Module Guide for EOMEE

Gabriela Sánchez Díaz

December 10, 2020

1 Revision History

Date	Version	Notes
20-11-2020	1.0	MG first draft

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
EOMEE	Equation-of-motion for excited states
UC	Unlikely Change
EOM	Equation of motion
IP	Ionization Potential
DIP	Double Ionization Potential
EA	Electron affinity
DEA	Double Electron affinity
Exc	Excitation
RDM	Reduced Density Matrix
TDM	Transition Density Matrix

See also the corresponding section in the SRS Documentation at SRS.

Contents

List of Tables

List of Figures

3 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 (**?**). We advocate a decomposition based on the principle of information hiding (**?**). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by **?**, as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented 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) is 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 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 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 follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 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 9 describes the use relation between modules.

4 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 4.1, and unlikely changes are listed in Section 4.2.

4.1 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 specific hardware on which the software is running and the libraries provided by the programming language.

AC2: The initial input data (its format and number of parameters).

AC3: How the different routines are called.

AC4: The format of the electron integrals (given in molecular orbital basis or spin orbitals basis)

AC5: The input format of the density matrices.

AC6: The Equation of motion (EOM) methods (more equation definitions could be added).

AC7: How the transition density matrices (TDMs) are implemented.

AC8: How the EOMs are solved.

AC9: The output data format.

4.2 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: How the EOMs are defined in terms of the one- and two-electron integrals and RDMs.

UC3: The representation of the EOM as a generalized eigenvalue problem.

5 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: Hardware-Hiding Module

M2: Control Module

M3: Input Module

M4: Integrals Module

M5: Reduced density matrices (RDMs) Module

M6: EOM Interface Module

M7: IP EOM Module

M8: EA EOM Module

M9: Excitation EOM Module

M10: DIP EOM Module

M11: DEA EOM Module

M12: Generalized Eigenvalue Solver Module.

M13: Output Module

Note that module M1, corresponding to the libraries provided by the operative system, is not going to be reimplemented. Also, the solver module M12 is a partial implementation given that it uses a linear solver and singular value decomposition algorithm form external libraries.

6 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.

Level 1	Level 2	Level 3
Hardware-Hiding Module		
Behaviour-Hiding Module	Control	
	Input	
	Integrals	
	RDMs	
	EOM Interface	IP EOM; EA EOM; DIP EOM; DEA EOM; Excitation EOM
	Output	
Software Decision Module	Solver	

Table 1: Module Hierarchy

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. 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. *EOMEE* means the module will be implemented by the EOMEE 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.

7.1 Hardware Hiding Modules (M1)

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

7.2 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: –

7.2.1 Control (M2)

Secrets: The organization of tasks execution.

Services: Provides the main program: Takes the inputs variables from the user to the Input module. Selects the EOM method to be solved.

Implemented By: EOMEE

7.2.2 Input (M3)

Secrets: The format and structure of the input data. How the values are verified.

Services: Verifies that the input data structure corresponds to the one used by the EOM module. Stores the input values.

Implemented By: EOMEE

7.2.3 Integrals (M4)

Secrets: The format and structure of the one- and two-electron integrals.

Services: Verify that the electron integral's format complies with the internal one from EOM. Store the integrals.

Implemented By: EOMEE

7.2.4 RDMs (M5)

Secrets: The data structure of the one- and two-RDMs and the verification of its values.

Services: Check that the RDMs have the correct format. Verify that their traces adds up to the correct number of particles for the system.

Implemented By: EOMEE

7.2.5 EOM interface (M6)

Secrets: The interface to define an EOM data structure from the input electron integrals and RDMs.

Services: Defines the data and functions to represent an EOM and its associated TDM.

Implemented By: EOMEE

7.2.6 IP EOM (M7)

Secrets: Implements the interface for the IP EOM.

Services: Provides the left- and right-hand side matrices corresponding to the IP EOM equation represented by a generalized eigenvalue problem (see IM1 in the SRS). Provides de TDMs given the density matrices and the eigenvectors solution of the EOM.

Implemented By: EOMEE

7.2.7 EA EOM (M8)

Secrets: Implements the interface for the EA EOM.

Services: Provides the left- and right-hand side matrices corresponding to the EA EOM equation represented by a generalized eigenvalue problem (see IM2 in the SRS). Provides de TDMs given the density matrices and the eigenvectors solution of the EOM.

