Inhaltsverzeichnis

Kapitel 2

Software-Systemdokumentation

2.1 Einleitung

Dieses Dokument beschreibt die Systemarchitektur des Klimasteuerungssystems. Als Grundlage dient die Anforderungsbeschreibung aus Kapitel 1.3

Der Entwurf wird analog zur Analyse objektorientiert sein und unter Verwendung der OMT-Methode erstellt. Als Notation wird wieder die UML (Unified Modeling Language) verwendet.

2.1.1 Entwurfsentscheidungen

EE	Beschreibung der Entwurfsentscheidung
EE1	Das Zielsystem ist ein zentraler Steuerungsrechner mit einem Portbaustein, über den • alle Sensoren abgefragt • alle Aktuatoren angesteuert werden können.
EE2	Parallelität wird durch Verwendung eines einfachen Schedulers simuliert.
EE3	Die Zeit wird durch eine wiederverwendbare Komponente Timer realisiert

Tabelle 1: Entwurfsentscheidungen

2.1.2 Portbausteine

Alle Sensoren und Aktoren werden über Portbausteine angesteuert. Jeder Typ von Aktoren und Sensoren besitzt einen eigenen Baustein mit einer ausreichenden Anzahl von Anschlüssen. Zur Ansteuerung liefert der Hardwarehersteller bereits vorgefertigte Libraries mit. Deshalb werden diese Bausteine im Objektmodell als eine Klasse realisiert

Es gibt folgende Arten von Eingabe- und Ausgabeports:

1. *Bit-Eingabeports* für Kontakte.
   Sie werden über int CallBitInputPort(int LineNr1, int LineNr2, int PortNr) abgefragt.
2. *Integer-Eingabeports* für Temperaturen.
   Sie werden über int CallIntInputPort(int LineNr1, int LineNr2, int PortNr) abgefragt.
3. *Bit-Ausgabeports* für einfache Aktoren.
   Sie werden über int CallBitOutputPort(int LineNr1, int LineNr2, int PortNr, int value) angesteuert.
4. *Integer-Ausgabeports* für komplexere Aktoren.
   Sie werden über int CallIntOutputPort(int LineNr1, int LineNr2, int PortNr, int value) angesteuert.

Die im Haus vorhandenen Aktoren und Sensoren werden dabei folgendermaßen zugeordnet:

- Bit-Eingabeports für Kontakte: Anwesenheitskontakt, Kippkontakt, Offenkontakt, Regenkontakt
- Integer-Eingabeports für Temperaturen: Außen-, Zimmertemperatur-, Heizkörpertemperaturfühler
- Bit-Ausgabeports für einfache Aktoren: Schwenk- und Kippmotoren der Fenster
- Integer-Ausgabeports für komplexere Aktoren: Kesseltemperatur, Heizkörperventil

Es werden folgende Portzuordnungen festgelegt:

Port	PortNr	Typ des Ports	LineNr1	LineNr2
Zimmertemperaturfühler	0	IntInput	Section-Nr	Zimmer-Nr
Anwesenheitskontakt	1	BitInput	Section-Nr	Zimmer-Nr
Heizkörpertemperaturfühler	2	IntInput	Section-Nr	Zimmer-Nr
Heizkörperventil	3	IntOutput	Section-Nr	Zimmer-Nr
Kippkontakt	4	BitInput	Section-Nr	Zimmer-Nr
Offenkontakt	5	BitInput	Section-Nr	Zimmer-Nr
Kippmotor	6	BitOutput	Section-Nr	Zimmer-Nr
Schwenkmotor	7	BitOutput	Section-Nr	Zimmer-Nr
Kesseltemperatur	8	IntOutput	0	0
Außentemperaturfühler	9	IntInput	0	0

Tabelle 2: Portzuordnung

Port	PortNr	Typ des Ports	LineNr1	LineNr2
Regenkontakt	10	BitInput	0	0

Tabelle 2: Portzuordnung

2.1.3 Parallelität

Die bisherigen Beschreibungen der Klassen gehen von einer inhärenten Parallelität von Klassen aus. Eine parallele Implementierung der Klassen wäre auf dem Zielrechner (bzw. auf dessen Betriebssystem) zwar möglich, dies würde die Behandlung von Kommunikations- und Synchronisationsprobleme erforderlich machen. Um die dadurch hinzukommende Komplexität zu vermeiden, wird die parallele Abarbeitung simuliert. Dies wird durch einen einfachen Scheduler realisiert, der bestimmte Methoden der Objekte nacheinander periodisch aktiviert.

2.1.3.1 Scheduler

Die Klasse Scheduler wurde im Rahmen des CoDEx-Projektes entworfen. Die nachfolgenden Anforderungen an den Scheduler und der Entwurf des Schedulers sind wörtlich aus dem Systementwurf des zum CoDEx-Projekt gehörenden Systems BAuOS entnommen.

2.1.3.1.1 Anforderungen

Der Entwurf enthält eine Reihe von Automaten, die parallel und unabhängig voneinander arbeiten. Diese Automaten sollen in der Implementierung quasi-parallel arbeiten. Die Realisierung soll so aussehen, daß eine bestimmte Methode eines Objekts in regelmäßigen Abständen ausgeführt wird. In dieser Methode wird das Verhalten des zu dem Objekt gehörigen Automaten nachgebildet. Für die periodische Aktivierung wird ein primitiver Scheduler benötigt, bei dem die auszuführenden Methoden registriert werden müssen.

2.1.3.1.2 Entwurf

Zur Realisierung der Anforderungen sieht der Entwurf zwei Klassen vor. Zum einen eine Basisklasse, von der alle Objekte abgeleitet werden müssen. Diese Basisklasse stellt eine virtuelle Funktion zur Verfügung, die in den abgeleiteten Klassen überladen werden muß und das Verhalten des zugehörigen Automaten nachbildet. Die zweite Klasse realisiert den Scheduler. Die periodische Aktivierung wird mit Hilfe des UNIX Signal-Mechanismus implementiert.

2.1.3.1.2.1 Die Basisklasse ScheduledFct

Jede Klasse, die eine Funktion hat, die periodisch aktiviert werden soll, muß von der Klasse `ScheduledFct` abgeleitet werden (der Name steht für 'Scheduled Function').

```
class ScheduledFct {

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

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

Diese Klasse verfügt über zwei Methoden. Die virtuell Methode `callBack` muß in abgeleiteten Klassen überdefiniert werden und realisiert diejenige Funktion, die periodisch aufgerufen werden kann. Wenn mehrere Funktionen periodisch ausgeführt werden sollen, dann müssen diese durch die Funktion `callBack` aufgerufen werden.

Die Methode `registerCallBack` sorgt dafür, daß das Objekt beim Scheduler für die periodische Aktivierung registriert wird. Diese Methode kann vom Objekt selber oder (wenn sie public vererbt wird) von außen aufgerufen werden. Ein Annullieren der Registrierung geschieht durch den Aufruf der Funktion `unregisterCallBack`, durch die die periodische Aktivierung wieder gestoppt werden kann. Die Implementierung der Methoden `registerCallBack` und `unregisterCallBack` wird im Abschnitt 2.1.3.1.2 beschrieben.

2.1.3.1.2.2 Die Klasse Scheduler

Der Scheduler wird durch die Klasse `Scheduler` realisiert:

```
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);
};
```

Instanzen

Im Kontrollsystem muß eine Instanz dieser Klasse existieren. Es darf nicht mehr als eine Instanz angelegt werden.

Benutzung

Der Konstruktor der Klasse `Scheduler` erwartet einen Parameter, der angibt in welchen Abständen die periodische Aktivierung stattfinden soll. Dieses Intervall wird in Mikrosekunden spezifiziert. Nach dem Erzeugen einer Instanz der Klasse `Scheduler` können sich Objekte, die von der Klasse `ScheduledFct` abgeleitet worden sind, bei dieser Instanz registrieren (Abschnitt 2.1.3.1.2.1).

Durch den Aufruf der Methode `start` beginnt der Scheduler mit der periodischen Aktivierung aller registrierten Objekte. Die Registrierung weiterer Objekte ist danach immer noch möglich. Durch den Aufruf der Methode `stop` kann die Arbeit des Schedulers unterbrochen und ggf. mit `start` wieder aufgenommen werden. Einen Einfluß auf die registrierten Objekte hat das nicht.

Implementierung

Die periodische Aktivierung von Methoden wird dadurch realisiert, daß regelmäßig ein UNIX Signal (`SIGALRM`) erzeugt und abgefangen wird. Im Zuge der Behandlung des Signals werden dann die Methoden ausgeführt.

UNIX ruft beim Eintreffen eines Signals eine Funktion auf, die zuvor als Signal-Handler eingetragen worden sein muß. Für diese Aufgabe kann nicht einfach eine Methode einer Klasse verwendet werden. Methoden erwarten beim Aufruf nämlich immer einen Zeiger auf das zugehörige Objekt (`this`-Pointer). Dieser Pointer wird beim Aufruf als Signal-Handler durch das Betriebssystem nicht übergeben. Das Problem kann dadurch umgangen werden, daß man für diesem Zweck eine statische Methode der Klasse heranzieht. Daher wird eine statische Funktion (`signalCallBack`) in die Klasse aufgenommen, die beim Eintreffen des Signals ausgeführt werden kann. Des weiteren verfügt die Klasse über ein statisches Mitglied (`myPtr`), das einen Pointer auf die Instanz des Scheduler's darstellt (wird durch den Konstruktor der Klasse gesetzt). Über diesen Pointer greift die statische Methode auf die Instanz des Schedulers zu und ruft die Funktion `schedule` auf, die alle registrierten Callbacks nacheinander ausführt.

Die Registrierung von Objekten geschieht ebenfalls über das statische Mitglied, das auf die Instanz des Schedulers zeigt. Die Methode `registerCallBack` der Klasse `ScheduledFct` ruft über diesen Zeiger die Methode `registerCallBack` des Schedulers auf. Damit `ScheduledFct::registerCallBack` auf das dieses Mitglied und die Methode der Klasse `Scheduler` zugreifen darf, muß die Klasse `ScheduledFct` als Friend der Klasse `Scheduler` deklariert werden.

2.1.4 Modellierung der Zeit

Die Modellierung der Zeit wird mit Hilfe einer Timer-Klasse vorgenommen.

2.1.4.1 Timer

Die Klasse Timer wurde im Rahmen des CoDEx-Projektes entworfen. Die nachfolgenden Anforderungen an den Timer und der Entwurf des Timers sind wörtlich aus dem Systementwurf des zum CoDEx-Projekt gehörenden Systems BAuOS entnommen.

2.1.4.1.1 Anforderungen

Die Klasse Timer stellt die aktuelle Zeit für das System zur Verfügung. Es wird nicht direkt auf einen Betriebssystemdienst zurückgegriffen, da zu Testzwecken bestimmte Manipulationen an der Zeit des Kontrollsystems vorgenommen werden sollen. Dazu gehören das Starten zu einem beliebigen Zeitpunkt, der nicht der realen Zeit entspricht und das Raffen der Zeit.

2.1.4.1.2 Entwurf

2.1.4.1.2.1 Die Klasse Timer

Der Zeitdienst wird durch die Klasse `Timer` realisiert.

