

---

# Supporting Reuse in Meshing Tool Development using Domain Analysis

Pedro O. Rossel<sup>1,2</sup>, María Cecilia Bastarrica<sup>1</sup>, and Nancy Hitschfeld-Kahler<sup>1</sup>

<sup>1</sup> Dept. of Computer Science, Universidad de Chile, Chile

<sup>2</sup> Depto. de Computación e Informática, Universidad Católica del Maule, Chile  
{prossel,cecilia,nancy}@dcc.uchile.cl

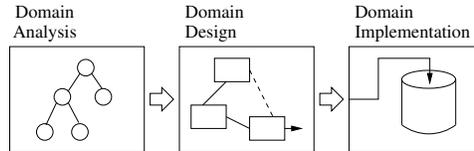
**Summary.** Meshing tools are highly complex software, and they have usually been developed one at a time and with ad-hoc methodologies. Developing new tools is thus expensive even though there may be similar tools already built. Counting on a systematic method for reusing complex meshing tool components would enhance productivity and enable trying new algorithms with a much lower cost. Software Product Lines (SPL) is a well established strategy for massive reuse, and Domain Analysis (DA) is the activity whose purpose is to identify potentially reusable assets within a SPL. There exist some techniques for performing DA but all of them are general and not necessarily appropriate for the particular case of a Meshing Tool SPL. In this paper we propose a method for DA specially suited for the case of a Meshing Tool SPL. We define the domain model, the process for building this model, and the way the collected information could be used for building new products of the SPL. We applied the method showing how it could support building two different meshing tools.

## 1 Introduction

According to the SEI [19], 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. The ultimate purpose of a SPL is to provide a reuse infrastructure that allows to achieve a high Return On Investment (ROI). In a SPL we can identify two main technical stages [24]: 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 [31].

Domain analysis (DA) is the process through which commonalities and variabilities are identified, captured and organized in a model with the purpose of making it available for reuse in future developments [22]. It has been identified as one of the most important factors for the success of software reuse [1]. In the context of SPL, domain analysis is the first step within the domain engineering stage, as shown in Figure 1 [24].



**Fig. 1.** Domain engineering stage

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 all scientific computing software. Provided that meshing tools are used in a variety of different application domains, they may require slightly different functionalities. As these tools have usually been developed with ad hoc methodologies, and without taking reuse as a goal, 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. Meshing tools have a good opportunity for reuse, but a reuse framework is required if the gains in productivity and quality are to be achieved.

The main steps of any mesh generation process are: generation of an initial mesh that fits the domain geometry, generation of an intermediate mesh that satisfies the density requirements specified by the user, generation of an improved mesh that satisfies the quality criteria, and generation of the final mesh in an appropriate output format. These shared steps have been identified as commonalities among all members of a meshing tool family. Variabilities may be approached in two different dimensions: by including or not certain steps, or by providing alternative implementations or algorithms for realizing the same chosen functionality. Even though some authors have already approached building meshing tools with SPL concepts in mind [2, 3, 27, 28], to the best of our knowledge, none of them has focus on DA with a systematic methodology specially designed for this particular domain.

In this paper we propose a method for domain analysis specially suited for the meshing tool domain. It uses features, goals and scenarios as a means for capturing the domain characteristics and we organize them in a formal model. We provide a method 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 method, validating it using two existing tools: Cubit and Triangle.

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

## 2 Related Work

We here discuss different techniques proposed for domain analysis in general, and for its application to meshing tools in particular as well as other approaches that have been followed to build meshing tools.

### 2.1 Domain Analysis Techniques

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 [24]. 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 [10].

Coplien et al. propose SVC [8], a method for conceptually addressing domain analysis within SPLs. There, the identification of the Scope, Variabilities and Commonalities of the product family are the main issues. There are some notations and techniques proposed for realizing SVC such as FAST [31], FORM [16] and PuLSE [4]. These methods are useful for any application domain and they generally cover the whole domain engineering stage. Moreover, 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 method.

Smith and Chen have applied SVC to the meshing tool domain [28] 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 [26]. 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

method 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 have a better performance.

