Testing Model Transformations

Amr Al Mallah

Based on: Validation in Model-Driven Engineering: Testing Model Transformations by Franck Fleurey, Jim Steel, Benoit Baudry

Presentation outline

- Model Driven Architecture
  - Models and meta models
  - Model transformations
  - Case study (UML to RDBMS transformation)
- Testing Model Transformations
- Functional Criteria (black box)
  - Meta model coverage
    - Partition analysis
    - UML to MOF meta-models
    - Representative values
    - Coverage items and test criterion
    - Summery and limitation

Presentation outline

- Effective Meta-Model
- Static Analysis (white box)
  - Effective Meta-Model
  - Representative values
- Automatically generating models
  - Systematic approach
    - Building models
    - Building the solution set
    - Discussion
  - Bacteriologic approach
  - Related Work

OO Vs MDA

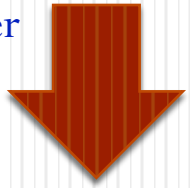
Problem Domain

Problem Domain

OO Vs MDA

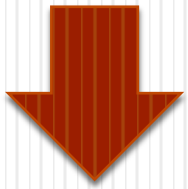
Problem Domain

Programmer



UML

Programmer

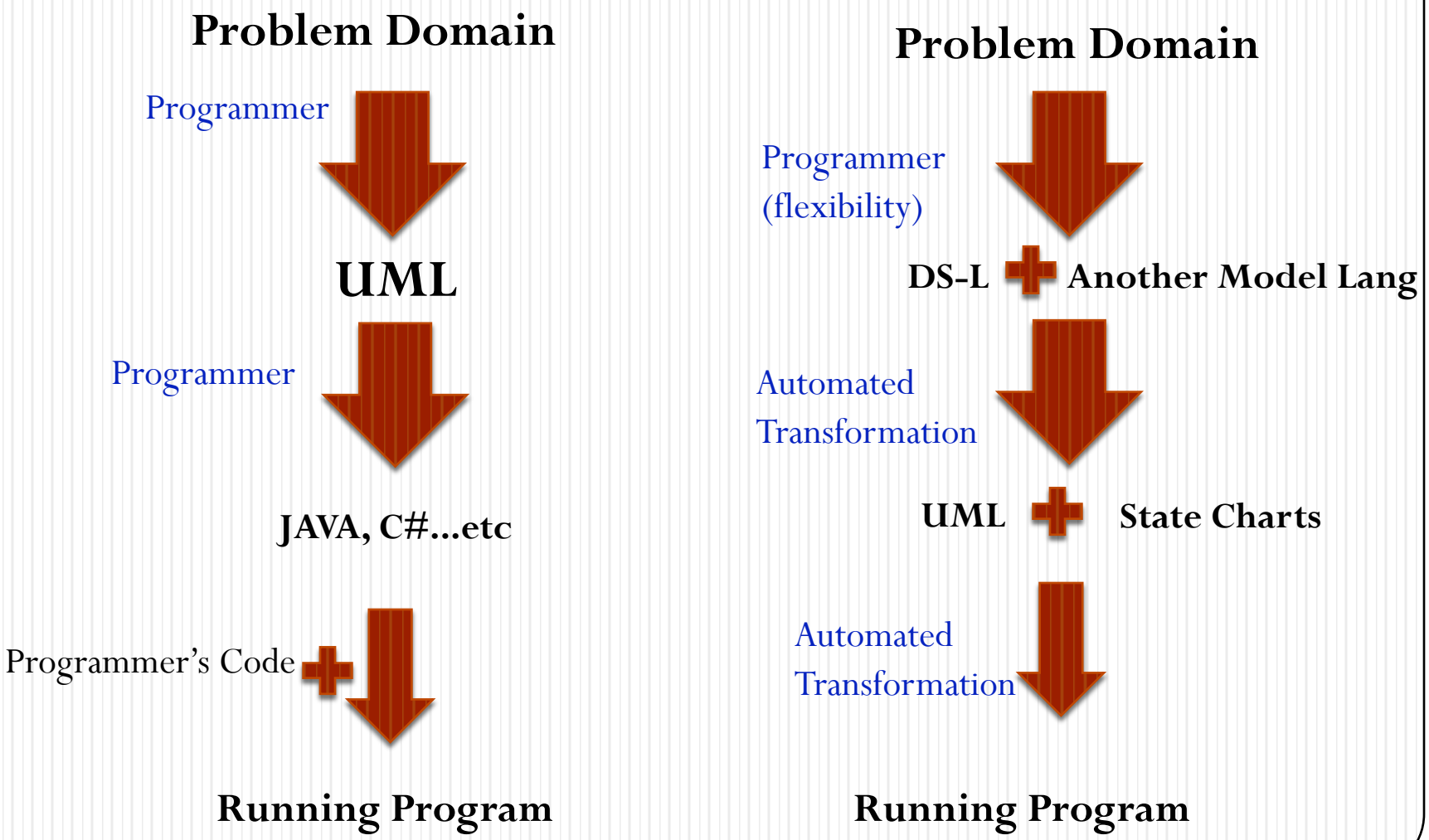


JAVA, C#...etc

Programmer's Code

Running Program

OO Vs MDA


Problem Domain
Programmer
UML
Programmer
JAVA, C#...etc
Programmer's Code
Running Program
Problem Domain
Programmer
(flexibility)
DS-L
Another Model Lang
Automated
Transformation
UML
State Charts
Automated
Transformation
Running Program

Meta Models

- Expressed using MOF(meta object facility).
- Define Possible models instances structure.

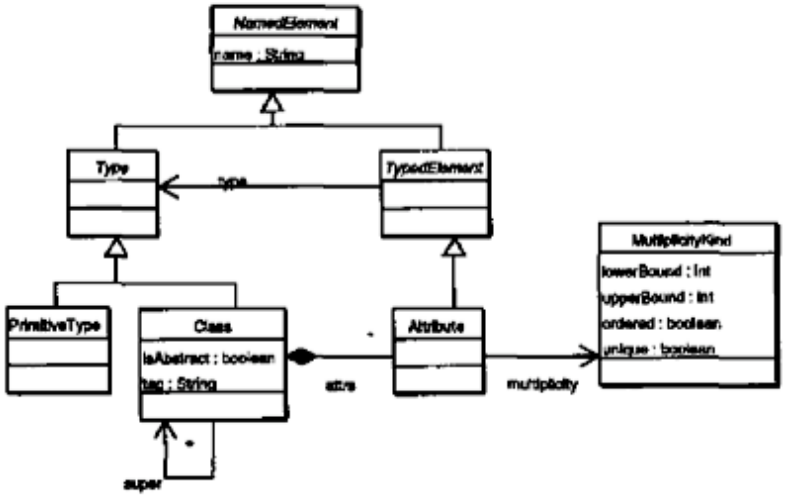


Figure 1. Extract of UML Meta-model

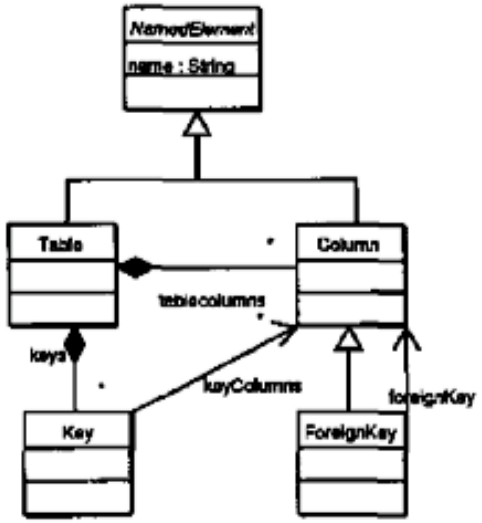


Figure 4. Meta-model for RDBMS Schemas

Model Transformations

- Model to model transformation
  - Traffic to Petri nets
- Model behavior (Graph rewriting , running programs ?)
  - Pac man movement

- Different formalisms can be used to describe and implement the transformation.
  - Atom3 : LHS ,RHS rules and some python code.
  - Motif : DEVS (PyDEVS, JavaDEVS….etc).
  - When to use what ! Like choosing a programming language .

