Techniques for Debugging Model-Transformation Failures

Saurabh Sinha
Pankaj Dhoolia, Senthil Mani, Vibha Sinha, Mangala Gowri
Model Transformation

- A *transform* is an application that converts a model to another model or to text
  - A model is a set of structured data captured in a well defined notation
  - Text output could be configuration files, code, XML, etc.
Model Transformation: Failing Executions

- What is wrong in the input model that caused the failure? (Fault localization)
- How can the faulty model be fixed? (Fault repair)
- Is an input model valid for a transform? (Model validation)
Investigating Model-Transformation Failures

- **Transform-user perspective: Goal is to locate input-model faults**
  - Some faults can be detected automatically (e.g., faults that violate metamodel constraints)
  - Other faults cannot be detected using model validators

- **Limitations of conventional fault-localization techniques**
  - Most techniques focus on program faults
  - Some techniques identify failure-relevant inputs (delta debugging, Penumbra)

- **Model traceability techniques not applicable to a large class of input-model faults**
  - Faults that cause an incorrect path to be traversed in the failing execution
  - Faults that result in missing output entities
  - “Missing input-model entity” faults
Techniques for Investigating Model-Transformation Failures

- Combination of static analysis and dynamic analysis
  
  - Static analysis for model validation
    - Infers code-level constraints from the transform code
    - Maps constraints to metamodel-level rules
    - Rules can be used to construct model validators
      
      Demystifying model transformations: An approach based on automated rule inference. OOPSLA 2009
  
  - Dynamic analysis for fault localization
    - Performs dynamic taint analysis to track flow of information from input model to output
    - Enables iterative fault localization on the input model
      
      Debugging model-transformation failures using dynamic tainting. ECOOP 2010
  
  - Dynamic analysis for fault repair
    - Collects metadata about accesses to model entities, conditionals, and loops
    - Performs pattern analysis over output taint log
      
      Automated support for repairing input-model faults. ASE 2010 (under review)
Outline of the Talk

- Static analysis for model validation
- Dynamic-tainting-based fault localization
  - Failure scenarios, example
  - Description of technique
  - Empirical evaluation
- Dynamic analysis for fault repair
- Summary and future work
Static Analysis for Model Validation: Overview of Approach

Step 1: Constraint Inference

- Transform
- Filters
- Precondition Analyzer
- Exception constraints
- Output constraints
Example Model-to-Model Transform

```xml
<model>
  <artifacts name="BusinessProcess">
    <attributes name="id" />
    <attributes name="name" type="integer"/>
  </artifacts>
</model>
```

```java
public void execute( EObject source, EObject target )
{
    1. Attribute attr = (Attribute)source;
    2. Property prop = (Property)target;
    3. PrimitiveType ptype = null;
    4. if ( attr.getName() != null ) {
        5. String type_src = attr.getType();
        6. if ( type_src.equals("String") )
            7. ptype = UMLUtilities.findType(...);
        8. if ( ptype != null )
            9. prop.setType(ptype);
    ...
```
Example Model-to-Model Transform

```xml
<model>
  <artifacts name="BusinessProcess">
    <attributes name="id" />
    <attributes name="name" type="integer"/>
  </artifacts>
</model>
```

```java
public void execute( EObject source, EObject target )
{
  1. Attribute attr = (Attribute)source;
  2. Property prop = (Property)target;
  3. PrimitiveType ptype = null;
  4. if ( attr.getName() != null ) {
      5. String type_src = attr.getType();
      6. if ( type_src.equals("String") )
          7. ptype = UMLUtilities.findType(...);
      8. if ( ptype != null )
          9. prop.setType(ptype);
...}
```
Failing Execution

Input Model

```xml
<model>
	<artifacts name="BusinessProcess">
		<attributes name="id" />
		<attributes name="name" type="integer" />
	</artifacts>
</model>
```

