Anhang B

Anleitung zur Erweiterung

B.1 Portierung

Zur Zeit ist WASTE nur auf *SunOS 4.1.4* ablauffähig. Bei der Portierung auf *Solaris* sind folgende Probleme zu lösen:

- Der Scheduler ist unter *Solaris* nicht ablauffähig, denn er benutzt den Systemaufruf *ualarm*, der unter *Solaris* eine andere Signatur hat als unter *SunOS*.
- Die Message-Handler benutzen beim Verbindungsaufbau Systemaufrufe. Sie sind als extern deklariert und haben unter *Solaris* eine andere Singnatur.
- Der von ILOG Views generierte Kode muß mit der Version, die unter *Solaris* arbeitet neu generiert werden und es müssen andere Bibliotheken im *makefile* hinzugelinkt werden.

B.2 Erweiterung der Benutzungsschnittstelle

Als Beispiel für die Erweiterung von WASTE werden ein Sensor und ein Aktor betrachtet. Der Sensor soll wie die Zimmertemperatur an der Benutzungsschnittstelle angezeigt werden. Der Aktor soll von der Benutzungsschnittstelle aus, ähnlich der *AnwesenheitsTemp* über einen Schiebebalken und einen Default-Button angesteuert werden. Diese Erweiterung betrifft alle Bereiche dieser Arbeit.

Die Erweiterung der Benutzungsschnittstelle wird mit Hilfe von ILOG Views realisiert. Hier werden nur einige Konzepte von ILOG Views aufgezeigt. Für nähere, detailliertere Anleitungen sei auf die entsprechenden Handbücher verwiesen. ILOG Views wird (unter *SunOS*) durch *start-ilog -is* gestartet. Es erscheinen vier Fenster: ein Arbeitsfenster mit dem Arbeitsbereich (Titel i*vstudio:nomane*), ein Fenster mit sog. Gadgets (das sind vordefinierte grafische Elementen, die mittels Drag-and-Drop auf dem Arbeitsbereich abgelegt werden können; Titel *Gadgets*), ein Applikationsfenster (Titel *ApplicationEditor: testapp*), in dem die Generierung des Kodes gesteuert wird und ein Fenster, das mitteilt, welche Version von ILOG Views gestartet wurde. Letzteres wird sofort wieder geschlossen. ILOG Views arbeitet mit Dateien der Endung *.ilv* (sie repräsentieren u.a. die Panel) und *.iva* (sie repräsentieren die Applikation). Diese Dateien befinden sich im Verzeichnis *~werner/projekt/Code/WasteUI/ILV.* Für die betrachtete Erweiterung sind nur die Dateien *mainPanel.ilv* und *wasteUI.iva* von Interesse. Die Datei *mainPanel.ilv* wird im Arbeitsfenster geöffnet. Nun werden mittels Drag-and-Drop die gewünschten grafischen Elemente von Gadget-Fenster aus auf der Benutzungsschnittstelle plaziert. Alternativ können bestehende Elemente auch einfach kopiert werden (dabei wird, wenn mehrere Elemente gleichzeitig kopiert werden, auch die Abstände zwischen den einzelnen Elementen beibehalten). Allen Elementen, die neu hinzugefügt oder einkopiert werden, sollte ein Namen und falls diese Elemente aktiv durch den Benutzer bedient werden sollen, auch ein Name für die entsprechende CallBack-Methode zugeteilt werden. Diese Namen werden im unteren Bereich den Arbeitsfensters in die dafür vorgesehenen Felder eingetragen und sollten sich aus Konsistenzgründen an den bereits verwendeten Namen orientieren. Durch ein Doppelklick auf ein Element wird ein Fenster geöffnet, in dem man weitere Einstellungen vornehmen kann. Um den Kode zu generieren wird im Applikationsfenster die Datei *wasteUI.iva* geöffnet. Im linken Teilbereich dieses Fensters wird der Begriff *mainPanel* markiert. Daraufhin erscheint im linken Teilbereich der Begriff *MainPanel*. Auch dieser wird markiert. Nun wird auf das Auge in der Icon-Zeile oben im Fenster geklickt. Daraufhin öffnet sich ein Fenster, in dem man folgende Einstellungen vornehmen muß: die *Callbacks* müssen abgewählt werden und unter System muß *Sparc_4_gnu* eingestellt werden. Anschließend wir auf *Apply* und *Close* gedrückt. In der Menüleiste wird unter dem Punkt *Code* der Punkt *Generate Panel Class* angewählt. Der Kode der Superklasse *MainPanel* wird daraufhin neu generiert. Nachdem die Dateien *mainPanel.ilv* und *wasteUI.iva* abgespeichert wurden, kann ILOG Views verlassen werden. Der neue Kode befindet sich im Verzeichnis *~werner/projekt/Code/WasteUI/ILV/Data* und muß zusammen mit der Datei *wasteUI.iva* ins Verzeichnis *~werner/projekt/Code/WasteUI/src* kopiert werden. Dort kann die Funktionalität der Erweiterung in die Datei *wasteMainPanel.cc* von Hand eingetragen werden. Dazu müssen die neuen CallBack-Methoden erstellt werden. Als Vorlagen können hierbei die Methoden *PresenceTempScrollBarCB* und *PresenceTempResetButtonCB* dienen.