```
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 ();
};
```

Instanzen

Im System gibt es eine Instanz dieser Klasse.

Benutzung

Das Ergebnis, das die Funktion `getActualTime` liefert wird nach der Formel

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

berechnet. D.h. die seit dem Start des Kontrollsystems abgelaufene Realzeit (`getSystemTime () - t0real`) wird mit den Beschleunigungsfaktor multipliziert. Dadurch erhält man die Zeitspanne, die für das Kontrollsystem währenddessen abgelaufen sein soll. Schließlich wird dies auf den vorgegebenen Startzeitpunkt des Systems aufaddiert. Damit erhält man die aktuelle Zeit für das Kontrollsystem.

Die verschiedenen Konstruktoren der Klasse sind vorhanden, um ggf. default-Werte automatisch zu setzen.

2.1.5 Benutzungsschnittstelle

Das Setzen der Werte durch den Benutzer erfolgt über eine Benutzungsschnittstelle (Control Panel). Diese wird durch ein autonomes Subsystem realisiert. Die Kommunikation zwischen Benutzungsschnittstelle und System erfolgt über die Methoden des Zimmers, die zum Setzen der Werte, wie ZeitspanneEintritt, ZeitspanneVerlassen, ZeitspanneAufheizen, AnwesenheitsTemperatur und AbwesenheitsTemperatur, dienen. Es wird davon ausgegangen, daß jedes Zimmers ein eigenes Control Panel besitzt.

2.2 UML-Entwurf

Nachfolgend werden für die Klassen des Entwurfs die Änderungen gegenüber der Analyse kurz beschrieben. Im Anschluß werden die Attribute und Methoden der Klassen mit Hilfe des Klassendiagramms und den Classtables erläutert. Abschließend werden die Sequenzdiagramme, die Kollaborationsdiagrammen, sowie die Statediagramme aufgeführt.

2.2.1 Kurzbeschreibung der Unterschiede zur Analyse

In den folgenden Abschnitten werden für jede Klasse die Änderungen beschrieben, die sich gegenüber der Analyse ergeben haben.

2.2.1.1 Klasse House

Die Klasse *RainChecker* wurde von der Klasse *House* übernommen.
Der Zugriff auf den Aussentemperaturfuehler soll nur noch über die Klasse *House* laufen. Somit ist eine Methode erforderlich, die die Aussentemperatur an alle Abschnitte und Zimmer weiterleitet (sie wird dort zu Halte- und Heiz-Temperaturberechnung benötigt).
Die Klasse *House* wird aufgrund ihres aktiven Verhaltens von der Klasse *ScheduledFct* abgeleitet.

2.2.1.2 Klasse PRTempController

Die Klasse *PRTempController* wird aufgrund ihres aktiven Verhaltens von der Klasse *ScheduledFct* abgeleitet.

2.2.1.3 Klasse PresenceController

Die Klasse *PresenceConroller* wird aufgrund ihres aktiven Verhaltens von der Klasse *ScheduledFct* abgeleitet.

2.2.1.4 Klasse Room

Die Klasse *Room* benötigt eine weitere „Durchreichfunktion“ für die Aussentemperatur.

2.2.1.5 Klasse RoomtemperatureController

Die Klasse *RoomtemperatureController* benachrichtigt nur noch bei Änderung der Zimmertemperatur die Klasse *Temperatureregulator*.
Aufgrund ihres aktiven Verhaltens wird die Klasse *RoomtemperatureController* von der Klasse *ScheduledFct* abgeleitet.

2.2.1.6 Klasse ScheduledFct

Neu hinzugekommen ist die Basisklasse *ScheduledFct*. Die repräsentiert die in Abschnitt 2.1.3.1.2.1 erläuterte Funktionalität.

2.2.1.7 Klasse Scheduler

Die Klasse *Scheduler* ist verantwortlich für die periodische Aktivierung verschiedener Methoden. Seine Funktionalität wurde in Abschnitt 2.1.3.1.2.2 beschrieben.

2.2.1.8 Klasse Section

Die Klasse *Section* hat sich gegenüber der Analyse nicht geändert.

2.2.1.9 Klasse Temperatureregulator

In der Klasse *Temperatureregulator* wurden mehrere Methoden, die logisch zusammengehören, durch eine einzige ersetzt (z.B. JemandDa() und NiemandDa() durch AnwesenheitSetzen()). Hinzugekommen sind auch entsprechende Attribute, so daß nur noch eine Änderung eines Wertes übertragen werden muß.
Zur Berechnung der Heiztemperaturen werden die Aussentemperatur und die Zimmertemperatur benötigt. Neue Attribute und Methoden in der Klasse *Temperatureregulator* setzen und speichern diese Temperaturen.
Die Klasse *Temperatureregulator* wird aufgrund ihres aktiven Verhaltens von der Klasse *ScheduledFct* abgeleitet.

2.2.1.10 Klasse Timer

Die Klasse *Timer* repräsentiert die in Abschnitt 2.1.4 beschriebene Klasse.

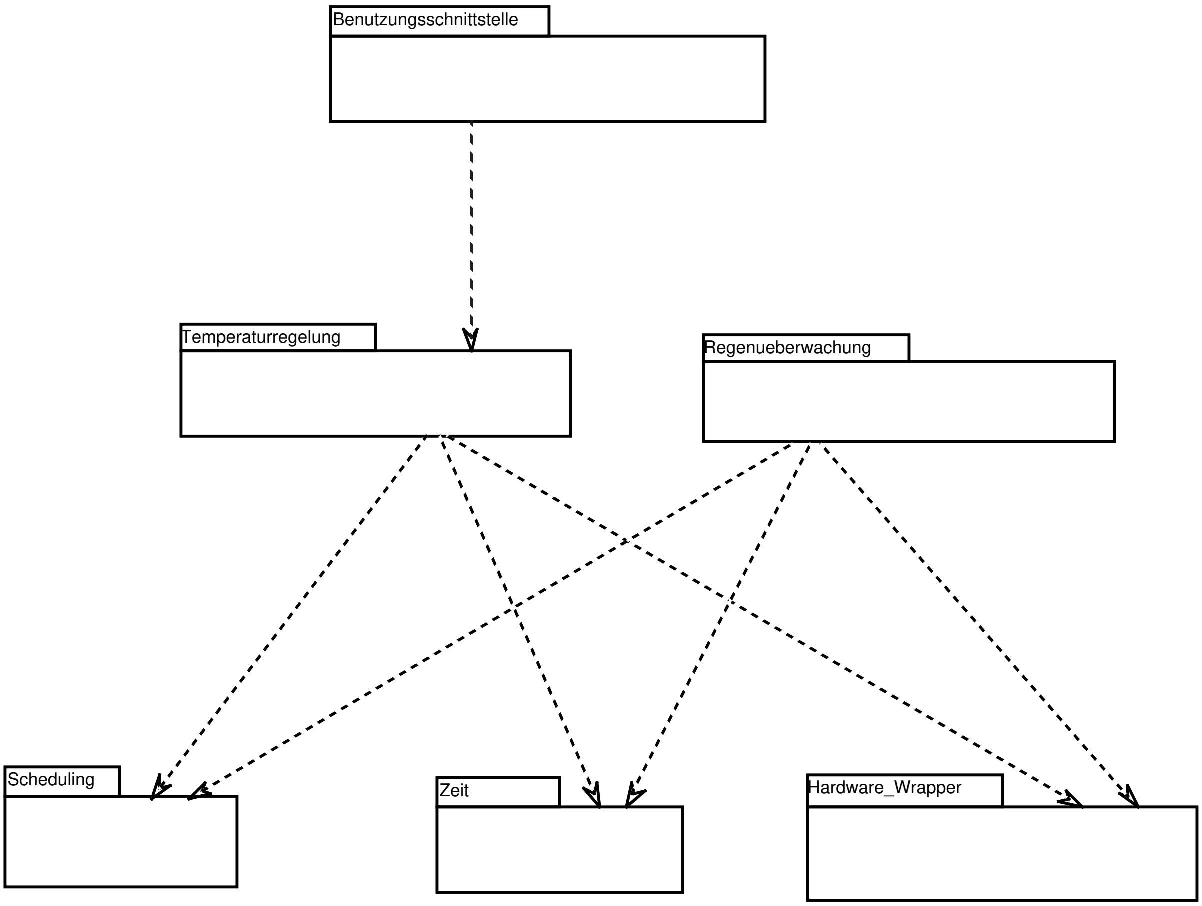
2.2.1.11 Klasse Window

Die Mitteilung eines veränderten Fensterzustands wird nicht mehr direkt an den Temperatureregulator gegeben, sondern sie läuft über die Klasse *Room*. Dies ist notwendig, da sonst keine Instanziierungsreihenfolge für die Klassen *Window* und *Temperatureregulator* festgelegt werden kann.
Die Klasse *Window* wird aufgrund ihres aktiven Verhaltens von der Klasse *ScheduledFct* abgeleitet.

2.2.2 UML-Klassendiagramme

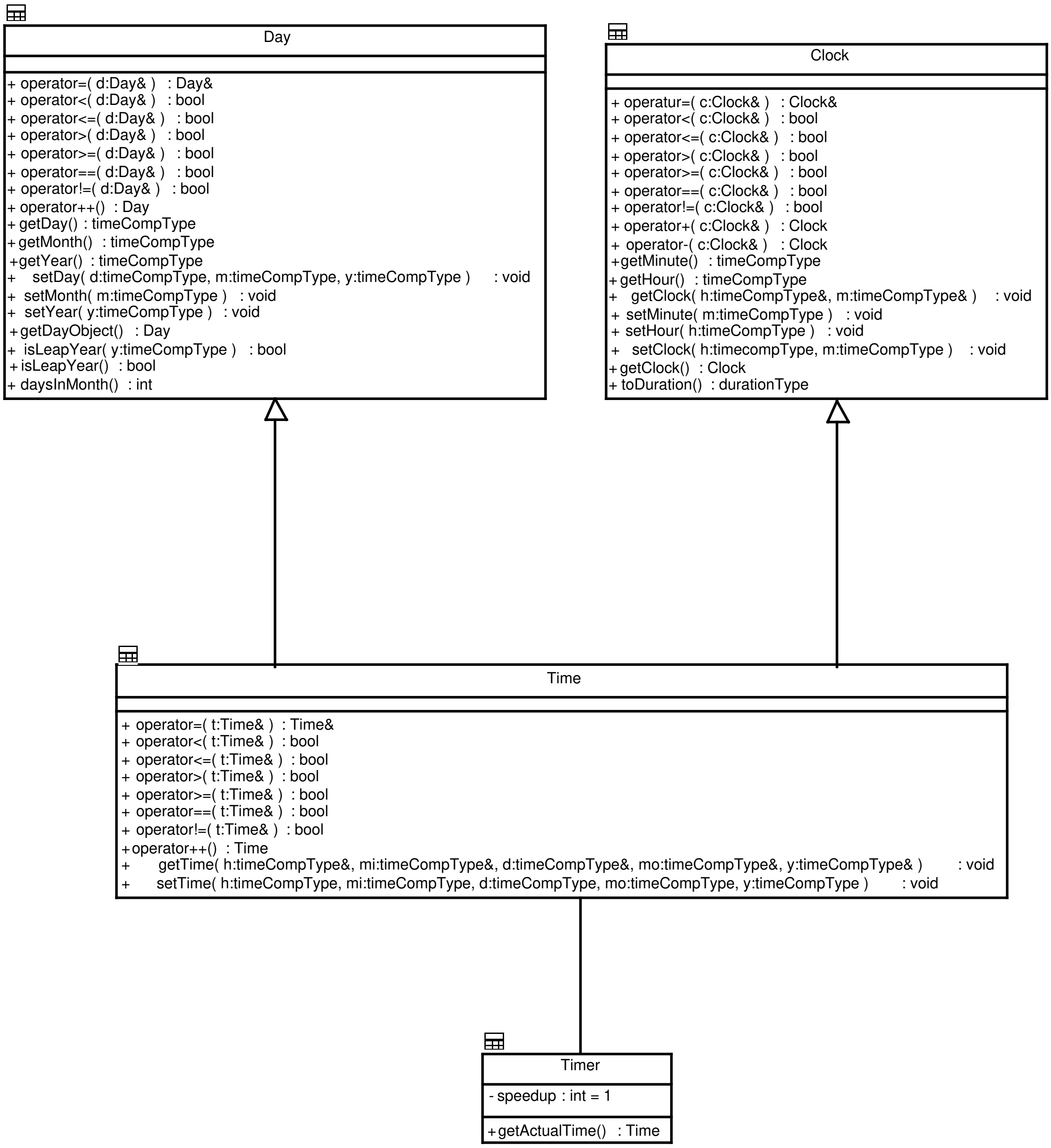
2.2.2.1 Packagestruktur

Dieses Diagramm zeigt die Aufteilung des Systems in einzelne Subsysteme (Packages). Die einzelnen Packages zeigen Sichten auf die einzelnen Klassen. Deshalb sind in einem Package nicht unbedingt alle Methoden und Attribute einer Klasse definiert.



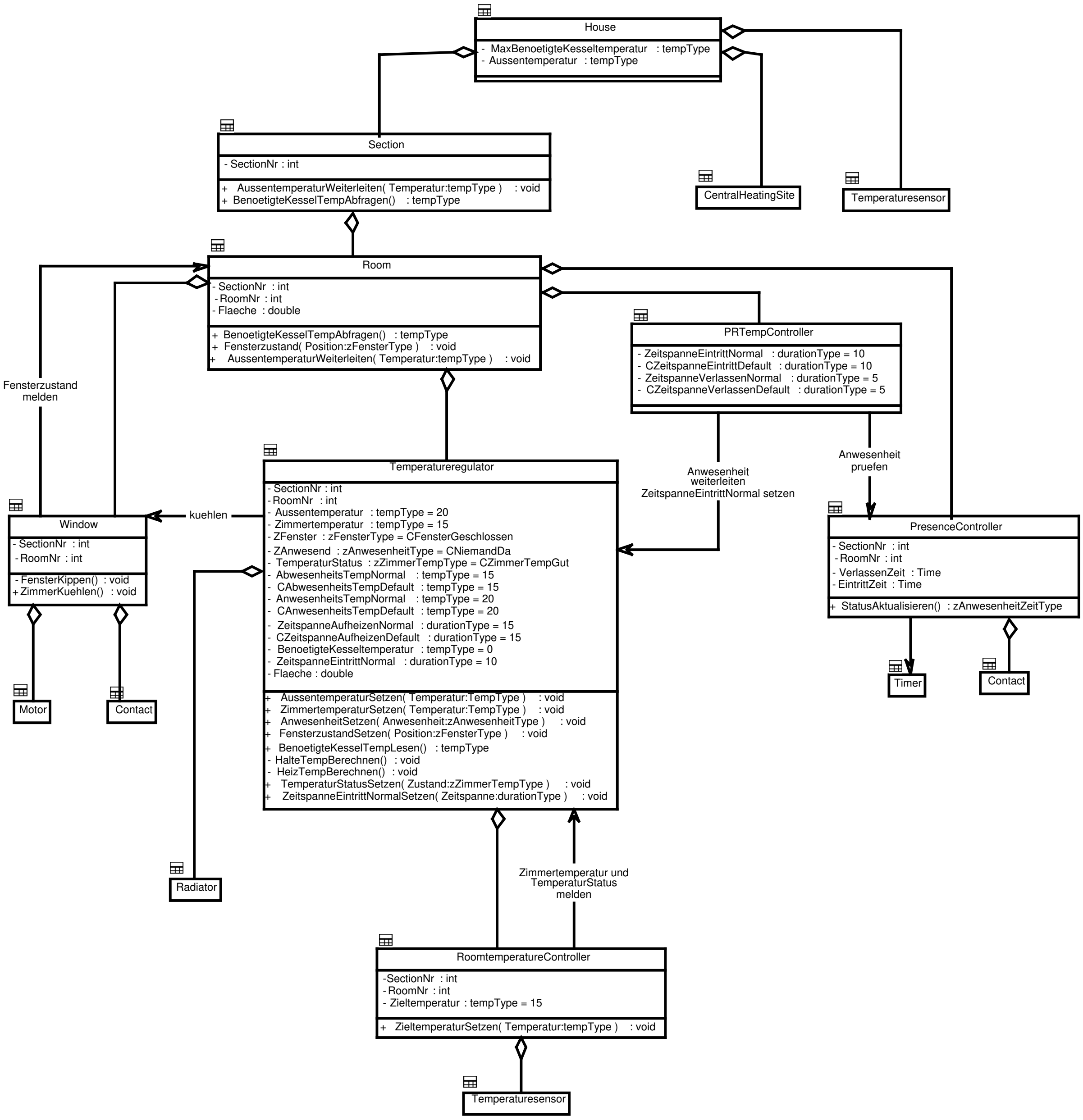
2.2.2.2 Zeit

Dieses Diagramm zeigt die Modellierung der Zeit.



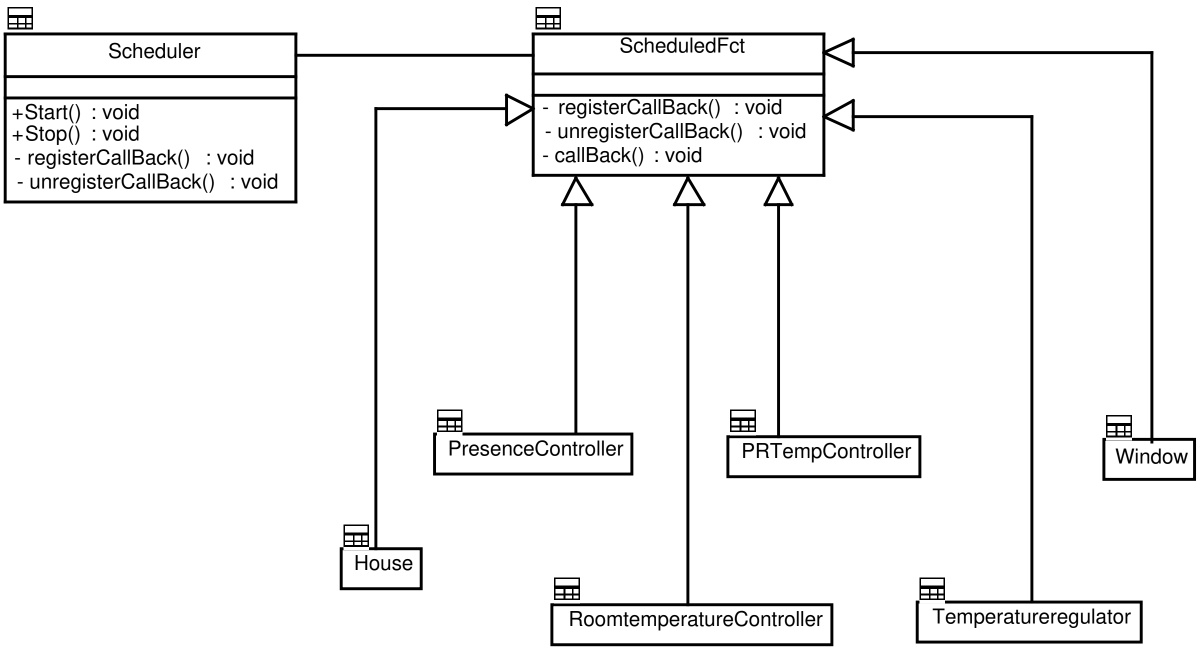
2.2.2.3 Temperaturregelung

Dieses Diagramm zeigt alle Klassen, die an der Temperaturregelung beteiligt sind.



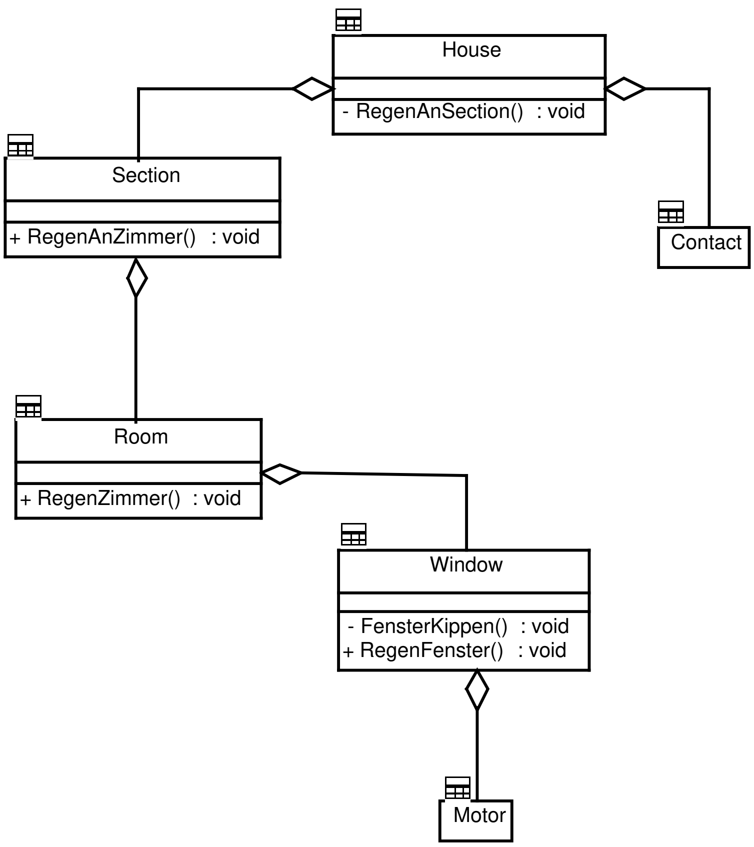
2.2.2.4 Scheduling

Aktive Klassen müssen von der Basisklasse ScheduledFct abgeleitet sein. Der Scheduler aktiviert diese dann periodisch.



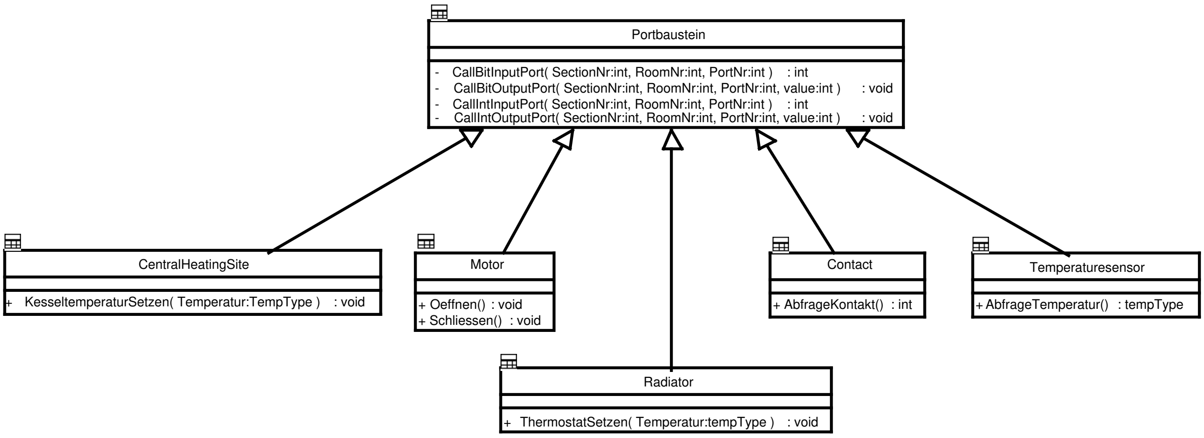
2.2.2.5 Regenueberwachung

Dieses Diagramm zeigt alle Klassen, die an der Regenüberwachung beteiligt sind.



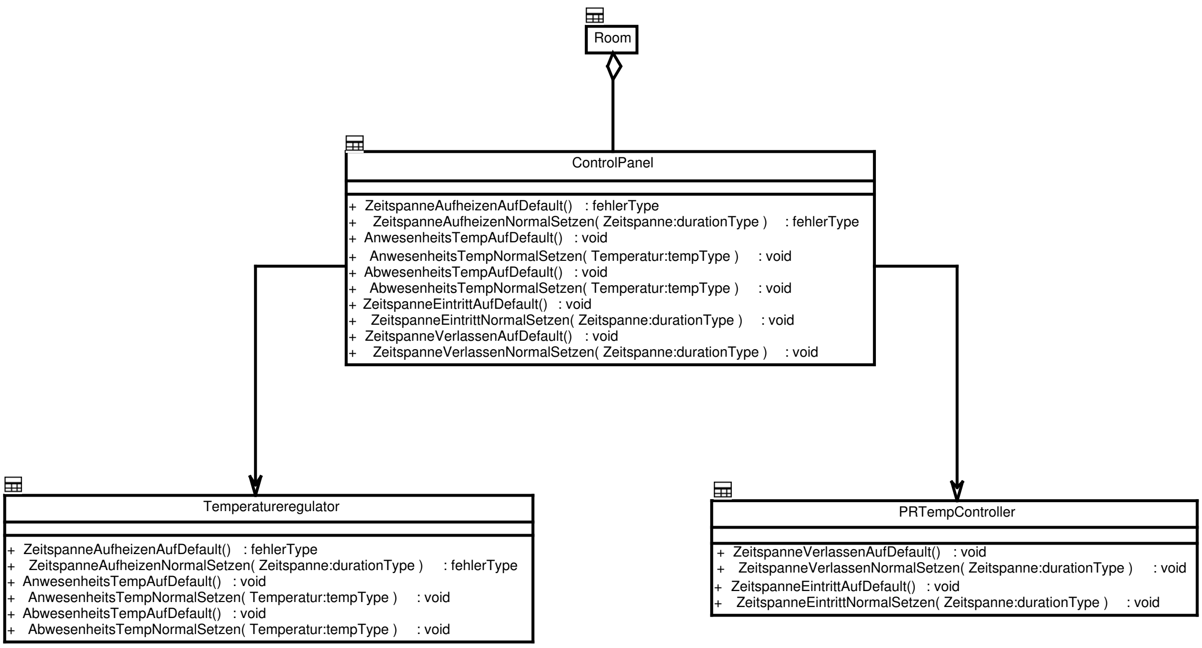
2.2.2.6 Hardware_Wrapper

Dieses Diagramm zeigt alle Klassen, die Sensoren oder Aktoren repräsentieren.



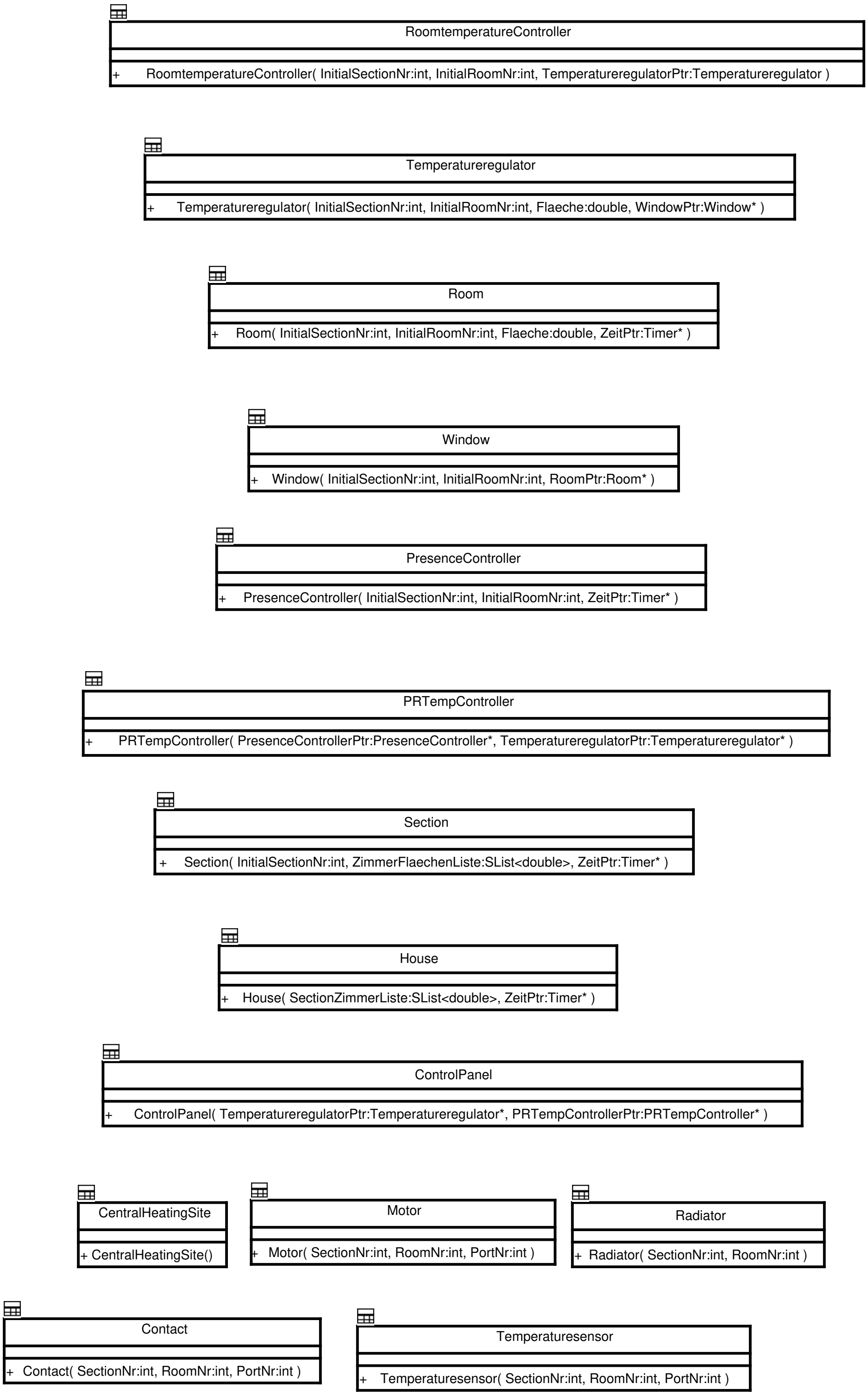
2.2.2.7 Benutzungsschnittstelle

Dieses Diagramm zeigt das Benutzungsschnittstellen-Subsystem



2.2.2.8 Konstruktoren

Dieses Diagramm zeigt die Konstuktoren der einzelnen Klassen.



2.2.3 UML-Classtables

Nachfolgend werden die Classtables der einzelnen Klassen aufgeführt, die die Attribute und Methoden der jeweiligen Klasse darstellen. Zur Erläuterung dieser Atrribute und Methoden befindet sich direkt im Anschluß an jedes Classtable eine Beschreibung der Attribute und Methoden.

2.2.3.1 Classtable der Klasse CentralHeatingSite

CentralHeatingSite		
Attribute	**Type**	**Default Value**
Operation	**Arguments**	**Return Type**
KesseltemperaturSetzen	Temperatur:TempType	void
CentralHeatingSite		
Inherited Class::Operation	**Arguments**	**Return Type**
Portbaustein::CallBitInputPort	SectionNr:int, RoomNr:int, PortNr:int	int
Portbaustein::CallBitOutputPort	SectionNr:int, RoomNr:int, PortNr:int, value:int	void
Portbaustein::CallIntInputPort	SectionNr:int, RoomNr:int, PortNr:int	int
Portbaustein::CallIntOutputPort	SectionNr:int, RoomNr:int, PortNr:int, value:int	void

2.2.3.1.1 Attribute der Klasse CentralHeatingSite

Keine.

2.2.3.1.2 Methodenschnittstellen der Klasse CentralHeatingSite

Signatur: **void KesseltemperaturSetzen(Temperatur:TempType)**

Beschreibung: Setzt die Temperatur des Heizkessels auf Temperatur ˚C. Verwendet hierfür die Methoden des Portbausteins.

2.2.3.2 Classtable der Klasse Clock

Clock		
Attribute	**Type**	**Default Value**
Operation	**Arguments**	**Return Type**
operatur=	c:Clock&	Clock&
operator<	c:Clock&	bool
operator<=	c:Clock&	bool
operator>	c:Clock&	bool
operator>=	c:Clock&	bool
operator==	c:Clock&	bool
operator!=	c:Clock&	bool
operator+	c:Clock&	Clock
operator-	c:Clock&	Clock
getMinute		timeCompType
getHour		timeCompType
getClock	h:timeCompType&, m:time-CompType&	void
setMinute	m:timeCompType	void
setHour	h:timeCompType	void
setClock	h:timecompType, m:timeCom-pType	void
getClock		Clock
toDuration		durationType

2.2.3.2.1 Attribute der Klasse Clock

Keine.

2.2.3.3 Classtable der Klasse Contact

Contact		
Attribute	**Type**	**Default Value**
Operation	**Arguments**	**Return Type**
AbfrageKontakt		int
Contact	SectionNr:int, RoomNr:int, PortNr:int	
Inherited Class::Operation	**Arguments**	**Return Type**
Portbaustein::CallBitInputPort	SectionNr:int, RoomNr:int, PortNr:int	int
Portbaustein::CallBitOutputPort	SectionNr:int, RoomNr:int, PortNr:int, value:int	void
Portbaustein::CallIntInputPort	SectionNr:int, RoomNr:int, PortNr:int	int
Portbaustein::CallIntOutputPort	SectionNr:int, RoomNr:int, PortNr:int, value:int	void

2.2.3.3.1 Attribute der Klasse Contact

Keine.

2.2.3.3.2 Methodenschnittstellen der Klasse Contact

Signatur: **int AbfrageKontakt()**

Beschreibung: Gibt den aktuellen Wert des Kontaktes zurück. Bentutzt hierfür die Methoden des Portbausteins.

2.2.3.4 Classtable der Klasse ControlPanel

ControlPanel		
Attribute	**Type**	**Default Value**

Operation	**Arguments**	**Return Type**
ZeitspanneAufheizenAufDefault		fehlerType
AnwesenheitsTempAufDefault		void
AbwesenheitsTempAufDefault		void
ZeitspanneAufheizenNormalSetzen	Zeitspanne:durationType	fehlerType
ControlPanel	Temperaturregula-torPtr:Temperaturregu-lator*, PRTempControl-lerPtr:PRTempControl-ler*	
AbwesenheitsTempNormalSetzen	Temperatur:tempType	void
AnwesenheitsTempNormalSetzen	Temperatur:tempType	void
ZeitspanneEintrittNormalSetzen	Zeitspanne:durationType	void
ZeitspanneEintrittAufDefault		void
ZeitspanneVerlassenAufDefault		void
ZeitspanneVerlassenNormalSetzen	Zeitspanne:durationType	void

2.2.3.4.1 Attribute der Klasse ControlPanel

Keine.

2.2.3.4.2 Methodenschnittstellen der Klasse ControlPanel

Signatur: **fehlerType ZeitspanneAufheizenAufDefault()**

Beschreibung: Ruft die entsprechende Methode des Temperaturregulators auf.

Signatur: **fehlerType ZeitspanneAufheizenNormalSetzen(Zeitspanne:durationType)**

Beschreibung: Ruft die entsprechende Methode des Temperaturregulators auf.

Signatur: **void AnwesenheitsTempAufDefault()**

Beschreibung: Ruft die entsprechende Methode des Temperaturregulators auf.

Signatur: **void AnwesenheitsTempNormalSetzen(Temperatur:tempType)**

Beschreibung: Ruft die entsprechende Methode des Temperaturregulators auf.

Signatur: **void AbwesenheitsTempAufDefault()**

Beschreibung: Ruft die entsprechende Methode des Temperaturregulators auf.

Signatur: **void AbwesenheitsTempNormalSetzen(Temperatur:tempType)**

Beschreibung: Ruft die entsprechende Methode des Temperaturregulators auf.

Signatur: **void ZeitspanneEintrittAufDefault()**

Beschreibung: Ruft die entsprechende Methode des PRTempController auf.

Signatur: **void ZeitspanneEintrittNormalSetzen(Zeitspanne:durationType)**

Beschreibung: Ruft die entsprechende Methode des PRTempController auf.

Signatur: **void ZeitspanneVerlassenAufDefault()**

Beschreibung: Ruft die entsprechende Methode des PRTempController auf.