Model Transformations, M to M

- Relate different meta models elements relationships (UML class to a RDBMS table).
- MT specifications VS MT implementation programs.
- Implementations examples:
  - JAVA
  - Domain specific languages (extensions of OCL)
  - Rules evaluation
  - DEVS
- For the example OCL in pre and post conditions, is considered as part of the specification

Model Transformations

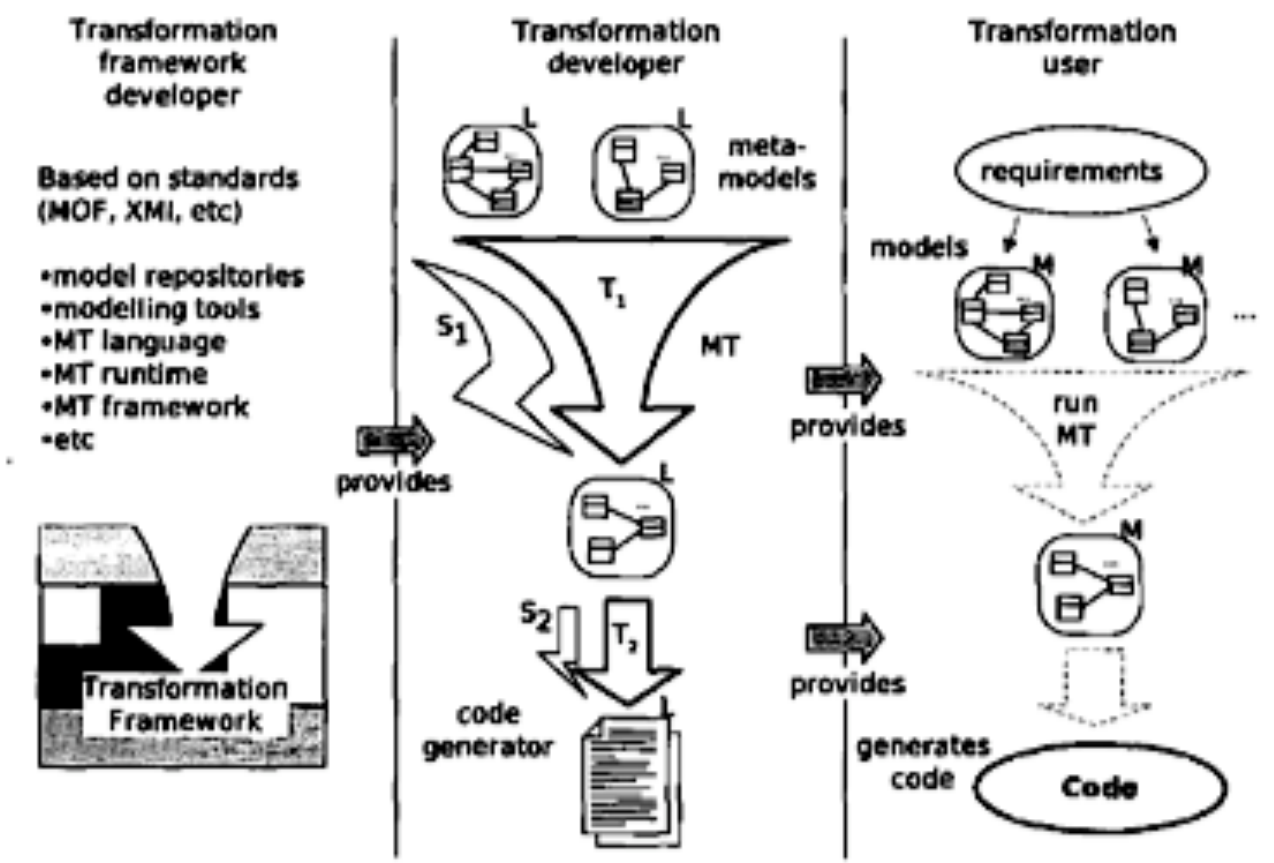


Figure 2. Separation of roles in Model-Driven Engineering

Model Transformations