Null Pointer Exception

```java
public void execute( EObject source, EObject target )
{
    1. Attribute attr = (Attribute)source;
    2. Property prop = (Property)target;
    3. PrimitiveType ptype = null;
    4. if ( attr.getName() != null ) {
        String type_src = attr.getType();
        5. if ( type_src.equals("String") )
            6. ptype = UMLUtilities.findType(...);
        7. if ( ptype != null )
            8. prop.setType(ptype);
    ...
```
Precondition Analysis: Exception Constraints

```java
public void execute( EObject source, EObject target )
{
    1. Attribute attr = (Attribute)source;
    2. Property prop = (Property)target;
    3. PrimitiveType ptype = null;
    4. if ( attr.getName() != null ) {
        5.     String type_src = attr.getType();
        6.     if ( type_src.equals("String") )
            6.         ptype = UMLUtilities.findType(...);
        7.     if ( ptype != null )
            8.         prop.setType(ptype);
        }
    }
}
```
Precondition Analysis: Exception Constraints

```java
public void execute( EObject source, EObject target )
{
    1. Attribute attr = (Attribute)source;
    2. Property prop = (Property)target;
    3. PrimitiveType ptype = null;
    4. if ( attr.getName() != null ) {
        5.    String type_src = attr.getType();
        6.    if ( type_src.equals("String") )
        7.        ptype = UMLUtilities.findType(...);
        8.    if ( ptype != null )
        9.        prop.setType(ptype);
    }
}
```

5. `attr != null`  
   attr.getType() = null

6. `type_src = null`

© 2009 IBM Corporation
Precondition Analysis: Exception Constraints

```java
public void execute( EObject source, EObject target )
{
    1. Attribute attr = (Attribute) source;
    2. Property prop = (Property) target;
    3. PrimitiveType ptype = null;
    4. if ( attr.getName() != null ) {
    5.       String type_src = attr.getType();
    6.       if ( type_src.equals("String") )
    7.             ptype = UMLUtilities.findType(...);
    8.       if ( ptype != null )
    9.             prop.setType(ptype);
    }
}
```
Precondition Analysis: Exception Constraints

```java
public void execute( EObject source, EObject target )
{
    1. Attribute attr = (Attribute)source;
    2. Property prop = (Property)target;
    3. PrimitiveType ptype = null;
    4. if ( attr.getName() != null ) {
        5.     String type_src = attr.getType();
        6.      if ( type_src.equals("String") )
        7.      ptype = UMLUtilities.findType(...);
        8.      if ( ptype != null )
        9.      prop.setType(ptype);
    }
}
```

1. `source.getName() != null`  
2. `source != null`  
3. `source.getType() = null`  
4. `attr.getName() != null`  
5. `attr != null`  
6. `attr.getType() = null`  
7. `type_src = null`

- **Null pointer Exception at Line 6**
Static Analysis for Model Validation: Overview of Approach

Step 1: Constraint Inference
- Transform
- Precondition Analyzer
- Filters
- Exception constraints
- Output constraints

Step 2: Rule Generation
- Input metamodel
- Output metamodel
- Rule Generator
- Validation rules
- Querying rules

Step 3: Validation & Comprehension
- Input model
- Validity Checker
- Violated rules
- Querying Tool
- Query results
- Output model
- User queries
Inference of Validation Rules

- Total of 369 validation rules
  - 306 rules for null-pointer exceptions, 60 rules for class-cast exceptions, 3 rules for array-index exceptions
On average, the participants spent

- 62 – 78% of the time in fixing the input model **without the rules**
- 38 – 22% of the time when **using the rules**
Transformation-Failure Scenarios

Fault Localization

Failure scenarios  Technique  Empirical

Fault

Failure

Fault propagation

Incorrect value

Incorrect string
Transformation-Failure Scenarios

Fault

Fault propagation

Failure

Incorrect value

Incorrect string

Failure scenarios

Model Validation

Fault Localization

Fault Repair

Future Work

Technique

Empirical

© 2009 IBM Corporation
Transformation-Failure Scenarios

Fault

Fault propagation

Failure

Incorrect value

Incorrect string
Transformation-Failure Scenarios

Fault Localization

Failure Scenarios

Fault

Incorrect value

Missing entity

Fault propagation

1

2

3

4

5

6

7

8

Fault

Incorrect string

Missing string

Failure

Incorrect value

Missing entity

Incorrect path

Incorrect value over correct path

Incorrect string

Missing string

Model Validation

Fault Repair

Future Work

Technique

Empirical

© 2009 IBM Corporation
Transformation-Failure Scenarios

Fault

1. Incorrect value
2. Missing entity

Fault propagation

3. Incorrect value
4. Incorrect string
5. Missing entity
6. Incorrect path
7. Incorrect value over correct path
8. Incorrect path

