Program Families in Scientific Computing

Spencer Smith, John McCutchan and Fang Cao

Computing and Software Department, McMaster University, smiths@mcmaster.ca

Abstract. This paper motivates how the quality of scientific computing applications can be improved by developing them as a program family. In particular, arguments are presented that a program family strategy improves the qualities of reusability and usability. The proposed methodology consists of determining the scientific computing family of interest, then performing a Commonality Analysis (CA) to document the terminology, commonalities (including goals and theoretical models) and variabilities (including assumptions, input variabilities and output variabilities). In the next step the CA is used as a basis for a Domain Specific Language (DSL) from whose instances program family members are automatically generated. The proposed approach is illustrated through two examples: a family of mesh generators and a family of virtual material testing laboratories.

1 Introduction

Over the past 50 years there has been considerable growth in the importance of scientific computing (SC), where SC is defined in this paper as the use of computer tools to analyze or simulate continuous mathematical models of real world systems of engineering or scientific importance so that we can better understand and potentially predict the system's behaviour. A small sample of some important applications of SC include the following: designing new automotive parts, analyzing the flow of blood in the body, and determining the concentration of a pollutant released into the water table. As these examples illustrate, SC can be used for tackling problems that impact such areas as manufacturing, financial planning, environmental policy, and the health, welfare and safety of communities. Given the important applications of SC, it is surprising that many aspects of the quality of this software, such as usability, reusability, verifiability, maintainability and portability are frequently neglected in practise. To make matters worse, the quality of scientific software is becoming increasingly more of an issue because the complexity and size of the problems that can be simulated on modern computers is constantly growing. Moreover, increasing complexity and size means an increase in the time and effort necessary to develop new SC software. The question for the future is how to meet the growing need to rapidly develop high quality software to solve large and complex numerical computation problems? The answer proposed in this paper is to improve productivity and software quality by adapting the software product line, or program family, strategy, which has been so successfully applied in other application domains [1, 2].

Many of the ideas associated with program families are not new to researchers in SC. For instance, Carette [3] shows how to create a family of efficient, type-safe Gaussian elimination algorithms by fixing different design decisions and then using code generation. Code generation is also used in ATLAS [4] and Blitz++ [5] to produce efficient and portable SC code for linear algebra and array processing. Other ideas related to program families, in particular software reuse and capturing domain knowledge, figure prominently in SC research on Problem Solving Environments (PSEs). A PSE is "a computer system that provides all the computational facilities necessary to solve a target class of problems efficiently" [6]. Two example PSEs are Matlab and Maple. Although the contributions listed above can improve the quality and productivity of scientific software, none of these examples follows a full program family development strategy. For instance, the requirements stage is underemphasized in all of the above cases. In addition, the domain covered by many PSEs is potentially too large to realize all of the benefits of program family development.

The current study looks at program families in SC starting from the requirements analysis, or more accurately the commonality analysis stage. The goal is to systematically identify and document the commonalities and variabilities between family members. A Domain Specific Language (DSL) is then designed for each family to facilitate rapid development of the family members. To keep the domains small enough to realize the full benefits of program family development this paper focuses on two sets of domains. The first set of domains is small multi-purpose SC tools. The members of this set of domains roughly correspond to the chapters in a typical introductory SC text. For example, typical domains include ordinary differential equation solvers, linear solvers, numerical integrators, mesh generators etc. The second set of domains of interest is solvers for specific physical problems. This set of domains involves solving a particular governing set of equations to solve for such dependent variables as displacement, velocity, pressure, temperature, concentration, etc.

The next section enumerates the reasons why it is appropriate and beneficial to develop SC applications as program families. After this motivation of SC families, the specific methodology promoted in this paper is presented. The next sections give two example SC program families: a family of mesh generators and a family of virtual material testing laboratories.

