Software Design Specification

for

MiniThermostat

Document Version <3.0>

Prepared by

Group Name: DoePwr

John Doe	0000000	doe@mcmaster.ca

Instructor:	Dr. K. Sartipi
Course:	Software Engineering 3M04/3K04
Lab Section:	X
Teaching Assistant:	Jane Doe
Date:	01/01/01

Contents

List of Figures

List of Tables

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	John Doe	Complete first version of the MiniThermostat Software Design Specification document	01/01/05
2.0	John Doe	1. Revised Component Diagram – added two interfaces: section 2 2. Revised Interface descriptions	01/20/05
3.0	John Doe	Added section 4, low-level design.	02/01/05

1 Introduction

This document provides a complete example of the third version of a Software Design Specification document for a small Thermostat software system (namely MiniThermostat). This document is primarily based on the developed earlier MiniThermostat SRS document. In the following sections, we specify the purpose of this document, the overview of the MiniThermostat system and the sources used in the production of this document.

1.1 Document Purpose

The purpose of this document is to provide a high-level overview of the architecture and a low-level design for the MiniThermostat Software system. In addition, it outlines the different strategies and methods used to produce the most effective architectural structure for the software.

1.2 System Overview

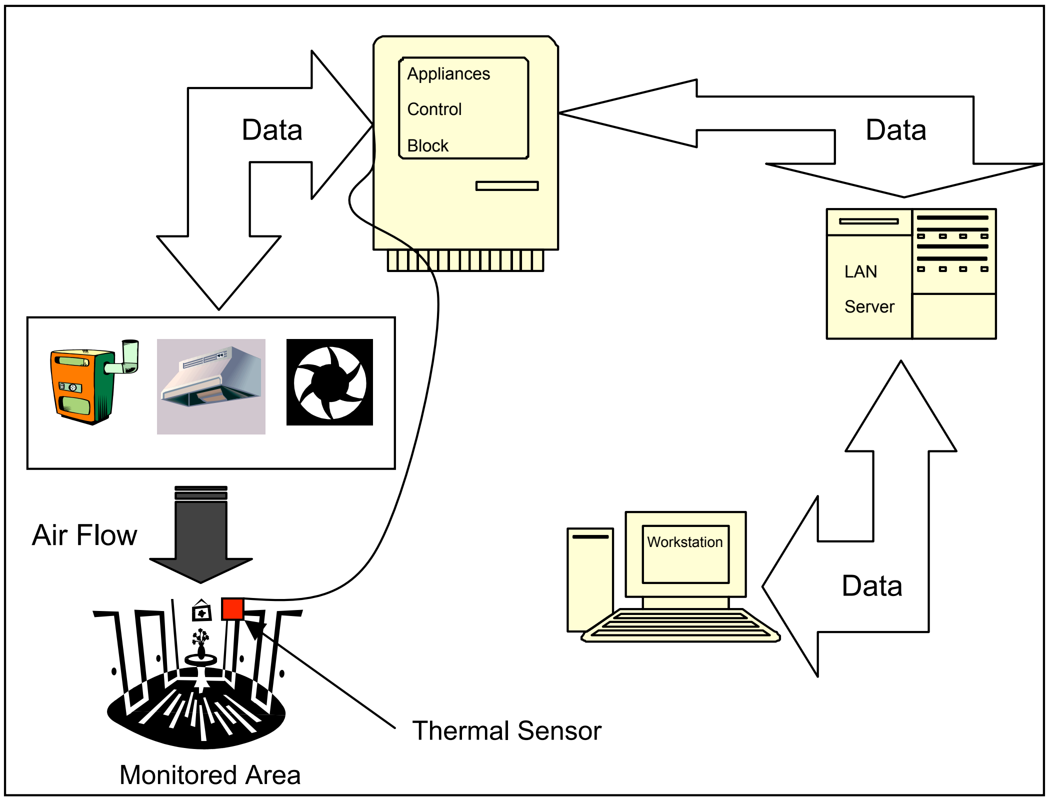

Figure 1: *MiniThermostat Environment*

MiniThermostat Software System runs on a single workstation computer. The minimal requirements related to the workstation, shall adhere to the ones specified in section 2.4 (Operating Environment) of the MiniThermostat SRS v. 1.0 document. The system will make a use of an appropriate LAN communication link, to communicate with the ACB. The ACB is programmed to receive data from a thermal sensor, located in a predefined area, and operate the power relays of three appliances: Furnace, Air Conditioner and a Fan. The system, will communicate with the

