Diplomarbeit

zur Erlangung des akademischen Grades eines Diplom-Informatikers (FH)

an der

Fachhochschule Furtwangen
Hochschule für Technik und Wirtschaft
Fachbereich Informatik
Studiengang Allgemeine Informatik

Thema:

Embedded PCs und Linux als Benutzerschnittstelle zu einem Server am Beispiel eines Zeiterfassungssystems

eingereicht im Sommersemester 2002 von
Daniel Dreher
Aschenreutestraße 8
78591 Durchhausen

erstellt bei der Firma
Interflex Datensysteme GmbH & Co. KG
Großwiesenstraße 24
78591 Durchhausen

Referent:
Prof. Dr. Wolfgang Bauer

Koreferent:
Dipl.-Ing. (FH) Manfred Klostermeier

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, daß ich die vorliegende Diplomarbeit selbständig und ohne unzulässige fremde Hilfe angefertigt habe. Die verwendeten Quellen und Hilfsmittel sind vollständig zitiert.

Durchhausen, den 30. Juli 2002

_____

Daniel Dreher
Aschenreutestraße 8
78591 Durchhausen

1 Inhaltsverzeichnis

1 Inhaltsverzeichnis 3

2 Einleitung 6

3 Motivation 8

4 Grundlagen der Zeiterfassung 10

4.1 Betriebsdatenerfassung (BDE) 10

4.2 Zeitwirtschaft 10

5 Konfigurierbare Clients im Allgemeinen 14

5.1 Analyse 14

5.2 Mögliche Modellierungen 14

5.2.1 Logik 15

5.2.2 Oberfläche 17

5.2.3 Bestehende fertige Modelle 19

5.2.4 Entscheidungsfindung 20

5.3 Das Interface-Action-Config Modell 21

5.3.1 Vorstellung des Modells 21

5.3.2 Ablauf am Beispiel Zeiterfassung 23

5.3.3 Protokollierung 24

5.3.4 Debugging 25

5.3.5 Problematik der Synchronisation 25

5.4 Allgemeine Lösungsansätze 26

5.4.1 Bildschirmtastatur 26

5.4.2 Oberflächenkonfigurierung 26

5.4.3 Offline-Fähigkeiten 27

6 Übertragung auf das Beispiel Zeiterfassungssystem 28

6.1 Analyse 28

6.1.1 Anforderungen 28

6.2 Benutzerschnittstelle 29

6.2.1 Bibliotheken 29

6.2.2 Verschiedene Steuerelemente 30

6.3 Aktionen 30

6.3.1 Synchronisation 30

6.3.2 Vorhandene Aktionstypen 31

6.4 Variablen 31

6.5 Konfiguration 32

6.5.1 Allgemeine Konfiguration – Knoten <general> 32

6.5.2 Konfiguration der Oberfläche – Knoten <view> 34

6.5.3 Konfiguration der Aktionen – Knoten <action> 36

6.5.4 Implementierung mit Hilfe von XML 39

6.6 Kapselung der externen Schnittstellen 40
6.6.1 Schnittstelle zum Aufruf externer Programm „shell“ 40
6.6.2 Schnittstelle zur Verwendung seriell angeschlossener Geräte 40
6.6.3 Schnittstelle zum Anschluß an System 6020-Buchungsschnittstelle „ifterapi“ bzw. „IfMulterapi“ 41
6.6.4 Schnittstelle zum Aufruf von Funktionen in dynamischen Bibliotheken 41
6.6.5 Schnittstelle zum Anschluß an die Programmierschnittstelle des System 6020 „api6020“ 42
6.7 Diverses 42
6.7.1 Zeichensätze und Unicode 42
6.7.2 Parsen von Strings 43
6.8 Hilfsprogramme und –bibliotheken 49
6.8.1 ifterapi bzw. IfMulterapi 49
6.8.2 Auswertung mathematischer Ausdrücke 55
6.8.3 Konvertierung UCS2 nach UTF-8 55
6.8.4 Test der seriellen Schnittstelle 55
6.8.5 Aufruf von dynamischen Bibliotheken 56
6.9 Erweiterbarkeit 56
6.10 Tests 56
7 Ausgewählte Probleme der Systemprogrammierung 57
7.1 Gepuffertes Erzeuger-Verbraucher-Problem 57
7.2 Aufruf von Unterprogrammen 58
7.2.1 Implementierung 58
7.2.2 Probleme 58
7.2.3 Lösung 59
7.3 Kritische Abschnitte 59
7.4 Warten auf Ressource 60
7.4.1 Implementierung 61
7.5 Hintereinanderabarbeiten von Aktionen bzw. Ansichten 63
7.5.1 Überblick 63
7.5.2 Sequentialisierung von Ereignissen 64
8 Embedded PCs 66
8.1 Hardware 66
8.2 Betriebssystem und Laufzeitumgebung 67
9 Zusammenfassung und Bewertung 68
10 Abbildungsverzeichnis 69
11 Tabellenverzeichnis 69

12 Quellen 70
12.1 Bücher und Loseblattsammlungen: 70
12.2 Internetseiten 71
12.2.1 Einzelne Seiten zu einem speziellen Thema 71
12.2.2 Referenzseiten zu größeren Themenbereichen 73
12.3 Dokumente 74
12.4 Quellen für Programmcode 74
12.5 Sonstiges 75
13 Anhang 76
13.1 Glossar 76
13.2 XML-Beispielkonfiguration 81
13.3 UTF-8-Codierung 86

2 Einleitung

„Computer sind dazu da, Probleme zu lösen, die wir ohne Computer gar nicht hätten.“ [GieT1]

Sicherlich ist dieser Ausspruch nicht hundertprozentig ernst gemeint, jedoch birgt er für viele Menschen ein Stückchen Wahrheit in sich. Auch als Informatiker kann man sich ihm leider nicht immer ganz verschließen. Darum sollte man etwas dagegen unternehmen. Und was bietet sich dazu besser an als die Schnittstelle zum Benutzer...

Client-Server-Architektur[Unb04] ist eine kooperative Informationsverarbeitung, bei der die Datenverarbeitung auf mehrere miteinander vernetzte Rechner aufgeteilt wird. Der Server stellt Dienste zur Verfügung, die der Client dann nutzt.

Server im Hintergrund spielen im Leben vieler Menschen eine Rolle, ohne daß sie sich dessen bewußt sind. An dieser Stelle seien nur einige Beispiele erwähnt: der Webserver im Intranet einer Firma, der einem Informationen zur aktuellen Situation liefert, der Mailserver, von dem man seine Briefe erhält, der Druckerserver im Firmennetzwerk oder das Warenwirtschaftssystem, das einem die Produktionsmittel verwaltet.
Mit all diesen Systemen kommt der normale Anwender nicht direkt in Berührung, sondern nur über den Client. Er bekommt Benutzerschnittstellen, mit deren Hilfe er mit dem System kommuniziert und nimmt nur diese wirklich wahr.
Eines unter diesen Systemen ist das System 6020 der Firma Interflex, das sich mit den Bereichen Zeitwirtschaft, Zutrittskontrolle, Betriebsdatenerfassung, Personaleinsatzplanung u. ä. beschäftigt.
Auch hier werden Schnittstellen benötigt, mit denen der Mensch mit dem System Daten austauschen kann. Bedingt durch die Komplexität des Systems und aufgrund vieler unterschiedlicher Anforderungen verschiedener Kunden ist es notwendig, die Schnittstellen so zu gestalten, daß sie den Bedürfnissen im Einzelfall angepaßt

werden können, aber auch einen möglichst großen Teil der Bedienung des Systems abdecken können.
Die Benutzerstelle sollte konfigurierbar sein, so daß die vom Serversystem angenommenen Abläufe leicht auf dem Client dargestellt werden können, wobei die Bedürfnisse des Eingebenden hinsichtlich der Dialoggestaltung berücksichtigt werden können.

An dieser Stelle möchte ich mich bei meinen Referenten Prof. Dr. Wolfgang Bauer und Manfred Klostermeier bedanken. Außerdem danke ich der Firma Interflex, die mir die Durchführung dieser Arbeit ermöglichte, speziell den Mitarbeitern, die mich unterstützten und stets ein offenes Ohr für mich hatten. Schließlich danke ich noch allen, die gelegentlich einen Blick in meine Arbeit warfen und mich so vor vielen kleinen Fehlern bewahrten.
Ich danke Konrad Zuse für die Erfindung des Computers und George Boole dafür, daß er die mathematischen Grundlagen für die Informatik gelegt hat.

3 Motivation

Heutzutage gibt es viele Stellen, an denen der Mensch mit einem informationstechnischen System kommunizieren soll. Dies gilt auch für die Applikationen der Firma Interflex, die Zeit und Betriebsdaten erfassen und Zutritt regeln.
Bislang war es zu diesem Zweck üblich an den Stellen, an denen die Daten anfielen (z. B. am Eingang der Produktionsstätte) kleine Terminals aufzuhängen, mit dessen Hilfe der Benutzer mit dem System kommuniziert. Die Benutzerschnittstelle war relativ bescheiden: Nur wenig Anzeige und ein paar Tasten. Solange es nur darum geht, dem System mitzuteilen, daß man gerne eintreten und mit der Arbeit beginnen möchte oder vorhat, den Betrieb für einen Dienstgang zu verlassen, wird dies auch in Zukunft noch ausreichen. Möchte man darüber hinaus noch seine Urlaubsplanung anzeigen lassen oder Produktionsdaten wie eine Projektnummer erfassen, sind die Grenzen des bestehenden Systems überschritten. Für Firmen, bei denen praktisch alle Mitarbeiter über einen PC am Arbeitsplatz verfügen, bietet die Firma Interflex bereits heute Systeme an, mit deren Hilfe man von seinem PC aus beispielsweise mit Hilfe eines Browsers[1] diese Aufgaben wahrnehmen kann. In der Produktion kommt es jedoch oft vor, daß nicht jeder Mitarbeiter einen eigenen PC hat. Es wird also ein System benötigt, mit dessen Hilfe man seine Aufgaben an einem zentralen Platz (z. B. am Eingang zur Produktionsstätte) erledigen kann. Ein normaler PC ist hier aus Kostengründen nicht sinnvoll. Außerdem wäre er auch zu anderen Zwecken verwendbar und stünde daher oft nicht für die buchungswilligen Mitarbeiter zur Verfügung. Ziel ist daher ein Rechner, der sich auf die Funktionalität Bedienung des Zeiterfassungssystems beschränkt.

Es gibt verschiedene Platinen mit relativ geringen Abmessungen und Touchscreen-Displays, die mit Pentium-kompatiblen Prozessoren arbeiten. Diese ließen sich dann beispielsweise einfach an der Wand am Eingang installieren.
Sollen derartige Terminals zu einem annehmbaren Preis verkauft werden, dürfen keine Lizenzkosten für das Betriebssystem entstehen, da diese bei jedem einzelnen Terminals anfallen. In der Vergangenheit hat die Firma Interflex bereits derartige Lösungen auf Basis normaler Rechner unter Windows, beispielsweise im Bereich der

[1] Produkt „WebClient“ der Firma Interflex

Personaleinsatzplanung, angeboten. Für einen größeren Markt sind die Kosten aber noch zu hoch. Daher hat sich Linux als Betriebssystem aufgedrängt, da einerseits keine Kosten entstehen, andererseits das System u.a. aufgrund des verfügbaren Quellcodes so eingeschränkt und angepaßt werden kann, daß es mit wenig Ressourcen auskommen kann.

Die oben erwähnten Lösungen haben neben der Betriebssystemproblematik einen weiteren Haken: Sie sind speziell zugeschnitten und lassen sich nicht parametrieren. Daraus ergibt sich für diese Arbeit die weitere Anforderung, die Konfiguration so flexibel zu gestalten, daß das Produkt in vielen verschiedenen Kontexten ohne Änderungen im Quellcode einsetzbar ist.

Ziel dieser Arbeit ist es also, die grundsätzliche Problematik der Anbindung von Embedded PCs unter Linux an ein Serversystem darzulegen und die Konfigurierbarkeit zu gewährleisten. Dies wird am Beispiel des Zeiterfassungssystems „6020“ der Firma Interflex gezeigt. Im Rahmen dessen sollte dann auch eine Applikation entwickelt werden, die es ermöglicht, eine Benutzerschnittstelle zu konfigurieren, welche die Bedienung des Zeiterfassungssystems ermöglicht. Diese sollte möglichst derartig gestaltet werden, daß sie durch die Systemberater der Firma Interflex so konfiguriert werden kann, daß ein Einsatz im produktiven Betrieb beim Kunden möglich ist. Ein Ziel dieser Arbeit ist die Erstellung eines lauffähigen Prototypen, der die im Rahmen der Untersuchung des Problems gewonnenen Erkenntnisse anwendet. In dieser Arbeit wird für die Applikation der Arbeitstitel "IfSoftterm“ verwendet.

Die Berücksichtigung von Aspekten wie Dialoggestaltung und Softwareergonomie tritt in dieser Arbeit eher in den Hintergrund, da dies der Verantwortung des Konfigurierenden obliegt.

4 Grundlagen der Zeiterfassung

In dieser Arbeit spielen die Abläufe und die Terminologie der Zeiterfassung eine grundliegende Rolle. Daher werden hier zum besseren Verständnis einige Grundbegriffe erklärt.

4.1 Betriebsdatenerfassung (BDE)

„Betriebsdatenerfassung (BDE) umfaßt die Maßnahmen, die erforderlich sind, um Betriebsdaten eines Produktionsbetriebs in maschinell verarbeitungsfähiger Form am Ort ihrer Verarbeitung bereitzustellen. BDE ist daher Oberbegriff für eine ganze Reihe von einzelnen Erfassungsverfahren wie Maschinendatenerfassung, mobile Datenerfassung, Prozeßdatenerfassung, Qualitätsdatenerfassung, Zeitdatenerfassung, Auftragsdatenerfassung, Lohndatenerfassung u.s.w.“[JunJ1]

BDE ist eine Untermenge des Computer Integrated Manufacturing (CIM), welches sich mit der Verwendung von Datenverarbeitung im Allgemeinen in der Produktion befaßt. Ein weiterer Bestandteil des CIMs, der auf Daten der BDE angewiesen ist, sind die Produktionsplanungs- und -steuerungssysteme (PPS).

