Metamodel-Driven Model Interpreter Evolution

Jing Zhang and Jeff Gray
Department of Computer and Information Sciences – The University of Alabama at Birmingham
http://www.cis.uab.edu/zhangj

This work is partially supported by NSF-CSR.

Statement of Purpose

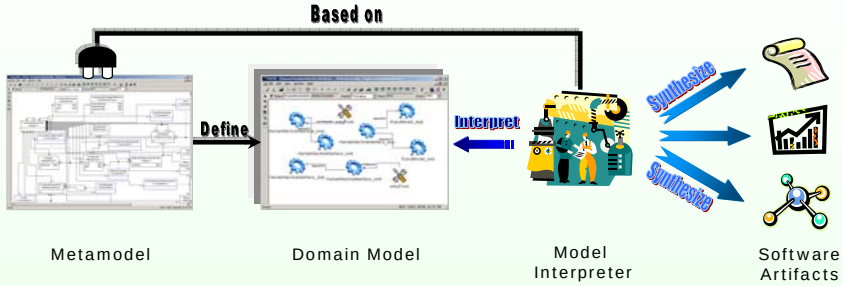
Domain-specific modeling (DSM) raises the level of abstraction by specifying a metamodel that is aligned to a particular problem domain and constructing model interpreters that synthesize the domain models into software artifacts. In the presence of new stakeholder requirements, it is possible that a metamodel undergoes numerous changes during periods of evolution. Consequently, there is a fundamental problem in keeping the model interpreters up to date with these changes. This poster presents two research objectives to facilitate model interpreter evolution in the presence of metamodel changes:

- Formalization of the interpreter implementation
- Evolution of the interpreter from model transformation specification

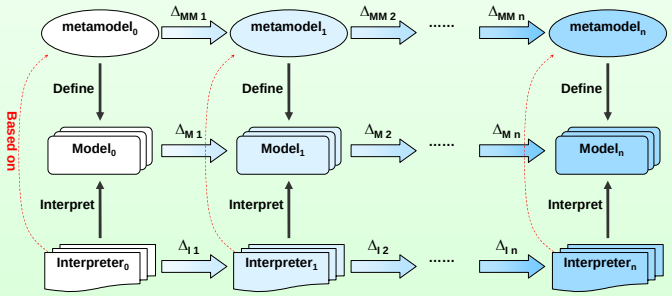
Background: Domain-Specific Modeling (DSM)

Key Characteristics of DSM

- Metamodeling is used to define a domain modeling language and the constraints within that domain.
- From the metamodel, a modeling environment is created for a specific domain.
- Domain experts work within the generated environment to create specific instances of domain models.
- The model interpreters traverse the internal representation of the model and generate new artifacts (e.g., XML configuration files, source code).



Evolution of Models and Interpreters in Terms of Metamodel Changes



Key Challenges

Challenge 1: Lack of formally-written model interpreter

- Different developers may program interpreters in various ways
- Hard to maintain and evolve such subjective realizations of model interpreters

Challenge 2: Lack of formal specification for metamodel transformation

- Metamodel transformation specifications must include the entire knowledge for the underlying interpreter evolution
- Δ_{MM} ? Δ_{I}

Technical Approach: Model Interpreter Formalization

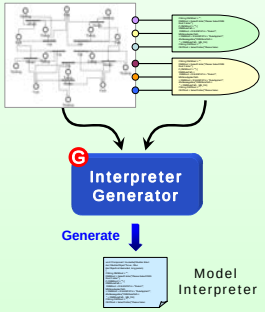
An interpreter implementation is composed of two parts:

Traversal Strategy:

- Formally specified on top of the metamodel
- Determines the control flow of the interpretation, i.e., which parts of the models are to be navigated, and in which order
- Provides binding and parameterization of the user actions to specific points

User Actions:

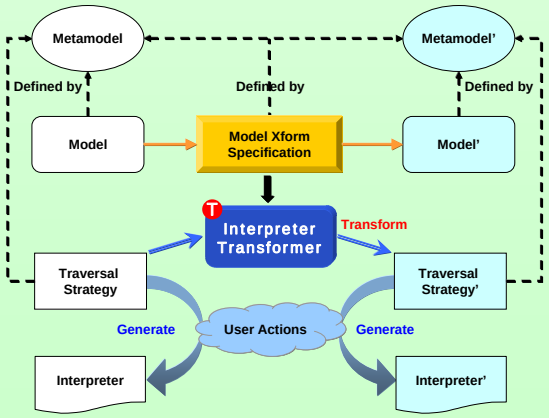
- A fragment of the user-defined code written in the Interpreter Language
- Denote what operations to take on visited model entities
- Capture the semantic intuition of an interpreter



Technical Approach: Interpreter Evolution from Model Transformation Specification

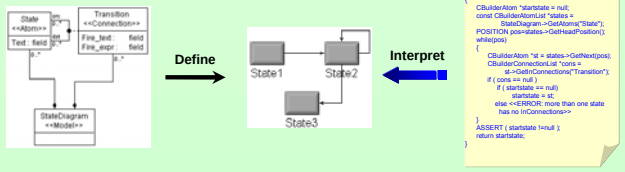
- Model transformation specification is used to define the transformation of the models that conform to two different metamodels. It is composed of pattern specification, replacement rule, and a set of constraints.
- The interpreter transformer analyzes the model transformation specifications, invokes the pattern matcher to locate the constituents to be transformed within the traversal strategy model, and eventually updates them according to the replacement rule specifications.
- User actions should remain intact because they embody the semantic intuition of a model interpreter.
- The transformed traversal strategy and user action code will be used to generate the new interpreter that can work under the new metamodel with the preserved semantics intuition.

Kernel Engines:
- Interpreter Generator (G)
- Interpreter Transformer (T)



Case Study: Evolution through Specialization of Domain Concepts

Old Domain



New Domain