In [17] a DA method based on goals and scenarios is proposed. It involved 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.

Park et al. [21] propose to use features, scenarios and goals for capturing the characteristics of the domain, as we do. However, and since their approach is general for any domain, they use a method that involves four successive specification levels. We found that for our specific meshing tool domain, a model with two levels is enough.

## 2.2 Developing Meshing Tools

Only in the last decade the development of meshing software has been approached from the software engineering point of view mainly applying object-oriented design and programming. Some of the work include the development of a software environment for the numerical solution of partial differential equations (Diffpack) [7], the design of generic extensible geometry interfaces between CAD modelers and mesh generators [20, 30], the design of object-oriented data structures and procedural classes for mesh generation [18], the computational geometry algorithm library CGAL [13], the definition of an optimal OO mesh representation that allows the programmer to build efficient algorithms (AOMD) [23], and algorithms that can be used independently of the concrete mesh representations [5], as well as a tool to support these algorithms (Grid Algorithms Library) [6]. More recently, formal methods were also used for improving reliability of mesh generation software [12].

In [32] a document driven approach for generating a family of parallel meshing tools is provided. The information used is similar to that included in our model, but the process presented is mainly a waterfall. Our process is iterative and the model is built incrementally so that feedback can be systematically incorporated.

Smith and Chen [28] 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. [3] propose a product line architecture for the meshing tool domain, and they show how tools could be built [2] using it, but they do not focus on a systematic DA method.

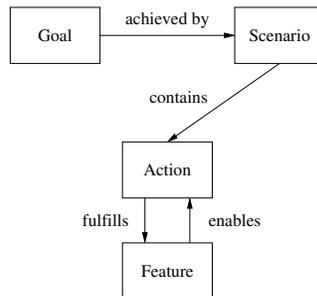
### 3 Domain Analysis

The domain model will be defined in terms of features, goals and scenarios; Section 3.1 defines this model together with the relationships among its constituent elements. In Section 3.2 we present the DA method, and Section 3.3 describes how the DA model is used within the Domain Engineering stage for generating reusable assets.

#### 3.1 Features, Goals and Scenarios

Our proposed method is based on features, goals and scenarios whose definition is taken and adapted from [21]. 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].

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 in a product line.



**Fig. 2.** Features, goals and scenarios

Figure 2, adapted from [21], represents the relationships that exist among features, goals, scenarios and actions. Scenarios capture real requirements as they describe real situations or concrete behaviors in terms of actions. Goals can be achieved through the execution of scenarios. Product features and goals are related indirectly, essentially through some behavior of scenarios, i.e. actions contained in the scenarios. We consider *GOAL*, *FEATURE*, and *ACTION* as primitive types. We define *SCENARIO* as a sequence of actions.

$$[GOAL, FEATURE, ACTION]$$

$$SCENARIO == \text{seq } ACTION$$

The following Z schema [29] *DomainModel* defines the elements that form part of our domain model. The *DomainModel* also includes the relationships between goals and scenarios (*By\_Scenario* [a]), and between scenarios and features (*By\_Feature* [b]); these relationships are inspired in [14].

<i>DomainModel</i>	
<i>Goals</i> : $\mathbb{P}$ <i>GOAL</i>	
<i>Scenarios</i> : $\mathbb{P}$ <i>SCENARIO</i>	
<i>Features</i> : $\mathbb{P}$ <i>FEATURE</i>	
<i>Actions</i> : $\mathbb{P}$ <i>ACTION</i>	
<i>By_Scenario</i> : <i>GOAL</i> $\leftrightarrow$ <i>SCENARIO</i>	[a]
<i>By_Feature</i> : <i>SCENARIO</i> $\leftrightarrow$ <i>FEATURE</i>	[b]
<i>Actions</i> = { <i>a</i> : <i>ACTION</i>   $\exists s \in \text{Scenarios} \wedge a \in \text{ran } s$ }	[c]
$\text{dom } \textit{By\_Scenario} \subseteq \textit{Goals}$	[d]
$\text{ran } \textit{By\_Scenario} \subseteq \textit{Scenarios}$	[e]
$\text{dom } \textit{By\_Feature} \subseteq \textit{Scenarios}$	[f]
$\text{ran } \textit{By\_Feature} \subseteq \textit{Features}$	[g]

The only actions that are identified are those that are derived from an already identified scenarios [c]. 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 [d,e,f,g].

