

Module Guide (MG) STEM Moiré GPA

Alexandre Pofelski
macid: pofelska
github: slimpotatoes

November 1, 2017

1 Revision History

Table 1: **Revision History**

Date	Version	Notes
31/10/2017	1.0	First Draft

Contents

1	Revision History	i
2	Introduction	1
3	Anticipated and Unlikely Changes	1
3.1	Anticipated Changes	1
3.2	Unlikely Changes	2
4	Module Hierarchy	2
5	Connection Between Requirements and Design	3
6	Module Decomposition	3
6.1	Hardware Hiding Modules (M??)	4
6.2	Behaviour-Hiding Module	4
6.2.1	Input Format Module (M??)	4
6.2.2	Etc.	4
6.3	Software Decision Module	4
6.3.1	Etc.	5
7	Traceability Matrix	5
8	Use Hierarchy Between Modules	6

List of Tables

1	Revision History	i
2	Module Hierarchy	3
3	Trace Between Requirements and Modules	5
4	Trace Between Anticipated Changes and Modules	5

List of Figures

1	Use hierarchy among modules	6
---	---------------------------------------	---

2 Introduction

The module guide document is providing to the reader the decomposition of STEM Moiré GPA into smaller pieces to help the implementation phase. The decomposition is based on the principle of information hiding [2] which interest relies on its flexibility to design change. The module design follows the rules layed out by [1], as follows:

- 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.

The module guide (MG) follows the Software Requirements Specification (SRS) available in the documentation tree. Terminologies, symbols and acronyms used in the document are described in the SRS document. The rest of the document is organized as follows. Section 3 lists the anticipated and unlikely changes of the software requirements. Section 4 summarizes the module decomposition that was constructed according to the likely changes. Section 5 specifies the connections between the software requirements and the modules. Section 6 gives a detailed description of the modules. Section section 7 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 8 describes the use relation between modules.

3 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 3.1, and unlikely changes are listed in section 3.2.

3.1 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.

AC 1 Hardware running STEM Moiré GPA

AC 2 SMH, ICREF, inputs file format

AC 3 Small strain and small gradient of strain assumption

AC 4 SMH simulation from a reference algorithm

AC 5 GPA algorithm

AC 6 Mask function for frequency isolation (in GPA)

AC 7 Fitting algorithm to get unstrained reference (in GPA)

AC 8 Output format

AC 9 Data plotting

3.2 Unlikely Changes

The unlikely changes are design decisions fixing some aspects of STEM Moiré GPA. The lack of modularity is compensated by a simplification in the software design. If any of the elements are modified, important changes would be expected in the design. Therefore the following decisions are expected to stay unchanged.

UC 1 SMH data type (2D arrays)

UC 2 Perfect 2D periodic scanning grid assumption

UC 3 Pixel data type (real number)

UC 4 Keyboard/mouse interface device with user

4 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M 1 Hardware Module

M 2 Input Format Module

M 3 Input GPA Module

M 4 Output Format Module

M 5 Output GPA Module

M 6 GPA Module

M 7 Fourier Transform Module

6 Module Decomposition

Modules are decomposed according to the principle of “information hiding” proposed by [1]. 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. Also indicate if the module will be implemented specifically for the software.

Only the 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.

6.1 Hardware Hiding Modules (M??)

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: OS

6.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour 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: –

6.2.1 Input Format Module (M??)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: [Your Program Name Here]

6.2.2 Etc.

6.3 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: –

6.3.1 Etc.

7 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
R1	M??, M??, M??, M??
R2	M??, M??
R3	M??
R4	M??, M??
R5	M??, M??, M??, M??, M??, M??
R6	M??, M??, M??, M??, M??, M??
R7	M??, M??, M??, M??, M??
R8	M??, M??, M??, M??, M??
R9	M??
R10	M??, M??, M??
R11	M??, M??, M??, M??

Table 3: Trace Between Requirements and Modules

AC	Modules
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 4: Trace Between Anticipated Changes and Modules

8 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. [3] 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

- [1] D. Parnas, P. Clement, and D. M. Weiss, “The modular structure of complex systems,” in *International Conference on Software Engineering*, pp. 408–419, 1984.
- [2] D. L. Parnas, “On the criteria to be used in decomposing systems into modules,” *Comm. ACM*, vol. 15, pp. 1053–1058, December 1972.
- [3] D. L. Parnas, “Designing software for ease of extension and contraction,” in *ICSE ’78: Proceedings of the 3rd international conference on Software engineering*, (Piscataway, NJ, USA), pp. 264–277, IEEE Press, 1978.