Kapitel 4

Komponenten-Entwicklung

4.1 Allgemeine Datentypen

4.1.1 Wieder verwendete Datentypen

Folgende Datentypen wurden im Rahmen früherer Projekte schon erstellt:

```
typedef unsigned int timeCompType
durationType timeCompType
```

Zusätzlich wurden die Klassen *MsgHeader* und *SocketStream* aus [Cod97] übernommen. Dabei wurde sie leicht modifiziert: Die Operatoren << und >> wurden aus der Klasse *MsgHeader* entfernt und in die Klasse *SockStream* integriert. Diese Veränderungen wurden vorgenommen, weil die Include-Struktur der Originalversionen innerhalb dieses Systems zu Übersetzungsproblemen führte.

4.1.2 C++ Klassen und Typdefinitionen

4.1.2.1 Die Klasse *MsgHeader*

```
#ifndef _MSGHEADER_H
#define _MSGHEADER_H
//
// Header file for class MsgHeader
//
//
// Author: Oliver Swienty (Codex)
//        modified by Sascha Werner
//-----------------------------------------------------------------------------------------
//
//     Every message that is exchanged between the user interface and  the
//     control system must start with a header; the purpose of the message
//     is identified by the value of the attribute msgType; so the
//     receiver can handle the message
//
//     Modify: the operators << and >> are moved to the class SocketStream
//
//-----------------------------------------------------------------------------------------


#include <msgType.h>
```

```
class MsgHeader
{

  unsigned long magic;
  // characteristic bit pattern
  // used to identify a byte stream as an object of this type
  static const unsigned long magicPattern;

msgType type;

public:
  MsgHeader (void);
  MsgHeader (msgType t);

  msgType getMsgType (void);
  bool isMsgHeader (void);
  // return true if the memory area considered a MsgHeader
  // is really a MsgHeader; this is founded by the presence of
  // the „magic“ bit pattern

}; // class MsgHeader


#endif // _MSGHEADER_H
```

4.1.2.2 Die Klasse *SocketStream*

```
#ifndef _SOCKET_STREAM_H
#define _SOCKET_STREAM_H
//
// Header file for class SocketStream
//
//
// Author: Oliver Swienty (Codex)
//        modified by Sascha Werner
//-----------------------------------------------------------------------------------------
//
//     this class delivers operators for a streambased socket connection
//     for the elementary C(++)-Data-Types
//
//     Modify: the operators << and >> are moved from the class MsgHeader and are
//            integrated in this class.
//            The same is true for the following types:
//            msgType and char
//
//-----------------------------------------------------------------------------------------



#include <stdlib.h>
#include <unistd.h>

// with msgHeader.h is also msgType.h included
#include <msgHeader.h>

class SocketStream  {
private:
  int fd;
  bool nullRead;
  // stores if the stream has read something or nothing
  // can be necessary for non blocking connections

public:
  SocketStream();
  // a default constructor (does nothing)
  SocketStream(int fd);
  // constructor which binds a filedescriptor to the SocketStream
```



```
  // operators for elementary data-types
  SocketStream& operator<<(char c);
  SocketStream& operator>>(char& c);
  SocketStream& operator<<(int i);
  SocketStream& operator>>(int& i);
  SocketStream& operator<<(long l);
  SocketStream& operator>>(long& l);
  SocketStream& operator<<(unsigned int u);
  SocketStream& operator>>(unsigned int& u);
  SocketStream& operator<<(unsigned long ul);
  SocketStream& operator>>(unsigned long& ul);
  SocketStream& operator<<(float f);
  SocketStream& operator>>(float& f);
  SocketStream& operator<<(double d);
  SocketStream& operator>>(double& d);

  // opertators for further data-types needed in WASTE
  SocketStream& operator<<(MsgHeader m);
  SocketStream& operator>>(MsgHeader &m);
  SocketStream& operator<<(msgType t);
  SocketStream& operator>>(msgType &t);

  bool emptyRead();
  // this method returns TRUE iff the last read was empty, i.e.
  // nothing was read (can occur in non-blocking connections)
};
#endif // _SOCKET_STREAM_H
```

4.1.3 Datentypen aus dem Entwurf

Im Rahmen des Entwurfs wurden folgende Datentypen erstellt:

C++ Typdefinitionen	Beschreibung
typedef enum {	An Hand dieser Nachrichtentypen erkennt der Message-Handler, was für eine Art Nachricht er empfangen hat und ob noch Parameter folgen. Die Nachrichtentypen haben dabei folgende Bedeutungen:
CmsgTypeStartIndicate,	Die Benutzungsschnittstelle erklärt, daß sie bereit ist und der Monitor aktiviert werden kann.
CmsgTypeSetExitRemarkTime,	Setze die *ZeitspanneVerlassenNormal* auf den Wert, der als Parameter folgt.
CmsgTypeSetEntryRemarkTime,	Setze die *ZeitspanneEintrittNormal* auf den Wert, der als Parameter folgt.
CmsgTypeSetHeatingUpTime,	Setze die *ZeitspanneAufheizenNormal* auf den Wert, der als Parameter folgt.
CmsgTypeSetExitRemarkTimeDe-fault,	Setze die *ZeitspanneVerlassenNormal* auf den Defaultwert bzw. setze den Defaultwert der *ZeitspanneVerlassenNormal* auf den Wert, der als Parameter folgt.
CmsgTypeSetEntryRemarkTime-Default,	Setze die *ZeitspanneEintrittNormal* auf den Defaultwert bzw. setze den Defaultwert der *ZeitspanneEintrittNormal* auf den Wert, der als Parameter folgt.

C++ Typdefinitionen	Beschreibung
CmsgTypeSetHeatingUpTimeDe-fault,	Setze die *ZeitspanneAufheizenNormal* auf den Defaultwert bzw. setze den Defaultwert der *ZeitspanneAufheizenNormal* auf den Wert, der als Parameter folgt.
CmsgTypeSetPresenceTemp,	Setze die *AnwesenheitsTempNormal* auf den Wert, der als Parameter folgt.
CmsgTypeSetAbsenceTemp,	Setze die *AbwesenheitsTempNormal* auf den Wert, der als Parameter folgt.
CmsgTypeSetPresenceTempDe-fault,	Setze die *AnwesenheitsTempNormal* auf den Defaultwert bzw. setze den Defaultwert der *AnwesenheitsTempNormal* auf den Wert, der als Parameter folgt.
CmsgTypeSetAbsenceTempDe-fault,	Setze die *AbwesenheitsTempNormal* auf den Defaultwert bzw. setze den Defaultwert der *AbwesenheitsTempNormal* auf den Wert, der als Parameter folgt.
CmsgTypeSetRoomTemp,	Zeige den Wert des folgenden Parameters als Zimmertempe-ratur an.
CmsgTypeSetOutdoorTemp,	Zeige den Wert des folgenden Parameters als Außentempera-tur an.
CmsgTypeSetPresenceState,	Der folgende Parameter beschreibt den Zustand des Anwe-senheitskontaktes.
CmsgTypeSetRainState,	Der folgende Parameter beschreibt den Zustand des Regen-sensors.
CmsgTypeSetButtomHungState,	Der folgende Parameter beschreibt den Zustand des Kippkon-taktes.
CmsgTypeSetWindowOpenState,	Der folgende Parameter beschreibt den Zustand des Offen-kontaktes.
CmsgTypeTerminateIndicate,	Die Benutzungsschnittstelle erklärt, daß die Verbindung abgebaut werden soll.
CmsgTypeTerminateConfirm}	Das System bestätigt den Abbauwunsch.
msgType	

4.2 Komponenten-Anforderungen

In diesem Abschnitt werden die Anforderungen an die einzelnen Klassen beschrieben. Dabei wird in erster Linie auf das funktionale Ein- und Ausgabeverhalten, die Aufruf- und Verbindungsstruktur sowie die Verwendung der Datentypen geachtet. Zuerst erscheinen die Klassen des Systems, die im Rahmen dieser Arbeit verändert wurden, anschließend werden die neu hinzugekommenen Klassen in alphabetischer Reihenfolge ausgeführt.

4.2.1 Die Klasse *AnwesenheitsController*

Die Klasse *AnwesenheitsController* ist im ursprünglichen System schon vorhanden. Sie wurde wie folgt erweitert:

4.2.1.1 Neue Methodenschnittstellen der Klasse *AnwesenheitsController*

Signatur: durationType *getEntryRemarkTimeDefault* (void)

Beschreibung: Liefert den Wert des Attributes *CZeitspanneEintrittDefault* in Minuten zurück.

Signatur: durationType *getExitRemarkTimeDefault* (void)

Beschreibung: Liefert den Wert des Attributes *CZeitspanneVerlassenDefault* in Minuten zurück.

4.2.2 Die Klasse *Haus*

Die Klasse *Haus* ist im ursprünglichen System schon vorhanden. Sie wurde wie folgt erweitert:

4.2.2.1 Neue Attribute der Klasse *Haus*

Es wird ein Attribut *schedulerPtr* eingefügt. Es speichert einen Zeiger, der auf den Scheduler zeigt.

4.2.2.2 Änderungen am Konstruktor der Klasse *Haus*

Nachdem ein Zimmer angelegt wurde, wird ihm über die Methode *setBuildingPtr* ein Zeiger auf das Haus übermittelt.

4.2.2.3 Änderungen am Destruktor der Klasse *Haus*

Der Destruktor der Klasse *Haus* wird neu eingefügt. Er meldet die callBack-Methode des Hauses beim Scheduler ab, stoppt den Scheduler und ruft die Destruktoren aller angelegten Zimmer und des Portbausteins auf.