4.2 Zeitwirtschaft

Die verschiedenen Grundbegriffe der Zeitwirtschaft sind folgendermaßen definiert; zu beachten ist, daß der Begriff Zeiterfassung oft synonym zu Zeitwirtschaft verwendet wird:

- **Zeiterfassung**[AdaB1]
  Erfassung nach Zeitarten (An- und Abwesenheiten) und Zeiträumen (von Datum/Uhrzeit bis Datum/Uhrzeit).
- **Zeitbewertung** [AdaB1]
  Soll-Ist-Vergleich (geplante und verfahrene Schicht) und Zuordnung der Zeitarten zu Lohnarten.
- **Zeitauswertung**[AdaB1]
  Informationen über An- und Abwesenheiten, aktuelle Kontenstände, kumulierte Zeitarten, summarische Informationen über Abteilungen, Kostenstellen etc., Statistiken.

- **Personaleinsatzplanung**[AdaB1]

  Anpassung des generellen Schichtplans (Dienstplans) an die aktuellen Gegebenheiten, Informationen über wann und wo benötigten Personalbedarf.

- **Konto**

  Der Begriff des Kontos wird in der Zeitwirtschaft ähnlich wie in der Betriebswirtschaft verwendet. Allerdings stellen die Konten nicht die Aktiva und Passiva des Betriebs dar, sondern verschiedene Aspekte der Arbeitskraft der Mitarbeiter. Übliche Konten sind die abgeleistete Arbeitszeit oder der restliche Jahresurlaubsanspruch. Eine der Interflex-Datenbanken [IFd01] definiert Konto folgendermaßen: „'Speicher' für Stunden, Tage, Einsätze einer Einsatzart oder einer Einsatzarten-Kategorie".

- **Buchung**

  Eine Buchung ist eine i.d.R. vom Mitarbeiter vorgenommene Änderung des Kontostandes. In der Praxis können verschiedene Buchungsarten vorkommen, die dann den aktuellen Zustand des Mitarbeiters (z.B. an- oder abwesend, auf Dienstgang, beim Arzt, krank, Urlaub) und verschiedene Konten beeinflussen. Folgende Buchungsarten sind verbreitet: Kommen, Gehen, Zutritt, Dienstgang Anfang und Ende, Kostenstelle ändern, Projekt ändern.

- **Zutritt**

  Zeitwirtschaft und Zutrittskontrolle sind oft eng miteinander verbunden. Da nicht jeder Wechsel eines Raums oder Gebäudes gleichzeitig für die Zeitwirtschaft von Interesse ist, ist es in den entsprechenden Systemen vorgesehen, Buchungen vorzunehmen, bei denen nur die Berechtigung zum Zutritt geprüft und im Erfolgsfall die Tür geöffnet wird.

- **Terminal**

  Buchungen werden von den Mitarbeitern an sog. Terminals vorgenommen. Dies sind elektronische Geräte (oft auf Mikrocontrollern basierend), auf welchen die Buchungen durchgeführt werden. Diese werden dann über Netzwerke an den Zeiterfassungsserver weitergegeben; ist dieser oder die Netzwerkverbindung nicht verfügbar, werden die Buchungen solange auf dem Terminal zwischengespeichert. Das Zeiterfassungssystem der Firma Interflex kennt darüber hinaus auch die Möglichkeit, aus den Terminals alte bereits an den Server abgegebene Buchungen erneut abzufragen. Diese werden im Rahmen der Systemkapazität auf den Terminals extra für diesen Fall gespeichert. Mit diesem als „alte Buchungen abholen“ oder ABU bezeichneten Verfahren läßt sich sicherstellen, daß im Falle eines Totalausfalls des Servers die Daten seit der letzten Datensicherung nicht verloren sind.

- **Fehlgründe**

  Im Prinzip werden bei der Zeitwirtschaft zwei Zustände für einen Mitarbeiter unterschieden: Anwesenheit und Abwesenheit.

  Bei Abwesenheit gibt es neben der normalen Freizeit verschiedenste Gründe, nicht im Betrieb zu sein; dabei wird unterschieden, ob diese Zeiten bezahlt sind oder nicht. Diese Gründe werden als Fehlgründe bezeichnet.

  Nach **[AdaB1]** gibt es die folgenden sieben Gruppen von Fehlgründen:

• Urlaub:	Tarif-, Schwerbehinderten-, Bildungs-, und Erziehungsurlaub, Sonderurlaub aus verschiedenen Gründen
• Arbeitsunfähigkeit:	Krank ohne Bescheinigung bis 3 Tage, Krank, Krank ohne Lohnfortzahlung, Unfall (Arbeits- oder Wegeunfall, privat), Kur, Arztbesuch

- Tarifliche und gesetzliche Freistellung: Eigene Eheschließung oder Heirat der Kinder, Tod der Eltern, Ehegatten oder Kinder, Geburt des eigenen Kindes, Silberne oder Goldene Hochzeit, Umzug, Wehrübung, Grundwehr- oder Zivildienst, Mutterschutz, Vorladung zu einer Behörde, Wahrnehmung öffentlicher Ehrenämter, Arbeitsjubiläum
- Betriebsbedingte Abwesenheiten: Dienstreise, Dienstgang, Betriebs- bzw. Personalratstätigkeit
- Aus- und Fortbildung: Berufsschule, interne und externe Seminare und Lehrgänge
- Arbeitsversäumnis: unentschuldigtes Fehlen
- Freizeitausgleich: Saldoabbau Gleitzeit, Gleitzeit ganzer oder halber Tag, Freizeitausgleich aus Überstunden, Freizeitausgleich aus betrieblicher Mehrarbeit, Brükkentage, sonstige Arbeitszeitverkürzungstage

- **Saldo**

  Im Bereich der Zeitwirtschaft bezeichnet Saldo die Differenz zwischen der zu leistenden und der geleisteten Arbeit. Es ist üblich, ihn in einem eigenen Konto zu verwalten.

5 Konfigurierbare Clients im Allgemeinen

5.1 Analyse

Folgende Anforderungen lassen sich für Applikationen, die als konfigurierbare Clients verwendet werden, erkennen.
Benötigt werden Funktionen

- zur Darstellung einer Oberfläche einschließlich der Reaktion auf vorgenommene Eingaben durch den Benutzer,
- zur Durchführung von Berechnungen und Manipulationen auf den zugrundeliegenden Daten,
- zur Kommunikation mit externen parametrierbaren Schnittstellen,
- zur internen Verwaltung der Applikation selbst.

Speziell bei der Oberfläche ist zu berücksichtigen, daß die Eingabeelemente den Erfordernissen eines Touchscreen-Bildschirms angemessen sein müssen. Beispielswiese für die Eingabe von Text wird hier eine Bildschirmtastatur benötigt.
Bei der Kommunikation mit externen Schnittstellen ist zu berücksichtigen, inwieweit ein Ausfall des Systems, zu dem man verbunden ist, berücksichtigt und abgefangen werden sollte.
Ziel war es, einen ersten, gut dokumentierten Prototyp zu erstellen, der dann zu einem lauffähigen Produkt weiterentwickelt werden kann.

5.2 Mögliche Modellierungen

Bei der Modellierung lassen sich zwei Gebiete voneinander unterscheiden:

- die Logik, d.h. die Abläufe, die der Applikation zugrunde liegen,
- die Oberfläche, d.h. den Teil des Programms, den der Benutzer sieht.

Beide Gebiete müssen konfiguriert werden, um den Erfordernissen gerecht zu werden. Für die Art und Weise, wie die Konfiguration vorgenommen werden könnte, wurden in dieser Arbeit verschiedene Ansätze untersucht.

5.2.1 Logik

5.2.1.1 bestehende Skriptsprache

Die Konfiguration der Applikation geschieht dadurch, daß in einer bestehenden Skriptsprache das Notwendige programmiert wird. Nur die Schnittstellen zum Serversystem oder zur Oberfläche werden über vorhandene oder zu diesem Zweck neu zu programmierende Bibliotheken angesprochen. Als Skriptsprachen kämen z. B. Tcl/Tk [Ref01], Perl[Ref02] oder Python[Ref03] in Frage.

Von einer Realisierung dieses Ansatzes wurde abgesehen, da dieses Vorgehen im Prinzip dem harten Kodieren des gesamten Programms entspricht. Gerade bei standardisierten Aufgaben wäre der Aufwand bei der Programmierung gegenüber der höheren Flexibilität nicht gerechtfertigt. Es wäre gegenüber dem Status quo (direktes Ausprogrammieren von Fall zu Fall) keine Verbesserung.

5.2.1.2 eigene Skriptsprache

Eine Alternative zu vorhandenen Skriptsprachen stellt die Entwicklung einer eigenen Sprache dar. Mit ihr müßte das Ansprechen der Schnittstellen sowie die Darstellung der Oberfläche und die Reaktion auf Eingaben realisiert werden können. Heutzutage sind jedoch graphische Benutzerschnittstellen (wie Windows oder X11) praktisch alle ereignisorientiert. Die Behandlung mehrerer asynchroner Eingabegeräte durch abwechselndes Abfragen aller Geräte ergibt unübersichtlichen Code. Eine eigene Skriptsprache müßte also entweder die Ereignisorientierung in einen unübersichtlichen sequentiellen Ablauf zwängen oder das asynchrone Ereignismodell abbilden. Im Prinzip müßte der Interpreter für diese Sprache vom Sprachumfang her beinahe so mächtig wie vorhandene Sprachen – beispielsweise VBScript oder Perl – sein. Eine solche Entwicklung würde den Rahmen dessen, was erreicht werden soll, bei weitem sprengen. Interflex verwendet eine eigene Skriptsprache namens PLI für die Aufbereitung von Buchungen. Sie kann bei dieser Fragestellung nicht verwendet werden, da sie weder Oberflächendarstellung noch Kommunikation über ein Netzwerk unterstützt.

5.2.1.3 Konfigurationsdateien

Ein weiterer Ansatz ist die Verwendung von Konfigurationsdateien, die es ermöglichen, aus einem Satz vorhandener Funktionen (die dann auch die externen Schnittstellen ansprechen) einen Ablauf zusammenzustellen. Die Darstellung von Abläufen durch eine Abfolge von Aktionsblöcken und ein paar Steueranweisungen wie Schleifen oder Verzweigungen wird nicht nur in der strukturierten Programmierung verwendet, sondern ist so intuitiv, daß ein dänischer Hersteller von Plastikspielzeugen seine Elektronik auf diese Art und Weise programmieren läßt.[Unb05]

Oft ist bei Aufgabestellungen im Bereich der Dateneingabe ein gewisser Ablauf (z. B. in Form eines Diagramms) vorgegeben. Je nach Eingaben wird dann auf weitere Teilabläufe verzweigt. Eine solche Konfiguration ist bei Interflex schon im Bereich der Betriebsdatenerfassung für die Modellierung von Abläufen im Einsatz. Sie ist auf einfachere Hardware ausgerichtet und läßt sich in diesem Zusammenhang nicht einsetzen, da sie auf die Hardware des verwendeten Controllers direkt zugeschnitten ist.
Diesem Ansatz wurde der Vorzug gegeben, da er eine verhältnismäßig einfache Konfiguration des Systems erlaubt und auch mit angemessenem Aufwand realisiert werden kann.

5.2.1.4 von der Oberfläche ausgehend/Browser-basiert (Skripting)

Mittels Webtechnik ließ sich dieses Problem auch angehen. Dabei würde die Logik auf dem eigentlichen Server angesiedelt und dort über server-seitiges Skripting eine Oberfläche erzeugt. Diese würde dann als HTML-Datei an den Client geschickt.
Nachteil dieser Lösung ist, daß nur auf die Oberfläche reagiert werden könnte. Auf externe Schnittstellen könnte nicht reagiert werden. Der Anschluß eines Gerätes (z. B. eines Ausweislesers) wird dadurch erschwert, daß dieser am Client-Rechner, bei dem der Benutzer steht, angeschlossen ist und nicht am Server. In diesem Fall müßte der Client selbst wieder Code ausführen, der die Kommunikation mit dem Ausweisleser gewährleistet.[1] Ein Standard-Browser ist technisch dazu in der Lage. Allerdings läßt er sich nicht so einschränken, daß Zugriffe des Benutzers auf das normale System verhindert werden können. Die Entwicklung eines eigenen Browser, der diese Probleme löst, übersteigt die Möglichkeiten eines solchen Projekts.

5.2.2 Oberfläche

5.2.2.1 Skriptsprache

Bei der Konfiguration der Oberfläche über eine Skriptsprache wird man mit dem bereits in 5.2.1.2 beschriebenen Problem konfrontiert, daß eine Skriptsprache keine Ereignisorientierung zuläßt, sondern immer nur eine Eingabe nach der anderen auswerten kann. Auf asynchrone Eingabegeräte ließe sich nicht reagieren. Würde eine Skriptsprache mit Ereignissteuerung und Nebenläufigkeit entwickelt, würde dies für die Konfiguration eines System einen derart hohen Programmieraufwand bedeuten, der nicht mehr weit von der generischen Entwicklung des Programms mit einer normalen Sprache und deren „harten“ Kodierung entfernt ist. Dies rechtfertigt den Aufwand nicht.

[1] in der Internetprogrammierung unter dem Begriff Client-seitiges Scripting bekannt

5.2.2.2 Das Modell Visual Basic

Eine ungewöhnliche Idee war es, die Entwicklungsumgebung von Visual Basic zu verwenden. Die Oberfläche würde ganz normal mit Hilfe der Entwicklungsumgebung zusammengestellt, allerdings würde der Codeteil nicht in Visual Basic geschrieben, sondern in der gewählten Skriptsprache. Das Ergebnis könnte dann von einer eigenen Applikation ausgewertet werden, da die frm-Dateien von Visual Basic ein definiertes Format haben. Ein Nachteil dieser Variante ist jedoch, daß ein normaler Mensch, der dieses System konfiguriert und der es z. B. gewohnt ist die Zeiterfassungsapplikation 6020 über deren VBA-Schnittstelle anzusprechen, von der Mischung der Sprachen stark verwirrt würde.

5.2.2.3 Konfigurationsdateien

Konfigurationsdateien (z. B. in XML) bieten eine gute Möglichkeit, die Oberfläche zu beschreiben. Die einzelnen Steuerelemente (oder in der X-Server-Sprache: Widgets) könnten über eigene Knoten abgebildet werden und mit Attributen versehen werden. Diese lassen sich dann flexibel auf dem Bildschirm anordnen. Die Reaktionen auf Änderungen lassen sich gleich mit angeben. Ein Nachteil ist, daß die Konfiguration von Verzweigungen u.ä. nicht so übersichtlich wird.

