

Teil 3

Software-Komponenten- dokumentation

Inhaltsverzeichnis

3.1	Einleitung	3
3.2	K(omponenten)-Anforderungen/K-Spezifikation	3
3.3	K-Realisierung	4
3.3.1	Klasse CentralHeatingSite	4
3.3.2	Klasse Contact	5
3.3.3	Klasse ControlPanel	6
3.3.4	Klasse House	9
3.3.5	Klasse Motor	12
3.3.6	Klasse PRTempController	14
3.3.7	Klasse Portbaustein	17
3.3.8	Klasse PresenceController	17
3.3.9	Klasse Radiator	20
3.3.10	Klasse Room	21
3.3.11	Klasse RoomtemperatureController	23
3.3.12	Klasse Section	26
3.3.13	Klasse Temperaturregulator	28
3.3.14	Klasse Temperaturesensor	38
3.3.15	Klasse Window	39
3.4	Verifikation	43
3.4.1	Verifikations-Checklisten	43
3.4.2	Klassifikation von Fehlern	44
3.4.3	Ergebnisse der Verifikation	44
3.5	Validation	45

Kapitel 3

Software-Komponentendokumentation

3.1 Einleitung

In diesem Dokument sind die Software-Komponenten des Systems dokumentiert. Zuerst folgen die Komponenten-Anforderungen und die Komponenten-Spezifikation. Im daran anschließenden Kapitel K-Realisierung wird der Code der erstellten Komponenten aufgelistet. Die Kapitel Verifikation und Validation bilden den Abschluß dieses Dokumentes.

3.2 K(omponenten)-Anforderungen/K-Spezifikation

Die K-Anforderungen wurden bereits in Kapitel 2.3 der Software-Systemdokumentation vollständig definiert.

3.3 K-Realisierung

Im folgenden Kapitel wird der Code der einzelnen Klassen aufgeführt. Die Umsetzung erfolgte nach den Richtlinien zur Umsetzung von Statemodells in C++-Kode. Desweiteren wurden die Programmierrichtlinien aus dem Projekt CoDEX bis auf folgende Ausnahmen beachtet:

- Die Bezeichnung der Elemente aus dem Entwurf werden ausnahmslos beibehalten, d.h. alle Klassen, Attribute und Methoden behalten ihre Bezeichnung aus dem Entwurf.
- Die Bezeichnung von Typen, die einen Zustand beschreiben, beginnt mit einem kleinen z gefolgt von dem eigentlichen Namen des Typen.
- Die Bezeichnung von Variablen, die einen Zustand beschreiben, beginnt mit einem großen Z, gefolgt von dem eigentlichen Namen der Variable.

3.3.1 Klasse CentralHeatingSite

Header-Datei:

```
#ifndef _CENTRALHEATINGSITE_HH
#define _CENTRALHEATINGSITE_HH
//
// Header file for class CentralHeatingSite
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse CentralHeatingSite stellt den Zugriff auf die Heizungsanlage
// bereit.
// Da es sich um eine spezielle Hardware handelt, ist diese Klasse vom
// Portbaustein abgeleitet.
//
//-----

#include <SimpleTypes.hh>
#include <Portbaustein.hh>

class CentralHeatingSite:public Portbaustein
{
private:

    int SectionNr;
    int RoomNr;
    int PortNr;

public:

    void KesseltemperaturSetzen(tempType Temperatur);
    CentralHeatingSite();

};

#endif
```

Quell-Kode:

```
//
// Implementation file for class CentralHeatingSite
//
// Author: Wolfgang Wagenbichler
//-----
```

```

//
// Die Klasse CentralHeatingSite stellt den Zugriff auf die Heizungsanlage
// bereit.
// Da es sich um eine spezielle Hardware handelt, ist diese Klasse vom
// Portbaustein abgeleitet.
//
//-----
#include <CentralHeatingSite.hh>

//=====
// ClassName: CentralHeatingSite
//=====

CentralHeatingSite::CentralHeatingSite()
{
    SectionNr = 0;
    RoomNr = 0;
    PortNr = 8;
} // CentralHeatingSite

void CentralHeatingSite::KesseltemperaturSetzen(tempType Temperatur)
{
    CallIntOutputPort (SectionNr, RoomNr, PortNr, Temperatur);
} // KesseltemperaturSetzen

```

3.3.2 Klasse Contact

Header-Datei:

```

#ifndef _CONTACT_HH
#define _CONTACT_HH
//
// Header file for class Contact
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Contact stellt den Zugriff auf spezielle Kontakt bereit.
// Da es sich um eine spezielle Hardware handelt, ist diese Klasse vom
// Portbaustein abgeleitet.
//
//-----
#include <Portbaustein.hh>

class Contact:public Portbaustein
{
private:
    int SectionNr;
    int RoomNr;
    int PortNr;

public:
    int AbfrageKontakt();
    Contact (int InitialSectionNr, int InitialRoomNr, int InitialPortNr);
};

```

```
#endif
```

Quell-Kode

```
//
// Implementation file for class Contact
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Contact stellt den Zugriff auf spezielle Kontakt bereit.
// Da es sich um eine spezielle Hardware handelt, ist diese Klasse vom
// Portbaustein abgeleitet.
//
//-----

#include <Contact.hh>

//=====
// ClassName: Contact
//=====

Contact::Contact(int InitialSectionNr, int InitialRoomNr, int InitialPortNr)
{
    SectionNr = InitialSectionNr;
    RoomNr = InitialRoomNr;
    PortNr = InitialPortNr;
} // Contact

int Contact::AbfrageKontakt()
{
    return CallBitInputPort (SectionNr, RoomNr, PortNr);
} // AbfrageKontakt()
```

3.3.3 Klasse ControlPanel

Header-Datei:

```
#ifndef _CONTROLPANEL_HH
#define _CONTROLPANEL_HH
//
// Header file for class ControlPanel
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Control besitzt kein aktives Verhalten. Sie leitet Werte, die
// vom Benutzer gesetzt werden koennen an die entsprechend zustaendige
// Klasse weiter.
//
//-----

#include <Temperatureregulator.hh>
#include <PRTempController.hh>

class ControlPanel
{
private:
    Temperatureregulator *TemperatureregulatorPtr;
```

```

PRTempController      *PRTempControllerPtr;

public:

    fehlerType ZeitspanneAufheizenAufDefault();
    fehlerType ZeitspanneAufheizenNormalSetzen (durationType Zeitspanne);
    void AnwesenheitsTempAufDefault();
    void AnwesenheitsTempNormalSetzen(tempType Temperatur);
    void AbwesenheitsTempAufDefault();
    void AbwesenheitsTempNormalSetzen(tempType Temperatur);
    void ZeitspanneEintrittAufDefault();
    void ZeitspanneEintrittNormalSetzen(durationType Zeitspanne);
    void ZeitspanneVerlassenAufDefault();
    void ZeitspanneVerlassenNormalSetzen(durationType Zeitspanne);

    ControlPanel (Temperatureregulator *InitialTemperatureregulatorPtr,
PRTempController *InitialPRTempControllerPtr);

};

#endif

```

Quell-Kode:

```

//
// Implementation file for class ControlPanel
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Control besitzt kein aktives Verhalten. Sie leitet Werte, die
// vom Benutzer gesetzt werden koennen an die entsprechend zustaeandige
// Klasse weiter.
//
//-----

#include <ControlPanel.hh>

//=====
// ClassName: ControlPanel
//=====

ControlPanel::ControlPanel (Temperatureregulator *InitialTemperatureregulatorPtr,
    PRTempController *InitialPRTempControllerPtr)
{
    // Attribute initialisieren

    TemperatureregulatorPtr = InitialTemperatureregulatorPtr;
    PRTempControllerPtr = InitialPRTempControllerPtr;
} // ControlPanel

//-----

fehlerType ControlPanel::ZeitspanneAufheizenAufDefault()
{
    return TemperatureregulatorPtr->ZeitspanneAufheizenAufDefault();
} // ZeitspanneAufheizenAufDefault

//-----

fehlerType ControlPanel::ZeitspanneAufheizenNormalSetzen (durationType Zeitspanne)
{

```

```
    return TemperatureregulatorPtr->ZeitspanneAufheizenNormalSetzen(Zeitspanne);
} // ZeitspanneAufheizenNormalSetzen
//-----

void ControlPanel::AnwesenheitsTempAufDefault()
{
    TemperatureregulatorPtr->AnwesenheitsTempAufDefault();
} // AnwesenheitsTempAufDefault
//-----

void ControlPanel::AnwesenheitsTempNormalSetzen(tempType Temperatur)
{
    TemperatureregulatorPtr->AnwesenheitsTempNormalSetzen(Temperatur);
} // AnwesenheitsTempNormalSetzen
//-----

void ControlPanel::AbwesenheitsTempAufDefault()
{
    TemperatureregulatorPtr->AbwesenheitsTempAufDefault();
} // AbwesenheitsTempAufDefault
//-----

void ControlPanel::AbwesenheitsTempNormalSetzen(tempType Temperatur)
{
    TemperatureregulatorPtr->AbwesenheitsTempNormalSetzen(Temperatur);
} // AbwesenheitsTempNormalSetzen
//-----

void ControlPanel::ZeitspanneEintrittAufDefault()
{
    PRTempControllerPtr->ZeitspanneEintrittAufDefault();
} // ZeitspanneEintrittAufDefault
//-----

void ControlPanel::ZeitspanneEintrittNormalSetzen(durationType Zeitspanne)
{
    PRTempControllerPtr->ZeitspanneEintrittNormalSetzen(Zeitspanne);
} // ZeitspanneEintrittNormalSetzen
//-----

void ControlPanel::ZeitspanneVerlassenAufDefault()
{
```

```

    PRTempControllerPtr->ZeitspanneVerlassenAufDefault();
} // ZeitspanneVerlassenAufDefault

//-----

void ControlPanel::ZeitspanneVerlassenNormalSetzen(durationType Zeitspanne)
{
    PRTempControllerPtr->ZeitspanneVerlassenNormalSetzen(Zeitspanne);
} // ZeitspanneVerlassenNormalSetzen