2 Why Program Families in Scientific Computing?

The program family approach can be recommended for SC because of the benefits it provides. Besides the benefits that apply to all program families, such as reduced development time, improved quality, reduced maintenance effort and the ability to cope with complexity, there are two qualities that show particular promise for improvement: reusability and usability. The need for reusability is seen by considering the long lists of similar SC programs available on the Internet. For instance, in the case of mesh generators a large number of similar programs have been written. A survey by Owen [7] shows 94 different mesh generator packages, with 61 of them generating triangular meshes, and 43 of these using the same algorithm of Delaunay triangulation. Program family development has the potential to significantly improve SC code with respect to reusability because the common code only has to be implemented once and the variabilties are handled systematically. Part of the explanation for limited reuse in SC is the frequent failure to achieve the quality of usability. Usability suffers because in many cases the only documentation for a program is the source code itself. A program family approach helps here because documentation is part of the methodology. Another reason that usability suffers in SC is the current trend to develop general purpose tools and physics solvers that are as widely applicable as possible. For instance, modern stress analysis programs allow for intricate three dimensional boundaries, complex constitutive equations and transient analysis. The level of sophistication is overwhelming to the engineer who may only want to find the stress in a rectangular plate made of a linear elastic solid. A program family approach allows generation of family members that have exactly the degree of complexity required for a particular domain. An additional benefit of customized applications is that the generation of the family member can potentially take advantage of the simplifications to produce more efficient code.

Given that the program family strategy can benefit SC software, the next question is whether SC software is a suitable candidate for development as a family? The mathematical nature of SC is one argument in favour of development as a program family, since mathematics reduces ambiguity and provides underpinnings that provide clear models for domain engineering. The suitability of the concept of a program family is further shown by the fact that appropriately selected domains within SC meet the three hypotheses for a program family as proposed by Weiss [8]:

The Redevelopement Hypothesis: This hypothesis requires that most software development involved in producing the family should be redevelopment, which means that there should be a significant portion of common requirements, design, and code between the family members. For the SC example of mesh generators there are many cases of large general purpose codes that have been constantly redeveloped over their lifetimes, such as QMESH [9] and CUBIT [10, 11] from Sandia National Laboratories. Moreover, a common model of software development in SC is to build a new program by modifying an existing program. In the case of special purpose software to solve physical problems modelling assumptions become the major distinguishing feature. For instance, Smith and Stolle [12] summarize papers on polymer film casting based on their simplifying assumptions, such as whether the problem is modelled as 1D, 2D or 3D, or whether the problem is isothermal or nonisothermal or whether the polymer is assumed viscous or viscoelastic.

The Oracle Hypothesis: This hypothesis requires that the types of changes that are likely to occur during the system's lifetime are predictable. This is certainly the case in SC where one can determine likely changes by consulting the large body of literature and mathematical theories on the topic. The literature on physical modelling is relatively stable, as the laws of physics are well understood and slow to change. In addition, there are many example SC applications that show how the software can evolve over time.

The Organizational Hypothesis: According to the organizational hypothesis, the program to be developed using the program family approach should be one that allows designers and developers to organize the software, as well as the development effort, in a way that the predicted changes can be made independently. If this assumption holds, then a predicted change will require changing only a few modules in the system. For some of the likely changes, such as changes in the user interface, visualization options, and output formats, the changes can be dealt with in an elegant way. However, in other cases it can be difficult to design for independent changes, such as when there is a strong coupling between an SC algorithm and the associated data structure. An example of this coupling is the dependence of the quad tree meshing algorithm on a quad tree data structure. Fortunately, suitable abstractions can assist with this problem. In the case of mesh generators there are examples of system's organized so that predicted algorithm and data structures changes can be made independently [13–15].

3 Proposed Methodology

The first step in the proposed methodology is to determine the SC program family of interest. The second step consists of a Commonality Analysis (CA) [16, 17] on this identified family. The CA can be seen as a method for summarizing the requirements for all potential tools that are considered to be within the scope of the project. The CA includes documentation of terminology, commonalities (including terminology, goals and theoretical models) and variabilities (including assumptions, input variabilities and output variabilities). The CA is then used as a basis for a DSL in the third step. A specification for a family member is written using the DSL and then the family member is generated from this specification.

A template for use when documenting a CA for SC software has been previously introduced [18]. This template is reproduced in Figure 1. The template includes a section listing potential system contexts, user characteristics and system constraints. The CA covers all the potential members of a program family, so it is not possible to know exactly what information to place in these sections; however, there will often be information that can be recorded on typical uses of the program family members. Although the information in this section cannot be presented as variabilities, because it does not represent requirements, the hints provided in the CA can later be refined during the process of application engineering.