5.2.2.4 HTML

Die Entwicklung von Oberflächen in HTML ist ein stark verbreitetes Verfahren. Clientseitig läßt sich in der Regel auf Standardhilfsmittel zurückgreifen. Bei Oberflächen für Touchscreens muß durch geeignete Programmierung dafür Sorge getragen werden, daß eine Bedienung mittels Fingerdruck möglich ist. Ein größeres Problem stellt es dar, daß auf dem Client keine vernünftigen oder nur proprietäre Möglichkeiten zur Ansteuerung lokaler Schnittstellen (z. B. Ausweisleser an serieller Schnittstelle) bietet, da HTML per se kein Ausführen von Code vorsieht und über Client-seitiges Scripting kein portabler Code für verschiedene Plattformen möglich ist.

5.2.3 Bestehende fertige Modelle

5.2.3.1 Model-View-Controller

Im Zuge von Smalltalk-80™ wurde von Adele Goldberg und anderen ein Framework namens Model-View-Controller[GoIA1] [GoIA2] entwickelt.

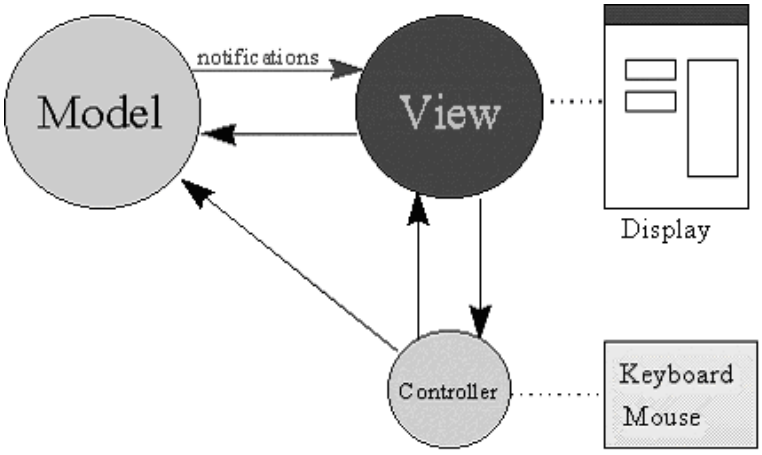

ABBILDUNG 1: MODEL-VIEW-CONTROLLER-MODELL[UNB01]

Es wurde allgemein für das Erstellen von Programmen mit graphischer Oberfläche entwickelt und nicht speziell für konfigurierbare Applikationen auf Touchscreens. Grundsatz dieses Modells ist die Trennung in die Darstellung der Daten „view“, die Interaktion des Benutzers „controller“ und den internen Zustand der Applikation „model“.

5.2.3.2 Model-View-Presenter

Das Model-View-Controller-Modell wurde in den folgenden Jahren weiterentwickelt.

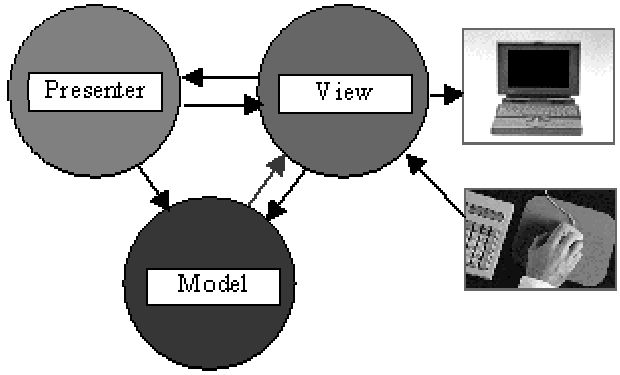

ABBILDUNG 2: MODEL-VIEW-PRESENTER-MODELL[UNB02]

Der Controller und die View sind in der Praxis oft verbunden (vgl. Java Swing). Daher wurden Controller und Model zum Presenter zusammengefaßt. [Unb02] Die View stellt die Daten des Models dar. Der Presenter paßt die Daten im Model aufgrund von Benutzereingaben an.

5.2.4 Entscheidungsfindung

In der Praxis lassen sich Oberfläche und Logik nur schwer trennen. Derjenige, der das System konfiguriert, weiß welche Abläufe es gibt und welche Oberflächenelemente welche Aktionen auslösen sollen. Dies gilt es auch in der Konfigurierung des Systems abzubilden. Die beiden oben genannte Modelle vermögen dies leider nur eingeschränkt, da sie auf die Programmierung im Allgemeinen und nicht auf die Konfiguration von Abläufen ausgelegt sind.

Darüber hinaus sollten folgende Kriterien erfüllt sein: Die Schnittstellen sollten klar definiert sein; von einer „richtigen“ Programmierung sollte zugunsten einer flexiblen Konfiguration abgesehen werden. Andererseits müssen sie so flexibel sein, daß die Funktionalität des Systems dahinter auch möglichst weitgehend ausgenutzt werden kann.

5.3 Das Interface-Action-Config Modell

Zur Bewältigung all dieser eben genannter Abläufe habe ich die Modelle Model-View-Controller und Model-View-Presenter weiterentwickelt. Die reine Aufteilung in die verschiedenen Module ist noch um einen Ablauf der einzelnen Aktionen ergänzt. Es erhielt den Namen „Interface-Action-Config“-Modell.

5.3.1 Vorstellung des Modells

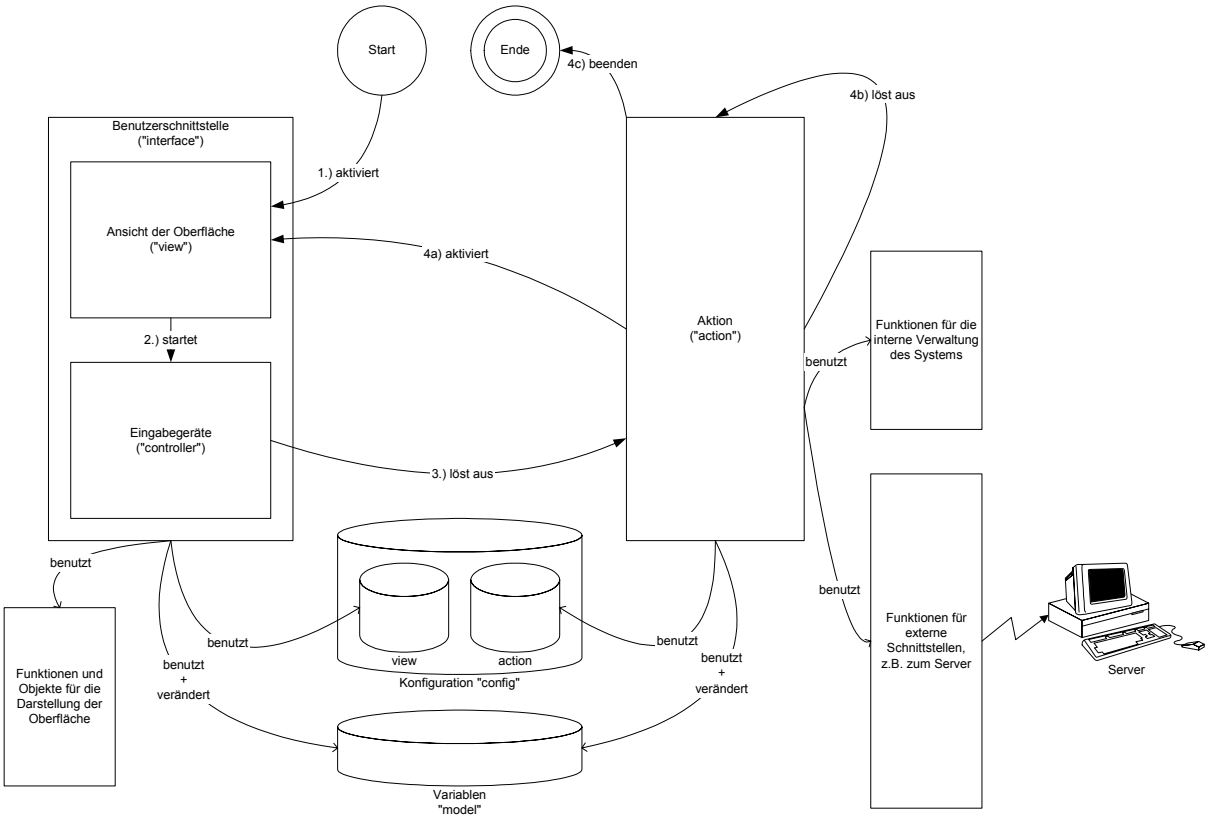

ABBILDUNG 3: INTERFACE-ACTION-CONFIG-MODELL

Mit Interface-Action-Config entwickelte Programme sind in zwei Module aufgeteilt.

Das Interface, welches die Oberflächenansicht „view“ und die Eingabe „controller“ einschließt, regelt die Darstellung zum Benutzer hin und die Verarbeitung seiner Eingaben.

Die Aktionen „action“ sind von der Oberfläche losgelöste Reaktionen auf Ereignisse, die entweder vom Benutzer über die GUI oder über externe Schnittstellen ausgelöst werden.

Eine globale Konfiguration „config“ enthält zweierlei Beschreibungen:

- die Beschreibung, wie die Oberfläche aussieht und wie auf Eingaben reagiert werden soll,

- die Beschreibung, welche Aktionen durchgeführt werden können und wie sie heißen.

Diese beiden Module benötigen jeweils eine gemeinsame Datenbasis, die sie manipulieren können. Diese wird zur Laufzeit ähnlich globalen Variablen bzw. Objekten in einer klassischen Programmiersprache angesprochen. Die einzelnen Daten der Datenbasis werden hier als Variablen bezeichnet.

Beide Module „action“ und „view“ manipulieren die in den Variablen gespeicherten Daten konkurrierend. So kann ein einer Ansicht zugehöriges Steuerelement eine Variable ändern, deren neuer Wert später von einer Aktion verarbeitet wird.

Schließlich kommen noch Schnittstellen dazu, welche Funktionen für die Aktionen bzw. Views zur Verfügung stellen, nämlich beispielsweise die GUI-Bibliothek oder die Schnittstelle zum Server-System. Diese sind dann nur indirekt über View oder Actions konfigurierbar.

5.3.2 Ablauf am Beispiel Zeiterfassung

Nun wird am Beispiel Zeiterfassung erläutert, wie ein solcher Ablauf in der Praxis aussehen könnte:

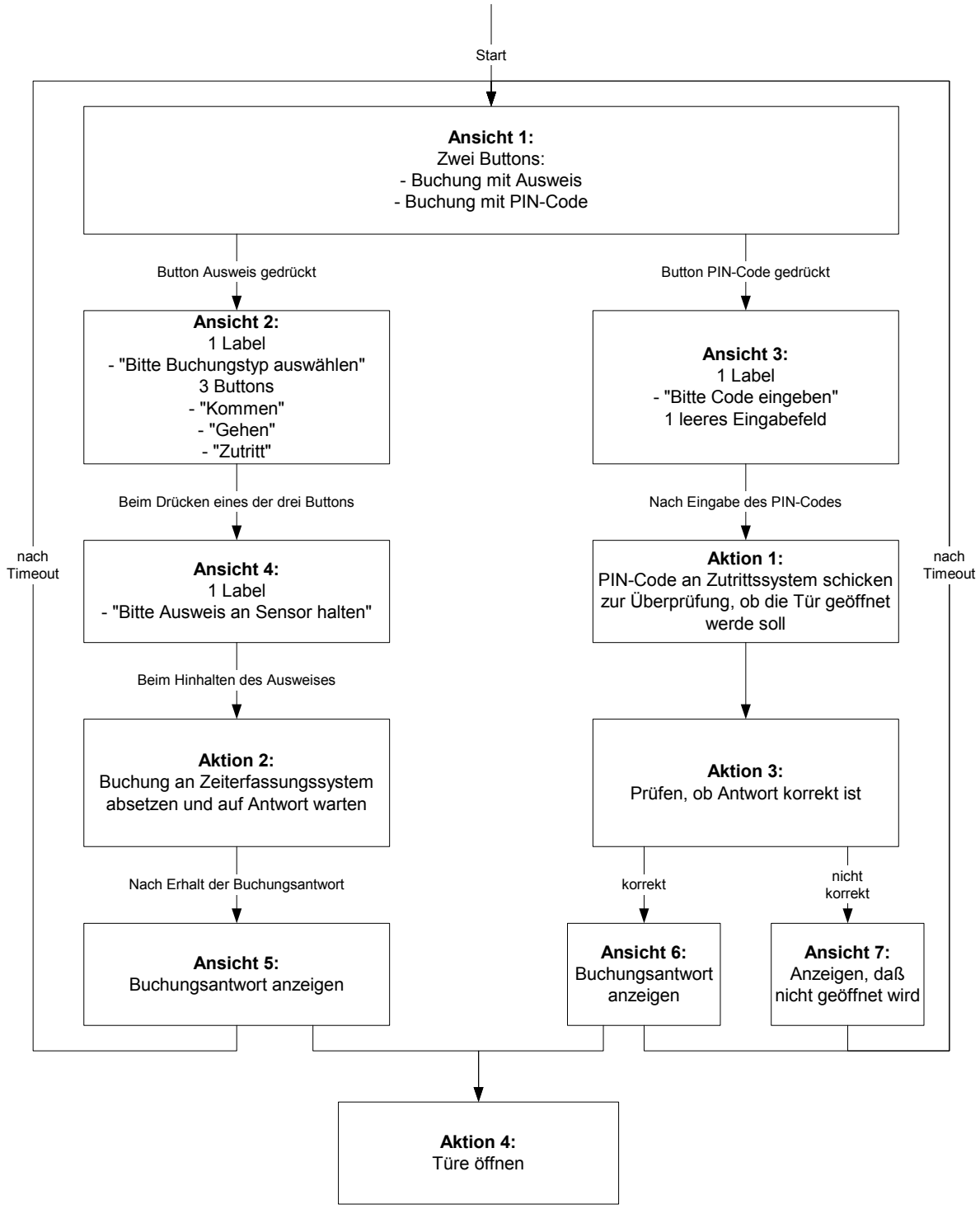

ABBILDUNG 4: BEISPIEL FÜR DERARTIGE ABLÄUFE

Hier läßt sich erkennen, wie aus Ansichten Ereignisse verarbeitet werden, die wiederum Ansichten anzeigen oder Aktionen auslösen. Letztere können dann ihrerseits weitere Aktionen oder Ansichten aktivieren.

