A Systematic Process for Defining Meshing Tool Software Product Line Domain Model

Pedro O. Rossel
Departamento de Ingeniería Informática, Universidad Católica de la Santísima Concepción
Alonso de Ribera 2850, Concepción, Chile
prossel@ucsc.cl

María Cecilia Bastarrica, Nancy Hitschfeld-Kahler
DCC, Universidad de Chile
Avda Blanco Encalada 2120, Santiago, Chile
{*cecilia,nancy*}*@dcc.uchile.cl*

Abstract

Once an organization decides to develop a software product line (SPL), one of the first activities that needs to be done is to build a domain model. Here, commonalities and variabilities are identified, as well as the particular characteristics that products of the SPL will have. Although there exist some methods proposed for domain modeling, they are general and not specifically designed for scientific software, let alone for meshing tools. Meshing tools are highly complex software for generating and managing geometrical discretizations. Due to this complexity, they have generally been developed by end users with ad-hoc methodologies and not applying well established software engineering practices. Nevertheless, many meshing tools with varying degrees of variability have been developed over the years, making them a good application domain for SPL. This paper proposes a systematic process for building the domain model, specially suited for the case of a meshing tool SPL. We formally define the structure of the domain model, the process for building this model in a rigorous way, and we apply it to produce a meshing tool domain model. Both, the model and the process, are described and exemplified along the paper.

1. Introduction

According to Northrop and Clements [22], a software product line (SPL) is a set of software intensive systems that share a managed set of characteristics, and that satisfies the needs of a particular market segment or mission, being developed using a set of common core assets in a preestablished fashion. These core assets include the product line architecture, reusable software components, and domain models, among others. In a SPL we can identify two main technical stages [26]: domain engineering where reusable core assets are developed, and application engineering where particular products are built by combining the assets already developed. Understanding and identifying common and variable aspects play a central role during the domain engineering stage. Commonalities are requirements that must hold for all products in the SPL, while variabilities are requirements that may or may not be present in a particular product, and as such define how SPL products may vary [35].

In the context of SPL, domain analysis is the first step within the domain engineering stage. It includes scoping and domain modeling. Scoping consists of defining which products are part of the SPL and which are not. Domain modeling is the process through which commonalities and variabilities are identified, captured and organized in a domain model with the purpose of characterizing the domain [25].

Meshing tools are sophisticated software due to the complexity of the concepts involved and the large number of interacting elements they manage. Meshing tools complexity mainly relies on the components involved, as is the case for most scientific computing software. Provided that meshing tools are used in a variety of different application domains, they may require slightly different functionality, algorithms for implementing this functionality, data representation, or format of the data used as input or output [23]. Also depending on the application domain, it may be required to have two or three dimensional meshes, each one using different types of basic modeling elements. For example, Figure 1 shows a triangulation surface mesh of a brain at the left and an internal view of the associated volume mixed-element mesh (compose of hexahedra, pyramids, prisms and tetrahedra) at the right; this application is used for brain surgery [19].

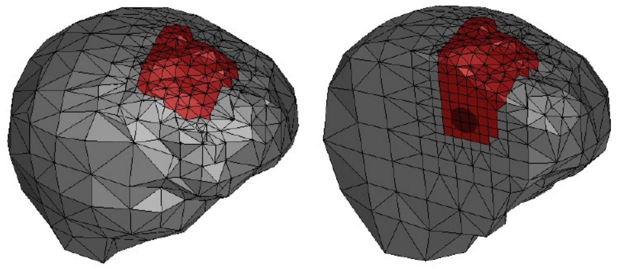

Figure 1. Modeling the brain

These tools have usually been developed with ad hoc methodologies, focusing on attributes like performance instead of reuse for building them. Without reuse in mind, every new tool needs to be developed from scratch even though it may involve algorithms already implemented and data structures already designed, all of them also used and tested. Other authors have done some efforts in the direction of reuse in the meshing tool domain [28, 29, 30, 31], but it is not a common practice yet. It is necessary to count on a reuse framework if the potential gains in productivity and quality SPL promises are to be achieved.

According to Smith and Chen [29] and Bastarrica and Hitschfeld-Kahler [1], most mesh generators can be abstracted as: input information, calculate a domain discretization (mesh), refine and/or improve the mesh, and finally output the results. These shared steps have been identified as commonalities among all members of a meshing tool family. On the other hand, variabilities may be seen in two different ways: by including or excluding certain steps in a mesh processing, or by providing alternative implementations or algorithms for realizing the same chosen functionality depending on the precise processing required and its particular characteristics (performance, memory use, amount and distribution of the mesh points, and scalability factors for managing complex domains, among others).

We have developed several meshing tools [5, 11, 12, 13, 20] following the Object-Oriented paradigm. Even though this paradigm promotes reuse, it has reached a point in its development and use, that it is difficult to obtain higher quality, lower cost, and shorter time in development than it has currently achieved [10]. If we need to develop other meshing tools for different needs, we can use the knowledge obtained during past developments, reusing the existing core assets as source code, architectures, and documentation among others, and considering the existing commonalities and variabilities, all of them within an organized framework such as that provided by SPLs.

In this paper we propose a process for building a domain model that is specially suited for the meshing tool domain. It integrates lexicon, features, goals and scenarios as a means for capturing the domain characteristics and we organize them in a formal model. We provide a process that organizes the way these elements are gathered and combined into a unified domain model, as well as clear iteration or termination conditions based on model consistency and completeness. We develop a domain model for the meshing tool SPL following the proposed process.

Some authors have already approached building meshing tools with SPL concepts in mind [1, 2, 29, 30], but to the best of our knowledge, none of them has focused on domain model with a systematic process specially designed for this particular domain. Even though there are other methods for building a domain model, we decided to create our own customized process because none of those available directly applies for our domain. Some of them are general and need tailoring [4, 17, 35], other ones are domain specific and need adapting [21], and other ones are product oriented and they need to count on many products already developed [3].

The main contribution of this paper is twofold: first, a rigorous process for building the domain model, and second, a formal domain model definition specially suited for the meshing tool domain.

The rest of the paper is organized as follows. In Section 2 we provide the definition of the domain model, as well as the proposed process for building it. Section 3 describes the process applied and the domain model obtained for the meshing tool SPL. Related work about both domain analysis methodologies in general and for developing meshing tools in particular are discussed in Section 4. Finally, conclusions and future work are presented in Section 5.