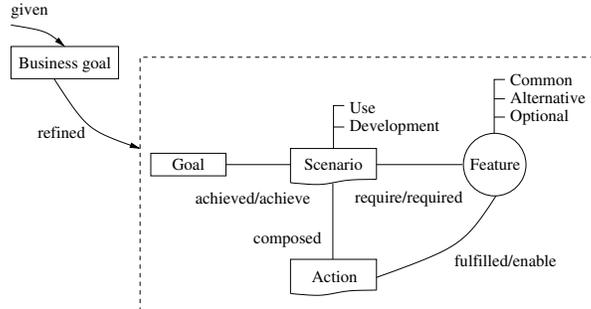
Although we may have a transient inconsistent domain model, at the end it needs to be consistent. The following schema refines the prior one by adding certain constraints. It also includes the definition of another relationship (*Attached* [h]) between actions and features that are necessary for fulfilling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

<i>ConsistentDomainModel</i>	
<i>DomainModel</i>	
<i>Attached</i> : <i>ACTION</i> $\leftrightarrow$ <i>FEATURE</i>	[h]
$\text{dom } \textit{By\_Scenario} = \textit{Goals}$	[i]
$\text{dom } \textit{By\_Feature} = \text{ran } \textit{By\_Scenario} = \textit{Scenarios}$	[j]
$\text{ran } \textit{By\_Feature} = \textit{Features}$	[k]
$\text{dom } \textit{Attached} = \textit{Actions}$	[l]
$\text{ran } \textit{Attached} = \textit{Features}$	[m]

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

### 3.2 Domain Analysis Method

Figure 3 summarizes the elements in the domain model and their relationships.

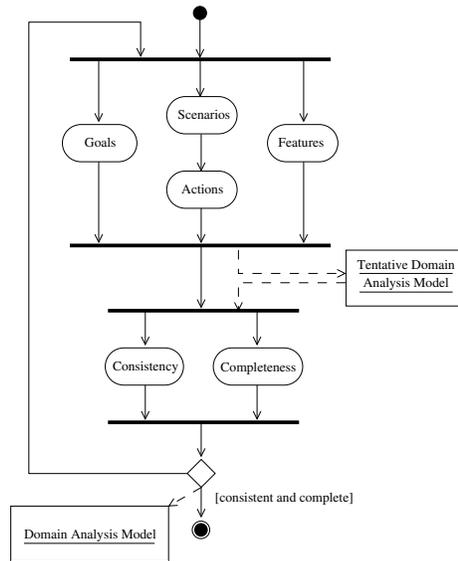


**Fig. 3.** Domain Analysis Products

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, optional or alternative. We use a feature model for specifying features following the notation proposed by Czarnecki and Helsen [11], and structured English for goals and scenarios.

Figure 4 shows an activity diagram for the DA method we propose. The domain expert and the domain analyst should interact in order to identify and specify goals, features, scenarios and actions, as well as their relationships. Once these activities are done, the domain expert checks for completeness by analyzing if the model elements captured are enough for building all products expected within the SPL scope. Meanwhile the domain analyst checks for consistency by verifying that the domain model satisfies all the consistency conditions. 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 step within the domain engineering stage.

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, so the model could be simple. Also, there are several pre-implemented components, already tested and with appropriately documented interfaces so it would not be extremely difficult to identify them as features; 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 DA method, but a much simpler one.



**Fig. 4.** Domain Analysis Method

### 3.3 Domain Analysis and Engineering

The second step in the Domain Engineering is the Domain Design. In this step, the product line architecture (PLA) is defined so that it can foster all identified features, and allows all identified scenarios to be carried out. As the goals refer to quality attributes, they influence the architectural style used for the PLA design. For the meshing tool domain it is relevant to be able to reuse several already developed software components so we also consider them when designing the candidate PLA. The final step, the Domain Implementation, mainly involves component implementation. According to the designed PLA, the existing components, and the identified features, we must implement the missing components according to the interfaces stated by the PLA.