4.2.2.4 Neue Methodenschnittstellen der Klasse *Haus*

Signatur: void *setSchedulerPtr* (Scheduler *s)

Beschreibung: Diese Methode speichert einen Zeiger auf den Scheduler im Attribut *schedulerPtr*.

4.2.2.5 Änderungen an der Verbindungsstuktur

Es wurde eine Azzoziation zum Scheduler hinzugefügt, die dazu dient, ihn zu stoppen.

4.2.3 Die Klasse *Portbaustein*

Die Klasse *Portbaustein* ist im ursprünglichen System bereits vorhanden. Sie wurde wie folgt erweitert:

4.2.3.1 Neue Attribute der Klasse *Portbaustein*

Es wird ein Attribut *portHandlerPtr*, in dem ein Zeiger auf den Port-Handler gespeichert wird, hinzugefügt.

4.2.3.2 Änderungen am Konstruktor der Klasse *Portbaustein*

Der Konstruktor wird neu hinzugefügt. Dort wird ein Port-Handler angelegt

4.2.3.3 Änderungen am Destruktor der Klasse *Portbaustein*

Der Destruktor wird neu hinzugefügt. Dort wird der Destruktor des Port-Handlers aufgerufen.

4.2.3.4 Neue Methodenschnittstellen der Klasse *Portbaustein*

Es kommen keine neuen Methoden hinzu, jedoch werden die bestehenden neu implementiert.

Signatur: void *CallBitOutputPort* (int PortNr, int LineNr, int value)

Beschreibung: Diese Methode überprüft, ob sie auf den Port *PortNr* angewandt werden darf. Wenn ja, so ruft sie die Methode *setBitValue* des Port-Handlers mit den Parametern *PortNr* und *value* auf, ansonsten führt sie zum Programmabbruch.

Signatur: void *CallIntOutputPort* (int PortNr, int LineNr, int value)

Beschreibung: Diese Methode überprüft, ob sie auf den Port *PortNr* angewandt werden darf. Wenn ja, so ruft sie die Methode *setIntValue* des Port-Handlers mit den Parametern *PortNr* und *value* auf, ansonsten führt sie zum Programmabbruch.

Signatur: int *CallBitInputPort* (int PortNr, int LineNr)

Beschreibung: Diese Methode überprüft, ob sie auf den Port *PortNr* angewandt werden darf. Wenn ja, so ruft sie die Methode *getBitValue* des Port-Handlers mit dem Parameter *PortNr* auf und liefert den erhaltenen Wert zurück, ansonsten führt sie zum Programmabbruch.

Signatur: int *CallIntInputPort* (int PortNr, int LineNr)

Beschreibung: Diese Methode überprüft, ob sie auf den Port *PortNr* angewandt werden darf. Wenn ja, so ruft sie die Methode *getIntValue* des Port-Handlers mit dem Parameter *PortNr* auf und liefert den erhaltenen Wert zurück, ansonsten führt sie zum Programmabbruch.

4.2.3.5 Änderungen an der Verbindungsstruktur

Es wurde ein Port-Handler aggregiert.

4.2.4 Die Klasse *Zimmer*

Die Klasse *Zimmer* ist im ursprünglichen System bereits vorhanden. Sie wurde wie folgt erweitert:

4.2.4.1 neue Attribute der Klasse *Zimmer*

Es wurde ein Zeiger *MsgHandlerPtr* hinzugefügt. Dieser zeigt auf den zum Zimmer gehörenden Message-Handler.

4.2.4.2 Änderungen am Konstruktor der Klasse *Zimmer*

Im den Konstruktor der Klasse Zimmer wurde der Aufruf des Konstruktors der Klasse *MsgHandlerRoom* eingefügt.

4.2.4.3 Änderungen am Destruktor der Klasse *Zimmer*

Der Destruktor der Klasse *Zimmer* wurde komplett neu hinzugefügt. Er meldet die CallBack-Methode beim Scheduler ab und ruft den Destruktor des aggregierten Message-Handlers auf. Außerdem werden die Objekte *Fenster*, *AnwesenheitsController* und *ZimmertemperaturController* mittels *delete* zerstört.

4.2.4.4 Neue Methodenschnittstellen der Klasse *Zimmer*

Signatur: tempType *getPresenceTempDefault* (void)

Beschreibung: Diese Methode liefert den Defaultwert der *AnwesenheitsTemp* in Grad Celsius zurück.

Signatur: tempType *getAbsenceTempDefault* (void)

Beschreibung: Diese Methode liefert den Defaultwert der *AbwesenheitsTemp* in Grad Celsius zurück.

Signatur: durationType *getHeatingUpTimeDefault* (void)

Beschreibung: Diese Methode liefert den Defaultwert der *ZeitspanneAufheizenNormal* in Minuten zurück.

Signatur: durationType *getEntryRemarkTimeDefault* (void)

Beschreibung: Diese Methode liefert den Defaultwert der *ZeitspanneEintrittNormal* in Minuten zurück.

Signatur: durationType *getExitRemarkTimeDefault* (void)

Beschreibung: Diese Methode liefert den Defaultwert der *ZeitspanneVerlassenNormal* in Minuten zurück.

Signatur: tempType *getRoomTemp* (void)

Beschreibung: Diese Methode liefert die aktuelle Raumtemperatur in Grad Celsius zurück.

Signatur: tempType *getOutdoorTemp* (void)

Beschreibung: Diese Methode liefert die aktuelle Außentemperatur in Grad Celsius zurück.

Signatur: int *getPresenceState* (void)

Beschreibung: Diese Methode liefert den aktuellen Zustand des Anwesenheitskontaktes zurück.

Signatur: int *getRainState* (void)

Beschreibung: Diese Methode liefert den aktuellen Zustand des Regensensors zurück.

Signatur: int *getWindowOpenState* (void)

Beschreibung: Diese Methode liefert den aktuellen Zustand des Offenkontaktes zurück.

Signatur: int *getButtomHungState* (void)

Beschreibung: Diese Methode liefert den aktuellen Zustand des Kippkontaktes zurück.

Signatur: void *setBuildingPtr* (Haus* h)

Beschreibung: Diese Methode ruft die Methode *setBuildingPtr* des Message-Handlers auf und übermittelt ihm so den Zeiger auf das Haus.

4.2.4.5 Änderungen an der Verbindungsstruktur

Zusätzlich zur bestehenden Verbindungsstruktur wurde eine Aggregation zu einemObjekt der Klasse *MsgHandlerRoom* eingefügt.

4.2.5 Die Klasse *HelpPanel*

Die Klasse *HelpPanel* wurde von ILOG Views generiert. Von ihr wird die Klasse *WasteHelpPanel* abgeleitet. Sie wird nicht verändert.

4.2.6 Die Klasse *MainPanel*

Die Klasse *MainPanel* wurde von ILOG Views generiert. Von ihr wird die Klasse *WasteMainPanel* abgeleitet. Sie wird nicht verändert.

4.2.7 Die Klasse *Mapper*

Dir Klasse *Mapper* die Funktionalität des Mappers, d.h. sie speichert die Datenstruktur, die die Umsetzung der Ports aus WASTE in die Aktoren und Sensoren des Testraumes/Simulators.

4.2.7.1 Attribute der Klasse *Mapper*

Die Klasse *Mapper* hat folgende Attribute:

Attribut	Erläuterung
mapperMatrix	Zeiger auf die zweidimensionale Matrix, in der die Ports den Aktoren und Sensoren zugeordnet werden.
portHandlerPtr	Zeiger auf den Port-Handler.
actHandledPort	Nummer des aktuell in der CallBack-Methode behandelter Ports.
actHandledActSens	Nummer des aktuell in der CallBack-Methode behandelten Aktors bzw. Sensors des Ports *actHandledPort*.
roomTemp	Die Raumtemperatur in Grad Celsius.
rainSensorHandlingInterval	Das Intervall, in dem der Regensensor abgefragt wird.
lastRainAmount	Der letzte erkannte Wert des Regensensors.
lastHandlingTime	Der Zeitpunkt, an dem zuletzt ein Aktor bzw. Sensor (außer dem Regensensor) behandelt wurde.
lastRainSensorHandlingTime	Der Zeitpunkt, an dem der Regensensor zuletzt abgefragt wurde.

4.2.7.2 Konstruktor der Klasse *Mapper*

Im Konstruktor des Mappers wird die Mappermatrix angelegt und initialisiert. Außerdem werden alle binäre Sensoren mit Hilfe des Message-Handlers in den Event-Mode geschaltet.

4.2.7.3 Destruktor der Klasse *Mapper*

Im Destruktor des Mappers werden alle binäre Sensoren mit Hilfe des Message-Handlers in den normalen Modus geschaltet und die Mappermatrix zerstört.

4.2.7.4 Methodenschnittstellen der Klasse *Mapper*

Signatur: void *setBitValue* (int port, int value)

Beschreibung: Diese Methode schreibt den Wert *value* in alle Value-Felder, der auf den Port *port* gemappten Aktoren. Außerdem wird in das value-Feld der letzten Zeile dieses Ports folgender Wert eingetragen: Anzahl der auf diesen Port gemappten Aktoren plus eins. Bei Port 6 (dem *Kippmotor*) wird der Wert *value* wie folgt umgerechnet: *value* = 1 - *value*. Dies ist nötig, da WASTE und der Testraum/Simulator die Werte 0 und 1 gegensätzlich interpretieren.