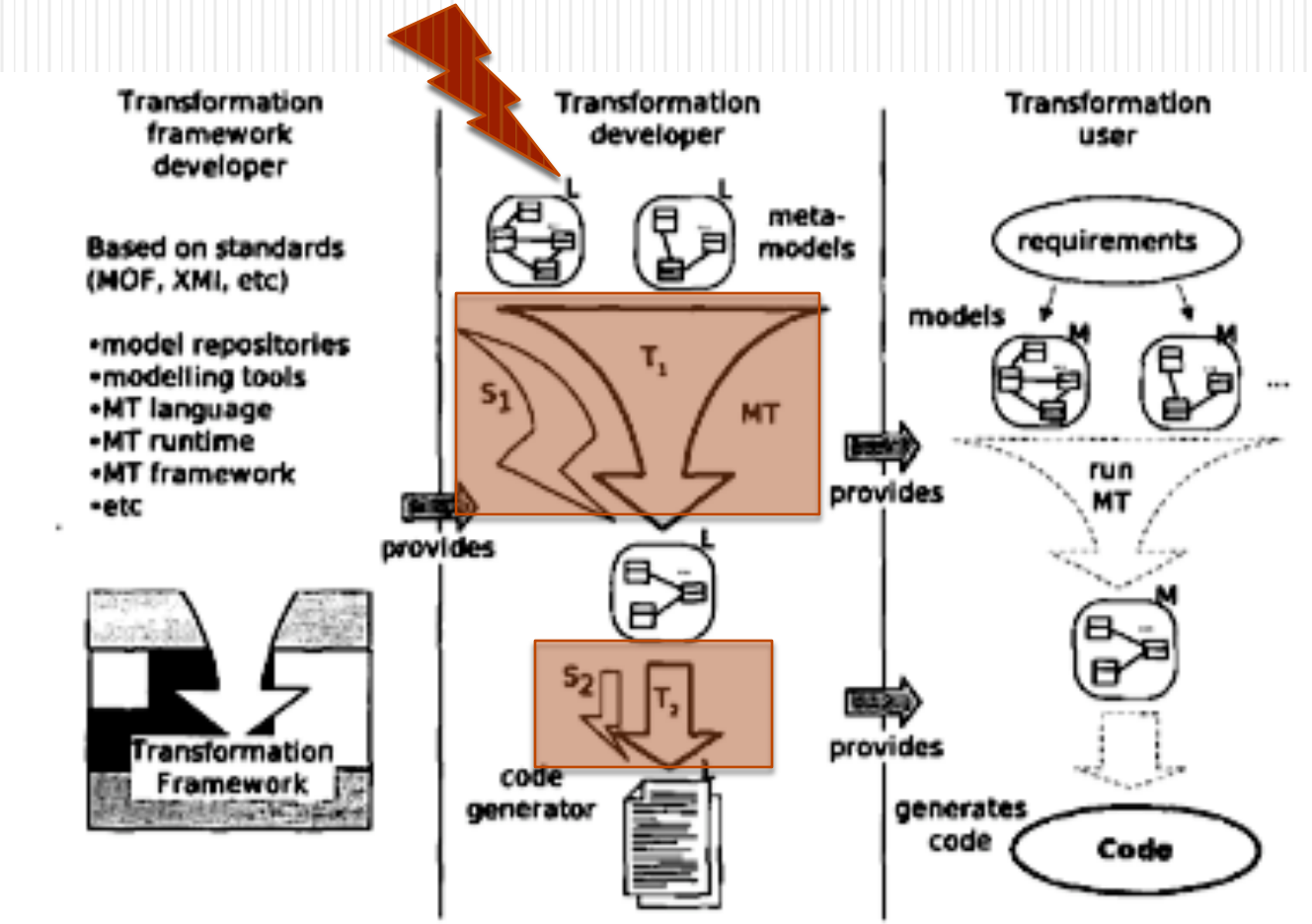


Figure 2. Separation of roles in Model-Driven Engineering

Testing Model Transformations

- An adequate testing process will reveal the following:
  - Errors in the implementation
  - Unspecified behavior due to incomplete specifications
  - Confidence in the quality of the implementation

- Two approaches to generate test data:
  - White box : based on the structure of the implementation
  - Black box : or functional, which is entirely based on the specifications
- Existing Vs Needed test generation techniques.