During Application Engineering, we choose the features that will be included in a particular product, and then the corresponding components will be arranged following the structure of the PLA, using it as a roadmap.

## 4 Domain Analysis for Meshing Tools

In this section, we apply the DA method to build the meshing tool domain model. We first obtained the business goal. Then we defined particular goals, and then some scenarios and features. Some goals, scenarios and features were related, but others were not. We proceeded with a second iteration mainly because the feature model was found to be incomplete. In the second iteration we advanced in the feature model (Figure 5).

The relationships among goals, features and scenarios were stated in tables so that consistency checks would result easier. Only when the domain expert and the domain analyst intuitively thought that the model could be ready they proceeded to check for termination conditions. Thus finally our domain analyst checked for domain model consistency using the *Consistent-DomainModel* schema, and the domain expert checked for completeness by determining if two candidates tools of the SPL could be built with the documented elements; in this case we use two already existing tools: Cubit and Triangle.

#### 4.1 Goals

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

##### Business Goal

Developing new robust meshing tools 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 a specific domain according to certain given criteria that depend on the particular application.
- 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:** Scalability in the geometry complexity of the problems modeled.
- G7:** Make easy the interchange of different implementations for components of the same type.
- G8:** Make easy to add a new kind of process to be applied to the mesh.
- G9:** Generation of meshes that fulfill the requirements of different numerical methods.

#### 4.2 Features

Figure 5 shows our feature model. The features **Geometry**, **Output format**, **Visualize** and **Mesh** are common to any meshing tool. **Generate initial mesh**, **Algorithm**, **Criterion**, **Region**, **Evaluate** and **Postprocess** are optionals, i.e., they can be present or not in a particular meshing tool. Moreover, the algorithms for **Move Boundary**, **Refine**, **Improve**, **Optimize** and **Derefine** can be all or any subset in a particular tool. Finally, a meshing tool must work with a **Mesh** in **2D** or **3D**, but no with both.

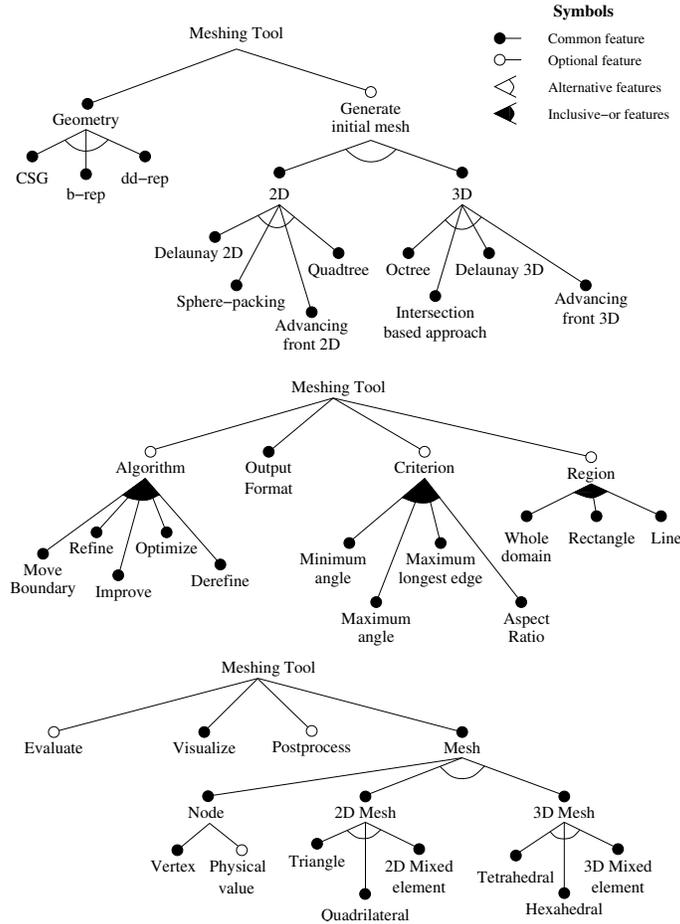


Fig. 5. A feature model for Meshing Tool domain

### 4.3 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 of different scenarios have the same identification because they are identical.