1. Reference Material: a) Table of Contents b) Table of Symbols c) Abbreviations and Acronyms
2. Introduction: a) Purpose of the Document b) Organization of the Document
3. General System Description: a) Potential System Contexts b) Potential User Characteristics c) Potential System Constraints
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

Fig. 1. Proposed Commonality Analysis Template

A "Terminology" section is commonplace in requirements documentation. In the case of the CA its inclusion is motivated by a need to clarify the domain concepts and to serve as a reference aid. The contents of this section consist of a list of mathematical concepts and their exact meaning. This section should provide enough information to allow understanding of the later sections "Goal Statement," "Theoretical Model," and "Variabilities." The terminology section is necessary in scientific computing for an unambiguous requirements specification because terminology often has subtly different meanings, even in very similar contexts.

The motivation of the goal statement section of the CA is to capture the goals in the requirements process. A goal, in this context, is a functional objective the system under consideration should achieve. The goal statements do not include nonfunctional objectives because nonfunctional requirements are not commonalities between family members. Goals provide criteria for sufficient completeness of a requirements specification and for requirements pertinence. Goals will be refined in the Section "Theoretical Models." The goal statements are intended to be written at a level that is easy to understand. The goal statements should briefly summarize the commonalities shared by all program family members.

The "Theoretical Models" section specifies the theory that all members of the SC family share. The model is presented as it would be presented in a mathematics textbook. That is, the model is specified as the ideal mathematical case, without reference to the limitations that an actual computer implementation will have to overcome. This is done so that there is a relatively uncomplicated reference model that all stakeholders can agree on and understand.

The section on input assumptions, which is mainly intended for program families of multi-purpose tools, emphasizes the importance of assumptions to SC for making the theoretical model something that can be solved. In many cases, if no constraints are placed on the theoretical model, then it cannot be solved numerically for all possible inputs. In the case of families of specific physical problem solvers, the assumptions section may be moved to be part of the "Commonalities" section, as in the case of specific physical problems the assumptions will often be shared by all family members.

For each of the variabilities in the CA there is a parameter of variation and a binding time. The parameter of variation specifies the type of the possible values for the variability. The binding time is the time in the software lifecycle when the variability is fixed. The binding time could be during specification of the requirements (specification time), or during building of the system (build time), or during execution of the system (run time). It is possible to have a mixture of binding times. For instance, a parameter of variation could have a binding time of "specification or build" to represent that the parameter could be set at specification time, or it could be postponed until the given family member is built. The presence of a DSL allows postponing the binding until build time. A DSL may also specify that the binding will be postponed until run time.

The traceability matrix takes the role of showing the relationship between the terminology, goals, theories, assumptions and variabilities. This matrix is important so that change can be tracked through the CA document. A traceability matrix with the same purpose was introduced in [19, 20]. The matrix summarizes the same information as the related commonality field that is suggested for each variability in the Weiss approach [17].

4 A Family of Mesh Generators

The sections that follow illustrate portions of a CA and a DSL for a multi-purpose family of SC tools for mesh generation. A mesh is a discretization of a geometric domain into small simple shapes, such as line segments in 1D, triangles or quadrilaterals in 2D, and tetrahedra or hexahedra in 3D. The principal application of interest for the current study is the finite element method. The files created by a mesh generator (MG) must describe the following: how the domain is decomposed into cells; the material properties; and the boundary conditions on the domain, with respect to applied tractions, prescribed displacements and fixity. The examples are based on several references [21–23]. To simplify the presentation, all MGs in this family are assumed to share the commonality of generating only two-dimensional structured meshes.

4.1 Goal

The goal statements listed below are functional objectives that an MG used as a finite element pre-processor should achieve. Each of the goals specifies the type of input information that is to be provided and then gives a description of the output that will be generated. The output will be in a format that can be understood by a finite element analysis program. Each of the goals is given a unique number (G?, where ? is a natural number) and a unique label (G*, where * is a string of characters). These numbers and labels will be used to reference the goal statements.

G1 (GMesh) Given a spatial domain (Ω), generate and output a mesh (M) that covers this domain.

G2 (GMaterial) Given information on the materials, material properties and the locations of the different materials, transform this information to agree with M and output the results.

G3 (GBoundCond) Given information on the boundary condition types, values and locations, transform this information so that it agrees with M and output the results.

G4 (GSystInfo) Given system information, such as numerical values needed by the subsequent processing step (for instance initial time, time step size, degree of implicitness, etc.) appropriately transform and output this information.