Signatur: void *setIntValue* (int port, int value)

Beschreibung: Diese Methode schreibt den Wert *value* in alle value-Felder, der auf den Port *port* gemappten Aktoren. Außerdem wird in das value-Feld der letzten Zeile dieses Ports folgender Wert eingetragen: Anzahl der auf diesen Port gemappten Aktoren plus eins.

Signatur: int *getBitValue* (int port)

Beschreibung: Diese Methode ließt die Anzahl der auf den Port *port* gemappten Sensoren. Ist diese Anzahl null, so wird der Defaultwert dieses Sensors aus dem value-Feld der ersten Zeile dieses Ports ausgelesen und zurückgegeben. Ansonsten werden die Wert der value-Felder der auf diesen Port gemappten Ports ausgelesen, durch ein logisches „Und" (da die Werte Integerzahlen sind, wird dies durch eine Multiplikation realisiert) verknüpft und der erhaltene Wert zurückgeliefert.

Signatur: int *getIntValue* (int port)

Beschreibung: Diese Methode ließt die Anzahl der auf den Port *port* gemappten Sensoren. Ist diese Anzahl null, so wird der Defaultwert dieses Sensors aus dem value-Feld der ersten Zeile dieses Ports ausgelesen und zurückgegeben. Ansonsten wird das arithmetischen Mittel der Werte der value-Felder der Sensoren, die auf diesen Port gemappt sind, gebildet und zurückgeliefert.

Signatur: void *updateValue* (char* sensor, float value)

Beschreibung: Diese Methode rundet den Wert *value* auf eine ganze Zahle (in WASTE werden alle Werte durch Integerzahlen dargestellt) und speichert diesen gerundeten Wert in der Variablen *intValue*. Anschließend durchsucht sie die Name-Felder der ganzen Matrix und vergleicht die Werte der Name-Felder mit dem String *sensor*. Wenn sie den entsprechenden Eintrag gefunden hat, wird überprüft, ob der dazugehörige Port der Port 4 (der Kippkontakt) ist. Ist dies der Fall, so wird der gerundete Wert wie in der Methode *setBitValue* umgerechnet. War der entsprechende Port der Port 10 (der Regensensor), so wird überprüft, ob der Wert *value* größer ist als der in dem Attribut *lastRainAmount* gespeicherte Wert. Ist dies der Fall, wird die Variable *intValue* gleich 1 ansonsten gleich 0 gesetzt. In beiden Fällen wird der Wert *value* im Attribut *lastRainAmount* gespeichert. Schließlich wird der Wert, der in der Variablen *intValue* steht, in das value-Feld des entsprechenden Sensors geschrieben.

Signatur: void *callBack* (void)

Beschreibung: Diese Methode überprüft zunächst, ob seit der letzten Behandlung eines Aktors mehr als zwei Sekunden vergangen sind. Ist dies der Fall, so wird überprüft, ob der Wert des Attributes *actHandledActSens* größer ist als die Anzahl der auf den Port *actHandledPort* gemappten Aktoren/Sensoren. Ist dies der Fall, so wird der Wert *actHandledPort* um eins modulo der Anzahl der Ports erhöht und der Wert in *actHandledActSens* auf eins gesetzt. Ansonsten wird abhängig vom Status dieses Aktors/Sensors (gespeichert im value-Feld der letzten Zeile) die Behandlungsroutine gewählt: Trägt der Status den Wert -1, d.h. der zu behandelnde Sensor ist ein normaler passiver Sensor, so wird dem Testraum/Simulator über den Message-Handler der Befehl erteilt, den aktuellen Wert dieses Sensors an WASTE zu melden. Trägt der Status den Wert 0, d.h. der zu behandelnde Aktor muß unabhängig davon, ob der entsprechende Port von WASTE angesprochen wurde, behandelt werden, so wird nochmals unterschieden, ob es sich um Port 3 (die *Heizungsventile*) handelt oder nicht. Ist dies nicht der Fall (er tritt in der jetzigen Version von WASTE nicht ein, wird hier auf Gründen der einfacheren Erweiterbarkeit mit betrachtet), so wird dem Testraum/Simulator über den Message-Handler mitgeteilt, daß dieser den Aktor auf den im value-Feld gespeicherten Wert setzen soll. Bei Port 3 wird überprüft, ob die gewünschte Heizkörpertemperatur (gespeichert im value-Feld des betrachteten Aktors) über oder unter der aktuellen Zimmertemperatur (gespeichert im Attribut *roomTemp*) liegt. Ist sie größer, so wird dem Testraum/Simulator über den Message-Handler mitgeteilt, daß er dieses Heizungsventil öffnen soll. Ansonsten wird ihm mitgeteilt, daß er es schließen soll. Trägt der Status einen Wert größer 1, d.h. der zu behandelte Aktor ist ein normaler Aktor und wurde seitdem er das letzte Mal betrachtet wurde durch das Kotrollsystem angesprochen, so wird dem Testraum/Simulator mitgeteilt, daß er diesen Aktor auf dem im value-Feld gespeicherten Wert setzen soll. Der Wert des Status wird außerdem um 1 verringert. Ist die zur Status gehörige Behandlung abgeschlossen, so wird der Wert in *actHandledActSens* um eins erhöht und der aktuelle Zeitpunkt im Attribut *lastHandlingTime* gespeichert. Zusätzlich zu dieser Behandlung der Aktoren/Sensoren im zwei-Sekunden-Takt, wird in dieser Methode auch der *Regensensor* (Port 10) mit seinem eigenen Takt behandelt. Ist sein Intervall seit seiner letzten Behandlung vergangen, so wird dem Testraum/Simulator über den Message-Handler mitgeteilt, daß er den aktuellen Wert des Regensensors an WASTE übermitteln soll. Außerdem dem wird der aktuelle Zeitpunkt der Behandlung im Attribut *lastRainSensorHandlingTime* gespeichert.

4.2.7.5 Verbindungsstruktur

Die Klasse *Mapper* ist zum Port-Handler aggregiert.

4.2.7.6 Die Klasse *MonitorUI*

Die Klasse *MonitorUI* übernimmt die Funktionalität des Monitors der Benutzungsschnittstelle. Sie bekommt Benutzereingaben mitgeteilt und entscheidet, wann eine Nachricht an das Zimmer gesendet werden muß.

4.2.7.7 Attribute der Klasse *MonitorUI*

Die Klasse *MonitorUI* hat folgende Attribute:

Attribut	Erläuterung
msgHandlerPtr	Zeiger auf den Message-Handler, zu dem der Monitor gehört.
presenceTemp	Letzter eingegebene Wert der *AnwesenheitsTempNormal*.
presenceTempHasChanged	Flag, das anzeigt, daß die *AnwesenheitsTempNormal* verändert wurde und die neue *AnwesenheitsTempNormal* noch nicht an WASTE übermittelt wurde.
presenceTempChangeTime	Zeitpunkt der letzten Änderung der *AnwesenheitsTempNormal*.
absenceTemp	Letzter eingegebene Wert der *AbwesenheitsTempNormal*.
absenceTempHasChanged	Flag, das anzeigt, daß die *AbwesenheitsTempNormal* verändert wurde und die neue *AbwesenheitsTempNormal* noch nicht an WASTE übermittelt wurde.
absenceTempChangeTime	Zeitpunkt der letzten Änderung der *AbwesenheitsTempNormal*.
heatingUpTime	Letzter eingegebene Wert der *ZeitspanneAufheizenNormal*.
heatingUpTimeHasChanged	Flag, das anzeigt, daß die *ZeitspanneAufheizenNormal* verändert wurde und die neue *ZeitspanneAufheizenNormal* noch nicht an WASTE übermittelt wurde.
heatingUpTimeChangeTime	Zeitpunkt der letzten Änderung der *ZeitspanneAnfheizenNormal*.
entryRemarkTime	Letzter eingegebene Wert der *ZeitspanneEintrittNormal*.
entryRemarkTimeHasChanged	Flag, das anzeigt, daß die *ZeitspanneEintrittNormal* verändert wurde und die neue *ZeitspanneEintrittNormal* noch nicht an WASTE übermittelt wurde.
entryRemarkTimeChangeTime	Zeitpunkt der letzten Änderung der *ZeitspanneEintrittNormal*.
exitRemarkTime	Letzter eingegebene Wert der *ZeitspanneVerlassenNormal*.
exitRemarkTimeHasChanged	Flag, das anzeigt, daß die *ZeitspanneVerlassenNormal* verändert wurde und die neue *ZeitspanneVerlassenNormal* noch nicht an WASTE übermittelt wurde.
exitRemarkTimeChangeTime	Zeitpunkt der letzten Änderung der *ZeitspanneVerlassenNormal*.
monitoringNeeded	Flag, das anzeigt, ob der Monitor aktiv ist oder nicht.

4.2.7.8 Konstruktor der Klasse *MonitorUI*

Im Konstruktor der Klasse *MonitorUI* wird nur die CallBack-Methode dem Scheduler bekannt gemacht.

4.2.7.9 Destruktor der Klasse *MonitorUI*

Im Destruktor der Klasse *MonitorUI* wird die CallBack-Methode beim Scheduler abgemeldet.

4.2.7.10 Methodenschnittstellen der Klasse *MonitorUI*

Signatur: void *callBack* (void)

Beschreibung: Wenn das Flag *monitoringNeeded* gesetzt ist, überprüft diese Methode zyklisch alle eingestellten Werte. Wurde ein Wert „xxx“ mit Hilfe eines Schiebereglers verändert (d.h. das Flag xxxHasChanged ist gesetzt) und ist seid der Veränderung mehr als eine Sekunde vergangen, so wird der neue Wert über den Message-Handler an WASTE übermittelt. Dabei wird das Flag xxxHasChanged zurück gesetzt.