ACB to operate these appliances and sample the temperature in the monitored area using functions provided by the manufacturer of the ACB in controlblock.lib library. MiniThermostat main purpose is to provide the end-user with the ability to control climate settings in the monitored area, by either changing the different settings directly or through user defined programs. For more details, please refer to MiniThermostat Software Requirements Specification document v. 1.0.

1.3 Definitions, Acronyms and Abbreviations

ACB – Appliances Control Block
GUI – Graphical User Interface
LAN – Local Area Network
UI – User Interface

1.4 Supporting Materials

The following standards apply:

J-STD-016-1995	IEEE/EIA Standard for Information Technology, Software Lifecycle Processes, Software Development, Acquirer-Supplier Agreement
IEEE-STD-P1063	IEEE Standard for Software User Documentation

The following texts and documents have been used in the process of developing this document:

[1] J. Rumbaugh et al. *Object Oriented Modeling & Design*, Upper Saddle River, NJ: Prentice Hall, 1991.

[2] C. Ghezzi et al. *Fundamentals of Software Engineering*. Upper Saddle River, NJ: Prentice Hall, 2003.

[3] J. Doe, *MiniThermostat Software Requirements Specification*, 2005.

1.5 Document Overview

The next section of the MiniThermostat SDS v. 3.0 provides the architectural view of the system. It illustrates how the system was decomposed into several main components and what were the reasons for this particular decomposition. Subsections of section two, describe in detail the components and their corresponding interfaces. Section three of the document, provides a Control view of the MiniThermostat system and explanations regarding the different state transitions. Finally, in section four the reader will find the low-level design of the system. Low-level design includes an overview of the modules and a detailed design of every module.

2 Architecture

This section provides the architecture design of the MiniThermostat software system. It includes the final version of the system Component Diagram which illustrates the different components, their interfaces and the dependencies between components. The main principles that led to the proposed design are:

- Anticipation of change
- Maintainability
- Separation of concerns
- Understandability

Section 2.2 discusses in detail the different components and how they adhere to the listed above principles. The next section provides the Component Diagram and the details about the proposed architecture.