2. Domain Model and Process

Our proposed domain model will include, as many other proposals do, features, goals, scenarios and lexicon, along with the relationships among elements of different kinds. We define our domain model in Section 2.1, its formalization in Section 2.2 and the process for building it in Section 2.3.

2.1. Elements of the Domain Model

Our proposed domain model structure is based on features, goals, scenarios and lexicon. The definition of features, goals and scenarios are inspired in the work of Park et al. [24].

The lexicon defines the domain vocabulary, and allows a better and shared understanding for all stakeholders involved in the domain [8]. Several other authors agree on the need of counting on a lexicon in the development of the SPL [7, 15, 18, 27, 29, 31, 35].

In the context of a product line, a goal is an objective of the business, the organization or the system that some stakeholder hopes to achieve with that product line. A scenario is a possible behavior limited to a set of interactions with the purpose of achieving some goals with the product line. Thus, a scenario is generally composed of a sequence of one or more actions corresponding to user or system interactions with products of a product line. Features are characteristics and abstractions of product functionalities, parameters and data storages in a SPL visible for stakeholders, and thus they can be viewed as effects achieved by some product behavior (external or internal). A feature is an attribute of a system that directly affects end-users [15].

2.2. Formalization of the Domain Model

For defining the elements that form part of our domain model and their relationships we use Z schemas [32]. Z schemas allow us to define elements that form part of a model as well as the invariants in which they are involved.

We consider *GOAL*, *FEATURE*, *ACTION*, *DESCRIPTION* and *CHAR* as primitive types. We define *SCENARIO* as a sequence of actions, and *TYPEF* as an ennumeration of the three different types of features.

$$[GOAL, FEATURE, ACTION, DESCRIPTION]$$

$$SCENARIO == \mathrm{seq}\, ACTION$$

$$\begin{array}{l} TYPEF ::= GroupedFeature \mid SolitaryFeature \\ \qquad\qquad \mid RootFeature \end{array}$$

Since features can be defined as mandatory (commonalities), or optional and/or alternative (variabilities), we define Feature as a name and a type, as specified in the next schema.

$$\begin{array}{|l} \hline Feature \\ name : \mathrm{seq}\, CHAR \\ type : TYPEF \\ \hline \end{array}$$

The schema *DomainModel* defines the elements that form part of our domain model. In this schema the variables are *Goals*, *Scenarios*, *Features*, *Actions* and *Lexicon*. The schema also includes the relationships between goals and scenarios (*By_Scenario* [a]), and between scenarios and features (*By_Feature* [b]); these relationships are inspired by the work of Kaindl [14]. Notice that defining the Lexicon as a function of *Feature* to *DESCRIPTION* is a simplification, since it would eventually be necessary to define other concepts of the application domain besides features. Defining the *Feature* Model just as a set of features is also a simplification, but their structure neither affects the *DomainModel* definition, nor its consistency specification.

$$\begin{array}{|ll} \hline DomainModel & \\ Goals : \mathbb{P}\, GOAL & \\ Scenarios : \mathbb{P}\, SCENARIO & \\ Features : \mathbb{P}\, Feature & \\ Actions : \mathbb{P}\, ACTION & \\ Lexicon : Feature \pfun DESCRIPTION & \\ By\_Scenario : GOAL \leftrightarrow SCENARIO & [a] \\ By\_Feature : SCENARIO \leftrightarrow Feature & [b] \\ \hline Actions = \bigcup \mathrm{ran}\, s \mid s \in Scenarios & [c] \\ \mathrm{dom}\, Lexicon \subseteq Features & [d] \\ \mathrm{dom}\, By\_Scenario \subseteq Goals & [e] \\ \mathrm{ran}\, By\_Scenario \subseteq Scenarios & [f] \\ \mathrm{dom}\, By\_Feature \subseteq Scenarios & [g] \\ \mathrm{ran}\, By\_Feature \subseteq Features & [h] \\ \hline \end{array}$$

The constraints that any *DomainModel* must satisfy are those invariants stated in the lower part of the schema. Among them we find the following. The identified actions are those derived from already identified scenarios [c]. There could be features that are not part of the lexicon [d] just because they have not been defined yet. Only those goals, scenarios and features that have been identified as part of the *DomainModel* can be related by the *By_Scenario* and *By_Feature* relations [e,f,g,h].

Although we may have a transient inconsistent domain model, at the end it needs to be consistent. The *ConsistentDomainModel* schema refines the prior one by adding certain constraints. It includes the previous schema *DomainModel*, and also includes the definition of another relationship (*Attached* [i]) between actions and features that are necessary for fulfilling them.

$$\begin{array}{|ll} \hline ConsistentDomainModel & \\ DomainModel & \\ Attached : ACTION \leftrightarrow Feature & [i] \\ \hline \mathrm{dom}\, Lexicon = Features & [j] \\ \mathrm{dom}\, By\_Scenario = Goals & [k] \\ \mathrm{dom}\, By\_Feature = \mathrm{ran}\, By\_Scenario & \\ \qquad\qquad = Scenarios & [l] \\ \mathrm{ran}\, By\_Feature = Features & [m] \\ \mathrm{dom}\, Attached = Actions & [n] \\ \mathrm{ran}\, Attached = Features & [o] \\ \hline \end{array}$$

Within a *ConsistentDomainModel*, all identified *Features* are described in the *Lexicon* [j], all identified *Goals* have a series of related scenarios [k], all identified *Scenarios* contribute to a certain goal and may also be fulfilled with the set of identified *Features* [l], and all *Features* contribute to the fulfillment of at least one scenario [m]. Finally, all identified *Actions* should be attached to at least one feature [n], and all *Features* are attached to at least one action [o]. We will use these conditions for checking consistency, one of the termination conditions of our proposed process.

2.3. Domain Model Construction Process

Figure 2 summarizes the products of the domain model process and their relationships.