4.2 Theoretical Model

In this section the goals are refined through the introduction of theoretical models. These models are specified by listing the required input and characterizing the generated output. The inputs and outputs are given in terms of the data required and generated, but the specification does not address the format of the data; format will be left as a variability. In the following $\mathbb{R}^2$ refers to a tuple of two real number ($\mathbb{R} \times \mathbb{R}$). Additional types and predicates are defined as needed. Each of the theoretical models is given a unique number and a unique label. Due to space considerations, only the model T1 corresponding to G1 is reproduced below.

T1 (TMesh) The theoretical model in this section describes how the original region is partitioned into elements of a mesh. The source of the requirements for a mesh can be found in [24].

Input Ω: regionT

Output M: set of elementT

The following must be true of the output M:

1. $\Omega = \Omega^* = \{\cup E : \text{elementT} | E \in M : E\}$ (The union of all of the sets in M exactly cover Ω.)
2. $\forall E : \text{elementT} | E \in M : \mathit{interior}(E) \neq \emptyset$ (The interior of every element in M is non empty.)
3. $\forall E_1, E_2 : \text{elementT} | E_1 \in M \wedge E_2 \in M \wedge E_1 \neq E_2 : \mathit{interior}(E_1) \cap \mathit{interior}(E_2) = \emptyset$ (The interior of the elements in M are pairwise disjoint.)

Supporting Definitions

regionT $= \{S : \text{set of } \mathbb{R}^2 | \mathit{connected}(S) \wedge \mathit{closed}(S) : S\}$

elementT = regionT

closed : set of $\mathbb{R}^2 \rightarrow$ boolean
closed$(S) \equiv$ This function returns *true* if the set S is closed, where closed means that the boundary of the region is included in the set.

connected : set of $\mathbb{R}^2 \rightarrow$ boolean
connected$(S) \equiv$ This function returns *true* if the set S cannot be separated into two disjoint sets, where neither of these sets is empty and the union of the two sets gives the original set S.

interior : set of $\mathbb{R}^2 \rightarrow$ set of $\mathbb{R}^2$
interior$(S) \equiv$ Given a closed set this function returns the interior of the set, which means all of the points inside the region, excluding the boundary.

4.3 Variabilities

The variabilities and associated parameters of variation for an MG are best summarized in tables, such as Table 1. The parameters of variation column lists the valid type for the variability. Binding times are not shown in the table because for the variabilities in this example all of the potential binding times (specification, build or run time) are valid options. The type for the file format is an XSLT (Extensible Stylesheet Language Transformation) file because an XSL file can be used to organize mesh data in different output file formats. There are two steps involved in the output file production process. First, after the mesh is produced, all the relevant data that may be useful in a finite element program will be stored in a separate XML document in a pre-defined structure. Second, according to the structure definition of the XML document that contains the mesh data, the XSL stylesheet file(s) must be created to specify the output file formats. The XSL files can be specified either at specification time or at run time. If the stylesheet is parsed without errors, the transformation process is performed automatically, and the output files will be created as indicated by the stylesheet. The XSLT processor adopted in the proof of concept implementation [23] was Xalan.

Table 1. Example Variabilities for a Family of MGs

Variability	**Parameter of Variation**
Local Topology Template	{Quad, Triangle1, ... , Triangle9} as shown in Figure 2
Number of nodes for an element	ℕ
Number of degrees of freedom at a node	ℕ
Material property names	Set of string
Types of boundary conditions	Set of {Neumann, Dirichlet, Mixed}
Output file format	Specified using XSLT