2.1 Overview

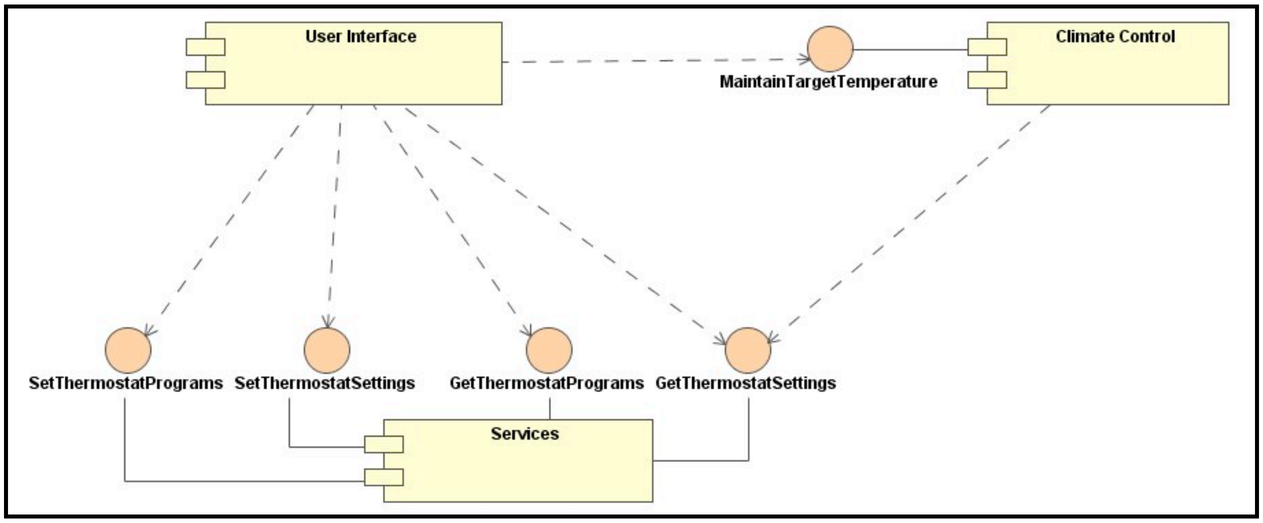

Figure 2: *MiniThermostat Component Diagram (Revised)*

As illustrated in Figure 2, MiniThermostat architecture consists of three main components:

1. User Interface
2. Climate Control
3. Services

In order to make the architecture more understandable, maintainable and adoptable for changes it was decomposed according to the different functional areas the system covers. All three components cover specific functional areas, as specified in the MiniThermostat SRS document, and each can be easily modified without affecting the others. By decomposing the system to only three main components, and providing clear and simple interfaces for each component, the architecture becomes more understandable and the system is easier to maintain.

2.2 User Interface

User Interface (UI) component provides the end-user with graphical interface, and sequences the different functions provided by the other components. UI does not provide any interfaces to other components and has five dependencies. UI depends on:

1. **SetThermostatPrograms / GetThermostatPrograms** interfaces provided by the *Services* component. Through these interfaces, UI provides the end-user with the ability to view and modify eight different programs. Every program includes the weekday and daytime when it is to be executed, and the target temperature the system should maintain.
2. **SetThermostatSettings / GetThermostatSettings** interfaces provided by the *Services* component. Through these interfaces, UI provides the end-user with the ability to modify the different MiniThermostat settings.
3. **MaintainTargetTemperature** interface provided by the *Climate Control* component. Through this interface, the main function of monitoring and controlling a climate in a predefined area can be achieved for the MiniThermostat system.

By restricting the functions of sequencing and interacting with the end-user, to this specific component, the system is more maintainable in a sense that every problem related to graphical interface can be traced directly, and exclusively to this component. In addition this component completely covers the *User Interface* functional area specified in MiniThermostat SRS, thus contributing to architecture understandability.

2.3 Services

The services provided by the MiniThermostat system to the end-user are:

1. View, modify and save eight different programs.
2. Change selected, user defined, settings (for details refer to MiniThermostat SRS).

Services component handles data storage and data retrieval tasks corresponding to the different MiniThermostat services. Services component does not depend on any other component of the system and provides four interfaces for other components. These interfaces are:

1. **SetThermostatPrograms / SetThermostatSettings** interfaces provide the other components with the ability to store any changes related to the different MiniThermostat settings or programs.
2. **GetThermostatSettings / GetThermostatPrograms** interfaces provide the other components with the ability to retrieve the current MiniThermostat Settings, or programs.

As with *UI*, the restrictions of data handling tasks to one specific component aids in error tracing and maintenance. Any error related to data handling and storage, or to one of the services can be traced directly to this component. In addition, this restriction adds to the understandability of the system since the component responsibilities cover only one functional area.

2.4 Climate Control

Climate Control component covers the responsibilities of obtaining the current target temperature and maintaining it. This is the only component that interfaces with controlblock.lib library. The functions from this library are used to operate the different appliances (the complete list of the functions can be found in Appendix A). Climate Control depends on one interface and provides one interface. Climate Control depends on:

1. **GetThermostatSettings** interface provided by the *Services* component. Climate Control uses this interface to obtain information related to the different MiniThermostat settings and most importantly the current target temperature.

The interface provided by the Climate Control component is **MaintainTargetTemperature**. UI depends on this interface to provide the system with the ability to monitor the current temperature in the area and maintain the target temperature.

3 High-Level Design

The High-Level Design section describes in further detail the interactions between the system components and their corresponding interfaces. To illustrate the dynamic behaviour of the MiniThermostat system, Control view using Statecharts has been used. Statecharts model the behaviour from the perspective of a single entity. The entity that is modelled in Figure 2 is the complete MiniThermostat system.

