A Model Transformation Approach to Automated Model Evolution

Ph.D. Defense

Yuehua Lin
liny@cis.uab.edu
http://www.cis.uab.edu/liny
07/06/2007

Advisor: Dr. Jeff Gray
Thesis Committee:
Dr. Barrett Bryant
Dr. Aniruddha Gokhale
Dr. Marjan Mernik
Dr. Chengcui Zhang
Overview of Presentation

Domain-Specific Modeling
Help to change models rapidly

Background

Motivation

Research Goals

My Approaches

Evaluation

Frequent model evolution is needed
Help to change models correctly

Automated Model transformation to model evolution

Model transformation testing
Model-Driven Engineering (MDE)

- **MDE**: specifies and generates software systems based on high-level models

- **Domain-Specific Modeling (DSM)**: a paradigm of MDE that uses notations and rules from an application domain

- **Metamodel**: defines a **Domain-specific Modeling language (DSML)** by specifying the entities and their relationships in an application domain

- **Model**: an instance of the metamodel

- **Model Transformation**: a process that converts one or more models to various levels of software artifacts (e.g., other models, source code)
Metamodel, Model and System

- **Conformance**: a model $M$ is conformant to (c2) a metamodel $MM$ if there exists a function to associate the elements of $M$ to those of $MM$.

- **Substitutability**: a model $M$ is a representation of ($repOf$) the system $S$ if for each question of a given set of questions, the model $M$ will provide exactly the same answer that the system $S$ would have provided in answering the same question.

Categories of Model Transformation

- **Model transformation**
  - **Exogenous Transformation**
    - (e.g., models to XML files, models to source code)
  - **Endogenous transformation**
    - (e.g., model scaling and model refactoring)

**Exogenous transformation vs. endogenous transformation**: whether the source model and the target model are conformant to different modeling languages, which are represented by different metamodels.

**Model evolution**: adapts models to changing requirements or environments by changing their internal structure. Also endogenous transformation.
A Domain-Specific Modeling Tool Suite

The Generic Modeling Environment (GME) adopts the DSM approach and provides a plug-in mechanism for extension.
Example DSMLs and Applications

1. Modeling

2. Stepwise model transformations

An application for a family entertainment center

An application for automotive manufacturing system
Motivation

The ability to evolve models rapidly is needed for

- Exploration of various design alternatives in terms of system-wide issues
  - E.g., understanding tradeoff between battery consumption and memory size of an embedded device
  - E.g., scaling a model to 800 nodes to examine performance implications; reduce to 500 nodes with same analysis

- System adaptation to changing requirements and environments
  - E.g., inserting logging and concurrency concerns to data communication models
  - E.g., weaving deployment concerns to component models for rapid configuration
Is it easy to add a model manually?

1. Select the model element icon and drag it to the editor window

2. Build the connections

3. Set the values of the properties
Making changes into large-scale system models is harder…

Deep hierarchy!

Growing size!

Crosscutting concerns!

Model scaling concerns!
Challenges of Model Evolution

- The size of models will continue to grow
  - Domain-specific models often have repetitive and nested structures
  - Large-scale system models may contain large quantities of components (>1000s)

- Accidental complexities of the modeling activity
  - Overwhelming amount of mouse clicking and typing required within a modeling tool to describe a change
  - Manually constructing and evolving models is laborious, time consuming and prone to errors

- Many model evolution concerns crosscut within the deep model hierarchy
  - logging, constraints

A general metric for determining the effectiveness of a modeling toolsuite comprises the degree of effort required to make a correct change to a set of models.
Research Focus: Ability to Evolve Models Rapidly

Existing MDE Technologies

Domain Applications

App 1 App 2 App 3

Model Interpretation

Existing Domain Model

Evolved Domain Model

App 1 App 2 App 3

Evolved Domain Applications

Rapid system construction for exploring system issues such as scalability and adaptability

Research focus: ability to evolve models rapidly
Alleviate the accidental complexity of modeling large-scale, complex applications by providing a model transformation approach to automate model evolution

Goals:
- Improve the productivity
- Increase the accuracy

To achieve the above goals, we need
- Techniques for specifying and executing tasks of model evolution
- Techniques for determining the correctness of the changes that are made to the model
Overview of My Approach