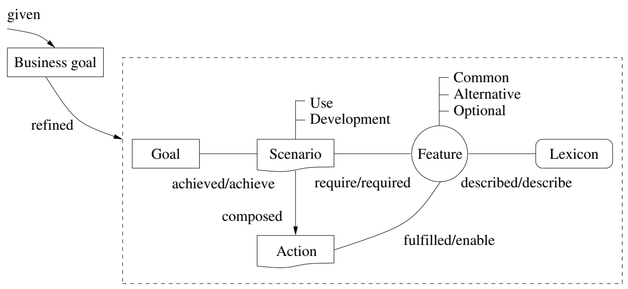

Figure 2. Domain model artifacts

The business goal establishes the purpose for developing products as a family. This goal is unique for the whole SPL, but there may be several particular goals. We distinguish two types of scenarios: development scenarios that are those followed whenever a product of the SPL is built, and use scenarios that are those followed by particular products once they are executed. Features are those data storage, parameters or functionalities identified for the potential products in the SPL; they may be either common (mandatory), optional or alternative. The lexicon involves terms necessary for understanding the domain and they can be any relevant thing into the domain.

With respect to the notation, we use a feature model for specifying features following the notation proposed by Czarnecki and Eisenecker [8], structured English for goals and scenarios, and natural language for lexicon. Even though the notations for goals, scenarios and lexicon are difficult to validate formally and automatically, they are used because they are easy to work and understand for any stakeholder, and previous training is not required.

Figure 3 shows an activity diagram for building the domain model. The domain expert(s) and the domain analyst(s) should interact in order to identify and specify goals, features, scenarios, actions, and terms of the lexicon as well as their relationships considering information from the stakeholders (included domain expert and analyst), available components developed in the domain, external information (e.g. emerging technology within a domain, market information and literature) and systems information (e.g. system documentation and existent systems developed in the domain). Systems information is optional in the activity diagram, meaning that it is not necessary to count on

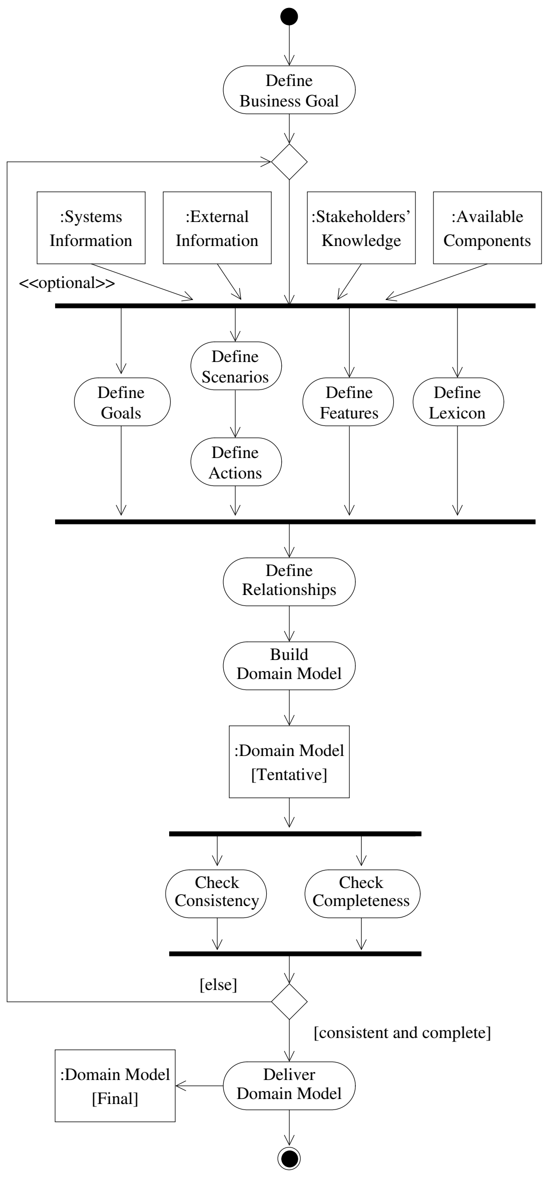

Figure 3. Domain model process

it for building the domain model. It generally occurs in a scenario where there are no products developed (or only a few), and the development of the SPL starts from scratch. Once these activities are done, the domain expert checks for completeness by analyzing if the model elements captured are enough for deeply understanding the domain and building all expected products. Meanwhile the domain analyst checks for consistency by verifying that the domain model satisfies all the consistency conditions indicated in *ConsistentDomainModel* schema, Section 2.2. If any of these conditions (completeness or consistency) does not hold, then the process iterates. Otherwise the domain model is ready and we can proceed to the following steps of the SPL development.

The process is influenced by the characteristics of the meshing tool domain. This domain is stable, and thus it is possible to count on domain experts that are familiar with good software engineering practices. Also, there are several pre-implemented components, already tested and with appropriately documented interfaces so it would not be extremely difficult to identify features from them; in this way features are naturally mapped to data storages, parameters or functionalities. Finally, the binding time for variabilities in meshing tools is fixed to design time, so it is not necessary to apply a completely general domain model method, but a much simpler one suffices.

Another important characteristic, that it is not exclusive for the meshing tool domain but has influenced our domain model process, is that we did not count on a large number of tools already developed, before building the domain model. That is why our process is intensive in the use of stakeholders' knowledge, external information and available components, and does not rely much on the information of existing systems. As a result, our process is feature oriented, and the feature model is our main artifact. Goals, scenarios and actions are useful for supporting the rationale of how the feature model was obtained. Furthermore, is known the fact that a major advantage of discussing a system in terms of features is that they bridge the gap between requirements and technical design decisions [34], i.e. features focus on the problem space and not on the solution space [8].

3. Domain Model for Meshing Tools

In this section, we apply the proposed process to build the meshing tool domain model. We first introduce the Lexicon for understanding the domain. Then we obtain the business goal and after that we define particular goals, some scenarios and features. Several terms of the lexicon are identified and defined. Some goals, scenarios and features are related, but others are not. We proceed then with a second iteration mainly because the feature model was found to be incomplete. In the second iteration we advance in the feature model.

The relationships among goals, features and scenarios are stated in tables, so that consistency checks would result easier. Only when the domain expert(s) and the domain analyst(s) intuitively think that the model could be ready, they proceed to check for termination conditions. Thus finally our domain analyst checks for domain model consistency using the constraints stated in the *ConsistentDomainModel* schema, and the domain expert checks for completeness by determining if the candidate tools of the SPL could be built with the documented elements.

In this point, it is worth mention that the meshing tool domain model presented in this paper is not complete. It is probable that the feature model, business goal and goals are more complete than scenarios and actions. A full meshing tool domain model that includes both characteristics of our developed tools and characteristics of the tools developed for others is beyond the scope of this paper. However, this model can be extended following the process presented in the previous section.

