Part 2

Software System documentation

Table of Contents

2.1 Introduction 5
2.1.1 Concept decisions.....5
2.1.2 Port components.....6
2.1.3 Parallelism.....7
2.1.3.1 Scheduler 7
2.1.4 Modeling of time 9
2.1.4.1 Timer 9
2.1.5 User Interface 10
2.2 UML-Concept 10
2.2.1 Short description of differences to the analysis 11
2.2.1.1 class House 11
2.2.1.2 class PRTempController 11
2.2.1.3 class PresenceController 11
2.2.1.4 class Room..... 11
2.2.1.5 class RoomtemperatureController.....11
2.2.1.6 class ScheduledFct..... 11
2.2.1.7 class Scheduler..... 11
2.2.1.8 class Section..... 12
2.2.1.9 class Temperatureregulator 12
2.2.1.10 class Timer 12
2.2.1.11 class Window 12
2.2.2 UML- class diagrams 13
2.2.2.1 Package structure..... 13
2.2.2.2 Time 14
2.2.2.3 Temperature adjustment control 15
2.2.2.4 Scheduling 16
2.2.2.5 Rain monitoring..... 16

2.2.2.6 Hardware_Wrapper………………………………………………… 17
2.2.2.7 User-Interface (Control Panel) ……………………………………... 17
2.2.2.8 Constructors………………………………………………………… 18
2.2.3 UML-Classtables 19
2.2.3.1 Classtable of class CentralHeatingSite…………………………………19
2.2.3.2 Classtable of class Clock ……………………………………….…. 20
2.2.3.3 Classtable of class ontact…………………………………………21
2.2.3.4 Classtable of class ControlPanel……………………………………22
2.2.3.5 Classtable of class Day ………………………………………..24
2.2.3.6 Classtable of class House……………………………………………25
2.2.3.7 Classtable of class Motor……………………………………………26
2.2.3.8 Classtable of class PRTempController ……………………….…27
2.2.3.9 Classtable of class Portbaustein………………………………..29
2.2.3.10 Classtable of class PresenceController 30
2.2.3.11 Classtable of class Radiator 31
2.2.3.12 Classtable of class Room 32
2.2.3.13 Classtable of class RoomtemperatureController 34
2.2.3.14 Classtable of class ScheduledFct 35
2.2.3.15 Classtable of class Scheduler 36
2.2.3.16 Classtable of class Section 37
2.2.3.17 Classtable of class Temperatureregulator 38
2.2.3.18 Classtable of class Temperaturesensor 42
2.2.3.19 Classtable of class Time 43
2.2.3.20 Classtable of class Timer 44
2.2.3.21 Classtable of class Window 45
2.2.4 UML-sequence diagrams 47
2.2.4.1 sequence diagrams for UseCase Presence………………..………….. 47
2.2.4.2 sequence diagrams for UseCase WindowPosition...……….………... 48
2.2.4.3 sequence diagrams for UseCase HeatingControl…………..………… 48
2.2 4..4 sequence diagrams for UseCase RainMonitoring 49
2.2.4.5 sequence diagrams for UseCase TemperatureTooHigh…..……………49
2.2.4.6 sequence diagrams for UseCase ValuesSetting……………………….. 50
2.2.5 UML-collaboration diagrams 51
2.2.5.1 Presence……………………………………………………….. 51
2.2.5.2 WindowPosition…………………………………..……………… 52
2.2.5.3 HeatingControl……………………………………...……………… 53

2.2.5.4 RainMonitoring …………………………………………… 54
2.2.5.5 TemperatureTooHigh ……………………………………………54
2.2.5.6 ValuesSetting ……………………………………………55

2.2.6 UML-state-diagrams 56

2.2.6.1 House ……………………………………………56
2.2.6.2 PRTempController ……………………………………………56
2.2.6.3 PresenceController ……………………………………………57
2.2.6.4 RoomtemperatureController……………………………………………58
2.2.6.5 Temperatureregulator……………………………………………59
2.2.6.6 Window……………………………………………61

2.2.7 Traceability matrices analysis <-> concept…………………………………………… 62

2.2.7.1 class ControlPanel ……………………………………………62
2.2.7.2 class House ……………………………………………62
2.2.7.3 class PRTempController ……………………………………………63
2.2.7.4 class RainChecker ……………………………………………63
2.2.7.5 class Room……………………………………………63
2.2.7.6 class RoomtemperatureController……………………………………………64
2.2.7.7 class Section……………………………………………64
2.2.7.8 class Temperatureregulator ……………………………………………64
2.2.7.9 class Window……………………………………………65

2.3 Component Requirements 66

2.3.1 reusable class……………………………………………66
2.3.2 **reusable general data type**……………………………………………66
2.3.3 **data types from UML-Concept**……………………………………………67
2.3.4 class definitions ……………………………………………67

2.3.4.1 class CentralHeatingSite……………………………………………68
2.3.4.2 class Cloc……………………………………………68
2.3.4.3 class Contact ……………………………………………68
2.3.4.4 class ControlPanel ……………………………………………69
2.3.4.5 class Day ……………………………………………70
2.3.4.6 class House ……………………………………………70
2.3.4.7 class Motor……………………………………………71
2.3.4.8 class PRTempController ……………………………………………71
2.3.4.9 class PortComponent……………………………………………72
2.3.4.10 class PresenceController ……………………………………………72
2.3.4.11 class Radiator……………………………………………73