5.3.3 Protokollierung

Ziel konfigurierbarer Applikationen ist es, dem Benutzer einen eingeschränkte Zugriff auf die Funktionalität des zugehörigen Serversystems zu geben. Im Fehlerfall ist es wichtig, nachvollziehen zu können, was der Benutzer in welcher Reihenfolge gemacht hat und welches Resultat er damit erzielt hat. So können bei Problemen notfalls auch manuell die Daten auf dem Serversystem nachgeführt werden. Es ist in den meisten Fällen sinnvoll, den Benutzer darüber zu informieren, daß die Verbindung zum Serversystem unterbrochen ist. Allerdings kann und soll er dieses Problem nicht selbst beheben. Dies sollte vom Administrator vorgenommen werden, der dann die Informationen aus dem Logfile benötigt, um die Ursache herauszufinden.
Es ist nicht damit getan, alle möglichen Abläufe einfach sequentiell in das Logfile zu schreiben. Eine Klassifizierung erleichtert die spätere Auswertung der Datei.
Folgende Klassen von Nachrichten bieten sich an:

Fataler Fehler:	Es ist schwerer Fehler aufgetreten, der einen weiteren Programmablauf unmöglich macht.
Fehler:	Es ist ein Fehler aufgetreten, bei dem eine gewünschte Funktion nicht ordnungsgemäß ausgeführt wurde, ein Wieterführen des Programmablaufs ist möglich
Warnung:	- Argumente wurden falsch gesetzt, die ignoriert werden; - Das Programm kann auf diese Art und Weise ausgeführt werden, es erscheint aber nicht sinnvoll; - Besondere Betriebsmodi werden dokumentiert, um sie besser im Logfile zu erkennen.
Information:	Alle wichtigen Abläufe sind erkennbar
Detaillierte Information:	Alle Teilschritte sind erkennbar.
Trace	Alle einzelnen Programmteile und deren Zustand sind nachvollziehbar.

5.3.4 Debugging

Im Interface-Action-Config-Modell werden Aktionen durchgeführt und Ansichten aktiviert. Beim Durchführen einer Aktion oder Wechseln einer Ansicht können sich die Inhalte der Variablen ändern.
Der Test der neu erstellten Konfigurationen könnte durch eine Art Einzelschrittmodus erleichtert werden. Bei diesem wird vor und nach jeder Aktion bzw. jedem Ansichtswechsel ein Fenster angezeigt, mit dessen Hilfe man die Inhalte der einzelnen Variablen lesen und editieren kann. Außerdem kann bestimmt werden, ob die folgende Ansicht oder Aktion übersprungen werden soll.

5.3.5 Problematik der Synchronisation

Bei der Übertragung dieses Modells in reale Programme sollte noch folgendes beachtet werden: Die X-Server-Bibliotheken unter Linux sind i.d.R. nicht thread-safe. Aus diesem Grund muß die Synchronisation zwischen den Threads von Hand erfolgen. Es muß sichergestellt sein, daß immer nur von einem Thread auf einmal auf den X-Server zugegriffen wird. Dies läßt sich mit einem globalen kritischen Abschnitt (vgl. 7.3) realisieren, der z.B. mit Hilfe eines Mutex realisiert werden kann.

5.4 Allgemeine Lösungsansätze

5.4.1 Bildschirmtastatur

Zur Absicherung des Systems ist es von Vorteil, keine normale Tastatur zuzulassen, da mit Hilfe dieser betriebssystemspezifische Tastenkombinationen gedrückt werden könnten, die man dann im System abschalten müßte. Es macht aber Sinn, alleine zu Wartungszwecken eine normale Tastatur anschließen. Aus diesem Grund sollte für die Eingabe von Text eine Tastatur dargestellt werden, die dann über den Touchscreen bedient wird.

Da die verschiedenen Eingabefelder ganz unterschiedliche Anforderungen an die benötigten Tasten stellen und verschiedene Sprachen unterschiedliche Zeichen verwenden, macht es wenig Sinn, nur eine vorgefertigte Universaltastatur zu verwenden. Daher sollte vorgesehen werden, daß verschiedene Layouts konfiguriert werden können, die dann den einzelnen Eingabefeldern zugewiesen werden können. Sinnvolle Bestandteile eines Layouts sind Position, Größe und Aussehen des Eingabefelds und der Tasten, sowie die Wirkung der Tasten (Eingabe eines Zeichens oder Sonderaktion wie Löschen oder Eingabe beenden). Desweiteren sollten noch Position, Größe und Aussehen eines Feldes für einen Eingabetext sowie einen Kommentartext, ob die Eingabe formal richtig ist, konfiguriert werden können.

5.4.2 Oberflächenkonfigurierung

Unter fast allen Bibliotheken für die Erstellungen graphischer Oberflächen ist es üblich, die GUI in verschiedene Objekte zu unterteilen, die je nach Betriebssystem als Controls, Steuerelemente oder Widgets bezeichnet werden. Dies gilt z.B. für die folgenden Systeme: Unix Gtk, Windows API, Borland TurboVision, QT, u.v.a.m..

Dieses Konzept sollte daher auch für die Konfigurierung einer Ansicht („View“) im Interface-Action-Config-Modell verwendet werden.

Es werden verschiedene Steuerelemente vorgesehen, die dann innerhalb der View positioniert werden können. Bei der Implementierung macht es Sinn, diese in Klassen zu kapseln, die von einer eigenen Steuerelementbasisklasse abgeleitet sind.

Bei der Implementierung dieser Klassen können – soweit vorhanden – dann die Steuerelemente der GUI-Bibliothek verwendet werden. Die Konfiguration beschreibt dann die einer View zugeordneten Steuerelemente und deren Attribute. Als Attribute kommt folgendes in Frage: Position auf dem Bildschirm, Größe, Farbgebung, Schriftart und -größe, Formatprüfungen und Wertebereiche. Dazu kommt noch die Definition der Aktionen, die bei Benutzerinteraktion ausgelöst werden.

5.4.3 Offline-Fähigkeiten

Man kann sich nie wirklich darauf verlassen, daß Netzwerkverbindungen immer bestehen oder daß das Serversystem zu jedem Zeitpunkt läuft.
Daher muß ein Verhalten für den Fall berücksichtigt werden, in dem kein Server zur Verfügung steht. Was das richtige Verhalten in diesem Fall ist, hängt stark vom konkreten System ab und wird daher in der Schnittstelle zum Serversystem behandelt werden.
Ein Zeiterfassungssystem würde beispielsweise in diesem Fall die vorgenommenen Buchungen puffern und später abliefern, während eine Maschinensteuerung in diesem Fall nur anzeigen zu bräuchte, daß zu diesem Zeitpunkt keine Bedienung möglich sei.

6 Übertragung auf das Beispiel Zeiterfassungssystem

Die bei der allgemeinen Betrachtung gewonnenen Erkenntnisse wurden bei der Applikation IfSoftterm eingebracht und angewandt.

6.1 Analyse

6.1.1 Anforderungen

Es sollte eine Applikation geschrieben werden, mit deren Hilfe die Funktionalität bestehender Terminals abgedeckt werden kann. Dabei handelt es sich i.w. um das Absetzen von Buchungen wie Kommen, Gehen oder Arztbesuch.
Als Zielplattform war ein Embedded PC mit Linux vorgesehen, da bekanntlich keine Lizenzkosten entstehen sollten, die das Produkt nicht wettbewerbsfähig gemacht hätten.
Zusätzlich zu den Funktionen bestehender Terminals sollte die Verwendung der Programmierschnittstelle des System 6020 vorgesehen werden, mit deren Hilfe weitere Aktionen wie Auswertung oder Anträge ermöglicht würden.
Die Benutzerschnittstelle und die verschiedenen gewünschten Abläufe sollten möglichst flexibel konfiguriert werden können. Weiterhin werden neben den bereits genannten Schnittstellen noch eine Schnittstelle zum Aufruf von anderen Programmen sowie eine Schnittstelle zu Ausweis- oder Barcodelesern benötigt.

6.2 Benutzerschnittstelle

6.2.1 Bibliotheken

Im Rahmen einer zügigen Entwicklung und um gut wartbare Software zu erstellen ist es sinnvoll, die Oberfläche mit einer geeigneten Bibliothek zu entwickeln.
Aus Gründen der Portabilität wurde für die Oberfläche das X11-System verwendet. Die SVGA-Bibliothek unter Linux, die ohne den Overhead eines Windowmanagers auskommt, wird nicht verwendet, da diese sich nicht automatisch auf ungewöhnliche Auflösungen einstellen läßt, mit denen bei Touchscreens gerechnet werden muß.
Für die Programmierung der Oberfläche wurden die beiden gebräuchlichsten Bibliotheken, das Gtk+-Toolkit[Ref08] der Gtk+-Gruppe und Qt[Ref09] der Firma Trolltech, untersucht.

6.2.1.1 Gtk

Das Gtk+-Toolkit bietet die wichtigsten Standardsteuerelemente vorgefertigt an. Es bietet Möglichkeiten zur Ereignisbehandlung und zum dynamischen Erzeugen von Widgets. Es unterliegt der GNU public licence und kann auch kommerziell ohne Lizenzgebühren eingesetzt werden.

6.2.1.2 Qt

Die Qt-Bibliotheken der Firma Trolltech haben einen höheren Umfang an vorhandenen Steuerelementen. Sie können nur zu nicht-kommerziellen Zwecken frei verwendet werden.

6.2.1.3 Wahl der Bibliothek

Für die Verwendung mit Touchscreens ist nur ein Teil der Steuerelemente geeignet. Es macht beispielsweise keinen Sinn, einen Drehknopf oder das Doppelklicken zu verwenden. Außerdem kann immer nur eine Untermenge der von der Bibliothek zur Verfügung gestellten Elemente verwendet werden, da jedes auch einzeln in die Interface-Action-Config-Umgebung eingebettet werden muß. Da die Gtk-Bibliothek alle notwendigen Widgets zur Verfügung stellt und darüber hinaus kostenfrei ist, wurde ihr der Vorzug gegeben.

6.2.2 Verschiedene Steuerelemente

Die folgende Tabelle zeigt, welche IfSoftterm-Steuerelemente es gibt und mit Hilfe welcher Gtk-Widgets sie realisiert wurden.

IfSoftterm Steuerelement	Gtk-Widget
label	GtkLabel
checkbox	GtkCheckButton
radiobutton	GtkRadioButton
button	GtkButton
inputfield	GtkEntry, extra Fenster für Eingabe
table	pro Feld ein GtkEntry, extra Fenster für Eingabe
listbox	GtkCombo
progressbar	GtkProgressBar
datetime	GtkLabel
image	GtkImage
helpbutton	GtkButton
cancelbutton	GtkButton

TABELLE 1: ZUORDNUNG STEUERELEMENTE ZU GTK-WIDGETS

Es wurden also die Gtk-Standard-Widgets in Klassen gekapselt und um die Informationen und das Verhalten so erweitert, daß sie im Interface-Action-Config-Modell verwendet werden können.

6.3 Aktionen

6.3.1 Synchronisation

Die Aktionen werden grundsätzlich sequentiell nacheinander ausgeführt, um Seiteneinflüsse zu vermeiden. Zu diesem Zweck ist die Ausführung der Aktionen in die Hauptereignisschleife eingebettet. Die Verarbeitung der eigentlichen Oberfläche ist aber höher priorisiert, damit der Benutzer nicht den Eindruck bekommt, daß das System nicht mehr reagiert. Aktionen können ebenso wie Ansichten von jeder Stelle im Programmablauf ausgelöst werden. Sie werden dann allerdings in einer Warteschlange aufbewahrt, damit immer nur eine Aktion gleichzeitig ausgeführt werden kann.

6.3.2 Vorhandene Aktionstypen

Die folgenden Arten von Aktionen sind im IfSoftterm implementiert:

- Verzweigung in Abhängkeit einer Variablen
- Aufruf externer Programm mit Übergabe von Daten an das Programm über die Standardein- und –ausgabe
- Absetzen von Buchungen über eine Fremdterminalschnittstelle des System 6020
- Aufruf von beliebigen Funktionen aus dynamischen Bibliotheken
- Festlegen der Reaktion auf serielle Eingaben (z.B. von Ausweislesern)
- Zusammenfassen von mehreren Variablen
- Parsen und Aufspalten einer Zeichenkette
- Einfache Arithmetik
- Anzeige einer Mitteilung (MessageBox)
- Aufruf einer Hilfe

6.4 Variablen

Wie in 5.3.1 angesprochen, benötigt das Interface eine gemeinsame Datenbasis, auf die konkurrierend zugegriffen werden kann. Sie wurde durch eine eigene Singleton-Klasse implementiert. Ein Singleton-Entwurfsmuster ist bei [GamE1] als eine Klasse definiert, die genau ein Exemplar besitzt und einen globalen Zugriffspunkt darauf bereitstellt. Es werden folgende Methoden benötigt: Ändern einer Variablen, Auslesen einer Variablen, Löschen einer Variablen und Löschen aller Variablen.

Da davon ausgegangen werden muß, daß mehrere Threads gleichzeitig auf die Datenbasis zugreifen, ist es notwendig, mittels eines kritischen Abschnitts (vgl. 7.3) sicherzustellen, daß die Zugriffe auf die Datenbasis nacheinander erfolgen.

Die Daten selbst werden in den Variablen als Zeichenketten gespeichert. Bei Aktionen, die statt dessen Zahlen benötigen, werden diese innerhalb der Aktion konvertiert. Ist eine Variable nicht definiert, wird eine leere Zeichenkette zurückgeben; leere Zeichenketten und solche, die nicht als Zahl erkannt werden, werden als Zahl 0 angenommen.