- S0** : Generate an initial mesh for a specific domain.
- A1** : Apply an algorithm for reading the geometry in the corresponding format.
- A2** : Apply an algorithm to generate the initial mesh.
- A3** : Store the mesh in a specified output format.
- A4** : Visualize the mesh.

- S1** : Generate a quality mesh from a domain geometry.
  - A1 : Apply an algorithm for reading the geometry in the corresponding format.
  - A2 : Apply an algorithm to generate the initial mesh.
  - 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.
- S2** : Generate a quality mesh 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 a quality criterion and a region where they will be applied.
  - A8 : Apply a refinement, improvement and/or optimization algorithm using the specified quality criterion and region.
  - A9 : If necessary, apply a derefinement algorithm using the specified quality criterion and region.
  - A3 : Store the mesh in a specified output format.
  - A4 : Visualize the mesh.
- S3** : Generate a mesh 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 a quality criterion and a region where they will be applied.
  - A10 : Apply the fastest improvement and/or optimization algorithm using the specified quality criterion and region.
  - A3 : Store the mesh in a specified output format.
  - A4 : Visualize the mesh.
- S4** : Generate a mesh with a 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 a quality criterion and a region where they will be applied.
  - A11 : Apply an memory efficient improvement and/or optimization algorithm using the specified quality criterion and region.
  - 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 a quality criterion and a region where they will be applied.

- A12 : Apply the fastest refinement algorithm using the specified quality criterion and region.
- A3 : Store the mesh in a specified output format.
- A4 : Visualize the mesh.
- S6** : Generate an initial mesh for any complex geometry.
  - A1 : Apply an algorithm for reading the geometry in the corresponding format.
  - A13 : Apply an algorithm to generate the initial mesh that can manage any complex geometry.
  - A3 : Store the mesh in a specified output format.
  - A4 : Visualize the mesh.
- S7** : Generate meshes for numerical method that require specific information (postprocess).
  - A14 : Read an already generated mesh.
  - A15 : Store the mesh
  - A16 : Apply post-process to the mesh.
  - A3 : Store the mesh in a specified output format.
  - A4 : Visualize the mesh.
- S8** : Evaluate meshes.
  - A14 : Read an already generated mesh.
  - A15 : Store the mesh
  - A17 : Evaluate the quality of the mesh.
  - A4 : Visualize the mesh.
- S9** : Generate a quality mesh from any input mesh.
  - A14 : Read an already generated mesh.
  - A15 : Store the mesh
  - 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.
- S10** : Incorporate a new visualizer.
  - A18 : Choose a new visualizer.
  - A19 : Implement the common interface in order to integrate the new visualizer into the tool.
- S11** : 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.
- S12** : 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.

**S13** : 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.

**S14** : 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.

Scenarios S0 to S9 are use scenarios, and S10 to S14 are development scenarios. This division is not strict, provided that one use scenario could be used for understanding how to build a product of the SPL.

#### 4.4 Consistency

Table 1 establishes the relationship between goals and scenarios (*By-Scenario*). We can see that all goals are achieved by at least one scenario, and that all scenarios achieve at least one goal. Table 2 establishes that all actions are fulfilled by at least one feature. By construction, the set of actions is the union of all actions required for fulfilling the specified scenarios. Table 3 (*By-Feature*) shows that every feature in the first level of the feature model is required for fulfilling at least one scenario. So all conditions in *ConsistentDomainModel* are satisfied.

