A Model Transformation Approach to Automated Model Construction and Evolution

Yuehua Lin *(liny@cis.uab.edu)*   **Jeff Gray *(gray@cis.uab.edu)***

Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham, AL 35294

http://www.cis.uab.edu/gray/Research/C-SAW/

1. Background

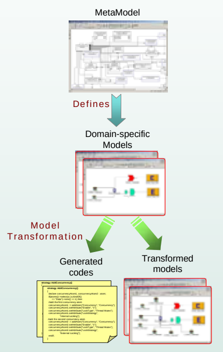
❖**Domain-Specific Modeling (DSM)**
address the complexity of software development by raising the specification of software to visual models at a higher-level of abstraction.

❖**MetaModel**
describes the entities and their relationships in an application domain.

❖**Models**
are instances of a metamodel to visually represent software at a higher-level of abstraction.

❖**Model Transformation**
is a process that creates or modifies the target software artifacts (e.g. models, codes) from the source models.

❖**Generic Modeling Environment (GME)**
is a modeling tool that allows one to define a domain-specific visual modeling language. http://www.isis.vanderbilt.edu/Projects/gme/.



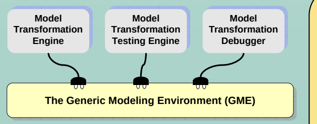
2. Problem Definition

The fundamental task of model construction and evolution for large scale software systems faces these challenges:
➢A large complex system can have thousands of coarse grained component models such that manually constructing or maintaining such models can be time consuming and prone to errors;
➢Many modeling toolsuites provide APIs at the source code level to manipulate models directly; but such an API approach requires model developers to learn and use low-level tools to program their tasks of transforming high-level models;
➢ It is difficult to capture and specify concerns during model construction and evolution which usually crosscut a model hierarchy;
➢Few modeling toolsuites provide testing and debugging facilities at an appropriate abstraction level to ensure the accuracy of model transformations for model evolution.

3. Overview of Research Goals

Objectives

➢ Provide a high-level model transformation language to perform specific tasks of model construction and evolution (e.g., model scalability) with aspect-oriented capabilities in specifying crosscutting modeling concerns.
➢ A core model transformation engine has been developed to automate such tasks, with support for testing and debugging of model transformation specifications.
➢ Address the fundamental issues of model comparison and visualization of model differences that are important to model transformation testing and debugging.


Model Transformation Engine
Model Transformation Testing Engine
Model Transformation Debugger
The Generic Modeling Environment (GME)

Overview of the C-SAW project

Three main components as GME plug-ins:

❖**Model Transformation Engine** is for executing model specifications;

❖**Testing engine** is for detecting errors in the transformation specification;

❖**Debugger** is for isolating errors in the transformation specifications.

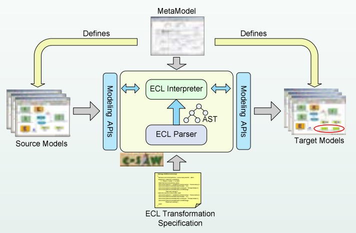
4. Model Transformation Engine: C-SAW

The Embedded Constraint Language (ECL)

➢ECL is a high-level transformation language supporting an imperative transformation style.

➢It provides numerous feature operations that support model navigation, selection and modification such as *select(), models(), atoms(), connections(), findAtom(), findModel(), addModel(), addAtom(), addConnection(), removeModel(), removeAtom(), setAttribute().*

➢It has aspect-oriented capabilities by specifying model transformation in two kinds of modular units:
▪**Aspect** specifies a crosscutting concern (e.g., a group of model elements crossing a model hierarchy).
▪**Strategy** specifies elements of computation (e.g., transformation behaviors) which will be bound to specific model elements defined by an aspect.



Benefits of ECL

❖ The accidental complexities of using the low-level details of the API are abstracted away in the ECL to provide a more intuitive representation for specifying model transformations.

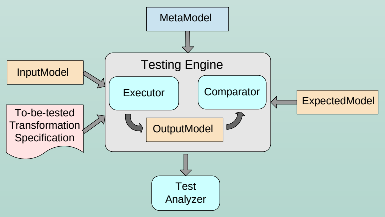
❖ It provides an aspect-oriented capability to specify crosscutting concerns that are typical in model adaptation and evolution

C-SAW: The Transformation Engine

➢It is implemented as a GME plug-in component, with a parser and an interpreter for ECL:
▪**The parser** generates an abstract syntax tree (AST) of the ECL specification.
▪**The interpreter** traverses this generated AST from top to bottom, and interprets it to perform a transformation by using modeling APIs provided by GME.