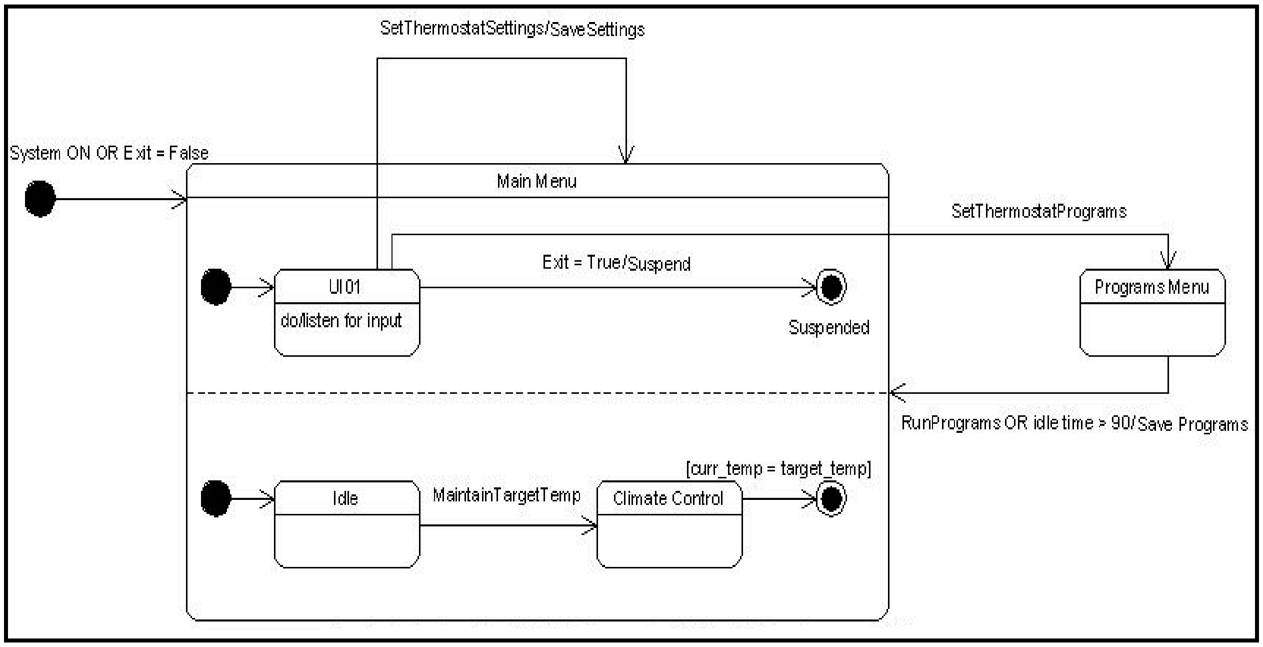

Figure 3: *Control View of the MiniThermostat System using Statecharts*

Two events will trigger the transition of the system from the initial state to the concurrent state *Main Menu*. The first event is the event of the system being turned ON (application started). The second event is for the case where the system is already ON, but is suspended. Since, even when suspended, the system responds to selected user inputs, the input that will generate the event *Exit = False*, will start-up the system.

The system starts by providing the end-user with the Main Menu interface and then waits for user inputs. The system performs the task of *Climate Control* concurrently. Initially the system is in the idle state, and the event *MaintainTargetTemp* triggers the transition to *Climate Control*. Every time, the end-user makes an input, this input is being checked for validity. Valid inputs will trigger events that will indicate whether a *Setting* or a *Program* related event occurred. For the case, when a setting is being changed *UI* uses the *SetThermostatSetting* interface provided by the *Services* component. *SetThemrostatSettings* is the service that triggers the transition back to the composite state *Main Menu*. This transition is necessary, so that the changed settings will be in effect. Before returning to the concurrent state, the action of *Save Settings* is executed.

The same logic applies for the case where a programs related change occurred (i.e., *View Programs* button pressed in UI01). *UI* uses the *SetThermostatPrograms* interface and the service that triggers the transition to *Programs Menu* state is *SetThermostatPrograms.* Transition back to the concurrent state *Main Menu* occurs when the end-user decides to run the modified programs. At this time, once again the transition to the concurrent state is necessary in order for the changes to be in effect. Before returning to the *Main Menu* state, the action of *Save Programs* is executed. This service is also provided through the *ManageThermostatPrograms* interface.