2.3.4.12 class Room 74
2.3.4.13 class RoomtemperatureController 75
2.3.4.14 class ScheduledFct 75
2.3.4.15 class Scheduler 76
2.3.4.16 class Section 76
2.3.4.17 class Temperatureregulator 77
2.3.4.18 class Temperaturesensor 78
2.3.4.19 class Time 78
2.3.4.20 class Timer 78
2.3.4.21 class Window 79
2.4 Verifications 80
2.4.1 Verification checklists........80
2.4.2 Classification of failures........82
2.4.3 Results of verification........83
2.5 Validation 84
2.5.1 Component test cases 84
2.5.1.1 class House 84
2.5.1.2 class PresenceController 86
2.5.1.3 class PRTempController 87
2.5.1.4 class ControlPanel 89
2.5.1.5 class Room........91
2.5.1.6 class RoomtemperatureController........92
2.5.1.7 class Temperatureregulator 95
2.5.1.8 class Window........102
2.5.2 Results of component tests 105
2.5.2.1 class House 105
2.5.2.2 class PresenceController 105
2.5.2.3 class PRTempController 106
2.5.2.4 class ControlPanel 106
2.5.2.5 class Room........107
2.5.2.6 class RoomtemperatureController........107
2.5.2.7 class Temperatureregulator 108
2.5.2.8 class Window........109

Chapter 2

Software System documentation

2.1 Introduction

This document describes the system architecture of the climatic control system. The description of requirements made in chapter 1.3 provide a basis.

Similarly to the analysis the concept will be object-oriented and created by using the OMT method. As notation again the UML (Unified Modeling LANGUAGE) is used.

2.1.1 Concept decisions

EE	Description of Concept decisions
EE1	The target system is a central control computer with port components over that • all sensors will be read • all actuators will be regulated
EE2	Parallelism is simulated by use of a simple scheduler.
EE3	The time is realized by a reusable component called Timer

Chart 1: Concept decisions

2.1.2 Port components

All sensors and actuators are regulated by port components. Each type of actuators and sensors possesses its own component with a sufficient number of connections. For control the hardware manufacturer provides predefined LIBRARIES. Therefore these components in the object model are realized as a class.

There are the following kinds of input and output ports:

1. *Bit-Input ports* for contacts.
   Inquired by command: int CallBitInputPort(int LineNr1, int LineNr2, int PortNr)
2. *Integer-Input ports* for temperatures.
   Inquired by command: int CallIntInputPort(int LineNr1, int LineNr2, int PortNr)
3. *Bit-Output ports* for simple actuators.
   Inquired by command: int CallBitOutputPort(int LineNr1, int LineNr2, int PortNr, int value)
4. *Integer-Output ports* for more complex actuators.
   Inquired by command: int CallIntOutputPort(int LineNr1, int LineNr2, int PortNr, int value)

The actuators and sensors existing in the house are assigned as follows:

- Bit-Input ports for contacts: presence contact, canted contact, open contact, rain contact
- Integer- Input ports for temperature: Outside-, Inside(room)temperature, Heater temperature sensor
- Bit-Output ports for simple actuators: Sway- und Cant Motors for the windows
- Integer- Output ports more complex actuators: Boiler temperature, Heater valve

The following port assignments are specified:

Port	PortNr	Port Type	LineNr1	LineNr2
Room temperature sensor	0	IntInput	Section-Nr	Room-Nr
Presence contact	1	BitInput	Section-Nr	Room-Nr
Heater temperature sensor	2	IntInput	Section-Nr	Room-Nr
Heater valve	3	IntOutput	Section-Nr	Room-Nr
Cant contact	4	BitInput	Section-Nr	Room-Nr
Open contact	5	BitInput	Section-Nr	Room-Nr
Cant motor	6	BitOutput	Section-Nr	Room-Nr
Sway motor	7	BitOutput	Section-Nr	Room-Nr
Boiler temperature	8	IntOutput	0	0
Outside Temperature Sensor	9	IntInput	0	0

Chart 2: Portassignment

Port	PortNr	Port Type	LineNr1	LineNr2
RainContact	10	BitInput	**0**	**0**

Chart 2: Portassignment

2.1.3 Parallelism

The past descriptions of the classes proceed from an inherent parallelism with classes. A parallel implementation of the classes would be possible on the target computer (and/or on its operating system), but for this a treatment of communication and synchronisation problems would be necessary. In order to avoid the complexity which is added thereby, parallel processing is simulated. This is realized by a simple scheduler, which activates certain methods of the objects successively periodically.

2.1.3.1 Scheduler

The class scheduler was developed in the context of the CoDEx project. The following requirements of the scheduler and the concept of the scheduler are literally taken from the system model of the BauOS-system belonging to the CoDEx project.

2.1.3.1 Requirements

The draft contains a set of automats, which work parallel and independently. These automats are to work in the implementation quasi-parallel. The realization is to look in such a way that a certain method of an object is implemented in regular intervals. Inside this method the behavior of the automat belonging to the object is simulated. For the periodic activation a primitive scheduler is needed, with which the executing methods must be registered.