6.5 Konfiguration

XML bietet folgende Vorteile gegenüber anderen Konfigurationsmöglichkeiten: Im Gegensatz zu einem proprietären Format läßt es sich mit Standardsoftware bearbeiten. Den einfachen Textdateien hat es die Möglichkeit zur Strukturierung der Daten voraus.
Weitere Informationen zu den Möglichkeiten und der Spezifierung von XML sind z.B. bei [Ref04] verfügbar.
Eine Beispielkonfiguration für das IfSoftterm ist im Anhang enthalten.
Jedes XML-Dokument besitzt einen Wurzelknoten `<prog>`, dem die drei Knotentypen `<general>`, `<view>` und `<action>` untergeordnet sind. Diese entsprechen den drei Modulen, die dem Interface-Action-Config-Modell sein Namen gaben.

6.5.1 Allgemeine Konfiguration – Knoten <general>

In diesem Knoten werden die allgemeinen Konfigurationseinstellungen vorgenommen. Dieser Knoten darf nur einmal vorkommen.

Zunächst werden diejenigen Knoten beschrieben, die sich allgemein bei derartigen Systemen verwenden lassen:

6.5.1.1 Knoten <keyboard>

Diese Knoten stellen jeweils einen Dialog dar, der aufgerufen wird, um mit einer Bildschirmtastatur Eingaben vorzunehmen. Er enthält einen eindeutigen Bezeichner zur Identifikation und globale Dialogeinstellungen (z. B. die Hintergrundfarbe). Er erlaubt es, Tastaturen für die Eingabe frei zu konfigurieren.
Sie enthält – jeweils frei positionierbar – ein Eingabefeld, ein Label zur Beschriftung des Eingabefelds, ein Label zur Anzeige, ob die Eingabe im richtigen Format ist, sowie beliebig viele Tasten.
Die Steuerelemente <inputfield> und <table> (vgl. 6.5.2.5 bzw. 6.5.2.6) verwenden diese Eingabehilfe und besitzen ein Attribut, mit dessen Hilfe sie das gewünschte Tastaturlayout und das Format der Eingabe festlegen können.

6.5.1.1.1 Knoten <key>

Hier wird eine einzelne Taste konfiguriert. Für sie werden Oberflächeneigenschaften wie Beschriftung mit Farbe und Schriftart, Größe, Position auf dem Bildschirm eingestellt. Darüber hinaus wird festgelegt, welches Zeichen mit ihr eingeben werden soll. Alternativ dazu kann sie auch mit einer der folgenden Sonderfunktionen belegt werden: Cursor bewegen, Löschen, Eingabe abschließen, Wechsel zwischen Ebenen (z. B. Shift-Taste) u.ä.

6.5.1.1.2 Knoten <entry>

Die Attribute dieses Knotens beschreiben das Eingabefeld des Bildschirmtastatureingabedialogs. Diese sind die oben genannten Oberflächeneigenschaften.

6.5.1.1.3 Knoten <label>

Zum Eingabefeld gehört auch eine Beschriftung. Deren Oberflächeneigenschaften lassen sich hier festlegen.

6.5.1.1.4 Knoten <oklabel>

Ein weiteres Label dient zur Darstellung von Meldungen, inwiefern die Eingabe das richtige Format besitzt.

6.5.1.2 Knoten <msgbox>

Das Anzeigen von Nachrichten mittels entsprechenden Dialogen wird häufig benötigt. Hierfür ist ein extra Aktionstyp (vgl. 6.5.3.6) vorgesehen, um nicht jedes Mal eine neue Ansicht mit allen Steuerelementen von Hand erstellen zu müssen. Das Aussehen dieser Standardnachrichtenbox läßt sich hier einstellen.

6.5.1.3 Knoten <text>

An vielen Stellen sollten einfach nur bestimmte, im konkreten Anwendungsfall feste, Texte angezeigt werden; sei es bei Fehlermeldungen oder bei der Anpassung von Wochentagen und Monatsnamen an die gewünschte Sprache. All diese Texte können mit einem Schlüssel in der Konfiguration abgelegt werden und über diesen angesprochen werden.

6.5.1.4 Knoten <log>

Wichtig bei allen derartigen Systemen ist es, im nachhinein nachvollziehen zu können, welche Aktionen auf einem solchen ausgeführt wurden. Speziell bei Fehlern lassen sich daraus Rückschlüsse für die Behebung derselben ziehen und die fehlerhaft gewordenen Daten notfalls auch von Hand korrigieren. Alle Einstellungen in diesem Zusammenhang (z. B. Pfad der Logdatei oder Loglevel) lassen sich hier vornehmen.

6.5.1.5 Knoten <startview>

Im Interface-Action-Config-Modell (vgl. Abbildung 3 in 5.3.1) wird beim Programmstart erst einmal eine Grundansicht aktiviert (im Schaubild mit *1.)* markiert). Diese wird in diesem Knoten festgelegt.

Jetzt folgen diejenigen Knoten, die speziell für das IfSoftterm zum Einsatz kommen:

6.5.1.6 Knoten <device>

An dieser Stelle werden seriell anschließbare Eingabegeräte parametriert. Dazu gehören z. B. Ausweisleser oder Barcodeleser.

6.5.1.7 Knoten <ifterapi>

Dieser Knoten beschäftigt sich mit der Parametrierung der Fremdterminal- und Buchungsschnittstelle des System 6020 namens ifterapi.

6.5.2 Konfiguration der Oberfläche – Knoten <view>

Dieser Knoten enthält die notwendigen Einstellungen für eine Ansicht im Sinne des Interface-Action-Config-Modells.
Es können beliebig viele Ansichten definiert werden, die über einen eindeutigen Bezeichner identifiziert werden.
Wie in vielen gebräuchlichen Oberflächenprogrammierungssystemen[1] wird auch im IfSoftterm eine Ansicht in verschiedene Steuerelemente („controls“) aufgeteilt.
Aus diesen läßt sich dann die Oberfläche zusammensetzen. Sie speichern ihre Daten in Variablen und lösen bei bestimmten Ereignissen Aktionen aus.

[1] so auch bei Unix X11 („Widget“), Windows oder MacOS („Controls“)

Der Knoten view selbst enthält darüber hinaus gewisse eigene Parameter wie die Hintergrundfarbe oder das Festlegen einer Reaktion auf Inaktivität.

Die folgenden Steuerelemente sind implementiert. Sie lassen sich im Prinzip auch allgemein bei derartigen Applikationen einsetzen. Es wurde Wert darauf gelegt, nur Steuerelemente zu verwenden, die sich auch auf einem Touchscreen problemlos bedienen lassen. Drehknöpfe, Menüs oder Tastaturkurzkombinationen brauchen also nicht berücksichtigt zu werden.

6.5.2.1 Knoten <label>

Dabei handelt es sich um ein auf die Oberflächeneigenschaften und den Text frei konfigurierbares Beschriftungsfeld.

6.5.2.2 Knoten <checkbox>

Eine Checkbox ist ein Eingabefeld mit zwei Werten aktiv und inaktiv (ersteres durch ein Häkchen markiert), das darüber hinaus über einen Beschriftungstext verfügt.

6.5.2.3 Knoten <radiobutton>

Im Gegensatz zur Checkbox werden bei Radiobuttons Gruppen definiert, wobei in jeder Gruppe immer nur genau ein Knopf aktiv sein kann.

6.5.2.4 Knoten <button>

Dies sind die Standardschaltflächen die durch Fingerdruck bedient werden.

6.5.2.5 Knoten <inputfield>

Auf diese Art und Weise wird ein einfaches Eingabefeld dargestellt. Beim Daraufklicken wird dann der Eingabedialog mit der Tastatur aktiviert, wenn das Feld nicht schreibgeschützt ist. Die gewünschte Bildschirmtastatur und das Verhalten bei Änderungen können über ein Attribut angegeben werden.
Eine Eingabe alleine von externen seriellen Geräten (vgl. 6.5.1.6) ist ebenfalls vorgesehen.

6.5.2.6 Knoten <table>

Eine Tabelle besteht aus tabellarisch angeordneten verschiedenen Eingabefeldern sowie ggf. Labels als Spalten- bzw. Zeilenüberschrift. Spalten- bzw. Zeilenanzahl sind zur Laufzeit änderbar. Die Inhalte werden einer Variablen zugeordnet.

6.5.2.7 Knoten <listbox>

Dieses Steuerelement bietet die Möglichkeit, aus mehreren Elementen einer Liste auszuwählen. Sowohl der ausgewählte Text als auch der Index des ausgewählten Elements können gespeichert werden.

6.5.2.8 Knoten <progressbar>

Mit dieser Anzeige läßt sich ein Fortschritt anzeigen. Eingaben sind nicht vorgesehen.

6.5.2.9 Knoten <datetime>

Dieses Element entspricht von der Konfigurierung her im wesentlichen einem Label. Allerdings wird hier nicht ein freier Text dargestellt, sondern die aktuelle Uhrzeit. Das Format der Anzeige läßt sich einstellen.

6.5.2.10 Knoten <image>

Bilder und Graphiken aus Dateien lassen sich mit diesem Element darstellen. Eine Reaktion für das Daraufklicken ist ebenfalls vorgesehen.

6.5.2.11 Knoten <helpbutton> und <cancelbutton>

Bei diesen beiden Elementen handelt es sich auch nur um Buttons, allerdings mit fest vorgegebenen Reaktionen. Beim helpbutton wird der Hilfedialog aufgerufen (vgl. 6.5.3.7), beim cancelbutton die Standard-Reset-Aktion durchgeführt.

6.5.3 Konfiguration der Aktionen – Knoten <action>

Der Knoten action enthält die notwendigen Einstellungen für eine Aktion im Sinne des Interface-Action-Config-Modells.
Es können beliebig viele – über einen eindeutigen Bezeichner identifizierbare – Aktionen definiert werden.

Da oft eine ganze Liste von Aktionen benötigt wird, die hintereinander ausgeführt werden, wird die Konfiguration über viele Folgeaktionen sehr unübersichtlich. Daher ist es an allen Stellen möglich, anstelle einer einzelnen Aktion auch eine durch Komma getrennte Liste von Aktionen anzugeben, die dann in dieser Reihenfolge ausgeführt werden.
Sofern die einzelnen Aktionstypen nichts anderes vorsehen, kann für jede Aktion entweder eine Folgeansicht oder Folgeaktion festgelegt werden, die dann im Anschluß ausgeführt wird.

Die folgenden Aktionen sind implementiert:

Zuerst werden wieder die allgemein verwendbaren Aktionen beschrieben:

6.5.3.1 Knoten <switch>

Mit Hilfe dieser Aktion wird eine Verzweigung implementiert. Eine Variable ist als Testvariable vorgesehen. Mit dem Wert dieser Variablen werden dann die in den case-Unterknoten festgelegten Vergleichsoperationen abgearbeitet. Sobald ein Vergleich zutrifft, wird die im entsprechenden case-Unterknoten festgelegte Folgeaktion bzw. Folgeansicht ausgeführt. Trifft keiner der Vergleiche in den case-Knoten zu, wird die im default-Knoten festgelegte Aktion bzw. Ansicht ausgeführt.

6.5.3.2 Knoten <concat>

Anhand eines Formatierungsstrings wird aus konstanten Texten und Variablen ein Wert für eine festgelegte Zielvariable zusammengesetzt.

6.5.3.3 Knoten <scan>

Anhand eines Formatierungsstrings und von Quelldaten (i.d.R. in einer Variablen) werden letztere geparst und in verschiedene andere Variablen aufgespaltet.

6.5.3.4 Knoten <formatno>

Anhand eines Formatierungsstrings wird der als Zahl interpretierte, angegebene Wert formatiert und in eine Variable geschrieben.

6.5.3.5 Knoten <calc>

Hier kann eine mathematische Formel aus den Grundrechenarten und gewissen Standardfunktionen (z.B. Quadratwurzel, Sinus usw.) festgelegt werden; sowohl konstante Zahlen als auch Variablen sind möglich. Diese Formel wird dann bei der Ausführung der Aktion ausgewertet.

6.5.3.6 Knoten <msgbox>

Mit dieser Aktion wird eine wie in 6.5.1.2 formatierte Nachrichtenbox dargestellt. Die Meldung, die ausgegeben werden soll, kann konfiguriert werden.

6.5.3.7 Knoten <help>

Ein Hilfedialog läßt sich mit dieser Aktion darstellen. Dabei kann wahlweise eine HTML-Datei angegeben werden oder direkt eine Überschrift zuzüglich zugehöriger Text.

6.5.3.8 Knoten <shell>

Auch der Aufruf externer Shellskripte oder Binärprogramme ist möglich. Dabei kann die Kommandozeile per Attribut festgelegt werden. Außerdem können Daten festgelegt werden, die an die Standardeingabe des Kindprozesses ausgegeben werden und die Standardausgabe desselben sowie dessen Errorlevel kann in Variablen gespeichert werden.

6.5.3.9 Knoten <socall>

Mithilfe dieser Aktion werden Funktionen aus dynamischen Bibliotheken aufgerufen. Der Name der Bibliothek und der Funktion wird angegeben. Die einzelnen Parameter und der Rückgabewert lassen sich mit deren Typ festlegen. Als Typen sind Integer-Zahl und String (Pointer auf Character-Feld) möglich. Parameter und Rückgabevariable werden über die Untertags <param> und <return> festgelegt.
Die Implementierung der Schnittstelle zu dynamischen Bibliotheken ist in 6.6.4 beschrieben.

Nun folgt die speziell für das Projekt IfSoftterm vorgesehene Aktion:

6.5.3.10 Knoten <ifterapi>

Das Buchen über die als „IfMulterapi“ bezeichnete Nachfolgeversion der Interflex-Buchungsschnittstelle ifterapi wird über diese Aktion abgewickelt. Neben dem Namen der gewünschten Schnittstelle können noch Variablen für das Speichern der Buchungsantwort sowie Aktionen beim Erhalt von Buchungsantworten oder beim Timeout angegeben werden. Die Zeit, die maximal auf eine Buchungsantwort gewartet werden soll ist ebenfalls einstellbar.
Die Schnittstellenparameter sind in der globalen Konfiguration (vgl. 6.5.1.7) festgelegt.

6.5.4 Implementierung mit Hilfe von XML

Nachdem die Entscheidung zur Verwendung vom XML gefallen ist, mußte eine Bibliothek zum Parsen der XML-Konfigurationsdatei ausgewählt werden. Unter Linux ist nur eine Bibliothek weit verbreitet: Die Xerces-Bibliothek[Ref06] aus dem Apache-Projekt. Darüber hinaus ist sie im Rahmen der Apache Software Licence frei verwendbar. Aus diesen Gründen wurde der DOM-Parser aus dieser Bibliothek eingesetzt. Die Performance des Parsens spielt nur eine sehr untergeordnete Rolle, da das Parsen der Konfiguration nur einmal beim Programmstart durchgeführt wird.