3.1. Lexicon

First of all, it is important to identify the terms that are essential to the domain because they allow the stakeholders to understand the basic concepts and use a common language while building and/or using a product.

In this section we describe a part of the vocabulary used in the meshing tool domain. It is not exhaustive, but it intends to illustrate how the lexicon is defined. This knowledge will be also useful for delimiting the scope of the SPL.

- Mesh: A mesh is a discretization of a domain geometry into simple cells. This discretization can be either composed by a unique element type, such as triangles, tetrahedra or hexahedra, or by a combination of different element types.

- Meshing Tool: It is a piece of software for generating and managing meshes.

- Refinement Criteria: These criteria control the number of points and size of the mesh elements, e.g., MaximumEdgeLength (all the edges must have a length less or equal than a threshold value) and MaximumArea (all the elements must have an area less or equal that a threshold value).

- Processes involved in mesh generation. The user of meshing tools usually specifies a domain geometry, some physical values associated to this geometry and some quality criteria, and wants to get an appropriate mesh in order to simulate some phenomena that occur in this domain. In order to do this, almost any meshing tool requires one or more of the following algorithms:

  Generation of Initial Mesh: The meshing tool takes as input a domain geometry and generates as output a discretization that represents the geometry as exact as possible. It may have the same number of final points as the input geometry or a different one.

  Optimization: This is a process that does not insert new points in the mesh in order to improve its quality. The optimization algorithms are strategies that move the current points in an adequate way so that the quality of the elements is improved according to some optimization criteria.

Derefinement: This process allows to generate a coarser mesh in regions with too many points. For example, if a MinimumEdgeLengh criterion is specified, all edges whose length is less than the MinimumEdgeLength value are eliminated.

3.2. Business Goal and Particular Goals

We have identified the business goal and the particular goals for this domain.

Business goal

Developing new meshing tools for different applications with minimum effort.

Goals

We have identified several particular goals in this domain. Here we present nine representative ones.

G1: Generation of good quality meshes for diverse domain geometries that fulfill certain given criteria in specific regions of the domain depending on the particular application requirements.

G2: Generation of meshes with the minimum amount of points that fulfill the application requirements.

G3: Generation of meshes in a reasonable CPU time.

G4: Generation of meshes using an efficient memory management.

G5: Scalability in the number of required mesh points.

G6: Make easy the interchange of different implementations for components of the same type.

G7: Make easy to add a new kind of process to be applied to the mesh.

G8: Generation of meshes that fulfill the requirements of different numerical methods.

3.3. Features

Feature diagram

Figures 4 and 5 show part of our feature model. According to Figure 4, the features **User Interface**, **2D Generate Initial Mesh**, **2D Input**, **2D Output**, **2D Visualize** and **2D Mesh** are common to any 2D meshing tool. **2D Algorithm**, **2D Criterion**, **2D Region**, **2D Evaluate**, **2D Move Boundary** and **2D Postprocess** are optionals, i.e., they can be present or not in a particular 2D meshing tool. The above is also applicable both for common and optional features for **3D** features. Moreover, the interaction styles for user interfaces **Command Language**, **Menu Selection**, **Direct Manipulation** and **Form Fill In** can be all or any subset in a particular tool. Finally, a meshing tool must work with meshes in two dimensions (**2D Meshing Tool**) or three dimension (**3D Meshing Tool**), but not with both.

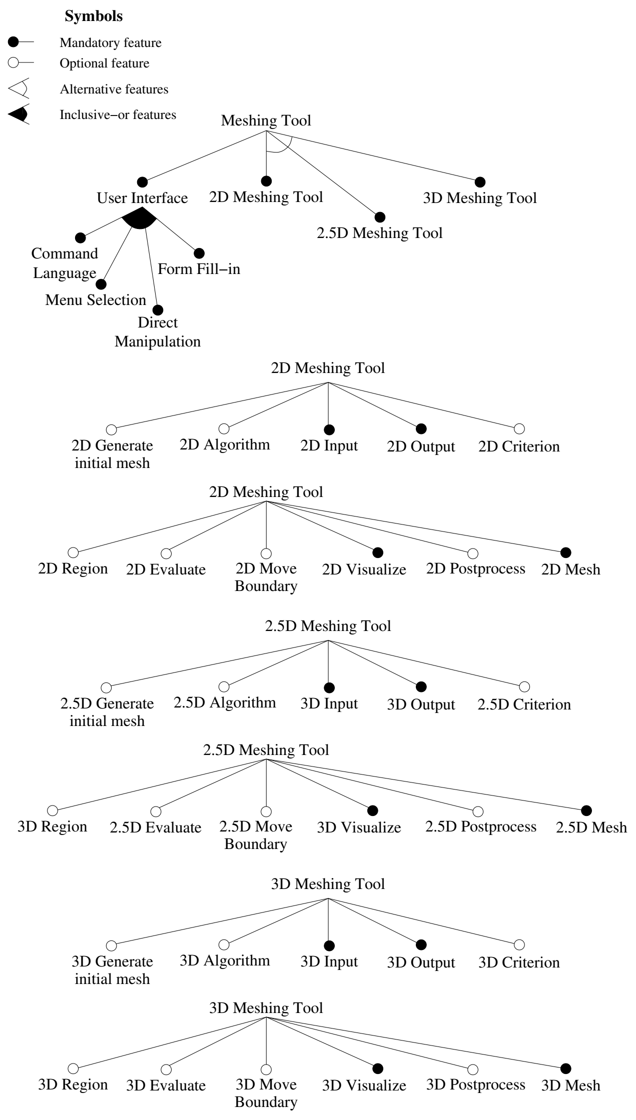

Figure 4. A feature model for meshing tool domain

Feature constraints

Feature models some times need to contain additional constraints [8] or composition rules [15] that cannot be expressed as mandatory and optional feature characteristics, or group characteristics as exclusive-or and inclusive-or groups. Even though these constraints could be represented