2.1.3.1.2 Concept

For the realization of the requirements the concept provides two classes. On the one hand a basic class, from which all objects must be derived. This base class makes a virtual function available, which must be overloaded in the derived classes and which simulates the behavior of the associated automat. The second class realizes the scheduler. The periodic activation is implemented with the help of the UNIX signal mechanism.

2.1.3.1.2.1 The Base class ScheduledFct

Each class, which has got a function and which is to be periodically activated, must be derived from the class ScheduledFct(stands for ‘Scheduled Function’).

```
class ScheduledFct {

public:
  virtual void callBack (void);
```

```
  void registerCallBack (void);
  void unregisterCallBack(void);
} // class ScheduledFct
```

This class has two methods. The virtual method callBack has to be over-defined in derived classes. It realizes that function, which can be periodically called. If several functions are to be periodically executed, then these must be called by the function callBack.

The method registerCallBack ensures the fact that the object is registered with the scheduler for the periodic activation. This method can be called by the object itself or (if it is left public) from outside. Canceling the registration is done via the call of the function unregisterCallBack by which the periodic activation can be stopped again. The implementation of the methods registerCallBack and unregisterCallBack will be described in section 2.1.3.1.2.

2.1.3.1.2.2 The class Scheduler

The Scheduler is realized by the class Scheduler:

```
class Scheduler {
  friend class ScheduledFct;

  static Scheduler *myPtr;

  int intervall;
  CallBackList callBacks;

  static void signalCallBack (void);
  void schedule (void);
  void registerCallBack (ScheduledFct *fct);

public:
  Scheduler (int interv);
  ~Scheduler (void);

  void start (void);
  void stop (void);
};
```

Instances

In the control system an instance of this class must exist. No more than one instance may exist.

Use

The constructor of the class scheduler expects a parameter, which indicates in which distances the periodic activation takes place in. This interval is specified in microseconds. After producing an instance of the class scheduler, objects which were derived from the class ScheduledFct can register itself at this instance(section 2.1.3.1.2.1).

By the call of the method ‘start’ the scheduler begins with the periodic activation of all registered objects. The registration of further objects is still possible thereafter. By the call of the method ‘stop’ the activity of the scheduler can be interrupted and be taken up if necessary with ‘start’ again. That does not have any influence on the registered objects.

Implementation

The periodic activation of methods is realized by the fact that regularly a UNIX signal (SIGALRM) is produced and intercepted. In the course of the treatment of the signal the methods are then implemented.
With the arrival of a signal UNIX calls a function, which must have been registered before as a Signal-Handler. For this task simply a method of a class cannot be used. Methods always expect a pointer with the call on the associated object (this-Pointer). At the call this pointer as Signal-Handler will not be passed by the operating system. The problem can be circumvent by consulting a static method of the class for this purpose. Therefore a static function (signalCallBack) is taken up to the class, which can be executed with the arrival of the signal. Furthermore the class possesses a static member (myPtr), that represents a pointer on the instance of the Scheduler (set by the constructor of the class). Over this pointer the static method accesses the instance of the scheduler and calls the function 'schedule', which executes all registered Callbacks successively.

The registration of objects also takes place over the static member, that points to the instance of the scheduler. The method registercallBack from the class ScheduledFct calls over this pointer the method registerCallBack from the scheduler. In order that ScheduledFct: :registerCallBack may access this member and the method of the class scheduler, the class ScheduledFct has to be declared as 'friend' of the class scheduler.

2.1.4 Modelling of the Time

The Modelling of time is made by a Timer-class.

2.1.4.1 Timer

The class timer was created in the context of the CoDEx project. The following requirements of the timer and the concept of the timer are literally taken from the system design of the system BAuOS belonging to the CoDEx project.

2.1.4.1.1 Requirements

The class Timer makes the current time available for the system. There is no reverting directly to an operating system service, because for test cases determined manipulations are to be made at the time of the control system. For that belongs starting at an arbitrary time, which not corresponds to the real time and the grub of time.

2.1.4.1.2 Concept

2.1.4.1.2.1 Class Timer

The Time service is represented by the class Timer.

```
class Timer {
  TimerInternalTimeType t0;
  // user given starting time of the control system
  TimerInternalTimeType t0real;
```

```
  // real world starting time of the control system
  int speedup;
  // speedup of control system in comparison with the real world
  TimerInternalTimeType getSystemTime ();

  // gets real world time from the operating system an converts it
  // to the requested type

public:
  Timer (void)
  Timer (Time t
  Timer (int speed
  Timer (Time t, int speed)

 BAuSysTimeType getActualTime ();
```

Instances

In the system there is an instance of this class..

Use

The result, which the function getActualTime sends, is calculated from the formula:

```
t0 + (getSystemTime () - t0real) * speedup
```

That means the elapsed real time since the start of the control system (GET system time () -t0real) is multiplied by the acceleration factor. Thus you receive the time interval, which is to have run off for the control system meanwhile. Finally this is added on the given point of starting time of the system. Thus you receive the current time for the control system.