6.5.4.1 Überprüfung der Struktur

Zunächst einmal wird die syntaktische Richtigkeit der XML-Datei mit Hilfe einer Document Type Definition (DTD) überprüft. Dabei handelt es sich um eine Beschreibung des syntaktisch Zulässigen in der XML-Datei. Ein Tutorial zur Verwendung von DTDs ist unter [Ref07] verfügbar. Der DOM-Parser kann optional eine Überprüfung der syntaktischen Richtigkeit anhand der DTD vornehmen und die Fehler darstellen.
Anschließend muß noch die semantische Richtigkeit überprüft werden. Dazu gehören solche Dinge wie das Vorhandensein einer Startansicht, daß konfigurierte Folgeansichten und -aktionen auch tatsächlich definiert sind oder daß ein angegebenes Tastaturlayout auch vorhanden ist.

6.5.4.2 Encoding

Es können alle Encodings verwendet werden, die vom DOM-Parser unterstützt werden. Ist keines angegeben, wird UTF-8 angenommen. Intern werden die Texte

immer in das UTF-8-Format umgewandelt, welches auch die Oberflächenbibliothek Gtk verwendet.

6.5.4.3 Speicherung der Konfiguration

Nach erfolgreichem Abschluß des Parsens werden die Informationen der XML-Datei in einer internen Konfigurationsstruktur abgebildet. Der Parser wird dann zur Laufzeit nicht mehr benötigt und kann aus dem Speicher entfernt werden. Beim Aufruf einer Aktion wird dann das zugehörige Konfigurationsobjekt verwendet, um den notwendigen Code auszuführen.

6.6 Kapselung der externen Schnittstellen

6.6.1 Schnittstelle zum Aufruf externer Programm „shell“

Eine der vielfältigsten Methoden zur Anbindung externer Systeme ist der Aufruf externer ausführbarer Programme[1]. Ist ein solcher möglich, kann man das bestehende System problemlos so erweitern, daß weitere Schnittstellen angesprochen werden können; in diesem Fall wird einfach ein zusätzliches Programm geschrieben, das die Daten aufnimmt und an die neue Schnittstelle weitergibt. Damit dies funktioniert, müssen Daten übergeben werden können. Dazu bieten sich die Standardein- und ausgabe an. In der Konfiguration wird festgelegt, welche Daten auf die Standardeingabe des Programms geschickt werden und welche von der Standardausgabe eingelesen werden. Die Implementierung dieser Schnittstelle ist nicht nur im Projekt IfSoftterm sinnvoll; sie ist unter 7.2 beschrieben.

6.6.2 Schnittstelle zur Verwendung seriell angeschlossener Geräte

Diese Schnittstelle ist speziell zum Ansprechen von Ausweislesern oder Barcodelesern, die seriell angeschlossen sind, vorgesehen. Diese Geräte schicken asynchron Daten, erwarten aber selbst keine. Die Schnittstelle läßt folgende Einstellungen zu: verwendete Kommunikationsschnittstelle, Variablen, in die die Daten gespeichert werden, auszuführende Aktionen, Protokolleinstellungen. Alternativ zum Speichern in Variablen ist auch eine Kopplung an ein Eingabefeld möglich.

---

[1] wie er in POSIX/C z. B. mit dem system-Befehl vorgenommen wird

Es ist möglich das Startzeichen „STX“ und das Endzeichen „ETX“ zu definieren, ebenso wie eine mögliche feste Satzlänge.
Implementiert wurde all dies über einen extra Thread, der blockierend von der Schnittstelle liest und dann die entsprechenden Verarbeitungsschritte durchführt.
Die Daten werden in mehreren Formaten gespeichert:: Rohdaten wie von der Schnittstelle gelesen sowie die einzelnen Protokollbestandteile wie Ausweisnummer mit Version, Kundensystemidentifikation oder einen Fehlercode. Ebenfalls kann ein Ereignis definiert werden, das beim Entfernen des Ausweises aus dem Lesebereich ausgelöst wird.

6.6.3 Schnittstelle zum Anschluß an System 6020-Buchungsschnittstelle „ifterapi“ bzw. „IfMulterapi“

Das IfSoftterm sollte alle Möglichkeiten eines normalen Terminals bereitstellen. Dazu wird die Buchungsschnittstelle zum System 6020 benötigt. Bislang stand allerdings nur unter Windows eine Schnittstelle zur Verfügung. Daher wurde zunächst eine entsprechende Bibliothek unter Unix erstellt und dann in das IfSoftterm eingebunden. Die erstellte Bibliothek ist unter 6.8.1 beschrieben. Die Parameter der Schnittstelle sind im Knoten „general“ der Konfiguration beschrieben. Die Initialisierungsroutine wird gleich zu Beginn des Programms anhand dieser Informationen aufgerufen. Danach steht eine Funktion zur Verfügung, die es ermöglicht, eine Buchung zu senden und eine definierte Zeit auf eine Buchungsantwort zu warten. Außerdem kann der Verbindungszustand abgefragt werden. Die Behandlung von nicht vorhandenen Verbindungen zum Server wird durch eine Pufferung innerhalb der Bibliothek gelöst.

6.6.4 Schnittstelle zum Aufruf von Funktionen in dynamischen Bibliotheken

Üblicherweise werden zum Aufruf von fremden Funktionen dynamische Bibliotheken verwendet. Daher erschien es prüfenswert, ob nicht auch das IfSoftterm solche aufrufen können soll.
Ein Pointer auf eine Funktion in einer dynamischen Bibliothek kann anhand der Namen der Bibliothek und der Funktion mit Hilfe der API-Funktionen `dlopen` und `dlhandle` problemlos gewonnen werden. Allerdings gibt es unter C++ zwar die Möglichkeit, Funktionen mit einer Variablen Anzahl Parameter zu schreiben (über die ANSI-C-Makros va_list etc.), allerdings ist es nicht möglich, eine beliebige Anzahl

Parameter zu übergeben. Es gäbe die Möglichkeit, die Parameter von Hand auf dem Stack zu legen; allerdings müßte dies dann in Inline-Assembler geschrieben werden. So wäre die Wartbarkeit und Portabilität des Programms nicht mehr gewährleistet. Alternativ dazu kann man auch einfach davon ausgehen, daß alle Parameter 32-bit-Werte sind und es nicht mehr als 10 Übergabeparameter gibt. Dann könnte man die Prototypen fest mit long-Parametern definieren und dann die Übergabeparameter auf long casten. Dieses Verfahren genügt für einen überwiegenden Teil der bestehenden dynamischen Bibliotheken und deckt alle Funktionen der 6020 API ab.

6.6.5 Schnittstelle zum Anschluß an die Programmierschnittstelle des System 6020 „api6020“

Da die 6020 API unter Linux als dynamische Bibliothek implementiert ist, deckt die unter 6.6.4 beschriebene Schnittstelle alle notwendige Funktionalität ab. Dies wurde mit einer geeigneten Beispielkonfiguration erfolgreich getestet.

6.7 Diverses

6.7.1 Zeichensätze und Unicode

Auch bei Beschränkung auf den europäischen und panamerikanischen Markt benötigt man verschiedenste Zeichensätze. Die üblichen Sprachen in diesem Raum werden allerdings von links nach rechts geschrieben und bestehen aus einzelnen nebeneinander geschrieben Zeichen. Allerdings reicht der alte ASCII-Zeichensatz mit 7 Bit nicht einmal für die deutsche Sprache aus. Auch mit den 8-bit-Zeichensätzen der ISO-8859-Reihe[CzyR1] läßt sich nicht alles abdecken. Schon ein Euro-Zeichen kann Probleme bereiten.

Das Gtk, die zur Darstellung auf dem X-Server verwendete Bibliothek, verwendet UTF-8 für Texte. Dabei handelt es sich um eine Codierung, die Unicode[Ref11] (UCS) in normale 8-Bit-Zeichen escaped. Dies hat den Vorteil, daß normale Strings zum Speichern der Daten verwendet werden können. Texte in 7-bit-ASCII-Codierung bleiben sogar gleich.

Eine Konvertierung von den verbreitetsten Codierungen in XML-Dokumenten nach UTF-8 ist in den Standardwerkzeugen wie der Xerces-Bibliothek[Ref06] bereits enthalten. Für die Schnittstellen nach außen kann eingestellt werden, ob die Ein- und Ausgabe in UTF-8 oder ISO-8859-1 codiert ist.

Mit Hilfe der UTF-8-Codierung ist es nun problemlos möglich, alle Sprachen, die von links nach rechts geschrieben werden abzudecken. Auf die Implementierung von Unterstützung für andere Sprachen konnte verzichtet werden, da der Kern des Systems 6020 dies genausowenig unterstützt.

6.7.2 Parsen von Strings

Da die in 6.4 angesprochenen Variablen intern immer nur Strings speichern, wurden diverse Funktion zur Manipulation und Interpretation von Strings benötigt. Zu diesem Zweck wurden die bereits erwähnten Aktionen switch, concat, scan und calc definiert. Auf das Parsen der mathematischen Ausdrücke bei der calc-Aktion wird in 6.8.2 näher eingegangen.

6.7.2.1 Zusammensetzen von Strings „concat“

Bei der Aktion „concat“ wird anhand eines Formatierungsstring aus mehreren Variablen ein neuer String gebildet. In Dollarzeichen eingeschlossener Text wird als Variablenname interpretiert; der Name einschließlich der umschließenden Dollarzeichen wird dann durch den Inhalt der Variablen ersetzt. Folgen die beiden Dollarzeichen direkt aufeinander, werden sie durch ein einfaches Dollarzeichen ersetzt.
Dabei wird entsprechend dem folgenden Automat vorgegangen:

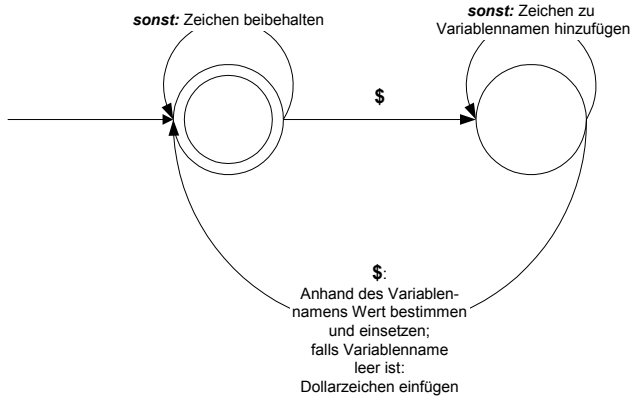

ABBILDUNG 5: AUTOMAT FÜR DAS ZUSAMMENSETZEN VON STRINGS

Es wird also Zeichen für Zeichen durchgegangen. Außer dem Dollarzeichen werden alle Zeichen in die Zielvariable übernommen. Sobald ein Dollarzeichen gefunden wird, beginnt das Zusammensetzen des Variablennamens, nach dem schließenden Dollarzeichen kommt die Auswertung der Variablen.

6.7.2.2 Auswerten von Strings „scan“

Neben dem Zusammensetzen von Strings gibt es auch die Möglichkeit einen String anhand eines Formatierungsstrings zu parsen und auf verschiedene Variablen zu verteilen. Der Formatierungsstring hat das folgende Format:

Alles außer der Formatangabe wird als konstanter Text interpretiert.

Eine Formatangabe beginnt und endet jeweils mit einem Prozentzeichen („**%**“) dazwischen kommt als erstes optional ein Bereich, dann eine Formatierungskonstante und schließlich optional ein in Dollarzeichen eingeschlossener Variablenname. Zwei direktaufeinanderfolgende Prozentzeichen werden als konstantes Prozentzeichen interpretiert. Die Bereichsangabe kann eines der folgenden Formate haben; in der Tabelle steht „**i**” für eine positive ganze Zahl, „**-**” für das Minuszeichen:

Format	Interpretation
i-i	es müssen mindestens so viele Zeichen wie bei der linken Zahl sein und dürfen höchstens so viele wie bei rechten sein.
i	es müssen genau **i** Zeichen sein
i-	es müssen mindestens **i** Zeichen sein
-i	es dürfen höchstens **i** Zeichen sien

TABELLE 2: BEREICHSANGABEN FÜR FORMATIERUNGSSTRINGS

Falls kein Bereich angegeben ist, sind beliebig viele Zeichen möglich.

Die Formatierungskonstanten bestehen aus genau einem Zeichen. Die folgende Tabelle legt die möglichen Werte dar:

Formatierungskonstante	Interpretation
i	ganze Zahl im Format `[+|-][0-9]`$^{+}$
f	Kommazahl im Format `[+|-][0-9]`$^{+}$`['.'[0-9]*]`$^{?}$
s	beliebiger String
z	String nur aus lateinischen Buchstaben `[A-Za-z]*`

TABELLE 3: MÖGLICHE FORMATIERUNGSKONSTANTEN FÜR FORMATIERUNGSSTRINGS

Zwei direkt aufeinanderfolgende Prozentzeichen werden als Konstante „**%**“ interpretiert.

Intern wird beim Parsen des zu interpretierenden Strings eine Liste von Elementen verwendet, die vom Formatierungsstringparser erzeugt wird. Jedes Element enthält entweder einen konstanten String, der beim Interpretieren des Strings verglichen wird, oder ein Formatierungselement mit Typ, Bereichsangabe und Zielvariable.