Signatur: void *setPresenceTemp* (int temp)

Beschreibung: Diese Methode wird vom MainPanel benutzt, um dem Monitor mitzuteilen, daß der Wert der *AnwesenheitsTempNormal* auf den Wert *temp* gesetzt wurde. Sie speichert den neuen Wert im Attribut *presenceTemp*, den Zeitpunkt der Eingabe in *presenceTempChangeTime* und setzt das Flag *presenceTempHasChanged*.

Signatur: void *setAbsenceTemp* (int temp)

Beschreibung: Diese Methode wird vom MainPanel benutzt, um dem Monitor mitzuteilen, daß der Wert der *AbwesenheitsTempNormal* auf den Wert *temp* gesetzt wurde. Sie speichert den neuen Wert im Attribut *absenceTemp*, den Zeitpunkt der Eingabe in *absenceTempChangeTime* und setzt das Flag *absenceTempHasChanged*.

Signatur: void *setHeatingUpTime* (int time)

Beschreibung: Diese Methode wird vom MainPanel benutzt, um dem Monitor mitzuteilen, daß der Wert der *ZeitspanneAufheizenNormal* auf den Wert *time* gesetzt wurde. Sie speichert den neuen Wert im Attribut *heatingUpTime*, den Zeitpunkt der Eingabe in *heatingUpTimeChangeTime* und setzt das Flag *heatingUpTimeHasChanged*.

Signatur: void *EntryRemarkTime* (int time)

Beschreibung: Diese Methode wird vom MainPanel benutzt, um dem Monitor mitzuteilen, daß der Wert der *ZeitspanneEintrittNormal* auf den Wert *time* gesetzt wurde. Sie speichert den neuen Wert im Attribut *entryRemarkTime*, den Zeitpunkt der Eingabe in *entryRemarkTimeChangeTime* und setzt das Flag *entryRemarkTimeHasChanged*.

Signatur: **void *ExitRemarkTime* (int time)**

Beschreibung: Diese Methode wird vom MainPanel benutzt, um dem Monitor mitzuteilen, daß der Wert der *ZeitspanneVerlassenNormal* auf den Wert *time* gesetzt wurde. Sie speichert den neuen Wert im Attribut *exitRemarkTime*, den Zeitpunkt der Eingabe in *exitRemarkTimeChangeTime* und setzt das Flag *exitRemarkTimeHasChanged*.

Signatur: **void *setPresenceTempDefault* (void)**

Beschreibung: Diese Methode wird vom MainPanel benutzt, um dem Monitor mitzuteilen, daß der Default-Button der *AnwesenheitsTemp* gedrückt wurde. Wenn das Flag *monitoringNeeded* gesetzt ist, so übermittelt sie dies über den Message-Handler an WASTE und setzt das Flag *presenceTempHasChanged* zurück.

Signatur: **void *AbsenceTempDefault* (void)**

Beschreibung: Diese Methode wird vom MainPanel benutzt, um dem Monitor mitzuteilen, daß der Default-Button der *AbwesenheitsTemp* gedrückt wurde. Wenn das Flag *monitoringNeeded* gesetzt ist, so übermittelt sie dies über den Message-Handler an WASTE und setzt das Flag *absenceTempHasChanged* zurück.

Signatur: **void *setHeatingUpTimeDefault* (void)**

Beschreibung: Diese Methode wird vom MainPanel benutzt, um dem Monitor mitzuteilen, daß der Default-Button der *ZeitspanneAufheizenNormal* gedrückt wurde. Wenn das Flag *monitoringNeeded* gesetzt ist, so übermittelt sie dies über den Message-Handler an WASTE und setzt das Flag *heatingUpTimeHasChanged* zurück.

Signatur: **void *setEntryRemarkTimeDefault* (void)**

Beschreibung: Diese Methode wird vom MainPanel benutzt, um dem Monitor mitzuteilen, daß der Default-Button der *ZeitspanneEintrittNormal* gedrückt wurde. Wenn das Flag *monitoringNeeded* gesetzt ist, so übermittelt sie dies über den Message-Handler an WASTE und setzt das Flag *entryRemarkTimeHasChanged* zurück.

Signatur: **void *setExitRemarkTimeDefault* (void)**

Beschreibung: Diese Methode wird vom MainPanel benutzt, um dem Monitor mitzuteilen, daß der Default-Button der *ZeitspanneVerlasssenNormal* gedrückt wurde. Wenn das Flag *monitoringNeeded* gesetzt ist, so übermittelt sie dies über den Message-Handler an WASTE und setzt das Flag *exitRemarkTimeHasChanged* zurück.

Signatur: **void *startTransmission* (void)**

Beschreibung: Diese Methode teilt WASTE durch Übermittlung des Nachrichtentyps *CmsgTypeStartIndicate* mit, daß der Start-Button gedrückt und damit die Benutzungsschnitstelle aktiviert wurde. Außerdem aktiviert sie den Monitor durch Setzten des Flags *monitoringNeeded*.

Signatur: **void *terminateConnection* (void)**

Beschreibung: Diese Methode initiiert den Verbindungsabbau. Sie veranlaßt das Senden des Nachrichtentyps *CmsgTypeTerminateIndicate* und setzt das Flag *monitoringNeeded* zurück. Von nun an stellt der Monitor keine Sendeanforderungen mehr an den Message-Handler.

4.2.7.11 Verbindungsstruktur der Klasse *MonitorUI*

Die Klasse *MonitorUI* hat eine Verbindung zu dem Message-Handler, zu dem sie gehört.

4.2.8 Die Klasse *MonitorRoom*

Die Klasse *MonitorRoom* übernimmt die Funktionalität des Monitors des Zimmers. Sie überwacht die Werte aller anzeigbaren Sensoren und „meldet“ jede Änderung über den Message-Handler an die Benutzungsschnittstelle.

4.2.8.1 Attribute der Klasse *MonitorRoom*

Die Klasse *MonitorRoom* hat folgende Attribute:

Attribut	Erläuterung
msgHandlerPtr	Zeiger auf den Message-Handler zu dem der Monitor gehört.
roomPtr	Zeiger auf das Zimmer zu dem der Message-Handler und damit auch der Monitor gehört.
monitoringNeeded	Flag, das anzeigt, ob der Monitor aktiv ist.
roomTemp	Letzter beobachteter Wert der Raumtemperatur.
outdoorTemp	Letzter beobachteter Wert der Außentemperatur.
rainState	Letzter beobachteter Zustand des *Regensensors*.
presenceState	Letzter beobachteter Zustand des *Anwesenheitskontaktes*.
windowOpenState	Letzter beobachteter Zustand des *Offenkontaktes*.
buttomHungState	Letzter beobachteter Zustand des *Kippkontaktes*.

4.2.8.2 Konstruktor der Klasse *MonitorRoom*

Im Konstruktor der Klasse *MonitorRoom* wird die CallBack-Methode dem Scheduler bekannt gemacht.

4.2.8.3 Destruktor der Klasse *MonitorRoom*

Im Destruktor der Klasse *MonitorRoom* wird die CallBack-Methode beim Scheduler abgemeldet.

4.2.8.4 Methodenschnittstellen der Klasse *MonitorRoom*

Signatur: void *setRoomPtr* (Zimmer *r)

Beschreibung: Diese Methode setzt den Zeiger *roomPtr* auf das Zimmer.

Signatur: void *startMonitoring* (void)

Beschreibung: Diese Methode setzt das Flag *monitoringNeeded*, und veranlaßt daß Senden aller nötigen Defaultwerte sowie der aktuellen Werte aller anzeigbaren Sensoren.

Signatur: void *stopMonitoring* (void)

Beschreibung: Diese Methode setzt das Flag *monitoringNeeded* zurück, in Zukunft werden keine Sendeanforderungen mehr gestellt.

Signatur: void *callBack* (void)

Beschreibung: Diese Methode wird zyklisch vom Scheduler augerufen. Sie realisiert das Monitoring der überwachten Werte und richtet gegebenenfalls Sendeanforderungen an den Message-Handler. Um dies zu entscheiden vergleicht sie die aktuellen Werte der Sensoren mit denen des letzten Zyklus.

4.2.8.5 Verbindungsstruktur der Klasse *MonitorRoom*

Die Klasse *MonitorRoom* hat Verbindungen zu dem Message-Handler, zu dem sie gehört, und zu dem Zimmer, zu dem der Message-Handler gehört.

4.2.9 Die Klasse *MsgHandlerUI*

Die Klasse *MsgHandlerUI* übernimmt die Funktionalität des Message-Handlers der Benutzungsschnittstelle. Sie ist stark angelehnt an die entsprechende Klasse aus [Cod97]. Sie überwacht ihren SocketStream und ruft in Abhängigkeit der empfangenen Nachrichten Methoden des MainPanels auf. Außerdem stellt sie Methoden zum Senden von Nachrichten bereit.

4.2.9.1 Attribute der Klasse *MsgHandlerUI*

Die Klasse *MsgHandlerUI* hat folgende Attribute:

Attribut	Erläuterung
sockStream	Zeiger auf den SocketStream, über welchen die Kommunikation mit WASTE läuft.
monitorPtr	Zeiger auf den aggregierten Monitor.
mainPanelPtr	Zeiger auf das MainPanel.

