Module Guide for a Parallel Mesh Generation Toolbox

Wen Yu

September 2008

Computing and Software

McMaster University

Contents

1 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team. The basic principle of the decomposition used here is the information hiding principle (Parnas et al., 1984). According to Parnas et al. (1984),

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) for the PMGT was developed. The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure of the PMGT and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it is can be used to check for consistency, feasibility and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as described in the following. Section 2 lists the anticipated and unlikely changes of the software requirements. Section 3 summarizes the module decomposition that was constructed according to the likely changes. Section 4 specifies the connections between the software requirements and the modules. Section 5 gives a detailed description of the modules. Section 6 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 7 describes the use relation between modules.

2 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 2.1, and unlikely changes are listed in Section 2.2.

2.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The data structure and algorithms for implementing the virtual memory of the system.

AC2: The data structure and algorithms for implementing the interface between the file and the system.

AC3: The data structure and algorithms for implementing the interface between the keyboard and the system.

AC4: The data structure and algorithms for screen display.

AC5: The format and structure of the initial input mesh.

AC6: The format and structure of the output mesh.

AC7: The mechanisms for validating the input and output meshes.

AC8: The data structure of a vertex.

AC9: The data structure of an edge.

AC10: The data structure of a cell.

AC11: The data structure of a mesh.

AC12: The algorithms for refining a mesh.

AC13: The algorithms for coarsening a mesh.

AC14: The shape of a cell, which is initially assumed to be a triangular.

2.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: There will always be a source of input data external to the PMGT software.

UC3: Output data are displayed to the output device.

UC4: The goal of the system is refining or coarsening a mesh.

UC5: The type of the mesh is unstructured.

UC6: The representation of an edge is a set of vertices.

UC7: The representation of a cell is a set of vertices.

UC8: A Cartesian coordinate system is used.

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Virtual Memory Module

M2: File Read/Write Module

M3: Keyboard Input Module

M4: Screen Display Module

M5: Input Format Module

M6: Output Format Module

M7: Service Module

M8: Vertex Module

M9: Edge Module

M10: Cell Module

M11: Mesh Module

M12: Refining Module

M13: Coarsening Module

Note that M1, M2, M3 and M4 are commonly used modules and are already implemented by the operating system. They will not need to be implemented again for PMGT.

4 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2. However, some connections are not obvious. The explanation below has the purpose of making these connections clear. The software requirements are documented in the SRS. They are also listed starting on page 12 for convenience.

The functionalities of refining a mesh (F1), and coarsening a mesh (F2) are achieved directly by M12 and M13, respectively. The functional requirement *MeshType* (F4) is related to the representation of mesh, which is contained in M9, M10, and M11. The algorithms for refining (M12) and coarsening (M13) also depend on the *MeshType* requirement. Another connection worth mentioning relates to the *DomainDimension* requirement (F6). All geometric information for the mesh, including dimension information, is stored in M8. Algorithms in M12 and M13 also relate to the dimension of the domain.

Level 1	**Level 2**	**Level 3**	**Level 4**
Hardware-Hiding Module	Extended Computer Module	Virtual Memory Module	
		File Read/Write Module	
	Device Interface Module	Keyboard Input Module	
		Screen Display Module	
Behavior-Hiding Module	Input Format Module		
	Output Format Module		
	Service Module		
Software Decision Module	Mesh Data Module	Entity Module	Vertex Module
			Edge Module
			Cell Module
		Mesh Module	
	Algorithm Module	Refining Module	
		Coarsening Module	

Table 1: Module Hierarchy

Some nonfunctional requirements, such as *Performance* (N1) and *Maintainability* (N7), are related to the overall quality of the system. These qualities depend on the implementation of all of the modules. The *Precision* requirement depends on modules related to calculation, which are the module M8, M9, M10, M11, M12 and M13.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *PMGT* means the module will be implemented by the PMGT software. Only leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented. Whether or not this module is implemented depends on the programming language selected. This decomposition is inspired by Chen (2003). The decomposition of the mesh data module is partly based on ElSheikh et al. (2004). One difference between the current design and ElSheikh et al. (2004) is that ElSheikh et al. (2004) has an explicit module for incidence and adjacency information. However, it is believed that where and how to store this information is an implementation decision that should be abstracted away at the design stage.