UML to RDBMS Transformation Specification

- **Description**: UML classes tagged as persistent a table with the same name is created, and for each attribute the class has a column is created.

UML to RDBMS Transformation Specification

- **Description**: UML classes tagged as persistent a table with the same name is created, and for each attribute the class has a column is created.
- **Parameters**: This transformation has two parameters, in parameters that takes in a UML model, and an out parameters that contains an RDBMS model .

UML to RDBMS Transformation Specification

- **Description**: UML classes tagged as persistent a table with the same name is created, and for each attribute the class has a column is created.
- **Parameters**: This transformation has two parameters, in parameters that takes in a UML model, and an out parameters that contains an RDBMS model .
- **Pre-condition**: expressed in OCL, it imposes a condition on the input model that each class name is unique.

UML to RDBMS Transformation Specification

- **Description**: UML classes tagged as persistent a table with the same name is created, and for each attribute the class has a column is created.
- **Parameters**: This transformation has two parameters, in parameters that takes in a UML model, and an out parameters that contains an RDBMS model .
- **Pre-condition**: expressed in OCL, it imposes a condition on the input model that each class name is unique.
- **Post-condition**: also expressed in OCL, it specify the result of the transformation. It states the for each persistent tagged class in input model a corresponding table exists with columns for the attributes , and they have the names.

Black box approach

- Based on the specifications.
- A top level specification formalism that could be used:
  - data structures are described as meta-models
  - transformations are operations with pre- and post- conditions.
- A partition analysis approach becomes applicable.

Black Box approach, M-M coverage

- Meta-Model coverage:
  - Data-centric Vs Control-centric.
  - Partition analysis.
    - Input classification.
    - Example: Program takes integer (X) as input a possible partition would be :

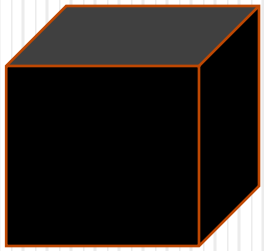
      {min, [min+1…-1],0,[1…max-1],max},

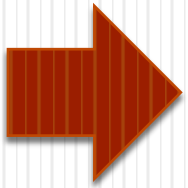
      but if a post condition of the program is (X > 2) , then a better partition would be:

      {min,[min+1….1],2,[3….max-1],max}

Black Box approach, m-m coverage

Input Model





Output Model

Meta-Model A
expressed in MOF

Black box
UML2RDBMS

Meta-Model B

Black Box approach, m-m coverage

- MOF are similar to UML class diagrams (classes, attributes, generalizations, associations)
- Use existing techniques to cover class diagrams :
  - Association end multiplicities : for each association each representative multiplicity pair should be covered.
  - Class Attribute: for each attribute each representative value must be covered
  - Generalization: each sub type must be covered.
- The Generalization rule doesn't apply in the case of testing transformation, since the focus is on the structure of the models rather than the behavior.

Black Box approach, m-m coverage

- Representative value can be found using two techniques:
  - Default partitioning :
    - Based on the structure or the type of data .
    - Ex : for a string {null,", "something" , "a_very_long_string"}
    - Ex: for [0..1] multiplicity => {0,1}
    - This can be automated using policies.
  - knowledge based partitioning :
    - Extracted from the specification (still a black box!)
    - In particular from the pre and post conditions.
    - Harder to automate.
- Combining the two results in a better set of values.