- Apply a high-level model transformation language to specify tasks of model change evolution
- Construct a model transformation engine to automate such tasks by executing the transformation specification
- Provide a model transformation testing engine for improving the correctness of model transformation, which requires support for model comparison

![Diagram showing tool chain]

Model Transformation Engine  
Model Transformation Testing Engine  
Model Comparator  
The Generic Modeling Environment (GME)
Research results are integrated into existing MDE technologies to help software evolution.
Part I: Toward Automated Model Evolution

- Related work and the limitations
  - Many modeling tools provide low-level APIs (typically in C++ and Java) to manipulate and transform models so that model transformation developers have to handle accidental complexities with these languages.
  - The research on applying model transformation techniques to address model evolution is limited and most of them haven’t been used in evolving real models.
  - Current model transformation languages primarily aim to solve problems across multiple metamodels, which inevitably involve complicated syntax and semantics.
    - Graphical Languages (e.g., GReAT and Fujaba)
    - Hybrid Languages (e.g., the ATLAS Transformation Language)
My Approach

Automated model transformation:

- A high-level model transformation language – Embedded Constraint Language (ECL)
  - The language to specify model evolution concerns

- A model transformation engine (C-SAW)
  - The machine to execute the specification

- Incremental extension of Dr. Gray’s work
  - Extended the language features (e.g., the new type system) and adapted the function interfaces to work as a plug-in in GME;
  - C-SAW is implemented to allow transformation performed within GME.
The Transformation Language
--The Embedded Constraint Language (ECL)

- ECL is a textual language that supports a procedural style of model transformation

- An extension of the Object Constraint Language (OCL)

- Type system: atom, model, atomList, modelList, etc

- Model navigation and selection operations: models(), findModel(), atoms(), findAtom(), etc

- Model transformation operations: addModel, removeModel and setAttribute(), etc
Reducing the Complexities of Transforming Models

Using GME C++ modeling APIs

```cpp
CbuilderFolder *rootFolder, string: modelName
CBuilderModel *result = null;
const CBuilderModelList *subModels;
subModels = ((CBuilderFolder *)rootFolder)->GetRootModels();
POSITION POS = subModels->GetHeadPosition();
while(POS){
    CBuilderModel *subModel = subModels->GetNext(POS);
    if(subModel->GetName() == modelName){
        result = subModel;
        return result;
    }
}
return result;
```

Using ECL

```ecl
rootFolder().findModel("aModelName");
```
C-SAW: The Model Transformation Engine

- **MetaModel**
  - Defines ECL Interpreter Mode
  - Defines ECL Parser
  - Defines CopyAtom strategy CopyAtom

- **ECL Transformation Specifications**

- **Implemented as a GME plug-in**
Address Model Evolution Concerns

**Model Scalability:** construct a large model by replicating elements of a simple base model.

---

**Model Adaptability:** weave changes to models to satisfy changing requirements or environments.

*Weave deployment concerns into component models*
Application Example: Replicating a Base Model to Address Scalability Issues

Model Scalability

- Base models must be expanded to explore alternative designs
- Model elements need to be replicated, in addition to all required connections
Scaling System Integration Modeling Language (SIML)

- Assists in specification of configuration of large-scale fault tolerant data processing systems
- Used to model several thousand processing nodes for high-performance physics applications at Fermi Accelerator Lab
- Structural feature: deeply nested structures
  - A system model may be composed of independent regions
  - Each region may be composed of local process nodes
  - Each local process node may contain primitive application models

Address Model scalability with C-SAW
This example shows the capability of C-SAW to scale up a system configuration model by increasing its region models from 1 to 9 and scaling up each region’s internal elements, and then building the necessary communication connections.
aspect Start() {
    scaleUpNode("L2L3Node", 5);
    scaleUpRegion("Region", 8);
}

strategy scaleUpNode(node_name : string; max : integer) {
    rootFolder().findFolder("System").findModel("Region").addNode(node_name,max,1);
}

strategy addNode(node_name, max, idx : integer) {
    declare node, new_node, input_port, node_input_port : object;
    if (idx<=max) then
        node := rootFolder().findFolder("System").findModel(node_name);
        new_node := addInstance("Component", node_name, node);
        input_port := findAtom("fromITCH");
        node_input_port := new_node.findAtom("fromITCH");
        addConnection("Interaction", input_port, node_input_port);
        addNode(node_name, max, idx+1);
    endif;
}

strategy scaleUpRegion(reg_name : string; max : integer) {
    rootFolder().findFolder("System").findModel("System").addRegion(reg_name,max,1);
}

strategy addRegion(region_name, max, idx : integer) {
    declare region, new_region, out_port, region_in_port, router, new_router : object;
    if (idx<=max) then
        region := rootFolder().findFolder("System").findModel(region_name);
        new_region := addInstance("Component", region_name, region);
        //more : skip for conciseness
    endif;
}
Part II: Toward Improving the Correctness of Model Transformation