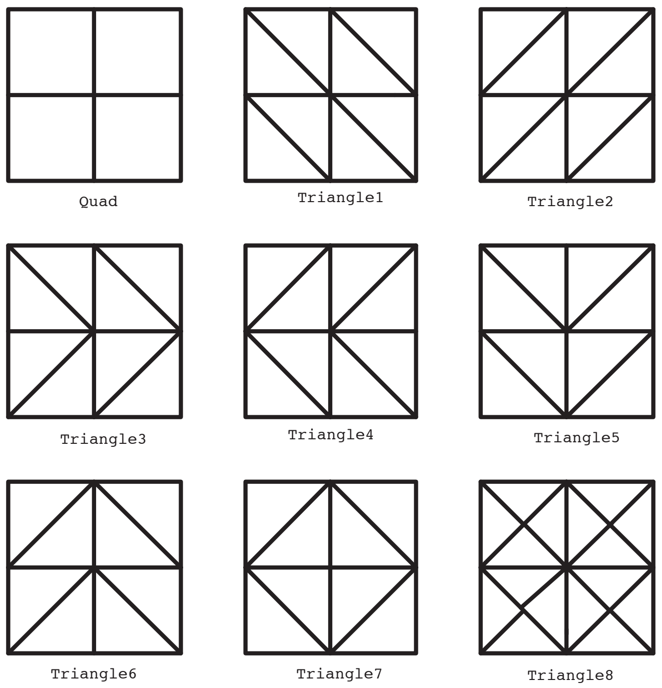

Fig. 2. Local Topology Patterns for Structured Meshes

4.4 DSL for a Mesh Generator Generator

In a proof of concept implementation [23] the DSL is developed as an XML document that customizes a Java object. XML data binding is used, in the current case via jaxb, to automatically generate Java code that represents the information in the XML document as an object in the computer's memory. This object is then used to customize the interface of the general purpose MG as it is loaded. This reference to a general purpose MG refers to a program that has run time binding for all of the variabilities in the family; that is, all of the variabilities are customizable at run time. The general purpose MG corresponds to an empty XML specification, as in the case of an empty specification no pre-run time customizations are made. If the XML specification sets any variabilities, then the corresponding portion of the user interface will show the specified values and not allow further modification by the user. Although not fully explored in the proof of concept implementation, the graphical user interface could provide a means to easily specify family members, since marshalling (converting the XML object to an XML document) could be used after the run-time variabilities have been set to generate an XML document with build time bindings.

Figure 3 is a sample specification of the element type for the mesh.This specification describe the triangular elements that the MG will generate. Each element consists of three nodes specified to be located at the three triangular vertices. The location of a node within an element is specified using the natural coordinate system, where the natural coordinates are based on fractions of the total area of the triangle. Although the natural coordinates consist of three real values, only two values are independent, as the definition of natural coordinates requires that the three values sum to 1.

```
<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>
            <node id="3">
                <location>0,0,1</location>
            </node>
        </nodeGeo>
    </geometrySpec>
</elementSet>
```

Fig. 3. XML specification of an element type

5 A Family of Virtual Laboratories

A virtual material testing laboratory is defined as a flexible environment that simulates real world material experiments. This example fits in the class of program families for specific physical problems. The example below, which is extracted from [25], is for a family of materials, which includes elastic, viscous, viscoplastic and plastic solids. The sections that follow summarize the "Goal" and "Theoretical Model" from the CA for a family of material models. To save space data definitions (D_*), assumptions (A_*) and variabilities (V_*), where * is a label, are not defined in this paper. Following the theoretical model, a portion of the DSL for this family is presented.

5.1 Goal

Number:	G1
Label:	G_StressDetermination
Description:	Given the deformation history of a material particle, determine the internal stress within the material particle.
Related Item:	T1

5.2 Theoretical Model

Number:	T1
Label:	T_ConstitEquation
Related Items:	A_CauchyStress, A_DeformationHistory, A_AdditivityPostulate, A_PerzynaConstit, A_ElasticConstit, A_DescriptionOfMotion, V_MaterialProperties
Input:	$\boldsymbol{\sigma}_0 : \mathbb{R}^6$ units of stress $t_{beg} : \mathbb{R}$ units of time $t_{end} : \mathbb{R}$ units of time $\dot{\boldsymbol{\epsilon}}(t) : \{t : \mathbb{R} \| t_{beg} \leq t \leq t_{end} : t\} \rightarrow \mathbb{R}^6$ units of 1/time mat_prop_val : string $\rightarrow \mathbb{R}$ $E : \{x : \mathbb{R} \| x \geq 0 : x\}$ units of stress $\nu : \{x : \mathbb{R} \| 0 < x \leq 0.5 : x\}$
Output:	$\boldsymbol{\sigma}(t) : \{t : \mathbb{R} \| t_{beg} \leq t \leq t_{end} : t\} \rightarrow \mathbb{R}^6$ such that $\dot{\boldsymbol{\sigma}} = \mathbf{D}\left(\dot{\boldsymbol{\epsilon}} - \gamma < \phi(F(\boldsymbol{\sigma}, \kappa)) > \frac{\partial Q(\boldsymbol{\sigma})}{\partial \boldsymbol{\sigma}}\right)$ and $\boldsymbol{\sigma}(t_{beg}) = \boldsymbol{\sigma}_0$ where $< \phi(F) >= \begin{cases} \phi(F) & \text{if} \quad F > 0 \\ 0 & \text{if} \quad F \leq 0 \end{cases}$ and the components of $\boldsymbol{\sigma}$ have units of stress