With the occurrence of event *Exit = True*, the system executes the action *Suspend* and enters the final state *Suspended*. Note that the system will still be in the concurrent state of *Climate Control.*

4 Low-Level Design

The following sections describe the low-level design of the MiniThermostat software system. Components described earlier, in section two, are further decomposed into modules. When decomposing the system into modules the major design principles followed were separation of concerns and information hiding. The next section provides an overview of the modules and later sections describe in detail the module interface and its design.

4.1 Modules Overview

MiniThermostat system was decomposed to 6 independent modules. A brief overview of every module is provided below:

Name:	**Command**
File Name:	main
Naming Convention:	All functions contained in this module shall have a prefix ma_
Short Description:	The Command Module is responsible for sequencing the other modules and functions.
Container Component:	This module is part of the User Interface component. Since sequencing is performed by the User Interface component, this task had to be isolated and confined to a single module.

Name:	**Main Menu Interface**
File Name:	man_menu
Naming Convention:	All functions contained in this module shall have a prefix mm_
Short Description:	Provides the user with an interface to change MiniThermostat main settings.
Container Component:	This module is part of the User Interface component. By isolating the functions of the Main Menu interface to a single module we were able to hide the actual implementation of this interface and achieved a system that is easier to maintain and change. In addition, this module can be substituted by a different one with a new graphical interface.

Name:	**Programs Menu Interface**
File Name:	prgrms_menu
Naming Convention:	All functions contained in this module shall have a prefix pm_

Short Description:	Provides the user with an interface to change MiniThermostat main settings.
Container Component:	As the previous modules, this module is part of the User Interface Component. The rationale for confining the functions of Programs Interface to a specific module is the same as for Main Menu Interface.

Name:	**Climate Control**
File Name:	app_cntrl
Naming Convention:	All functions contained in this module shall have a prefix ac_
Short Description:	Operates appliances according to the different temperature settings.
Container Component:	This module is actually the Climate Control component since its functionality is already quite narrow there was no need to decompose it further.

Name:	**Settings Data Manager**
File Name:	settings_mngr
Naming Convention:	All functions contained in this module shall have a prefix sd_
Short Description:	Handles data storage and retrieval tasks related to MiniThermostat settings.
Container Component:	This module is part of the Services component. Services component by definition, deals with two different types of data and several different files. By restricting the management of settings data to this module and hiding the actual methods of data storage and retrieval related to settings, an independent, replaceable and easily maintainable structure was achieved.

Name:	**Programs Data Manager**
File Name:	programs_mngr
Naming Convention:	All functions contained in this module shall have a prefix pd_
Short Description:	Sorts the programs by order and handles data storage and retrieval tasks related to MiniThermostat programs.
Container Component:	This module is part of the Services component. The rationale for restricting the functions programs data management to

this module is the same as for Settings Data Manager.

4.2 Module Specifications

The following sections provide a more detailed overview of module interface specifications and its design. To specify the interfaces we list the module internal state variables and then provide the semantics for its access functions. In the design section we provide a list of internal functions and describe the design of each function using Statechart diagrams. (Please note that in this example we have included specification for only two modules. The rest of the modules can be easily specified in a similar manner.)

4.2.1 Climate Control Module

As stated earlier, *Climate Control* module provides the system with the most important functionality – operating the furnace, air conditioner and the fan to maintain the target temperature.

4.2.1.1 Interface Specification

Used External Modules:	controlblock.lib; Settings Data Manager;

Used External Data Type:	SettingsStruct = { v_Fan: String
	v_Season: String
	target_Temp: Integer
	}

Internal State Variables:	Fan: Boolean AC: Boolean Furnace: Boolean Curr_Temp: Integer Settings: SettingsStruct
Internal Constants:	THRESH:=3

Access Function/s	***Description***
ac_Maintain_Target_Temp()	This function does not have any inputs, or outputs. It will fetch the most current settings and temperature in the area. Based on the extracted information it will power ON/OFF the appliances.