Signatur: **void ZeitspanneVerlassenNormalSetzen(Zeitspanne:durationType)**

Beschreibung: Ruft die entsprechende Methode des PRTempController auf.

2.2.3.5 Classtable der Klasse Day

Day		
Attribute	**Type**	**Default Value**
Operation	**Arguments**	**Return Type**
operator=	d:Day&	Day&
operator<	d:Day&	bool
operator<=	d:Day&	bool
operator>	d:Day&	bool
operator>=	d:Day&	bool
operator==	d:Day&	bool

operator!=	d:Day&	bool
operator++		Day
getDay		timeCompType
getMonth		timeCompType
getYear		timeCompType
setDay	d:timeCompType, m:timeCompType, y:timeCompType	void
setMonth	m:timeCompType	void
setYear	y:timeCompType	void
getDayObject		Day
isLeapYear	y:timeCompType	bool
isLeapYear		bool
daysInMonth		int

2.2.3.5.1 Attribute der Klasse Day

Keine.

2.2.3.6 Classtable der Klasse House

House		
Attribute	**Type**	**Default Value**
MaxBenoetigteKesseltemperatur	tempType	
Aussentemperatur	tempType	
Operation	**Arguments**	**Return Type**
RegenAnSection		void
House	SectionZimmerListe:SList<double>, ZeitPtr:Timer*	
Inherited Class::Operation	**Arguments**	**Return Type**

ScheduledFct::registerCall-Back		void
ScheduledFct::unregisterCall-Back		void
ScheduledFct::callBack		void

2.2.3.6.1 Attribute der Klasse House

Attribut	Erläuterung
Aussentemperatur	Die momentane Aussentemperatur.
MaxBenoetigteKesseltempe-ratur	Die maximal benötigte Kesseltemperatur.

2.2.3.6.2 Methodenschnittstellen der Klasse House

Die Klasse House besitzt keine Methodenschnittstellen.

2.2.3.7 Classtable der Klasse Motor

Motor		
Attribute	**Type**	**Default Value**
Operation	**Arguments**	**Return Type**
Oeffnen		void
Schliessen		void
Motor	SectionNr:int, RoomNr:int, PortNr:int	
Inherited Class::Operation	**Arguments**	**Return Type**
Portbaustein::CallBitInputPort	SectionNr:int, RoomNr:int, PortNr:int	int
Portbaustein::CallBitOutputPort	SectionNr:int, RoomNr:int, PortNr:int, value:int	void

Portbaustein::CallIntInputPort	SectionNr:int, RoomNr:int, PortNr:int	int
Portbaustein::CallIntOutputPort	SectionNr:int, RoomNr:int, PortNr:int, value:int	void

2.2.3.7.1 Attribute der Klasse Motor

Keine.

2.2.3.7.2 Methodenschnittstellen der Klasse Motor

Signatur: **void Oeffnen()**

Beschreibung: Gibt den Befehl zum Öffnen des Motors an den Aktor. Benutzt hierfür die Methoden des Portbausteins.

Signatur: **void Schliessen()**

Beschreibung: Gibt den Befehl zum Schliessen des Motors an den Aktor. Benutzt hierfür die Methoden des Portbausteins.

2.2.3.8 Classtable der Klasse PRTempController

PRTempController		
Attribute	**Type**	**Default Value**
ZeitspanneEintrittNormal	durationType	10
ZeitspanneVerlassenNormal	durationType	5
CZeitspanneEintrittDefault	durationType	10
CZeitspanneVerlassenDefault	durationType	5
Operation	**Arguments**	**Return Type**
ZeitspanneVerlassenAufDefault		void
ZeitspanneVerlassenNormalSetzen	Zeitspanne:durationType	void
ZeitspanneEintrittAufDefault		void

ZeitspanneEintrittNormalSetzen	Zeitspanne:durationType	void
PRTempController	PresenceControllerPtr:PresenceController*, TemperatureregulatorPtr:Temperatureregulator*	
Inherited Class::Operation	**Arguments**	**Return Type**
ScheduledFct::registerCallBack		void
ScheduledFct::unregisterCallBack		void
ScheduledFct::callBack		void

2.2.3.8.1 Attribute der Klasse PRTempController

Attribut	Erläuterung
ZeitspanneEintrittNormal	Zeitspanne in Minuten, die angibt, wie lange sich eine Person im Zimmer befinden muß, bis das System regelnd eingreift.
ZeitspanneVerlassenNormal	Zeitspanne in Minuten, die angibt, wie lange die Temperatur AnwesenheitsTempNormal nach Verlassen aller Personen noch gehalten werden soll.
CZeitspanneEintrittDefault	Defaultwert für ZeitspanneEintritt in Minuten.
CZeitspanneVerlassenDefault	Defaultwert für ZeitspanneVerlassen in Minuten.

2.2.3.8.2 Methodenschnittstellen der Klasse PRTempController

Signatur: **void ZeitspanneVerlassenAufDefault()**

Beschreibung: Setzt das Attribut ZeitspanneVerlassenNormal auf den Defaultwert ZeitspanneVerlassenDefault.

Signatur: **void ZeitspanneVerlassenNormalSetzen(Zeitspanne:durationType)**

Beschreibung: Setzt das Attribut ZeitspanneVerlassenNormal auf Zeitspanne Minuten.

Signatur: **void ZeitspanneEintrittAufDefault()**

Beschreibung: Setzt das Attribut ZeitspanneEintrittNormal auf den Wert des Attributes Zeitspanne-EintrittDefault.

Signatur: **void ZeitspanneEintrittNormalSetzen(Zeitspanne:durationType)**

Beschreibung: Setzt das Attribut ZeitspanneEintrittNormal auf Zeitspanne Minuten.

2.2.3.9 Classtable der Klasse Portbaustein

Portbaustein		
Attribute	**Type**	**Default Value**
Operation	**Arguments**	**Return Type**
CallBitInputPort	SectionNr:int, RoomNr:int, PortNr:int	int
CallBitOutputPort	SectionNr:int, RoomNr:int, PortNr:int, value:int	void
CallIntInputPort	SectionNr:int, RoomNr:int, PortNr:int	int
CallIntOutputPort	SectionNr:int, RoomNr:int, PortNr:int, value:int	void

2.2.3.9.1 Attribute der Klasse Portbaustein

Keine.

2.2.3.9.2 Methodenschnittstellen der Klasse Portbaustein

2.2.3.10 Classtable der Klasse PresenceController

PresenceController		
Attribute	**Type**	**Default Value**
EintrittZeit	Time	
VerlassenZeit	Time	
SectionNr	int	

RoomNr	int	
Operation	**Arguments**	**Return Type**
StatusAktualisieren		zAnwesenheitZeitType
PresenceController	InitialSectionNr:int, InitialRoomNr:int, ZeitPtr:Timer*	
Inherited Class::Operation	**Arguments**	**Return Type**
ScheduledFct::registerCallBack		void
ScheduledFct::unregisterCallBack		void
ScheduledFct::callBack		void

2.2.3.10.1 Attribute der Klasse PresenceController

Attribut	Erläuterung
SectionNr	Nummer des Abschnitts, in dem sich der PresenceController befindet.
RoomNr	Nummer des Zimmers, in dem sich der PresenceController befindet.
EintrittZeit	Zeitpunkt, seit dem ein Benutzer anwesend ist.
VerlassenZeit	Zeitpunkt, seit dem der letzte Benutzer das Zimmer verlassen hat.

2.2.3.10.2 Methodenschnittstellen der Klasse PresenceController

Signatur: **zAnwesenheitZeitType StatusAktualisieren()**

Beschreibung: Gibt den Wert des Anwesenheitskontaktes zurück und die Zeitspanne, wie lange sich eine Person im Zimmer befindet bzw. das Zimmer leer ist.

2.2.3.11 Classtable der Klasse Radiator

Radiator		
Attribute	**Type**	**Default Value**

Operation	Arguments	Return Type
ThermostatSetzen	Temperatur:tempType	void
Radiator	SectionNr:int, RoomNr:int	
Inherited Class::Operation	**Arguments**	**Return Type**
Portbaustein::CallBitInputPort	SectionNr:int, RoomNr:int, PortNr:int	int
Portbaustein::CallBitOutputPort	SectionNr:int, RoomNr:int, PortNr:int, value:int	void
Portbaustein::CallIntInputPort	SectionNr:int, RoomNr:int, PortNr:int	int
Portbaustein::CallIntOutputPort	SectionNr:int, RoomNr:int, PortNr:int, value:int	void

2.2.3.11.1 Attribute der Klasse Radiator

Keine.

2.2.3.11.2 Methodenschnittstellen der Klasse Radiator

Signatur: **void ThermostatSetzen(Temperatur:tempType)**

Beschreibung: Setzt das Heizkörperthermostat auf Temperatur ˚C. Benutzt hierfür die Methoden des Portbausteins.

2.2.3.12 Classtable der Klasse Room

Room		
Attribute	**Type**	**Default Value**
SectionNr	int	
RoomNr	int	
Flaeche	double	
Operation	**Arguments**	**Return Type**

BenoetigteKesselTempAbfragen		tempType
RegenZimmer		void
Fensterzustand	Position:zFensterType	void
AussentemperaturWeiterleiten	Temperatur:tempType	void
Room	InitialSectionNr:int, InitialRoomNr:int, Flaeche:double, ZeitPtr:Timer*	

2.2.3.12.1 Attribute der Klasse Room

Attribut	Erläuterung
SectionNr	Nummer des Abschnitts, in dem sich das Zimmer befindet.
RoomNr	Nummer des Zimmers. Wird für die Aufrufe der Portbaustein-Methoden benötigt.
Flaeche	

2.2.3.12.2 Methodenschnittstellen der Klasse Room

Signatur: **void Fensterzustand(Position:zFensterType)**

Beschreibung: Leitet den Fensterzustand an den Temperatureregulator weiter.

Signatur: **void AussentemperaturWeiterleiten(Temperatur:tempType)**

Beschreibung: Leitet die Aussentemperatur an den Temperatureregulator weiter.

Signatur: **tempType BenoetigteKesselTempAbfragen()**

Beschreibung: Ruft die Methode BenoetigteKesselTempLesen() des Temperatureregulators auf und liefert als Ergebnis die benötigte Kesseltemperatur des Zimmers zurück.

Signatur: **void RegenZimmer()**

Beschreibung: Ruft RegenFenster() von Fenster auf.

2.2.3.13 Classtable der Klasse RoomtemperatureController

RoomtemperatureController		
Attribute	**Type**	**Default Value**
Zieltemperatur	tempType	15
SectionNr	int	
RoomNr	int	
Operation	**Arguments**	**Return Type**
ZieltemperaturSetzen	Temperatur:tempType	void
RoomtemperatureController	InitialSectionNr:int, InitialRoomNr:int, TemperaturereregulatorPtr:Temperatureregulator	
Inherited Class::Operation	**Arguments**	**Return Type**
ScheduledFct::registerCallBack		void
ScheduledFct::unregisterCallBack		void
ScheduledFct::callBack		void

2.2.3.13.1 Attribute der Klasse RoomtemperatureController

Attribut	Erläuterung
Zieltemperatur	Temperatur, die im Raum gehalten werden soll.
SectionNr	Nummer des Abschnitts, in dem sich der RoomtemperatureController befindet.
RoomNr	Nummer des Zimmers, in dem sich der RoomtemperatureController befindet.

2.2.3.13.2 Methodenschnittstellen der Klasse RoomtemperatureController

Signatur: **void ZieltemperaturSetzen(Temperatur:tempType)**

Beschreibung: Setzt den Wert des Attributes Zieltemperatur auf Temperatur ˚C.

2.2.3.14 Classtable der Klasse ScheduledFct

ScheduledFct		
Attribute	**Type**	**Default Value**
Operation	**Arguments**	**Return Type**
registerCallBack		void
unregisterCallBack		void
callBack		void

2.2.3.14.1 Attribute der Klasse ScheduledFct

Keine.

2.2.3.14.2 Methodenschnittstellen der Klasse ScheduledFct

2.2.3.15 Classtable der Klasse Scheduler

Scheduler		
Attribute	**Type**	**Default Value**
Operation	**Arguments**	**Return Type**
Start		void
Stop		void
registerCallBack		void
unregisterCallBack		void

2.2.3.15.1 Attribute der Klasse Scheduler

Keine.

2.2.3.15.2 Methodenschnittstellen der Klasse Scheduler

Signatur: **void Start()**

Beschreibung: Started den Scheduler.

Signatur: **void Stop()**

Beschreibung: Stopped den Scheduler.

2.2.3.16 Classtable der Klasse Section

Section		
Attribute	**Type**	**Default Value**
SectionNr	int	
Operation	**Arguments**	**Return Type**
BenoetigteKesselTempAbfragen		tempType
AussentemperaturWeiterleiten	Temperatur:tempType	void
RegenAnZimmer		void
Section	InitialSectionNr:int, ZimmerFlaechenListe:SList<double>, ZeitPtr:Timer*	

2.2.3.16.1 Attribute der Klasse Section

Attribut	Erläuterung
SectionNr	Nummer des Abschnitts. Wird für die Aufrufe der Portbaustein-Methoden benötigt.

2.2.3.16.2 Methodenschnittstellen der Klasse Section