```

3.3.4 Klasse House

Header-Datei:

```

#ifndef _HOUSE_HH
#define _HOUSE_HH
//
// Header file for class House
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse House fragt regelmaessig den Regenkontakt ab und meldet
// neu einsetzenden Regen an alle Abschnitte. Weiterhin wird in
// regelmaessigen Abstaenden die benoetigte Kesseltemperatur aller Abschnitte
// abgefragt und an die Heizungsanlage weitergegeben.
// Da das Haus ueber den zentralen Aussentemperaturfuehler verfuegt, wird
// eine Aenderung der Aussentemperatur an alle Abschnitte weitergeleitet.
//
//-----

#include <SimpleTypes.hh>
#include <Slist.hh>
#include <ScheduledFct.hh>
#include <CentralHeatingSite.hh>
#include <Temperaturesensor.hh>
#include <Contact.hh>
#include <Section.hh>
#include <Timer.hh>

class House:public ScheduledFct
{
private:

    enum {CkeinRegen, CRegen} stateHouse;

    tempType MaxBenoetigteKesseltemperatur;
    tempType Aussentemperatur;

    CentralHeatingSite *CentralHeatingSitePtr;
    Temperaturesensor *Aussentemperaturfuehler;
    Contact *Regenkontakt;

    int ZRegenkontakt;

    Timer *Zeit;

```

```

SList <Section *> SectionPtrListe;

private:

    void RegenAnSection();
    void AussentemperaturAnSection(tempType Temperatur);

public:

    void callBack();
    House (SList<double> InitialSectionZimmerListe, Timer* ZeitPtr);

};

#endif

```

Quell-Kode:

```

//
// Implementation file for class House
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse House fragt regelmaessig den Regenkontakt ab und meldet
// neu einsetzenden Regen an alle Abschnitte. Weiterhin wird in
// regelmaessigen Abstaenden die benoetigte Kesseltemperatur aller Abschnitte
// abgefragt und an die Heizungsanlage weitergegeben.
// Da das Haus ueber den zentralen Aussentemperaturfuehler verfuegt, wird
// eine Aenderung der Aussentemperatur an alle Abschnitte weitergeleitet.
//
//-----

#include <House.hh>

//=====
// ClassName: House
//=====

House::House(SList<double> InitialSectionZimmerListe, Timer* ZeitPtr)
{

    double AktFlaeche;
    double NrFlaeche;
    Section *AktSectionPtr;
    int AktSectionNr = 1;
    int i;
    SList<double> ZimmerFlaechenListe;

    // Attribute initialisieren

    Zeit = ZeitPtr;
    MaxBenoetigteKesseltemperatur = 10;
    Aussentemperatur = 10;
    ZRegenkontakt = offen;

    // Startzustand setzen

    stateHouse = CkeinRegen;

    // Instanzen von aggregierten Klassen erzeugen

    CentralHeatingSitePtr = new CentralHeatingSite();
    Aussentemperaturfuehler = new Temperaturesensor(0,0,9);
    Regenkontakt = new Contact(0,0,10);

    // Sections erzeugen

```

```

SectionPtrListe.clear();
InitialSectionZimmerListe.reset();

while (InitialSectionZimmerListe.next(NrFlaeche) != notempty) {
    // Flaechenliste fuer Section generieren
    ZimmerFlaechenListe.clear();
    for (i=0 ; i < (int)NrFlaeche; i++) {
        InitialSectionZimmerListe.next(AktFlaeche);
        ZimmerFlaechenListe.append(AktFlaeche);
    }

    AktSectionPtr = new Section (AktSectionNr, ZimmerFlaechenListe, Zeit);
    SectionPtrListe.append (AktSectionPtr);
    AktSectionNr++;
} // while

// Callback regelmaessig ausfuehren.
registerCallBack();
} // House

//-----

void House::RegenAnSection()
{
    Section *AktSectionPtr;

    SectionPtrListe.reset();
    // benachrichtige alle Sections, dass es angefangen hat zu regnen

    while (SectionPtrListe.next(AktSectionPtr) != notempty)
        AktSectionPtr->RegenAnZimmer();
} // RegenAnSection

//-----

void House::AussentemperaturAnSection(tempType Temperatur)
{
    Section *AktSectionPtr;

    SectionPtrListe.reset();
    // benachrichtige alle Section, dass sich die Aussentemperatur
    // geaendert hat.

    while (SectionPtrListe.next(AktSectionPtr) != notempty)
        AktSectionPtr->AussentemperaturWeiterleiten(Temperatur);
} // AussentemperaturAnSection

//-----

void House::callBack()
{
    // regelmaessig benoetigte Kesseltemp abfragen

    tempType benoetigteKesseltemperatur;
    tempType maxKesseltemperatur;
    Section *AktSectionPtr;

    maxKesseltemperatur=0;
    benoetigteKesseltemperatur=0;
    SectionPtrListe.reset();
    while (SectionPtrListe.next(AktSectionPtr) != notempty) {

```

```

    benoetigteKesseltemperatur=AktSectionPtr->BenoeitigteKesselTempAbfragen();
    if (benoetigteKesseltemperatur > maxKesseltemperatur)
        maxKesseltemperatur = benoetigteKesseltemperatur;
} // while

if (maxKesseltemperatur != MaxBenoeitigteKesseltemperatur) {
    MaxBenoeitigteKesseltemperatur = maxKesseltemperatur;
    CentralHeatingSitePtr->KesseltemperaturSetzen(maxKesseltemperatur);
}

// zu den regelmaessigen Aufgaben gehoert auch die Ueberpruefung der
// Aussentemperatur

tempType ZAussentemperaturfuehler;

ZAussentemperaturfuehler = Aussentemperaturfuehler->AbfrageTemperatur();
// eine geaenderte Aussentemperatur an alle Sections weiterleiten
if (ZAussentemperaturfuehler != Aussentemperatur) {
    Aussentemperatur = ZAussentemperaturfuehler;
    AussentemperaturAnSection(Aussentemperatur);
}

// Regeneueberpruefung

switch (stateHouse) {

case CkeinRegen:
    if (ZRegenkontakt == geschlossen) {
        // Exit-Aktionen

        // neue Konfig berechnen
        stateHouse = CRegen;

        // Entry-Aktionen
        RegenAnSection();
    } // if
    else {
        // Do-Aktivitaet
        ZRegenkontakt = Regenkontakt->AbfrageKontakt();
    } // else
    break;

case CRegen:
    if (ZRegenkontakt == offen) {
        // Exit-Aktionen

        // neue Konfig. berechnen
        stateHouse = CkeinRegen;

        // Entry-Aktionen
    } // if
    else {
        // Do-Aktivitaet
        ZRegenkontakt = Regenkontakt->AbfrageKontakt();
    } // else
    break;

} // switch

} // callBack

```

3.3.5 Klasse Motor

Header-Datei:

```
#ifndef _MOTOR_HH
```

```

#define _MOTOR_HH
//
// Header file for class Motor
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Motor stellt den Zugriff auf alle Motoren bereit.
// Da es sich um eine spezielle Hardware handelt, ist diese Klasse vom
// Portbaustein abgeleitet.
//
//-----

#include <SimpleTypes.hh>
#include <Portbaustein.hh>

class Motor:public Portbaustein
{
private:

    int SectionNr;
    int RoomNr;
    int PortNr;

public:

    void Oeffnen();
    void Schliessen();
    Motor(int InitialSectionNr, int InitialRoomNr, int InitialPortNr);

};

#endif

```

Quell-Kode:

```

//
// Implementation file for class Motor
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Motor stellt den Zugriff auf alle Motoren bereit.
// Da es sich um eine spezielle Hardware handelt, ist diese Klasse vom
// Portbaustein abgeleitet.
//
//-----

#include <Motor.hh>

//=====
// ClassName: Motor
//=====

Motor::Motor(int InitialSectionNr, int InitialRoomNr, int InitialPortNr)
{

    SectionNr = InitialSectionNr;
    RoomNr = InitialRoomNr;
    PortNr = InitialPortNr;

} // Motor

void Motor::Oeffnen()
{

    CallBitOutputPort (SectionNr, RoomNr, PortNr, oeffnen);

```

```

} // Oeffnen

void Motor::Schliessen()
{
    CallBitOutputPort (SectionNr, RoomNr, PortNr, schliessen);
} // Schliessen