4.2.1.2 Detailed Design

This section provides the detailed internal design of the *Climate Control* module functions. Statechart diagrams have been used to illustrate the transitions between the different states of the state variables specified in section 4.2.1.1

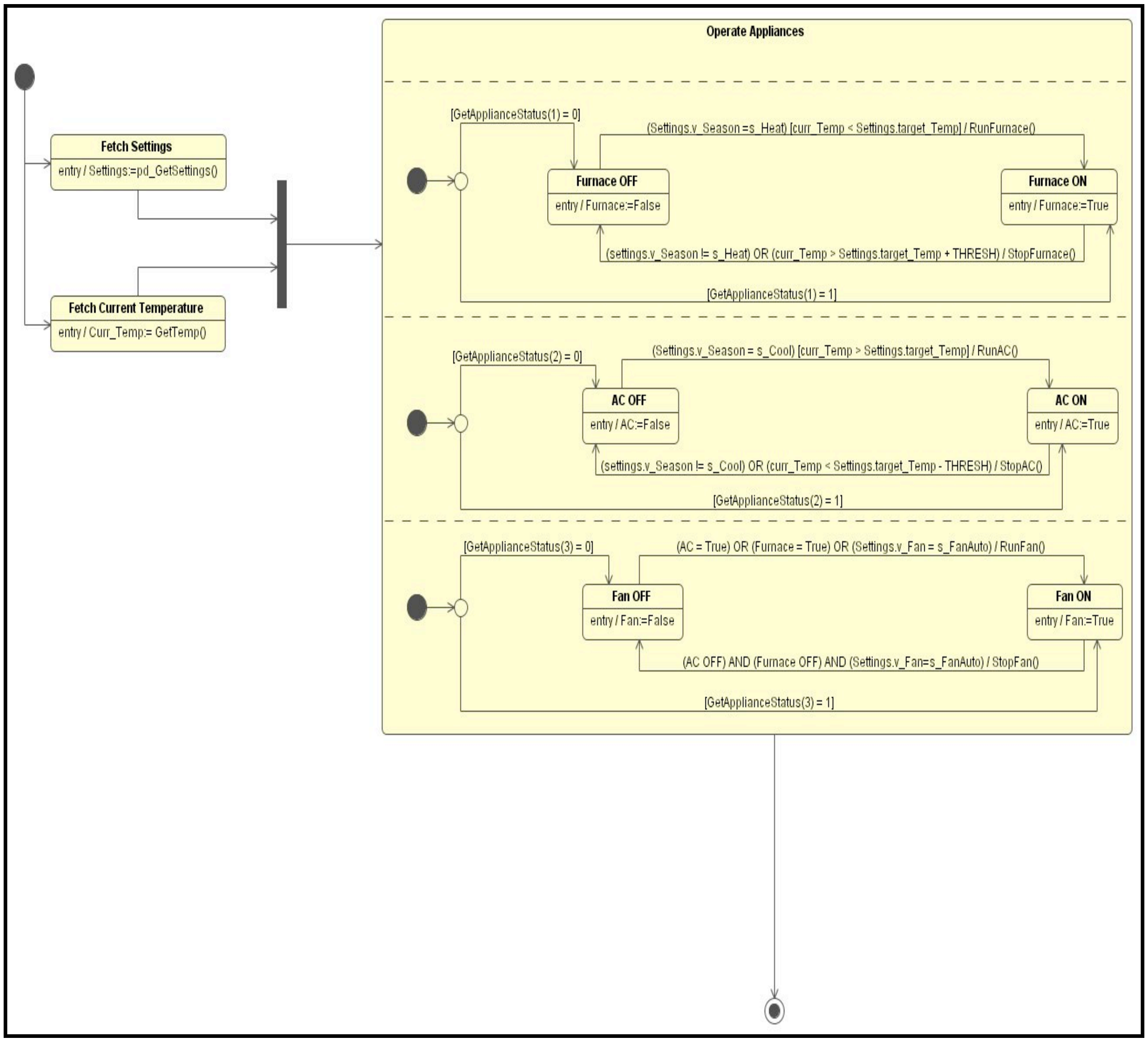
Operate Appliances
Fetch Settings
entry / Settings:=pd_GetSettings()
Fetch Current Temperature
entry / Curr_Temp:= GetTemp()
[GetApplianceStatus(1) = 0]
(Settings.v_Season =s_Heat) [curr_Temp < Settings.target_Temp] / RunFurnace()
Furnace OFF
entry / Furnace:=False
Furnace ON
entry / Furnace:=True
(settings.v_Season != s_Heat) OR (curr_Temp > Settings.target_Temp + THRESH) / StopFurnace()
[GetApplianceStatus(1) = 1]
[GetApplianceStatus(2) = 0]
(Settings.v_Season = s_Cool) [curr_Temp > Settings.target_Temp] / RunAC()
AC OFF
entry / AC:=False
AC ON
entry / AC:=True
(settings.v_Season != s_Cool) OR (curr_Temp < Settings.target_Temp - THRESH) / StopAC()
[GetApplianceStatus(2) = 1]
[GetApplianceStatus(3) = 0]
(AC = True) OR (Furnace = True) OR (Settings.v_Fan = s_FanAuto) / RunFan()
Fan OFF
entry / Fan:=False
Fan ON
entry / Fan:=True
(AC OFF) AND (Furnace OFF) AND (Settings.v_Fan=s_FanAuto) / StopFan()
[GetApplianceStatus(3) = 1]