Problem:
- Model transformation specifications can be erroneous, resulting in unintended changes to the input models
- It is essential to ensure the correctness of the specifications before they are applied to a collection of models or reused across similar domains
- How to improve the correctness of model transformation?

Related work and the limitations
- Formal methods (e.g., model checking and theorem proving) are common techniques; however, such techniques are hard to use in general practice
- There is a lack of engineering approach such as executable testing that may be applied to current modeling practice

My Approach

Model Transformation Testing:

- Apply execution-based testing to model transformations
- A model transformation testing engine *M2MUnit* has been constructed to realize this vision
- Provide a test oracle as model comparison to determine the test result
A Model Transformation Testing Framework
Transformation Testing Engine: M2MUnit

A testing engine performs all the tests for testing a specification, which has three components: 1. Executor 2. Comparator 3. Test analyzer

Test oracle as model comparison: A test passes if there are no differences between the output and the expected models; otherwise, the test fails.
Part III: Model Comparison

Also called model differentiation, the capability to identify the mappings and differences between two models.

Answer these questions:

*Are they equivalent?*
*If no, what are the differences?*
Three Types of Model Differences

1) Missing connection (in red circle)
2) An extra atom (in blue rectangle)
3) Different attribute value (in green highlight)
Critical Issues on Model Comparison

- **Model comparison algorithms**: detecting the mappings and differences between two models by comparing all the elements and their properties within these models

- **Visualization of model differences**: use graphical symbols and colors to highlight all possible kinds of model differences (e.g., a missing element, or an element that has different values for some properties) in a structured way
Challenges and Related Work

**Challenges**
- Traditional differencing techniques applied to text files or structured data with tree structure
- Theoretically, generic model differencing is similar to the graph isomorphism problem, which is NP-hard

**Related work and limitations**
- Generic model differencing algorithms work on architectural models and data models
- In practice, modeling language-specific differencing algorithms may be more efficient
- Algorithms for differencing UML models are based on a single metamodel
- Few reports on differencing domain-specific models that may conform to different metamodels
My Approach
Formalizing a Model as a Graph

• A model can be represented formally as a hierarchical graph that consists of a set of nodes and edges, which are typed, named and attributed.

• The annotated type and name information of model elements are combined together as signatures, which are the non-structural syntactical information defined by the metamodel.

• The containment and connection relationships defined in the metamodel represent structural syntactical information.
My Approach (cont’d)

- Model Differencing Algorithms
  - The comparison starts from the two root models and then continues to the child sub-models
  - At each level, two metrics (i.e., signature matching and edge similarity) are combined to detect the mapped nodes and the different nodes between each pair of models
  - Based on the results of node comparison, all the edges are computed to discover all the edge mappings and differences.

- Features
  - Metamodel independent by using meta information
  - Achieve polynomial time in complexity
1. **Node signature matching:** $M_1$’s $E = M_2$’s $E$, $M_1$’s $F = M_2$’s $F$, $M_1$’s $B = M_2$’s $B$, $M_1$’s $C = M_2$’s $C$, $M_1$’s $D = M_2$’s $D$

2. **Maximal edge similarity:** $M_1$’s $A = M_2$’s $A'$

3. **Edge matching:** $E(a,b)$, $E(a,c)$, $E(a,d)$, $E(e,f)$ in $M_1$ and $M_2$ are matched to each other

4. **Model differences:** $A''$, $E(a'',c)$, $E(a'', b)$ of $M_2$
Visualization of Model Differences in a Tree View

(a) The host model: M1

(b) The candidate model: M2

(c) Model differences

DSMDiff implemented as a GME Plug-in

need to be deleted!

need to be added!

need to be changed!
When structural matching cannot find a unique candidate that has maximal edge similarity, the result is nondeterministic: in this case, $M_1$’s $A = M_2$’s $A'$ but $M_1$’s $B = M_2$’s $B'$ or $M_2$’s $B''$?

