Handling Crosscutting Constraints in Domain-Specific Modeling

Institute for Software Integrated Systems (ISIS) – Vanderbilt University

http://www.isis.vanderbilt.edu

This work is supported by the DARPA Information Technology Office (DARPA/ITO), under the Program Composition for Embedded Systems (PCES) program, Contract Number: F33615-00-C-1695.

PROJECT OBJECTIVE

This representation and analysis project is investigating the combination of Model Integrated Computing (MIC) and Aspect-Oriented Programming (AOP) composition technologies. In particular, the concepts of aspect-orientation will be applied at a higher level of abstraction - at the modeling level. An additional goal is to develop a framework for building domain-specific weavers. The specific goals of the project are to develop:

- A domain-specific, graphical language that captures the functional design of real-time embedded systems,
- A weaving process that maps high-level invariant properties and system requirements to design constraints affecting specific model regions, and
- A generation process that customizes components and composes real-time embedded systems.

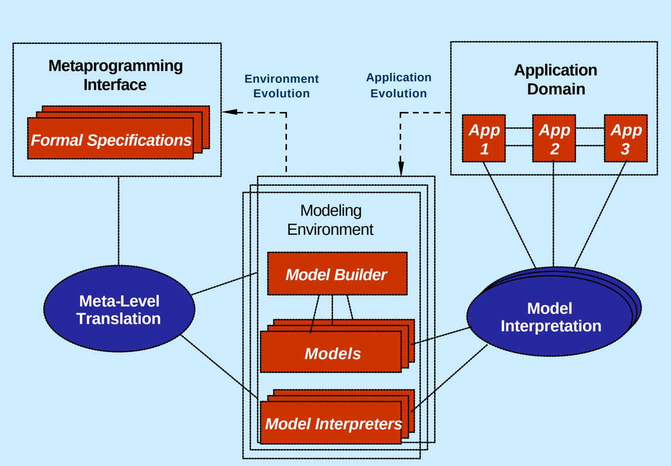
This project will help produce domain-specific and even application-specific modeling tools that will enable systems engineers to configure, analyze, and validate complex real-time embedded systems in a more intuitive manner.

BACKGROUND: DOMAIN-SPECIFIC MODELING

Key Characteristics of MIC

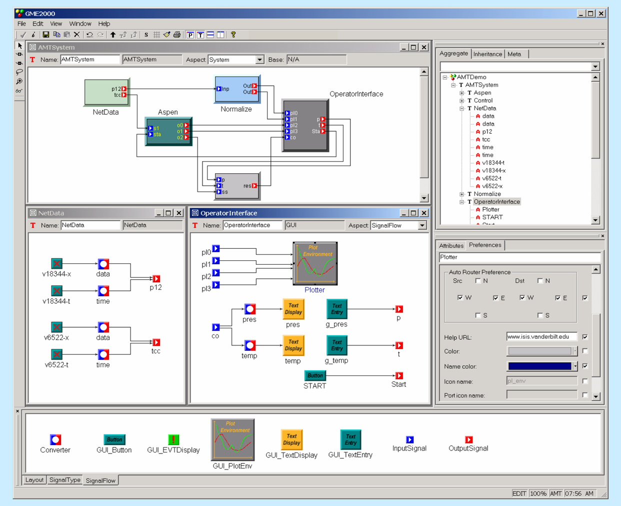
- Metamodeling is used to define a domain modeling language and the constraints within that domain.
- From the metamodel, a modeling environment is created for a specific domain.
- Domain experts work within the generated environment to create specific instances of domain models.
- Domain models can then be interpreted. This can result in an analysis of a model, or even synthesis into an application.