Figure 4: *ac_Maintain_Target_Temp Statechart*

4.2.2 Programs Data Manager Module

Programs Data Manager provides other modules with the ability to store and retrieve eight user defined Thermostat programs.

4.2.2.1 Interface Specification

Used External Modules:	NONE
Used External Data Type:	NONE

Exported Data Type:	ProgramStruct = { i_Weekday: String
	i_Time: integer
	i_Temp:integer
	}

Exported Constants	SIZE = 8
Internal State Variables:	Error:Boolean
Internal Constants:	NONE

Access Function/s	***Description***
ProgramStruct *pd_GetPrograms ()	This function does not require any inputs and it returns a pointer to a location in memory where the programs were extracted. It will allocate a memory block that will fit an array of type ProgramStruct of size SIZE, access the files where the programs are saved, read from the file and store the results in the array. If for some reason the function fails to perform the operation the return value will be NULL.
int pd_SavePrograms (ProgramStruct Array[])	This function will receive as an argument an array of type ProgramStruct and store it in a special file. If successful the function will return 1 and 0 otherwise. The array must be of size SIZE.

4.2.2.2 Detailed Design

Internal Function/s	***Description***
pd_SortPrograms (ProgramStruct Array[])	This function will accept an array of type ProgramStruct and sort it in the following order: Day ➔ Time. The array must be of size SIZE.

Below we have provided Statecharts to illustrate the internal design of the access functions.