B.3 Erweiterung der Anbindung der Benutzungsschnittstelle

B.3.1 Erweiterung der Klasse *MonitorUI*

Der Monitor der Benutzungsschnittstelle (die Klasse *MonitorUI*) muß Methoden bereitstellen, die es dem MainPanel ermöglichen, die zusätzlichen Benutzereingaben zu „melden“. Dazu müssen folgende Attribute und Methoden hinzugefügt werden: *actorValue*, *actorValueHasChanged*, *actorValueChangeTime*, *setActorValue* (value) und *setActorValueDefault* (). Weiterhin muß die Überwachung des Aktors in die CallBack-Methode integriert werden. Sie läuft analog zur Überwachung der anderen Aktoren (z.B. der *PresenceTempScrollBar* und des *PresenceTempResetButtons*) ab. Für die Header, die dabei generiert werden können, müssen im File *msgType.h* zwei weitere Nachrichtentypen *CmsgTypeSetActorValue* und *CmsgTypeSetActorValueDefault* eingetragen werden.

B.3.2 Erweiterung der Klasse *MsgHandlerRoom*

Der Message-Handler des Zimmers (die Klasse *MsgHandlerRoom*) muß auf den Empfang der neuen Nachrichtentypen *CmsgTypeSetActorValue* und *CmsgTypeSetActorValueDefault* vorbereitet werden. Dazu ist das große *switch*-Konstrukt in der CallBack-Methode analog zum Empfang von z.B. *CmsgTypeSetPresenceTemp* und *CmsgTypeSetPresenceTempDefault* zu erweitern.

B.3.3 Erweiterung der Klasse *MonitorRoom*

Der Monitor des Zimmers (die Klasse *MonitorRoom*) muß in der CallBack-Methode zusätzlich den Zustands des neuen Sensors überwachen. Dies geschieht analog zur Überwachung von z.B. der Raumtemperatur. Für den Header, der dabei generiert werden kann, muß im File *msgType.h* noch ein weiterer Nachrichtentyp *CmsgTypeSetSensorValue* eingetragen werden. Außerdem muß in die Methode *startMonitoring* die Übermittlung des Defaultwertes für den Aktor analog zur Übermittlung des Defaultwertes des *AnwesenheitsTemp* eingefügt werden. Dabei wird der Nachrichtentyp *CmsgTypeSetActorValueDefault* genutzt.

B.3.4 Erweiterung der Klasse *MsgHandlerUI*

Der Message-Handler der Benutzungsschnittstelle (die Klasse *MsgHandlerUI*) muß auf den Empfang der Nachrichtentypen *CmsgTypeSetSensorValue* und *CmsgTypeSetActorValueDefault* vorbereitet werden. Die Behandlung dieser Nachrichtentypen wird in die Callback-Methode analog zur Behandlung von z.B der Nachrichtentypen *CmsgTypeSetRoomTemp* und *CmsgTypeSetPresenceTempDefault* in das große *switch*-Konstrukt integriert.

B.3.5 Erweiterung der Klasse *Zimmer*

Die Klasse *Zimmer* muß folgende Methoden bereitstellen: *setActorValue* (value), *setActorValueDefault* (), *getActorValueDefault* (), *getSensorValue* (). Sie entsprechen den Methoden *AnwesenheitsTempNormalSetzen* (Temperatur), *AnwesenheitsTempAufDefault* (), *getPresence-TempDefault* () und *getRoomTemp* () und werden vom Monitor bzw. Message-Handler genutzt, um Werte in das Kontrollsystem zu übermitteln bzw. Werte aus dem Kontrollsystem auszulesen.