Model Integrated Computing (MIC)



Generic Modeling Environment (GME)

- Extendable/modular component-based architecture that supports MIC
- Consists of a metaprogrammable graphical editor, a model constraint checker, and metamodeling environment
- Key features:
  - Type inheritance
  - Model persistence (bi-directional XML, database)
  - Open API through COM
- Used in numerous industrial domains:
  - Automotive, avionics, electrical utilities, digital signal processing, chemical plants



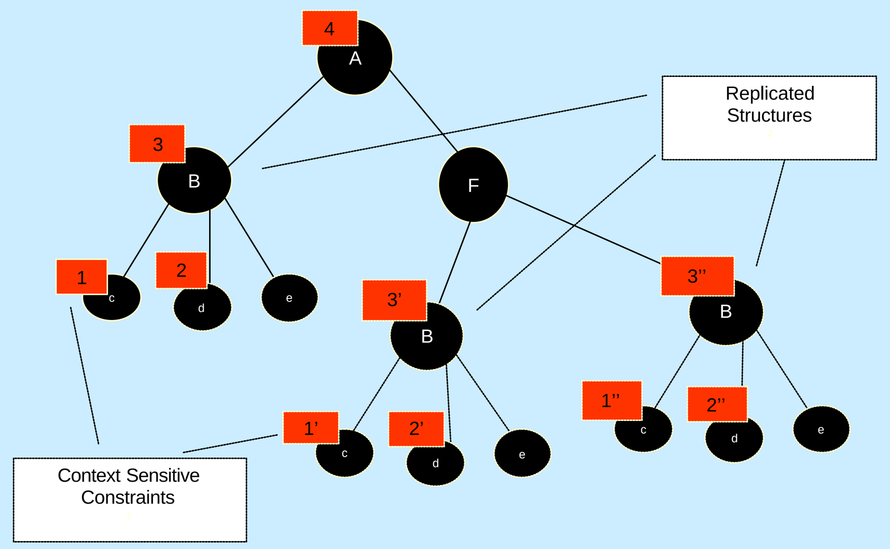
CHALLENGES: CROSSCUTTING CONSTRAINTS

Observation

- A real-time system's requirements, algorithms, resources, and behavior is captured as a "design-space," representing a multitude of alternative system implementations.
- Navigation of the design space is assisted by the specification of design constraints as components within a set of hierarchical, multiple-view models.
- These constraints are scattered across the hierarchy of a model.
- Therefore, constraints represent a type of crosscutting concern within domain-specific modeling.

Consequences

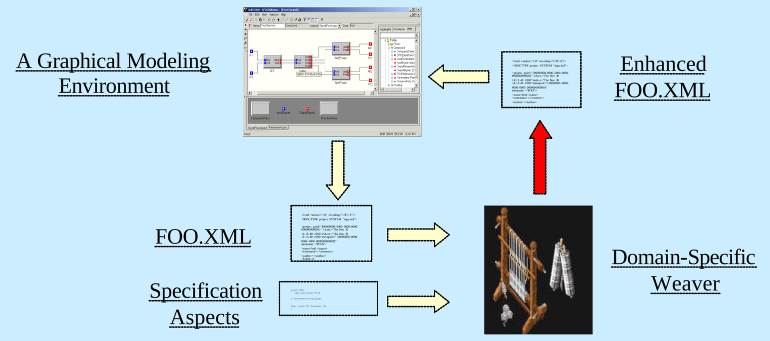
- The crosscutting nature makes it difficult to maintain and reason about the effect and purpose of each constraint.
- Managing these distributed constraints becomes extremely difficult as system size increases.
- Experimenting with different architectures, synchronization methods, network protocols, etc., becomes error-prone and labor intensive.



TECHNICAL APPROACH: DOMAIN-SPECIFIC WEAVER

A Constraint Weaver

- A solution that isolates the constraints as a separate area of concern will improve the manageability of our models. This can be accomplished with a constraint weaver.
- The input to the domain-specific constraint weaver consists of:
  - The XML representation of the model, as exported from the GME. This model is most likely void of any constraints.
  - A set of specification aspects provided by the modeler. These are used to specify the locations in the model where constraints are to be added by strategies.
- The output of the weaving process is a new description of the model in XML that contains new constraints that have been integrated throughout the model by the weaver.