Possible improvements:
- Use new rule: the node with maximal already mapped neighbors is selected as the mapping
- Allow human interaction to pick the best candidate
Limitations and Improvements (cont’d)

Toward a smaller set of model differences:

- A new type of model difference needs to be introduced: Move, which may reference the subtree in M1, and its new container in M2.
Experimental Evaluation

- C-SAW has helped address model scalability and adaptability issues for different applications such as embedded avionics, computational physics and performance analysis. Example languages include:
  - Embedded System Modeling Language (ESML): modeling real-time mission computing embedded avionics applications (e.g., Boeing Bold Stroke)
  - The Stochastic Reward Net Modeling Language (SRNML): Specifying properties of high-performance physics experiments
  - Platform-Independent Component Modeling Language (PICML): specifying component-based systems.
  - Event QoS Aspect Language (EQAL): used to configure a large collection of federated event channels for mission-computing avionics applications
  - Others: Available from the Escher repository and Vanderbilt university, which makes modeling artifacts developed from NSF projects available for experimentation
A Summary of Applications of C-SAW

1. Aspect weaving new features in the Embedded System Modeling Language (ESML)

2. Model Scalability in 4 different DSMLs

3. Weaving deployment aspects in Platform-Independent Component Modeling Language (PICML)
Experimental Results

The experimental results have shown using C-SAW to perform model evolution is significantly faster and more accurate than manual processes.

- When scaling an EQAL model from 3 sites to 8 sites, a model engineer needs to insert more than 120 new elements and almost 150 connections, which requires more than 300 mouse clicks.
- When SIML models were scaled to more than 64 nodes, the manual process deteriorated taking several days with multiple errors.
- Using a model transformation, SIML models have been scaled up to 2500 nodes, flexibility for scaling up or down can be achieved through parameterization.
- The time to create the model transformation by a user unfamiliar with the domain: < 1.5 hours.
Contribution

- Automating model transformations for evolving models rapidly and correctly
- Applying software engineering processes such as testing to model transformations
- Developing algorithms for comparing models in the context of Domain-Specific Modeling
- The developed tools are available as open source and have been used by several external users in their research (several users are international, including a very active user in Colombia)
Publications

- Book chapters: 3, Journal papers: 5, Conference and workshop papers: 9

**Journal papers:**
- **Lin Y., Gray J. and Jouault F.,** “DSMDiff: A Differencing Tool for Domain-Specific Models,” conditionally accepted by *European Journal of Information Systems* (Special Issue on Model-Driven Systems Development).

**Book chapter:**
Future Research Directions

- Model transformation by example (MTBE)
  - Assists end-users in forming transformation rules through recorded interaction with the host modeling tool

- More work on model transformation testing
  - Automated test generation, test specification language to define test/test suites and their execution
  - Metamodel-based test coverage criteria

- Model transformation debugger for ECL
  - Setting breakpoints
  - Stepping through one statement at a time
  - Reviewing the values of the local variables and status of affected models
Acknowledgement:

This project was previously funded by the DARPA Program Composition for Embedded Systems (PCES) program, and currently supported by the National Science Foundation under CSR-SMA-0509342.

Video demonstrations, software downloads, and papers available at: http://www.cis.uab.edu/gray/Research/C-SAW/

Questions?
Back Up Slides
MDE, MDA, DSM and MIC

MDA is an OMG’s initiative for specifying and generating software based on models. The modeling language is UML, key concepts are PIM and PSM.

MDE is a big picture.

DSM is a *paradigm of MDE* that focuses on using concepts and rules that are familiar to end users for a specific domain to define a modeling language.

MIC is a specific DSM approach, developed by Vanderbilt university. It aims at providing solutions to develop systems that the physical hardware and software are very tied to each other. For example, automotive manufacturing systems and avionics systems.

