Incremental Test Case Generation for UML-RT Models

Eric James Rapos, Juergen Dingel
{eric,dingel}@cs.queensu.ca
Modeling & Analysis in Software Engineering Group
School of Computing
Queen's University, Kingston, Ontario, Canada

Motivation

Furthering of Research in Model Driven Development

- Improve usability of MDD techniques
- Develop tools for developers
- Work on cutting edge research

Improve Efficiency of Test Case Generation

- Automatic regeneration of test cases can be inefficient and sometimes redundant
- Make only the necessary changes to a test case
- Use an incremental process, to coincide with the MDD process

Understand Effects of Model Transformations

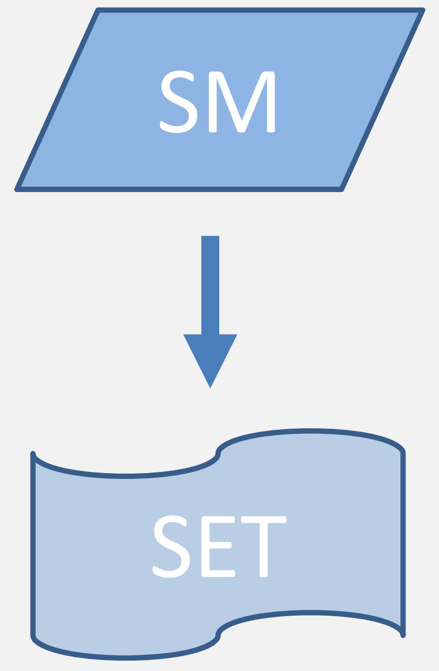
- Each type of change to model will have certain effects on the Symbolic Execution Tree and test cases
- We hope to categorize all typical model evolution steps in order to understand how they effect the artifacts of MDD

Resources

1. [Sel11] Bran V. Selic, "A Short Catalogue of Abstraction Patterns for Model-Based Software Engineering", to appear in Journal of Software and Informatics (2011)
2. [ZD11a] K Zurowska, J Dingel, "Symbolic Execution of UML-RT State Machines", DRAFT (2011)
3. [ZD11b] K Zurowska, J Dingel, "Modular Symbolic Execution of Communicating and Hierarchically Composed UML-RT State Machines", DRAFT (2011)
4. [UKB10] Engin Uzuncaova, Sarfraz Khurshid, Don S. Batory, "Incremental Test Generation for Software Product Lines", IEEE Trans. Software Eng. 36(3): 309-322 (2010)
5. IBM Rational Software Architect Real-Time Edition (RSA-RTE) - http://www-947.ibm.com/support/entry/portal/Overview/Software/Rational/Rational_Software_Architect_RealTime_Edition
6. Eclipse Modeling Framework (EMF) - http://www.eclipse.org/modeling/emf/