**Table 1.** Relationships between Goals and Scenarios

Goals	Scenario
G1, G2, G3, G4, G9	S0
G1, G9	S1
G2, G3, G4, G5, G9	S2
G1, G3, G9	S3
G1, G4	S4
G1, G3, G5, G9	S5
G6	S6
G1, G3, G9	S7
G1, G2, G9	S8
G1, G9	S9
G7	S10
G3, G4, G5, G7	S11
G3, G4, G5, G8	S12
G7	S13
G7	S14

It is necessary for a detailed consistency to advance in developing scenarios and actions that include features of deeper detail levels.

#### 4.5 Completeness

In order to check for completeness, our domain expert checked if two well know meshing tools, Triangle [25] and Cubit [9], could be built with the elements

**Table 2.** Relationship between Actions and Features

Action	Features
A1	Geometry
A2	Generate initial mesh, Mesh
A3	Output Format, Mesh
A4	Visualize, Mesh
A5	Region, Criterion
A6	Algorithms, Criterion, Region, Mesh
A7	Evaluate, Mesh
A8	Algorithms, Criterion, Region, Mesh
A9	Algorithms, Criterion, Region, Mesh
A10	Algorithms, Criterion, Region, Mesh
A11	Algorithms, Criterion, Region, Mesh
A12	Algorithms, Criterion, Region, Mesh
A13	Geometry, Generate initial mesh, Mesh
A14	Geometry
A15	Mesh
A16	Postprocess, Mesh
A17	Evaluate, Mesh
A18	Visualize
A19	Visualize, Output Format, Mesh
A20	Algorithms
A21	Algorithms, Criterion, Region, Mesh
A22	Algorithms
A23	Algorithms, Criterion, Region, Mesh
A24	Algorithms, Criterion, Mesh
A25	Algorithms, Criterion, Mesh
A26	Geometry, Generate initial mesh, Mesh
A27	Geometry, Generate initial mesh, Mesh

**Table 3.** Relationship between Features and Scenarios

Feature	Scenarios
Geometry	S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S14
Generate initial mesh	S0, S1, S2, S3, S4, S5, S6, S14
Algorithms	S1, S2, S3, S4, S5, S9, S11, S12, S13
Output Format	S0, S1, S2, S3, S4, S5, S6, S7, S9, S10
Criterion	S1, S2, S3, S4, S5, S9, S11, S12, S13
Region	S1, S2, S3, S4, S5, S9, S11, S12
Evaluate	S1, S8, S9
Visualize	S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10
Postprocess	S7
Mesh	S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14

specified in the domain model. Development scenarios are not present in this analysis because the identification of the elements of the domain was done by using the tools, and not by developing them.

### Triangle

Triangle is a well known open source 2D mesh generator that allows the user to generate quality 2D triangulations. As input it can read either a **geometry** defined by a boundary representation (**b-rep** feature) or an already generated mesh (**dd-rep** feature). In order to **generate an initial mesh**, it provides two variations of the **Delaunay 2D** feature: the **constrained Delaunay\*** algorithm and the **conforming Delaunay\*** algorithm. The **mesh** contains

the **Triangle** and **Vextex** features. As **criterion** feature it provides two alternatives: **Maximum area\*** and **Minimum angle**. As **Refine** feature it provides the **Delaunay refinement\*** algorithm in order to generate refined meshes. The same algorithm can also be used as an implementation of the **Improve** feature. The **Region** feature is always the **Whole domain**. For the **Postprocess** feature, there are two algorithms available: one for **generating and storing the Voronoi Diagram\*** of the generated triangulation and another for **checking the mesh consistency\***. As **output format** it can be the **Triangle\*** format (.node and .ele) or the **.off\*** format. The first one requires that the **visualize** feature be **ShowMe\*** and the second one requires **Geomview\***.

The related scenarios are: S0, S1, S3, S4, S5, S6, S7, S9.