Signatur: **void AussentemperaturWeiterleiten(Temperatur:tempType)**

Beschreibung: Leitet die Aussentemperatur an alle Zimmer weiter.

Signatur: **tempType BenoetigteKesselTempAbfragen()**

Beschreibung: Fragt alle Zimmer nach der benötigten Kesseltemperatur durch Aufruf der Methode BenoetigteKesselTempAbfragen(). Liefert das Maximum aller Temperaturen als Ergebnis zurück.

Signatur: **void RegenAnZimmer()**

Beschreibung: Meldet neu einsetzenden Regen an alle Zimmer.

2.2.3.17 Classtable der Klasse Temperatureregulator

Temperatureregulator		
Attribute	**Type**	**Default Value**
AbwesenheitsTempNormal	tempType	15
AnwesenheitsTempNormal	tempType	20
ZeitspanneAufheizenNormal	durationType	15
BenoetigteKesseltemperatur	tempType	0
CZeitspanneAufheizenDefault	durationType	15
CAbwesenheitsTempDefault	tempType	15
CAnwesenheitsTempDefault	tempType	20
ZFenster	zFensterType	CFensterGeschlossen
ZAnwesend	zAnwesenheitType	CNiemandDa
Aussentemperatur	tempType	20
SectionNr	int	
RoomNr	int	
Flaeche	double	
Zimmertemperatur	tempType	15
TemperaturStatus	zZimmerTempType	CZimmerTempGut
ZeitspanneEintrittNormal	durationType	10
Operation	**Arguments**	**Return Type**
BenoetigteKesselTempLesen		tempType
HalteTempBerechnen		void
HeizTempBerechnen		void

FensterzustandSetzen	Position:zFensterType	void
AnwesenheitSetzen	Anwesenheit:zAnwesen-heitType	void
ZimmertemperaturSetzen	Temperatur:TempType	void
AussentemperaturSetzen	Temperatur:TempType	void
TemperaturStatusSetzen	Zustand:zZimmerTem-pType	void
ZeitspanneAufheizenAufDefault		fehlerType
ZeitspanneAufheizenNormalSetzen	Zeitspanne:durationType	fehlerType
AnwesenheitsTempAufDefault		void
AnwesenheitsTempNormalSetzen	Temperatur:tempType	void
AbwesenheitsTempAufDefault		void
AbwesenheitsTempNormalSetzen	Temperatur:tempType	void
ZeitspanneEintrittNormalSetzen	Zeitspanne:durationType	void
Temperatureregulator	InitialSectionNr:int, InitialRoomNr:int, Flaeche:double, WindowPtr:Window*	
Inherited Class::Operation	**Arguments**	**Return Type**
ScheduledFct::registerCallBack		void
ScheduledFct::unregisterCallBack		void
ScheduledFct::callBack		void

2.2.3.17.1 Attribute der Klasse Temperatureregulator

Attribut	Erläuterung
SectionNr	Nummer des Abschnitts, zu dem der Temperatureregulator gehört.
RoomNr	Nummer des Zimmers, zu dem der Temperatureregulator gehört.
Aussentemperatur	Momentane Aussentemperatur
Zimmertemperatur	momentane Zimmertemperatur
ZFenster	Zustand des Fensters

Attribut	Erläuterung
ZAnwesend	Gibt an, ob sich eine Person im Zimmer befindet oder nicht. Dieses Attribut wird erst gesetzt, nachdem die Verlassen- und Eintritt-Zeitspannen abgelaufen sind.
TemperaturStatus	Gibt an, ob die Zimmertemperatur gut, zu tief oder zu hoch ist.
AbwesenheitsTempNormal	Temperatur in ˚C, die im Zimmer gehalten werden soll, wenn das Zimmer leer ist und die Zeitspanne ZeitspanneVerlassenNormal verstrichen ist.
CAbwesenheitsTempDefault	Defaultwert für AbwesenheitsTempNormal in ˚C.
AnwesenheitsTempNormal	Temperatur in ˚C, die im Zimmer gehalten werden soll, wenn eine Person im Zimmer anwesend ist und die Zeitspanne ZeitspanneEintrittNormal verstrichen ist.
CAnwesenheitsTempDefault	Defaultwert für AnwesenheitsTempNormal in ˚C.
ZeitspanneAufheizenNormal	Zeitspanne, die angibt, nach wieviel Minuten die Anwesenheitstemperatur nach Betreten des Zimmers durch eine Person erreicht sein muß. Diese Zeitspanne muß größer sein als die Zeitspanne ZeitspanneEintrittNormal.
CZeitspanneAufheizenDefault	Defaultwert für ZeitspanneAufheizenNormal in Minuten.
BenoetigteKesseltemperatur	Die zum Aufheizen oder Halten der Zimmertemperatur benötigte minimale Kesseltemperatur.
Flaeche	Fläche des zugehörigen Zimmers in qm.
ZeitspanneEintrittNormal	Zeitspanne in Minuten, die angibt, wie lange sich eine Person im Zimmer befinden muß, bis das System regelnd eingreift. Wird für die Berechnung der Aufheiztemperatur benötigt und durch den PRTempController gesetzt.

2.2.3.17.2 Methodenschnittstellen der Klasse Temperatureregulator

Signatur: **void AussentemperaturSetzen(Temperatur:TempType)**

Beschreibung: Setzt das Attribut Aussentemperatur auf Temperatur ˚C.

Signatur: **void ZimmertemperaturSetzen(Temperatur:TempType)**

Beschreibung: Setzt das Attribut Zimmertemperatur auf Temperatur ˚C.

Signatur: **void AnwesenheitSetzen(Anwesenheit:zAnwesenheitType)**

Beschreibung: Setzt das Attribut ZAnwesend auf einen der Anwesenheitswerte.

Signatur: **void FensterzustandSetzen(Position:zFensterType)**

Beschreibung: Setzt das Attribut ZFenster auf den entsprechenden Fensterzustand.

Signatur: **tempType BenoetigteKesselTempLesen()**

Beschreibung: Liefert die im Attribut BenoetigteKesseltemperatur festgehaltene vom Zimmer benötigte Kesseltemperatur zurück.

Signatur: **void TemperaturStatusSetzen(Zustand:zZimmerTempType)**

Beschreibung: Setzt das Attribut TemperaturStatus auf den entsprechenden Zimmertemperaturstatus.

Signatur: **void ZeitspanneEintrittNormalSetzen(Zeitspanne:durationType)**

Beschreibung: Setzt das Attribut ZeitspanneEintrittNormal auf die entsprechende Zeitspanne.

Signatur: **fehlerType ZeitspanneAufheizenAufDefault()**

Beschreibung: Setzt den Wert des Attributes ZeitspanneAufheizenNormal auf den Wert des Attributes ZeitspanneAufheizenDefault. Liefert CWertUngueltig zurück, wenn die Zeitspanne-AufheizenDefault nicht 5 Minuten größer ist als die ZeitspanneEintrittNormal.

Signatur: **fehlerType ZeitspanneAufheizenNormalSetzen(Zeitspanne:durationType)**

Beschreibung: Setzt den Wert des Attributes ZeitspanneAufheizenNormal auf Zeitspanne Minuten. Liefert CWertUngueltig zurück, wenn die ZeitspanneAufheizenNormal nicht mehr als 5 Minuten größer ist als die ZeitspanneEintrittNormal.

Signatur: **void AnwesenheitsTempAufDefault()**

Beschreibung: Setzt den Wert des Attributes AnwesenheitsTempNormal auf den Wert des Attributes AnwesenheitsTempDefault.

Signatur: **void AnwesenheitsTempNormalSetzen(Temperatur:tempType)**

Beschreibung: Setzt den Wert des Attributes AnwesenheitsTempNormal auf Temperatur ˚C.

Signatur: **void AbwesenheitsTempAufDefault()**

Beschreibung: Setzt den Wert des Attributes AbwesenheitsTempNormal auf den Wert des Attributes AbwesenheitsTempDefault.

Signatur: **void AbwesenheitsTempNormalSetzen(Temperatur:tempType)**

Beschreibung: Setzt den Wert des Attributes AbwesenheitsTempNormal auf Temperatur ˚C.

2.2.3.18 Classtable der Klasse Temperaturesensor

Temperaturesensor		
Attribute	**Type**	**Default Value**
Operation	**Arguments**	**Return Type**
AbfrageTemperatur		tempType
Temperaturesensor	SectionNr:int, RoomNr:int, PortNr:int	
Inherited Class::Operation	**Arguments**	**Return Type**
Portbaustein::CallBitInputPort	SectionNr:int, RoomNr:int, PortNr:int	int
Portbaustein::CallBitOutputPort	SectionNr:int, RoomNr:int, PortNr:int, value:int	void
Portbaustein::CallIntInputPort	SectionNr:int, RoomNr:int, PortNr:int	int
Portbaustein::CallIntOutputPort	SectionNr:int, RoomNr:int, PortNr:int, value:int	void

2.2.3.18.1 Attribute der Klasse Temperaturesensor

Keine.

2.2.3.18.2 Methodenschnittstellen der Klasse Temperaturesensor

Signatur: **tempType AbfrageTemperatur()**

Beschreibung: Gibt die aktuelle Temperatur zurück. Benutzt hierfür die Methoden des Portbausteins.

2.2.3.19 Classtable der Klasse Time

Time		
Attribute	**Type**	**Default Value**

Operation	**Arguments**	**Return Type**
operator<	t:Time&	bool
operator<=	t:Time&	bool
operator++		Time
getTime	h:timeCompType&, mi:time-CompType&, d:timeCom-pType&, mo:timeCompType&.y:time-CompType&	void
setTime	h:timeCompType, mi:time-CompType, d:timeCompType, mo:timeCompType, y:time-CompType	void
operator=	t:Time&	Time&
operator>	t:Time&	bool
operator>=	t:Time&	bool
operator==	t:Time&	bool
operator!=	t:Time&	bool

2.2.3.19.1 Attribute der Klasse Time

Keine.

2.2.3.20 Classtable der Klasse Timer

Timer		
Attribute	**Type**	**Default Value**
speedup	int	1
Operation	**Arguments**	**Return Type**
getActualTime		Time

2.2.3.20.1 Attribute der Klasse Timer

Attribut	Erläuterung
speedup	Beschleunigung der Systemzeit im Vergleich zur realen Zeit.

2.2.3.20.2 Methodenschnittstellen der Klasse Timer

Die Klasse Timer besitzt keine Methodenschnittstellen.

2.2.3.21 Classtable der Klasse Window

Window		
Attribute	**Type**	**Default Value**
SectionNr	int	
RoomNr	int	
Operation	**Arguments**	**Return Type**
FensterKippen		void
ZimmerKuehlen		void
RegenFenster		void
Window	InitialSectionNr:int, InitialRoomNr:int, RoomPtr:Room*	
Inherited Class::Operation	**Arguments**	**Return Type**
ScheduledFct::registerCallBack		void
ScheduledFct::unregisterCallBack		void
ScheduledFct::callBack		void

2.2.3.21.1 Attribute der Klasse Window

Attribut	Erläuterung
SectionNr	Nummer des Abschnitts, zu dem das Fenster gehört.
RoomNr	Nummer des Zimmers, zu dem das Fenster gehört.

2.2.3.21.2 Methodenschnittstellen der Klasse Window

Signatur: **void ZimmerKuehlen()**

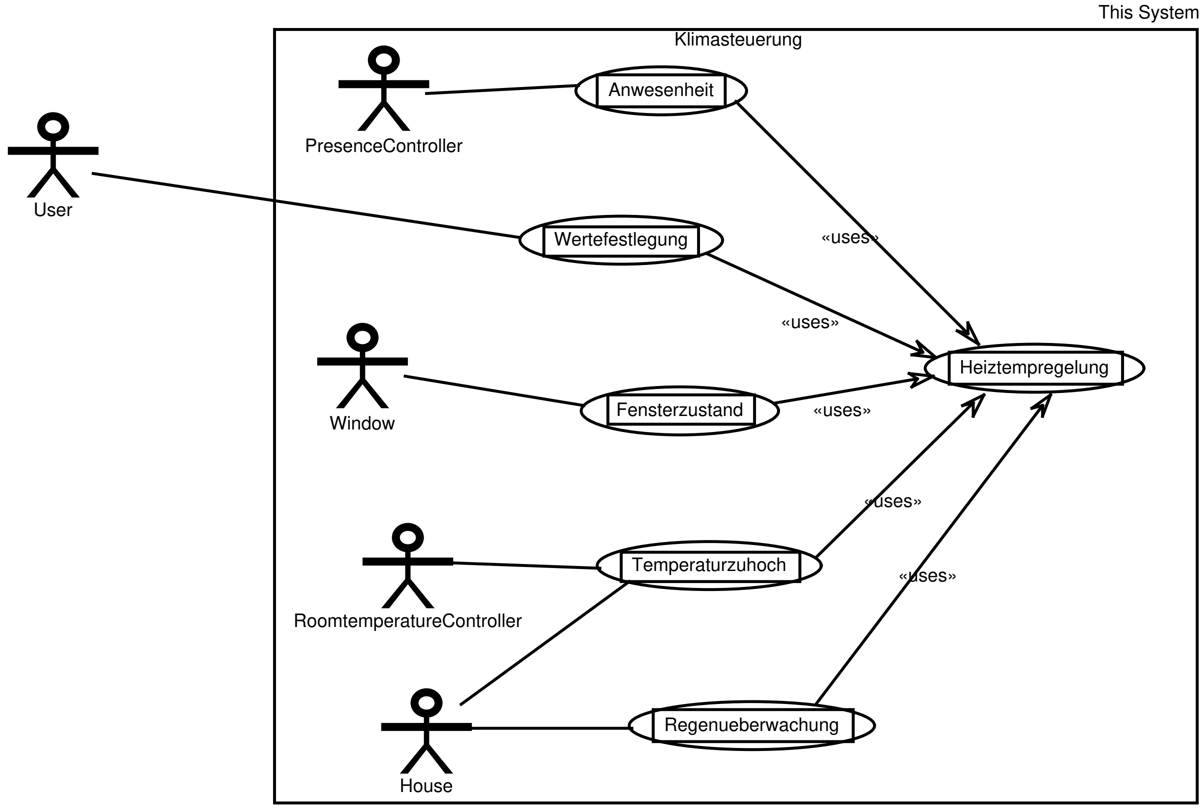
Beschreibung: Kühlt das Zimmer durch den Aufruf der Methode FensterKippen().

Signatur: **void RegenFenster()**

Beschreibung: Ruft FensterKippen() auf.

2.2.4 UML Use Case-Diagramm

Da die Sequenzdiagramme des Entwurfs entsprechend den Use Cases der Analyse geordnet sind, wird an dieser Stelle das Use Case-Diagramm erneut wiederholt.

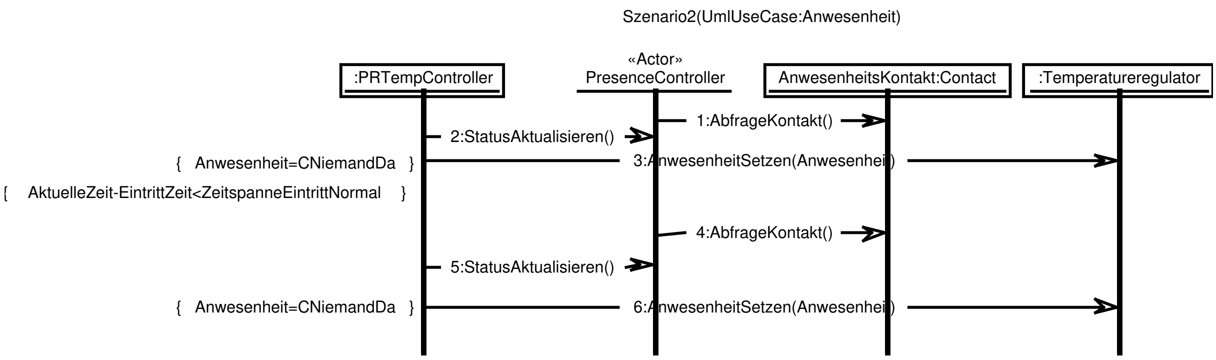


2.2.5 UML-Sequenzdiagramme

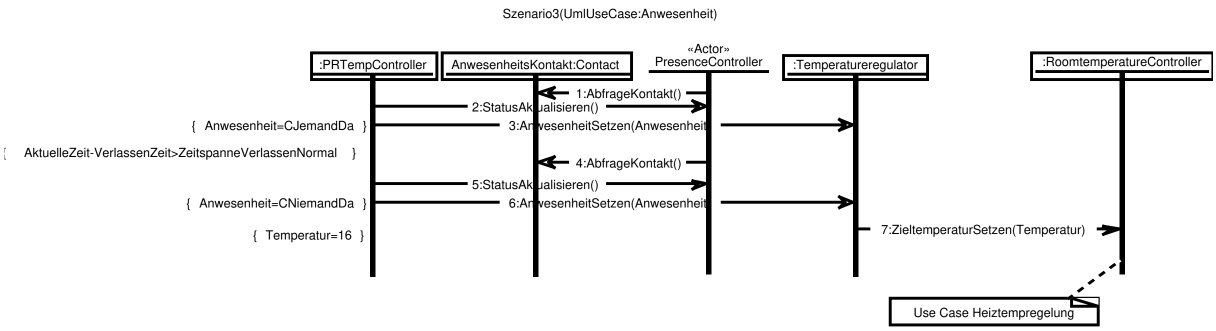
Die UML-Sequenzdiagramme des Entwurfs, basieren auf denen der UML Analyse, d.h. sie basieren auf den Szenarien der Analyse.Die Klassen in den UML-Sequenzdiagrammen wurden entsprechend den Änderungen im UML-Klassendiagramm angepasst. Alle Events aus der Analyse wurden in Methodenaufrufe aus dem Entwurf transformiert.

2.2.5.0.1 Sequenzdiagramme zu UseCase Anwesenheit

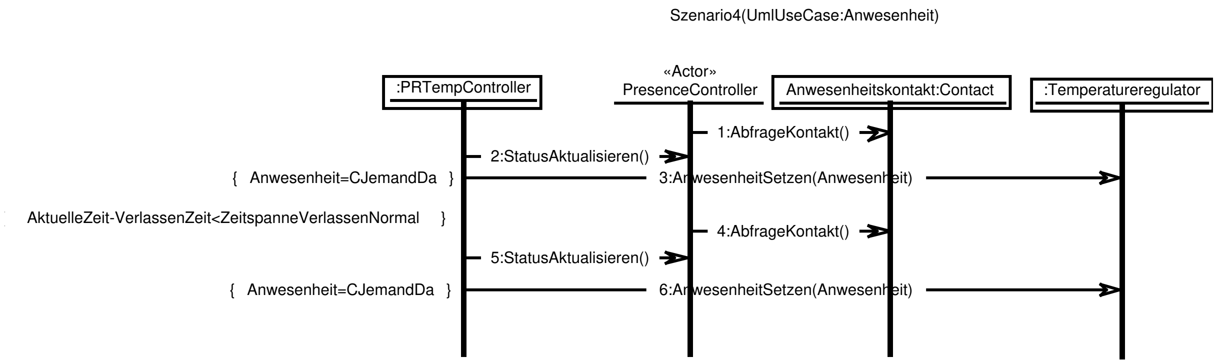
Sequence-Diagram zu Szenario2:



Sequence-Diagram zu Szenario3:

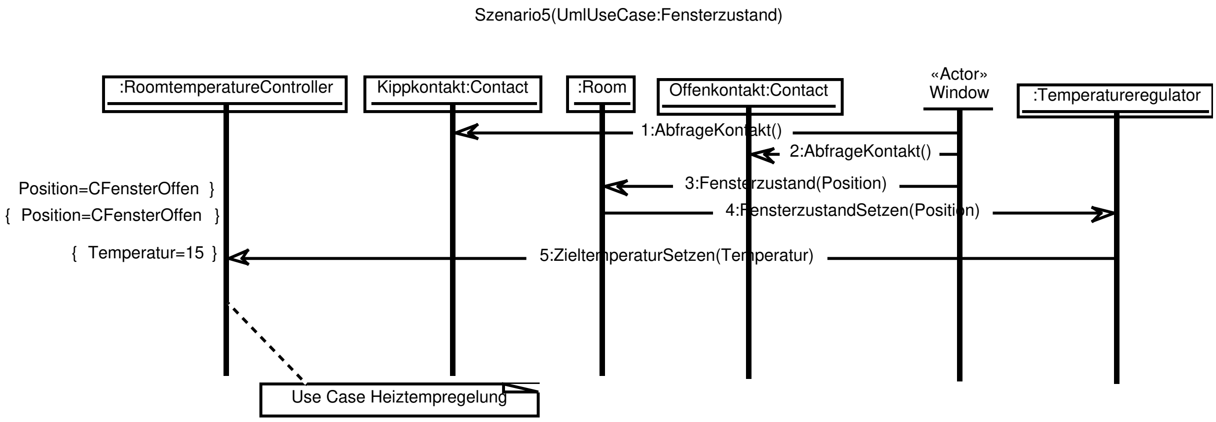


Sequence-Diagram zu Szenario4:



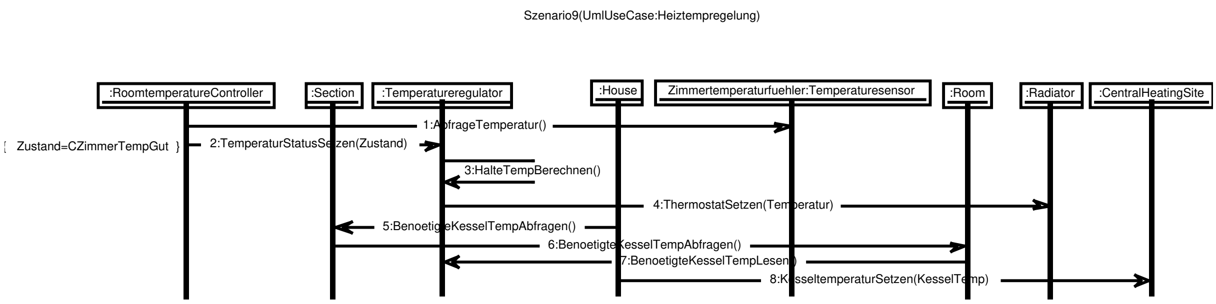
2.2.5.0.2 Sequenzdiagramme zu UseCase Fensterzustand

Sequence-Diagram zu Szenario5:

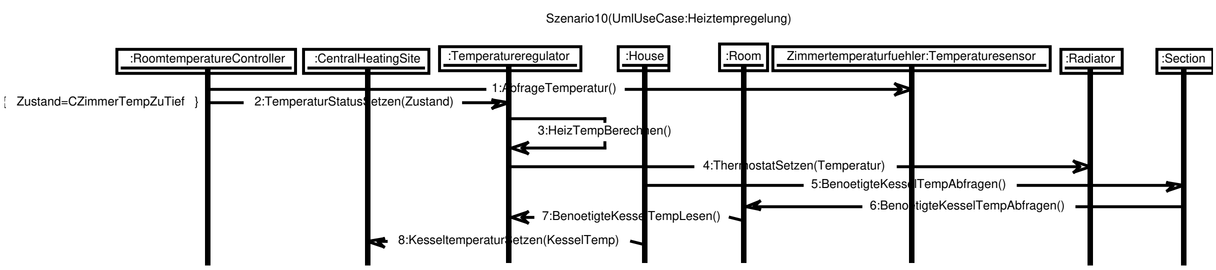


2.2.5.0.3 Sequenzdiagramme zu UseCase Heiztempregelung

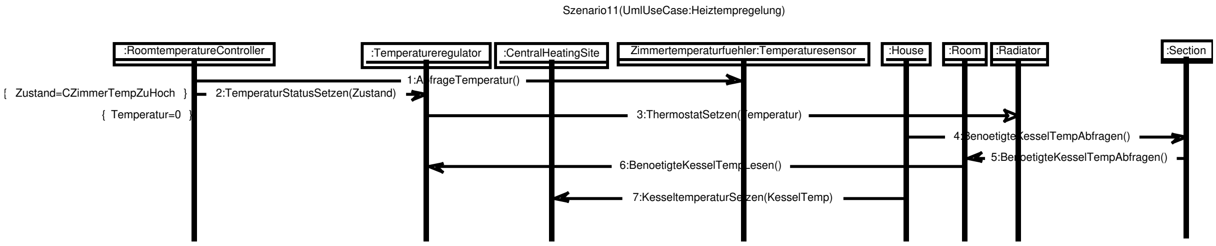
Sequence-Diagram zu Szenario9:



Sequence-Diagram zu Szenario10:

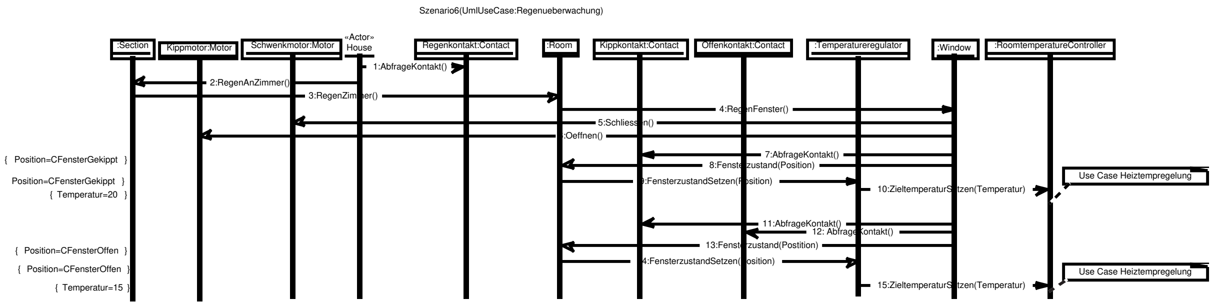


Sequence-Diagram zu Szenario11:



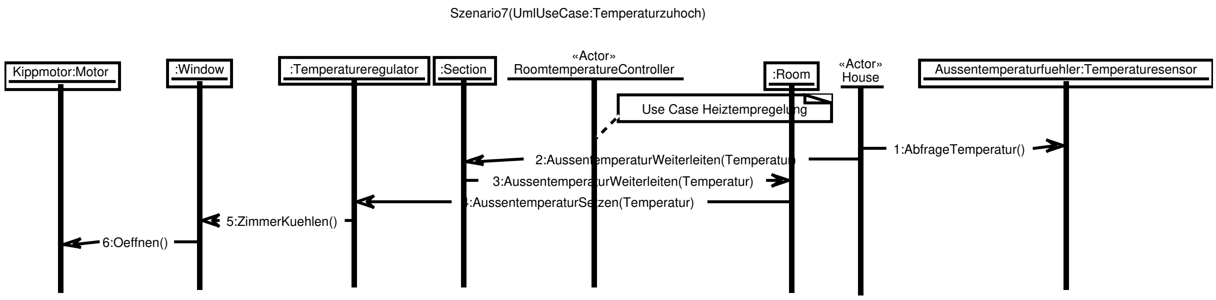
2.2.5.0.4 Sequenzdiagramme zu UseCase Regenueberwachung

Sequence-Diagram zu Szenario6:



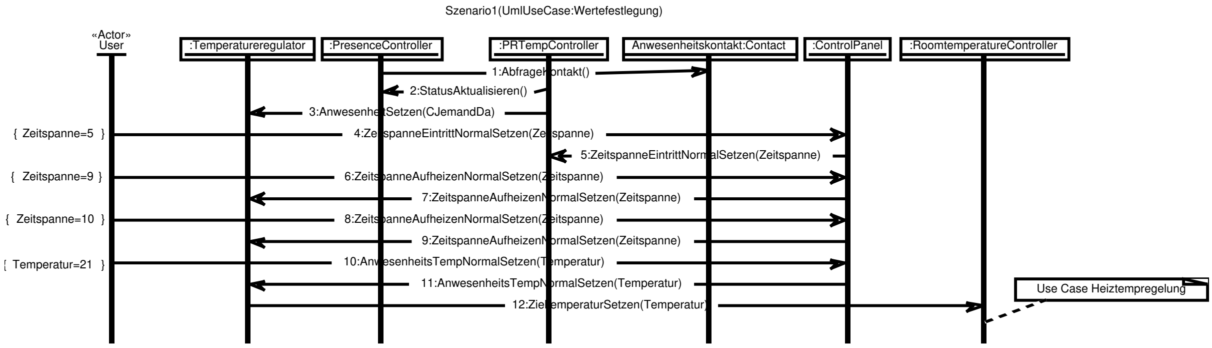
2.2.5.0.5 Sequenzdiagramme zu UseCase Temperaturzuhoch

Sequence-Diagram zu Szenario7:

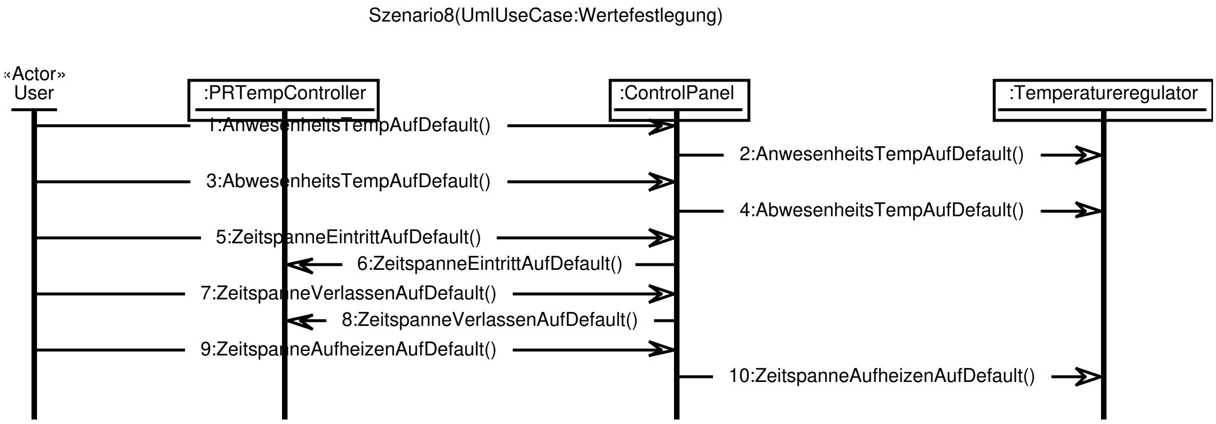