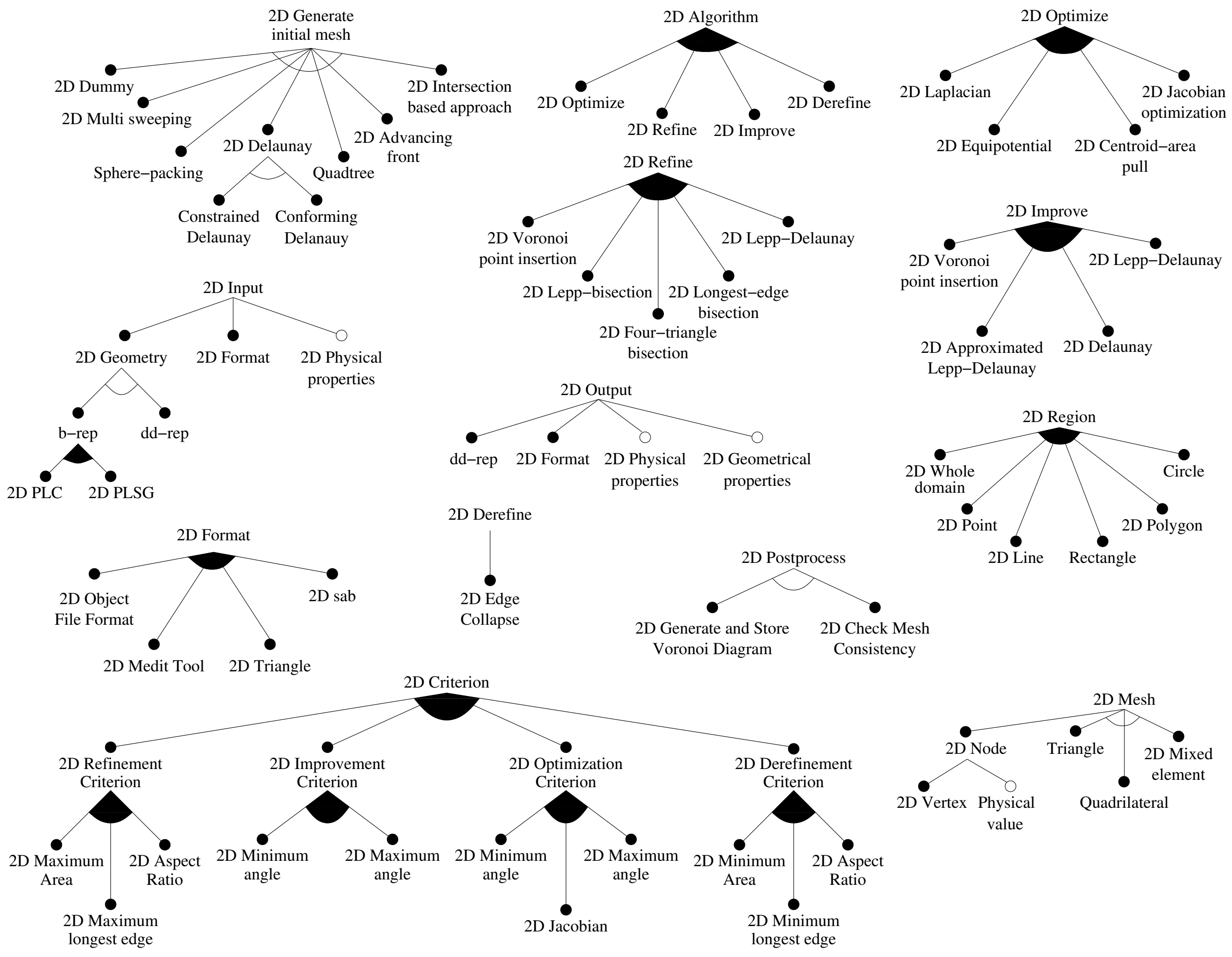

Figure 5. A feature model for meshing tool domain, continuation

in a graphic fashion, we prefer to organize them in a table to enhance understandability of the feature model.

Table 1 shows some of the existent constraints for 2D.

Table 1. Constraints between features

Feature	Constraint	Feature
2D Algorithm	REQUIRES	2D Mesh
2D Algorithm	REQUIRES	2D Region
2D Generate Initial Mesh	REQUIRES	2D Input
2D Lepp-Delaunay	REQUIRES	Triangle
2D	EXCLUDES	3D
2D Lepp-bisection	EXCLUDES	Rectangle
2D Move Boundary	EXCLUDES	2D Region
2D Postprocess	EXCLUDES	2D Region
Quadtree	EXCLUDES	Triangle

3.4. Scenarios and actions

We here detail a list of use and development scenarios for particular meshing tools as well as their corresponding sequence of actions. Notice that some actions that take part in different scenarios have the same identification because they are identical.

Scenarios s0 to s8 are use scenarios, and s9 to s13 are development scenarios. Even though this division is strict, some stakeholders could work with use scenarios for understanding how to build a product of the SPL. The important issue is to classify the scenarios in one of the two kinds.

Notice that each scenario could be applied both two and three dimensions. For this reason, we built generic scenarios and actions.

S0 : Generate Delaunay meshes for convex domains.

- **A1** : Apply an algorithm for reading the geometry in the corresponding format.
- **A2** : Apply a Delaunay algorithm to generate the initial mesh.
- **A3** : Store the mesh in a specified output format.
- **A4** : Visualize the mesh.

S1 : Generate quality Delaunay meshes for PLC domains.

- **A1** : Apply an algorithm for reading the PLC geometry in the corresponding format.
- **A2** : Apply the Conforming Delaunay algorithm in order to generate the initial mesh.
- **A5** : Select refinement and/or improvement criteria, and regions where they will be applied.
- **A6** : Apply the Voronoi Point Insertion algorithm using the specified quality criteria and regions.
- **A7** : If desired, evaluate the quality of the mesh elements.
- **A3** : Store the mesh in a specified output format.
- **A4** : Visualize the mesh.

S2 : Generate quality meshes with a minimal number of final mesh points.

- **A1** : Apply an algorithm for reading the geometry in the corresponding format.
- **A2** : Apply an algorithm to generate the initial mesh.
- **A5** : Select refinement, improvement and/or optimization criteria, and regions where they will be applied.
- **A8** : Apply the refinement, improvement and/or optimization algorithm that minimizes the number of inserted points using the specified quality criteria and regions.
- **A9** : If necessary, apply a derefinement algorithm using the specified derefinement criterion and region.
- **A3** : Store the mesh in a specified output format.
- **A4** : Visualize the mesh.

S3 : Generate meshes with approximated quality as fast as possible.