```

3.3.6 Klasse PRTempController

Header-Datei:

```

#ifndef _PRTEMPCONTROLLER_HH
#define _PRTEMPCONTROLLER_HH
//
// Header file for class PRTempController
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse PRTempController berechnet die Anwesenheitswerte fuer die
// Temperaturregelung. Da erst nach einer ZeitspanneEintrittNormal bzw.
// ZeitspanneVerlassenNormal die Temperaturregelung eingreift, werden
// diese Zeitspannen bei der Anwesenheitskontrolle beruecksichtigt.
//
//-----

// forward declarations
// class Temperatureregulator;

#include <SimpleTypes.hh>
#include <ScheduledFct.hh>
#include <PresenceController.hh>
#include <Temperatureregulator.hh>

class PRTempController:public ScheduledFct
{
private:

    enum {CKeinerDa, CEinerDa} statePRTempController;

    durationType ZeitspanneEintrittNormal;
    durationType CZeitspanneEintrittDefault;
    durationType ZeitspanneVerlassenNormal;
    durationType CZeitspanneVerlassenDefault;

    PresenceController *PresenceControllerPtr;
    Temperatureregulator *TemperatureregulatorPtr;

    zAnwesenheitZeitType AnwStatus;

public:

    void callBack();
    void ZeitspanneVerlassenAufDefault();
    void ZeitspanneVerlassenNormalSetzen(durationType Zeitspanne);
    void ZeitspanneEintrittAufDefault();
    void ZeitspanneEintrittNormalSetzen(durationType Zeitspanne);
    PRTempController (PresenceController *PControllerPtr,
        Temperatureregulator *TemperatureregPtr);
};

```

```
#endif
```

Quell-Kode:

```
//
// Implementation file for class PRTempController
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse PRTempController berechnet die Anwesenheitswerte fuer die
// Temperaturregelung. Da erst nach einer ZeitspanneEintrittNormal bzw.
// ZeitspanneVerlassenNormal die Temperaturregelung eingreift, werden
// diese Zeitspannen bei der Anwesenheitskontrolle beruecksichtigt.
//
//-----

#include <PRTempController.hh>
#include <iostream.h>

//=====
// ClassName: PRTempController
//=====

PRTempController::PRTempController (PresenceController
    *InitialPresenceControllerPtr,
    Temperatureregulator
    *InitialTemperatureregulatorPtr)
{
    // Attribute initialisieren
    ZeitspanneEintrittNormal = 10;
    CZeitspanneEintrittDefault = 10;
    ZeitspanneVerlassenNormal = 5;
    CZeitspanneVerlassenDefault = 5;

    PresenceControllerPtr = InitialPresenceControllerPtr;
    TemperatureregulatorPtr = InitialTemperatureregulatorPtr;

    AnwStatus.ZAnwesenheitskontakt = CNiemandDa;
    // Zeispanne soll bei Systemtstart abgelaufen sein
    AnwStatus.ZeitDiff = ZeitspanneVerlassenNormal+1;

    // Startzustand setzen
    statePRTempController=CKeinerDa;

    // CallBack regelmaessig ausfuehren.
    registerCallBack();
} // PRTempController

//-----

void PRTempController::ZeitspanneVerlassenAufDefault()
{
    ZeitspanneVerlassenNormal = CZeitspanneVerlassenDefault;
} // ZeitspanneVerlassenAufDefault

//-----

void PRTempController::ZeitspanneVerlassenNormalSetzen(durationType Zeitspanne)
{
    ZeitspanneVerlassenNormal = Zeitspanne;
} // ZeitspanneVerlassenNormalSetzen
```

```
//-----  
  
void PRTempController::ZeitspanneEintrittAufDefault()  
{  
  
    ZeitspanneEintrittNormal = CZeitspanneVerlassenDefault;  
    TemperatureregulatorPtr->ZeitspanneEintrittNormalSetzen (ZeitspanneEintrittNormal);  
  
} // ZeitspanneEintrittAufDefault  
  
//-----  
  
void PRTempController::ZeitspanneEintrittNormalSetzen(durationType Zeitspanne)  
{  
  
    ZeitspanneEintrittNormal = Zeitspanne;  
    TemperatureregulatorPtr->ZeitspanneEintrittNormalSetzen(Zeitspanne);  
  
} // ZeitspanneEintrittNormalSetzen  
  
//-----  
  
void PRTempController::callBack()  
{  
  
    switch (statePRTempController) {  
  
        case CKeinerDa:  
            if (AnwStatus.ZAnwesenheitskontakt == CJemandDa &&  
AnwStatus.ZeitDiff >= ZeitspanneEintrittNormal) {  
                // Exit-Aktionen  
  
                // neue Konfig berechnen  
                statePRTempController = CEinerDa;  
  
                // Entry-Aktionen  
                TemperatureregulatorPtr->AnwesenheitSetzen (CJemandDa);  
                TemperatureregulatorPtr->callBack();  
            } // if  
            else {  
                // Do-Aktivitaet  
                AnwStatus=PresenceControllerPtr->StatusAktualisieren();  
            } // else  
            break;  
  
        case CEinerDa:  
            if (AnwStatus.ZAnwesenheitskontakt == CNiemandDa &&  
AnwStatus.ZeitDiff >= ZeitspanneVerlassenNormal) {  
                // Exit Aktionen  
  
                // neue Konfig berechnen  
                statePRTempController = CKeinerDa;  
  
                // Entry-Aktionen  
                TemperatureregulatorPtr->AnwesenheitSetzen (CNiemandDa);  
                TemperatureregulatorPtr->callBack();  
            } // if  
            else {  
                // Do-Aktivitaet  
                AnwStatus=PresenceControllerPtr->StatusAktualisieren();  
            } //else  
            break;  
  
    } // switch  
} // end of callBack()
```

3.3.7 Klasse Portbaustein

Header-Datei:

```
#ifndef _PORTBAUSTEIN_HH
#define _PORTBAUSTEIN_HH
//
// Header file for class Portbaustein
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Portbaustein stellt den Zugriff auf die Harware bereit.
//
//
//-----

class Portbaustein
{
public:

    Portbaustein();
    int CallBitInputPort (int SectionNr, int RoomNr, int PortNr);
    void CallBitOutputPort (int SectionNr, int RoomNr, int PortNr, int value);
    int CallIntInputPort (int SectionNr, int RoomNr, int PortNr);
    void CallIntOutputPort (int SectionNr, int RoomNr, int PortNr, int value);

};

#endif
```

Quell-Kode:

kein Quell-Kode vorhanden.

3.3.8 Klasse PresenceController

Header-Datei:

```
#ifndef _PRESENCECONTROLLER_HH
#define _PRESENCECONTROLLER_HH
//
// Header file for class PresenceController
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse PresenceController fragt den Wert des Anwesenheitskontaktes ab
// und liefert bei Aufruf der Funktion StatusAktualisieren() unter
// Beruecksichtigung des Eintritt- und Verlassen-Zeitpunktes
// den Anwesenheitszustand sowie die Zeitspanne seit der letzten Zustands-
// aenderung zurueck
//
//
//-----

#include <SimpleTypes.hh>
#include <ScheduledFct.hh>
#include <Timer.hh>
#include <Contact.hh>
```

```

class PresenceController: public ScheduledFct
{
private:

    enum {CNiemandAnwesend, CJemandAnwesend} statePresenceController;

    int SectionNr;
    int RoomNr;
    int ZAnwesenheitskontakt;
    Time EintrittZeit;
    Time VerlassenZeit;
    Timer *Zeit;
    Contact *Anwesenheitskontakt;

public:

    void callBack();
    PresenceController(int InitialSectionNr, int InitialRoomNr, Timer *SystemZeit);
    ZAnwesenheitZeitType StatusAktualisieren();
};

#endif

```

Quell-Kode:

```

//
// Implementation file for class PresenceController
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse PresenceController fragt den Wert des Anwesenheitskontaktes ab
// und liefert bei Aufruf der Funktion StatusAktualisieren() unter
// Beruecksichtigung des Eintritt- und Verlassen-Zeitpunktes
// den Anwesenheitszustand sowie die Zeitspanne seit der letzten Zustands-
// aenderung zurueck
//
//
//-----

#include <PresenceController.hh>

//=====
// ClassName: PresenceController
//=====

PresenceController::PresenceController (int InitialSectionNr,
int InitialRoomNr,
Timer *SystemZeit)
{
    // Attribute initialisieren
    SectionNr = InitialSectionNr;
    RoomNr = InitialRoomNr;
    Zeit = SystemZeit;
    EintrittZeit = Zeit->getActualTime();
    VerlassenZeit = Zeit->getActualTime();

    ZAnwesenheitskontakt = offen;
    // Startzustand setzen
    statePresenceController = CNiemandAnwesend;

    // Instanzen von aggregierten Klassen erzeugen
    Anwesenheitskontakt = new Contact(SectionNr, RoomNr, 1);

    // Callback regelmaessig ausfuehren.
    registerCallBack();
}

```

```

} // PresenceController

//-----

zAnwesenheitZeitType PresenceController::StatusAktualisieren()
{
    zAnwesenheitZeitType AnwStatus;
    Time AktuelleZeit;

    AktuelleZeit = Zeit->getActualTime();

    switch (statePresenceController){

    case CNiemandAnwesend:
        AnwStatus.ZAnwesenheitskontakt = CNiemandDa;
        AnwStatus.ZeitDiff = AktuelleZeit.toDuration()-VerlassenZeit.toDuration();
        return AnwStatus;

    case CJemandAnwesend:
        AnwStatus.ZAnwesenheitskontakt = CJemandDa;
        AnwStatus.ZeitDiff = AktuelleZeit.toDuration()-EintrittZeit.toDuration();
        return AnwStatus;

    } // end of switch

} // end of StatusAktualisieren()

//-----

void PresenceController::callBack()
{

    switch (statePresenceController) {

    case CNiemandAnwesend:
        if (ZAnwesenheitskontakt==geschlossen) {
            // Exit-Aktionen

            // neue Konfig. berechnen
            statePresenceController = CJemandAnwesend;

            // Entry-Aktionen
            EintrittZeit = Zeit->getActualTime();
        } // if
        else {
            // Do-Aktivitaet
            ZAnwesenheitskontakt = Anwesenheitskontakt->AbfrageKontakt();
        } // else
        break;

    case CJemandAnwesend:
        if (ZAnwesenheitskontakt==offen) {
            // Exit-Aktionen

            // neue Konfig. berechnen
            statePresenceController = CNiemandAnwesend;

            // Entry-Aktionen
            VerlassenZeit = Zeit->getActualTime();
        } // if
        else {
            // Do-Aktivitaet
            ZAnwesenheitskontakt = Anwesenheitskontakt->AbfrageKontakt();
        } // else
        break;

    } // switch
} // end of callBack()