The different constructors of the class are present, in order to set if necessary default values automatically.

2.1.5 User Interface

Setting the values by the user is made by a user interface (control panel). This is realized by an autonomous subsystem. Communication between user interface and system is made by the methods of the room, which serve for setting the values, like TimeIntervalEntrance(ZeitspanneEintritt), TimeIntervalLeaving(ZeitspanneVerlassen),TimeIntervalHeating(ZeitspanneAufheizen),PresenceTem perature(AnwesenheitsTemperatur)and AbsenceTemperature(AbwesenheitsTemperatur). One assumes each room possesses its own control panel(User-Interface).

2.2 UML-Concept

In the following the modifications are described briefly for the classes of the concept in relation to the analysis. Afterwards the attributes and methods of the classes are described with the help of the class diagram and the Classtables. Finally the sequence diagrams, the collaboration diagrams, as well as the state diagrams are specified..

2.2.1 Short description of the differences to the analysis

In the following sections the modifications are described for each class, in relation to the analysis arose.

2.2.1.1 class House

The class *RainChecker* was taken over by the class *House*. That access to the outside temperature sensor shall only run over the class *House*. Thus a method is necessary, which passes the outside temperature on at all sections and rooms (there is needed to calculate holding and heating temperature). The class *House* is derived due to its active behavior of the class *ScheduledFct*.

2.2.1.2 class PRTempController

The class *PRTempController* is derived due to its active behavior of the class ScheduledFct.

2.2.1.3 class PresenceController

The class *PresenceConroller* is derived due to its active behavior of the class *ScheduledFct*t.

2.2.1.4 class Room

The class *Room* needs a further "Hand-Over function" for the outside temperature.

2.2.1.5 class RoomtemperatureController

The class *RoomtemperatureController* informs the class *TemperatureRegulator* only on change of the room temperature. Due to its active behavior the class *RoomtemperatureController* is derived from the class *ScheduledFct*.

2.2.1.6 class ScheduledFct

Newly the base class *ScheduledFct* was added. It represents the functionality described in section 2.1.3.1.2.1.

2.2.1.7 class Scheduler

The class *Scheduler* is responsible for the periodic activation of different methods. Its functionality was described in section 2.1.3.1.2.2.

2.2.1.8 class Section

The class *Section* did not change in relation to the analysis.

2.2.1.9 class Temperatureregulator

In the class *Temperatureregulator* several methods, which belong together logically, were replaced by only one (e.g. SomeoneThere() [JemandDa()] and NobodyThere() [NiemandDa()] with PresenceSet() [AnwesenheitSetzen()]). Were added also appropriate attributes, so that only a change of a value still have to be assigned.
For the calculation of the heating temperatures the outside temperature and the room temperature are needed. New attributes and methods in the class *Temperatureregulator* set and store these temperatures.
The class *Temperatureregulator* is derived due to its active behavior of the class *ScheduledFct*.

2.2.1.10 class Timer

The class *Timer* represents the class described in section 2.1.4.