- **A1** : Apply an algorithm for reading the geometry in the corresponding format.
- **A2** : Apply an algorithm to generate the initial mesh.
- **A5** : Select improvement and/or optimization criteria, and regions where they will be applied.
- **A10** : Apply the fastest improvement and/or optimization approximated algorithm using the specified quality criteria and regions.
- **A3** : Store the mesh in a specified output format.
- **A4** : Visualize the mesh.

S4 : Generate meshes with minimal quality that optimizes the memory used.

- **A1** : Apply an algorithm for reading the geometry in the corresponding format.
- **A2** : Apply an algorithm to generate the initial mesh.
- **A5** : Select refinement and/or improvement criteria and regions where they will be applied.
- **A11** : Apply a memory efficient refinement and/or improvement algorithm using the specified quality criteria and regions.
- **A3** : Store the mesh in a specified output format.
- **A4** : Visualize the mesh.

S5 : Generate large meshes in a reasonable cpu time.

- **A1** : Apply an algorithm for reading the geometry in the corresponding format.
- **A2** : Apply an algorithm to generate the initial mesh.
- **A5** : Select refinement criteria and a regions where they will be applied.
- **A12** : Apply the fastest refinement algorithm using the specified criteria and regions.
- **A3** : Store the mesh in a specified output format.
- **A4** : Visualize the mesh.

S6 : Generate meshes for numerical method that require specific information (postprocess).

- **A14** : Read an already generated mesh.
- **A15** : Store the mesh in its internal representation.
- **A16** : Apply post-process to the mesh.
- **A3** : Store the mesh in a specified output format.
- **A4** : Visualize the mesh.

S7 : Evaluate meshes.

- **A14** : Read an already generated mesh.
- **A15** : Store the mesh in its internal representation.
- **A17** : Evaluate the quality of the mesh.
- **A4** : Visualize the mesh.

S8 : Adapt and improve the quality mesh of an already generated mesh.

- **A14** : Read an already generated mesh.
- **A15** : Store the mesh in its internal representation.
- **A5** : Select a quality criterion and a region where they will be applied.
- **A6** : Apply an improvement and/or optimization algorithm using the specified quality criterion and region.
- **A7** : If desired, evaluate the quality of the mesh elements.
- **A3** : Store the mesh in a specified output format.
- **A4** : Visualize the mesh.

S9 : Incorporate a new visualizer.

- **A18** : Choose a new visualizer.

- **A19** : Implement the common interface in order to integrate the new visualizer into the tool.

S10 : Incorporate a new algorithm to the existing ones.

- **A20** : Identify and implement the component that represents the algorithm we want to add.
- **A21** : Implement the common interface in order to integrate the new algorithm into the tool.

S11 : Incorporate a new kind of mesh processing.

- **A22** : Identify and implement the component that represents the process we want to add.
- **A23** : Implement the common interface in order to integrate the new process into the tool.

S12 : Incorporate a new criterion for refine, improve, optimize and/or derefine algorithms.

- **A24** : Define the criterion we want to add.
- **A25** : Implement the common interface for this criterion.

S13 : Incorporate a new approach to generate an initial mesh.

- **A26** : Design a new algorithm to generate an initial mesh.
- **A27** : Implement the common interface for any initial mesh algorithm.

3.5. Consistency

Table 2 establishes the relationship between goals and scenarios (*By-Scenario*, relationship [a] in the *Domain-Model* schema). We can see that all goals are achieved by at least one scenario, and that all scenarios achieve at least one goal, fulfilling condition [k]in *ConsistentDomain-Model*. Table 3 (relationship [i]) establishes that all actions are fulfilled by at least one feature, fulfilling conditions [n] and [o] in *ConsistentDomainModel*. By construction, the set of actions is the union of all actions required for fulfilling the specified scenarios. Table 4 (*By-Feature*, relationship [b]) shows that every feature in the first level of the feature model is required for fulfilling at least one scenario. According to invariant [j] of the schema *Consistent-DomainModel* we must assure that every feature in feature model will be present and described in the lexicon. This is achieved by simple inspection of the feature model and lexicon. Following a similar procedure, all other conditions in the *ConsistentDomainModel* schema can also be proved.

Due to the symmetry shown in the feature model by **2D Meshing Tool** and **3D Meshing Tool** features and because each action is related to the feature in **2D** and **3D** at the same time, we built Table 3 without putting the prefix **2D** or **3D** to each feature. The same occurs in Table 4 for Features and Scenarios.

Table 2. Relationships between Goals and Scenarios

Goals	Scenario
G1, G2, G3, G4, G8	S0
G1, G8	S1
G2, G3, G4, G5, G8	S2
G1, G3, G8	S3
G1, G4	S4
G1, G3, G5, G8	S5
G1, G3, G8	S6
G1, G2, G8	S7
G1, G8	S8
G6	S9
G3, G4, G5, G6	S10
G3, G4, G5, G7	S11
G6	S12
G6	S13

Table 3. Relationship between Actions and Features

Action	Features
A1	Input
A2	Generate initial mesh, Mesh
A3	Output, Mesh
A4	Visualize, Mesh
A5	Region, Criterion
A6	Algorithm, Criterion, Region, Mesh
A7	Evaluate, Mesh
A8	Algorithm, Criterion, Region, Mesh
A9	Algorithm, Criterion, Region, Mesh
A10	Algorithm, Criterion, Region, Mesh
A11	Algorithm, Criterion, Region, Mesh
A12	Algorithm, Criterion, Region, Mesh
A14	Input
A15	Mesh
A16	Postprocess, Mesh
A17	Evaluate, Mesh
A18	Visualize
A19	Visualize, Output, Mesh
A20	Algorithm
A21	Algorithm, Criterion, Region, Mesh
A22	Algorithm
A23	Algorithm, Criterion, Region, Mesh
A24	Algorithm, Criterion, Mesh
A25	Algorithm, Criterion, Mesh
A26	Input, Generate initial mesh, Mesh
A27	Input, Generate initial mesh, Mesh

Table 4. Relationship between Features and Scenarios

Feature	Scenarios
Generate initial mesh	S0, S1, S2, S3, S4, S5, S13
Algorithm	S1, S2, S3, S4, S5, S8, S10, S11, S12
Input	S0, S1, S2, S3, S4, S5, S6, S7, S8, S13
Output	S0, S1, S2, S3, S4, S5, S6, S8, S9
Criterion	S1, S2, S3, S4, S5, S8, S10, S11, S12
Region	S1, S2, S3, S4, S5, S8, S10, S11
Evaluate	S1, S7, S8
Visualize	S0, S1, S2, S3, S4, S5, S6, S7, S8, S9
Postprocess	S6
Mesh	S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13

It is necessary for a detailed consistency to advance in developing scenarios and actions that include features of deeper levels of detail. Moreover, we need to complete Tables 3 and 4 because features of the first level as **2D Move Boundary**, **3D Move Boundary** and **User Interface** are not included.