```

3.3.9 Klasse Radiator

Header-Datei:

```

#ifndef _RADIATOR_HH
#define _RADIATOR_HH
//
// Header file for class Radiator
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Radiator stellt den Zugriff auf die Heizkoerper bereit.
// Da es sich um eine spezielle Hardware handelt, ist diese Klasse vom
// Portbaustein abgeleitet.
//
//-----

#include <SimpleTypes.hh>
#include <Portbaustein.hh>

class Radiator:public Portbaustein
{
private:

    int SectionNr;
    int RoomNr;
    int PortNr;

public:

    void ThermostatSetzen (tempType Temperatur);
    Radiator (int InitialSectionNr, int InitialRoomNr);

};

#endif

```

Quell-Kode:

```

//
// Implementation file for class Radiator
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Radiator stellt den Zugriff auf die Heizkoerper bereit.
// Da es sich um eine spezielle Hardware handelt, ist diese Klasse vom
// Portbaustein abgeleitet.
//
//-----

#include <Radiator.hh>

//=====
// ClassName:
//=====

Radiator::Radiator(int InitialSectionNr, int InitialRoomNr)
{

```

```

    SectionNr = InitialSectionNr;
    RoomNr = InitialRoomNr;
    PortNr = 3;

} // Radiator

void Radiator::ThermostatSetzen(tempType Temperatur)
{
    CallIntOutputPort (SectionNr, RoomNr, PortNr, Temperatur);
} // ThermostatSetzen

```

3.3.10 Klasse Room

Header-Datei:

```

#ifndef _ROOM_HH
#define _ROOM_HH
//
// Header file for class Room
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Room besitzt kein aktives Verhalten. Sie erzeugt bei ihrer
// Instanziierung alle aggregierten Klassen und bildet dann nur noch
// „Durchreichfunktionen“ an.
//
//-----

// forward declarations
class ControlPanel;

#include <SimpleTypes.hh>
#include <PresenceController.hh>
#include <Temperatureregulator.hh>
#include <PRTempController.hh>
#include <Window.hh>
#include <ControlPanel.hh>

class Room
{
private:

    int SectionNr;
    int RoomNr;
    double Flaeche;

    Timer                *Zeit;
    Temperatureregulator *TemperatureregulatorPtr;
    PRTempController     *PRTempControllerPtr;
    PresenceController   *PresenceControllerPtr;
    Window               *WindowPtr;
    ControlPanel         *ControlPanelPtr;

public:

    tempType BenoetigteKesselTempAbfragen();
    void Fensterzustand (zFensterType Position);
    void AussentemperaturWeiterleiten (tempType Temperatur);
    void RegenZimmer();

    Room (int InitialSectionNr,
int InitialRoomNr,
double InitialFlaeche,

```

```

Timer* SystemZeit);

};

#endif

```

Quell-Kode:

```

//
// Implementation file for class Room
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Room besitzt kein aktives Verhalten. Sie erzeugt bei ihrer
// Instanziierung alle aggregierten Klassen und bildet dann nur noch
// „Durchreichfunktionen“ an.
//
//-----

#include <Room.hh>

//=====
// ClassName: Room
//=====

Room::Room (int InitialSectionNr, int InitialRoomNr,
            double InitialFlaeche, Timer *SystemZeit)
{
    // Attribute initialisieren
    SectionNr = InitialSectionNr;
    RoomNr = InitialRoomNr;
    Flaeche = InitialFlaeche;
    Zeit = SystemZeit;

    // Instanzen von aggregierten Klassen erzeugen

    PresenceControllerPtr = new PresenceController (SectionNr, RoomNr, Zeit);
    WindowPtr = new Window (SectionNr, RoomNr, this);
    TemperatureregulatorPtr = new Temperatureregulator (SectionNr,
        RoomNr,
        Flaeche,
        WindowPtr);

    PRTempControllerPtr = new PRTempController(PresenceControllerPtr,
        TemperatureregulatorPtr);

    ControlPanelPtr = new ControlPanel (TemperatureregulatorPtr,
        PRTempControllerPtr);
} // Room

//-----

tempType Room::BenoetigteKesselTempAbfragen()
{
    return TemperatureregulatorPtr->BenoetigteKesselTempLesen();
} // BenoetigteKesselTempAbfragen

//-----

void Room::Fensterzustand(zFensterType Position)
{

```

```

    TemperatureregulatorPtr->FensterzustandSetzen(Position);
    TemperatureregulatorPtr->callBack();

} // Fensterzustand

//-----

void Room::AussentemperaturWeiterleiten(tempType Temperatur)
{
    TemperatureregulatorPtr->AussentemperaturSetzen (Temperatur);
    TemperatureregulatorPtr->callBack();

} // AusentemperaturWeiterleiten

//-----

void Room::RegenZimmer()
{
    WindowPtr->RegenFenster();

} // RegenZimmer

```

3.3.11 Klasse RoomtemperatureController

Header-Datei:

```

#ifndef _ROOMTEMPERATURECONTROLLER_HH
#define _ROOMTEMPERATURECONTROLLER_HH
//
// Header file for class RoomtemperatureController
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse RoomtemperatureController ueberprueft regelmaessig die
// Zimmertemperatur und meldet eine Veraenderung des Zustandes an
// den Temperatureregulator
//
//-----

// forward declaration of class Temperatureregulator
class Temperatureregulator;

#include <SimpleTypes.hh>
#include <ScheduledFct.hh>
#include <Contact.hh>
#include <Temperaturesensor.hh>
#include <Temperatureregulator.hh>

class RoomtemperatureController: public ScheduledFct
{
private:
    enum {CZuHoch, CGut, CZuTief} stateRoomtemperatureController;

    int SectionNr;
    int RoomNr;

    tempType Zieltemperatur;

    tempType ZZimmertemperatur;

```

```

    Temperatureregulator *TemperatureregulatorPtr;
    Temperatursensor *Zimmertemperaturfuehler;
public:

    void ZieltemperaturSetzen (tempType Temperatur);
    void callBack();
    RoomtemperatureController(int InitialSectionNr,
        int InitialRoomNr,
        Temperatureregulator *InitialTempregPtr);
};

#endif

```

Quell-Kode:

```

//
// Implementation file for class RoomtemperatureController
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse RoomtemperatureController ueberprueft regelmaessig die
// Zimmertemperatur und meldet eine Veraenderung des Zustandes an
// den Temperatureregulator
//
//-----

#include <RoomtemperatureController.hh>

//=====
// ClassName: RoomtemperatureController
//=====

RoomtemperatureController::RoomtemperatureController (int InitialSectionNr,
    int InitialRoomNr,
    Temperatureregulator
    *InitialTempregPtr)
{
    // Attribute initialisieren
    SectionNr = InitialSectionNr;
    RoomNr = InitialRoomNr;
    TemperatureregulatorPtr = InitialTempregPtr;
    ZZimmertemperatur = 15;
    Zieltemperatur = 15;

    // Startzustand setzen
    stateRoomtemperatureController = CGut;

    // Instanzen von aggregierten Klassen erzeugen
    Zimmertemperaturfuehler = new Temperatursensor(SectionNr, RoomNr, 0);

    // CallBack regelmaessig ausfuehren.
    registerCallBack();
} // Roomtemperaturecontroller

//-----

void RoomtemperatureController::ZieltemperaturSetzen(tempType Temperatur)
{
    Zieltemperatur = Temperatur;
}

//-----

void RoomtemperatureController::callBack()
{

