Program Families in Scientific Computing

Spencer Smith, John McCutchan and Fang Cao

Department of Computing and Software
McMaster University

The 7th OOPSLA Workshop on Domain-Specific Modeling
Outline

2. Proposed Methodology
3. Multipurpose Tool: A Mesh Generator Generator
Advantages of Program Families to SC?

- **Usual benefits**
  - Reduced development time
  - Improved quality
  - Reduced maintenance effort
  - Increased ability to cope with complexity

- **Reusability**
  - Underused potential for reuse in SC
  - Reuse commonalities
  - Systematically handle variabilities

- **Usability**
  - Documentation often lacking in SC
  - Documentation part of program family methodology
  - Create family members that are only as general purpose as necessary

- **Improved performance**
Is SC Suited to a Program Family Approach?

- The redevelopment hypothesis
  - A significant portion of requirements, design and code should be common between family members
  - Common model of software development in SC is to rework an existing program
  - Progress is made by removing assumptions

- The oracle hypothesis
  - Likely changes should be predictable
  - Literature on SC, example systems, mathematics

- The organizational hypothesis
  - Design so that predicted changes can be made independently
  - Tight coupling between data structures and algorithms
  - Need a suitable abstraction
Proposed Methodology

1. Identify family of interest
   - Specific physical model?
   - Multipurpose tool?

2. Commonality analysis
   - Terminology
   - Commonalities
   - Variabilities
   - Parameters of variation
   - Binding time

3. Domain Specific Language (DSL)

4. Generation of family members
Commonality Analysis

1. Reference Material: a) Table of Contents b) Table of Symbols c) Abbreviations and Acronyms
4. Commonalities: a) Background Overview b) Terminology Definition c) Goal Statements d) Theoretical Models
5. Variabilities: a) Input Assumptions b) Calculation c) Output
6. Traceability Matrix
7. References
A Mesh Generator Generator
Mesh Generator (MG) Goals

G1 Input spatial domain $\Omega$ output a mesh $M$ that covers this domain.

G2 Transform information on the materials, material properties and the locations of the different materials

G3 Transform information on the boundary condition types, values and locations

G4 Transform system information, such as numerical algorithm parameters
Element Variability

Location of nodes: sequence of LocationT
Number of dof at nodes: sequence of $\mathbb{N}$
LocationT = tuple of $(L_1 : \text{natT}, L_2 : \text{natT}, L_3 : \text{natT})$
natT = $\{ s : \mathbb{R} | 0 \leq s \leq 1 : s \}$
Local Topology Variability

Quad

Triangle1

Triangle2

Triangle3

Triangle4

Triangle5

Triangle6

Triangle7

Triangle8
DSL Using XML

```xml
<elementSet>
  <geometrySpec>
    <shape>triangle1</shape>
    <nodeGeo count="3">
      <node id="1">
        <location>1,0,0</location>
      </node>
      <node id="2">
        <location>0,1,0</location>
      </node>
      ...
    </nodeGeo>
  </geometrySpec>
</elementSet>
```
Proof of Concept Implementation

- XML document that customizes a Java object
- The Java object customizes the general purpose MG as it is loaded
- General purpose MG
  - All variabilities bound at run-time
  - Corresponds to an empty XML specification
A Virtual Material Testing Laboratory
Given the deformation history of a material particle, determine the internal stress within the material particle.
Theoretical Model Input

\[ \sigma_0 : \mathbb{R}^6 \]

\[ t_{\text{beg}} : \mathbb{R} \]

\[ t_{\text{end}} : \mathbb{R} \]

\[ \dot{\epsilon}(t) : \{ t : \mathbb{R} \mid t_{\text{beg}} \leq t \leq t_{\text{end}} : t \} \rightarrow \mathbb{R}^6 \]

\[ \text{mat}\_\text{prop}\_\text{val} : \text{string} \rightarrow \mathbb{R} \]

\[ E : \{ x : \mathbb{R} \mid x \geq 0 : x \} \]

\[ \nu : \{ x : \mathbb{R} \mid 0 < x \leq 0.5 : x \} \]
Theoretical Model Output

\[ \sigma(t) : \{ t : \mathbb{R} | t_{beg} \leq t \leq t_{end} : t \} \rightarrow \mathbb{R}^6 \] such that

\[ \dot{\sigma} = D \left( \dot{\epsilon} - \gamma < \phi(F(\sigma, \kappa)) > \frac{\partial Q(\sigma)}{\partial \sigma} \right) \text{ and } \sigma(t_{beg}) = \sigma_0 \]
Variabilities

- $F = F(\sigma, \kappa) : \mathbb{R}^6 \times \mathbb{R} \to \mathbb{R}$
- $Q = Q(\sigma) : \mathbb{R}^6 \to \mathbb{R}$
- $\kappa = \kappa(\epsilon^{vp}) : \mathbb{R}^6 \to \mathbb{R}$
- $\phi = \phi(F) : \mathbb{R} \to \mathbb{R}$
- $\gamma : \mathbb{R}$
- $mat\_prop\_names : set\ of\ string$
Code Generation

- Specify variabilities
- Symbolically calculate terms needed by numerical algorithm, including $\frac{\partial Q}{\partial \sigma}$, $\frac{\partial F}{\partial \sigma}$, etc.
- Symbolic processing avoids tedious and error-prone hand calculations
  - Reduces workload
  - Allows non-experts to deal with new problems
  - Increases reliability
- Use Maple Computer Algebra System for model manipulation
- Convert math expressions into C expressions using “CodeGeneration”
- Inline into a C++ class defining the material model
- A finite element program can this interface to realize the numerical algorithm
BNF of DSL for $F$

\[
\langle \text{expression} \rangle \rightarrow \langle \text{number} \rangle | \\
(\langle \text{expression} \rangle) | \\
\langle \text{expression} \rangle \wedge \langle \text{expression} \rangle | \\
\langle \text{expression} \rangle \ast \langle \text{expression} \rangle |
\]

... 

\[
\langle \text{simulation-variable-F} \rangle | \langle \text{user-defined-constants} \rangle \\
\langle \text{simulation-variable-F} \rangle \rightarrow \text{Kappa} | \langle \text{simulation-variable-stress} \rangle | \langle \text{simulation-variable-stress-macros} \rangle \\
\langle \text{simulation-variable-stress} \rangle \rightarrow \text{SigmaXX}|\text{SigmaYY}|\text{SigmaZZ}|\text{SigmaXY}|\text{SigmaYZ}|\text{SigmaXZ} \\
\langle \text{simulation-variable-stress-macros} \rangle \rightarrow \text{Sxx}|\text{Syy}|\text{Szz}|\text{Sxy}|\text{Syz}|\text{Sxz}|\text{Sm}|\text{J2}|\text{J3}|q \\
\langle \text{user-defined-constants} \rangle \rightarrow \langle \text{string} \rangle
\]
Concluding Remarks

- SC software is a great candidate for development as a program family
- Produce programs that are as special or general purpose as needed
- Improve reusability, usability and reliability
- Potential to improve performance
- A commonality analysis facilitates the design of a DSL
- Symbolic processing and code generation are very useful techniques