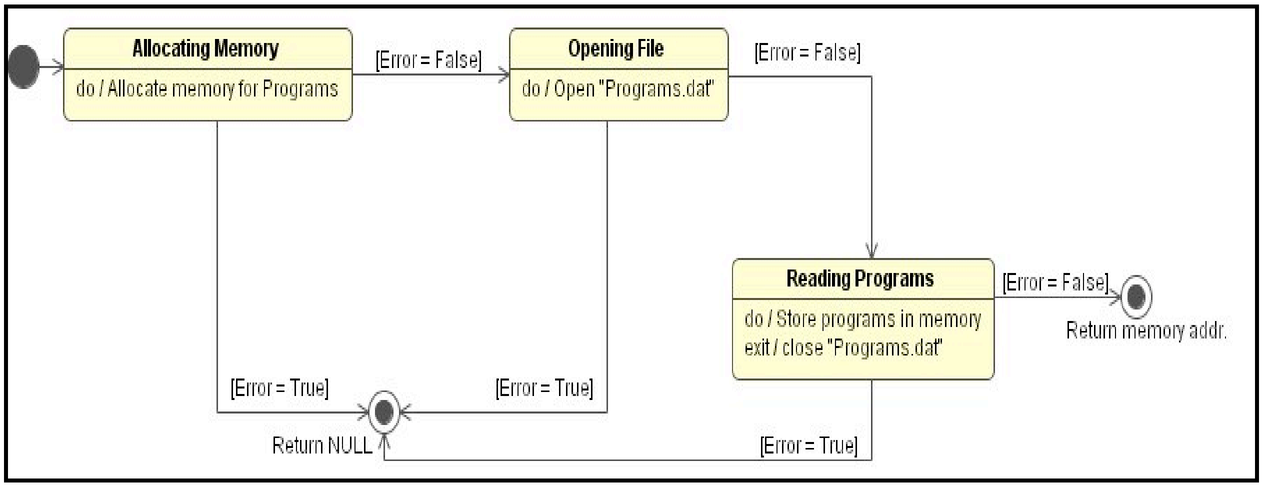

Figure 5: *pd_GetPrograms() Function Statechart*

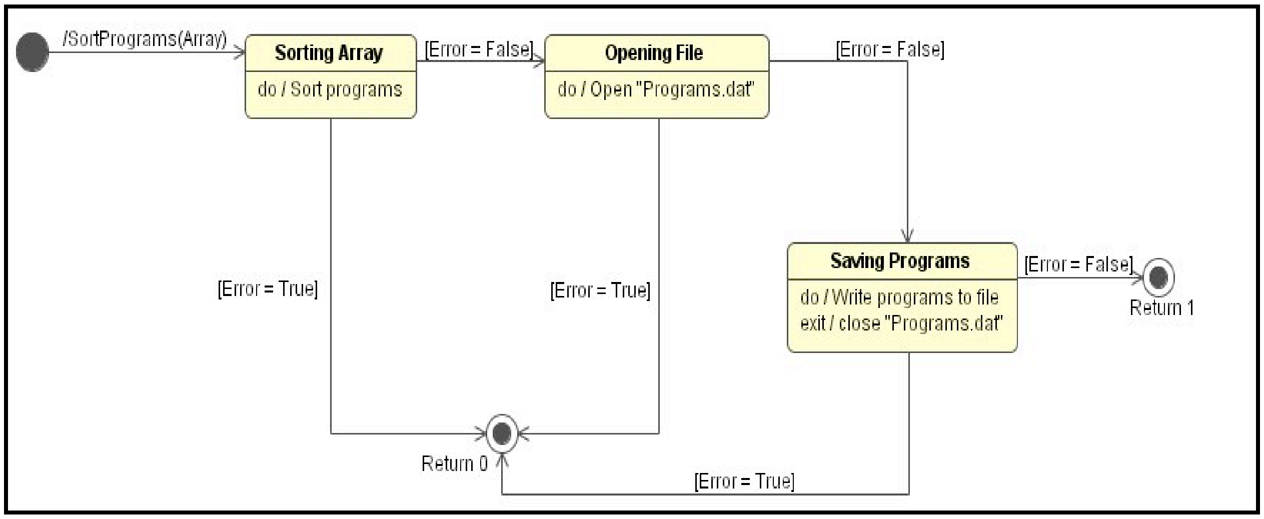

Figure 6: *pd_SavePrograms() function Statechart*

Appendix A – controlblock.lib Functions

The following functions were provided in the library controlblock.lib by the manufacturer of the ACB:

Table 1: *ACB functions*

Function:	**Description:**
int GetTemp(void)	Returns then current temperature in the monitored area. The temperature is returned in degrees Celsius and is rounded up to the closest integer.
void RunFurnace(void)	Powers ON the Furnace power relay
void RunAC(void)	Powers ON the Air Condictioner power relay
void RunFan(void)	Powers ON the Fan power relay
void StopFurnace(void)	Powers OFF the Furnace power relay
void StopAC(void)	Powers OFF the Air Conditioner power relay
void StopFan(void)	Powers OFF the Fan power relay
int GetAppliancesStatus(int appliance)	Given a valid input (1|2|3), where: Furnace == 1 AC == 2 Fan == 3 This function will return a 1 if the appliance is ON and 0 if the appliance is OFF. For any other input the return value will be 2.