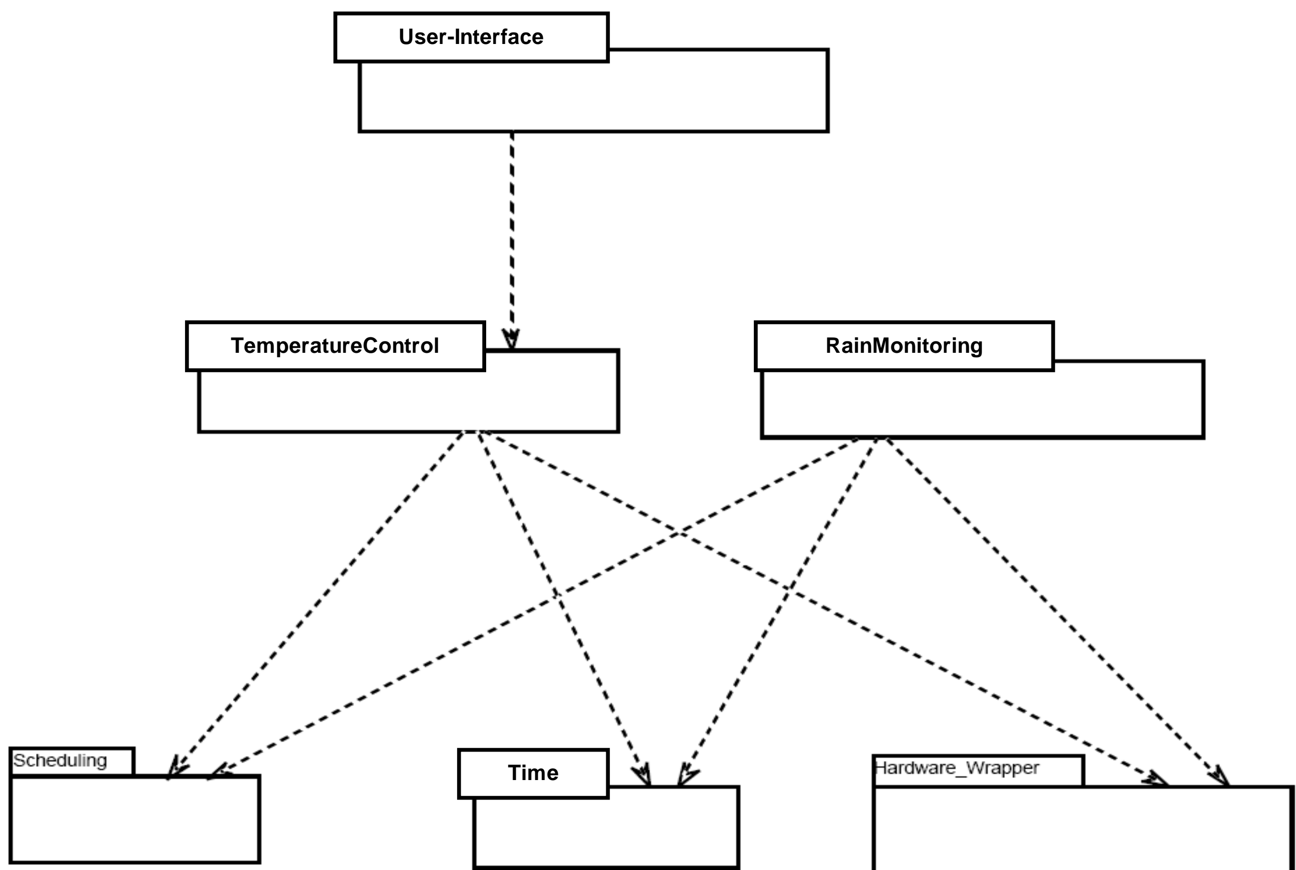
2.2.1.11 class Window

The report of a changed window condition is not given any longer directly to the *Temperatureregulator*, but it runs over the class *Room*. This is necessary, since otherwise no Instance sequence for the classes *Window* and *Temperatureregulator* can be specified. The class *Window* is derived due to its active behavior of the class *ScheduledFct*.