3.6. Completeness

As we have mentioned, completeness is necessary to check against the needs of the stakeholders. In that sense, the domain experts are called to verify this condition, reviewing the different artifacts of the domain model.

Other possibility is to check against existing meshing tools. That is beyond the scope of this paper.

4. Related Work

We here discuss different techniques proposed for domain analysis (DA) in general. We work with domain analysis because domain modeling is part of it, and therefore domain model is a output of the domain analysis. For details about other domain analysis methods, Succi et al. [33] provide a survey.

Domain analysis is the process of identifying, collecting, organizing, and representing the relevant information in a domain, based upon the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within a domain [27]. Although it is a general purpose process, it has been identified as one of the most appropriate forms of requirements engineering in the context of a SPL [8].

Coplien et al.[6] propose SCV (Scope, Commonalities and Variabilities), a method for conceptually addressing domain analysis within SPL. There are some notations and techniques proposed for realizing SCV such as FAST [35], FORM [16] and PuLSE [4]. These methods are useful for any application domain and they generally cover the whole domain engineering stage. All these methods propose well defined processes for building the domain model. Our approach goes a step further by formalizing the domain model definition and thus we are also able to precisely define iteration/termination conditions for our proposed process. Furthermore, we do not need to tailor the approach, because it is specific for the meshing tool domain.

Smith and Chen [29] have applied SCV to the meshing tool domain using FAST. Even though their approach is systematic, they do not take full advantage of the meshing tool domain characteristics because they apply a general DA method for scientific computing software [28, 31]. For example, we have noticed that the binding time for variabilities in meshing tools is fixed: which features are included is always decided at product design time, and which particular implementation is chosen for each included feature is decided at compilation time. In this way, our documentation is more compact and the process is simpler because a default binding time is used. This default binding time allows us to make decisions at a higher level of abstraction, and thus yielding simpler tools that would probably have a better performance.

Kim et al. [17] propose a DA method based on goals and scenarios. It involves four information levels: business, service, interaction and internal, each of them refining the previous one. This method is appropriate for characterizing a domain where the expert has little experience on software engineering. Meshing tool developers are usually knowledgeable in software engineering, so we were able to simplify the domain model. Our model includes the business goal for the SPL and a single level where the complete model is defined. Also Park et al. [24] propose to use features, scenarios and goals for capturing the characteristics of the domain, as we do. However, since their approach is general for any domain, they use a method that involves three successive specification levels. We found that for our specific meshing tool domain, a model with two levels is enough. Both the method of Kim et al. [17] and Park et al. [24] have no termination condition. This is necessary because both have four and three abstraction levels of requirements respectively and the stakeholders need to know when the domain model is correct and finished.

Niemelä and Immonen [21] introduce QRF (Quality Requirements of a software Family) method, which explicitly focuses on how quality requirements have to be defined, represented and transformed to architectural models. This method is appropriated structured and each step is clearly defined. Even though we think this method could be used for defining the meshing tool domain model, it needs tailoring because it was probed in a particular domain, different from meshing tool domain. Moreover, it requires the existence of already developed products (in our case this condition is optional). Finally, it is not clear the gap between the problem space and the solution space, and it could represent a risk for no experimented stakeholders. This characteristic could be because the focus of the method is in transformation of the requirements to architectures.

Douta et al. [9] present a new and interesting approach to commonality and variability analysis called CompAS, for the specific domain of computer assisted orthopaedic surgery. This approach cenerstre its efforts in the analysis of the evolution of the domain to effectively determine which features should be included as common or variable. The method bases its source of information for building the domain model in the publicly available literature (e.g. books, articles and standards) to overcome the lack of systems documentation in that domain. CompAS only suggest to the domain analyst to regularly consult domain expert for a correct, consistent and complete functional decomposition. This method is domain-specific as ours, and also stakeholders play a relevant and clearly defined role.

Smith and Chen [30] researched meshing tool requirements with a SPL perspective, but no procedure is provided for using the products of this method for actually building meshing tools. Also Bastarrica et al. [2] propose a product line architecture for the meshing tool domain, and they show how tools could be built [1] using it, but they do not focus on a systematic DA method.

5. Conclusions

We presented a domain modeling process specially suited for the meshing tool domain, showing how its characteristics could be specified using a model based on features, scenarios, goals and lexicon. We propose a rigorous process with activities, roles and clear termination conditions. It is also customized avoiding activities that general processes include but are not relevant here, such as determining the binding time of the identified variabilities.

Deciding when requirements are complete is generally a difficult issue. The termination conditions provided in our process give a systematic means for verifying if the elements included in the domain model allow us to build all the products within the SPL scope.

The Domain Model presented is the basis for building a Meshing Tool SPL. In that sense and considering this model, we have implemented a set of software components that implement the functionality identified by the features. We have also designed a candidate PLA based on the specified goals and scenarios. Currently we are in the process of building a framework that automates the product engineering making use of all reusable assets.

Acknowledgment

The work of Pedro O. Rossel was partially supported by grant No. UCH 0109 from MECESUP, Chile and by grant of the Departamento de Postgrado y Postítulo de la Vicerrectoría de Asuntos Académicos of Universidad de Chile. Pedro O. Rossel thanks Dr. Krzysztof Czarnecki for fruitful discussions.

References

[1] M. C. Bastarrica and N. Hitschfeld-Kahler. Designing a product family of meshing tools. *Advances in Engineering Software*, 37(1):1–10, Jan. 2006.

[2] M. C. Bastarrica, N. Hitschfeld-Kahler, and P. O. Rossel. Product Line Architecture for a Family of Meshing Tools. In *Proceedings of the 9th International Conference on Software Reuse (ICSR 2006)*, volume 4039 of *Lecture Notes in Computer Science*, pages 403–406. Springer, June 2006.

[3] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. PuLSE: A Methodology to Develop Software Product Lines. In *Proceedings of the 5th Symposium on Software Reusability (SSR'99)*, pages 122–131. ACM Press, May 1999.