Der Automat für die Analyse des Formatierungsstrings sieht folgendermaßen aus; nicht angegebene Pfade gehen in den Fehlerzustand über:

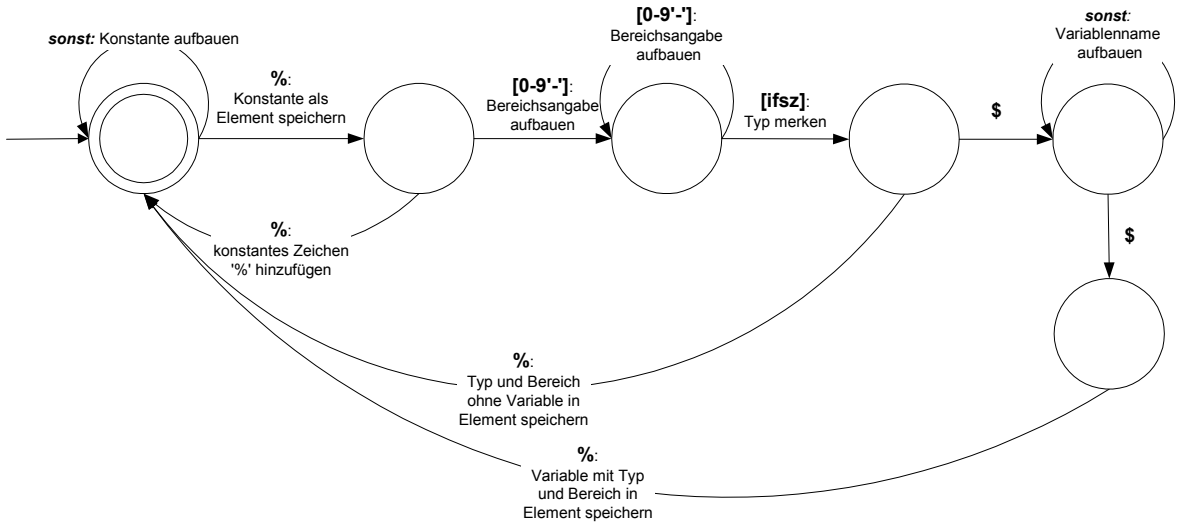

ABBILDUNG 6: AUTOMAT FÜR DAS PARSEN DES FORMATIERUNGSSTRINGS

Beim Parsen des zu interpretierenden Strings werden Automaten für die verschiedenen Arten von einzulesenden Strings benötigt. Die Anzahl der bereits abgearbeiteten Zeichen wird jeweils im Automat mitgeführt. Gibt es keine Zeichen mehr, bevor die Mindestzahl an Zeichen erreicht wird oder wird die Höchstzahl an Zeichen erreicht und der Automat ist nicht in einem Endzustand, wird dies als Fehler angesehen. Eine Besonderheit bei den beiden Automaten für das Einlesen von Strings ist, daß diese auch das folgende konstante Zeichen (wenn eine Konstante folgt) auswerten und dieses als Abbruch der Zeichenfolge interpretieren.

Nun folgen die Automaten für die einzelnen Formatierungskonstanten.

Für Zahlen werden die folgenden beiden Automaten verwendet.

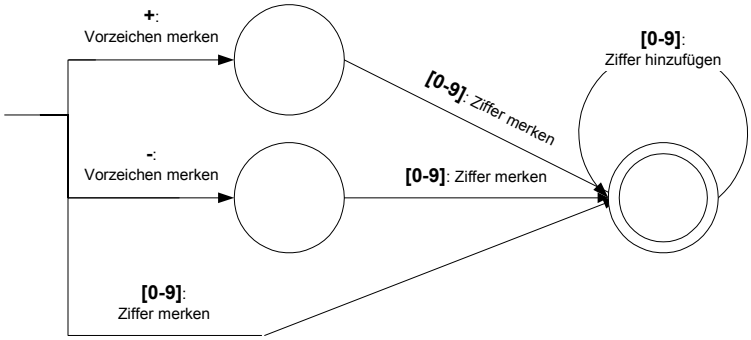

ABBILDUNG 7: AUTOMAT FÜR GANZE ZAHLEN „%i%“

Ganze Zahlen werden eingelesen, in dem zunächst ein mögliches Vorzeichen und anschließend die Ziffern gelesen werden.

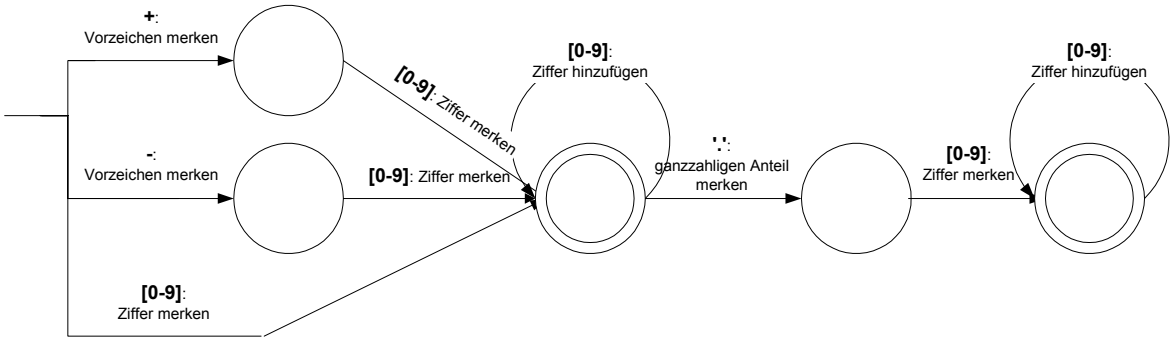

ABBILDUNG 8: AUTOMAT FÜR KOMMAZAHLEN „%f%“

Bei Kommazahlen wird ebenfalls zunächst das optionale Vorzeichen eingelesen; danach kommen die Ziffern des ganzzahligen Anteils und schließlich optional das Dezimaltrennzeichen und die Nachkommastellen.

Für Zeichenketten werden folgende Abläufe verwendet:

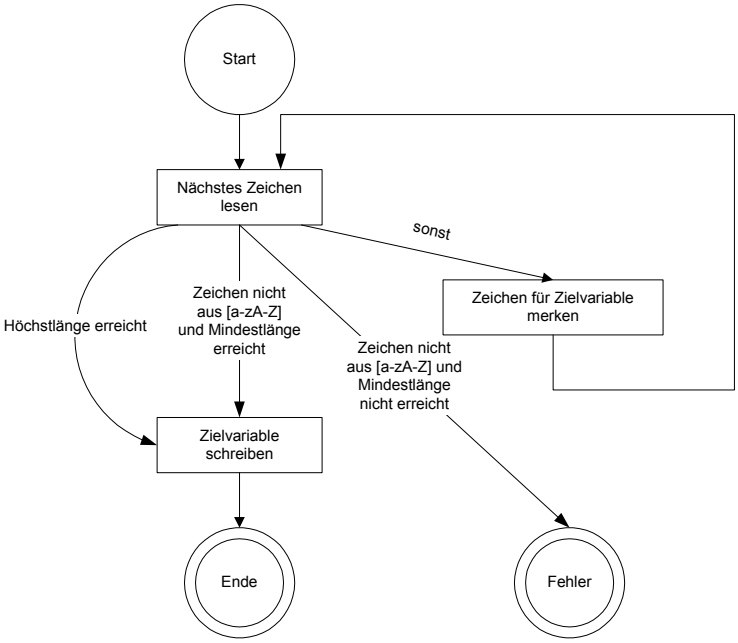

ABBILDUNG 9: ABLAUF FÜR BUCHSTABENFOLGEN „%z%“

Bei Buchstabenfolgen (Kennung „%z%“) wird jedes Zeichen eingelesen, bis entweder die Höchstlänge erreicht ist oder ein Zeichen auftritt, das $\notin$ [a-zA-z] ist. Wird ein unzulässiges Zeichen vor dem Erreichen der Mindestlänge entdeckt, ist dies ein Fehler.

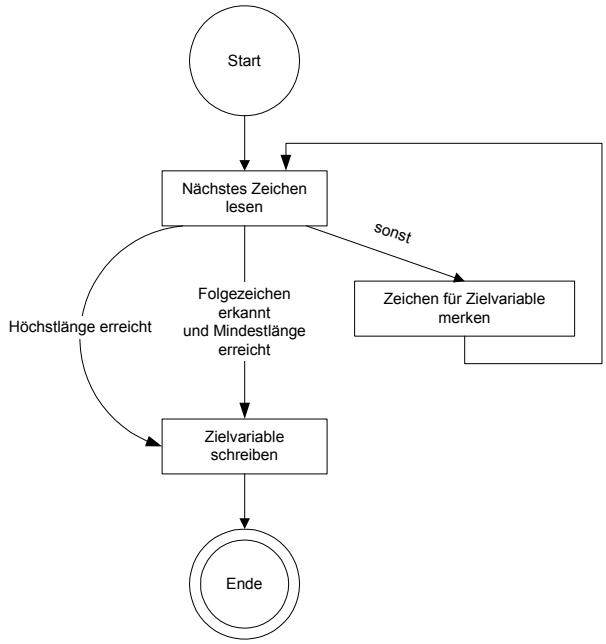

ABBILDUNG 10: ABLAUF FÜR ZEICHENKETTEN „%S%“

Beim Automat für beliebige Zeichen (Kennung „%s%“) mit angegebener Minimal- und Maximallänge wird solange gelesen, bis entweder (nach Erreichen der Mindestlänge) das auf die Formatierungsangabe folgende Zeichen erkannt wird oder die Maximallänge überschritten wird.

Konstante Ausdrücke werden Zeichen für Zeichen verglichen und müssen genau übereinstimmen.

6.7.2.3 Formatieren von Zahlen „formatno“

Die in den Variablen angegebenen String werden oft als Zahlen verwendet. In diesem Fall wird noch die Aktion „formatno“ benötigt, die den angegebenen String als Zahl interpretiert und diese Zahl anhand einer Formatierungsangabe in einen String mit dem angegebenen Format umwandelt und in einer Variablen speichert.

Die Formatierungsangabe besteht aus folgenden fünf jeweils durch eine Tilde „**~**“ getrennten Angaben:

a) Zeichen, mit dem vor dem Komma auf die gewünschte (Minimal-)Länge aufgefüllt wird (wird auch als Padding bezeichnet),
b) gewünschte Anzahl Zeichen vor dem Komma (als Bereichsangabe), Leerstring bedeutet so viele Zeichen wie nötig (Position 1 wird dann ignoriert),
c) gewünschtes Dezimaltrennzeichen (wird nicht ausgegeben, wenn genau 0 Nachkommastellen erwartet werden),
d) gewünschte Anzahl Zeichen nach dem Komma (als Bereichsangabe), Leerstring bedeutet so viele Zeichen wie nötig (Position 1 wird dann ignoriert),
e) Zeichen, mit dem nach dem Komma auf die gewünschte (Minimal-)Länge aufgefüllt wird.

Die Bereichsangaben haben dasselbe Format wie oben in Tabelle 2 beschrieben.

Die Umwandlung der Zahlen läuft nach folgendem Schema ab:

<table>
<tr><td colspan="2">Quellstring in Fließkommazahl umwandeln</td></tr>
<tr><td colspan="2">Felder der Formatierung auswerten</td></tr>
<tr><td colspan="2">Bereichsangaben parsen</td></tr>
<tr><td>Werden Nachkommastellen benötigt?
Wahr</td><td>Falsch</td></tr>
<tr><td>Bestimmung der Vorkommastellen und Umwandlung in String</td><td>Vorkommastellen der Fließkommazahl in ganze Zahl umwandeln</td></tr>
<tr><td>Anhängen des Paddings für die Vorkommastellen</td><td>Ganze Zahl in String umwandeln</td></tr>
<tr><td>Bestimmung der Nachkommastellen und Umwandlung in String</td><td>Anhängen des Paddings für die Vorkommastellen</td></tr>
<tr><td>Führendes "0." abschneiden</td><td rowspan="4">%</td></tr>
<tr><td>Anhängen des Paddings für die Nachkommastellen</td></tr>
<tr><td>Überzählige Nachkommastellen hinten abschneiden</td></tr>
<tr><td>Vorkommastellen, Dezimaltrennzeichen und Nachkommastellen konkatenieren</td></tr>
<tr><td colspan="2">String in Zielvariable schreiben</td></tr>
</table>

ABBILDUNG 11: FORMATIEREN VON ZAHLEN

6.7.2.4 Escape-Sequenzen

Für den Benutzer ist es umständlich, nicht auf der Tastatur vorhandene Sonderzeichen in Texte einzufügen. Daher ist es sinnvoll, eine Möglichkeit zu bieten, innerhalb der Konfigurationsdatei beliebige Unicode-Zeichen zu definieren. In der Konfiguration des IfSoftterm ist es daher möglich, eine Sequenz zu verwenden, die folgendermaßen aufgebaut ist: Zuerst kommt ein Backslash („\“), dann eine hexadezimale Zahl, die dem gewünschten Unicode-Zeichencode entspricht und zum Abschluß das Zeichen „**h**“. Beim Einlesen der Konfiguration werden dann diese Sequenzen durch die entsprechenden Unicode-Zeichen ersetzt.

Darüber hinaus sind folgende mit Backslash beginnende Sequenzen definiert:

Sequenz	Bedeutung
\%	**%**
\$	**$**
****	****
\n	*Zeilenumbruch*
\r	*Wagenrücklauf*
\t	*Tabulator*
\'	**"**

TABELLE 4: ESCAPE-SEQUENZEN

6.8 Hilfsprogramme und –bibliotheken

6.8.1 ifterapi bzw. IfMulterapi

Bislang gab es neben dem rohen IT-Term-Protokoll[1] zum Anschluß von Terminals an das System 6020 nur noch eine Schnittstelle für die Verwendung von Fremd- bzw. Softwareterminals. Sie heißt „ifterapi“ und ist eine Bibliothek, welche die TCP-Verbindung zu Fremdterminalschnittstelle der 6020 kapselt. Es kann immer nur eine Instanz der Verbindung existieren. Sie kann nur unter Windows verwendet werden.