The Process

1. Symbolic Execution

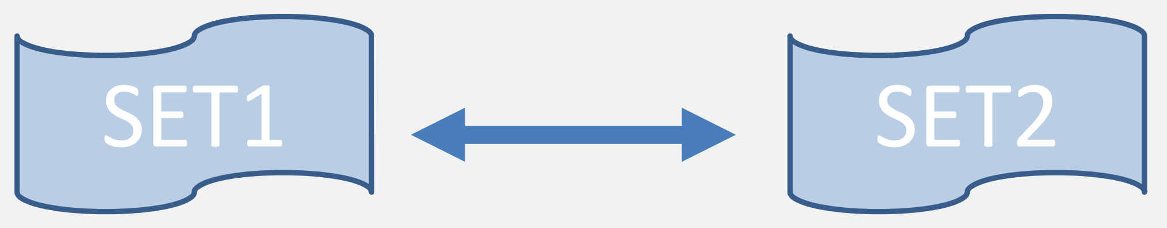


Input: UML-RT State Machine

Output: Symbolic Execution Tree in Model Form

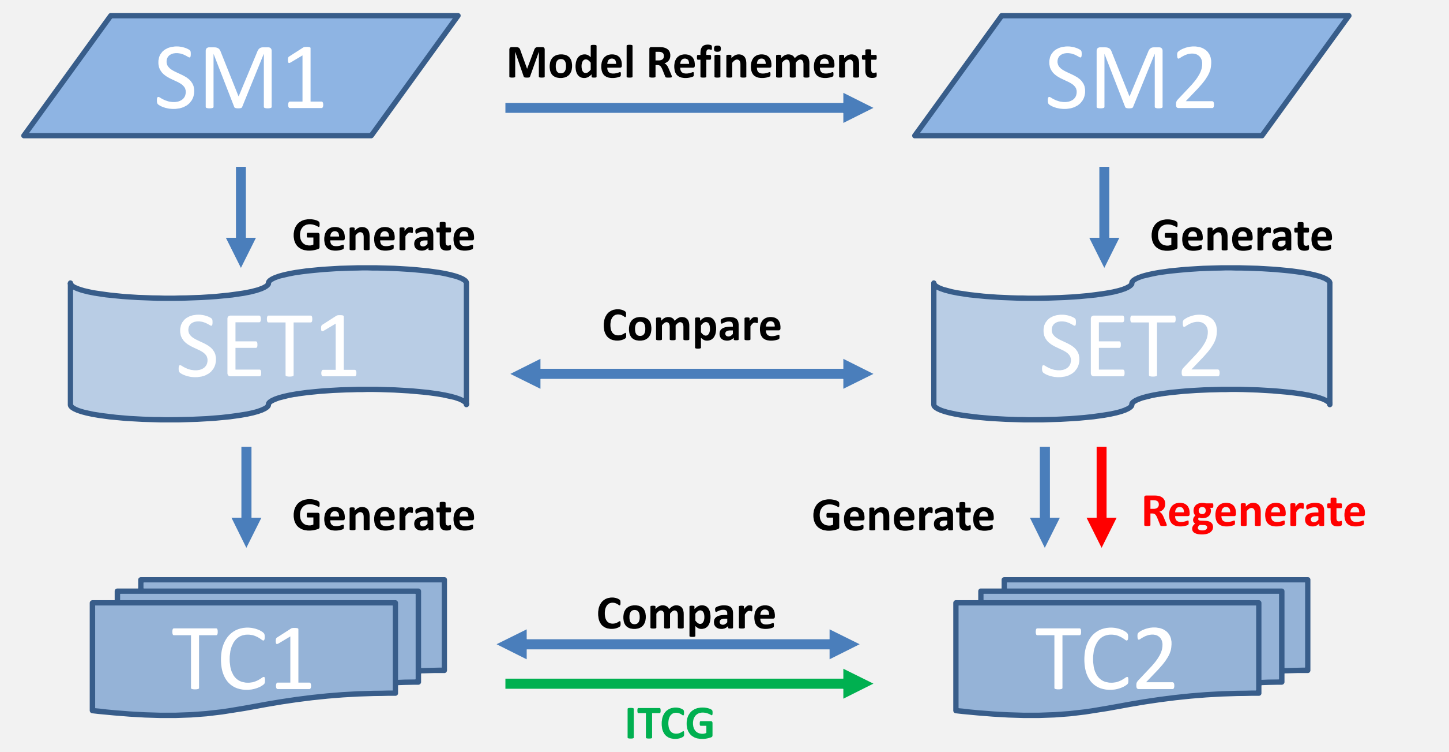
Details: This is existing work done by another member of the MASE Group [ZD11a][ZD11b]. The Model is symbolically Executed and output in a usable Ecore Model format for later processing.

2. Tree Differencing

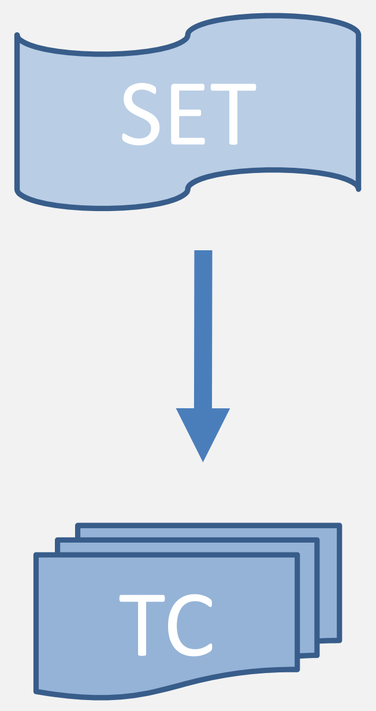


Input: Two Symbolic Execution Trees generated from Step 1
Output: A set of differences between the trees
Details: This is the current focus of my work. Being able to accurately determine how two SETs differ will help determine the effect of model changes on execution.

Overview



3. Test Case Generation

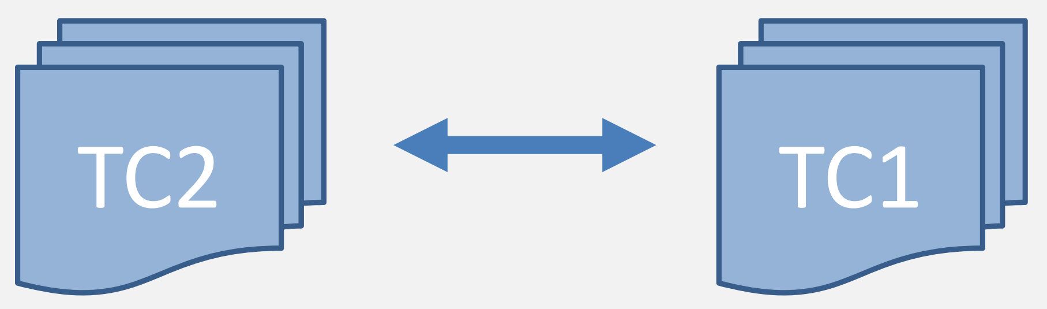


Input: Symbolic Execution Tree generated from Step 1

Output: A set of test cases for the State Machine that corresponds to this SET

Details: Using existing algorithms, Test Cases will be generated using the SET as input. This will be done in a manner that ensures completeness of the test cases.