4.2.9.2 Konstruktor der Klasse *MsgHandlerUI*

Im Konstruktor der Klasse *MsgHandlerUI* wird die Verbindung zu dem Message-Handler des Kontrollsystems mittels eines *SocketStreams* etabliert. Dabei fungiert dieser Message-Handler als Client und der des Kontrollsystems als Server (das Verfahren des Verbindungsaufbaus wurde aus [Cod97] übernommen). Außerdem wird ein Monitor angelegt. Schließlich wird die CallBack-Methode beim Scheduler angemeldet.

4.2.9.3 Destruktor der Klasse *MsgHandlerUI*

Im Destruktor der Klasse *MsgHandlerUI* wird die CallBack-Methode beim Scheduler abgemeldet, der *SocketStream* wird gelöscht und der Destruktor des aggregierten Monitors aufgerufen.

4.2.9.4 Methodenschnittstellen der Klasse *MsgHandlerUI*

Signatur: void *sendMsg* (msgType type, int parameter)

Beschreibung: Diese Methode sendet eine Nachricht des Types *type* mit dem Parameter *parameter* an WASTE.

Signatur: void *sendMsg* (msgType type)

Beschreibung: Diese Methode sendet eine Nachricht des Types *type* ohne Parameter an WASTE.

Signatur: void *callBack* (void)

Beschreibung: Diese Methode überwacht den *SocketStream*, empfängt und verarbeitet eingehende Nachrichten. Dabei werden die entsprechenden Methoden des MainPanels aufgerufen. Ihr grundsätzlicher Aufbau wurde aus [Cod97] übernommen.

Signatur: void *setMainPanelPtr* (WasteMainPanel* ptr)

Beschreibung: Diese Methode speichert einen Zeiger auf das MainPanel im Attribut *MainPanelPtr*.

4.2.9.5 Verbindungsstruktur der Klasse *MsgHandlerUI*

Die Klasse *MsgHandlerUI* hat Verbindungen zu dem aggregierten Monitor und zum MainPanel.

4.2.10 Die Klasse *MsgHandlerPB*

4.2.10.1 Attribute der Klasse *MsgHandlerPB*

Die Klasse *MsgHandlerBP* hat folgende Attribute:

Attribut	Erläuterung
sockStream	Zeiger auf den SocketStream, über welchen die Kommunikation mit dem Testraum/Simulator läuft.
portHandlerPtr	Zeiger auf den Port-Handler, zu dem der Message-Handler gehört.

4.2.10.2 Konstruktor der Klasse *MsgHandlerPB*

Im Konstruktor wird die Verbindung zum Testraum/Simulator initialisiert (der Kode dazu wird aus dem Kontraktur der Klasse *MsgHandlerUI* kopiert). Der Parameter des Konstruktors ist ein Zeiger auf den Port-Handler.

4.2.10.3 Destruktor der Klasse *MsgHandlerPB*

Im Destruktor wird der Socket drei Sekunden lang ausgelesen. Anschließend wird der *SocketStream* zerstört.

4.2.10.4 Methodenschnittstellen der Klasse *MsgHandlerPB*

Signatur: void *sendRequest* (char* sensor)

Beschreibung: Diese Methode sendet dem Testraum/Scheduler den Befehl, den Wert des Sensors *sensor* an diesem Message-Handler zu übermitteln.

Signatur: void *sendOrder* (char* actor, int value)

Beschreibung: Diese Methode sendet dem Testraum/Scheduler den Befehl, den Aktor *actor* auf den Wert *value* einzustellen.

Signatur: void *makeActive* (char* sensor)

Beschreibung: Diese Methode sendet dem Testraum/Scheduler den Befehl, den Sensors *sensor* in den Event-Mode zu schalten.

Signatur: void *makePassive* (char* sensor)

Beschreibung: Diese Methode sendet dem Testraum/Scheduler den Befehl, den Sensors *sensor* in den Normal-Mode zu schalten.

Signatur: void callBack (void)

Beschreibung: Diese Methode überwacht den SocketStream. Wird eine Nachricht empfangen, wird anhand des ersten empfangenen Zeichens entschieden um welchen Typ Nachricht es sich handelt. Bestätigungen beginnen mit ‚A‘, Fehlermeldungen mit ‚E‘ - alles weitere (in der zur Zeit gültigen Spezifikation des Schnittstelle des Testraumes) müssen Antworten sein, die Sensorenwerte enthalten. Nun liest sie die Nachricht vollständig ein (d.h. bis zum ‚^A‘). Da die Spezifikation der Schnittstelle des Testraumes in Zukunft erweitert werden könnte (insb. um Nachrichtentypen die auch mit ’A’ bzw. ’E’ beginnen, aber keine Bestätigungen bzw. Fehlermeldungen sind, wird die komplett empfangene Nachricht zusätzlich überprüft, ob sie wirklich eine Bestätigung bzw. Fehlermeldung war. Bestätigungen werden ignoriert, Fehlermeldungen und unbekannte Nachrichtentypen führen dagegen zum Abbruch. War die eingegangene Nachricht die Meldung eines Sensorenwertes, wird aus ihr den Namen des Sensors und den Wert extrahiert und beides mit der Methode *updateValue* an den Port-Handler gemeldet.

4.2.10.5 Verbindungsstruktur

Die Klasse *MsgHandlerPB* ist zum Port-Handler aggregiert.

4.2.11 Die Klasse *MsgHandlerRoom*

Die Klasse *MsgHandlerRoom* übernimmt die Funktionalität des Message-Handlers des Zimmers. Sie ist stark angelehnt an die entsprechende Klasse aus [Cod97]. Sie überwacht ihren *SocketStream* und ruft in Abhängigkeit der empfangenen Nachrichten Methoden ihres Zimmers bzw. des Hauses auf. Außerdem stellt sie Methoden zum Senden von Nachrichten bereit.

4.2.11.1 Attribute der Klasse *MsgHandlerRoom*

Die Klasse *MsgHandlerRoom* hat folgende Attribute:

Attribut	Erläuterung
roomPtr	Zeiger auf das Zimmer, zu dem der Message-Handler gehört.
buildingPtr	Zeiger auf das Haus.
monitorPtr	Zeiger auf den aggregierten Monitor.
sockStream	Zeiger auf den *SocketStream* über welchen die Kommunikation mit der Benutztungsschnittstelle läuft.
uiRunning	Flag, das anzeigt, ob die Benutzungsschnittstelle schon aktiviert wurde.

4.2.11.2 Konstruktor der Klasse *MsgHandlerRoom*

Im Konstruktor der Klasse *MsgHandlerRoom* wird die Verbindung über zum Message-Handler der Benutzungsschnitstelle über einen *SocketStream* etabliert. Dabei fungiert dieser Message-Handler als Server und der der Benutzungsschnitstelle als Client (das Verfahren des Verbindungsaufbaus wurde aus Cod97 übernommen). Außerdem wird ein Monitor angelegt. Schließlich wird die CallBack-Methode beim Scheduler angemeldet.

4.2.11.3 Destruktor der Klasse *MsgHandlerRoom*

Im Destruktor der Klasse *MsgHandlerRoom* wird die CallBack-Methode beim Scheduler abgemeldet. Der *SocketStream*, der die Verbindung zur Benutzungsschnittstelle realisiert wirt gelöscht und der Destruktor des aggregierten Monitors aufgerufen.

4.2.11.4 Methodenschnittstellen der Klasse *MsgHandlerRoom*

Signatur: void *sendMsg* (msgType type, int parameter)

Beschreibung: Diese Methode sendet eine Nachricht des Types *type* mit dem Parameter *parameter* an die Benutzungsschnittstelle.

Signatur: void *sendMsg* (msgType type)

Beschreibung: Diese Methode sendet eine Nachricht des Types *type* ohne Parameter an die Benutzungsschnittstelle.

Signatur: void callBack (void)

Beschreibung: Diese Methode überwacht den *SocketStream*, empfängt und verarbeitet eingehende Nachrichten. Dabei werden die entsprechenden Methoden des Zimmers aufgerufen. Ihr grundsätzlicher Aufbau wurde aus [Cod97] übernommen.

Signatur: void *setRoomPtr* (Zimmer* r)

Beschreibung: Diese Methode speichert einen Zeiger auf das Zimmer in *roomPtr*.

Signatur: void *setBuildingPtr* (Haus *h)

Beschreibung: Diese Methode speichert einen Zeiger auf das Haus in *buildingPtr*.

4.2.11.5 Verbindungsstruktur der Klasse *MsgHandlerRoom*

Die Klasse *MsgHandlerRoom* hat Verbindungen zum aggregierten Monitor, zu dem Zimmer, zu dem sie selbst gehört und zum Haus.

4.2.12 Die Klasse *MsgHeader*

Die Klasse *MsgHeader* stammt größtenteils aus [Cod97]. Sie wurde wie folgt modifiziert: die Operatoren << und >> wurden entfernt (siehe 4.1.1).

4.2.13 Die Klasse *PortHandler*

4.2.13.1 Attribute der Klasse *PortHandler*

Die Klasse *PortHandler* hat folgende Attribute:

Attribut	Erläuterung
mapperPtr	Zeiger, auf den Mapper, der die Umsetzung der Ports in Aktoren und Sensoren kapselt.
msgHandlerPtr	Zeiger, auf den Message-Handler, der der Verbindung zum Testraum/Simulator kapselt.

4.2.13.2 Konstruktor der Klasse *PortHandler*

Im Konstruktor des Port-Handlers werden der Mapper und der Message-Handler angelegt.

4.2.13.3 Destruktor der Klasse *PortHandler*

Im Destruktor des Port-Handlers werden die Destruktoren des Mappers und des Message-Handlers aufgerufen. Dabei muß der Destruktor des Mappers zuerst aufgerufen werden, denn dieser greift noch auf Methoden des Mappers zurück.

