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.
EE4	Die Anbindung des Portbausteins an den Simulator erfolgt mit Hilfe von Interprozeßkommunikation.
EE5	Die Benutzungsschnittstelle wird als autonomes Teilsystem realisiert.

Tabelle 1: Entwurfsentscheidungen

2.1.2 Portbaustein

Alle Sensoren und Aktuatoren werden über einen Portbaustein angesteuert. Jeder Typ von Sensoren und Aktuatoren besitzt eine eigene Methode zur Eingabe und Ausgabe von Werten.

Es gibt folgende Arten von Eingabe- und Ausgabeports:

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

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

- Bit-Eingabeports für Kontakte: motion detector, pushbutton, status line
- Integer-Eingabeports für Temperaturen und Helligkeitssensoren: room temperature sensor, water temperature sensor, outdoor temperature sensor, outdoor light sensor
- Bit-Ausgabeports für einfache Aktuatoren: pulse
- Integer-Ausgabeports für komplexere Aktoren: blind, radiator valve, dimmable light

Es werden folgende Portzuordnungen festgelegt:

Port	**PortNr**	**Typ des Ports**	**LineNr1**	**LineNr2**	**SensorAktuatorNr**
room temperature sensor	0	IntInput	Section-Nr	Zimmer-Nr	rts1 - rts3
water temperature sensor	1	IntInput	Section-Nr	Zimmer-Nr	wts1 - wts3
motion detector	2	BitInput	Section-Nr	Zimmer-Nr	imd1 - imd3
status line	3	BitInput	Section-Nr	Zimmer-Nr	sl1 - sl2
outdoor light sensor	4	IntInput	0	0	ols1 - ols6
outdoor temperature sensor	5	IntInput	0	0	ots
light	6	BitOutput	Section-Nr	Zimmer-Nr	pulse1 - pulse2

Tabelle 2: Portzuordnung

Port	**PortNr**	**Typ des Ports**	**LineNr1**	**LineNr2**	**SensorAktuatorNr**
dimmable light	7	IntOutput	Section-Nr	Zimmer-Nr	dll1 - dll2
blind	7	IntOutput	Section-Nr	Zimmer-Nr	blind
radiator valve	8	IntOutput	Section-Nr	Zimmer-Nr	rad. valve

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 Synchronisationsproblemen 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 von Werten durch den Benutzer oder den Hausmeister erfolgt über Benutzungsschnittstellen (Control Panel). Diese wird durch ein autonomes Subsystem realisiert. Die Kommunikation zwischen Benutzungsschnittstelle und System erfolgt über die Strukturklasse, in der das ControlPanel integriert ist, z.B. beim ControlPanel des Raumes über die Methoden der Klasse *Room*.

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 ScheduledFct

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

2.2.1.2 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.

2.2.1.3 Klasse Timer

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