2.2.5.0.6 Sequenzdiagramme zu UseCase Wertefestlegung

Sequence-Diagram zu Szenario1:



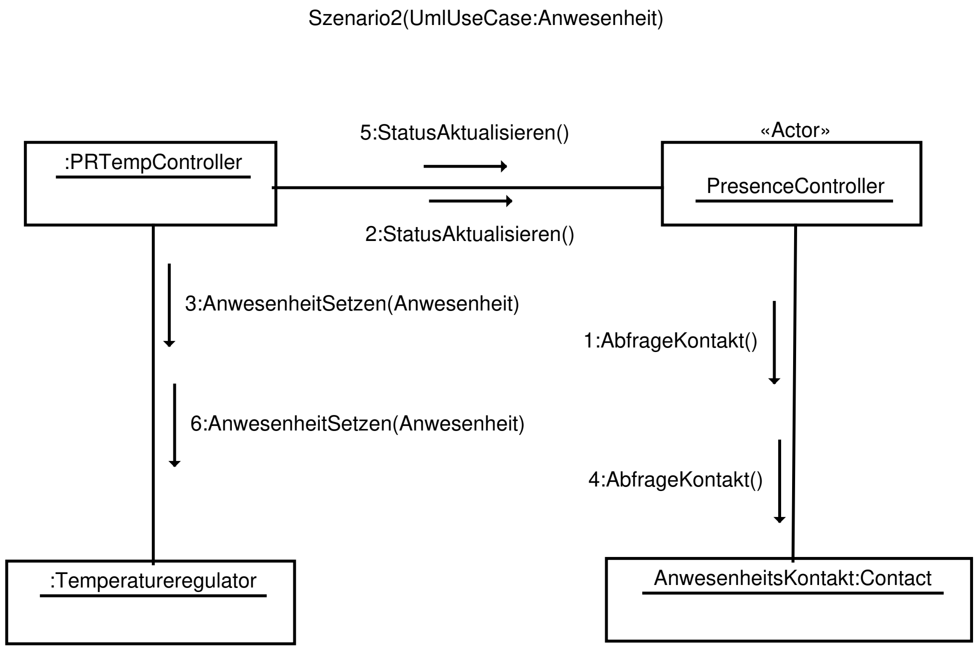
Sequence-Diagram zu Szenario8:



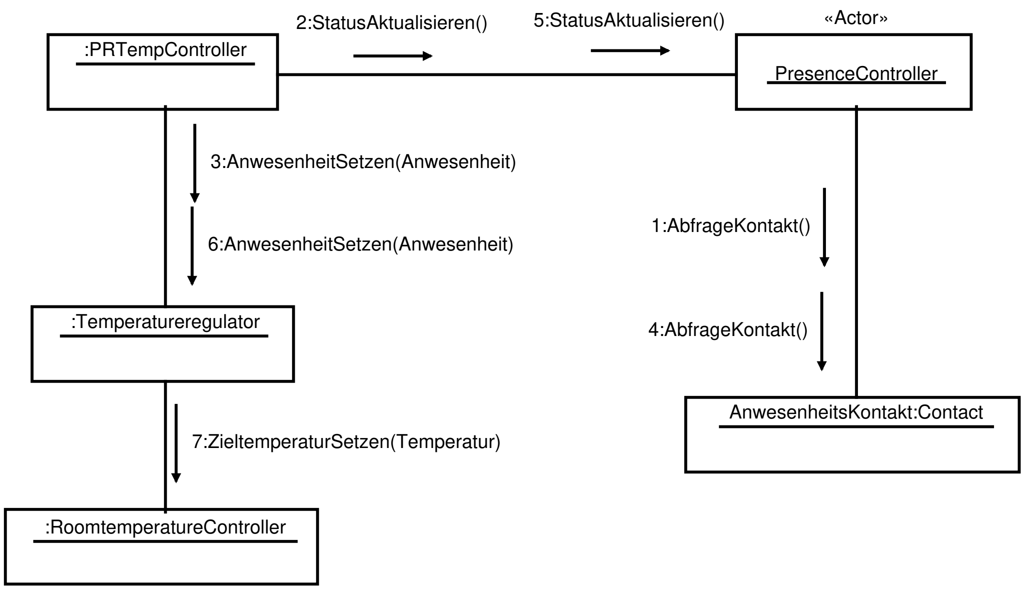
2.2.6 UML-Kollaborationsdiagramme

Im folgenden sind die Kollaborationsdiagramme der Objekte der UML-Klassendiagramme aufgeführt. Sie wurden mit Hilfe des Werkzeugs StP/UML automatisch aus den Sequenzdiagrammen abgeleitet.

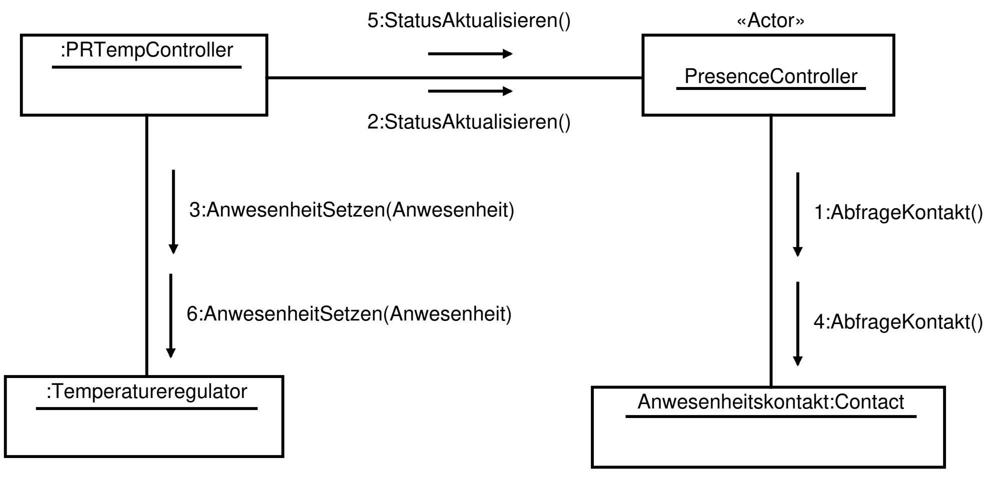
2.2.6.1 Anwesenheit



Szenario3(UmlUseCase:Anwesenheit)

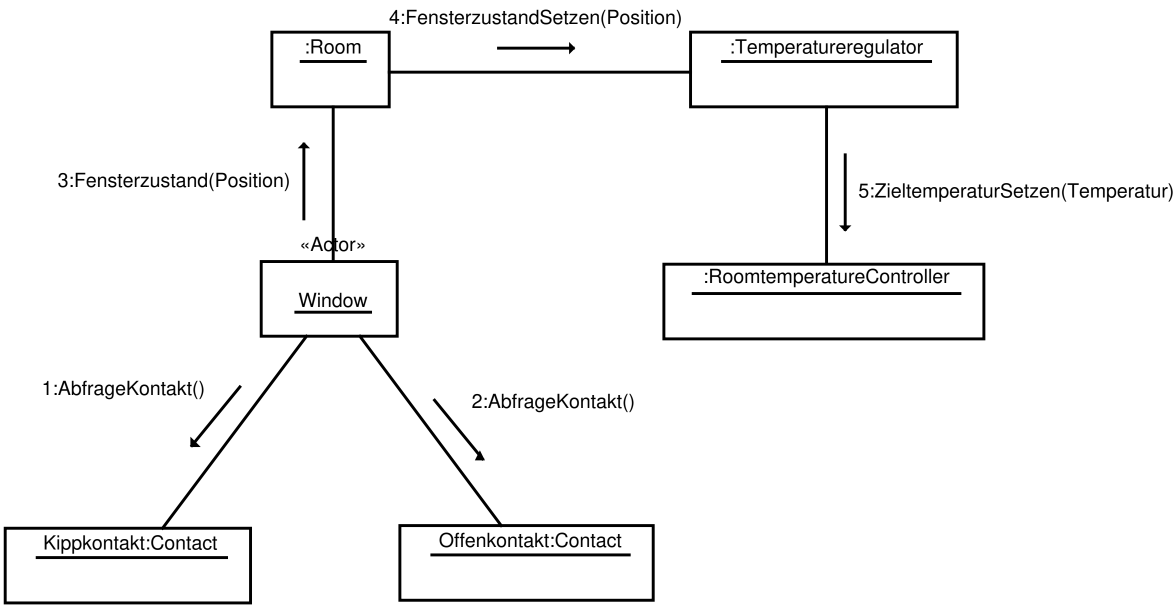


Szenario4(UmlUseCase:Anwesenheit)



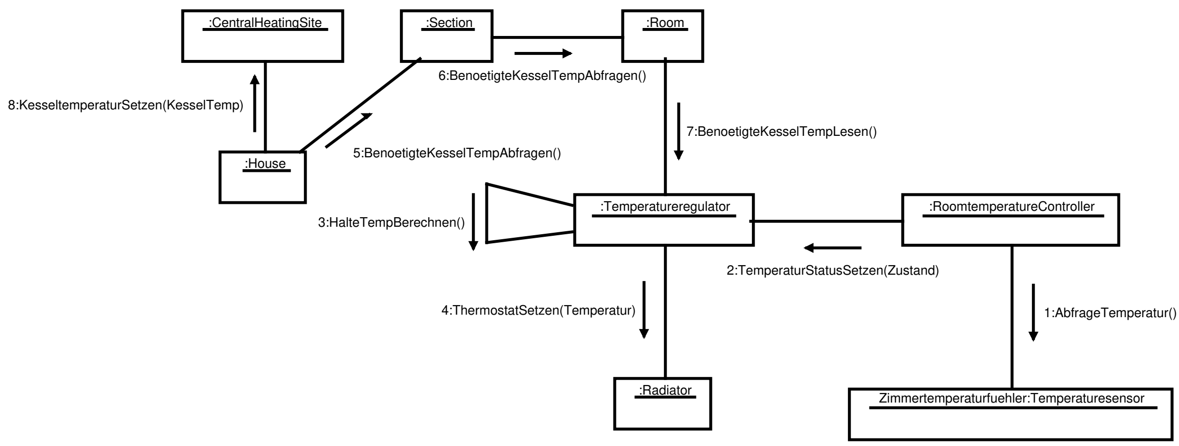
2.2.6.2 Fensterzustand

Szenario5(UmlUseCase:Fensterzustand)

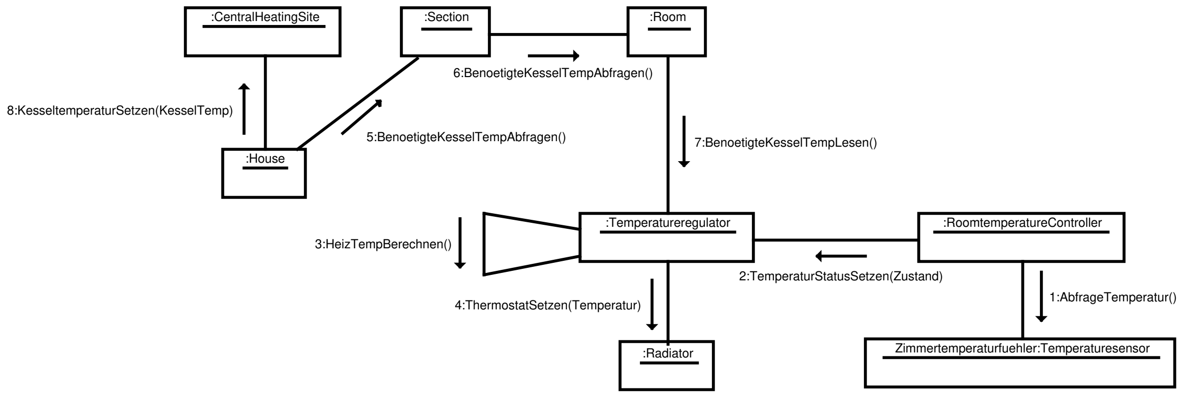


2.2.6.3 Heiztempregelung

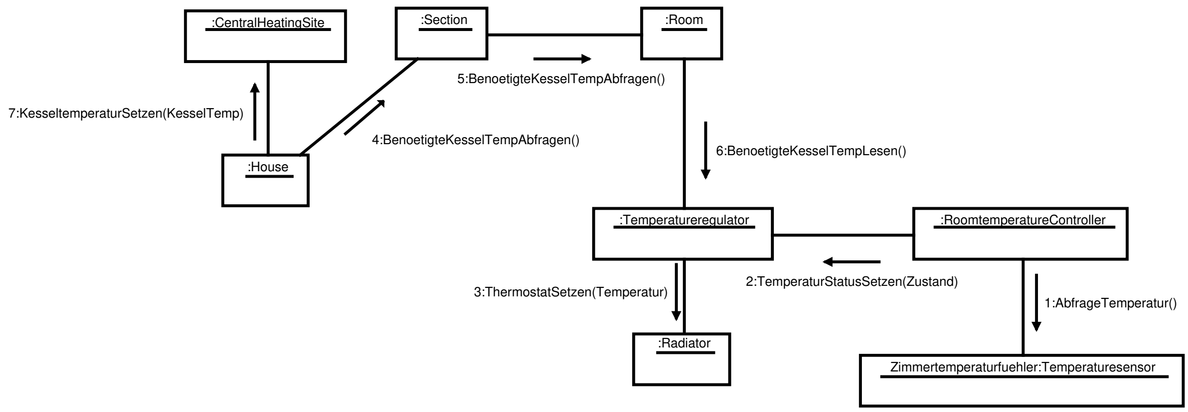
Szenario9(UmlUseCase:Heiztempregelung)



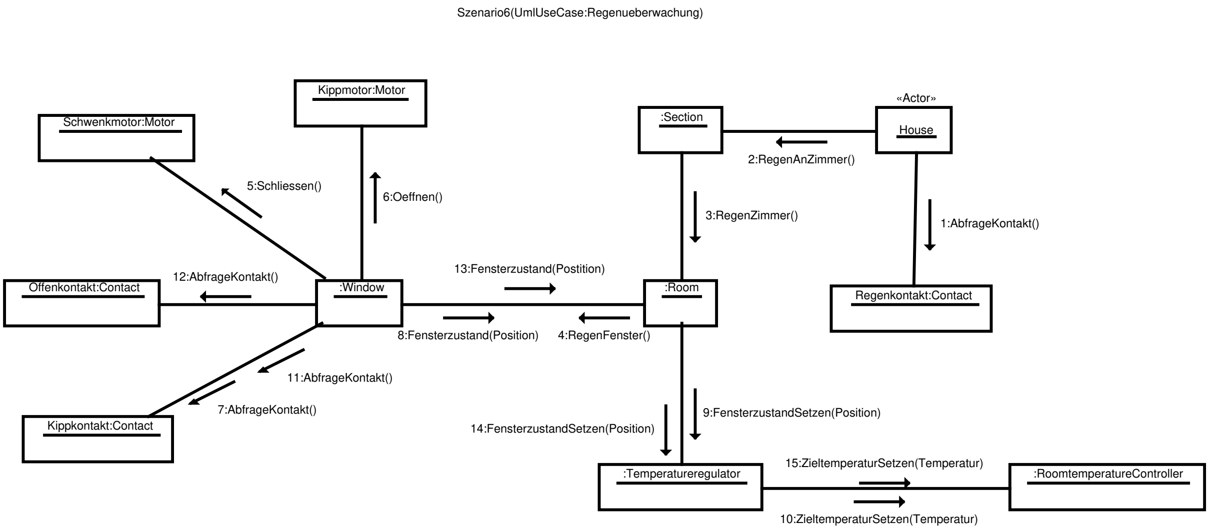
Szenario10(UmlUseCase:Heiztempregelung)



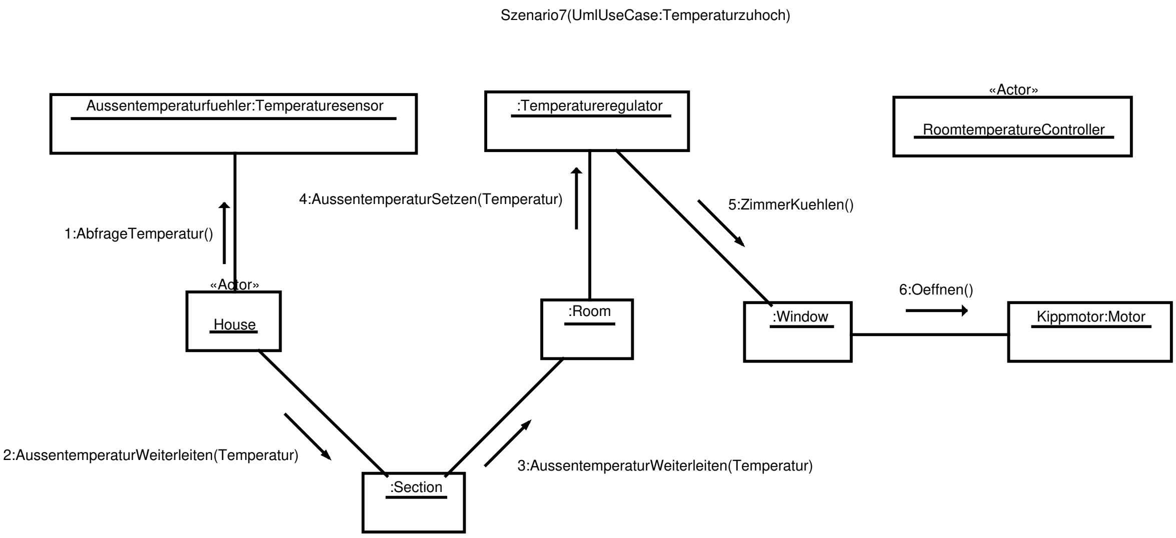
Szenario11(UmlUseCase:Heiztempregelung)



2.2.6.4 Regenueberwachung

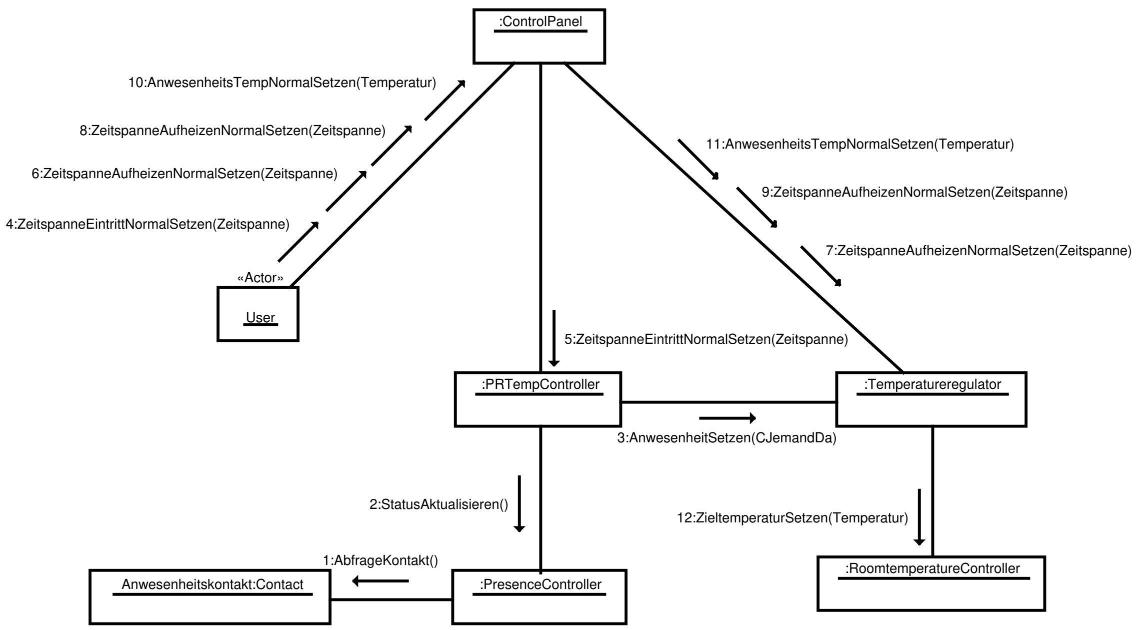


2.2.6.5 Temperaturzuhoch

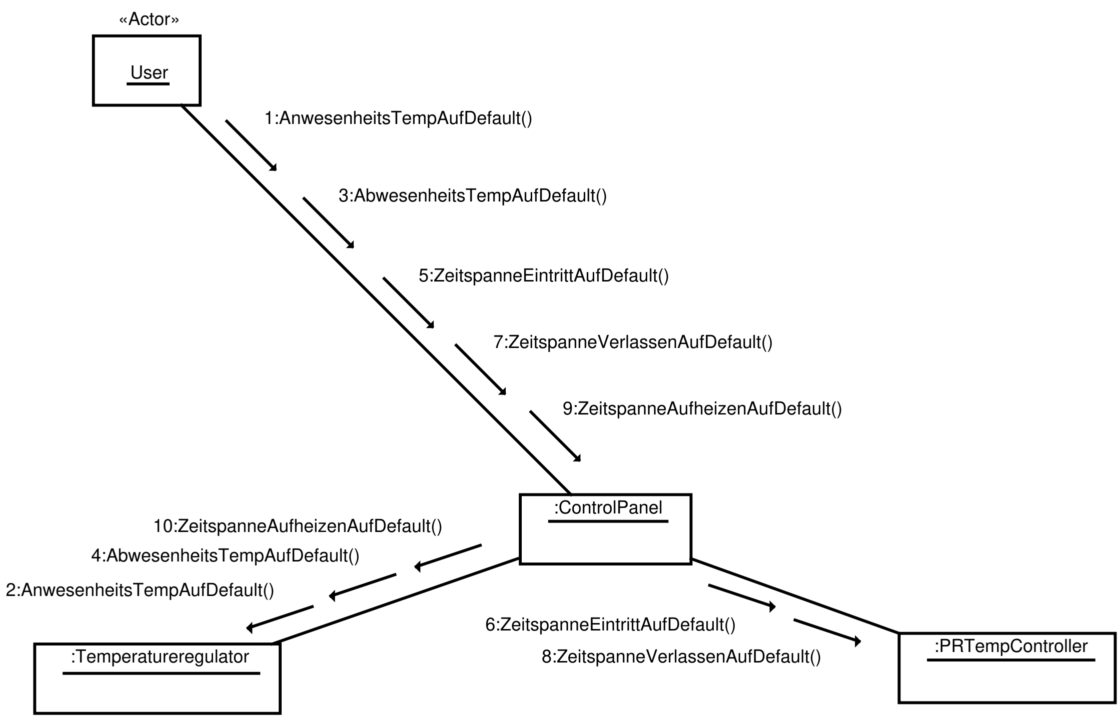


2.2.6.6 Wertefestlegung

Szenario1(UmlUseCase:Wertefestlegung)

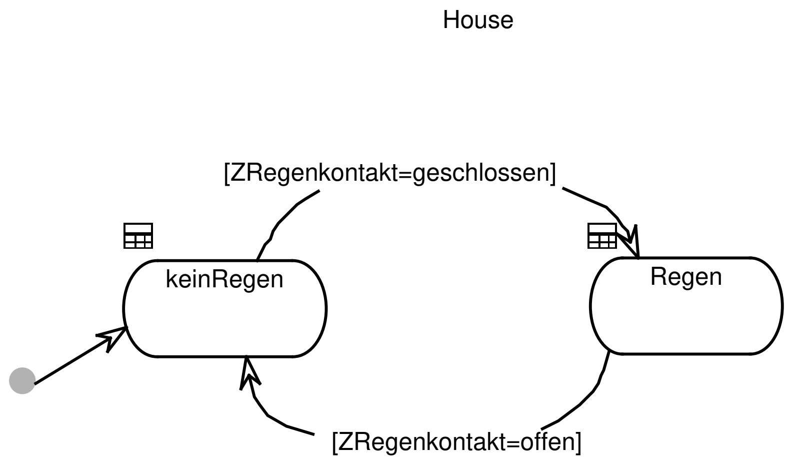


Szenario8(UmlUseCase:Wertefestlegung)



2.2.7 UML-State-Diagramme

2.2.7.1 House



Entry-Aktion für State: Regen

RegenAnSection()

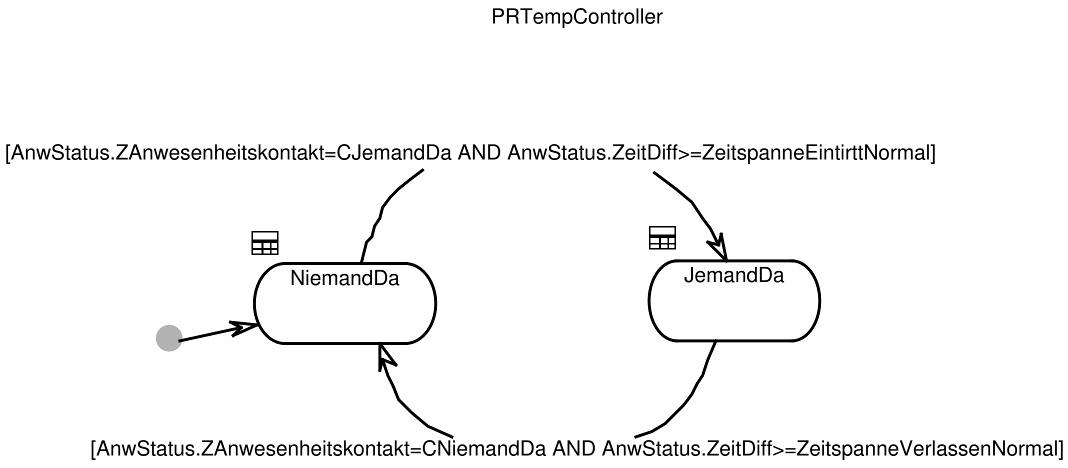
Do-Aktivität für State: Regen

ZRegenkontakt=Regenkontakt.AbfrageKontak()

Do-Aktivität für State: keinRegen

ZRegenkontakt=Regenkontakt.AbfrageKontakt()

2.2.7.2 PRTempController



Entry-Aktion für State: NiemandDa

Temperatureregulator.AnwesenheitSetzen(CNiemandDa)

Do-Aktivität für State: NiemandDa

AnwStatus=PresenceController.StatusAktualisieren()

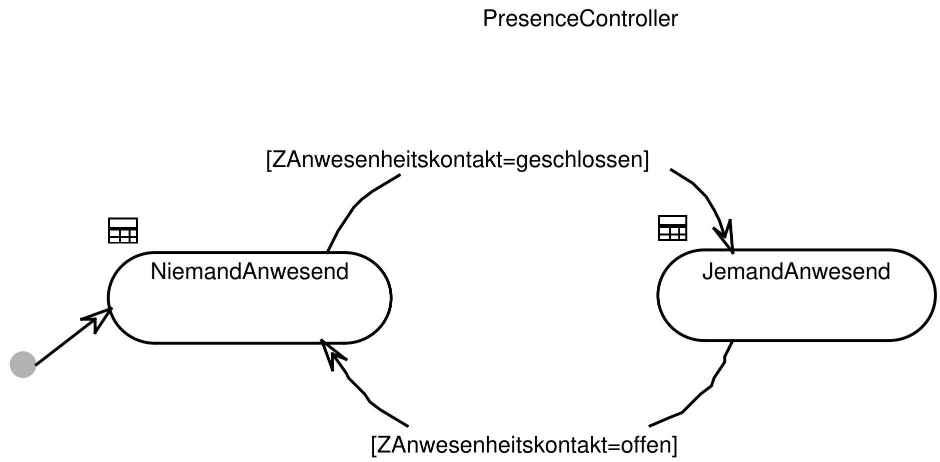
Entry-Aktion für State: JemandDa

Temperatureregulator.AnwesenheitSetzen(CJemandDa)

Do-Aktivität für State: JemandDa

AnwStatus=PresenceController.StatusAktualisieren()

2.2.7.3 PresenceController



Entry-Aktion für State: NiemandAnwesend

VerlassenZeit=Zeit.getActualTime()

Do-Aktivität für State: NiemandAnwesend

ZAnwesenheitskontakt=Anwesenheitskontakt.AbfrageKontakt()

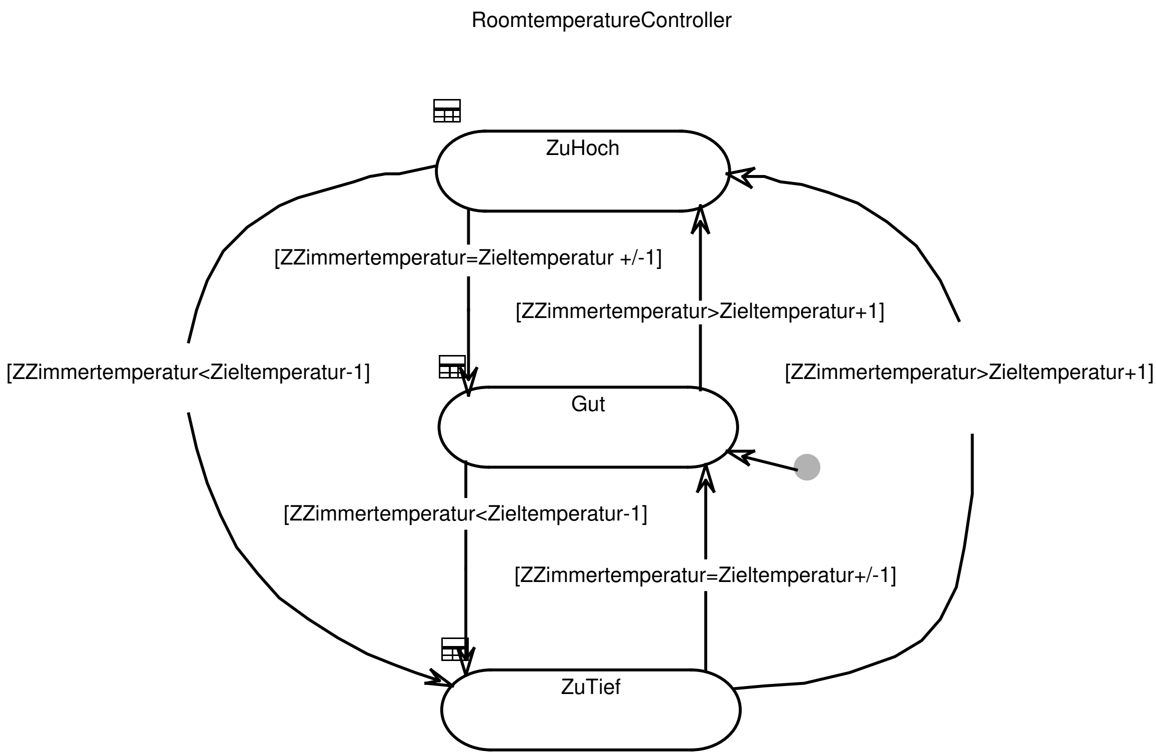
Entry-Aktion für State: JemandAnwesend

EintrittZeit=Zeit.getActualTime()

Do-Aktivität für State: JemandAnwesend

ZAnwesenheitsKontakt=Anwesenheitskontakt.AbfrageKontakt()

2.2.7.4 RoomtemperatureController



Entry-Aktion für State: ZuHoch

Temperatureregulator.ZimmertemperaturSetzen(ZZimmertemperatur)
Temperatureregulator.TemperaturStatusSetzen(CZimmerTempZuHoch)

Do-Aktivität für State: ZuHoch

ZZimmertemperatur=Zimmertemperaturfuehler.AbfrageTemperatur()

Entry-Aktion für State: Gut

Temperatureregulator.ZimmertemperaturSetzen(ZZimmertemperatur)
Temperatureregulator.TemperaturStatusSetzen(CZimmerTempGut)

Do-Aktivität für State: Gut

ZZimmertemperatur=Zimmertemperaturfuehler.AbfrageTemperatur()

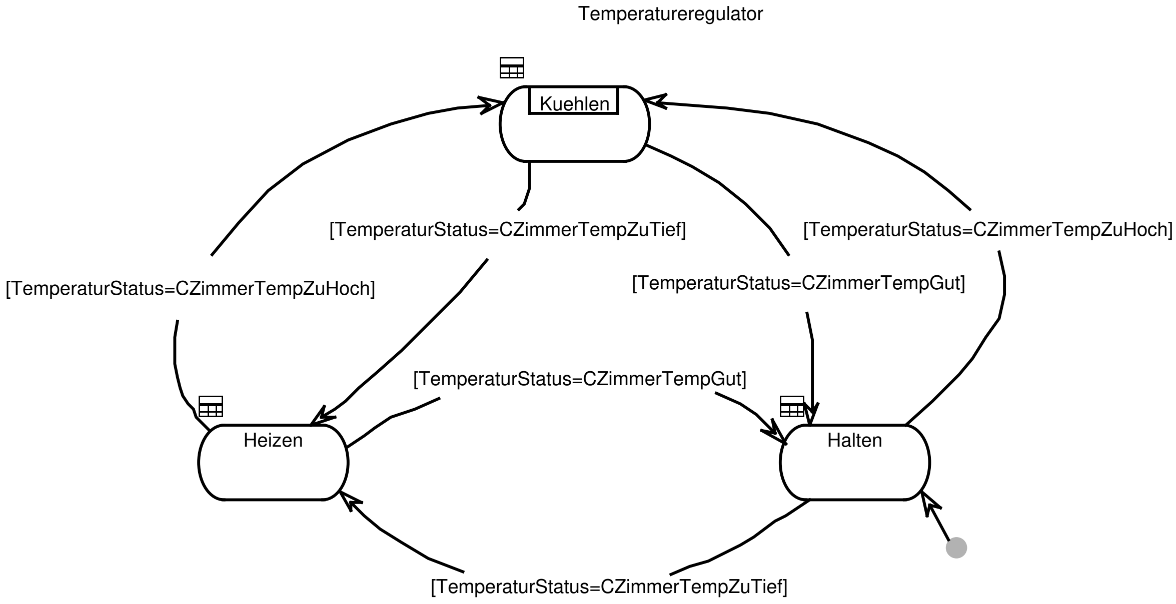
Entry-Aktion für State: ZuTief

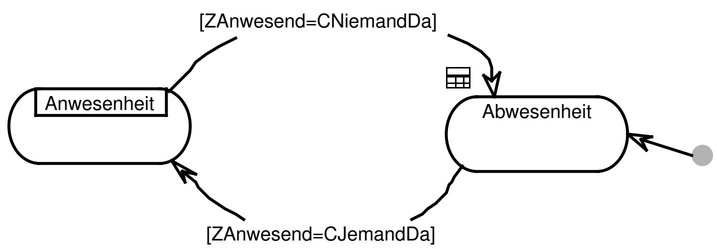
Temperatureregulator.ZimmertemperaturSetzen(ZZimmertemperatur)
Temperatureregulator.TemperaturStatusSetzen(CZimmerTempZuTief)

Do-Aktivität für State: ZuTief

ZZimmertemperatur=Zimmertemperaturfuehler.AbfrageTemperatur()

2.2.7.5 Temperatureregulator





Entry-Aktion für State: Heizen

HeizTempBerechnen()
Radiator.ThermostatSetzen(BenoetigteKesselTemp)

Entry-Aktion für State: Halten

Radiator.ThermostatSetzen(BenoetigteKesseltemperatur)
HalteTempBerechnen()

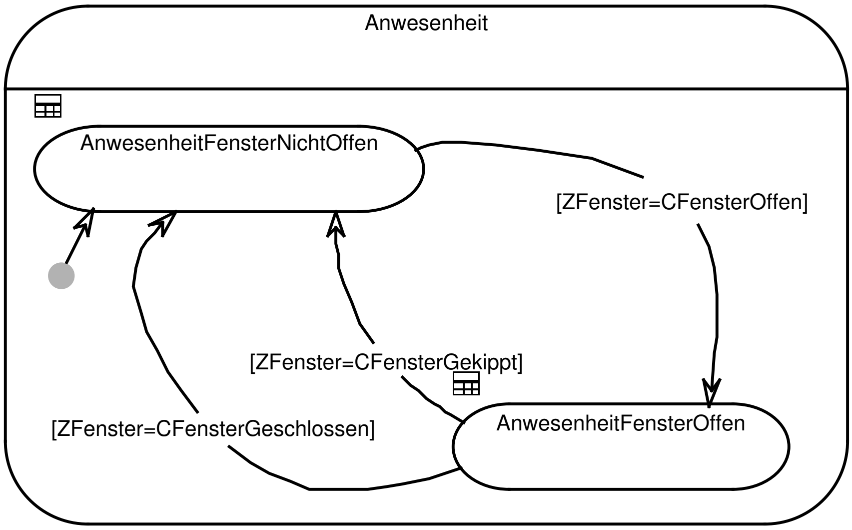
Entry-Aktion für State: Kuehlen

Radiator.ThermostatSetzen(0)
BenoetigteKesseltemperatur=0

Entry-Aktion für State: Abwesenheit

RoomtemperatureController.ZieltemperaturSetzen(AbwesenheitsTempNormal)
Zieltemperatur=AbwesenheitsTempNormal

Temperatureregulator



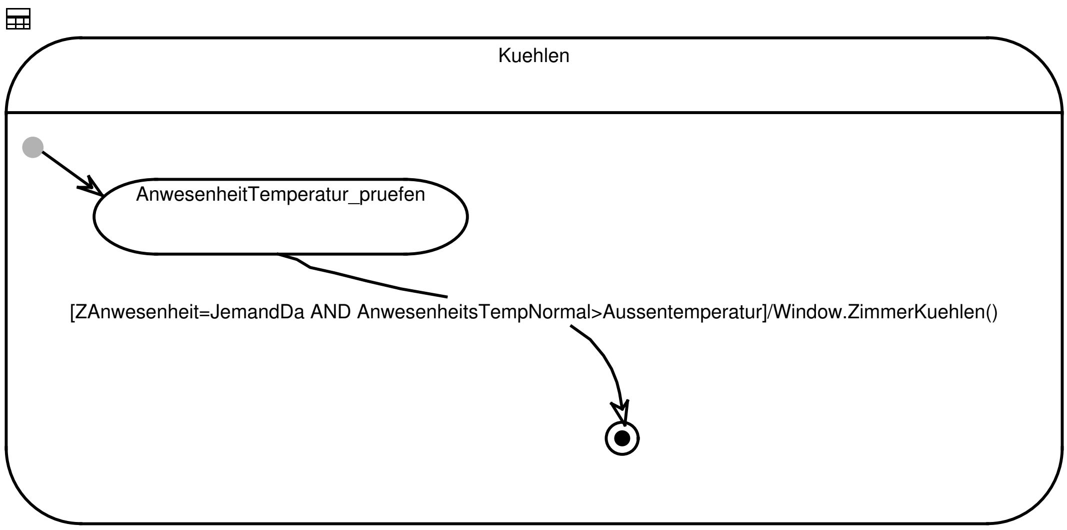
Entry-Aktion für State: AnwesenheitFensterNichtOffen

RoomtemperatureController.ZieltemperaturSetzen(AnwesenheitsTempNormal)
Zieltemperatur=AnwesenheitsTempNormal

Entry-Aktion für State: AnwesenheitFensterOffen

RoomtemperatureController.ZieltemperaturSetzen(AbwesenheitsTempNormal)
Zieltemperatur=AbwesenheitsTempNormal

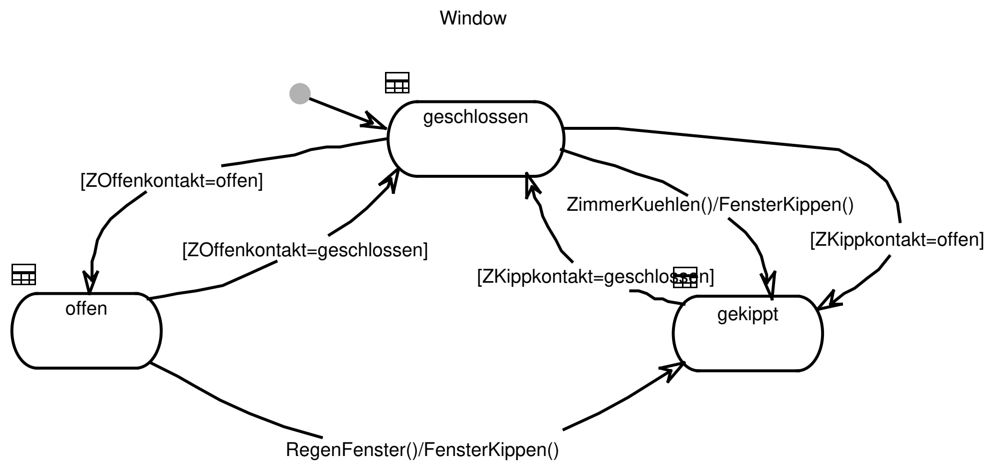
Temperatureregulator



Entry-Aktion für State: Kuehlen

Radiator.ThermostatSetzen(0)
BenoetigteKesseltemperatur=0

2.2.7.6 Window



Entry-Aktion für State: geschlossen

Room.Fensterzustand(CFensterGeschlossen)

Do-Aktivität für State: geschlossen

ZOffenkontakt=Offenkontakt.AbfrageKontakt()
ZKippkontakt=Kippkontakt.AbfrageKontakt()

Entry-Aktion für State: offen

Room.Fensterzustand(CFensterOffen)

Do-Aktivität für State: offen

ZOffenkontakt=Offenkontakt.AbfrageKontakt()
ZKippkontakt=Kippkontakt.AbfrageKontakt()

Entry-Aktion für State: gekippt

Room.Fensterzustand(CFensterGekippt)

Do-Aktivität für State: gekippt

ZOffenkontakt=Offenkontakt.AbfrageKontakt()
ZKippkontakt=Kippkontakt.AbfrageKontakt()

2.2.8 Verfolgbarkeitsmatrizen Analyse <-> Entwurf

Nachfolgend für jede Klasse der Analyse angegeben, durch welche Attribute und Methoden des Entwurfs die jeweiligen Attribute und Methoden der Analyse repräsentiert werden.

Die Methoden der Klassen *CentralHeatingSite*, *Contact*, *Motor*, *Radiator* und *Temperaturesensor* haben sich gegenüber der Analyse nicht geändert. Sie werden deshalb im folgenden nicht aufgeführt.

2.2.8.1 Klasse ControlPanel

	ControlPanel									
	AbwesenheitsTempAufDefault()	AbwesenheitsTempNormalSetzen()	AnwesenheitsTempAufDefault()	AnwesenheitsTempNormalSetzen()	ZeitspanneAufheizenAufDefault()	ZeitspanneAufheizenNormalSetzen()	ZeitspanneEintrittAufDefault()	ZeitspanneEintrittNormalSetzen()	ZeitspanneVerlassenAufDefault()	ZeitspanneVerlassenNormalSetzen()
AbwesenheitsTempAufDefault()	X									
AbwesenheitsTempNormalSetzen()		X								
AnwesenheitsTempAufDefault()			X							
AnwesenheitsTempNormalSetzen()				X						
ZeitspanneAufheizenAufDefault()					X					
ZeitspanneAufheizenNormalSetzen()						X				
ZeitspanneEintrittAufDefault()							X			
ZeitspanneEintrittNormalSetzen()								X		
ZeitspanneVerlassenAufDefault()									X	
ZeitspanneVerlassenNormalSetzen()										X

2.2.8.2 Klasse House

	House			
	MaxBenoetigteKesseltemperatur	Aussentemperatur	RegenAnSechtion()	callBack()
KesselTempAktualisieren()				X
RegenHaus()			X	

2.2.8.3 Klasse PRTempController

	PRTempController							
	ZeitspanneEintrittNormal	ZeitspanneVerlassenNormal	CZeitspanneEintrittDefault	CZeitspanneVerlassenDefault	ZeitspanneVerlassenAufDefault()	ZeitspanneVerlassenNormalSetzen()	ZeitspanneEintrittAufDefault()	ZeitspanneEintrittNormalSetzen()
ZeitspanneEintrittDefault			X					
ZeitspanneEintrittNormal	X							
ZeitspanneVerlassenDefault				X				
ZeispanneVerlassenNormal		X						
ZeitspanneEintrittAufDefault()							X	
ZeitspanneEintrittNormalSetzen()								X
ZeitspanneVerlassenAufDefault()					X			
ZeitspanneVerlassenNormalSetzen()						X		

2.2.8.4 Klasse RainChecker

Die Klasse RainChecker verfügte in der Analyse weder über Methoden noch über Attribute.

2.2.8.5 Klasse Room

	Room		
	Flaeche	BenoetigteKesselTempAbfragen()	RegenZimmer()
Flaeche	X		
BenoetigteKesselTempAbfragen()		X	
RegenZimmer()			X

2.2.8.6 Klasse RoomtemperatureController

	RTC		Temperatureregulator	
	Zieltemperatur	ZieltemperaturSetzen()	Zimmertemperatur	ZimmertemperaturSetzen()
Zieltemperatur	X			
AbfrageZimmerTemperatur()			X	X
ZieltemperaturSetzen()		X		

2.2.8.7 Klasse Section

	Section	
	BenoetigteKesselTempAbfragen()	RegenAnZimmer()
BenoetigteKesselTempAbfragen()	X	
RegenAbschnitt()		X

2.2.8.8 Klasse Temperatureregulator

	Temperatureregulator																							
	AbwesenheitsTempNormal	AnwesenheitsTempNormal	ZeitspanneAufheizenNormal	BenoetigteKesseltemperatur	CZeitspanneAufheizenDefault	CAbwesenheitsTempDefault	CAnwesenheitsTempDefault	ZFenster	ZAnwesend	TemperaturStatus	ZeitspanneEintrittNormal	BenoetigteKesselTempLesen()	HalteTempBerechnen()	HeizTempBerechnen()	FensterzustandSetzen()	AnnwesenheitSetzen()	TemperaturStatusSetzen()	ZeitspanneAufheizenAufDefault()	ZeitspanneAufheizenNormalSetzen()	AnwesenheitsTempAufDefault()	AnwesenheitsTempNormalSetzen()	AbwesenheitsTempAufDefault()	AbwesenheitsTempNormalSetzen()	ZeitspanneEintrittNormalSetzen()
AbwesenheitsTempDefault						X																		
AbwesenheitsTempNormal	X																							
AnwesenheitsTempDefault							X																	
AnwesenheitsTempNormal		X																						
BenoetigteKesseltemperatur				X																				
Restzeit			X								X													X

	Temperatureregulator																							
	AbwesenheitsTempNormal	AnwesenheitsTempNormal	ZeitspanneAufheizenNormal	BenoetigteKesseltemperatur	CZeitspanneAufheizenDefault	CAbwesenheitsTempDefault	CAnwesenheitsTempDefault	ZFenster	ZAnwesend	TemperaturStatus	ZeitspanneEintrittNormal	BenoetigteKesselTempLesen()	HalteTempBerechnen()	HeizTempBerechnen()	FensterzustandSetzen()	AnnwesenheitSetzen()	TemperaturStatusSetzen()	ZeitspanneAufheizenAufDefault()	ZeitspanneAufheizenNormalSetzen()	AnwesenheitsTempAufDefault()	AnwesenheitsTempNormalSetzen()	AbwesenheitsTempAufDefault()	AbwesenheitsTempNormalSetzen()	ZeitspanneEintrittNormalSetzen()
ZeitspanneAufheizenDefault					X																			
ZeitspanneAufheizenNormal			X																					
AbwesenheitsTempAufDefault()																						X		
AbwesenheitsTempNormalSetzen()																							X	
AnwesenheitsTempAufDefault()																				X				
AnwesenheitsTempNormalSetzen()																					X			
BenoetigteKesselTempLesen()												X												
FensterGekippt()								X							X									
FensterGeschlossen()								X							X									
FensterOffen()								X							X									
HalteTempBerechnen()													X											
HeizTempBerechnen()														X										
JemandDa()									X							X								
NiemandDa()									X							X								
ZeitspanneAufheizenAufDefault()																		X						
ZeitspanneAufheizenNormalSetzen()																			X					
ZimmerTempGut()										X							X							
ZimmerTempZuHoch()										X							X							
ZimmerTempZuTief()										X							X							

2.2.8.9 Klasse Window

	Windo w		
	FensterKippen()	ZimmerKuehlen()	RegenFenster()
FensterKippen()	X		
RegenFenster()			X
ZimmerKuehlen()		X	

2.3 Komponenten-Anforderungen

2.3.1 Allgemeine Datentypen

2.3.1.1 Wiederverwendete Klassen

Dieser Abschnitt listet die Datantypen und Klassen auf, die im Rahmen früherer Projekte erstellt worden sind und hier wiederverwendet werden.

C++ Klassen und Typdefinitionen
typedef unsigned int timeCompType
durationType timeCompType

C++ Klassen und Typdefinitionen