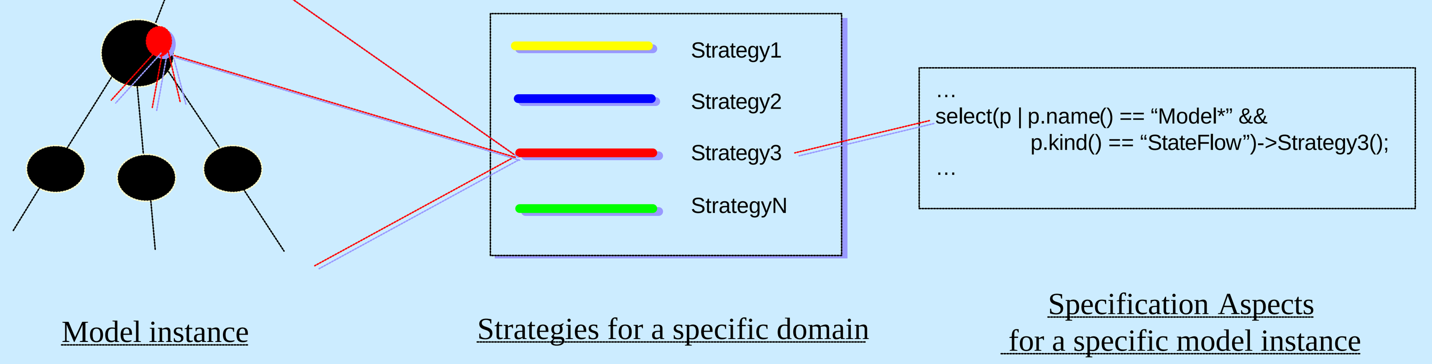


How Constraints are Weaved

- Specification aspects describe specific points in the model where a strategy is to be applied. This ability to quantify over the model space is similar to a pointcut descriptor in AspectJ.
- Strategies are domain-specific rules for specifying elements of computation, propagation, or application of specific properties to the model nodes.
- Strategies are generic in the sense that they are not bound to specific nodes in the model.

Benefits

- Because constraints are modularized apart from the models, different sets of specification aspects can be weaved into the same model for experimental "what if" scenarios.
- Because much of the redundancy of constraint application is removed, the effect of each constraint on the global system can be better understood. This localization of constraints improves modular reasoning.



EMBEDDED CONSTRAINT LANGUAGE (ECL)

Properties of ECL

- ECL is an extension of the Object Constraint Language (OCL)
- Arithmetic operators
  - +, -, *, /, =, <, >, <=, >=, <>
- Logical operators
  - and, or, xor, not, implies, if/then/else
- Collection operator (->), Property operator (.)
- Operations on collections:
  - collection->size() : integer
  - collection ->forAll(x | f(x)) : Boolean
  - collection ->exists(x | f(x)) : Boolean
  - collection ->select(x | f(x)) : collection

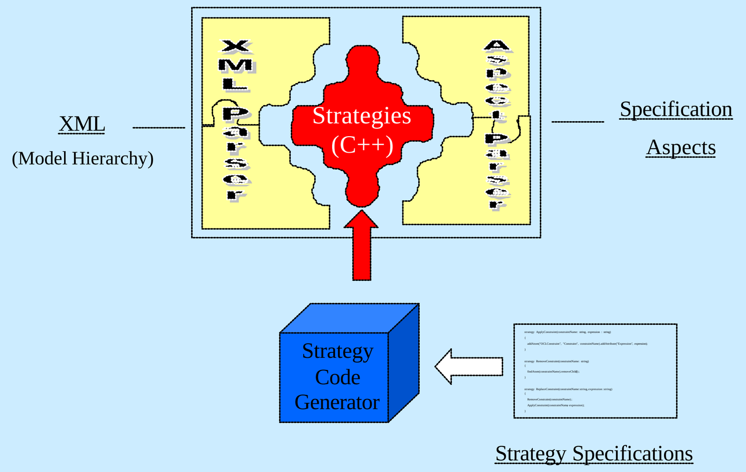
Operations on Model Objects

- Traditional OCL has been strictly a declarative query language
- New uses require an imperative procedural style
- Addition of side effects into model
  - Examples:
    - addAtom(…), findAtom(…)
    - addAttribute(…), findAttribute(…)
    - removeNode(…)
- Support for recursion
- Chaining of strategies (procedure calls)
- Inlined C++ code