The theoretical model states that given the material property values (mat_prop_val), including E (Elastic modulus) and ν (Poisson's ratio), and the input parameters describing the deformation ($\dot{\boldsymbol{\epsilon}}(t)$) of the material particle over the relevant history from time t_{beg} to time t_{end} solve the governing differential equation for the stress history ($\boldsymbol{\sigma}(t)$) with the initial condition that the stress tensor $\boldsymbol{\sigma}(t_{beg}) = \boldsymbol{\sigma}_0$. The governing differential equation depends on the elastic constitutive matrix ($\mathbf{D} : \mathbb{R}^{6\times6}$), the fluidity parameter ($\gamma : \mathbb{R}$), the function ϕ ($\phi : \mathbb{R} \rightarrow \mathbb{R}$), the yield function ($F(\boldsymbol{\sigma}, \kappa) : \mathbb{R}^6 \times \mathbb{R} \rightarrow \mathbb{R}$), the plastic potential function ($Q(\boldsymbol{\sigma}) : \mathbb{R}^6 \rightarrow \mathbb{R}$), the stress tensor ($\boldsymbol{\sigma} : \mathbb{R}^6$) and the hardening parameter ($\kappa : \mathbb{R}^6 \rightarrow \mathbb{R}$), which measures the accumulated strain. In the theoretical model T1 the equation $F = 0$ defines a surface in 6 dimensional stress space. Inside the surface ($F < 0$) the material response will be purely elastic, outside the surface the response is viscoplastic. The plastic potential Q gives the direction of the viscoplastic strain increment. Details on material behaviour modelling can be found in [26–28].

5.3 DSL and Implementation of MatGen

A program called MatGen was developed to automatically generate source code to simulate a new material model [25]. A new material model is defined in the DSL by the four functions F, Q, κ, and ϕ and the constant γ. The numerical algorithms that are used to simulate material models of this form require partial derivatives of these functions, such as $\frac{\partial Q}{\partial \boldsymbol{\sigma}}$. Therefore, the code generator will need to derive additional expressions from the functions given in the DSL description. The user should not be required to compute all of the expressions needed for a numerical simulation, MatGen should perform this tedious and error prone work for the user. A compiler is needed which can compute the needed derivatives and output source code. The Maple computer algebra system provides such a compiler. The DSL designed for MatGen consists of a small subset of the language accepted by Maple. Maple performs the symbolic differentiation and then converts from the mathematical expressions into C expressions using the "CodeGeneration" function. These C expressions are inlined into the C++ class defining the material model. This C++ class can then be used by another program, such as MatCalc [25], to simulate virtual experiments or other computational mechanics problems. A portion of the DSL is listed below for the expression for F. The listing is in Backus-Naur form (BNF), extended with some regular expression operations. The extended BNF syntax includes [...] to denote optional portions of expressions and + to denote one or more repetitions. Bold font is used to represent the terminal tokens.

$\langle$expression$\rangle \rightarrow \langle$number$\rangle$ |
($\langle$expression$\rangle$**)** |
$\langle$expression$\rangle$**^**$\langle$expression$\rangle$ |
$\langle$expression$\rangle$*****$\langle$expression$\rangle$ |
$\langle$expression$\rangle$**/**$\langle$expression$\rangle$ |
$\langle$expression$\rangle$**+**$\langle$expression$\rangle$ |
$\langle$expression$\rangle$**−**$\langle$expression$\rangle$ |
−$\langle$expression$\rangle$ |
sin($\langle$expression$\rangle$**)**|**arcsin(**$\langle$expression$\rangle$**)**|**cos(**$\langle$expression$\rangle$**)**|**arccos(**$\langle$expression$\rangle$**)** |
ln($\langle$expression$\rangle$**)**|**log(**$\langle$expression$\rangle$**)** |
$\langle$simulation-variable-F$\rangle$ | $\langle$user-defined-constants$\rangle$
$\langle$number$\rangle \rightarrow$[$\langle$sign$\rangle$]$\langle$digit$\rangle$+[$\langle$decimal-point$\rangle\langle$digit$\rangle$+]
$\langle$sign$\rangle \rightarrow$**+**|**-**