➢To perform a transformation, one input to C-SAW are source models, another input is the transformation tasks specified in ECL. These specifications are executed by C-SAW to weave changes into source models and generate the target models.

5. Model Transformation Testing and Debugging



A **testing engine** performs all the tests for testing a specification, which has three components:

➢**Executor** executes the to-be-tested specification on the input model to generate the output mode.

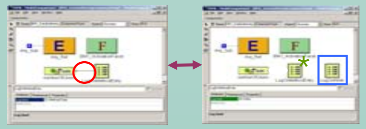
➢**Comparator** compares the output model (i.e., the actual result) to the expected model (i.e. the expected result) and collects the results of comparison. ***A test passes if there is no differences between the output and the expected models; otherwise, the test fails.***

➢**Test analyzer** visualizes the model differences to assist in comprehending the testing results.

A **debugger** isolates the cause of a transformation error with the ability to step through individual lines of the transformation specification.

Critical Issues
❖**Model Comparison Algorithms:** determine whether the two models are syntactically equivalent by comparing all the elements and their properties within these models.
❖**Visualization of model differences:** use graphical symbols and colors to indicate all possible kinds of model differences (e.g., a missing element, or an element that has different values for some properties).

The Example of Model Differences
➢**Missing connection** (in red circle)
➢**An extra atom** (in blue rectangle)
➢**Different attribute value** (in green highlight)

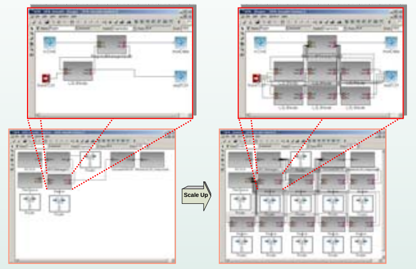


The Expected Model   **The Output Model**

6. Experimental Results and Case Study

Experimental results

❖helped to evolve a mission computing avionics application provided by Boeing through model transformations;
❖used in addressing the important issue of model scalability;
❖supported evolution of system models to propagate constraints into models.



Source Model   **Target Model**

Model Scalability

This example shows the capability of C-SAW to scale up a system configuration model by increasing its region and associated router from 1 to 9 and building the necessary connections.

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

strategy addRegion(region_name, max, idx : integer)     //recursively add regions
{
        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);

                //add connections to the new region;
                out_port := findModel("TheSource").findAtom("eventData");
                region_in_port := new_region.findAtom("fromITCH");
                addConnection("Interaction", out_port, region_in_port);

                //add a new router and connect it to the new region
                router := findAtom("Router");
                new_router := copyAtom(router, "Router");
                addConnection("Router2Component", new_router, new_region);
                addRegion(region_name, max, idx+1);
        endif;
}
```

ECL transformation specification to scale up a model

7. Conclusion and Future Work

The primary contribution of this research is **an investigation into automated model transformation that considers additional issues of testing and debugging to assist in determining the correctness of model construction and evolution.**

The future work to be performed for this research will focus on deeper investigation into the testing and debugging issues for C-SAW transformations. The focus will be on the fundamental issues for testing, such as model comparison and visualization, as applied to transformation specifications. Initial results have been published in [5].

References

1. Alanen, M. and Porres, I., "Difference and Union of Models," Proceedings of the UML Conference, Springer-Verlag LNCS 2863, San Francisco, California, October 2003, pp. 2-17.
2. Gray, J., Bapty, T., Neema, S., and Tuck, J., "Handling Crosscutting Constraints in Domain-Specific Modeling," Communications of the ACM, October 2001, pp. 87-93.
3. Gray, J., Lin, Y., Zhang, J., Nordstrom, S., Gokhale, A., and Neema, S., "Replicators: Transformations to Address Model Scalability," accepted - to appear in the proceedings of Model Driven Engineering Languages and Systems (MoDELS), Montego Bay, Jamaica, October 2005.
4. Gray, J., Zhang, J., Lin, Y., Wu, H., Roychoudhury, S., Sudarsan, R., Neema, S., Shi, F., and Bapty, T., "Model-Driven Program Transformation of a Large Avionics Application," Generative Programming and Component Engineering (GPCE 2004), Springer-Verlag LNCS 3286, Vancouver, BC, October 2004, pp.361-378.
5. Lin, Y., Zhang, J., and Gray, J., "A Framework for Testing Model Transformations," Model-driven Software Development - Research and Practice in Software Engineering, accepted for publication in 2005, Springer-Verlag (Book Chapter).