[4] J. Bayer, D. Muthig, and T. Widen. Customizable Domain Analysis. In *Proceedings of the 1st International Symposium Generative and Component-Based Software Engineering (GCSE'99)*, volume 1799 of *Lecture Notes in Computer Science*, pages 178–194. Springer, Sept. 1999.

[5] F. Contreras. Adaptación de una Herramienta de Generación de Mallas Geométricas 3D a una Nueva Arquitectura. Computer Science Engineering Thesis. Departamento de Ciencias de la Computación, Universidad de Chile, Dec. 2007. In Spanish.

[6] J. Coplien, D. Hoffman, and D. M. Weiss. Commonality and Variability in Software Engineering. *IEEE Software*, 15(6):37–45, Nov./Dec. 1998.

[7] K. Czarnecki, M. Antkiewicz, C. H. P. Kim, S. Lau, and K. Pietroszek. Model-Driven Software Product Lines. In *Proceedings of the Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2005)*, pages 126–127. ACM Press, Oct. 2005.

[8] K. Czarnecki and U. W. Eisenecker. *Generative Programming. Methods, Tools, and Applications*. Addison Wesley, May 2000.

[9] G. Douta, H. Talib, O. Nierstrasz, and F. Langlotz. CompAS: A new approach to commonality and variability analysis with applications in computer assisted orthopaedic surgery. *Information and Software Technology*, 2008. doi:10.1016/j.infsof.2008.05.017.

[10] J. Greenfield and K. Short. Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 16–27, New York, NY, USA, 2003. ACM Press.

[11] N. Hitschfeld, C. Lillo, A. Cáceres, M. C. Bastarrica, and M. C. Rivara. Building a 3D Meshing Framework Using Good Software Engineering Practices. In *Proceedings of the IFIP Workshop on Advanced Software Engineering (IWASE 2006)*, volume 219, pages 162–170. Springer, Aug. 2006.

[12] N. Hitschfeld, L. Villablanca, J. Krause, and M. C. Rivara. Improving the quality of meshes for the simulation of semiconductor devices using Lepp-based algorithms. *International Journal for Numerical Methods in Engineering*, 58(2):333–347, Sept. 2003.

[13] N. Hitschfeld-Kahler. Generation of 3d mixed element meshes using a flexible refinement approach. *Engineering with Computers*, 21(2):101–114, Dec. 2005.

[14] H. Kaindl. A Design Process Based on a Model Combining Scenarios with Goals and Functions. *IEEE Transactions on Systems, Man, and Cybernetics–Part A: Systems and Humans*, 30(5):537–551, Sept. 2000.

[15] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA). Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Nov. 1990.

[16] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. FORM: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5(1):143–168, Jan. 1998.

[17] J. Kim, M. Kim, and S. Park. Goal and scenario based domain requirements analysis environment. *Journal of Systems and Software*, 79(7):926–938, July 2006.

[18] K. Lee, K. C. Kang, and J. Lee. Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In *Proceedings of the 7th International Conference on Software Reuse (ICSR-7)*, volume 2319 of *Lecture Notes in Computer Science*, pages 62–77. Springer, Apr. 2002.

[19] C. Lobos, M. Bucki, N. Hitschfeld-Kahler, and Y. Payan. Mixed-element Mesh for an Intra-operative Modeling of the Brain Tumor Extraction. In *Proceedings of the 16th International Meshing Roundtable*, pages 387–404. Springer, Oct. 2007.

[20] C. Melo. Desarrollo de una Herramienta que Genera Mallas de Superficie Compuestas de Cuadriláteros para Modelar el Crecimiento de Arboles. Computer Science Engineering Thesis. Departamento de Ciencias de la Computación, Universidad de Chile, Apr. 2008. In Spanish.

[21] E. Niemelä and A. Immonen. Capturing quality requirements of product family architecture. *Information and Software Technology*, 49(11-12):1107–1120, Nov. 2007.

[22] L. Northrop and P. Clements. A Framework for Software Product Line Practice, version 5.0. http://www.sei.cmu.edu-/productlines/framework.html, 2008. Accessed on November.

[23] S. J. Owen. http://www.andrew.cmu.edu/user/sowen-/mesh.html, 2008. Accessed on November.

[24] S. Park, M. Kim, and V. Sugumaran. A scenario, goal and feature-oriented domain analysis approach for developing software product lines. *Industrial Management & Data Systems*, 104(4):296–308, May 2004.

[25] R. Prieto-Díaz. Domain Analysis: An Introduction. *SIGSOFT Software Engineering Notes*, 15(2):47–54, Apr. 1990.

[26] SEI. Domain Engineering. http://www.sei.cmu.edu/domain-engineering/domain_eng.html, 2008. Accessed on November.

[27] SEI. Domain Modeling. http://www.sei.cmu.edu/domain-engineering/domain_model.html, 2008. Accessed on November.

[28] S. Smith. Systematic Development of Requirements Documentation for General Purpose Scientific Computing Software. In *Proceedings of the 14th IEEE International Conference on Requirements Engineering (RE 2006)*, pages 205–215. IEEE Computer Society, Sept. 2006.

[29] S. Smith and C.-H. Chen. Commonality Analysis for Mesh Generating Systems. Technical Report CAS-04-10-SS, Department of Computing and Software, McMaster University, Canada, Oct. 2004.

[30] S. Smith and C.-H. Chen. Commonality and Requirements Analysis for Mesh Generating Software. In *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering (SEKE'2004)*, pages 384–387, June 2004.

[31] S. Smith, J. McCutchan, and F. Cao. Program Families in Scientific Computing. In *Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling (DSM'07)*, Oct. 2007.

[32] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, 2nd edition, 1992.

[33] G. Succi, A. Valerio, T. Vernazza, M. Fenaroli, and P. Predonzani. Framework extraction with domain analysis. *ACM Computing Surveys*, 32(1es):12, Mar. 2000.

[34] M. Svahnberg, J. van Gurp, and J. Bosch. A taxonomy of variability realization techniques. *Software: Practice and Experience*, 35(8):705–754, July 2005.

[35] D. M. Weiss. Commonality Analysis: A Systematic Process for Defining Families. In *Proceedings of the 2nd International ESPRIT ARES Workshop*, volume 1429 of *Lecture Notes in Computer Science*, pages 214–222. Springer, Feb. 1998.