Failure

1. Incorrect string
2. Missing string

Model Validation  Fault Localization  Failure scenarios  Technique  Empirical  Fault Repair  Future Work

© 2009 IBM Corporation
Example Model-to-Text Transform

```plaintext
Function main()
    foreach model element in the input file do
        invoke ApplyModel()

Function ApplyModel(Element model)

[1]  foreach property element in model do
[2]      if property.isGen = "nameValue" then
[3]          write content of property.name
[4]          write string literal "="
[5]          write content of property.val
[6]          write newline
[7]      else if property.isGen is not null then
[8]          write content of property.def
[9]          write newline
```
Example Model-to-Text Transform

```plaintext
Function main()
    foreach model element in the input file do
        invoke ApplyModel()
    endforeach

Function ApplyModel(Element model)

    foreach property element in model do
        if property.isGen = "nameValue" then
            write content of property.name
        else if property.isGen is not null then
            write content of property.def
        else
            write new line
        endif
    endforeach

name1=value1 value2=value2
```
Failure-inducing Input

```python
Function main()
  foreach model element in the input file do
    invoke ApplyModel()

Function ApplyModel(Element model)

[1] foreach property element in model do
[2]   if property.isGen == "nameValue" then
[3]     write content of property.name
[4]     write string literal "="
[5]     write content of property.val
[6]     write newline
[7]   else if property.isGen is not null then
[8]     write content of property.def
[9]     write newline

name1=value1
default2
```

```
Dynamic Analysis for Fault Localization: Overview of Approach

- **Taint Initialization**
  - Associate taint marks with input-model entities

- **Taint Propagation**
  - Propagate taint marks to the output string
  - Classify taint marks (data, control, loop)

- **Taint-log Analysis**
  - Compute the fault space incrementally
Taint Initialization

```
model
  property
    isGen nameValue
    name name1
    val value1
    def default1
  property
    isGen something
    name name2
    val value2
    def default2
```
Taint Propagation (Data Taints)

Model Validation  Fault Localization  Fault Repair  Future Work
Failure scenarios  Technique  Empirical

Function main()
    foreach model element in the input file do
        invoke ApplyModel()
    
Function ApplyModel(Element model)
[1]    foreach property element in model do
[2]        if property.isGen = “nameValue” then
[3]            write content of property.name
[4]        write string literal “=”
[5]        write content of property.val
[6]        write newline
[7]        else if property.isGen is not null then
[8]            write content of property.def
[9]            write newline

name1 = value1 
default2 

name2 = value2 

name = name1
value = value2
Taint Propagation (Data Taints)

Function main()
   foreach model element in the input file do
      invoke ApplyModel()

Function ApplyModel(Element model)
[1]   foreach property element in model do
[2]      if property.isGen = "nameValue" then
[3]         write content of property.name
[4]      end if
[5]      write content of property.val
[6]      write newline
[7]   else if property.isGen is not null then
[8]      write content of property.def
[9]      write newline

name1 = value1 \ndefault2 \n
Taint Propagation (Data Taints)

Function main()
    foreach model element in the input file do
        invoke ApplyModel()

Function ApplyModel(Element model)
[1]    foreach property element in model do
[2]        if property.isGen = "nameValue" then
[3]            write content of property.name
[4]            write string literal "="
[5]            write content of property.val
[6]            write newline
[7]        else if property.isGen is not null then
[8]            write content of property.def
[9]            write newline

name1 = value1
\ndefault2\n
name2 = something
\ndefault2
Taint Propagation

- **Data taint marks**
  - Create traceability for value propagation from input model to output model
  - Propagated at each assignment statement and statement that constructs the output string

- **Control taint marks**
  - Create traceability for input model entities that influence the outcome of predicates
  - Propagated at conditional statements
Taint Propagation (Control Taints)

Function main()
    foreach model element in the input file do
        invoke ApplyModel()
Function ApplyModel(Element model)
[1]    foreach property element in model do
[2]        if property.isGen = “nameValue” then
[3]            write content of property.name
[4]            write string literal “=”
[5]            write content of property.val
[6]            write newline
[7]        else if property.isGen is not null then
[8]            write content of property.def
[9]            write newline

name1 = value1 \ndefault1 \n
name2 = value2 \ndefault2 \n
Taint Propagation (Control Taints)

Function main()
    foreach model element in the input file do
        invoke ApplyModel()

Function ApplyModel(Element model)
    foreach property element in model do
        if property.isGen = "nameValue" then
            write content of property.name
            write string literal "="
            write content of property.val
            write newline
        else if property.isGen is not null then
            write content of property.def
            write newline

name1 = value1
default2

Model Validation
Fault Localization
Failure scenarios
Technique
Empirical
Fault Repair
Future Work
Taint Propagation (Control Taints)

Function main()
    foreach model element in the input file do
        invoke ApplyModel()
    Function ApplyModel(Element model)
        foreach property element in model do
            if property.isGen = “nameValue” then
                write content of property.name
            write string literal “=”
            write content of property.val
            write newline
            else if property.isGen is not null then
                write content of property.def
            write newline
            name1 = value1 \n            default2 \n            value2
Taint Propagation (Control Taints)

Function main()
    foreach model element in the input file do
        invoke ApplyModel()
    Function ApplyModel(Element model)
        [1] foreach property element in model do
        [2] if property.isGen = "nameValue" then
        [3] write content of property.name
        [4] write string literal "="
        [5] write content of property.val
        [6] write newline
        [7] else if property.isGen is not null then
        [8] write content of property.def
        [9] write newline

name1 = value1
\ndefault2
\n
Taint Propagation

- **Data taint marks**
  - Create traceability for value propagation from input model to output model
  - Propagated at each assignment statement and statement that constructs the output string

- **Control taint marks**
  - Create traceability for input model entities that influence the outcome of predicates
  - Propagated at conditional statements: based on hammock decomposition of the control-flow graph

- **Loop taint marks**
  - Create traceability for input model entities that represent “collections”
  - Propagated at looping constructs
Taint Propagation (Loop Taints)