The Goals are: G1, G2, G3, G4, G5, G6, G9

## Cubit

The Cubit tool suite provides an environment for geometry construction and mesh generation. The main tools are: (a) the Cubit geometry and mesh generation toolkit for both the geometry modeling and for the generation of quadrilateral and hexahedral meshes, (b) the Common Geometry Module for providing the functionality of solid modeling engines allowing the creation of the domain geometry, (c) the tool VEREDICT for mesh verification, (d) the graphical user interface CLARO for the mesh visualization. For the generation of **3D hexahedral meshes**, the **geometry** feature is implemented as **b-rep** and is built by the Common Geometry Module. The algorithms available for **generating an initial mesh** feature are **3D mapping\***, **hex sweeping\*** and **multi sweeping\***. As **optimizing** features, we find the **Laplacian\***, **equipotential\***, **centroid-area pull\*** and **Jacobian optimization\*** algorithms, among others. As the **criterion** feature, it provides the **Jacobian\***, **aspect ratio**, and **minimum** and **maximum angle** among others. The **mesh** feature is represented by and **hexahedral** mesh with **vertex** and **physical value** features. A **postprocess** feature is the conversion of the mesh from **hexahedra to tetrahedra\***. As the input/output **format**, it provides **ACIS sat\***, **sab\***, **STEP\*** and **IGES\***, among others. The **mesh evaluation** feature is implemented in the tool **VEREDICT\*** and the **visualize** feature with the graphical user interfaces **CLARO\***.

The related scenarios are: S0, S1, S3, S5, S7, S8.

The goals are: G1, G3, G5, G6, G9.

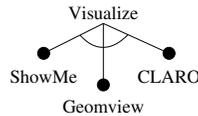
## Completeness evaluation

All features are highlighted and those that are not present in the feature model (Figure 5) also have an \* attached. Since there are several of these features, we conclude that our domain model is not complete. Then it is necessary, for

completeness, to add the missing features to the feature model and probably also adjust the scenarios and goals.

This incompleteness may lead us to think that our domain model is not good. The description of the tools used for checking completeness is quite detailed. But even though not all required features are found in the feature model, we are able to easily identify them and find the right place where the model should be extended. This quality is due to the iterativity of the proposed DA method and the structured notation used.

As a consequence of the completeness evaluation, we must for example include in the feature model of Figure 5 three new sub-features of feature **Visualize: ShowMe, Geomview** and **CLARO**, as we show in Figure 6.



**Fig. 6.** Sub-features for Visualize

With these new features, and according to our method, we need to iterate again. It is clear that new features may probably need new scenarios and goals because consistency would probably be affected with the inclusion of new elements in the domain model. In the particular case of the Visualize feature refined in Figure 6, the new features CLARO, Geomview and ShowMe should be included in *Features*, and all scenarios related to Visualize should be adapted to the existence of these features. Action A4 is related to feature Visualize, as can be seen in Table 2. Then action A4 will be rewritten as follows and replaced in all scenarios:

A4 : Visualize the mesh, choosing one of the following options: CLARO, Geomview and ShowMe.

## 5 Conclusions

We presented a DA method specially suited for the meshing tool domain, showing how its characteristics could be specified using a model based on features, scenarios and goals. Our method proposes a process with clear activities, roles and a clear termination condition. 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. We also use a simplified domain model because in the meshing tool domain complexity is encapsulated within components and domain experts generally are familiar with software engineering practices.

Deciding when requirements are complete is generally a difficult issue. The termination conditions provided in our method give a systematic means for

verifying if the elements included in the domain model allow us to build all the products included in the SPL scope.

We have implemented a set of software components that implements the functionality identified by the features, but we still need to complete it. We have also designed a candidate PLA. Once we finish this ongoing work we will be able to let real meshing tool experts build their own tools.