META-WEAVER FRAMEWORK

Creating New Weavers

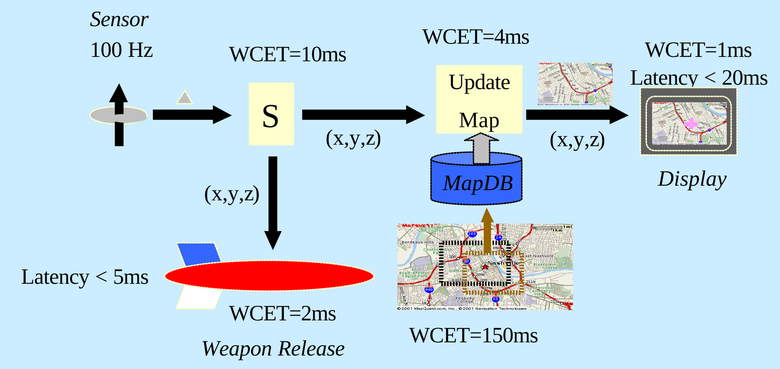
- Each specific GME metamodeling paradigm introduces different types of modeling elements, syntax, and semantics. Therefore, different weavers are needed for different paradigms.
- Strategies are used to aid in the rapid construction of new domain-specific weavers. ECL constraints can succinctly capture the specification of these strategies.
- A code generator translates the strategies into C++ code that is then compiled within the weaver framework. Each domain can then be considered as being componentized within the weaver.

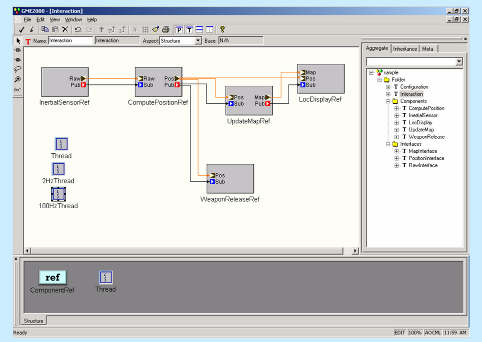


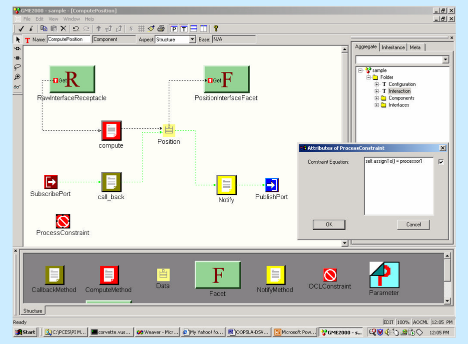
EXAMPLE: PROCESSOR ASSIGNMENT

Description

Given 5 components for a weapons deployment system, this example demonstrates the weaving of constraints that represent the processor assignment for each component. The strategy is based upon the worst-case execution time (WCET) for each component.







```
strategy ApplyConstraint(constraintName : string, expression : string)
{
  addAtom("OCLConstraint", "Constraint", constraintName).addAttribute("Expression", expression);
}

strategy Assign(limit : int)
{
  <<static int accumulateWCET = 0; static int processNum = 1; int currentWCET; >>
  findAtom("compute").findAttributeNode("WCET").getInt(currentWCET);
  <<accumulateWCET = accumulateWCET + currentWCET; >>
  if (limit < accumulateWCET) then
    <<accumulateWCET = currentWCET; processNum++; >>
    ReportNewProcessor();
  endif;
  <<CComBSTR aConstraint = "self.assignTo() = processor" + XMLParser::itos(processNum); >>
  ApplyConstraint("ProcessConstraint", aConstraint);
}

strategy ProcessorAssignment(limit : int)
{
  models("Component")->forAll(Assign(limit));
}
```