```
Function main()
    foreach model element in the input file do
        invoke ApplyModel()
    Function ApplyModel(Element model)
        foreach property element in model do
            if property.isGen = "nameValue" then
                write content of property.name
                write string literal "="
                write content of property.val
                write newline
            else if property.isGen is not null then
                write content of property.def
                write newline
        end
    end
end
```

```
| name1 | value1 | \n | default2 | \n | t4,d   | t5,d   | t11,d |
|-------|-------|-----|
| t3,c  | t8,c  |
| t2,c  | t7,c  |
| t1,l  |
```
Taint-Log Analysis for Incremental Fault Localization

Error marker (incorrect substring)
Incremental Expansion of the Fault Space

- **Incorrect substring**
  - Initial fault space: Start at a non-empty data taint
  - Fault-space expansion: Iteratively identify enclosing control taints (in reverse order of scope nesting)

- **Missing substring**
  - Initial fault space: Start at an empty data taint or an empty control taint
  - Fault-space expansion: Iteratively identify enclosing control taints (in reverse order of scope nesting)
Fault Localization for Missing Substrings

Function main()
    foreach model element in the input file do
        invoke ApplyModel()

Function ApplyModel(Element model)
[1]    foreach property element in model do
[2]        if property.isGen = "nameValue" then
[3]            write content of property.name
[4]            write string literal "="
[5]            write content of property.val
[6]            write newline
[7]        else if property.isGen is not null then
[8]            write content of property.def
[9]            write newline

    name1 = value1 
    name2 = value2
Fault Localization for Missing Substrings

```plaintext
Function main()
    foreach model element in the input file do
        invoke ApplyModel()

Function ApplyModel(Element model)
[1]    foreach property element in model do
[2]        if property.isGen = "nameValue" then
[3]            write content of property.name
[4]            write string literal "="
[5]            write content of property.val
[6]            write newline
[7]        else if property.isGen is not null then
[8]            write content of property.def
[9]            write newline
```