B.3.6 Erweiterung der Klasse *WasteMainPanel*

Die Klasse WasteMainPanel muß folgendes Attribut und folgende Methoden bereitstellen: *actorValueDefault*, *setActorValueDefault* (value) und *showSensorValue* (value). Sie entsprechen dem Attribut *presenceTempDefault* und den Methoden *setPresenceTempDefault* (value) und *showRoomTemp* (value) und dienen dem Message-Handler dazu Werte an die Benutzungsschnittstelle zu übermitteln.

B.4 Erweiterung der Anbindung der Aktoren und Sensoren

Für die Erweiterung der Anbindung der Aktoren und Sensoren werden folgende Annahmen gemacht: der Sensor ist ein binärer Sensor, bekommt Port 11 zugewiesen und wird über die Methode *CallBitInputPort* angesprochen. Außerdem existieren im Testraum/Simulator die realen Sensoren *sen1* und *sen2* die auf diesen Sensor gemappt werden. Der Aktor ist ein gewöhnlicher Aktor, bekommt Port 12 zugewiesen und wird über die Methode *CallIntOutPutPort* angesprochen. Sein Defaultwert ist „1“. Für ihn steht im Testraum/Simulator der Aktor *act1* zur Verfügung. Umrechnungen sind keine nötig.

B.4.1 Erweiterung der Klasse *Portbaustein*

In der Klasse *Portbaustein* sind die Methoden *CallBitInputPort* und *CallIntOutputPort* zu erweitern. In beiden Methoden müssen in den *switch*-Konstrukten die neuen Ports (also bei *CallBitInputPort* der Port 11 und bei *CallIntOutputPort* der Port 12) als *case* eingetragen werden, so daß sie die neuen Ports bei einem Aufruf akzeptieren und ihn zum Port-Handler weiter leiten.

B.4.2 Erweiterung der Klasse *PortHandler*

Die Klasse *PortHandler* stellt nur Durchreichmethoden zur Verfügung. Sie braucht nicht erweitert zu werden.

B.4.3 Erweiterung der Klasse *Mapper*

Da sich die Anzahl der Ports auf 13 erhöht hat, muß dieser Wert in der Datei *mapper.h* der Konstanten *maxPorts* zugewiesen werden. Andere Konstanten (max. Anzahl der Aktoren/Sensoren pro Port und Länge der Namen der Aktoren/Sensoren) werden nicht verändert. In der Klasse *Mapper* muß im Konstruktor die Initialisierung der Spalten (d.h. der Ports) 11 und 12 eingetragen werden. Sie muß wie folgt aussehen:

```
// Port 11 is the Sensor, 2 phys. sensors, values stays 0
mapperMatrix[11][0].name  = „#A&S“;
mapperMatrix[11][0].value = 2;      // # of phys. sensors

mapperMatrix[11][1].name  = „sen1“;
mapperMatrix[11][2].name  = „sen2“;

mapperMatrix[11][maxPhysActSens].name  = „stat“;
mapperMatrix[11][maxPhysActSens].value = -2;   //active sensor


// Port 12 is the Aktor, 1 phys. actor
mapperMatrix[12][0].name  = „#A&S“;
mapperMatrix[12][0].value = 1;      // # of phys. actors

mapperMatrix[12][1].name  = „act“;
mapperMatrix[12][1].value = 1;      // defaultvalue is 1

mapperMatrix[12][maxPhysActSens].name  = „stat“;
mapperMatrix[12][maxPhysActSens].value = 1;   //normal actor
```

Weitere Erweiterungen sind nicht nötig, denn es werden keine neuen Klassen von Aktoren/Sensoren (die bisherigen Stati reichen aus) und keine neuen Sonderbehandlungen eingeführt.

B.4.4 Erweiterung der Klasse *MsgHandlerPB*

Da die Namen der Aktoren und Sensoren den Methoden der Klasse *MsgHandlerPB* als Parameter übergeben werden und diese auf die Funktionalität der Methoden keinen Einfluß haben, braucht die Klasse *MsgHandlerPB* nicht verändert zu werden.