _ for “Don’t Care” variables
  - Warnings for variable usage

Notes: The Engine

- Standalone option
- Eclipse-based
  - Syntax-highlighting editor with linked feedback for errors & warnings, outline view
  - Source-level debugger
  - Build system
    - Transformation applications
    - URI mappings
  - Pragmatics: println, continue despite failure, java invocation (dangerous)

Notes: Stratification

- Rules must be stratified
  - I.e. a rule cannot depend on its own negation
  - E.g. cannot check for existence of a target object and then create it
  - Hence no-check on target models. Tracking hopefully allows a happy medium
  - Investigate streaming (serial transformations) as a solution

Evaluation

- Large-scale evaluation
  - Generation of test frameworks from UML diagrams (Dai ‘04)
  - Model-merge for change propagation (Metke ‘05)
  - Health Record translation and Xform generation
    - Very large, many models, many subtleties
- Open-source under investigation

Conclusion

The goal is to allow the user to focus on ***what*** the transformation does, not ***how*** it does it.

For more…

http://www.dstc.edu.au/Research/Projects/Pegamento/tefkat/

Or just google for ‘tefkat’