## References

1. G. Arango. A Brief Introduction to Domain Analysis. In *1994 ACM Symposium on Applied Computing (SAC '94)*, pages 42–46. ACM Press, 1994.
2. M. C. Bastarrica and N. Hitschfeld-Kahler. Designing a product family of meshing tools. *Advances in Engineering Software*, 37(1):1–10, 2006.
3. M. C. Bastarrica, N. Hitschfeld-Kahler, and P. O. Rossel. Product Line Architecture for a Family of Meshing Tools. In *9th International Conference on Software Reuse (ICSR 2006)*, volume 4039 of *Lecture Notes in Computer Science*, pages 403–406. Springer, 2006.
4. J. Bayer, D. Muthig, and T. Widen. Customizable Domain Analysis. In *Generative and Component-Based Software Engineering, First International Symposium (GCSE'99)*, volume 1799 of *Lecture Notes in Computer Science*, pages 178–194. Springer, 1999.
5. G. Berti. *Generic Software Components for Scientific Computing*. PhD Dissertation, TU Cottbus, 2000.
6. G. Berti. A Generic Toolbox for the Grid Craftsman. In *17th GaMM-Seminar Leipzig on "Construction of Grid Generation Algorithms"*, pages 1–28, 2001.
7. A. M. Bruaset and H. P. Langtangen. A Comprehensive Set of Tools for Solving Partial Differential Equations; Diffpack, 1996. <http://citeseer.ist.psu.edu/bruaset96comprehensive.html>.
8. J. Coplien, D. Hoffman, and D. M. Weiss. Commonality and Variability in Software Engineering. *IEEE Software*, 15(6):37–45, 1998.
9. CUBIT Tool Suite, 2007. <http://cubit.sandia.gov/>.
10. K. Czarnecki and U. W. Eisenecker. *Generative Programming. Methods, Tools, and Applications*. Addison Wesley, 2000.
11. K. Czarnecki and S. Helsen. Classification of Model Transformation Approaches. In *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.
12. A. H. ElSheikh, S. Smith, and S. E. Chidiac. Semi-formal design of reliable mesh generation systems. *Advances in Engineering Software*, 35(12):827–841, 2004.
13. A. Fabri. CGAL - The Computational Geometry Algorithm Library. In *10th Annual International Meshing Roundtable*, pages 137–142, 2001.
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, 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, 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*, 5:143–168, 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, 2006.
18. A. V. Mobley, J. R. Tristano, and C. M. Hawkings. An Object-Oriented Design for Mesh Generation and Operation Algorithms. In *10th Annual International Meshing Roundtable*, pages 179–183, 2001.
19. L. Northrop and P. Clements. A Framework for Software Product Line Practice. Version 5.0, 2007. <http://www.sei.cmu.edu/productlines/framework.html>.
20. M. Panthaki, R. Sahu, and W. Gerstle. An Object-Oriented Virtual Geometry Interface. In *6th Annual International Meshing Roundtable*, pages 67–81, 1997.
21. 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, 2004.
22. R. Prieto-Díaz. Domain Analysis: An Introduction. *SIGSOFT Software Engineering Notes*, 15(2):47–54, 1990.
23. J.-F. Remacle and M. S. Shephard. An algorithm oriented mesh database. *International Journal for Numerical Methods in Engineering*, 58(2):349–374, 2003.
24. SEI. Domain Engineering, 2007. [http://www.sei.cmu.edu/domain-engineering/domain\\_eng.html](http://www.sei.cmu.edu/domain-engineering/domain_eng.html).
25. J. R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *First ACM Workshop on Applied Computational Geometry*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer, 1996.
26. S. Smith. Systematic Development of Requirements Documentation for General Purpose Scientific Computing Software. In *14th IEEE International Conference on Requirements Engineering (RE 2006)*, pages 205–215. IEEE Computer Society, 2006.
27. 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, Ontario, Canada, 2004.
28. S. Smith and C.-H. Chen. Commonality and Requirements Analysis for Mesh Generating Software. In *Sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2004)*, pages 384–387, 2004.
29. J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, second edition, 1992.
30. T. J. Tautges. The common geometry module (CGM): A generic, extensible, geometry interface. In *9th Annual International Meshing Roundtable*, pages 337–347, 2000.
31. D. M. Weiss. Commonality Analysis: A Systematic Process for Defining Families. In *Development and Evolution of Software Architectures for Product Families, Second International ESPRIT ARES Workshop*, volume 1429 of *Lecture Notes in Computer Science*, pages 214–222. Springer, 1998.
32. W. Yu. A Document Driven Methodology for Improving the Quality of a Parallel Mesh Generation Toolbox. Master’s thesis, McMaster University, 2007.