4.2.13.4 Methodenschnittstellen der Klasse *PortHandler*

Signatur: void *setBitValue* (int port, int value)

Beschreibung: Diese Methode ist eine Durchreichmethode. Sie ruft die Methode *setBitValue* (mit den Parametern *port* und *value*) des Mappers auf.

Signatur: void *setIntValue* (int port, int value)

Beschreibung: Diese Methode ist eine Durchreichmethode. Sie ruft die Methode *setIntValue* (mit den Parametern *port* und *value*) des Mappers auf.

Signatur: int *getBitValue* (int port)

Beschreibung: Diese Methode ist eine Durchreichmethode. Sie ruft die Methode *getBitValue* (mit dem Parameter *port*) des Mappers auf und liefert den erhaltenen Wert zurück.

Signatur: **int *getIntValue* (int port)**

Beschreibung: Diese Methode ist eine Durchreichmethode. Sie ruft die Methode *getIntValue* (mit dem Parameter *port*) des Mappers auf und liefert den erhaltenen Wert zurück.

Signatur: **void *updateValue* (char* sensor, float value)**

Beschreibung: Diese Methode ist eine Durchreichmethode. Sie ruft die Methode *updateValue* (mit den Parametern *sensor* und *value*) des Mappers auf.

Signatur: **void *sendRequest* (char* sensor)**

Beschreibung: Diese Methode ist eine Durchreichmethode. Sie ruft die Methode *sendRequest* (mit dem Parameter *sensor*) des Message-Handlers auf.

Signatur: **void *sendOrder* (char* actor, int value)**

Beschreibung: Diese Methode ist eine Durchreichmethode. Sie ruft die Methode *sendOrder* (mit den Parametern *actor* und *value*) des Message-Handlers auf.

Signatur: **void *makeActive* (char* sensor)**

Beschreibung: Diese Methode ist eine Durchreichmethode. Sie ruft die Methode *makeActive* (mit dem Parameter *sensor*) des Message-Handlers auf.

Signatur: **void *makePassive* (char* sensor)**

Beschreibung: Diese Methode ist eine Durchreichmethode. Sie ruft die Methode *makePassive* (mit dem Parameter *sensor*) des Message-Handlers auf.

4.2.13.5 Verbindungsstruktur

Zum Port-Handler sind der Mapper und der Message-Handler aggregiert. Der Port-Handler selbst ist wiederum zum Portbaustein aggregiert.

4.2.14 Die Klasse *QuitPanel*

Die Klasse *QuitPanel* wurde von ILOG Views generiert. Von ihr wird die Klasse *WasteQuitPanel* abgeleitet. Sie wird nicht verändert.

4.2.15 Die Klasse *SocketStream*

Die Klasse *SocketStream* stammt aus [Cod97]. Sie wurde wie folgt modifiziert: es wurden Operatoren << und >> für Objekte der Typen *MsgHeader* und *MsgType* hinzugefügt (siehe 4.1.1).

4.2.16 Die Klasse *WasteHelpPanel*

Die Klasse *WasteHelpPanel* wurde von der Klasse *HelpPanel* abgeleitet. Sie realisiert die Funktionalität des HelpPanels.

4.2.16.1 Attribute der Klasse *WasteHelpPanel*

Es wurden folgende neue Attribute hinzugefügt.:

Attribut	Erläuterung
mainPanelPtr	Zeiger auf das MainPanel.

4.2.16.2 Methodenschnittstellen der Klasse *WasteHelpPanel*

Signatur: **void *HelpCloseButtonCB* (IlvGraphic *g)**

Beschreibung: Diese Methode ruft die Methode *clearHelpPanel* des MainPanels auf, welche das helpPanel schließt.

Signatur: **void *setMainPanelPtr* (WasteMainPanel* ptr)**

Beschreibung: Diese Methode speichern einen Zeiger auf das MainPanel im Attribut *mainPanelPtr*.

4.2.16.3 Verbindungsstruktur

Das *WasteHelpPanel* hat eine Azzoziation zum MainPanel.

4.2.17 Die Klasse *WasteMainPanel*

Die Klasse *WasteMainPanel* wurde von der Klasse *MainPanel* abgeleitet. Sie realisiert die Funktionalität des MainPanels.

4.2.17.1 Attribute der Klasse *WasteMainPanel*

Es wurden folgende neue Attribute hinzugefügt.:

Attribut	Erläuterung
systemActivated	Flag, das anzeigt, ob der Start-Button bereits gedrückt wurde ober nicht.
visibleHelpPanel	Flag, das anzeigt, ob zur Zeit ein HelpPanel geöffnet ist oder nicht.
visibleQuitPanel	Flag, das anzeigt, ob zur Zeit ein QuitPanel geöffnet ist oder nicht.
showSensors	Flag, das anzeigt, ob die optionalen Sensoren zur Zeit angezeigt werden oder nicht.

Attribut	Erläuterung
monitorPtr	Zeiger auf den Monitor.
msgHandlerPtr	Zeiger auf den Message-Handler.
mainApplicationPtr	Zeiger auf die Application.
schedulerPtr	Zeiger auf den Scheduler.
helpPanelPtr	Zeiger auf das HelpPanel, falls es geöffnet ist.
quitPanelPtr	Zeiger auf das QuitPanel, falls es geöffnet ist.
presenceTempDefault	Hier wird der Defaultwert der *AnwesenheitsTemp* für die Dauer der Sitzung gespeichert.
absenceTempDefault	Hier wird der Defaultwert der *AbwesenheitsTemp* für die Dauer der Sitzung gespeichert.
heatingUpTimeDefault	Hier wird der Defaultwert der *ZeitspanneAufheizenNormal* für die Dauer der Sitzung gespeichert.
entryRemarkTimeDefault	Hier wird der Defaultwert der *ZeitspanneEintrittNormal* für die Dauer der Sitzung gespeichert.
exitRemarkTimeDefault	Hier wird der Defaultwert der *ZeitspanneVerlassenNormal* für die Dauer der Sitzung gespeichert.
presenceDetectorState	Hier wird der aktuelle Zustand des *Anwesenheitskontaktes* gespeichert.
rainSensorState	Hier wird der aktuelle Zustand des *Regensensors* gespeichert
buttomHungContactState	Hier wird der aktuelle Zustand des *Kippkontaktes* gespeichert
windowOpenContactstate	Hier wird der aktuelle Zustand des *Offenkontaktes* gespeichert

4.2.17.2 Methodenschnittstellen der Klasse *WasteMainPanel*

Signatur: void *StartQuitButtonCB* (IlvGraphic* g)

Beschreibung: Diese Methode wird aufgerufen wenn der Start- bzw. Quit-Button gedrückt wurde. Beim ersten Aufruf (*systemActivated* ist noch nicht gesetzt) repräsentiert der zugehörige Button den Start-Button, d.h. die Aufschrift des Buttons wird zu „Quit" geändert, der Text „System is ready" erscheint im *SystemMessageField*, die Benutzungsschnittstelle wird durch Setzten des Flags *systemActivated* aktiviert und die Methode *startTransmission* des Monitors wird aufgerufen. Dieser wird so aktiv und teilt WASTE mit, daß die Benutzungsschnittstelle aktiviert wurde. Beim jedem weiteren Aufruf (*systemActivted* ist schon gesetzt) repräsentiert der zugehörige Button den Quit-Button, d.h. im *SystemMessageField* wird der Text „System is ready" angezeigt und falls noch kein QuitPanel geöffnet ist (*visibleQuitPanel* nicht gesetzt), so wird ein QuitPanel geöffnet, *visibleQuitPanel* gesetzt und ein Zeiger auf das QuitPanel in *quitPanelPtr* gespeichert.

Ansonsten (*visibleQuitPanel* schon gesetzt) wird das bereits angezeigte QuitPanel auf den Desktop hervor geholt.

Signatur: void *HelpButtonCB* (IlvGraphic* g)

Beschreibung: Diese Methode wird aufgerufen, wenn der Help-Button gedrückt wurde. Wenn zur Zeit kein HelpPanel angezeigt wird (*visibleHelpPanel* nicht gesetzt), so wird ein HelpPanel geöffnet, *visibleHelpPanel* gesetzt und ein Zeiger auf das HelpPanel in *helpPanelPtr* gespeichert. Ansonsten (*visibleHelpPanel* schon gesetzt) wird das bereits angezeigte HelpPanel auf den Desktop hervor geholt.

Signatur: void *ShowHideButtonCB* (IlvGraphic* g)

Beschreibung: Diese Methode wird aufgerufen, wenn der Show- bzw. Hide-Button gedrückt wurde. Sie überprüft zuerst, ob die Benutzungsschnittstelle schon aktiviert wurde (d.h. ob *systemActivated* gesetzt ist). Falls nicht, so wird eine entsprechende Fehlermeldung im *SystemMessageField* angezeigt. Ansonsten wird das Flag *showSensors* überprüft. Ist es gesetzt (die optionalen Sensorenwerte werden angezeigt), so wurde der Hide-Button betätigt. Seine Beschriftung wird in „Show" geändert und die Name- und ValueLabel der Sensorenwerte werden mit Leerzeichen beschrieben. Außerdem wird *showSensors* zurück gesetzt. Ist es nicht gesetzt, so wurde der Show-Button betätigt. Seine Beschriftung wird in „Hide" geändert und die Name- und ValueLabel der Sensorenwerte werden mit ihren Namen und den aktuellen in den entsprechenden Attributen zwischengespeicherten Werten beschrieben. Außerdem wird *showSensors* gesetzt. In beiden Fällen wird eine eventuelle Fehlermeldung im *SystemMessageField* gelöscht.