⟨decimal-point⟩→**.**
⟨string⟩→⟨character⟩+
⟨character⟩→**a...z**|**A...Z**
⟨digit⟩→**0**|**1**|**2**|**3**|**4**|**5**|**6**|**7**|**8**|**9**
⟨simulation-variable-F⟩→**Kappa**|⟨simulation-variable-stress⟩|⟨simulation-variable-stress-macros⟩
⟨simulation-variable-stress⟩→**SigmaXX**|**SigmaYY**|**SigmaZZ**|**SigmaXY**|
SigmaYZ|**SigmaXZ**
⟨simulation-variable-stress-macros⟩→**Sxx**|**Syy**|**Szz**|**Sxy**|**Syz**|**Sxz**|**Sm**|**J2**|**J3**|**q**
⟨user-defined-constants⟩→⟨string⟩

The expression for $F = F(\boldsymbol{\sigma}, \kappa)$ is a function that can only depend on the 6 components of the stress tensor $\boldsymbol{\sigma}$ (**SigmaXX**, **SigmaYY**, etc.), the hardening parameter (**Kappa** (κ)), stress macros (**Sxx**, **Syy**, etc.) and user defined constants. The stress macros are functions that often arise in continuum mechanics, such as the stress invariants and the deviatoric stress invariants. These macros only reference the allowed stress components. As an example, **Sm** = **SigmaXX** + **SigmaYY** + **SigmaZZ**. The user defined constants correspond to the material properties needed by the material model.

6 Concluding Remarks

This paper motivates how the quality of SC applications can be improved by developing them as a program family. The motivation for development of SC software as program families comes from the potential benefits of reduced development time, improved reusability, improved usability, reduced maintenance effort and the ability to cope with complexity. The justification for the appropriateness of SC software to the program family approach comes from the mathematical underpinnings of SC and the satisfaction of the redevelopment, oracle and organizational hypotheses. The program family strategy is further justified by demonstrating a methodology for its application. The proposed methodology consists of determining the scientific computing family of interest, then performing a CA to document the terminology, commonalities (including goals and theoretical models) and variabilities (including assumptions, input variabilities and output variabilities). Following this, the CA is used as a basis for a DSL from whose instances program family members are automatically generated. Further demonstration of the value of the program family approach is given in the form of two examples. One example is a family of MGs, which is an example drawn from the the set of domains of multi-purpose SC tools. The other example, a family of virtual material testing laboratories, comes from the set of domains of special purpose physics solvers.
In the future, the use of the program family approach can be further justified by providing additional examples. In particular the examples could emphasize the quality of performance to a greater extent than the current examples, by taking advantage of the simplifications possible for the more specialized family members. The current examples have separated multi-purpose tools and specific physics solvers, but these two sets of domains are related, since tools are used to solve physical problems; therefore, integration of multiple families to produce SC applications should be a topic of future investigation.

Acknowledgements

The financial support of Shared Hierarchical Academic Research Computing Network (SHARCNET) and the Natural Sciences and Engineering Research Council (NSERC) are gratefully acknowledged. The comments from the anonymous reviewers are also greatly appreciated.

References

1. Clements, P., Northrop, L.M.: Software product lines: practices and patterns. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)
2. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer-Verlag (2005)
3. Carette, J.: Gaussian elimination: A case study in efficient genericity with MetaOCaml. Science of Computer Programming **62**(1) (2006) 3–24
4. Whaley, R.C., Petitet, A., Dongarra, J.J.: Automated empirical optimization of software and the ATLAS project. Parallel Computing **27**(1–2) (2001) 3–35
5. Veldhuizen, T.L.: Arrays in Blitz++. In: Proceedings of the 2nd International Scientific Computing in Object-Oriented Parallel Environments (ISCOPE'98), Lecture Notes in Computer Science, Springer-Verlag (1998)
6. Rice, J.R., Boisvert, R.F.: From scientific software libraries to problem-solving environments. IEEE Computational Science & Engineering **3**(3) (Fall 1996) 44–53
7. Owen, S.J.: A survey of unstructured mesh generation technology. In: Proceedings 7th International Meshing Roundtable, Dearborn, MI (October 1998)