5.1 Hardware-Hiding Module

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: –

5.1.1 Extended Computer Module

Secrets: The number of processors, the instruction set of the computer, and the computer's capacity for performing concurrent operations.

Services: Provides an instruction set including the operations on application-independent data types, sequence control operations, and general I/O operations.

Implemented By: –

5.1.1.1 Virtual Memory Module (M1)

Secrets: The hardware addressing methods for data and instructions in real memory.

Services: Presents a uniformly addressable virtual memory.

Implemented By: OS

5.1.1.2 File Read Write Module (M2)

Secrets: The data structure and algorithms for implementing the interface between the file and the system.

Services: Provides an interface between the storage of the system and the IO devices.

Implemented By: OS

5.1.2 Device Interface Module

Secrets: Characteristics of the present devices not likely to be shared by replacement devices.

Services: Provides virtual devices to be used by the rest of software.

Implemented By: –

5.1.2.1 Keyboard Input Module (M3)

Secrets: The data structure and algorithms for implementing the interface between the keyboard and the system.

Services: Retrieves the user inputs from the keyboard and communicates the information with other parts of the system.

Implemented By: OS

5.1.2.2 Screen Display Module (M4)

Secrets: The data structure and algorithms to display graphics and text on the screen.

Services: Provides an interface between the system and the screen so the system can display information on the screen through the use of programs in the module.

Implemented By: OS

5.2 Behavior-Hiding Module

Secrets: The contents of the required behaviors.

Services: Includes programs that provide externally visible behavior of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

5.2.1 Input Format Module (M5)

Secrets: The format and structure of the initial input mesh.

Services: Converts the input mesh to the data structured used in PMGT.

Implemented By: PMGT

5.2.2 Output Format Module (M6)

Secrets: The format and structure of the output mesh.

Services: Converts the output mesh to an output file.

Implemented By: PMGT

5.2.3 Service Module (M7)

Secrets: The algorithm for validating meshes.

Services: Checks if the input and output meshes are valid.

Implemented By: PMGT

5.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

5.3.1 Entity Module

Secrets: The data structure of a mesh entity, including vertex, edge, and cell.

Services: Stores the complete mesh information generated, and also provides programs to import and export the mesh information.

Implemented By: –

5.3.1.1 Vertex Module (M8)

Secrets: The data structure of a vertex.

Services: Stores the complete vertex information generated and provides programs to import and export the vertex information. The operations on vertices are also included in this module.

Implemented By: PMGT

5.3.1.2 Edge Module (M9)

Secrets: The data structure of an edge.

Services: Stores the complete edge information generated and provides programs to import and export the edge information. The operations on edges are also included in this module.

Implemented By: PMGT

5.3.1.3 Cell Module (M10)

Secrets: The data structure of a cell.

Services: Stores the complete cell information generated and provides programs to import and export the cell information. The operations on cells are also included in this module.

Implemented By: PMGT

5.3.1.4 Mesh Module (M11)

Secrets: The data structure of a mesh.

Services: Stores the complete mesh information generated and provides programs to import and export the cell information. The operations on meshes are also included in this module.

Implemented By: PMGT

5.3.2 Mesh Algorithm Module

Secrets: Algorithms for refining and coarsening a mesh.

Services: Refining and coarsening a mesh.

Implemented By: –

5.3.2.1 Refining Module (M12)

Secrets: Algorithms for refining a mesh.

Services: Refining a mesh.

Implemented By: MPGT

5.3.2.2 Coarsening Module (M13)

Secrets: Algorithms for coarsening a mesh.

Services: Coarsening a mesh.

Implemented By: MPGT

6 Traceability Matrix

A traceability matrix can be used for checking the completeness of the current design. In this section, there are two matrices, the traceability matrix for requirements and the traceability matrix for anticipated changes. The module names and their corresponding numbers are can be found in Section 3

6.1 Traceability Matrix for Requirements

The traceability matrix in Table 2 makes a connection between the modules and the requirements. Modules are listed in the first row and requirements are listed in the first column. If a module, say A, satisfies a requirement, say B, and A is in j-th column and B in i-th row, then there is a check mark ✓in the cell of the i-th row and the j-th column. There is a special column "Doc." It represents the documentation of PMGT. the "Doc" entry is used to fulfill the requirement *Help* (F16). The names of the requirements and their corresponding numbers are listed below for convenience.