Black Box approach, M-M coverage

Meta-Model element	Representative values
Class::name : String	Null, '' , 'something'
Class::isAbstract : Bool	True, False
Class::tag : String	Null, '' , 'something' , **'CMP'**
Class -> attribute : [0..*]	[0] ,[1], [>1]
Attribute::name : String	Null, '' , 'something'

- These representative values need to be combined to be sufficient in covering all cases.
- We combine them into Coverage items .Which are constraints on input models of an MTP.
- Some of these constraints maybe not applicable

Black Box approach, M-M coverage

- Summary:
  - Find representative values for attributes and associations of the input meta model .
  - Take the Cartesian product of each class and association.
  - Use pre and post conditions to remove invalid combinations.
- Limitations:
  - The amount of non-relevant test cases will be significant.
  - Because not all the meta model elements may be relevant for a specific transformation
  - Ex : UML meta model in the case of UML2RBDMS transform.

Black Box approach, Effective M-M

- Determine a subset of the meta model that is relevant .
- A pragmatic approach is to use the specifications and the OCL pre/post conditions.
- UML2RDBMS example :

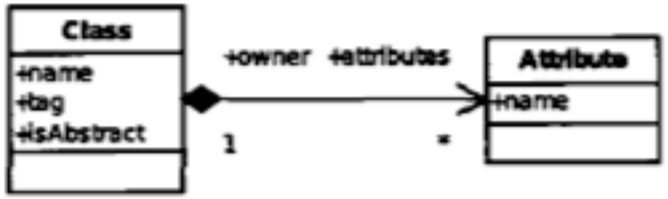


Figure 7. Effective meta-model

Black Box approach, Effective M-M

Table 2. Coverage items

	Cri.	Element	Values
1	CA	Class	Null, *, *
9	CA	Class	'', True, Null
10	CA	Class	'', True, ''
11	CA	Class	'', True, 'something'
12	CA	Class	'', True, 'CMP'
13	CA	Class	'', False, Null
14	CA	Class	'', False, ''
15	CA	Class	'', False, 'something'
16	CA	Class	'', False, 'CMP'
17	CA	Class	'something', True, Null
18	CA	Class	'something', True, ''
19	CA	Class	'something', True, 'something'
20	CA	Class	'something', True, 'CMP'
21	CA	Class	'something', False, Null
22	CA	Class	'something', False, ''
23	CA	Class	'something', False, 'something'
24	CA	Class	'something', False, 'CMP'
25	CA	Attribute	Null
26	CA	Attribute	''
27	CA	Attribute	'something'
28	AEM	Class->Att.	[1-0]
29	AEM	Class->Att.	[1-1]
30	AEM	Class->Att.	[1-(>1)]

For the class "Class" the sequence of values {x,y,z} corresponds respectively to the values of the attributes Class::name, Class::isAbstract and Class::tag.

Black Box approach, Effective M-M

Table 2. Coverage items

	Cri.	Element	Values
1	CA	Class	Null, *, *
9	CA	Class	'', True, Null
10	CA	Class	'', True, ''
11	CA	Class	'', True, 'something'
12	CA	Class	'', True, 'CMP'
13	CA	Class	'', False, Null
14	CA	Class	'', False, ''
15	CA	Class	'', False, 'something'
16	CA	Class	'', False, 'CMP'
17	CA	Class	'something', True, Null
18	CA	Class	'something', True, ''
19	CA	Class	'something', True, 'something'
20	CA	Class	'something', True, 'CMP'
21	CA	Class	'something', False, Null
22	CA	Class	'something', False, ''
23	CA	Class	'something', False, 'something'
24	CA	Class	'something', False, 'CMP'
25	CA	Attribute	Null
26	CA	Attribute	''
27	CA	Attribute	'something'
28	AEM	Class->Att.	[1-0]
29	AEM	Class->Att.	[1-1]
30	AEM	Class->Att.	[1-(>1)]

For the class "Class" the sequence of values {x,y,z} corresponds respectively to the values of the attributes Class::name, Class::isAbstract and Class::tag.

White Box approach

- Using static analysis of the MTP, few criteria can be enhanced :
  - Effective Meta model can be extracted by looking the Implementation .
    - Compare to the specification meta model . Validation & Verification.
    - Applicability of transformation to an evolved Meta model.
  - Representative values:
    - Algorithm can look for values used in assignments or comparisons to attributes, and use them in the representative values.