```

```
switch (stateRoomtemperatureController) {  
  
case CGut:  
    if (ZZimmertemperatur > Zieltemperatur+1) {  
        // Exit-Aktionen  
  
        // neue Konfig. berechnen  
        stateRoomtemperatureController = CZuHoch;  
  
        // Entry-Aktionen  
        TemperatureregulatorPtr->ZimmertemperaturSetzen(ZZimmertemperatur);  
        TemperatureregulatorPtr->TemperaturStatusSetzen(CZimmerTempZuHoch);  
        TemperatureregulatorPtr->callBack();  
    } // if  
    else if (ZZimmertemperatur < Zieltemperatur-1) {  
        // Exit-Aktionen  
  
        // neue Konfig. berechnen  
        stateRoomtemperatureController = CZuTief;  
  
        // Entry-Aktionen  
        TemperatureregulatorPtr->ZimmertemperaturSetzen(ZZimmertemperatur);  
        TemperatureregulatorPtr->TemperaturStatusSetzen(CZimmerTempZuTief);  
        TemperatureregulatorPtr->callBack();  
    } // else if  
    else {  
        // Do-Aktivitaet  
        ZZimmertemperatur = Zimmertemperaturfuehler->AbfrageTemperatur();  
    } // else  
    break;  
  
case CZuHoch:  
    if (ZZimmertemperatur < Zieltemperatur-1) {  
        // Exit-Aktionen  
  
        // neue Konfig. berechnen  
        stateRoomtemperatureController = CZuTief;  
  
        // Entry-Aktionen  
        TemperatureregulatorPtr->ZimmertemperaturSetzen(ZZimmertemperatur);  
        TemperatureregulatorPtr->TemperaturStatusSetzen(CZimmerTempZuTief);  
        TemperatureregulatorPtr->callBack();  
    } // if  
    else if (ZZimmertemperatur >= Zieltemperatur-1 &&  
            ZZimmertemperatur <= Zieltemperatur+1) {  
        // Exit-Aktionen  
  
        // neue Konfig. berechnen  
        stateRoomtemperatureController = CGut;  
  
        // Entry-Aktionen  
        TemperatureregulatorPtr->ZimmertemperaturSetzen(ZZimmertemperatur);  
        TemperatureregulatorPtr->TemperaturStatusSetzen(CZimmerTempGut);  
        TemperatureregulatorPtr->callBack();  
    } // else if  
    else {  
        // Do-Aktivitaet  
        ZZimmertemperatur = Zimmertemperaturfuehler->AbfrageTemperatur();  
    } // else  
    break;  
  
case CZuTief:  
    if (ZZimmertemperatur > Zieltemperatur+1) {  
        // Exit-Aktionen  
  
        // neue Konfig. berechnen  
        stateRoomtemperatureController = CZuHoch;  
  
        // Entry-Aktionen  
        TemperatureregulatorPtr->ZimmertemperaturSetzen(ZZimmertemperatur);
```

```

    TemperatureregulatorPtr->TemperaturStatusSetzen(CZimmerTempZuHoch);
    TemperatureregulatorPtr->callBack();
} // if
else if (ZZimmertemperatur >= Zieltemperatur-1 &&
        ZZimmertemperatur <= Zieltemperatur+1) {
    // Exit-Aktionen

    // neue Konfig. berechnen
    stateRoomtemperatureController = CGut;

    // Entry-Aktionen
    TemperatureregulatorPtr->ZimmertemperaturSetzen(ZZimmertemperatur);
    TemperatureregulatorPtr->TemperaturStatusSetzen(CZimmerTempGut);
    TemperatureregulatorPtr->callBack();
} // else if
else {
    // Do-Aktivitaet
    ZZimmertemperatur = Zimmertemperaturfuehler->AbfrageTemperatur();
} // else
break;

} // switch
} // end of callBack()

```

3.3.12 Klasse Section

Header-Datei:

```

#ifndef _SECTION_HH
#define _SECTION_HH
//
// Header file for class Section
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Section besitzt kein aktives Verhalten. Sie erzeugt bei ihrer
// Instanziierung alle aggregierten Klassen und bildet dann nur noch
// „Durchreichfunktionen“ an.
//
//-----

#include <Slist.hh>
#include <SimpleTypes.hh>
#include <Room.hh>
#include <Timer.hh>

class Section
{
private:

    int SectionNr;

    Timer *Zeit;
    SList<Room*> RoomPtrListe;

public:

    void RegenAnZimmer();
    void AussentemperaturWeiterleiten(tempType Temperatur);
    tempType BenoetigteKesselTempAbfragen();
    Section (int InitialSectionNr,
            SList<double> InitialZimmerFlaechenListe,
            Timer* ZeitPtr);
};

```

```
#endif
```

Quell-Kode:

```
//
// Implementation file for class Section
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Section besitzt kein aktives Verhalten. Sie erzeugt bei ihrer
// Instanziierung alle aggregierten Klassen und bildet dann nur noch
// „Durchreichfunktionen“ an.
//
//-----

#include <Section.hh>

//=====
// ClassName: Section
//=====

Section::Section(int InitialSectionNr,
  SList<double> InitialZimmerFlaechenListe,
  Timer* ZeitPtr)
{

  double AktFlaeche;
  Room *AktRoomPtr;
  int AktRoomNr = 1;
  int i;

  // Attribute initialisieren

  SectionNr = InitialSectionNr;
  Zeit = ZeitPtr;

  // Instanzen von aggregierten Klassen erzeugen
  // Zimmer erzeugen
  RoomPtrListe.clear();
  RoomPtrListe.reset();
  InitialZimmerFlaechenListe.reset();

  while (InitialZimmerFlaechenListe.next(AktFlaeche) == notempty) {
    AktRoomPtr = new Room (SectionNr, AktRoomNr, AktFlaeche, Zeit);
    RoomPtrListe.append (AktRoomPtr);
    AktRoomNr++;
  } // while

} // Section

//-----

void Section::RegenAnZimmer()
{

  Room *AktRoomPtr;

  RoomPtrListe.reset();
  // benachrichtige alle Zimmer, dass es angefangen hat zu regnen

  while (RoomPtrListe.next(AktRoomPtr) == notempty)
    AktRoomPtr->RegenZimmer();

} // RegenAnZimmer

//-----
```

```

void Section::AussentemperaturWeiterleiten(tempType Temperatur)
{
    Room *AktRoomPtr;

    RoomPtrListe.reset();
    // benachrichtige alle Zimmer dass sich die Aussentemperatur
    // geaendert hat.

    while (RoomPtrListe.next(AktRoomPtr) == notempty)
        AktRoomPtr->AussentemperaturWeiterleiten(Temperatur);
} // AussentemperaturWeiterleiten

//-----

tempType Section::BenoetigteKesselTempAbfragen()
{
    tempType MaxKesselTemp=0;
    tempType AktKesselTemp;
    Room *AktRoomPtr;

    RoomPtrListe.reset();
    // frage alle Zimmer nach der benoetigten Kesseltemperatur
    // und gebe das maximum zurueck.

    while (RoomPtrListe.next(AktRoomPtr) == notempty) {
        AktKesselTemp = AktRoomPtr->BenoetigteKesselTempAbfragen();
        if (AktKesselTemp > MaxKesselTemp )
            MaxKesselTemp = AktKesselTemp;
    } // while

    return MaxKesselTemp;
} // BenoetigteKesselTempAbfragen

```

3.3.13 Klasse Temperatureregulator

Header-Datei:

```

#ifndef _TEMPERATUREREGULATOR_HH
#define _TEMPERATUREREGULATOR_HH
//
// Header file for class Temperatureregulator
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Temperatureregulator berechnet bei Bedarf Heiz- und
// Haltetemperaturen und stellt diese am Heizkoerper ein.
//
//-----

// forward delcarations
class Window;
class PRTempController;
class RoomtemperatureController;

#include <ScheduledFct.hh>
#include <SimpleTypes.hh>
#include <Window.hh>
#include <RoomtemperatureController.hh>
#include <Radiator.hh>
#include <math.h>

```

```

class Temperatureregulator : public ScheduledFct
{
private:

    enum {CHeizen, CHalten, CKuehlen} stateTemperaturregelung;
    enum {CAT_pruefen, CEnde} stateKuehlen;

    enum {CANwesenheit, CABwesenheit} stateAnAbwesenheit;
    enum {CANwesenheitFensterNichtOffen,
CANwesenheitFensterOffen } stateAnwesenheit;

    int SectionNr;
    int RoomNr;
    double Flaeche;
    Window *WindowPtr;

    tempType Aussentemperatur;
    tempType Zimmertemperatur;
    zFensterType ZFenster;
    zAnwesenheitType ZAnwesend;
    zZimmerTempType TemperaturStatus;

    durationType ZeitspanneAufheizenNormal;
    durationType CZeitspanneAufheizenDefault;
    durationType ZeitspanneEintrittNormal;
    tempType CABwesenheitsTempDefault;
    tempType CANwesenheitsTempDefault;
    tempType AbwesenheitsTempNormal;
    tempType AnwesenheitsTempNormal;
    tempType BenoetigteKesseltemperatur;
    tempType Zieltemperatur;

    Radiator *RadiatorPtr;
    RoomtemperatureController *RoomtemperatureControllerPtr;

private:
    void HalteTempBerechnen();
    void HeizTempBerechnen();

    void callBackTemperaturregelung();
    void callBackAnAbwesenheit();
    void callBackAnwesenheit();

public:

    void callBack();
    fehlerType ZeitspanneAufheizenNormalSetzen(durationType Zeitspanne);
    fehlerType ZeitspanneAufheizenAufDefault();
    void AbwesenheitsTempNormalSetzen(tempType Temperatur);
    void AbwesenheitsTempAufDefault();
    void AnwesenheitsTempNormalSetzen(tempType Temperatur);
    void AnwesenheitsTempAufDefault();

    void AussentemperaturSetzen (tempType Temperatur);
    void ZimmertemperaturSetzen (tempType Temperatur);
    void AnwesenheitSetzen (zAnwesenheitType Anwesenheit);
    void FensterzustandSetzen (zFensterType Position);
    tempType BenoetigteKesselTempLesen();
    void TemperaturStatusSetzen (zZimmerTempType Zustand);
    void ZeitspanneEintrittNormalSetzen (durationType Zeitspanne);
    Temperatureregulator (int InitialSectionNr,
int InitialRoomNr,
double InitialFlaeche,
Window *InitialWindowPtr);

};

#endif