Implemented By: EOMEE

7.2.8 DIP EOM (M10)

Secrets: Implements the interface for the DIP EOM.

Services: Provides the left- and right-hand side matrices corresponding to the DIP EOM equation represented by a generalized eigenvalue problem (see IM4 in the SRS). Provides de TDMs given the density matrices and the eigenvectors solution of the EOM.

Implemented By: EOMEE

7.2.9 DEA EOM (M11)

Secrets: Implements the interface for the DEA EOM.

Services: Provides the left- and right-hand side matrices corresponding to the DEA EOM equation represented by a generalized eigenvalue problem (see IM5 in the SRS). Provides de TDMs given the density matrices and the eigenvectors solution of the EOM.

Implemented By: EOMEE

7.2.10 Excitation EOM (M9)

Secrets: Implements the interface for the Excitation EOM.

Services: Provides the left- and right-hand side matrices corresponding to the Excitation EOM equation represented by a generalized eigenvalue problem (see IM3 in the SRS). Provides de TDMs given the density matrices and the eigenvectors solution of the EOM.

Implemented By: EOMEE

7.2.11 Output (M13)

Secrets: The format and structure of the output data.

Services: Returns the transition energies, coefficient and TDMs to the user.

Implemented By: EOMEE

7.3 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: –

7.3.1 Solver (M12)

Secrets: The algorithm to solve a generalized eigenvalue problem.

Services: Provides the solution to the EOM system of equations. Uses the left- and right-hand side matrices of the EOM method, a keyword to indicate some matrix orthogonalization algorithm and threshold value for the numerical stability of matrix inversion.

Implemented By: EOMEE and SciPy

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements (Table 2) and between the modules and the anticipated changes (Table 3). Table 3 should show a one to one correspondence between AC and modules. Modules M7 to M11 are implementations of the EOM Interface module (M6), and can be modified independently without changing the interface, therefore, there can still be considered that the previous

requirement is fulfilled for AC6. In the case of AC7, related to de implementation of the TDMs, it maps to the same modules as AC6, therefore the condition is relaxed. This is done taking into consideration that each EOM method has an associated TDM, therefore, these should be implemented as part of the EOM module.

Req.	**Modules**
R1	M1, M3, M2, M4
R2	M3, M5
R3	M3, M4
R4	M6, M7, M8, M10, M11,M9, M12, M2
R5	M13, M2

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M3
AC3	M2
AC4	M4
AC5	M5
AC6	M6, M7, M8, M9, M10, M11
AC7	M6, M7, M8, M9, M10, M11
AC8	M12
AC9	M13

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** 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.

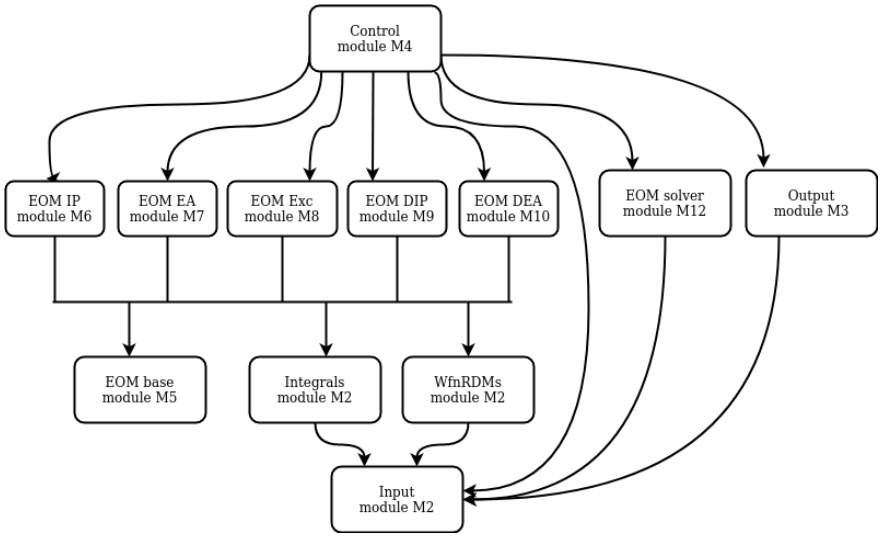

Figure 1: Use hierarchy among modules