Signatur: void *PresenceTempScrollBarCB* (IlvGraphic* g)

Beschreibung: Diese Methode wird bei der Benutzung des Schiebereglers der *AnwesenheitsTempNormal* aufgerufen. Sie überprüft zuerst, ob die Benutzungsschnittstelle noch nicht aktiviert wurde (d.h. *systemActivated* ist nicht gesetzt) oder ein QuitPanel geöffnet ist (d.h. *visibleQuitPanel* ist gesetzt). Falls ja, so wird eine entsprechende Fehlermeldung im *SystemMessageField* angezeigt und der Schieberegler wieder auf den Wert eingestellt, der im dazugehörigen Anzeigefeld angezeigt wird (d.h. die Benutzung wird rückgängig gemacht und damit ignoriert. Ansonsten wird der neue Wert des Scheibereglers im Anzeigefeld angezeigt und über die Methode *setPresenceTemp* an den Monitor gemeldet. Dabei wird eine eventuelle Fehlermeldung im *SystemMessageField* gelöscht.

Signatur: void *AbsenceTempScrollBarCB* (IlvGraphic* g)

Beschreibung: Diese Methode wird bei der Benutzung des Schiebereglers der *AbwesenheitsTempNormal* aufgerufen. Sie überprüft zuerst, ob die Benutzungsschnittstelle noch nicht aktiviert wurde (d.h. *systemActivated* ist nicht gesetzt) oder ein QuitPanel geöffnet ist (d.h. *visibleQuitPanel* ist gesetzt). Falls ja, so wird eine entsprechende Fehlermeldung im

SystemMessageField angezeigt und der Schieberegler wieder auf den Wert eingestellt, der im dazugehörigen Anzeigefeld angezeigt wird (d.h. die Benutzung wird rückgängig gemacht und damit ignoriert. Ansonsten wird der neue Wert des Scheibereglers im Anzeigefeld angezeigt und über die Methode *setAbsenceTemp* an den Monitor gemeldet. Dabei wird eine eventuelle Fehlermeldung im *SystemMessageField* gelöscht.

Signatur: **void *HeatingUpTimeScrollBarCB* (IlvGraphic* g)**

Beschreibung: Diese Methode wird bei der Benutzung des Schiebereglers der *ZeitspanneAufheizenNormal* aufgerufen. Sie überprüft zuerst, ob die Benutzungsschnittstelle noch nicht aktiviert wurde (d.h. *systemActivated* ist nicht gesetzt) oder ein QuitPanel geöffnet ist (d.h. *visibleQuitPanel* ist gesetzt). Falls ja, so wird eine entsprechende Fehlermeldung im *SystemMessageField* angezeigt und der Schieberegler wieder auf den Wert eingestellt, der im dazugehörigen Anzeigefeld angezeigt wird (d.h. die Benutzung wird rückgängig gemacht und damit ignoriert. Ansonsten wird überprüft, ob der neue Wert des Scheibereglers mindestens fünf Minuten größer ist als der Wert, der im Anzeigefeld der *ZeitspanneEintrittNormal* steht. Ist dies nicht der Fall, so wird die Eingabe wie zuvor ignoriert und eine entsprechende Fehlermeldung angezeigt. Ansonsten wird der neue Wert des Schiebereglers im Anzeigefeld angezeigt und über die Methode *setHeatingUpTime* an den Monitor gemeldet. Dabei wird eine eventuelle Fehlermeldung im *SystemMessageField* gelöscht.

Signatur: **void *EntryRemarkTimeScrollBarCB* (IlvGraphic* g)**

Beschreibung: Diese Methode wird bei der Benutzung des Schiebereglers der *ZeitspanneEintrittNormal* aufgerufen. Sie überprüft zuerst, ob die Benutzungsschnittstelle noch nicht aktiviert wurde (d.h. *systemActivated* ist nicht gesetzt) oder ein QuitPanel geöffnet ist (d.h. *visibleQuitPanel* ist gesetzt). Falls ja, so wird eine entsprechende Fehlermeldung im *SystemMessageField* angezeigt und der Schieberegler wieder auf den Wert eingestellt, der im dazugehörigen Anzeigefeld angezeigt wird (d.h. die Benutzung wird rückgängig gemacht und damit ignoriert. Ansonsten wird überprüft, ob der neue Wert des Scheibereglers mindestens fünf Minuten kleiner ist als der Wert, der im Anzeigefeld der *ZeitspanneAufheizenNormal* steht. Ist dies nicht der Fall, so wird die Eingabe wie zuvor ignoriert und eine entsprechende Fehlermeldung angezeigt. Ansonsten wird der neue Wert des Schiebereglers im Anzeigefeld angezeigt und über die Methode *setEntryRemarkTime* an den Monitor gemeldet. Dabei wird eine eventuelle Fehlermeldung im *SystemMessageField* gelöscht.

Signatur: **void *ExitRemarkTimeScrollBarCB* (IlvGraphic* g)**

Beschreibung: Diese Methode wird bei der Benutzung des Schiebereglers der *ZeitspanneVerlassenNormal* aufgerufen. Sie überprüft zuerst, ob die Benutzungsschnittstelle noch nicht aktiviert wurde (d.h. *systemActivated* ist nicht gesetzt) oder ein QuitPanel geöffnet ist (d.h. *visibleQuitPanel* ist gesetzt). Falls ja, so wird eine entsprechende Fehlermeldung

im *SystemMessageField* angezeigt und der Schieberegler wieder auf den Wert eingestellt, der im dazugehörigen Anzeigefeld angezeigt wird (d.h. die Benutzung wird rückgängig gemacht und damit ignoriert. Ansonsten wird der neue Wert des Scheibereglers im Anzeigefeld angezeigt und über die Methode *setExitRemarkTime* an den Monitor gemeldet. Dabei wird eine eventuelle Fehlermeldung im *SystemMessageField* gelöscht.

Signatur: **void *PresenceTempResetButtonCB* (IlvGraphic* g)**

Beschreibung: Diese Methode wird bei der Benutzung des Default-Buttons der *AnwesenheitsTempNormal* aufgerufen. Sie überprüft zuerst, ob die Benutzungsschnittstelle noch nicht aktiviert wurde (d.h. *systemActivated* ist nicht gesetzt) oder ein QuitPanel geöffnet ist (d.h. *visibleQuitPanel* ist gesetzt). Falls ja, so wird eine entsprechende Fehlermeldung im *SystemMessageField* angezeigt. Ansonsten wird der Defaultwert (gespeichert in *presenceTempDefault*) im Anzeigefeld angezeigt und dies über die Methode *setPresenceTempDefault* an den Monitor gemeldet. Dabei wird eine eventuelle Fehlermeldung im *SystemMessageField* gelöscht.

Signatur: **void *AbsenceTempResetButtonCB* (IlvGraphic* g)**

Beschreibung: Diese Methode wird bei der Benutzung des Default-Buttons der *AbwesenheitsTempNormal* aufgerufen. Sie überprüft zuerst, ob die Benutzungsschnittstelle noch nicht aktiviert wurde (d.h. *systemActivated* ist nicht gesetzt) oder ein QuitPanel geöffnet ist (d.h. *visibleQuitPanel* ist gesetzt). Falls ja, so wird eine entsprechende Fehlermeldung im *SystemMessageField* angezeigt. Ansonsten wird der Defaultwert (gespeichert in *absenceTempDefault*) im Anzeigefeld angezeigt und dies über die Methode *setAbsenceTempDefault* an den Monitor gemeldet. Dabei wird eine eventuelle Fehlermeldung im *SystemMessageField* gelöscht.

Signatur: **void *HeatingUpTimeResetButtonCB* (IlvGraphic* g)**

Beschreibung: Diese Methode wird bei der Benutzung des Default-Buttons der *ZeitspanneAufheizenNormal* aufgerufen. Sie überprüft zuerst, ob die Benutzungsschnittstelle noch nicht aktiviert wurde (d.h. *systemActivated* ist nicht gesetzt) oder ein QuitPanel geöffnet ist (d.h. *visibleQuitPanel* ist gesetzt). Falls ja, so wird eine entsprechende Fehlermeldung im *SystemMessageField* angezeigt. Ansonsten wird überprüft, ob der Defaultwert (gespeichert in heatingUpTimeDefault) mindestens fünf Minuten größer ist als der Wert, der im Anzeigefeld der *ZeitspanneEintrittNormal* steht. Ist dies nicht der Fall, so wird eine entsprechende Fehlermeldung angezeigt. Ansonsten wird der Defaultwert im Anzeigefeld angezeigt und über die Methode *setHeatingUpTimeDefault* an den Monitor gemeldet. Dabei wird eine eventuelle Fehlermeldung im *SystemMessageField* gelöscht.

Signatur: **void *EntryRemarkTimeResetButtonCB* (IlvGraphic* g)**

Beschreibung: Diese Methode wird bei der Benutzung des Default-Buttons der *ZeitspanneEintrittNormal* aufgerufen. Sie überprüft zuerst, ob die Benutzungsschnittstelle noch nicht aktiviert wurde (d.h. *systemActivated* ist nicht gesetzt) oder ein QuitPanel geöffnet ist (d.h. *visibleQuitPanel* ist gesetzt). Falls ja, so wird eine entsprechende Fehlermeldung im *SystemMessageField* angezeigt. Ansonsten wird überprüft, ob der Defaultwert (gespeichert in entryRemarkTimeDefault) mindestens fünf Minuten kleiner ist als der Wert, der im Anzeigefeld der *ZeitspanneAufheizenNormal* steht. Ist dies nicht der Fall, so wird eine entsprechende Fehlermeldung angezeigt. Ansonsten wird der Defaultwert im Anzeigefeld angezeigt und über die Methode *setEntryRemarkTimeDefault* an den Monitor gemeldet. Dabei wird eine eventuelle Fehlermeldung im *SystemMessageField* gelöscht.