```

Quell-Kode:

```

//
// Implementation file for class Temperatureregulator
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Temperatureregulator berechnet bei Bedarf Heiz- und
// Haltetemperaturen und stellt diese am Heizkoerper ein.
//
//-----

#include <Temperatureregulator.hh>

//=====
// ClassName: Temperatureregulator
//=====

Temperatureregulator::Temperatureregulator (int InitialSectionNr,
      int InitialRoomNr,
      double InitialFlaeche,
      Window *InitialWindowPtr)
{
  // Attribute initialisieren
  SectionNr = InitialSectionNr;
  RoomNr = InitialRoomNr;
  Flaeche = InitialFlaeche;
  WindowPtr = InitialWindowPtr;

  Aussentemperatur = 15;
  Zimmertemperatur = 15;
  ZFenster = CFensterGeschlossen;
  ZAnwesend = CNiemandDa;
  TemperaturStatus = CZimmerTempGut;

  ZeitspanneAufheizenNormal = 15;
  CZeitspanneAufheizenDefault = 15;
  ZeitspanneEintrittNormal = 10;
  CAbwesenheitsTempDefault = 15;
  CAnwesenheitsTempDefault = 20;
  AbwesenheitsTempNormal = 15;
  AnwesenheitsTempNormal = 20;
  BenoetigteKesseltemperatur = 15;
  Zieltemperatur = 15;

  // Startzustand setzen
  stateTemperaturregelung = CHalten;
  stateAnAbwesenheit = CAbwesenheit;

  // Instanzen von aggregierten Klassen erzeugen
  RoomtemperatureControllerPtr = new RoomtemperatureController (SectionNr,
    RoomNr,
    this);
  RadiatorPtr = new Radiator (SectionNr, RoomNr);

  // Callback regelmaessig ausfuehren.
  registerCallBack();
} // Temperatureregulator

//-----

void Temperatureregulator::HalteTempBerechnen()
{
  tempType diff;

```

```

double WA; //Waermeabgabe
double WV; //Waermeverlust
double EB; //Energiebedarf

// Berechnung laut Formel Gleichgewicht

diff = Aussentemperatur-Zimmertemperatur;
if (diff==0)
    diff = 1;
WA = 0.01*Flaeche*diff;
switch (ZFenster) {
case CFensterGeschlossen:
    WV=0.005*Flaeche*diff;
    break;
case CFensterGekippt:
    WV=0.0075*Flaeche*diff;
    break;
case CFensterOffen:
    WV=0.01*Flaeche*diff;
    break;
} // switch
EB=20*Flaeche*diff;
// zur berechneten Kesseltemperatur werden noch 0.5 addiert, damit
// die Umwandlung von int zu double berichtigt wird, da bei der Rundung
// der double-Wert hinter dem Komma abgeschnitten wird
// Bsp.: aus 3.8 wuerde bei Rundung 3 werden. Jetzt 3.8+0.5 = 4.3 ergibt 4
BenoetigteKesseltemperatur= (int)
    (((WV/WA)*(Zieltemperatur-Aussentemperatur)+Zieltemperatur))+0.5);
} // HalteTempBerechnen

//-----

void Temperatureregulator::HeizTempBerechnen()
{
    tempType diff;
    double Restzeit;
    double WA; //Waermeabgabe
    double WV; //Waermeverlust
    double EB; //Energiebedarf
    double e_hoch;

    // Berechnung laut Formel Aufheizen

    Restzeit=(ZeitspanneAufheizenNormal-ZeitspanneEintrittNormal)*60;
    diff = Aussentemperatur-Zimmertemperatur;
    if (diff == 0)
        diff = 1;
    WA = 0.01*Flaeche*diff;
    switch (ZFenster) {
    case CFensterGeschlossen:
        WV=0.005*Flaeche*diff;
        break;
    case CFensterGekippt:
        WV=0.0075*Flaeche*diff;
        break;
    case CFensterOffen:
        WV=0.01*Flaeche*diff;
        break;
    } // switch
    EB=20*Flaeche*diff;
    e_hoch = exp (((WA+WV)/-EB)*Restzeit);
    // zur berechneten Kesseltemperatur werden noch 0.5 addiert, damit
    // die Umwandlung von int zu double berichtigt wird, da bei der Rundung
    // der double-Wert hinter dem Komma abgeschnitten wird
    // Bsp.: aus 3.8 wuerde bei Rundung 3 werden. Jetzt 3.8+0.5 = 4.3 ergibt 4
    BenoetigteKesseltemperatur =
        (int) (( (

```

```

        ((WA+WV)/WA)*
        ((Zieltemperatur-(Zimmertemperatur*e_hoch))/
        (1-e_hoch))
    )
    - (WV*Aussentemperatur/WA
    )+0.5);
} // HeizTempBerechnen

//-----

void Temperatureregulator::callBack()
{
    // callBack Methoden von Zimmer entsprechend den parallelen Subzustaenden
    // im State-Diagramm

    callBackAnAbwesenheit();
    callBackTemperaturregelung();
} // callBack

//-----

void Temperatureregulator::callBackTemperaturregelung()
{
    switch (stateTemperaturregelung) {

case CHalten:
    if (TemperaturStatus == CZimmerTempZuHoch) {
        // Exit-Aktionen

        // neue Konfig berechnen
        stateTemperaturregelung = CKuehlen;
        stateKuehlen = CAT_pruefen;

        // Entry-Aktionen
        RadiatorPtr->ThermostatSetzen(0);
        BenoetigteKesseltemperatur = 0;
    } // if
    else if (TemperaturStatus == CZimmerTempZuTief) {
        // Exit-Aktionen

        // neue Konfig berechnen
        stateTemperaturregelung = CHeizen;

        // Entry-Aktionen
        HeizTempBerechnen();
        RadiatorPtr->ThermostatSetzen(BenoetigteKesseltemperatur);
    } // else if
    else {
        // Do-Aktivitaet
    } // else
    break;

case CHeizen:
    if (TemperaturStatus == CZimmerTempZuHoch) {
        // Exit-Aktionen

        // neue Konfig berechnen
        stateTemperaturregelung = CKuehlen;
        stateKuehlen = CAT_pruefen;

        // Entry-Aktionen
        RadiatorPtr->ThermostatSetzen(0);
        BenoetigteKesseltemperatur = 0;
    } // if

```

```
else if (TemperaturStatus == CZimmerTempGut) {
    // Exit-Aktionen

    // neue Konfig berechnen
    stateTemperaturregelung = CHalten;

    // Entry-Aktionen
    HalteTempBerechnen();
    RadiatorPtr->ThermostatSetzen(BenoetigteKesseltemperatur);
} // else if
else {
    // Do-Aktivitaet

} // else
break;

case CKuehlen:
    if (TemperaturStatus == CZimmerTempGut) {
        // Exit-Aktionen

        // neue Konfig berechnen
        stateTemperaturregelung = CHalten;

        // Entry-Aktionen
        HalteTempBerechnen();
        RadiatorPtr->ThermostatSetzen(BenoetigteKesseltemperatur);
    } // if
    else if (TemperaturStatus == CZimmerTempZuTief) {
        // Exit-Aktionen

        // neue Konfig berechnen
        stateTemperaturregelung = CHEizen;

        // Entry-Aktionen
        HeizTempBerechnen();
        RadiatorPtr->ThermostatSetzen(BenoetigteKesseltemperatur);
    } // else if
    else {
        // Do-Aktivitaet

        // Substates behandeln
        switch (stateKuehlen) {

            case CAT_pruefen:
if (ZAnwesend == CJemandDa &&
    AnwesenheitsTempNormal > Aussentemperatur) {
    // Exit - Aktionen

    // Event senden
    WindowPtr->ZimmerKuehlen();

    // neue Konfig berechnen
    stateKuehlen = CEnde;

    // Entry-Aktionen
} // if
else {
    // Do-Aktivitaet
} // else
break;

            case CEnde:
// Keine Aktionen
break;
        } // switch (stateKuehlen)

    } // else
    break;

} // switch
```

```
} // callBackTemperaturregelung

//-----

void Temperatureregulator::callBackAnAbwesenheit()
{
    switch (stateAnAbwesenheit) {

    case CABwesenheit:
        if (ZAnwesend == CJemandDa) {
            // Exit-Aktionen

            // neue Konfig. berechnen
            stateAnAbwesenheit = CANwesenheit;
            stateAnwesenheit = CANwesenheitFensterNichtOffen;

            // Entry-Aktionen
            RoomtemperatureControllerPtr->ZieltemperaturSetzen(AnwesenheitsTempNormal);
            Zieltemperatur=AnwesenheitsTempNormal;
            RoomtemperatureControllerPtr->callBack();
        } // if
        else {
            // Do-Aktivitaet
        } // else
        break;

    case CANwesenheit:
        if (ZAnwesend == CNiemandDa) {
            // Exit-Aktionen

            // neue Konfig. berechnen
            stateAnAbwesenheit = CABwesenheit;

            // Entry-Aktionen
            RoomtemperatureControllerPtr->ZieltemperaturSetzen(AbwesenheitsTempNormal);
            Zieltemperatur=AbwesenheitsTempNormal;
            RoomtemperatureControllerPtr->callBack();
        } // if
        else {
            // Do-Aktivitaet

            // Substates behandeln
            callBackAnwesenheit();
        } // else
        break;
    } // switch
} // callBackAnAbwesenheit

//-----

void Temperatureregulator::callBackAnwesenheit()
{
    switch (stateAnwesenheit) {
    case CANwesenheitFensterNichtOffen:
        if (ZFenster == CFensterOffen) {
            // Exit-Aktionen

            // neue Konfig. berechnen
            stateAnwesenheit = CANwesenheitFensterOffen;

            // Entry-Aktionen
            RoomtemperatureControllerPtr->ZieltemperaturSetzen(AbwesenheitsTempNormal);
            Zieltemperatur=AbwesenheitsTempNormal;
            RoomtemperatureControllerPtr->callBack();
        }
    }
}
```

```

    } // if
    else {
        // Do-Aktivitaet
    } // else
    break;

case CANwesenheitFensterOffen:
    if (ZFenster == CFensterGekippt) {
        // Exit-Aktionen

        // neue Konfig. berechnen
        stateAnwesenheit = CANwesenheitFensterNichtOffen;

        // Entry-Aktionen
        RoomtemperatureControllerPtr->ZieltemperaturSetzen(AnwesenheitsTempNormal);
        Zieltemperatur=AnwesenheitsTempNormal;
        RoomtemperatureControllerPtr->callBack();
    } // if
    else if (ZFenster == CFensterGeschlossen) {
        // Exit-Aktionen

        // neue Konfig. berechnen
        stateAnwesenheit = CANwesenheitFensterNichtOffen;

        // Entry-Aktionen
        RoomtemperatureControllerPtr->ZieltemperaturSetzen(AnwesenheitsTempNormal);
        Zieltemperatur=AnwesenheitsTempNormal;
        RoomtemperatureControllerPtr->callBack();
    } // else if
    else {
        // Do-Aktivitaet
    } // else
    break;

} // switch

} // callBackAnwesenheit

//-----

fehlerType Temperatureregulator::ZeitspanneAufheizenNormalSetzen (durationType Zeitspanne)
{
    // ZeitspanneAufheizenNormal mus mind. 5 Minuten gresser sein
    // als die ZeitspanneEintrittNormal, sonst Wert ungueltig

    if ((Zeitspanne - ZeitspanneEintrittNormal) >= 5) {
        ZeitspanneAufheizenNormal = Zeitspanne;
        return CKeinFehler;
    } // if
    else
        // Differenz < 5 Minuten
        return CWertUnguelteig;

} // ZeitspanneAufheizenNormalSetzen

//-----

fehlerType Temperatureregulator::ZeitspanneAufheizenAufDefault()
{

    // ZeitspanneAufheizenNormal muss mind. 5 Minuten groesser sein
    // als die ZeitspanneEintrittNormal, sonst Wert ungueltig

    if ((CZeitspanneAufheizenDefault-ZeitspanneEintrittNormal)>=5) {
        ZeitspanneAufheizenNormal = CZeitspanneAufheizenDefault;
        return CKeinFehler;
    } // if
    else
        // Differenz < 5 Minuten