```
Klasse Clock:
class Clock
{
  friend ostream& operator<< (ostream& os, const Clock &c);
  friend istream& operator>> (istream& is, Clock &c);
   // these operators must be friends because they need direct access to
the
 // data members or base classes

private:
  static const int ChoursPerDay;
  // this const stores the number of hours per day;
  // this value is used in some calculations in the class
  static const int CminutesPerHour;
  // this const stores the number of minutes per hour;
 // this value is used in some calculations in the class
  timeCompType hour, minute;
  // these members represent the main data components of this class,
  // storing a certain point in time

public:
  Clock (const timeCompType h, const timeCompType m);
  Clock (const Clock &c);
  Clock (const durationType dur);
  Clock (void);
  // ~Clock: no members need to be handled by destructor

  Clock& operator= (const Clock &c);
  // memberwise assignment

  bool operator< (const Clock &c) const;
  // c1 < c2 iff
 // c1.hour < c2.hour || (c1.hour == c2.hour && c1.minute < c2.minute)

  bool operator<= (const Clock &c) const;
  // c1 <= c2 iff c1 < c2 || c1 == c2

  bool operator> (const Clock &c) const;
  // c1 > c2 iff !(c1 <= c2)

  bool operator>= (const Clock &c) const;
  // c1 >= c2  iff !(c1 < c2)

  bool operator== (const Clock &c) const;
  // c1 == c2 iff c1.hour == c2.hour && c1.minute == c2.minute
```

C++ Klassen und Typdefinitionen

```
bool operator!= (const Clock &c) const;
  // c1 != c2 iff !(c1== c2)

  Clock operator+ (const Clock &c) const;
  // c3 = c1 + c2:
  //    c3.hour = c1.hour + c2.hour + minuteOverflow;
 //    c3.minute = (c1.minute + c2.minute) % CminutesPerHour

  Clock operator- (const Clock &c) const;
  // c3 = c1 - c2 computes time span from object c1 to object c2, e.g.:
  // 21:00 - 22:30 = 1:30
  // attention: 23:00 - 1:15 = 2:1;
  // so, this operator doesn‘t really implement would usually be expected
  // from the minus operator

  timeCompType getMinute (void) const;
  // returns value of member minute

  timeCompType getHour (void)const ;
  // returns returns value of member hour

  void getClock (timeCompType &h, timeCompType &m) const;
  // returns values of members minute and hour through the reference
  // parameters

  void setMinute (const timeCompType m);
  // sets member minute to new value

  void setHour (const timeCompType h);
  // sets member hour to new value

  void setClock (const timeCompType h, const timeCompType m);
  // sets members hour and minute to new values

  Clock getClock (void) const;
  // returns a copy of this object

  durationType toDuration (void) const;
  // converts the stored point in time to a duration;
  // therefore the following value is returned:
  //   hour * CminutesPerHour + minutes

  operator durationType () const;
  // type cast from Clock to durationType;
}; // class Clock
```

C++ Klassen und Typdefinitionen

```
Klasse Day:
class Day
{
  friend ostream& operator<< (ostream& os, const Day &d);
  friend istream& operator>> (istream& is, Day &d);
   // these operators must be friends because they need direct access to
the
  // data members or base classes

private:
  static const int daysPerMonthArray[2][13];
  // this array contains the number of days of every month in a year;
  // daysPerMonthArray[0] contains the values for non-leap-years and
  // daysPerMonthArray[1] for leap-years;
  // daysPerMonthArray[?][0] is not used; for the sake of simpler access
  // the values are stored in daysPerMonthArray[?][1] ... [12]

  timeCompType dayNo, month, year;
  // main data members of this class which represent a specific date

public:
  Day (const timeCompType d, const timeCompType m, const timeCompType y);
  Day (const Day &d);
  Day (void);
  // ~Day: no members need to be handled by destructor

  Day& operator= (const Day &d);
  // memberwise assignment

  bool operator< (const Day &d) const;
  // d1 < d2 iff
  // d1.year < d2.year
  // || ((d1.year == d2.year) && (d1.month < d2.month))
  // || ((d1.year == d2.year) && (d1.month == d2.month)
  // && (d1.dayNo < d2.dayNo))

  bool operator<= (const Day &d) const;
  // d1 <= d2 iff d1 < d2 || d1 == d2

  bool operator> (const Day &d) const;
  // d1 > d2 iff !(d1 <= d2)
  bool operator>= (const Day &d) const;
  // d1 >= d2 iff !(d1 < d2)

  bool operator== (const Day &d) const;
  // d1 == d2 iff d1.year == d2.year && d1.month == d2.month
  // && d1.dayNo == d2.dayNo
```

C++ Klassen und Typdefinitionen

```
bool operator!= (const Day &d) const;
// d1 != d2 iff !(d1 == d2)

Day operator++ (int);
// postfix version of operator ++; computes next day and stores
// it in the object; delivers a copy of the object as value

timeCompType getDay (void) const;
// returns value of member dayNo

timeCompType getMonth (void) const;
// returns value of member month

timeCompType getYear (void) const;
// returns value of member year

void getDayMonth (timeCompType &d, timeCompType &m) const;
// returns values of members dayNo and month through the reference
// parameters

void getDay (timeCompType &d, timeCompType &m, timeCompType &y) const;
// returns values of members dayNo, month and year through the reference
// parameters

void setDay (const timeCompType d);
// sets member dayNo to new value

void setMonth (const timeCompType m);
// sets member month to new value

void setYear (const timeCompType y);
// sets member year to new value

void setDay (const timeCompType d, const timeCompType m, const
             timeCompType y);
// sets members dayNo, month and year to new values

void setDayFromDayIndex (const int d, const timeCompType y);
// the day index is the number of a day in a year (1.1. is day 1 and
// 31.12. is day 365 or 366 in a leap year); the parameter year is
// necessary to be able to tell wether February has 28 or 29 days;
// the methods set the members dayNo, month and year according to the
// date specified by the parameters of the method
```

C++ Klassen und Typdefinitionen

```
  Day getDayObject (void) const;
  // return a copy of the object

  int getDayInYear (void) const;
  // computes and returns the day index of the date stored in the object

  bool isLeapYear (const timeCompType y) const;
  // true iff the year specified by the parameter is a leap year

  bool isLeapYear (void) const;
  // true iff the member year specifies a leap year

  int daysInMonth (const timeCompType m, const timeCompType y) const;
  // computes and returns the number of days the month specified by the
  // parameters m(onth) and y(ear) has; the parameter y is necessary in
  // order to take care of leap years

  int daysInMonth () const;
  // computes and returns the number of days the month specified by the
  // members month and year has
}; // class Day
```

C++ Klassen und Typdefinitionen

```
Klasse SListBase:

class SListBase {
        slink *last;
        slink *current;
        int  endofsearch;
        int length;

    public:
        void insert(slink *a);
        void append(slink *a);
        slink *get();

        void clear() { last = 0; } // nothing to delete ?
        SListBase()  { last = 0; length = 0; }
        SListBase( slink* a )  { last = a->next = a; length = 1;}
                        // stupid constructor ??

        void reset();
        slink *next();
        slink *replacecurr( slink* a );
        slink *getcurr();
        int isEmpty() const
                { return (last == 0); }
        void insBeforeCurr(slink *a);

        int getLength() { return length; }

}; // class SListBase
```

C++ Klassen und Typdefinitionen

```
Klasse Slist:
 template <class T> class SList : public SListBase {
  public:
  SList(){} // constructor
  SList(SList<T>& Source)
       // copy constructor
       // necessary if lists are member of lists
  {
    T tmp;
    this->clear();
    Source.reset();
    while (Source.next(tmp) == notempty) {
      append(tmp);
    } }

  ~SList() // destructor
  {
    T tmp;
    // first of all, destroy list-entries
    while (get(tmp) == notempty);
  }

  void insert(const T& a)
       // Insert at head of list
  { SListBase::insert(new Tlink<T>(a)) ; }

  void insertBeforeCurr(const T& a)
       // Insert at head of list
  { SListBase::insBeforeCurr(new Tlink<T>(a)) ; }

  SList<T>& operator=(SList<T>& Source)
       // Assignment of Lists
       // initially destroys destination list elements
       // assigns a list of copied(!) source list elements
  {
    T tmp;
    if (this != &Source) {
      // no self assignment
      // first of all, destroy list-entries
      while (get(tmp) == notempty);
      // Copy Source into destination
      Source.reset();
      while (Source.next(tmp) == notempty) {
        append(tmp);
      }
   }
    return *this;
  }
```

C++ Klassen und Typdefinitionen

```
void append(const T& a)
     // append to tail
{
  SListBase::append(new Tlink<T>(a));
}

SListError get(T& ret)
     // gets element and deletes it from List
     // returns empty if lidt is empty and notempty otherwise
{
  Tlink<T> *link;
  SListError RetVal;
  (slink *)link = SListBase::get();
  if (link) {
    ret = link->info;
    RetVal = notempty;
    delete link;
  } else {
    RetVal = empty;
  }
  return RetVal;
}

void reset()
     // to be called before list can be traversed using next()
     // („const‟ is actually a lie, inserted to specify that
     // traversing the list does not really modify it)
{ SListBase::reset(); }


int getLength()
     // returns length of list
{ return SListBase::getLength(); }

SListError next(T& ret)
     // gets element and from list without deleting them
     // returns empty if list is empty and notempty otherwise
     // („const‟ is actually a lie, inserted to specify that
     // traversing the list does not really modify it)
{
  Tlink<T> *link;
  SListError RetVal;
  (slink *)link = SListBase::next();
  if (link) {
    ret = link->info;
    RetVal = notempty;
  } else {
    RetVal = empty;
  }
  return RetVal;
}
```

C++ Klassen und Typdefinitionen

```
  void deletecurr()
        // deletes list element, that was
        // reached using next
  {
    Tlink<T> *tmp;
    (slink *)tmp = SListBase::getcurr();
    if (tmp) {
      delete tmp;
    } else {
      cerr << „deletecurr: attempt to empty list\n“;
    }
  }

  void replacecurr(T& a)
        // replaces list element, that was
        // reached using next, by a copy(!)
        // of element a
  {
    Tlink<T> *tmp;
    (slink *)tmp = SListBase::replacecurr(new Tlink<T>(a));
    if (tmp) delete tmp;
  }

}; // class Slist
```

C++ Klassen und Typdefinitionen

```
Klasse Time:
class Time : public Clock, public Day
{
  friend ostream& operator<< (ostream& os, const Time &t);
  friend istream& operator>> (istream& is, Time &t);
   // these operators must be friends because they need direct access to
the
  // data members or base classes

public:
  Time (const timeCompType h, const timeCompType mi,
       const timeCompType d, const timeCompType mo, const timeCompType y);
  Time (const Time &t);
  Time (void);
  // ~Time: no members need to be handled by destructor

  Time& operator= (const Time &t);
  // memberwise assignment

  bool operator< (const Time &t) const;
  // t1 < t2 iff t1.day < t2.day || (t1.day == t2.day && t1.time < t2.time)

  bool operator<= (const Time &t) const;
  // t1 <= t2 iff t1 < t2 || t1 == t2

  bool operator> (const Time &t) const;
  // t1 > t2 iff !(t1 <= t2)

  bool operator>= (const Time &t) const;
  // t1 >= t2 iff !(t1 < t2)

  bool operator== (const Time &t) const;
  // t1 == t2 iff t1.day == t2.day && t1.time == t2.time

  bool operator!= (const Time &t) const;
  // t1 != t2 iff !(t1 == t2)

  Time operator++ (int);
  // postfix version of operator ++; computes next minute; in case of an
  // overflow hours, days, .... are updated as well;
  // the changes are stored in the object and a copy of the unchanged
  // object is returned as the operator`s value