4. Test Case Differencing



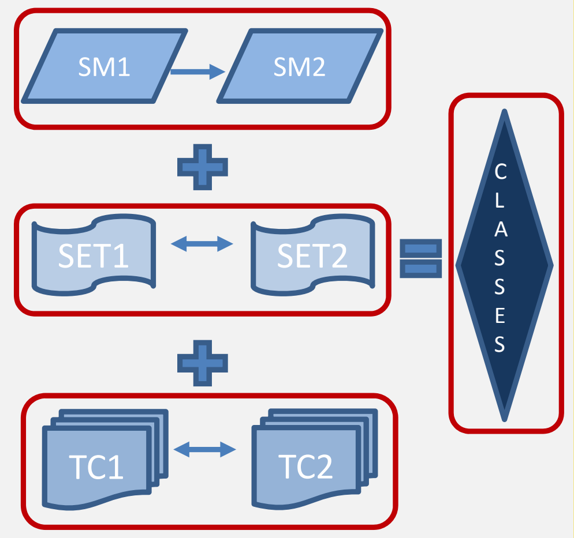
Input: Two sets of Test Cases generated from Step 3

Output: A set of differences between the Test Cases

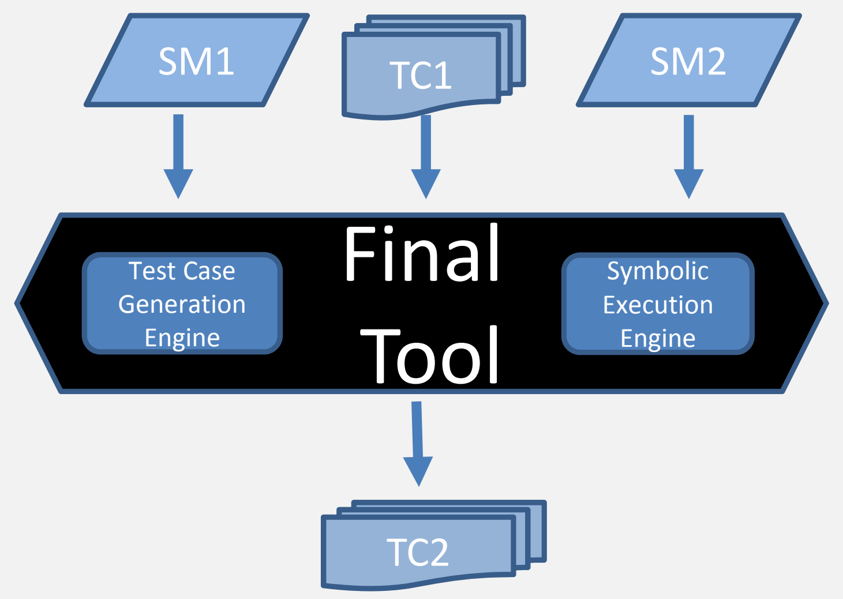
Details: By comparing the differences in generated test cases, the goal is to determine how a model change will effect a test case. This can be done by looking at which test cases have been removed, added, and/or changed. This step is purely part of discovery, and will not be used in the Incremental Test Case Generation process.

5. Classification of Model Evolution

Input: The sets of differences from Steps 2 and 4 & model evolution
Output: A defined set of classifications to be used in the tool
Details: Using the observations from Steps 2 and 4, create a set of classifications that will generalize for any model change, and how that change effects the generated test cases.



6. Tool Development



Input: Original State Machine, Generated Test Case, Modified State Machine

Output: Incrementally Generated Test Case for Modified State Machine

Details: Using the rules from Step 5, and other information from previous steps, the tool will intuitively modify the original test cases as needed.

Expected Outcomes

A Set of Classifications for Model Evolution

- For each standard model evolution step, determine its effect on both the Symbolic Execution Tree and the Test Cases
- Investigate non-standard evolution as well to determine possible effects
- Formulate a set of classifications based on findings

Better Understanding of State Machine Evolution

- The above classifications will not only be useful in our work, but as a better understanding of the MDD Process
- By better understanding the evolution process, it is our goal to improve the toolset used in MDD and Test Case Generation for UML-RT Models

A Software Implementation

- **Input to tool:** original model, test case for original model, and the evolved model
- **Functionality:** Use "The Process" to determine effects on test case
- **Output from tool:** modified test case for evolved model
- **Future:** Potential for integration with development environment