```

```
        return CWertUnguelting;
    } // ZeitspanneAufheizenAufDefault

//-----
void Temperatureregulator::AbwesenheitsTempNormalSetzen(tempType Temperatur)
{
    AbwesenheitsTempNormal = Temperatur;

    // ggf. Zieltemperatur aktualisieren
    if (stateAnAbwesenheit == CABwesenheit ||
        (stateAnAbwesenheit == CAnwesenheit &&
         stateAnwesenheit == CAnwesenheitFensterOffen)) {
        RoomtemperatureControllerPtr->ZieltemperaturSetzen (Temperatur);
        Zieltemperatur = Temperatur;
    }
} // AbwesenheitsTempNormalSetzen

//-----
void Temperatureregulator::AbwesenheitsTempAufDefault()
{
    AbwesenheitsTempNormal = CABwesenheitsTempDefault;

    // ggf. Zieltemperatur aktualisieren
    if (stateAnAbwesenheit == CABwesenheit ||
        (stateAnAbwesenheit == CAnwesenheit &&
         stateAnwesenheit == CAnwesenheitFensterOffen)) {
        RoomtemperatureControllerPtr->ZieltemperaturSetzen (CABwesenheitsTempDefault);
        Zieltemperatur = CABwesenheitsTempDefault;
    }
} // AbwesenheitsTempAufDefault

//-----
void Temperatureregulator::AnwesenheitsTempNormalSetzen(tempType Temperatur)
{
    AnwesenheitsTempNormal = Temperatur;

    // ggf. Zieltemperatur aktualisieren
    if (stateAnAbwesenheit == CAnwesenheit &&
        stateAnwesenheit == CAnwesenheitFensterNichtOffen) {
        RoomtemperatureControllerPtr->ZieltemperaturSetzen (Temperatur);
        Zieltemperatur=Temperatur;
    }
} // AnwesenheitsTempNormalSetzen

//-----
void Temperatureregulator::AnwesenheitsTempAufDefault()
{
    AnwesenheitsTempNormal = CAnwesenheitsTempDefault;

    // ggf. Zieltemperatur aktualisieren
    if (stateAnAbwesenheit == CAnwesenheit &&
        stateAnwesenheit == CAnwesenheitFensterNichtOffen) {
        RoomtemperatureControllerPtr->ZieltemperaturSetzen (CAnwesenheitsTempDefault);
        Zieltemperatur=CAnwesenheitsTempDefault;
    }
}
```

```
    }  
} // AnwesenheitsTempAufDefault  
  
//-----  
void Temperatureregulator::AussentemperaturSetzen (tempType Temperatur)  
{  
    Aussentemperatur = Temperatur;  
} // AussentemperaturSetzen  
//-----  
void Temperatureregulator::ZimmertemperaturSetzen (tempType Temperatur)  
{  
    Zimmertemperatur = Temperatur;  
} // ZimmertemperaturSetzen  
//-----  
void Temperatureregulator::AnwesenheitSetzen (zAnwesenheitType Anwesenheit)  
{  
    ZAnwesend = Anwesenheit;  
} // AnwesenheitSetzen  
//-----  
void Temperatureregulator::FensterzustandSetzen (zFensterType Position)  
{  
    ZFenster = Position;  
} // FensterzustandSetzen  
//-----  
tempType Temperatureregulator::BenoetigteKesselTempLesen()  
{  
    return BenoetigteKesseltemperatur;  
} // BenoetigteKesselTempLesen  
//-----  
void Temperatureregulator::TemperaturStatusSetzen (zZimmerTempType Zustand)  
{  
    TemperaturStatus = Zustand;  
} // TemperaturStatusSetzen  
//-----  
void Temperatureregulator::ZeitspanneEintrittNormalSetzen (durationType Zeitspanne)  
{
```

```

    ZeitspanneEintrittNormal = Zeitspanne;
} // ZeitspanneEintrittNormalSetzen

```

3.3.14 Klasse Temperaturesensor

Header-Datei:

```

#ifndef _TEMPERATURESENSOR_HH
#define _TEMPERATURESENSOR_HH
//
// Header file for class Temperaturesensor
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Temperaturesensor stellt den Zugriff auf die Temperatursensoren
// bereit.
// Da es sich um eine spezielle Hardware handelt, ist diese Klasse vom
// Portbaustein abgeleitet.
//
//-----

#include <SimpleTypes.hh>
#include <Portbaustein.hh>

class Temperaturesensor:public Portbaustein
{
private:

    int SectionNr;
    int RoomNr;
    int PortNr;

public:

    tempType AbfrageTemperatur();
    Temperaturesensor (int InitialSectionNr,
        int InitialRoomNr,
        int InitialPortNr);

};

#endif

```

Quell-Kode:

```

//
// Implementation file for class Temperaturesensor
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Temperaturesensor stellt den Zugriff auf die Temperatursensoren
// bereit.
// Da es sich um eine spezielle Hardware handelt, ist diese Klasse vom
// Portbaustein abgeleitet.
//
//-----

#include <Temperaturesensor.hh>

//=====
// ClassName: Temperaturesensor

```

```
//=====

Temperaturesensor::Temperaturesensor(int InitialSectionNr,
    int InitialRoomNr,
    int InitialPortNr)
{
    SectionNr = InitialSectionNr;
    RoomNr = InitialRoomNr;
    PortNr = InitialPortNr;
} // Temperaturesensor

tempType Temperaturesensor::AbfrageTemperatur()
{
    return CallIntInputPort (SectionNr, RoomNr, PortNr);
} // AbfrageTemperatur
```

3.3.15 Klasse Window

Header-Datei:

```
#ifndef _WINDOW_HH
#define _WINDOW_HH
//
// Header file for class Window
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Window fragt regelmaessig den Offen- und Kippkontakt ab
// und meldet einen veraenderten Fensterzustand an die Klasse Room.
// Weiterhin werden Methoden zum Kuehlen des Zimmers und zur Reaktion
// auf Regen zur Verfuegung gestellt.
//
//-----

// forward declarations
class Room;
class Temperatureregulator;

#include <SimpleTypes.hh>
#include <ScheduledFct.hh>
#include <Motor.hh>
#include <Contact.hh>
#include <Room.hh>

class Window: public ScheduledFct
{
private:

    enum {Coffen, Cgeschlossen, Cgekippt} stateWindow;

    int SectionNr;
    int RoomNr;

    Motor *KippmotorPtr;
    Motor *SchwenkmotorPtr;
    Contact *OffenkontaktPtr;
    Contact *KippkontaktPtr;

    Room* RoomPtr;

    int ZOffenkontakt;
```

```

    int ZKippkontakt;

private:

    void FensterKippen();

public:

    void callBack();
    void RegenFenster();
    void ZimmerKuehlen();
    Window (int InitialSectionNr,
            int InitialRoomNr,
            Room *InitialRoomPtr);
};

#endif