```
<table>
<thead>
<tr>
<th>name1 = value1</th>
</tr>
</thead>
<tbody>
<tr>
<td>t4,d</td>
</tr>
<tr>
<td>t5,d</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>name2</th>
</tr>
</thead>
<tbody>
<tr>
<td>t3,c</td>
</tr>
<tr>
<td>t2,c</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>name1</th>
</tr>
</thead>
<tbody>
<tr>
<td>t3</td>
</tr>
<tr>
<td>t4</td>
</tr>
<tr>
<td>t5</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>value1</th>
</tr>
</thead>
<tbody>
<tr>
<td>t4</td>
</tr>
<tr>
<td>t5</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>default1</th>
</tr>
</thead>
<tbody>
<tr>
<td>t6</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>default2</th>
</tr>
</thead>
<tbody>
<tr>
<td>t7</td>
</tr>
<tr>
<td>t8</td>
</tr>
<tr>
<td>t9</td>
</tr>
<tr>
<td>t10</td>
</tr>
</tbody>
</table>
```

© 2009 IBM Corporation
Fault Localization for Missing Substrings

Function main()
    foreach model element in the input file do
        invoke ApplyModel()

Function ApplyModel(Element model)
[1] foreach property element in model do
[2] if property.isGen = "nameValue" then
[3]   write content of property.name
[4]   write string literal "="
[5]   write content of property.val
[6]   write newline
[7] else if property.isGen is not null then
[8]   write content of property.def
[9]   write newline

name1 = value1 \n
<table>
<thead>
<tr>
<th>t1</th>
<th>model</th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td>property</td>
</tr>
<tr>
<td>t3</td>
<td>isGen  nameValue</td>
</tr>
<tr>
<td>t4</td>
<td>name name1</td>
</tr>
<tr>
<td>t5</td>
<td>val value1</td>
</tr>
<tr>
<td>t6</td>
<td>def default1</td>
</tr>
<tr>
<td>t7</td>
<td>property</td>
</tr>
<tr>
<td>t8</td>
<td>name name2</td>
</tr>
<tr>
<td>t9</td>
<td>val value2</td>
</tr>
<tr>
<td>t10</td>
<td>def default2</td>
</tr>
</tbody>
</table>

© 2009 IBM Corporation
Fault Localization for Missing Substrings

```
model
  property
    isGen nameValue
    name name1
    val value1
    def default1
  property
    name name2
    val value2
    def default2
```

```
name1 = value1
```

Error marker (incorrect substring)
Implementation for XSL-based Transforms

Process & Artifacts:
- XSLT
- Translet (Java Program)
- Control taint instrumented Program
- Control and data taint instrumented Program
- Taint Log
- Taint Index

Our Components:
- Control taint Instrumenter
- Taint Aspects
- Taint API

Out Of Box:
- Apache XSLTC
- Apache BCEL
- WALA
- AspectJ
- Apache XSLTC
- JDOM

Technique:
- Empirical

Failure scenarios
- Fault Localization
- Fault Repair
- Future Work

Model Validation
- Failure scenarios
- Technique
- Empirical
Empirical Evaluation

- **Two studies**
  - Fault-space reduction
  - Significance of control taints

- **Experimental subjects**
  - Six XSL transforms: Java classes, Java interfaces, configuration file, property file
  - 376–13270 Java bytecode instructions
  - Input model size: 38000–40000 entities

- **Faulty input generation**
  - Data mutation on valid inputs: four mutation operators
  - 913 faulty inputs
Study 1: Fault-Space Reduction

- Maximum reduction achieved for
  - 468 (51%) of the 913 faulty inputs
  - At least 50% of the faulty inputs for four of the subjects

- Better than 94% reduction for all subjects and faulty inputs
Study 2: Significance of Control Taints

- Overall subjects, 83% of the faulty inputs required control-taint propagation
- For one subject, all faults required control-taint propagation
Automated Support for Fault Repair

- **Metadata collection**
  - Entity accesses
  - Conditional statements
  - Selections in loops

- **Fault-index analysis**
  - Analyzes metadata associated with the fault index
  - Computes repair actions

- **Pattern analysis**
  - Identifies output fragments that are similar to the incorrect output fragment
  - Computes repair actions (based on the metadata) that will lead to a different output at the error marker
Summary and Future Work

- Techniques for debugging model-transformation failures
  - Static analysis for inferring model-validation rules (model validation)
  - Dynamic-taint analysis for localizing input-model faults (fault localization)
  - Dynamic analysis for repairing input-model faults (fault repair)

- Experimentation: additional types of transforms, more subjects
- Another technique for fault repair: predicate switching
- Interactive visual interfaces
- Chained transformations
- Support for identifying error markers
Questions