Signatur: void *ExitRemarkTimeResetButtonCB* (IlvGraphic* g)

Beschreibung: Diese Methode wird bei der Benutzung des Default-Buttons der *ZeitspanneVerlassenNormal* aufgerufen. Sie überprüft zuerst, ob die Benutzungsschnittstelle noch nicht aktiviert wurde (d.h. *systemActivated* ist nicht gesetzt) oder ein QuitPanel geöffnet ist (d.h. *visibleQuitPanel* ist gesetzt). Falls ja, so wird eine entsprechende Fehlermeldung im *SystemMessageField* angezeigt. Ansonsten wird der Defaultwert (gespeichert in *exitRemarkTimeDefault*) im Anzeigefeld angezeigt und dies über die Methode *setExitRemarkTimeDefault* an den Monitor gemeldet. Dabei wird eine eventuelle Fehlermeldung im *SystemMessageField* gelöscht.

Signatur: void *setMainApplicationPtr* (WasteUI* ptr)

Beschreibung: Diese Methode speichert einen Zeiger auf die Applikation in *mainApplicationPtr*.

Signatur: void *setSchedulerPtr* (Scheduler* ptr)

Beschreibung: Diese Methode speichert einen Zeiger auf den Scheduler in *schedulerPtr*.

Signatur: void *setMonitorPtr* (MonitorUI* ptr)

Beschreibung: Diese Methode speichert einen Zeiger auf den Monitor in *monitorPtr*.

Signatur: void *setMsgHandlerPtr* (MsgHandlerUI* ptr)

Beschreibung: Diese Methode speichert einen Zeiger auf den Message-Handler in *msgHandlerPtr*. Außerdem benutzt sie die Methode *setMainPanelPtr* um dem Message-Handler einen Zeiger auf das MainPanel zu übermitteln.

Signatur: void *clearHelpPanel* (void)

Beschreibung: Diese Methode schließt das HelpPanel und setzt *visibleHelpPanel* zurück.

Signatur: void *clearQuitPanel* (void)

Beschreibung: Diese Methode schließt das QuitPanel und setzt *visibleQuitPanel* zurück.

Signatur: void *setPresenceTempDefault* (int temp)

Beschreibung: Diese Methode speichert den Wert *temp* in *presenceTempDefault* und zeigt ihn im entsprechenden Anzeigefeld und am entsprechenden Schieberegler an.

Signatur: void *setAbsenceTempDefault* (int temp)

Beschreibung: Diese Methode speichert den Wert *temp* in *absenceTempDefault* und zeigt ihn im entsprechenden Anzeigefeld und am entsprechenden Schieberegler an.

Signatur: void *setHeatingUpTimeDefault* (int time)

Beschreibung: Diese Methode speichert den Wert *time* in *heatingUpTimeDefault* und zeigt ihn im entsprechenden Anzeigefeld und am entsprechenden Schieberegler an.

Signatur: void *setEntryRemarkTimeDefault* (int time)

Beschreibung: Diese Methode speichert den Wert *time* in *entryRemarkTimeDefault* und zeigt ihn im entsprechenden Anzeigefeld und am entsprechenden Schieberegler an.

Signatur: void *setExitRemarkTimeDefault* (int time)

Beschreibung: Diese Methode speichert den Wert *time* in *exitRemarkTimeDefault* und zeigt ihn im entsprechenden Anzeigefeld und am entsprechenden Schieberegler an.

Signatur: void *showRoomTemp* (int temp)

Beschreibung: Diese Methode zeigt den Wert *temp* im Anzeigefeld der Raumtemperatur an.

Signatur: void *showOutdoorTemp* (int temp)

Beschreibung: Diese Methode zeigt den Wert *temp* im Anzeigefeld der Außentemperatur an.

Signatur: void *showPresenceDetectorState* (int state)

Beschreibung: Diese Methode speichert den Wert *state* in *presenceDetectorState*. Ist *showSensors* gesetzt wird das entsprechende NameLabel in Abhängigkeit von *state* neu beschrieben.

Signatur: void *showRainSensorState* (int state)

Beschreibung: Diese Methode speichert den Wert *state* in *rainSensorState*. Ist *showSensors* gesetzt wird das entsprechende NameLabel in Abhängigkeit von *state* neu beschrieben.

Signatur: void *showButtomHungContactState* (int state)

Beschreibung: Diese Methode speichert den Wert *state* in *buttomHungContactState*. Ist *showSensors* gesetzt wird das entsprechende NameLabel in Abhängigkeit von *state* neu beschrieben.

Signatur: void *showWindowOpenContactState* (int state)

Beschreibung: Diese Methode speichert den Wert *state* in *windowOpenContactState*. Ist *showSensors* gesetzt wird das entsprechende NameLabel in Abhängigkeit von *state* neu beschrieben.

Signatur: void *terminatePrepare* (void)

Beschreibung: Diese Methode schließt das geöffnete QuitPanel und ruft die Methode *terminateConnection* des Monitors auf.

Signatur: void *terminateFinish* (void)

Beschreibung: Diese Methode stoppt den Scheduler, löscht das MainPanel und die Application und beendet das Programm.

4.2.17.3 Verbindungsstruktur

Die Klasse *WasteMainPanel* hat Verbindungen zum Monitor und Message-Handler der Benutzungsschnittstelle, zum Scheduler, zur Applikation und zu eventuellen Help- und QuitPanels.

4.2.18 Die Klasse *WasteQuitPanel*

Die Klasse *WasteQuitPanel* wurde von der Klasse *QuitPanel* abgeleitet. Sie realisiert die Funktionalität des QuitPanels.

4.2.18.1 Attribute der Klasse *WasteQuitPanel*

Es wurden folgende neue Attribute hinzugefügt.:

Attribut	Erläuterung
mainPanelPtr	Zeiger auf das MainPanel.

4.2.18.2 Methodenschnittstellen der Klasse *WasteQuitPanel*

Signatur: void *QuitConfirmButtonCB* (IlvGraphic* g)

Beschreibung: Diese Methode wird aufgerufen, wenn der Yes-Button des QuitPanels gedrückt wurde. Sie ruft die Methode *terminatePrepare* des MainPanels auf.

Signatur: void *QuitRefuseButtonCB* (IlvGraphic* g)

Beschreibung: Diese Methode wird aufgerufen, wenn der No-Button des QuitPanels gedrückt wurde. Sie ruft die Methode *clearQuitPanel* des MainPanels auf.

Signatur: void *setMainPanelPtr* (WasteMainPanel* ptr)

Beschreibung: Diese Methode speichern einen Zeiger auf das MainPanel im Attribut *mainPanelPtr*.

4.2.18.3 Verbindungsstruktur

Das *WasteQuitPanel* hat eine Azzoziation zum MainPanel.

4.2.19 Die Klasse *WasteUI*

Die Klasse *WasteUI* wurde von ILOG Views generiert. Sie repräsentiert die Applikation. In ihrem Code befindet sich die Main-Methode (wird von ILOG Views so generiert). Nachdem die Applikation mittels *new* angelegt und durch run gestartet wurde, wird u.a. die Methode *makePanels* aufgerufen.

4.2.19.1 Attribute der Klasse *WasteUI*

Es wurden folgende neue Attribute hinzugefügt.:

Attribut	Erläuterung
mainPanelPtr	Zeiger auf das MainPanel.
msgHandlerPtr	Zeiger auf den Message-Handler.
schedulerPtr	Zeiger auf den Scheduler

4.2.19.2 Methodenschnittstellen der Klasse *WasteUI*

Signatur: void *makePanels* (void)

Beschreibung: Diese Methode wurde von ILOG Views generiert. Sie wurde wie folgt erweitert: in ihr das MainPanel angelegt und angezeigt. Außerdem werden dem MainPanel über die Methoden *setMainApplicationPtr*, *setMsgHandlerPtr* und *setSchedulerPtr* Zeiger auf die Applikation, den Message-Handler und den Scheduler übermittelt.

Signatur: void *setSchedulerPtr* (Scheduler* ptr)

Beschreibung: Diese Methode speichert den Zeiger *ptr* auf den Scheduler im Attribut *schedulerPtr*.

Signatur: void *setMsgHandlerPtr* (MsgHandlerUI* ptr)

Beschreibung: Diese Methode speichert den Zeiger *ptr* auf den Message-Handler im Attribut *msgHandlerPtr*.

4.2.19.3 Verbindungsstruktur

Die Applikation hat eine Aggregation zum MainPanel.

4.2.20 Sonstige Komponenten

4.2.20.1 Die Datei *msgType.h*

Diese Datei enthält nur die Typdefinition des Types *msgType*, wie er in 4.1.2 beschrieben wurde.

4.2.20.2 Die Datei *wasteCS.C*

Die Datei *wasteCS.C* enthält die Main-Methode des Kontrollsystems. In ihr werden ein Scheduler, ein Portbaustein, eine Zimmerflächenliste mit einem Zimmer, ein Timer und das Haus angelegt. Dem Haus wird anschließend ein Zeiger auf den Scheduler mittels der Methode *setSchedulerPtr* übermittelt. Nun wird der Scheduler gestartet.