  void getTime (timeCompType &h, timeCompType &mi,
  timeCompType &d, timeCompType &mo, timeCompType &y) const;
   // returns all time components (from the day and time)in the value
result
```

C++ Klassen und Typdefinitionen

`// parameters`
 	
`void setTime (const timeCompType h, const timeCompType mi,`
`const timeCompType d, const timeCompType mo, const`
`timeCompType y);`
`// sets new values for the day and time`
`}; // class Time`

2.3.1.2 Datentypen aus dem UML-Entwurf

Die folgenden Datentypen basieren auf Typen und Klassen, die im UML-Entwurf verwendet werden.

C++ Typdefinition	Beschreibung
`typedef enum {CWertUngueltig, CKeinFehler} fehlerType`	Mögliche Rückgabewerte bei Fehlern.
`typedef enum {CJemandDa, CNiemandDa } zAnwesenheitType`	Mögliche Werte bei Benutzung von Anwesenheitswerten.
`typedef struct{ zAnwesenheitType ZAnwesenheitsKontakt; durationType ZeitDiff; } ZAnwesenheitZeitType`	Rückgabetyp des PresenceControllers bei Statusabfrage.
`typedef enum {CFensterGekippt, CFensterOffen, CFensterGeschlossen } zFensterType`	Mögliche Werte bei Benutzung des Fensterzustands.
`typedef enum {CZimmerTempZuTief, CZimmerTempGut, CZimmerTempZuHoch} zZimmerTempType`	Mögliche Werte für den Zimmertemperaturzustand.

2.3.2 Klassendefinitionen

Im folgenden werden zu jeder Klasse definiert:

- Assoziationen zu Objekten anderer Klassen

  Dabei werden Assoziationen nur in die Richtung, in die eine Klasse Methoden einer anderen aufruft, durch einen Zeiger auf diese modelliert.

- Aggregationen von Objekten zu anderen Klassen
- Methodenschnittstellen von Konstruktoren, Destruktoren oder privaten Methoden. Die public-Methoden der Klassen wurden bereits im Kapitel 2.2.3 beschrieben.
- Imports von anderen Klassen

2.3.2.1 Klasse CentralHeatingSite

2.3.2.1.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.1.2 Aggregationen von Objekten anderer Klassen

Keine

2.3.2.1.3 Methodenschnittstellen

Signatur: **CentralHeatingSite()**

Beschreibung: Konstruktor der Klasse.

2.3.2.1.4 Imports

Importierte Datentypen

Herkunft: SimpleTypes

- tempType

2.3.2.2 Klasse Clock

2.3.2.2.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.2.2 Aggregationen von Objekten anderer Klassen

Keine

2.3.2.2.3 Methodenschnittstellen

2.3.2.2.4 Imports

Keine.

2.3.2.3 Klasse Contact

2.3.2.3.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.3.2 Aggregationen von Objekten anderer Klassen

Keine

2.3.2.3.3 Methodenschnittstellen

Signatur: **Contact(SectionNr:int, RoomNr:int, PortNr:int)**

Beschreibung: Konstruktor der Klasse.

2.3.2.3.4 Imports

Keine.

2.3.2.4 Klasse ControlPanel

2.3.2.4.1 Assoziationen zu Objekten anderer Klassen

Pointername	Klassenname des referenzierten Objektes
TemperatureregulatorPtr	Temperatureregulator
PRTempControllerPtr	PRTempController

2.3.2.4.2 Aggregationen von Objekten anderer Klassen

Keine

2.3.2.4.3 Methodenschnittstellen

Signatur: **ControlPanel(TemperatureregulatorPtr:Temperatureregulator*, PRTempControllerPtr:PRTempController*)**

Beschreibung: Konstruktor der Klasse.

2.3.2.4.4 Imports

Importierte Methoden

Herkunft: Temperatureregulator

- ZeitspanneAufheizenNormalSetzen
- AnwesenheitsTempNormalSetzen
- AnwesenheitsTempAufDefault
- AbwesenheitsTempAufDefault
- ZeitspanneAufheizenAufDefault

Herkunft: PRTempController

- ZeitspanneEintrittNormalSetzen
- ZeitspanneEintrittAufDefault
- ZeitspanneVerlassenAufDefault

Importierte Datentypen

Herkunft: SimpleTypes

- fehlerType
- durationType
- tempType

2.3.2.5 Klasse Day

2.3.2.5.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.5.2 Aggregationen von Objekten anderer Klassen

Keine

2.3.2.5.3 Methodenschnittstellen

2.3.2.5.4 Imports

Keine.

2.3.2.6 Klasse House

2.3.2.6.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.6.2 Aggregationen von Objekten anderer Klassen

Pointername	Klassenname des aggregierten Objektes
SectionPtrListe	Section
ContactPtr	Contact
TemperaturesensorPtr	Temperaturesensor
CentralHeatingSitePtr	CentralHeatingSite

2.3.2.6.3 Methodenschnittstellen

Signatur: **House(SectionZimmerListe:SList<double>, ZeitPtr:Timer*)**

Beschreibung: Konstruktor der Klasse.

Signatur: **void RegenAnSection()**

Beschreibung: Meldet neu einsetzenden Regen an alle Sections.

2.3.2.6.4 Imports

Importierte Methoden

Herkunft: Section

- BenoetigteKesselTempAbfragen
- AussentemperaturWeiterleiten
- RegenAnZimmer

Herkunft: Contact

- AbfrageKontakt

Herkunft: Temperaturesensor

- AbfrageTemperatur

Herkunft: CentralHeatingSite

- KesseltemperaturSetzen

Importierte Datentypen
Herkunft: SimpleTypes

- tempType

2.3.2.7 Klasse Motor

2.3.2.7.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.7.2 Aggregationen von Objekten anderer Klassen

Keine

2.3.2.7.3 Methodenschnittstellen

Signatur: **Motor(SectionNr:int, RoomNr:int, PortNr:int)**

Beschreibung: Konstruktor der Klasse.

2.3.2.7.4 Imports

Importierte Datentypen
Herkunft: SimpleTypes

- tempType

2.3.2.8 Klasse PRTempController

2.3.2.8.1 Assoziationen zu Objekten anderer Klassen

Pointername	Klassenname des referenzierten Objektes
PresenceControllerPtr	PresenceController
TemperatureregulatorPtr	Temperatureregulator

2.3.2.8.2 Aggregationen von Objekten anderer Klassen

Keine

2.3.2.8.3 Methodenschnittstellen

Signatur: **PRTempController(PresenceControllerPtr:PresenceController*, TemperatureregulatorPtr:Temperatureregulator*)**

Beschreibung: Konstruktor der Klasse.

2.3.2.8.4 Imports

Importierte Methoden

Herkunft: PresenceController

- StatusAktualisieren

Herkunft: Temperatureregulator

- AnwesenheitSetzen

Importierte Datentypen

Herkunft: SimpleTypes

- durationType
- zAnwesenheitZeitType
- zAnwesenheitType

2.3.2.9 Klasse Portbaustein

2.3.2.9.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.9.2 Aggregationen von Objekten anderer Klassen

Keine

2.3.2.9.3 Methodenschnittstellen

Signatur: **int CallBitInputPort(SectionNr:int, RoomNr:int, PortNr:int)**

Beschreibung: Liest einen Wert von einem Sensor ein.

Signatur: **void CallBitOutputPort(SectionNr:int, RoomNr:int, PortNr:int, value:int)**

Beschreibung: Schickt einen Wert an einen Aktor.

Signatur: **int CallIntInputPort(SectionNr:int, RoomNr:int, PortNr:int)**

Beschreibung: Liest einen Wert von einem Sensor ein.

Signatur: **void CallIntOutputPort(SectionNr:int, RoomNr:int, PortNr:int, value:int)**

Beschreibung: Schickt einen Wert an einen Aktor.

2.3.2.9.4 Imports

Keine.

2.3.2.10 Klasse PresenceController

2.3.2.10.1 Assoziationen zu Objekten anderer Klassen

Pointername	Klassenname des referenzierten Objektes
ZeitPtr	Timer

2.3.2.10.2 Aggregationen von Objekten anderer Klassen

Pointername	Klassenname des aggregierten Objektes
ContactPtr	Contact

2.3.2.10.3 Methodenschnittstellen

Signatur: **PresenceController(InitialSectionNr:int, InitialRoomNr:int, ZeitPtr:Timer*)**

Beschreibung: Konstruktor der Klasse.

2.3.2.10.4 Imports

Importierte Methoden

Herkunft: Contact

- AbfrageKontakt

Importierte Datentypen

Herkunft: SimpleTypes

- durationType
- zAnwesenheitsType
- zAnwesenheitZeitType

2.3.2.11 Klasse Radiator

2.3.2.11.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.11.2 Aggregationen von Objekten anderer Klassen

Keine

2.3.2.11.3 Methodenschnittstellen

Signatur: **Radiator(SectionNr:int, RoomNr:int)**

Beschreibung: Konstruktor der Klasse.

2.3.2.11.4 Imports

Importierte Datentypen

Herkunft: SimpleTypes

- tempType

2.3.2.12 Klasse Room

2.3.2.12.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.12.2 Aggregationen von Objekten anderer Klassen

Pointername	Klassenname des aggregierten Objektes
PresenceControllerPtr	PresenceController
WindowPtr	Window
TemperatureregulatorPtr	Temperatureregulator
PRTempControllerPtr	PRTempController
ControlPanelPtr	ControlPanel

2.3.2.12.3 Methodenschnittstellen

Signatur: **Room(InitialSectionNr:int, InitialRoomNr:int, Flaeche:double, ZeitPtr:Timer*)**

Beschreibung: Konstruktor der Klasse.

2.3.2.12.4 Imports

Importierte Methoden

Herkunft: Window

- RegenFenster

Herkunft: Temperatureregulator

- AussentemperaturSetzen
- FensterzustandSetzen
- BenoetigteKesselTempLesen

Importierte Datentypen

Herkunft: SimpleTypes

- tempType
- zFensterType

2.3.2.13 Klasse RoomtemperatureController

2.3.2.13.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.13.2 Aggregationen von Objekten anderer Klassen

Pointername	Klassenname des aggregierten Objektes
TemperaturesensorPtr	Temperaturesensor

2.3.2.13.3 Methodenschnittstellen

Signatur: **RoomtemperatureController(InitialSectionNr:int, InitialRoomNr:int, TemperatureregulatorPtr:Temperatureregulator)**

Beschreibung: Konstruktor der Klasse.

2.3.2.13.4 Imports

Importierte Methoden

Herkunft: Temperatureregulator

- TemperaturStatusSetzen

Herkunft: Temperaturesensor

- AbfrageTemperatur

Importierte Datentypen

Herkunft: SimpleTypes

- tempType
- zZimmerTempType

2.3.2.14 Klasse ScheduledFct

2.3.2.14.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.14.2 Aggregationen von Objekten anderer Klassen

Keine

2.3.2.14.3 Methodenschnittstellen

Signatur: **void registerCallBack()**

Beschreibung: Registriert das Objekt beim Scheduler für die periodische Aktivierung.

Signatur: **void unregisterCallBack()**

Beschreibung: Stoppt die periodische Ausführung des Objekts durch den Scheduler.

Signatur: **void callBack()**

Beschreibung: Realisiert die Methode, die durch den Scheduler periodisch aktiviert wird.

2.3.2.14.4 Imports

Keine.

2.3.2.15 Klasse Scheduler

2.3.2.15.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.15.2 Aggregationen von Objekten anderer Klassen

Keine

2.3.2.15.3 Methodenschnittstellen

Signatur: **void registerCallBack()**

Beschreibung: Fügt eine CallBack-Routine zur periodischen Ausführung in den Scheduler ein.

Signatur: **void unregisterCallBack()**

Beschreibung: Löscht die entsprechende CallBack-Methode aus der Scheduler-Liste.

2.3.2.15.4 Imports

Keine.

2.3.2.16 Klasse Section

2.3.2.16.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.16.2 Aggregationen von Objekten anderer Klassen

Pointername	Klassenname des aggregierten Objektes
RoomPtrListe	Room

2.3.2.16.3 Methodenschnittstellen

Signatur: **Section(InitialSectionNr:int, ZimmerFlaechenListe:SList<double>, ZeitPtr:Timer*)**

Beschreibung: Konstruktor der Klasse.

2.3.2.16.4 Imports

Importierte Methoden

Herkunft: Room

- AussentemperaturWeiterleiten
- RegenZimmer
- BenoetigteKesselTempAbfragen

Importierte Datentypen

Herkunft: SimpleTypes

- tempType

2.3.2.17 Klasse Temperatureregulator

2.3.2.17.1 Assoziationen zu Objekten anderer Klassen

Pointername	Klassenname des referenzierten Objektes
WindowPtr	Window

2.3.2.17.2 Aggregationen von Objekten anderer Klassen

Pointername	Klassenname des aggregierten Objektes
RoomtemperatureControl-lerPtr	RoomtemperatureController
RadiatorPtr	Radiator

2.3.2.17.3 Methodenschnittstellen

Signatur: **Temperatureregulator(InitialSectionNr:int, InitialRoomNr:int, Flaeche:double, WindowPtr:Window*)**

Beschreibung: Konstruktor der Klasse.

Signatur: **void HalteTempBerechnen()**

Beschreibung: Berechnet die Haltetemperatur. Benutzt dafür die in den entsprechenden Attributen gespeicherte Aussentemperatur, Zimmertemperatur, Fensterzustand, benoetigteKesseltemperatur, ZeitspanneEintrittNormal, ZeitspanneAufheizenNormal sowie den Wert den Attributs Flaeche.

Signatur: **void HeizTempBerechnen()**

Beschreibung: Berechnet die Aufheiztemperatur. Benutzt dafür die in den entsprechenden Attributen gespeicherte Aussentemperatur, Zimmertemperatur, Fensterzustand, ZeitspanneEintrittNormal, ZeitspanneAufheizenNormal sowie den Wert des Attributs Flaeche.

2.3.2.17.4 Imports

Importierte Methoden

Herkunft: RoomtemperatureController

- ZieltemperaturSetzen

Herkunft: Window

- ZimmerKuehlen

Herkunft: Radiator

- ThermostatSetzen

Importierte Datentypen

Herkunft: SimpleTypes

- durationType
- zAnwesenheitType
- tempType
- zFensterType
- fehlerType
- zZimmerTempType

2.3.2.18 Klasse Temperaturesensor

2.3.2.18.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.18.2 Aggregationen von Objekten anderer Klassen

Keine

2.3.2.18.3 Methodenschnittstellen

Signatur: **Temperaturesensor(SectionNr:int, RoomNr:int, PortNr:int)**

Beschreibung: Konstruktor der Klasse.

2.3.2.18.4 Imports

Importierte Datentypen

Herkunft: SimpleTypes

- tempType

2.3.2.19 Klasse Time

2.3.2.19.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.19.2 Aggregationen von Objekten anderer Klassen

Keine

2.3.2.19.3 Methodenschnittstellen

2.3.2.19.4 Imports

Keine.

2.3.2.20 Klasse Timer

2.3.2.20.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.20.2 Aggregationen von Objekten anderer Klassen

Keine

2.3.2.20.3 Methodenschnittstellen

2.3.2.20.4 Imports

Keine.

2.3.2.21 Klasse Window

2.3.2.21.1 Assoziationen zu Objekten anderer Klassen

Keine

2.3.2.21.2 Aggregationen von Objekten anderer Klassen

Pointername	Klassenname des aggregierten Objektes
MotorPtr	Motor
ContactPtr	Contact

2.3.2.21.3 Methodenschnittstellen

Signatur: **Window(InitialSectionNr:int, InitialRoomNr:int, RoomPtr:Room*)**

Beschreibung: Konstruktor der Klasse.

Signatur: **void FensterKippen()**

Beschreibung: Schließt ggf. den Offenkontakt und öffnet anschließend den Kippkontakt mit Hilfe des Kipp- und des Schwenkmotors.

2.3.2.21.4 Imports

Importierte Methoden

Herkunft: Room

- Fensterzustand

Herkunft: Motor

- Oeffnen
- Schliessen

Herkunft: Contact

- AbfrageKontakt

Importierte Datentypen

Herkunft: SimpleTypes

- zFensterType

2.4 Verifikation

Es wurde keine explizite Verifikation im Sinne einer Überprüfung anhand von Checklisten vorgenommen. Die Verifikation der Software-Systemdokumentation war ein kontinuierlicher Prozeß und fand in Form von Besprechungen statt.

2.5 Validation

2.5.1 Komponententestfälle

Testende-Kriterium

Bei der Validation der Komponenten werden folgende Testende-Kriterien zugrunde gelegt:

1. Überdeckung der Kanten im Statemodell der Klasse.
2. Bei Klassen die kein State-Modell besitzen, weil sie nur zum Setzen und Abfragen von Werten dienen, sowie bei den Methoden der unter Punkt 1. fallenden Klassen, die nicht durch die Überdeckung des State-Diagramms erfaßt werden, sind die Testfälle folgendermaßen zu kreieren bzw. zu erweitern:
    - Es werden alle definierten Rückgabewerte der einzelnen Methoden wenigstens einmal angenommen.
    - Methoden ohne Rückgabewert werden wenigstens einmal aufgerufen.
    - Methoden, die zum Setzen von Werten dienen, werden dadurch überprüft, indem die entsprechenden Werte anschließend direkt oder indirekt wieder abgefragt werden. Sollte eine Abfrage des gerade gesetzten Wertes nicht möglich sein, so wird die Methode nicht getestet.
    - Methoden, die zum Abfragen von Werten dienen, werden dadurch überprüft, indem die Werte, von denen die Rückgabewerte der Methoden abhängen, vorher gezielt gesetzt worden sind oder, wenn der Rückgabewert von externen Parametern abhängt, eben diese entsprechend gewählt werden.

Bemerkungen

- Die Zahlen, die in Klammern hinter den Methoden angegeben ist, stellt den bei der Durchführung des Test erwarteten Ausgabewert dar (z.B. JemandDa ist gleichbedeutend mit 0).

2.5.1.1 Klasse House

Folgende Stubs sind für den Test erforderlich:

- CentralHeatingSite
    - void Kesseltemperatursetzen (tempType Temperatur)
- Temperaturesensor
    - tempType AbfrageTemperatur()
- Contact
    - int AbfrageKontakt()
- Section
    - Section (InitialSectionNr, SList<double> ZimmerFlaechenListe, Timer* Zeit)
    - void RegenAnZimmer()

- AussentemperaturWeiterleiten (tempType Temperatur)
- tempType BenoetigteKesselTempAbfragen()

KompTest1: Es beginnt zu regnen. Parallel dazu wird die Maximal benötigte Kesseltemperatur durch das Haus bestimmt und die Aussentemperatur regelmäßig abgefragt.

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
	House(SectionZimmer-Liste,ZeitPtr)			Init -> keinRegen
1.	callBack()	->Section.Benoetigte-KesselTempAbfragen()	Section.BenoetigteKesselTempAbfragen() <- 15	keinRegen -> keinRegen
		->CentralHeating-Site.Kesseltemperatur-Setzen(15)	CentralHeating-Site.Kesseltemperatur-Setzen() <- Ausgabe: Kesseltemp auf 15 Grad gesetzt	
		->Aussentemperatur-fuehler.AbfrageTemperatur()	Aussentemperaturfuehler.AbfrageTemperatur() <- 15	
		->Section.AussentemperaturWeiterleiten(15)	Section.AussentemperaturWeiterleiten() <- Ausgabe: Aussentemperatur 15 Grad	
		->Regenkontakt.AbfrageKontakt()	Regenkontakt.Abfrage-Kontakt() <- geschlossen (0)	
2.	callBack()	->Section.Benoetigte-KesselTempAbfragen()	Section.BenoetigteKesselTempAbfragen() <- 15	keinRegen -> Regen
		->Aussentemperatur-fuehler.AbfrageTemperatur()	Aussentemperatur.AbfrageTemperatur() <- 15	
		->Section.RegenAn-Zimmer()	Section.RegenAnZimmer() <- Ausgabe: Regenmeldung an Section	

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
3.	callBack()	->Section.Benoetigte-KesselTempAbfragen()	Section.BenoetigteKes-selTempAbfragen() <- 15	Regen -> Regen
		->Aussentemperatur-fuehler.AbfrageTempe-ratur()	Aussentemperaturfueh-ler.AbfrageTempera-tur() <- 15	
		->Regenkontakt.Abfra-geKontak()	Regenkontakt.Abfrage-Kontakt() <- offen (1)	
4.	callBack()	->Section.Benoetigte-KesselTempAbfragen()	Section.BenoetigteKes-selTempAbfragen() <- 15	Regen -> keinRegen
		->Aussentemperatur-fuehler.AbfrageTempe-ratur()	Aussentemperaturfueh-ler.AbfrageTempera-tur() <- 15	
		->Regenkontakt.Abfra-geKontakt()	Regenkontakt.Abfrage-Kontakt() <- offen (1)	

2.5.1.2 Klasse PresenceController

Folgende Stubs sind für den Test erforderlich:

- Contact
  - Contact (int SectionNr, int RoomNr, int PortNr)
  - int AbfrageKontakt()
- Timer
  - Time getActualTime()

KompTest2: Benutzer betritt das Zimmer und verläßt es wieder nach einiger Zeit..

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
	Presence-Controller(InitialSectionNr, InitialRoomNr, Zeit)	->Verlassen-Zeit=Zeit.getActual-Time() ->Eintritt-Zeit=Zeit.getActual-Time()	Zeit.getActualTime() <-(0,0,1,1,1999) Zeit.getActualTime() <-(0,0,1,1,1999)	Init -> NiemandAnwesend
1.	StatusAktualisieren()	->Zeit.getActualTime()	Zeit.getActualTime() <- (0,1,1,1,1999)	
		Ausgabe : CNiemandDa seit 1 Minuten		
2.	callBack()	->Anwesenheitskontakt.AbfrageKontakt()	Anwesenheitskontakt.AbfrageKontakt() <- geschlossen (0)	NiemandAnwesend -> NiemandAnwesend
3.	StatusAktualisieren()	->Zeit.getActualTime()	Zeit.getActualTime() <- (0,5,1,1,1999)	
		Ausgabe: CNiemandDa seit 5 Minuten		
4.	callBack()	->Zeit.getActualTime()	Zeit.getActualTime() <- (0,10,1,1,1999)	NiemandAnwesend -> JemandAnwesend
5.	StatusAktualisieren	->Zeit.getActualTime()	Zeit.getActualTime() <- (0,12,1,1,1999)	
		Ausgabe: CJemandDa seit 2 Minuten		
6.	callBack()	->Anwesenheitskontakt.AbfrageKontakt()	Anwesenheitskontakt.AbfrageKontakt() <- offen (1)	JemandAnwesend -> JemandAnwesend
7.	callBack()	->Zeit.getActualTime()	Zeit.getActualTime() <- (0,20,1,1,1999)	JemandAnwesend -> NiemandAnwesend

2.5.1.3 Klasse PRTempController

Folgende Subs sind für den Test erfoderlich:

- PresenceController
  - zAnwesenheitZeitType StatusAktualisieren()
- Temperatureregulator
  - void AnwesenheitSetzen(zAnwesenheitType Anwesenheit)

- void ZeitspanneEintrittNormalSetzen(durationType Zeitspanne)

KompTest3: Benutzer betritt das Zimmer und verläßt es wieder nach einiger Zeit. Zuvor setzt er noch die ZeitspanneEintrittNormal auf 21 Minuten.

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
	PRTempcontroller(PresenceControllerPtr,TemperatureregulatorPtr)			Init -> NiemandDa
1.	callBack()	->PresenceController.StatusAktualisieren()	PresenceController.StatusAktualisieren() <- (CJemandDa,5)	NiemandDa -> NiemandDa
2.	callBack()	->PresenceController.StatusAktualisieren()	PresenceController.StatusAktualisieren() <- (CJemandDa,20)	NiemandDa -> NiemandDa
3.	callBack()	->Temperatureregulator.AnwesenheitSetzen(CJemandDa)	Tempratureregulator.AnwesenheitSetzen() <- Ausgabe: Status auf Anwesenheit gesetzt	NiemandDa -> JemandDa
4.	ZeitspanneEintrittNormalSetzen (21)	->Temperatureregulator.ZeitspanneEintrittNormalSetzen(21)	Temperatureregulator.ZeitspanneEintrittNormalSetzen() <- Ausgabe ZeitspanneEintrittNormal auf 21 Minuten	
5.	callBack()	->PresenceController.StatusAktualisieren()	PresenceController.StatusAktualisieren() <- (CNiemandDa, 1)	JemandDa -> JemandDa
6.	callBack()	->PresenceController.StatusAktualisieren()	PresenceController.StatusAktualisieren() <- (CNiemandDa, 10)	JemandDa -> JemandDa
7.	callBack()	->Temperauremregulator.AnwesenheitSetzen(CNiemandDa)	Temperatureregulator.AnwesenheitSetzen() <- Ausgabe: Status auf Abwesenheit gesetzt	JemandDa ->NiemandDa

2.5.1.4 Klasse ControlPanel

Folgende Stubs sind für den Test erforderlich:

- Temperatureregulator
  - fehlerType ZeitspanneAufheizenAufDefault()
  - fehlerType ZeitspanneAufheizenNormalSetzen (durationType Zeitspanne)
  - void AnwesenheitsTempAufDefault()
  - void AnwesenheitsTempNormalSetzen (tempType Temperatur)
  - void AbwesenheitsTempAufDefaul()
  - void AbwesenheitsTempNormalSetzen (tempType Temperatur)
- PRTempController
  - void ZeitspanneVerlassenAufDefault()
  - void ZeitspanneVerlassenNormalSetzen (durationType Zeitspanne)
  - void ZeitspanneEintrittAufDefault()
  - void ZeitspanneEintrittNormalSetzen (durationType Zeitspanne)

KompTest4: In diesem Testfall werden alle Methoden gemäß den oben unter 2. beschriebenden Kriterien aufgerufen.

Schritt	**Vom Treiber aufgerufene Methoden der zu testenden Klasse**	**Erwartetes Verhalten der zu testenden Klasse**
	ControlPanel (TemperatureregulatorPtr,PRTempControllerPtr)	
1.	->ZeitspanneAufheizenAufDefault()	Temperatureregulator.ZeitspanneAufheizenAufDefault() <- Ausgabe: ZeitspanneAufheizen auf Default gesetzt
2.	->ZeitspanneAufheizenNormalSetzen(25)	Temperatureregulator.ZeitspanneAufheizenNormalSetzen() <- Ausgabe: Zeitspanne Aufheizen auf 25 Minuten gesetzt
3.	->AnwesenheitsTempAufDefault()	Temperatureregulator.AnwesenheitsTempAufDefault() <- Ausgabe: AnwesenheitsTemp auf Default gesetzt.
4.	->AnwesenheitsTempNormalSetzen(24)	Temperatureregulator.AnwesenheitsTempNormalSetzen() <- Ausgabe: AnwesenheitsTemp auf 24 Grad gesetzt.
5.	->AbwesenheitsTempAufDefault()	Temperatureregulator.AbwesenheitsTempAufDefault() <- Ausgabe: AbwesenheitsTemp auf Default gesetzt.

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse
6.	->AbwesenheitsTempNormalSetzen(24)	Temperatureregulator.AbwesenheitsTemp-NormalSetzen() <- Ausgabe: Abwesenheits-Temp auf 24 Grad gesetzt.
7.	->ZeitspanneVerlassenAufDefault()	PRTempController.ZeitspanneVerlassenAuf-Default() <- Ausgabe: ZeitspanneVerlassen auf Default gesetzt.
8.	->ZeitspanneVerlassenNormalSetzen(5)	PRTempController.ZeitspanneVerlassenNor-malSetzen() <- Ausgabe: ZeitspanneVerlas-sen auf 5 Minuten gesetzt.
9.	->ZeitspanneEintrittAufDefault()	PRTempController.ZeitspanneEintrittAufDe-fault() <- Ausgabe: ZeitspanneEintritt auf Default gesetzt.
10.	->ZeitspanneEintrittNormalSetzen(10)	PRTempController.ZeitspanneEintrittNor-malSetzen() <- Ausgabe: ZeitspanneEintritt auf 5 Minuten gesetzt.

2.5.1.5 Klasse Room

Folgende Stubs sind für den Test erforderlich:

- ControlPanel
  - ControlPanel (Temperatureregulator *TempregPtr, PRTempController *PRTempControllerPtr)
- PresenceController
  - PresenceController (int SectionNr, int RoomNr, Timer* Zeit)
- PRTempController
  - PRTempController (PresenceController* PresenceControllerPtr, Temperatureregualtr* Temperatureguraltor Ptr)
- Temperatureregulator
  - Temperatureregulator (int SectionNr, int RoomNr, double Flaeche, Window* InitialWindowPtr)
  - tempType BenoetigteKesselTempLesen()
  - FensterzustandSetzen (zFensterType Zustand)
  - AussentemperaturWeiterleiten (tempType Temperatur)
- Window
  - Window (SectionNr, RoomNr, Room* RoomPtr)
  - RegenFenster()

KompTest5: In diesem Testfall werden alle Methoden gemäß den oben unter 2. beschriebenden Kriterien aufgerufen

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse
	Room (SectionNr, double Flaeche, Timer* Zeit)	
1.	->BenoetigteKesselTempAbfragen()	Temperatureregulator.BenoetigteKessel-TempLesen() <- 20, Ausgabe: KesselTemp Abgefragt.
2.	->Fensterzustand(CFensterOffen)	Temperatureregulator.FensterzustandSet-zen() <- Ausgabe: Fensterzustand: Offen
3.	->AussentemperaturWeiterleiten(12)	Temperatureregulator.AussentemperaturSet-zen() <-Ausgabe: Aussentemperatur 12 Grad
4.	->RegenZimmer()	Window.RegenFenster() <- Ausgabe: Regen an Fenster gemeldet.

2.5.1.6 Klasse RoomtemperatureController

Folgende Stubs sind für den Test erforderlich:

- Temperatureregulator
  - void ZimmertemperaturSetzen (tempType Temperatur)
  - void TemperaturStatusSetzen (zZimmerTempType)
- Temperaturesensor
  - Temperaturesensor (int SectionNr, int RommNr, int PortNr)
  - tempType AbfrageTemperatur()

KompTest6: Die Zimmertemperatur überschreitet die Zieltemperatur. Danach sinkt die Zimmertemperatur unter die Zieltemperatur.

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
	RoomtemperatureController(SectionNr, RoomNr, Temperaturregualtor Ptr)			Init -> Gut
1.	callBack()	->Zimmertemperaturfuehler.AbfrageTemperatur()	Zimmertemperaturfuehler.AbfrageTemperatur() <- 28	Gut -> Gut
2.	callBack()	-> Temperatureregulator.ZimmertemperaturSetzen(28)	Temperatureregulator.ZimmertemperaturSetzen() <- Ausgabe: Zimmertemperatur auf 28 Grad	Gut -> ZuHoch
		-> Temperatureregulator.TemperaturStatusSetzen(CZimmerTemp ZuHoch)	Temperatureregualtor.ZimmertemperaturSetzen() <- Ausgabe: Zimmertemperaturstatus: ZuHoch	
3.	callBack()	->Zimmertemperaturfuehler.AbfrageTemperatur()	Zimmertemperaturfuehler.AbfrageTemperatur() <- 10	ZuHoch -> ZuHoch
4.	callBack()	-> Temperatureregulator.ZimmertemperaturSetzen(10)	Temperatureregualtor.ZimmertemperaturSetzen() <- Ausgabe: Zimmertemperatur auf 10 Grad	ZuHoch -> ZuTief
		-> Temperatureregulator.TemperaturStatusSetzen(CZimmerTemp ZuTief)	Temperatureregualtor.TemperaturStatusSetzen() <- Ausgabe: Zimmertemperaturstatus: ZuTief	
5.	callBack()	->Zimmertemperaturfuehler.AbfrageTemperatur()	Zimmertemperaturfuehler.AbfrageTemperatur() <- 15	ZuTief -> ZuTief

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
6.	callBack()	-> Temperaturregulator.Zimmertemperatur-Setzen(15)	Temperaturregualtor.Zimmertemperatur-Setzen() <- Ausgabe: Zimmertemperatur auf 15 Grad	ZuTief -> Gut
		-> Temperaturregulator.TemperaturStatus-Setzen(CZimmerTemp Gut)	Temperaturregualtor.TemperaturStatus-Setzen() <- Ausgabe: Zimmertemperaturstatus: Gut	

KompTest7: Die Zimmertemperatur sinkt unter die Zieltemperatur. Danach steigt die Zimmertemperatur über die Zieltemperatur.

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
	Rommtem-peratureController(SectionNr, RoomNr, TemperaturregualtorPtr)			Init -> Gut
1.	callBack()	->Zimmertemperatur-fuehler.AbfrageTemperatur()	Zimmertemperatur-fuehler.AbfrageTemperatur() <- 10	Gut -> Gut

Schritt	**Vom Treiber aufgerufene Methoden der zu testenden Klasse**	**Erwartetes Verhalten der zu testenden Klasse**	**Gewünschte Reaktion der zu erstellenden Stubs**	**Überdeckte Kante im Statemodel**
2.	callBack()	-> Temperaturregulator.Zimmertemperatur-Setzen(10)	Temperaturregulator.Zimmertemperatur-Setzen() <- Ausgabe: Zimmertemperatur auf 10 Grad	Gut -> ZuTief
		-> Temperaturregulator.TemperaturStatus-Setzen(CZimmerTemp ZuHoch)	Temperaturregualtor.Zimmertemperatur-Setzen() <- Ausgabe: Zimmertemperaturstatus: ZuTief	
3.	callBack()	->Zimmertemperatur-fuehler.AbfrageTemperatur()	Zimmertemperatur-fuehler.AbfrageTemperatur() <- 20	ZuTief -> ZuTief
4.	callBack()	-> Temperaturregulator.Zimmertemperatur-Setzen(20)	Temperaturregualtor.Zimmertemperatur-Setzen() <- Ausgabe: Zimmertemperatur auf 20 Grad	ZuTief -> ZuHoch
		-> Temperaturregulator.TemperaturStatus-Setzen(CZimmerTemp ZuHoch)	Temperaturregualtor.TemperaturStatus-Setzen() <- Ausgabe: Zimmertemperaturstatus: ZuHoch	
5.	callBack()	->Zimmertemperatur-fuehler.AbfrageTemperatur()	Zimmertemperatur-fuehler.AbfrageTemperatur() <- 15	ZuHoch -> ZuHoch
6.	callBack()	-> Temperaturregulator.Zimmertemperatur-Setzen(15)	Temperaturregualtor.Zimmertemperatur-Setzen() <- Ausgabe: Zimmertemperatur auf 15 Grad	ZuHoch -> Gut
		-> Temperaturregulator.TemperaturStatus-Setzen(CZimmerTemp Gut)	Temperaturregualtor.TemperaturStatus-Setzen() <- Ausgabe: Zimmertemperaturstatus: Gut	

2.5.1.7 Klasse Temperaturregulator

Folgende Stubs sind für den Test erforderlich:

- RoomtemperatureController
  - RoomtemperatureController (SectionNr, RoomNr, TemperatureregualtorPtr)
  - void ZieltemperaturSetzen(tempType Temperatur)
- Radiator
  - Radiator (SectionNr, RoomNr, PortNr)
  - void ThermostatSetzen (tempType Temperatur)
- Window
  - void ZimmerKuehlen()

KompTest8: Zunächst muß das Zimmer aufgeheizt werden. Die Aufheiztemperatur wird berechnet. Anschließend wird die Zieltemperatur erreicht und die Haltetemperatur wird berechnet.
Eine Person betritt das Zimmer, oeffnet es und kippt es anschließend.

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
	Temperatur-eregualtor (SectionNr, RoomNr, Flaeche, WindowPtr)			Temperaturregelung: Init -> Halten
				AnAbwesenheit: Init -> Abwesenheit
1.	Zimmertem-peraturSet-zen(10)	-	-	-
2.	Temperatur-StautsSet-zen(CZimme rTempZu-Tief)	-	-	-
3.	callBack()	-> Radiator.Thermo-statSetzen ()	Radiator.Thermostat-Setzen <- Ausgabe: Thermostat auf 45 Grad gesetzt	Temperaturregelung: Halten -> Heizen
4.	Zimmertem-peraturSet-zen(15)	-	-	-

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
5.	TemperaturStatusSetzen(CZimmerTempGut)	-	-	-
6.	callBack()	-> Radiator.ThermostatSetzen()	Radiator.ThermostatSetzen <- Ausgabe: Thermostat auf 15 Grad gesetzt	Temperaturregelung: Heizen -> Halten
7.	AnwesenheitSetzen(CJemandDa)	-	-	-
8.	callBack()	-> RoomtemperatureController.ZieltemperaturSetzen(20)	-> RoomtemperatureController.ZieltemperaturSetzen <- Ausgabe: Zieltemperatur 20 Grad	AnAbwesenheit: Abwesenheit -> Anwesenheit Init->AnwesenheitFensterNichtOffen
9.	FensterzustandSetzen(CFensterOffen)	-	-	-
10.	callBack()	-> RoomtemperatureController.ZieltemperaturSetzen(15)	->RoomtemperatureController.ZieltemperaturSetzen <- Ausgabe: Zieltemperatur 15 Grad	Anwesenheit: AnwesenheitFensterNichtOffen -> AnwesenheitFensterOffen
11.	FensterzustandSetzen(CFensterGekippt)	-	-	-
12.	callBack()	-> RoomtemperatureController.ZieltemperaturSetzen(20)	-> RoomtemperatureController.ZieltemperaturSetzen <- Ausgabe: Zieltemperatur 20 Grad	Anwesenheit: AnwesenheitFensterOffen -> AnwesenheitFensterNichtOffen

KompTest9: Ein Benutzer betritt das Zimmer. Dadurch muß das Zimmer gekühlt werden. Der

Benutzer verläßt das Zimmer.

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
	Temperatureregualtor(SectionNr, RoomNr, Flaeche, WindowPtr)			Temperaturregelung: Init -> Halten
				AnAbwesenheit: Init -> Abwesenheit
1.	AnwesenheitSetzen(CJemandDa)	-	-	-
2.	AussentemperaturSetzen(28)	-	-	-
3.	ZimmertemperaturSetzen(25)	-	-	-
4.	TemeraturStatusSetzen(CZimmerTempZuHoch)	-	-	-
5.	callBack()	-> Radiator.ThermostatSetzen(0)	Radiator.ThermostatSetzen <- Ausgabe: Thermostat auf 0 Grad	Temperaturregelung: Halten -> Kuehlen Kuehlen: Init-> AnwesenheitTemperatur_pruefen
		-> Roomtemperature-Controller.ZieltemperaturSetzen(20)	RoomtemeratureController.ZieltemperaturSetzen() <- Ausgabe: Zieltemeratur 20 Grad	AnAbwesenheit: Abwesenheit -> Anwesenheit
6.	AussentemperaturSetzen(18)	-	-	-
7.	callBack()	-> Window.ZimmerKuehlen()	Window.ZimmerKuehlen() <- Ausgabe: Zimmer wird gekuehlt	Kuehlen: AnwesenheitTemperatur_pruefen -> Ende

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
8.	Fensterzu-standSet-zen(CFenster Gekippt)	-	-	-
9.	Zimmertem-peraturSet-zen(20)	-	-	-
10.	Temperatur-StatusSet-zen(CZimme rTempGut)	-	-	-
11.	callBack()	-> Radiator.Thermo-statSetzen()	Radiator.Thermostat-Setzen <- Ausgabe: Thrmostat auf 15 Grad gesetzt	Temperaturregelung: Kuehlen -> Halten
12.	Anwesen-heitSet-zen(CNiema ndDa)	-	-	-
13.	callBack()	-> Roomtemperature-Controller.Zieltempera-turSetzen(15)	RoomtemperatureCon-troller.Zieltemperatur-Setzen() <- Zieltemperatur auf 15 Grad gesetzt	AnAbwesenheit: Anwesenheit -> Abwe-senheit

KompTest10:Eine Person betritt das Zimmer. Das Zimmer muß aufgeheizt werden. Der Benutzer öffnet das Fenster, das Zimmer muß gekühlt werden. Anschließend wird das

Fenster wieder geschlossen und das Zimmer kann wieder aufgeheizt werden.

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
	Temperatur-eregualtor (SectionNr, RoomNr, Flaeche, WindowPtr)			Temperaturregelung: Init -> Halten
				AnAbwesenheit: Init -> Abwesenheit
1.	Anwesen-heitSet-zen(CJeman dDa)	-	-	-
2.	callBack()	-> Roomtemperature-Controller.Zieltempera-turSetzen(20)	-> Roomtemperature-Controller.Zieltempera-turSetzen <- Ausgabe: Zieltemperatur 20 Grad	AnAbwesenheit: Abwesenheit -> Anwe-senheit Init->AnwesenheitFen-sterNichtOffen
3.	Temperatur-StatusSet-zen(CZimme rTempZu-Tief)	-	-	-
4.	callBack()	-> Radiator.Thermo-statSetzen()	Radiator.Thermostat-Setzen() <- Ausgabe: Thermostat auf 52 Grad gesetzt	Temperaturregelung: Halten -> Heizen
5.	Zimmertem-peraturSet-zen(25)	-	-	-
6.	Aussentem-peraturSet-zen(25)	-	-	-
7.	Temperatur-StatusSet-zen(CZimme rTempZu-Hoch)	-	-	-

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
8.	callBack()	-> Radiator.ThermostatSetzen(0)	Radiator.ThermostatSetzen() <- Ausgabe: Thermostat auf 0 Grad gesetzt	Temperaturregelung: Heizen -> Kuehlen
9.	FensterzustandSetzen(CFenster Offen)	-	-	-
10.	callBack()	-> RoomtemperatureController.ZieltemperaturSetzen(15)	RoomtemperatureController.ZieltemperaturSetzen() <- Ausgabe: Zieltemperatur auf 15 Grad gesetzt	Anwesenheit: AnwesenheitFensterNichtOffen -> AnwesenheitFensterOffen
11.	FensterZustandSetzen(CFenster Geschlossen)	-	-	-
12.	callBack()	-> RoomtemperatureController.ZieltemperaturSetzen(20)	RoomtemperatureController.ZieltemperaturSetzen() <- Ausgabe: Zieltemperatur auf 20 Grad gesetzt	Anwesenheit: AnwesenheitFensterOffen -> AnwesenheitFensterNichtOffen
13.	ZimmertemperaturSetzen(18)	-	-	-
14.	TemperaturStatusSetzen(CZimmerTempZuTief)	-	-	-
15.	callBack()	-> Radiator.ThermostatSetzen()	Radiator.ThermostatSetzen() <- Ausgabe: Thermostat auf 29 Grad gesetzt	Temperaturregelung: Kuehlen -> Heizen

2.5.1.8 Klasse Window

Folgende Stubs sind für die Test erforderlich:

- Room

- void Fensterzustand (zFensterType Position)

- Contact
  - Contact (SectionNr, RoomNr, PortNr)
  - int AbfrageKontakt()
- Motor
  - Motor (SectionNr, RoomNr, PortNr)
  - void Schliessen()
  - void Oeffnen()

KompTest11:Das Fenster wird geöffnet. Anschließend beginnt es zu regnen und das Fenster muß gekippt werden.

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
	Window(SectionNr, RoomNr)			Init -> geschlossen
1.	callBack()	Offenkontakt.AbfrageKontakt()	Offenkontakt.AbfrageKontakt() <- offen (1)	geschlossen -> geschlossen
		Kippkontakt.AbfrageKontakt()	Kippkontakt.AbfrageKontakt() <- geschlossen(0)	
2.	callBack()	Room.Fensterzustand(CFensterOffen)	Room.Fensterzustand() <- Ausgabe: Fensterzustand offen mitgeteilt	geschlossen -> offen
3.	RegenFenster()	Schwenkmotor.Schliessen()	Schwenkmotor.Schliessen <- Ausgabe: Fenster schliessen	offen -> gekippt
		Kippmotor.Oeffnen()	Kippmotor.Oeffen <- Ausgabe: Fenster kippen	
		Room.Fensterzustand(CFensterGekippt)	Room.Fensterzustand() <- Ausgabe: Fensterzustand gekippt mitgeteilt	

KompTest12:Das Fenster wird zum Kuehlen gekippt. Der Benutzer öffnet das Fenster vollständig und schließt es danach wieder.

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
	Window(SectionNr, RoomNr)			Init -> geschlossen
1.	Zimmer-Kuehlen()	Kippmotor.Oeffnen()	Kippmotor.Oeffen <- Ausgabe: Fenster kippen	geschlossen -> gekippt
		Room.Fensterzustand(CFensterGekippt)	Room.Fensterzustand() <- Ausgabe: Fensterzustand gekippt mitgeteilt	
2.	callBack()	Offenkontakt.AbfrageKontakt()	Offenkontakt.AbfrageKontakt() <- geschlossen (0)	gekippt -> gekippt
		Kippkontakt.AbfrageKontakt()	Kippkontakt.AbfrageKontakt() <- geschlossen(0)	
3.	callBack()	Room.Fensterzustand(CFensterGeschlossen)	Room.Fensterzustand() <- Ausgabe: Fensterzustand geschlossen mitgeteilt	gekippt -> geschlossen

KompTest13:Benutzer öffnet und schließt das Fenster. Danach wird es vom Benutzer gekippt.

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
	Window(SectionNr, RoomNr)			Init -> geschlossen

Schritt	Vom Treiber aufgerufene Methoden der zu testenden Klasse	Erwartetes Verhalten der zu testenden Klasse	Gewünschte Reaktion der zu erstellenden Stubs	Überdeckte Kante im Statemodel
1.	callBack()	Offenkontakt.AbfrageKontakt()	Offenkontakt.AbfrageKontakt() <- offen (1)	geschlossen -> geschlossen
		Kippkontakt.AbfrageKontakt()	Kippkontakt.AbfrageKontakt() <- geschlossen(0)	
2.	callBack()	Room.Fensterzustand(CFensterOffen)	Room.Fensterzustand() <- Ausgabe: Fensterzustand offen mitgeteilt	geschlossen -> offen
3.	callBack()	Offenkontakt.AbfrageKontakt()	Offenkontakt.AbfrageKontakt() <- geschlossen (0)	offen -> offen
		Kippkontakt.AbfrageKontakt()	Kippkontakt.AbfrageKontakt() <- geschlossen(0)	
4.	callBack()	Room.Fensterzustand(CFensterGeschlossen)	Room.Fensterzustand() <- Ausgabe: Fensterzustand geschlossen mitgeteilt	offen -> geschlossen
5.	callBack()	Offenkontakt.AbfrageKontakt()	Offenkontakt.AbfrageKontakt() <- geschlossen (0)	geschlossen -> geschlossen
		Kippkontakt.AbfrageKontakt()	Kippkontakt.AbfrageKontakt() <- offen (1)	
6.	callBack()	Room.Fensterzustand(CFensterGekippt)	Room.Fensterzustand() <- Ausgabe: Fensterzustand gekippt mitgeteilt	geschlossen -> gekippt

2.5.2 Ergebnisse des Komponententests

In diesem Kapitel werden zu jedem Komponentest die Ergebnisse des Testdurchlaufs angegeben.

2.5.2.1 Klasse House

Ergebnis zu KompTest1:

```
Testfall? (1):
TESTFALL:1
```

```
************************************************
1. Aufruf von callBack():
Kesseltemperatur auf 15 Grad gesetzt.
2. Aufruf von callBack():
Regenmeldung an Section
3. Aufruf von callBack():
4. Aufruf von callBack():
```

2.5.2.2 Klasse PresenceController

Ergebnis zu KompTest2:

```
Testfall? (2):
TESTFALL:2
************************************************
1. Aufruf von StatusAktualisieren():
CNiemandDa(1) seit 1 Minuten.
2. Aufruf von callBack():
3. Aufruf von StatusAktualisieren():
CNiemandDa(1) seit 5 Minuten.
4. Aufruf von callBack():
5. Aufruf von StatusAktualisieren():
CJemandDa(0) seit 2 Minuten.
6. Aufruf von callBack():
7. Aufruf von callBack():
```

2.5.2.3 Klasse PRTempController

Ergebnis zu KompTest3:

```
Testfall? (3):
TESTFALL:3
************************************************
1. Aufruf von callBack():
2. Aufruf von callBack():
3. Aufruf von callBack():
Status auf CJemandDa(0) gesetzt.
4. Aufruf von ZeitspanneEintrittNormalSetzen(21):
ZeitspanneEintrittNormal auf 21 Minuten gesetzt.
5. Aufruf von callBack():
6. Aufruf von callBack():
7. Aufruf von callBack():
Status auf CNiemandDa(1) gesetzt.
```

2.5.2.4 Klasse ControlPanel

Ergebnis zu KompTest4:

```
Testfall? (4):
TESTFALL:4
```

```
**********************************************
1. Aufruf von ZeitspanneAufheizenAufDefault():
ZeitspanneAufheizen auf Default gesetzt
2. Aufruf von ZeitspanneAufheizenNormalSetzen(25):
ZeitspanneAufheizen auf 25 Minuten gesetzt.
3. Aufruf von AnwesenheitsTempAufDefault():
AnwesenheitsTemp auf Default gesetzt.
4. Aufruf von AnwesenheitsTempNormalSetzen(24)
AnwesenheitsTemp auf 24 Grad gesetzt.
5. Aufruf von AbwesenheitsTempAufDefault():
AbwesenheitsTemp auf Default gesetzt.
6. Aufruf von AbwesenheitsTempNormalSetzen(24):
AbwesenheitsTemp auf 24 Grad gesetzt.
7. Aufruf von ZeitspanneVerlassenAufDefault():
ZeitspanneVerlassen auf Default gesetzt.
8. Aufruf von ZeitspanneVerlassenNormalSetzen(5):
ZeitspanneVerlassen auf 5 Minuten gesetzt
9. Aufruf von ZeitspanneEintrittAufDefault():
ZeitspanneEintritt auf Default gesetzt.
10. Aufruf von ZeitspanneEintrittNormalSetzen(10):
ZeitspanneEintritt auf 10 Minuten gesetzt
```

2.5.2.5 Klasse Room

Ergebnis zu KompTest5:

```
Testfall? (5):
TESTFALL:5
**********************************************
1. Aufruf von BenoetigteKesselTempAbfragen():
 (Kesseltemp abgefragt20)
2. Aufruf von Fensterzustand(CFensterOffen):
Fensterzustand auf CFensterOffen(1) gesetzt.
3. Aufruf von AussentemperaturWeiterleiten(12):
Aussentemperatur auf 12 Grad gesetzt.
4. Aufruf von RegenAnZimmer():
Regen an Fenster gemeldet.
```

2.5.2.6 Klasse RoomtemperatureController

Ergebnis zu KompTest6:

```
Testfall? (6-7):
TESTFALL:6
**********************************************
1. Aufruf von callBack():
2. Aufruf von callBack():
Zimmertemperatur auf 28 Grad gesetzt.
Zimmertemperaturstatus: CZimmerTempZuHoch(2)
3. Aufruf von callBack():
4. Aufruf von callBack():
Zimmertemperatur auf 10 Grad gesetzt.
```

```
Zimmertemperaturstatus: CZimmerTempZuTief(0)
5. Aufruf von callBack():
6. Aufruf von callBack():
Zimmertemperatur auf 15 Grad gesetzt.
Zimmertemperaturstatus: CZimmerTempGut(1)
```

Ergebnis zu KompTest7:

```
Testfall? (6-7):
TESTFALL:7
*************************************************
1. Aufruf von callBack():
2. Aufruf von callBack():
Zimmertemperatur auf 10 Grad gesetzt.
Zimmertemperaturstatus: CZimmerTempZuTief(0)
3. Aufruf von callBack():
4. Aufruf von callBack():
Zimmertemperatur auf 20 Grad gesetzt.
Zimmertemperaturstatus: CZimmerTempZuHoch(2)
5. Aufruf von callBack():
6. Aufruf von callBack():
Zimmertemperatur auf 15 Grad gesetzt.
Zimmertemperaturstatus: CZimmerTempGut(1)
```

2.5.2.7 Klasse Temperatureregulator

Ergebnis zu KompTest8:

```
Testfall? (8-10):
TESTFALL:8
*************************************************
1. Aufruf von ZimmertemperaturSetzen(10):
2. Aufruf von TemperaturStatusSetzen(CZimmerTempZuTief):
3. Aufruf von callBack():
Thermostat auf 45 Grad gesetzt.
4. Aufruf von ZimmertemperaturSetzen(15):
5. Aufruf von TemperaturStatusSetzen(CZimmerTempGut):
6. Aufruf von callBack():
Thermostat auf 15 Grad gesetzt.
7. Aufruf von AnwesenheitSetzen(CJemandDa):
8. Aufruf von callBack():
Zieltemperatur 20 Grad
9. Aufruf von FensterzustandSetzen(CFensterOffen):
10. Aufruf von callBack():
Zieltemperatur 15 Grad
11. Aufruf von FensterzustandSetzen(CFensterGekippt):
12. Aufruf von callBack():
Zieltemperatur 20 Grad
```

Ergebnis zu KompTest9:

```
Testfall? (8-10):
```

```
TESTFALL:9
*************************************************
1. Aufruf von AnwesenheitSetzen(CJemandDa):
2. Aufruf von AussentemperaturSetzen(28):
3. Aufruf von ZimmertemperaturSetzen(25):
4. Aufruf von TemperaturStatusSetzen(CZimmerTempZuHoch):
5. Aufruf von callBack():
Thermostat auf 0 Grad gesetzt.
Zieltemperatur 20 Grad
6. Aufruf von AussentemperaturSetzen(18):
7. Aufruf von callBack():
Zimmer wird gekuehlt.
8. Aufruf von FensterzustandSetzen(CFensterGekippt):
9. Aufruf von ZimmertemperaturSetzen(20):
10. Aufruf von TemperaturStatusSetzen(CZimmerTempGut):
11. Aufruf von callBack():
Thermostat auf 22 Grad gesetzt.
12. Aufruf von AnwesenheitSetzen(CNiemandDa):
13. Aufruf von callBack():
Zieltemperatur 15 Grad
```

Ergebnis zu KompTest10:

```
Testfall? (8-10):
TESTFALL:10
*************************************************
1. Aufruf von AnwesenheitSetzen(CJemandDa):
2. Aufruf von callBack():
Zieltemperatur 20 Grad
3. Aufruf von TemperaturStatusSetzen(CZimmerTempZuTief):
4. Aufruf von callBack():
Thermostat auf 52 Grad gesetzt.
5. Aufruf von ZimmertemperaturSetzen(25):
6. Aufruf von AussentemperaturSetzen(25):
7. Aufruf von TemperaturStatusSetzen(CZimmerTempZuHoch):
8. Aufruf von callBack():
Thermostat auf 0 Grad gesetzt.
9. Aufruf von FensterzustandSetzen(CFensterOffen):
10. Aufruf von callBack():
Zieltemperatur 15 Grad
11. Aufruf von FensterzustandSetzen(CFensterGeschlossen):
12. Aufruf von callBack():
Zieltemperatur 20 Grad
13. Aufruf von ZimmertemperaturSetzen(18):
14. Aufruf von TemperaturStatusSetzen(CZimmerTempZuTief):
15. Aufruf von callBack():
Thermostat auf 29 Grad gesetzt.
```

2.5.2.8 Klasse Window

Ergebnis zu KompTest11:

```
Testfall? (11-13):
TESTFALL:11
*************************************************
1. Aufruf von callBack():
2. Aufruf von callBack():
Fensterzustand auf CFensterOffen(1) gesetzt
3. Aufruf von RegenFenster():
Fenster wird durch Schwenkmotor geschlossen.
Fenster wird durch Kippmotor gekippt.
Fensterzustand auf CFensterGekippt(0) gesetzt
```

Ergebnis zu KompTest11:

```
Testfall? (11-13):
TESTFALL:12
*************************************************
1. Aufruf von ZimmerKuehlen():
Fenster wird durch Kippmotor gekippt.
Fensterzustand auf CFensterGekippt(0) gesetzt
2. Aufruf von callBack():
3. Aufruf von callBack():
Fensterzustand auf CFensterGeschlossen(2) gesetzt
```

Ergebnis zu KompTest13:

```
Testfall? (11-13):
TESTFALL:13
*************************************************
1. Aufruf von callBack():
2. Aufruf von callBack():
Fensterzustand auf CFensterOffen(1) gesetzt
3. Aufruf von callBack():
4. Aufruf von callBack():
Fensterzustand auf CFensterGeschlossen(2) gesetzt
5. Aufruf von callBack():
6. Aufruf von callBack():
Fensterzustand auf CFensterGekippt(0) gesetzt
```