```

Quell-Kode:

```

//
// Implementation file for class Window
//
// Author: Wolfgang Wagenbichler
//-----
//
// Die Klasse Window fragt regelmaessig den Offen- und Kippkontakt ab
// und meldet einen veraenderten Fensterzustand an die Klasse Room.
// Weiterhin werden Methoden zum Kuehlen des Zimmers und zur Reaktion
// auf Regen zur Verfuegung gestellt.
//
//-----

#include <Window.hh>

//=====
// ClassName: Window
//=====

Window::Window (int InitialSectionNr, int InitialRoomNr,
                Room *InitialRoomPtr)
{
    // Attribute initialisieren
    SectionNr = InitialSectionNr;
    RoomNr = InitialRoomNr;
    RoomPtr = InitialRoomPtr;

    // Startzustand setzen
    stateWindow = Cgeschlossen;
    ZOffenkontakt = geschlossen;
    ZKippkontakt = geschlossen;

    // Instanzen von aggregierten Klassen erzeugen
    KippmotorPtr = new Motor (SectionNr, RoomNr, 6);
    SchwenkmotorPtr = new Motor (SectionNr, RoomNr, 7);

    OffenkontaktPtr = new Contact (SectionNr, RoomNr, 5);
    KippkontaktPtr = new Contact (SectionNr, RoomNr, 4);

    // CallBack regelmaessig ausfuehren.
    registerCallBack();
} // Window

//-----

```

```
void Window::FensterKippen()
{
    // Falls das Fenster offen ist, muss es zum kippen zuerst geschlossen
    // werden.

    if (stateWindow == Coffen) {
        SchwenkmotorPtr->Schliessen();
    }

    // dann eigentliches Kippen

    KippmotorPtr->Oeffnen();

    // aktualisiere Zustand der Variablen
    ZKippkontakt = offen;
    ZOffenkontakt = geschlossen;
    // oder auch ZKippkontakt = KippkontaktPtr->AbfrageKontakt()
} // FensterKippen

//-----

void Window::RegenFenster()
{
    if (stateWindow == Coffen) {
        // Exit-Aktionen

        // neue Konfig berechnen
        FensterKippen();
        stateWindow = Cgekippt;

        // Entry-Aktionen
        RoomPtr->Fensterzustand (CFensterGekippt);
    }
} // RegenFenster

//-----

void Window::ZimmerKuehlen()
{
    if (stateWindow == Cgeschlossen) {
        // Exit-Aktionen

        // neue Konfig berechnen
        FensterKippen();
        stateWindow=Cgekippt;

        // Entry-Aktionen
        RoomPtr->Fensterzustand (CFensterGekippt);
    } // if
} // ZimmerKuehlen

//-----

void Window::callBack()
{
    switch (stateWindow) {

    case Cgeschlossen:
        if (ZOffenkontakt == offen) {
            // Exit-Aktionen

            // neue Konfig berechnen
            stateWindow = Coffen;

            // Entry-Aktionen
            RoomPtr->Fensterzustand (CFensterOffen);
        }
    }
}
```

```
} // if
else if (ZKippkontakt == offen) {
    // Exit-Aktionen

    // neue Konfig berechnen
    stateWindow=Cgekippt;

    // Entry-Aktionen
    RoomPtr->Fensterzustand (CFensterGekippt);
} // else if
else {
    // Do -Aktivitaet
    ZOffenkontakt = OffenkontaktPtr->AbfrageKontakt();
    ZKippkontakt = KippkontaktPtr->AbfrageKontakt();
}
break;

case Coffen:
    if (ZOffenkontakt == geschlossen) {
        // Exit-Aktionen

        // neue Konfig berechnen
        stateWindow = Cgeschlossen;

        // Entry-Aktionen
        RoomPtr->Fensterzustand (CFensterGeschlossen);
    } // if
    else {
        // Do-Aktivitaet
        ZOffenkontakt = OffenkontaktPtr->AbfrageKontakt();
        ZKippkontakt = KippkontaktPtr->AbfrageKontakt();
    }
    break;

case Cgekippt:
    if (ZKippkontakt == geschlossen) {
        // Exit-Aktionen

        // neue Konfig berechnen
        stateWindow = Cgeschlossen;

        // Entry-Aktionen
        RoomPtr->Fensterzustand (CFensterGeschlossen);
    } // if
    else {
        // Do-Aktivitaet
        ZOffenkontakt = OffenkontaktPtr->AbfrageKontakt();
        ZKippkontakt = KippkontaktPtr->AbfrageKontakt();
    }
    break;

} // switch

} // callBack
```

3.4 Verifikation

In diesem Abschnitt wird der Komponenten-Kode verifiziert. In Kapitel 3.4.1 wird angegeben, welche Aspekte bei der Verifikation zu beachten sind. Die Ergebnisse der Verifikation werden ab Kapitel 3.4.3 festgehalten.

3.4.1 Verifikations-Checklisten

In diesem Abschnitt sind die Aspekte aufgelistet, die bei der Verifikation der einzelnen Dokumente zu beachten sind.

Überprüfe zu jeder Klasse die .hh und .cc-Datei auf:

Einhaltung der Dokumentationsrichtlinien:

1. Wurde die C++-Syntax eingehalten?

Da die Inspektion des Codes stattfindet, wenn der Code bereits (in ein Objectfile) übersetzt ist, erübrigt sich eine Überprüfung auf syntaktische C++-Fehler.

2. Wurden die Programmierrichtlinien eingehalten?

Hier ist insbesondere die Einhaltung der Programmierrichtlinien zu überprüfen (Namenskonventionen, Vermeidung von bestimmten Anweisungstypen, Gestaltung und Dokumentation des Codes, usw.).

Interne Konsistenz des Codes:

1. *Wenn einer Variablen ein Wert zugewiesen wird (die Variable wird definiert):* Wird die Variable im weiteren verwendet (referenziert)?

Zwei Arten von Anomalien können gefunden werden: DD (Definiert, Definiert) und DU (Definiert, undefiniert). Diese Anomalien sind Fehler, die nicht direkt zu einem Fehlverhalten des Systems führen.

2. *Wenn eine Variable referenziert wird:* Ist der Variablen zuvor ein Wert zugewiesen worden?

Auf diese Weise werden UR (undefiniert, referenziert) Anomalien gefunden. UR-Anomalien sind Fehler, die zu einem Fehlverhalten des Systems führen.

Konsistenz zum Entwurf:

1. Sind alle Attribute aus den Classtablets im Code enthalten?
2. Sind alle Methoden mit Parametern und Rückgabetyphen aus den Classtablets im Code enthalten?
3. Geben die implementierten Methoden die gewünschte Funktionalität (beschrieben in den Methodenschnittstellen des Entwurfs und den Aktivitätsdiagrammen) wieder?
4. Sind alle Datentypen aus dem UML-Entwurf in der Datei SimpleTypes.hh definiert?
5. Sind Assoziationen und Aggregation durch die im Entwurf angegebenen Pointer realisiert?
6. Importiert die Klasse die im Entwurf unter Imports aufgeführten Klassen?
7. Entspricht der Code dem zugehörigem Zustandsdiagramm?

Hier ist insbesondere zu überprüfen, ob die Richtlinien für die Ableitung von C++-Code aus OMT/UML-Zustandsdiagrammen eingehalten wurden.

3.4.2 Klassifikation von Fehlern

Die während der Verifikation aufgedeckten Fehler werden den folgenden Fehlerklassen zugeordnet. Um die Fehler zu klassifizieren, bitte die Fehlerklassen in der angegebenen Reihenfolge durchgehen:

Unvollständigkeit (U):

Es fehlen Informationen zu einem Objekt. Beispielsweise ist eine Klasse nicht im Data-Dictionary beschrieben, obwohl sie im Klassendiagramm vorhanden ist.

Falsche Information (F):

Eine gegebene Information zu einem Objekt ist offensichtlich falsch. Beispielsweise kann ein Zustand im Statediagramm der Klasse Fenster nie erreicht werden.

Überspezifikation und irrelevante Information (R):

Eine gegebene Information ist überflüssig, da sie nicht benötigt wird. Beispielsweise existiert eine Klasse Kesseltemperatur im Klassendiagramm. Diese wird aber für die erwartete Systemfunktionalität nicht benötigt.

Inkonsistenzen oder Widersprüche (I):

Eine gegebene Information zu einem Objekt ist inkonsistent oder widersprüchlich in bezug auf eine andere Information zu diesem Objekt. Beispielsweise ist im Klassendiagramm eine Methode einer Klasse definiert, die im Statediagramm mit ähnlichem (aber nicht gleichem) Bezeichner auftaucht.

Vorwärtsreferenzen und sonstige Fehler (S):

Alle Fehler, die sich nicht den anderen Fehlerklassen zuordnen lassen.

3.4.3 Ergebnisse der Verifikation

In der nachfolgenden Tabelle sind alle Fehler eingetragen, die während der Verifikation gefundenen wurden. Jeder Fehler ist dabei einer der oben angegebenen Fehlerklassen zugeordnet.

FNr	betroffenes Diagramm	betroffenes Element	allgemeine Fehlerbeschreibung	Fehlerklasse
I-FNr 1	Header-File	Klasse Section	Attribute sind als public definiert	I
I-FNr 2	Code-File	Klasse Room	Kommentare fehlen	U
I-FNr 3	Code-File	Klasse Temperatureregulator	State-Diagramm nicht nach Richtlinien implementiert (daher fehlerhafte Funktion)	I

Tabelle 1: Verifikationsergebnisse

Die Strategien zur Behebung der oben aufgeführten Fehler werden in der nachfolgenden Tabelle aufgeführt:

Referenz auf FNr	Strategie zur Behebung des Fehlers	Behebung des Fehlers durchgeführt ? (J/N)
I-FNr 1	Attribute private definieren	J
I-FNr 2	Kommentare hinzufügen	J
I-FNr 3	Implementierung nach Richtlinien	J

Tabelle 2: Fehlerbehebungsstrategien

3.5 Validation

Ein White-Box Test wurde nicht durchgeführt.