DSM may solve problems that the hardware and software don’t have to tie to each other. For example, a banking system does nothing with hardware.
My Approach

Automated model transformation:
- A high-level model transformation language (ECL)
  - The language to specify model evolution concerns
- A model transformation engine (C-SAW)
  - The machine to execute the specification

Address model evolution issues:
- Model scalability: construct a large model by replicating elements of a simple based model.
- Model adaptability: weave changes to models that crosscut the model representation’s hierarchy
  - Weave deployment concerns into component models
  - Weave logging and concurrency concerns to component interaction models
strategy scaleUpNode(node_name : string; max : integer) {
    rootFolder().findFolder("System").findModel("Region").addNode(node_name, max, 1);
}

strategy addNode(node_name, max, idx : integer) {
    declare node, new_node, input_port, node_input_port : object;
    if (idx<=max) then
        node := rootFolder().findFolder("System").findModel(node_name);
        new_node := addInstance("Component", node_name, node);
        input_port := findAtom("fromITCH");
        node_input_port := new_node.findAtom("fromITCH");
        addConnection("Interaction", input_port, node_input_port);
        addNode(node_name, max, idx+1);
    endif;
}

strategy scaleUpRegion(reg_name : string; max : integer) {
    rootFolder().findFolder("System").findModel("System").addRegion(reg_name, max, 1);
}

strategy addRegion(region_name, max, idx : integer) {
    declare region, new_region, out_port, region_in_port, router, new_router : object;
    if (idx<=max) then
        region := rootFolder().findFolder("System").findModel(region_name);
        new_region := addInstance("Component", region_name, region);
        out_port := findModel("TheSource").findAtom("eventData");
        region_in_port := new_region.findAtom("fromITCH");
        addConnection("Interaction", out_port, region_in_port);
        router := findAtom("Router");
        new_router := copyAtom(router, "Router");
        addConnection("Router2Component", new_router, new_region);
        addRegion(region_name, max, idx+1);
    endif;
}
Node signature matching: 1’s E = 2’s E, 1’s F = 2’s F, 1’s C=2’s C, 1’s D=2’s D
Maximal edge similarity: 1’s A = 2’s A’ then 1’s B == B’ or B”? 

Given a node in M1, to find a matched node using maximal edge similarity, the result is deterministic only when all of their neighbors are already mapped.

When structural matching can not find a unique candidate, human interaction or 

Pick the remained node that has the maximal mapped neighbors
Related Work

• Graphical Transformation Language
  – Typified by graph grammars and graph rewriting techniques (e.g. GReAT and Fujaba)
  – provide a visual notation to specify graphical patterns of the source and target models (e.g., a subgraph of a graph).
  – It can be tedious to use purely graphical notations to describe complicated computation algorithms

• Hybrid Languages
  – Combine imperative and declarative constructs (e.g. the ATLAS Transformation Language and Yet Another Transformation Language)
  – Declarative constructs specify source and target patterns, and imperative constructs implement sequences of instructions.
  – Embedding predefined patterns render complicated syntax and semantics
Is it easy to create a model manually?

1. Select the model element icon and drag it to the editor window.

2. Build the connections.

3. Set the values of the properties.
Goal

- Develop automated techniques for model evolution that can
  - Improve the productivity
  - Increase the accuracy
- To achieve the above goal, we need
  - Techniques for specifying and executing tasks of model evolution
  - Techniques for determining the correctness of the changes that are made to the model
Publications

- Book chapters: 2
- Journal papers: 3 published, 2 conditionally accepted, 1 under review
- Conference and workshop papers: 9
Representative Publications

Journal papers:

• Lin Y., Gray J. and Jouault F., “DSMDiff: A Differencing Tool for Domain-Specific Models,” conditionally accepted by European Journal of Information Systems (Special Issue on Model-Driven Systems Development).


Book chapter:
Address Model Evolution Concerns

Model Scalability: construct a large model by replicating elements of a simple base model.

Model Adaptability: weave changes to models that crosscut the model representation’s hierarchy

Weave deployment concerns into component models

Crosscutting model concerns

Multiple Levels of Hierarchy

Modularizing crosscutting concerns

Component models

Deployment models
Experimental Results

C-SAW is a **modeling-language independent** model transformation engine that does not require extra customization efforts for different DSMLs.

The experimental results have shown using C-SAW to perform model evolution is significantly **faster** and **more accurate** than manual processes.

When SIML models were scaled to more than 64 nodes, the manual process deteriorated taking **several days** with multiple errors.

Using a model transformation, SIML models have been scaled up to 2500 nodes, flexibility for scaling up or down can be achieved through parameterization.

The time to create the model transformation by a user unfamiliar with the domain: **< 1.5 hours**