F1: RefiningMesh

F2: CoarseningMesh

F3: RefiningOrCoarsening

F4: MeshType

F5: ElmShape

F6: DomainDimension

F7: Conformal

F8: InputDefinition

F9: RCInstruction

F10: OutputStorage

F11: VertexUniqueID

F12: ElmUniqueID

F13: ElmTopology

F14: OutElmOrder

F15: OutVertexOrder

F16: Help

N1: Performance

N2: Precision

N3: Exception

N4: Portability

N5: LookAndFeel

N6: Usability

N7: Maintainability

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	Doc
F1												✓		
F2													✓	
F3												✓	✓	
F4									✓	✓	✓	✓	✓	
F5							✓			✓				
F6								✓				✓	✓	
F7							✓					✓	✓	
F8		✓			✓									
F9												✓	✓	
F10	✓	✓				✓								
F11						✓								
F12						✓								
F13						✓								
F14						✓								
F15						✓								
F16														✓
N1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
N2					✓	✓	✓	✓	✓	✓	✓	✓	✓	
N3	✓	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓	
N4	✓	✓	✓	✓	✓	✓	✓							
N5					✓	✓	✓	✓	✓	✓	✓	✓	✓	
N6			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
N7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 2: Traceability Matrix: Modules and Requirements

6.2 Traceability Matrix for Anticipated Changes

The traceability matrix in Table 3 illustrates the relationship between modules and anticipated changes listed in Section 2. If there is a ✓in an entry of the matrix, the change specified in that row is hidden in the module of the corresponding column.

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13
AC1	✓												
AC2		✓											
AC3			✓										
AC4				✓									
AC5					✓								
AC6						✓							
AC7							✓						
AC8								✓					
AC9									✓				
AC10										✓			
AC11											✓		
AC12												✓	
AC13													✓
AC14							✓			✓			

Table 3: Traceability Matrix: Modules and Anticipated Changes

7 Use Hierarchy between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

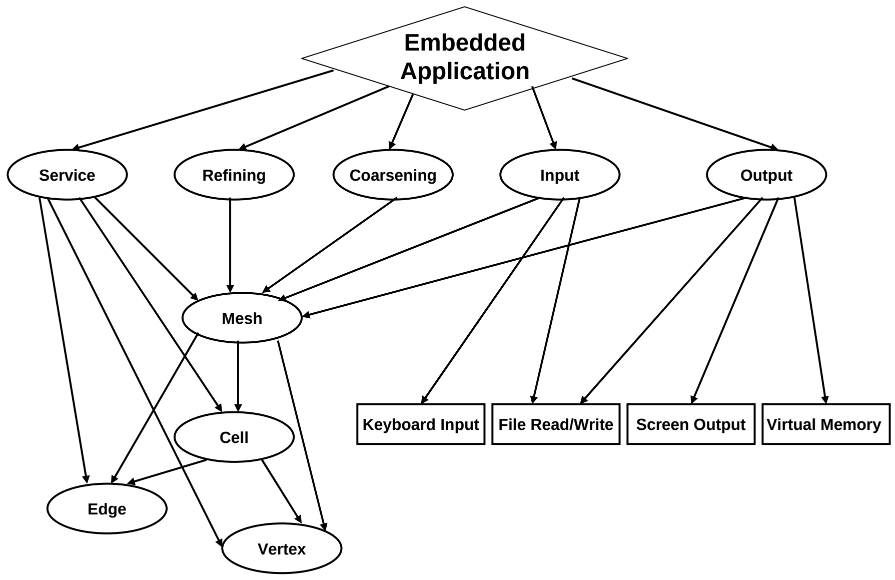

Figure 1: Use Hierarchy among Modules

References

Chien-Hsien Chen. A software engineering approach to developing mesh generators. Master's thesis, McMaster University, Novermber 2003.

A. H. ElSheikh, S. Smith, and S. E. Chidiac. Semi-formal design of reliable mesh generation systems. *Adv. Eng. Softw.*, 35(12):827–841, 2004. ISSN 0965-9978.

D. L. Parnas, P. C. Clements, and D. M. Weiss. The modular structure of complex systems. In *ICSE '84: Proceedings of the 7th international conference on Software engineering*, pages 408–417, Piscataway, NJ, USA, 1984. IEEE Press. ISBN 0-8186-0528-6.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.