SE 3XA3: Software Requirements Specification
Finite State Machine Simulator

Team #16, NonDeterministic Key
Anhao Jiao (jiaoa3)
Kehao Huang (huangk53)
Xunzhou Ye (yex33)

18 March 2022

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
March 18	0.1	Partial complete MG
April 12	1.0	added use hierarchy

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. 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 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) 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 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

AC1: The specific operating system on which the software is running.

AC2: The format of the input data.

AC3: The format of the output data.

AC4: The instruction on how to use the software.

2.2 Unlikely Changes

UC1: Input/Output devices (Input: Keyboard, Output: Picture, and Screen).

UC2: The purpose of the software to generate LaTeX snippets.

3 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.

M1: Hardware-Hiding Module

M2: Machine Module

M3: Util Module

M4: State Module

M5: Condition Module

M6: Transition Module

M7: EventData Module

M8: Event Module

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Machine Module
Software Decision Module	Util Module State Module Condition Module Transition Module EventData Module Event Module

Table 2: Module Hierarchy

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 3.

The Machine Module is the main class of the software that merge and connects all other modules and to ensure the software's functionality.

Software Decision Modules(State Module, Condition Module, Transition Module, EventData Module, Event Module) have all the functionalities and qualities that specified in the SRS.

The State Module is built to satisfy the requirement that users can define valid states for a FSM. The Condition Module is made to meet the requirement that warps a user-defined predicate. The Transition Module is built to satisfy the requirement that users can define transitions between two valid states. The EventData and Event modules are made to meet the requirement that relevant data and feedback will be provided to the users after receiving a series of inputs.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding". 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.

5.1 Hardware Hiding Modules

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

5.2 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: N/A

5.2.1 Machine Module

Secrets: Finite state machine

Services: Allow the user to define states and transitions, and to simulate transitions.

Implemented By: Python Libraries

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 Util Module

Secrets: N/A

Services: Provide internal utilities to be used by other modules.

Implemented By: Python Libraries

5.3.2 State Module

Secrets: State

Services: Store the states of finite state machines defined by the user.

Implemented By: Python Libraries

5.3.3 Condition Module

Secrets: Condition

Services: Wraps a user-defined predicate in order to be evaluated repetitively.

Implemented By: Python Libraries

5.3.4 Transition Module

Secrets: Transition

Services: Representation of a transition managed by a "Machine" instance.

Implemented By: Python Libraries

5.3.5 EventData Module

Secrets: Data of "Event" instance

Services: Collect relevant data related to the ongoing transition attempt.

Implemented By: Python Libraries

5.3.6 Event Module

Secrets: Event

Services: Represents a collection of transitions assigned to the same trigger.

Implemented By: Python Libraries

6 Traceability Matrix

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

Req.	Modules
FR1	M2
FR2	M3, M4, M6
FR3	M3, M4, M6
FR4	M6, M5, M8
FR5	M2, M7, M8
FR6	M2
NFR1	M2
NFR2	M2
NFR3-4	M1, M2, M3, M??, M??, M??
NFR5-6	M2
NFR7-23	M1, M2, M3, M4, M5, M6, M7, M8

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M2, M3, M4, M6
AC3	M2, M3, M4, M6
AC4	M2

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

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

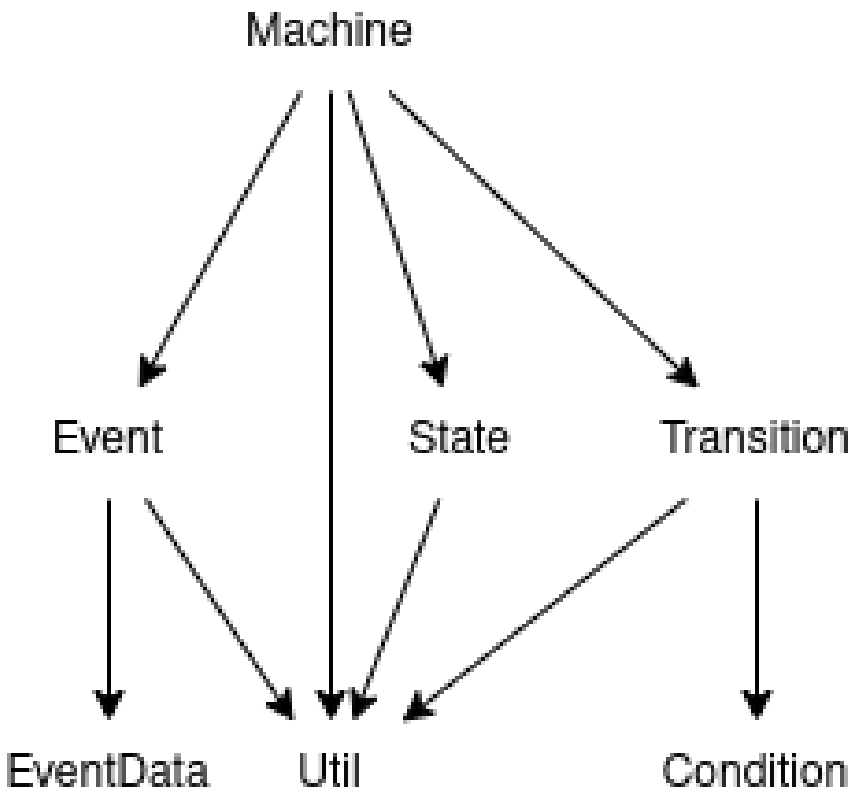

Figure 1: Use hierarchy among modules

References