2.2.2 UML class diagrams

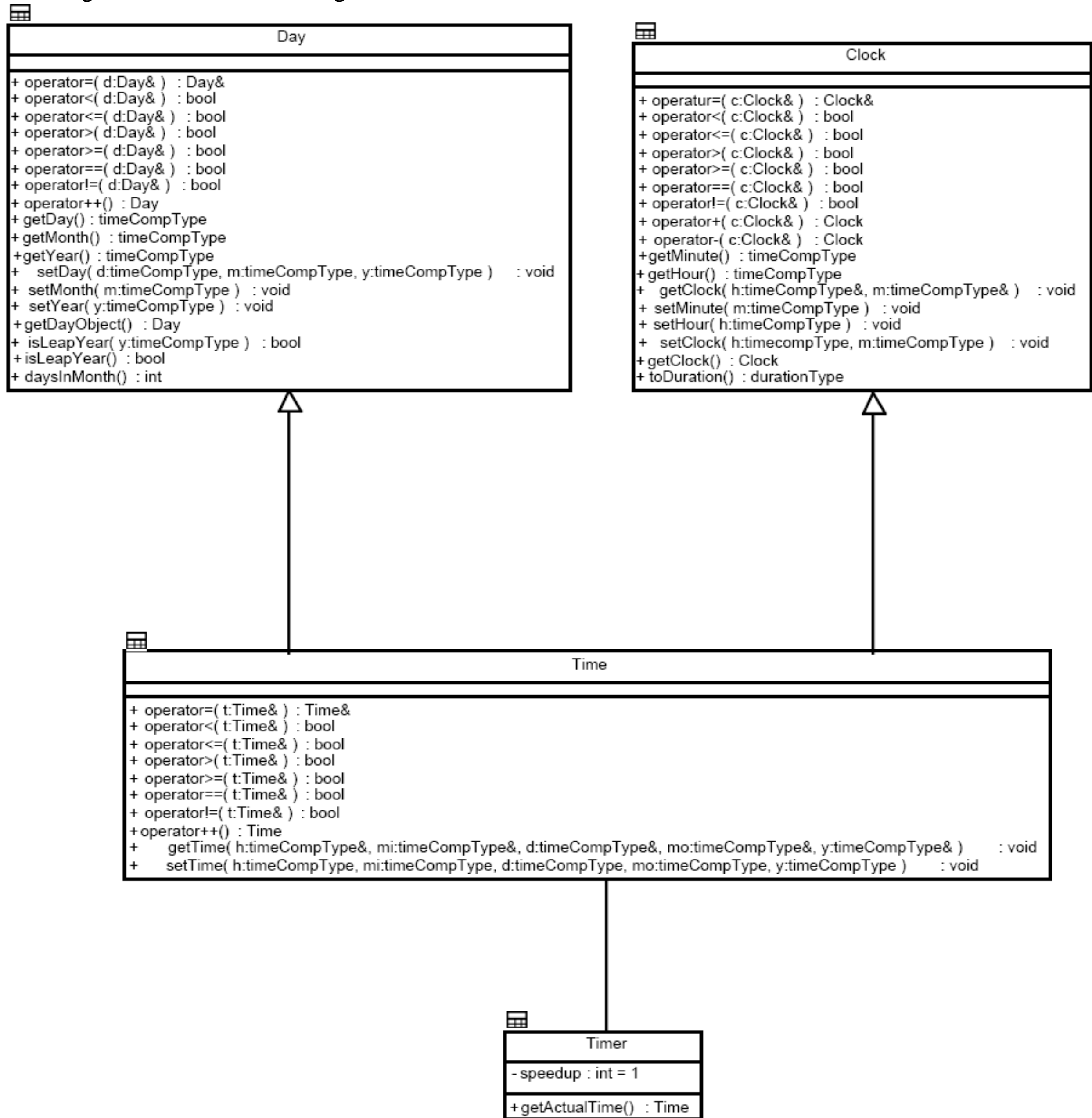
2.2.2.1 Package structure

This diagram shows the allocation of the system into individual subsystems (Packages). The individual packages shows a sighting to the individual classes. Therefore not necessarily all methods and attributes of a class are defined in a package.



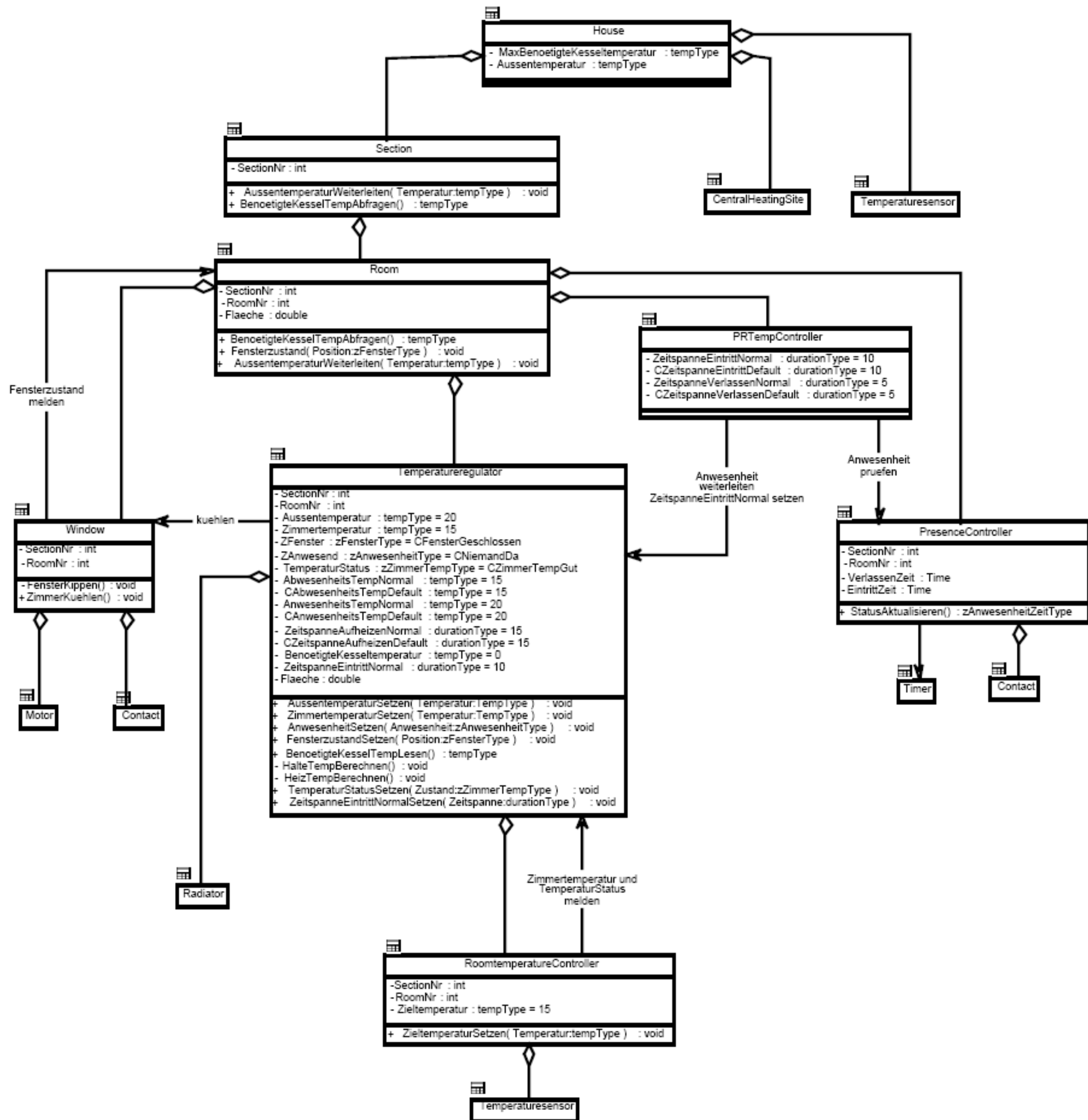
2.2.2.2 Time

This diagram shows the modeling of the time:



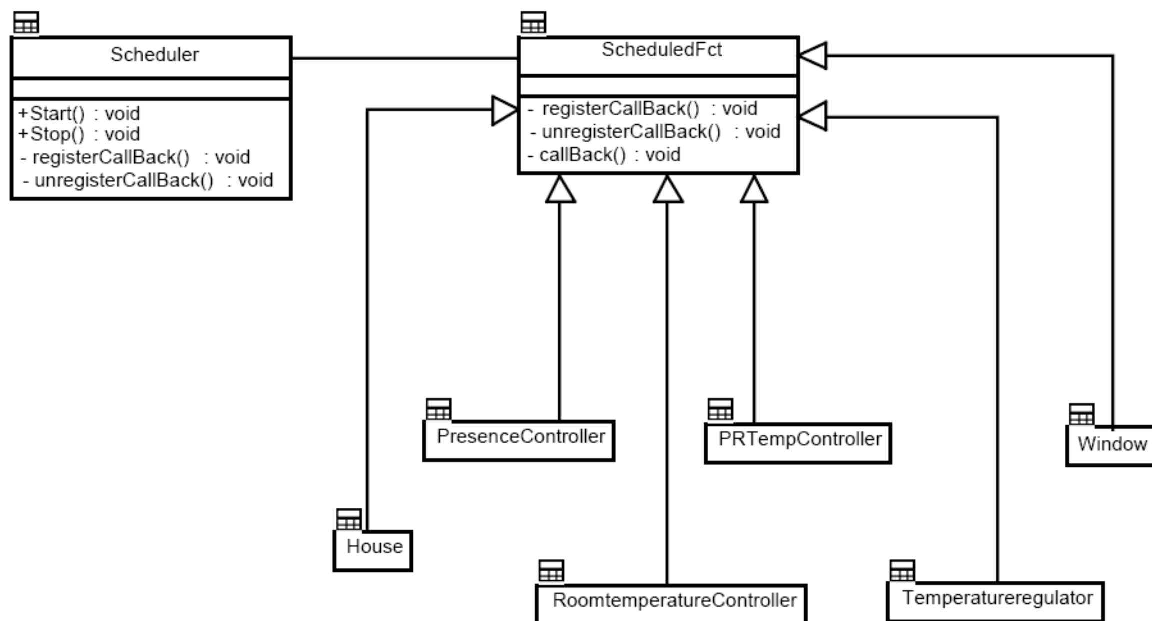
2.2.2.3 Temperature adjustment Control

This diagram shows all classes, which are involved in the temperature control.



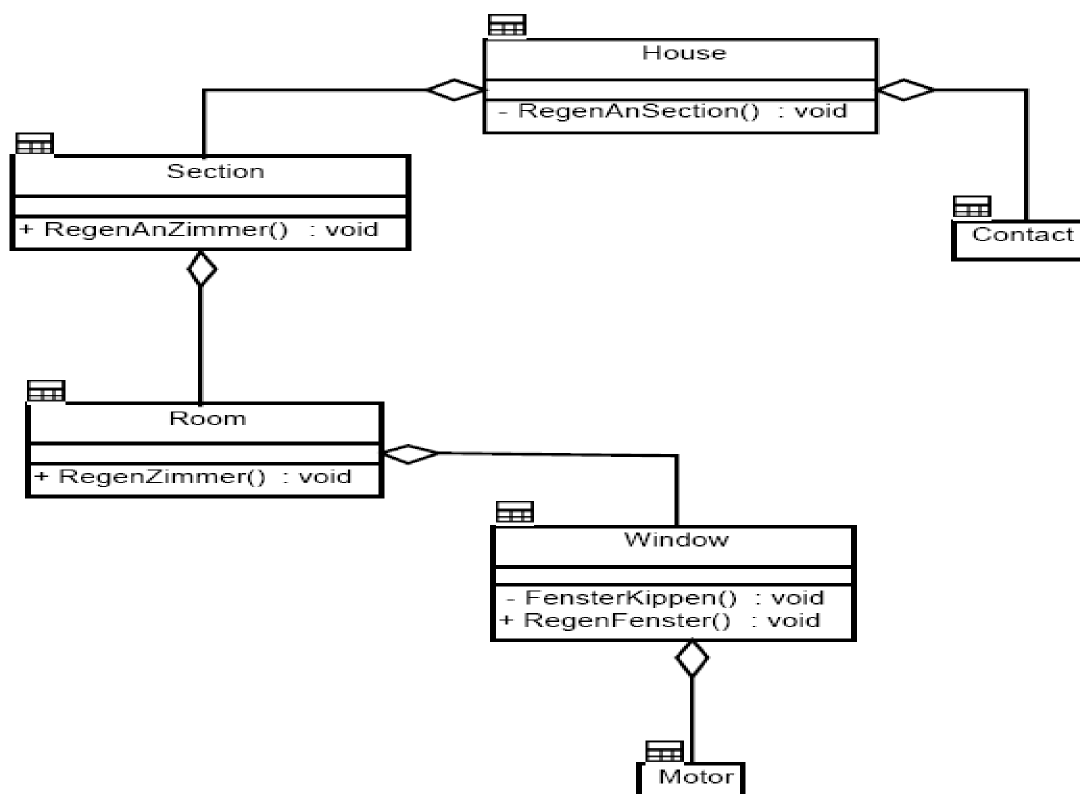
2.2.2.4 Scheduling

Active classes must be derived from the basis class ScheduledFct. The scheduler activates these then periodically.