Automatically generating models

- Issues when generating models from meta models is :
  - Which classes to generate ?
  - What values the attributes should take ?

Automatically generating models

- Issues when generating models from meta models is :
  - Which classes to generate ?
  - What values the attributes should take ?
  - It should be guided by the previous analysis (coverage items)
  - Systematic Vs Bacteriologic approach .

Automatically generating models

- Systematic approach:
  - Based on building models and instantiating the attributes to include as much coverage items as possible .
  - Iterative procedure :
    - Generate a model that includes one coverage item and instantiate it.
    - Generate any other elements that the generated model may depend on.
    - Instantiate the attribute values by first looking at the items in the coverage list and choose what allows for more coverage .(uncovered).
    - The process repeats until no more models need to be generated (all coverage items have been covered)
  - The problem is that test models depend highly on the order of the coverage items were picked. And may not be optimal.

Automatically generating models

- Bacteriological approach:
  - An existing initial set of test cases (models) should be present.
  - At each iteration the algorithm:
    - Ranking: the current test cases are ranked according to their potential overall contribution to the coverage of the solution set. This can be the number of unfulfilled coverage items that are covered by the test case.
    - Memorization: during this step, the best test case is added to the solution set using a selection policy .
    - Filtering: involves removing useless test cases from the candidate set to keep the size reasonable.(example the ones that don't contribute)
    - Mutation: this steps create new test cases , where the best test cases are selected and a mutation operator is applied to them to create new test cases .

Automatically generating models

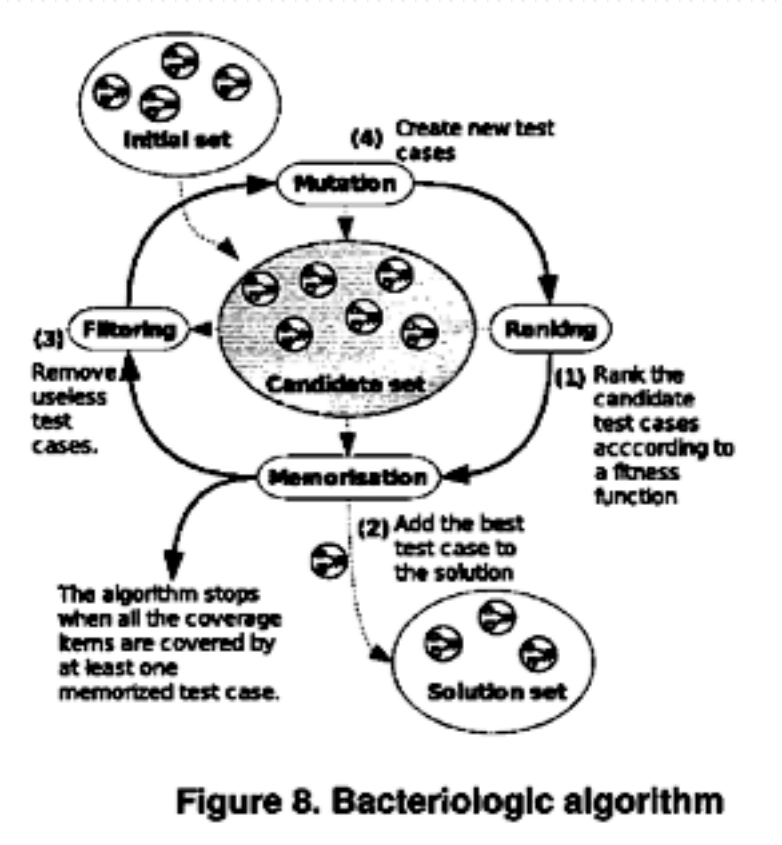


Figure 8. Bacteriologic algorithm

Key challenges

- Automatic generations of test data and oracle
- Comparing expected Vs actual (model comparison)
  - Model difference
  - Semantic equivalence.