8. Weiss, D.M.: Defining families: The commonality analysis. Submitted to IEEE Transactions on Software Engineering (1997)
9. Jones, R.E.: The QMESH mesh generation package. Association for Computing Machinery SIGNUM Newsletter, vol. 10, no. 4, pp. 31-34 (December 1975)
10. Blacker, T.D., Kerr, R.A., Knupp, P., Leland, R.W., Melander, D.J., Mitchell, S.A., Owen, S.J., Sheperd, J.F., Tautges, T.J., White, D.R., Benzley, S., Borden, M.J., Jankovich, S.R., Kraftcheck, J., Lu, Y., Meyers, R.J., Stephenson, M., Storm, S., Nielsen, E., Yoeu, R.: Cubit mesh generation environment, volume 1: User's manual. Technical Report SAND94-1100, Sandia National Laboratories, Albuquerque, New Mexico (May 1994)
11. Tautges, T., Blacker, T., Mitchell, S.: The whisker weaving algorithm: A connectivity-based method for constructing all-hexahedral finite element meshes (1995)
12. Smith, W.S., Stolle, D.F.E.: Numerical simulation of film casting using an updated Lagrangian finite element algorithm. Polymer Engineering and Science **43**(5) (2003) 1105–1122
13. Berti, G., Bader, G.: Design principles of reusable software components for the numerical solution of PDE problems. presented at the the 14th GAMM-Seminar Kiel on Concepts of Numerical Software (January 1998)
14. Chen, C.H.: A software engineering approach to developing mesh generators. Master's thesis, McMaster University, Hamilton, Ontario, Canada (2003)
15. ElSheikh, A.H., Smith, W.S., Chidiac, S.E.: Semi-formal design of reliable mesh generation systems. Advances in Engineering Software **35**(12) (2004) 827–841
16. Cuka, D.A., Weiss, D.M.: Specifying executable commands: An example of FAST domain engineering. Submitted to IEEE Transactions on Software Engineering (1997) 1 – 12
17. Weiss, D., Lai, C.: Software Product Line Engineering. Addison-Wesley (1999)
18. Smith, W.S.: Systematic development of requirements documentation for general purpose scientific computing software. In: Proceedings of the 14th IEEE International Requirements Engineering Conference, RE 2006, Minneapolis / St. Paul, Minnesota (2006) 209–218
19. Smith, W.S., Lai, L., Khedri, R.: Requirements analysis for engineering computation. In Muhanna, R., Mullen, R., eds.: Proceedings of the NSF Workshop on Reliable Engineering Computing, Savannah, Georgia (2004) 29–51
20. Smith, W.S., Lai, L.: A new requirements template for scientific computing. In Ralyté, J., Ȧgerfalk, P., Kraiem, N., eds.: Proceedings of the First International Workshop on Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes, SREP'05, Paris, France, In conjunction with 13th IEEE International Requirements Engineering Conference (2005) 107–121
21. Smith, W.S., Chen, C.H.: Commonality and requirements analysis for mesh generating software. In Maurer, F., Ruhe, G., eds.: Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering (SEKE 2004), Banff, Alberta (2004) 384–387
22. Smith, W.S., Chen, C.H.: Commonality analysis for mesh generating systems. Technical Report CAS-04-10-SS, McMaster University, Department of Computing and Software (2004)
23. Cao, F.: A program family approach to developing mesh generators. Master's thesis, McMaster University (April 2006)
24. Frey, P.J., George, P.L.: Mesh Generation Application to Finite Elements. Hermes Science Europe Ltd. (2000)
25. McCutchan, J.: A generative approach to a virtual material testing laboratory. Master's thesis, McMaster University (2007)
26. Malvern, L.E.: Introduction to the Mechanics of Continuous Medium. Prentice Hall (1969)
27. Mase, G.E.: Schaum's Outline of Theory and Problems of Continuum Mechanics. McGraw-Hill Publishing Company (1970)
28. Zienkiewicz, O., Taylor, R.L.: The Finite Element Method For Solid and Structural Mechanics. 6th edn. Elsevier Butterworth-Heinemann (2005)