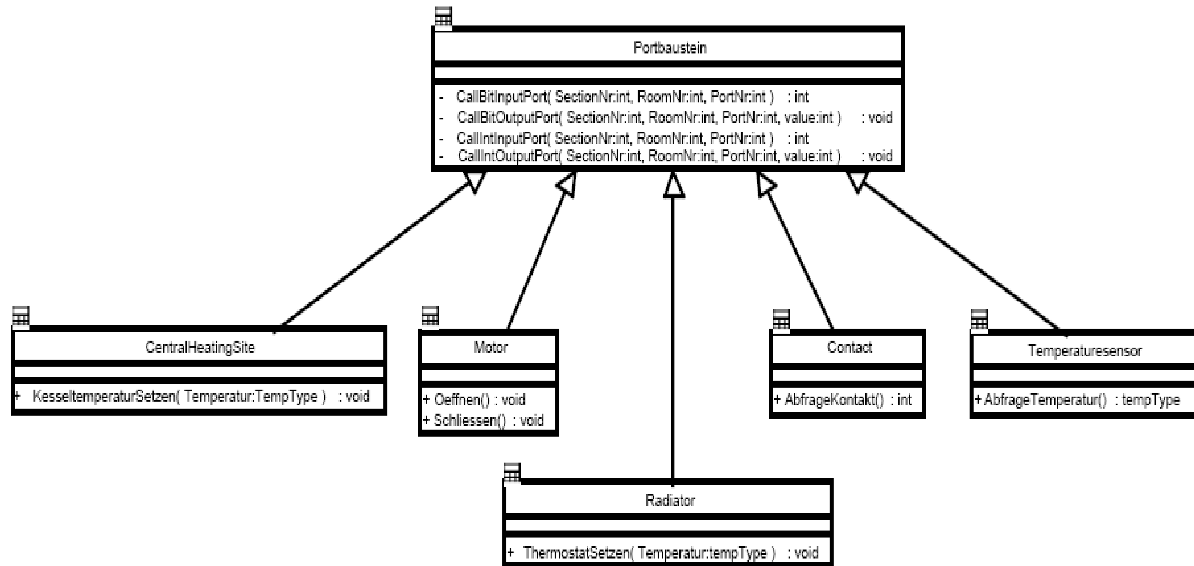
2.2.2.5 Rain Monitoring

This diagram shows all classes, which are involved in the rain monitoring.



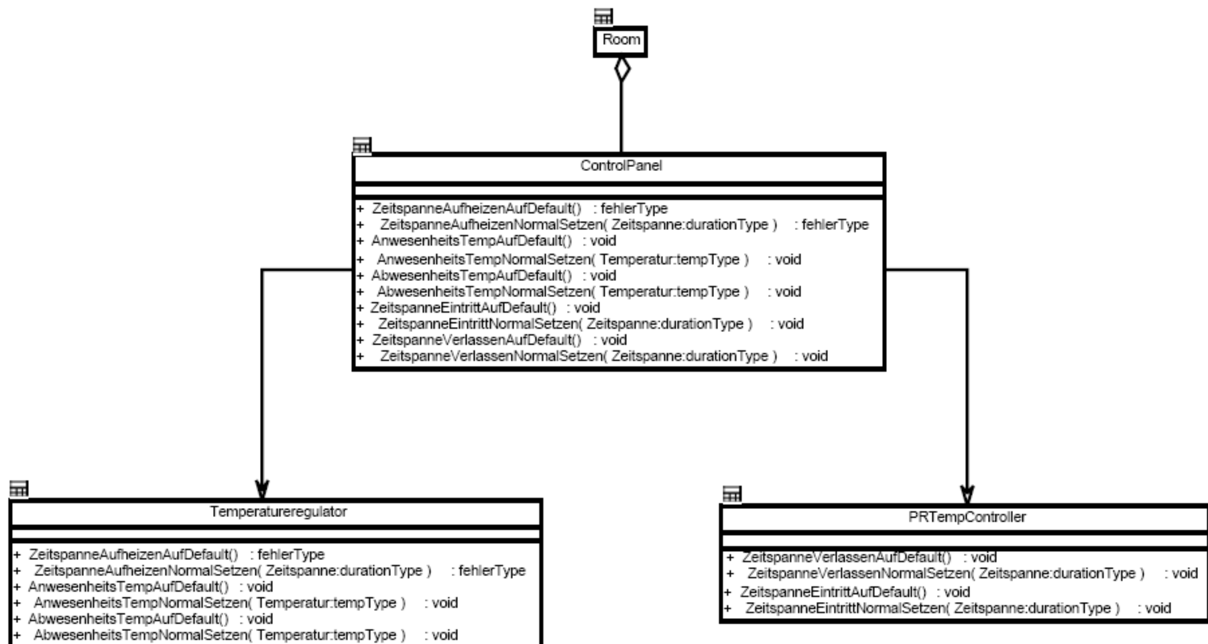
2.2.2.6 Hardware_Wrapper

This diagram shows all classes, which represent sensors or actuators.



2.2.2.7 User Interface

This diagram shows the user interface subsystem



2.2.2.8 Constructors

This diagramm shows the constructors of the several classes.