6.8.1.1 Probleme mit der bestehenden Bibliothek

Für den Einsatz im Projekt IfSoftterm ist die bisherige Bibliothek „ifterapi“ aus folgenden Gründen nicht geeignet. Es kann immer nur eine Verbindung vom 6020 Server aus geben, da dieser TCP-mäßig als Client fungiert und die ifterapi als TCP-Server. Da davon ausgegangen werden muß, daß mehr als ein Software-Terminal benötigt

[1] proprietäres Protokoll der Firma Interflex zur Kommunikation zwischen Terminals und dem 6020 Server

wird, müßte ohnehin eine Programm dazwischengeschalten werden, das einerseits mittels der ifterapi den 6020-Server bedient, andererseits die Verbindung zu den Terminals aufrechterhält. Dazu kommt, daß dieser Verteiler am besten auf dem Rechner läuft, auf dem auch der Server des System 6020 läuft, da die Terminals von der Hardware bei größeren Systemen zu schwach sind und ein extra Rechner nicht immer zur Verfügung steht. Dieser ist oft aber kein Windows-System, sondern eine der unterstützten Unix-Plattformen. Ein einfaches Portieren des Code der ifterapi scheidet deswegen aus, weil die neue Verteiler-Applikation den bisherigen Ablauf nicht direkt übernehmen kann, wie der folgende Absatz darlegt:
Der Ablauf der Fremdterminalschnittstelle kennt eine Funktion ABU (Alte Buchungen abholen), mit deren Hilfe das System 6020 die Terminals auffordern kann, alle Buchungen ab einem bestimmten Zeitpunkt erneut zu schicken. Auf diese Art und Wiese wird gewährleistet, daß auch bei einem Absturz mit kompletten Datenverlust auf dem 6020 Server durch Rückspielen der Datensicherung und anschließendem erneuten Abholen der Buchungen von den Terminals ein korrekter Datenbestand vorhanden ist. Dieser Mechanismus wird in der ifterapi von der Applikation, welche die Bibliothek einbindet, realisiert. In diesem Zusammenhang sollte diese Aufgabe vom Verteiler vorgenommen werden.
Schließlich muß auch die Kommunikation zwischen Terminal und Verteiler über TCP abgewickelt werden, da eine Bindung als Bibliothek über entfernte Rechner nicht möglich ist.

6.8.1.2 Die neue Schnittstelle „IfMulterapi“

Aus den oben genannten Gründen wird die bisherige Schnittstelle ergänzt und angepaßt. Sie erhält den Arbeitstitel „IfMulterapi“.
Es wird einerseits eine Verteiler-Applikation benötigt. Sie verhält sich gegenüber dem System 6020 wie die alte ifterapi. Intern muß sie die Buchungen so speichern, daß die Daten, auch bei Programmabstürzen oder wenn der 6020 Server gerade nicht verfügbar ist, erhalten bleiben und der oben genannten ABU-Mechnismus (Abholen alter Buchungen) realisiert werden kann. Dies kann nur über Dateien realisiert werden, da dies mit normalen PCs nicht sichergestellt werden kann.
Sie muß ebenfalls die Verbindungen zu den einzelnen Terminals verwalten, für die sie TCP-Server ist. Dazu wurde ein einfaches Protokoll realisiert, welches das Senden der Buchungen und das Lesen der Buchungsantworten ermöglicht. Da auch die

Buchungssätze im Interflex-eigenen Containerformat nur druckbare Zeichen enthalten, wurde es so gestaltet, daß es zeilenweise im Textmodus eingegeben werden kann und auch die Antwortdaten in einem solchen Format kommen. Dies hat den Vorteil, daß zu Testzwecken auch Standard-Telnet-Programme verwendet werden können.

Beim Problem um die Pufferung der Daten handelt es sich um einen Spezialfall des Erzeuger-Verbraucher-Problems. Müßten diese nicht aus Sicherheitsgründen auf Platte gespeichert werden, könnte man einfach eine Message-Queue verwenden, wobei die Verbindungsthreads zu den Terminals die Daten hineinschreiben und der Verbindungsthread zum System 6020 sie wieder herausholt. Wie dieses Problem im allgemeinen gelöst werden kann, wird in 7.1 näher beschrieben.

Das Verwenden von normalem Dateizugriff ist nicht praktikabel. Die Verwaltung von Lese- und Schreibposition muß im Programm geregelt werden, da die verwendeten Systeme allesamt nur einen Schreib-Lese-Zeiger kennen. Bei einer Aufforderung zum Abholen alter Buchungen muß der Zeiger von Hand (byteweise) durch die Sätze zurückgesetzt werden, da die Sätze unterschiedlich groß sein können. Genau dieses Problem lösen aber die index-sequentiellen Dateien (ISAM[DIS01]), die auch eine der möglichen Datenbanken des System 6020 sind. ISAM steht für Indexed Sequential Access Method, also um eine Methode, Daten satzweise abzulegen und über Schlüssel darauf zuzugreifen. Man kann in Dateien satzweise nach einem Schlüssel sortiert navigieren, es gibt einen Lese- und einen Schreibzeiger. Der Overhead gegenüber reinen Dateien ist gering. Es können Schlüssel definiert werden, mit deren Hilfe beispielsweise schnell der erste Satz ab einem bestimmten Datum gefunden werden kann. Aus diesen Gründen wird im Verteiler nun eine ISAM-Datei verwendet. Einziger Nachteil ist, daß die Bibliothek nicht thread-sicher ist. Daher muß die Synchronisation zwischen Lesen- und Schreiben im Verteiler gewährleistet werden.

Das folgende Schaubild zeigt den Aufbau der Verteilerapplikation IfMulterapi:

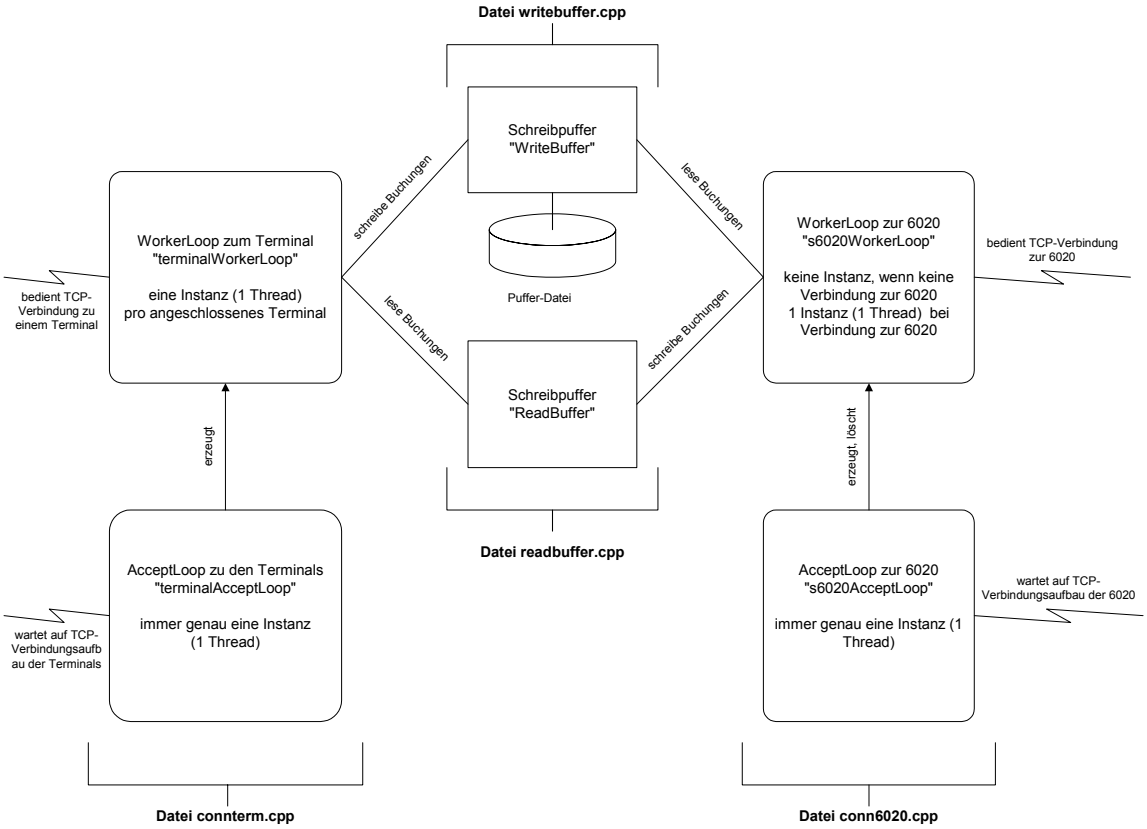

ABBILDUNG 12: AUFBAU DER VERTEILERAPPLIKATION IFMULTERAPI

Ein weiteres Schaubild beschreibt den gesamten Aufbau des Anschlusses eines Terminals an das System 6020:

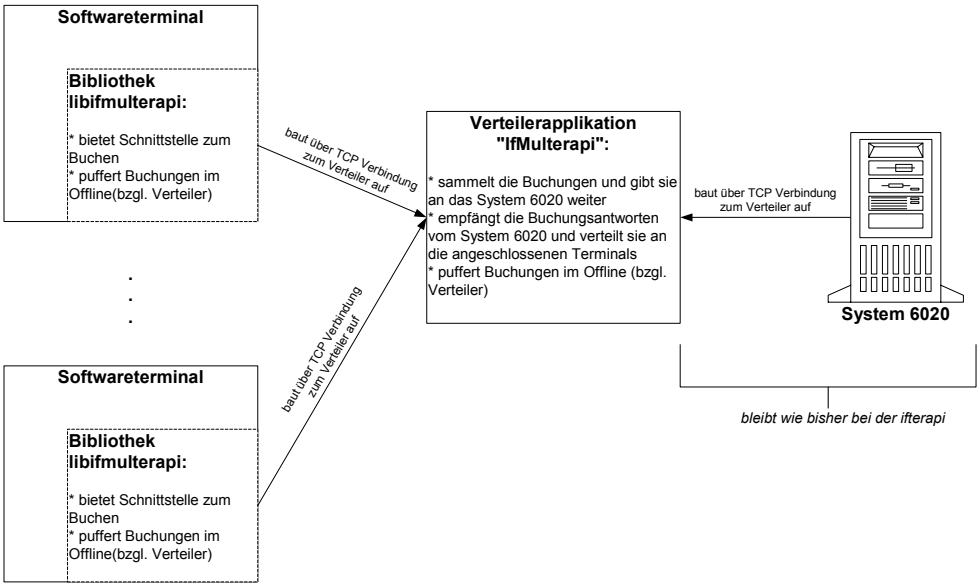

ABBILDUNG 13: ANSCHLUß EINES TERMINALS AN DAS SYSTEM 6020

6.8.1.3 Datensicherheit bei der Übertragung

Sofern das System nicht in geschützten Netzen abläuft, kann dieses TCP-basierende Protokoll auch einfach über SSH getunnelt werden. Dabei wird eine sichere, verschlüsselte Verbindung erstellt, durch die dann die TCP-Daten gesendet werden.
Das Prinzip des Tunnelns über SSH wird anhand dieses Schaubilds beschrieben:

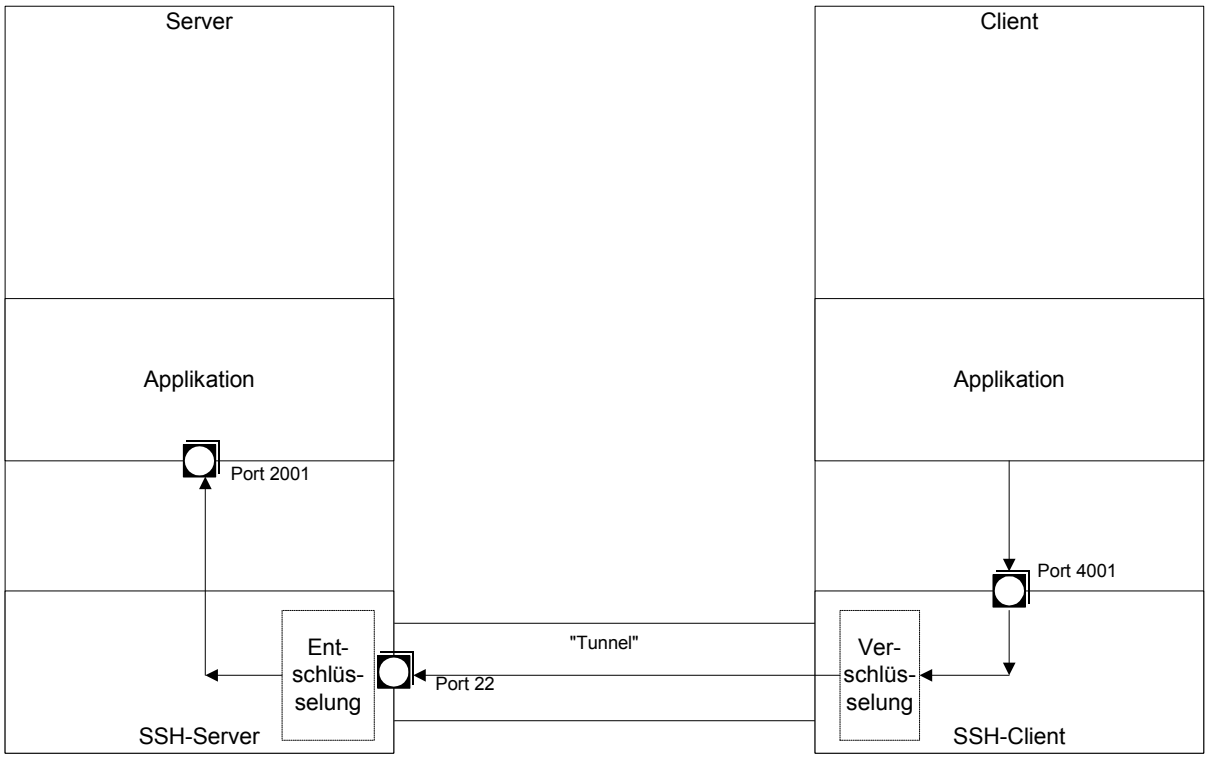

ABBILDUNG 14: TUNNELN VON TCP-VERBINDUNGEN ÜBER SSH

Das SSH-Protokoll sieht die Möglichkeit vor, innerhalb der verschlüsselten Verbindung zwischen zwei Rechnern andere TCP-Verbindungen durchzuleiten.
Soll nun eine Applikation auf dem Client eine sichere Verbindung zu einer Applikation auf dem Server (hier zu Port 2001) aufnehmen, wird folgendermaßen vorgegangen: Der Client öffnet eine SSH-Verbindung zum Server (hier auf Port 22). Beim Aufbau der Verbindung wird angegeben, daß in diesem Fall der Port 4001 auf dem Client an den Port 2001 auf dem Server weitergegeben werden soll. So kann sich die Client-Applikation mit Port 4001 auf dem lokalen Rechner verbinden und mit diesem TCP-Port so kommunizieren, also ob er direkt mit der Server-Applikation verbunden wäre. Die Server-Applikation erhält eine Verbindung zum SSH-Server, der dann die Daten so bereitstellt, wie es auch eine direkt angeschlossene Client-Applikation gemacht hätte.

Der Port der IfMulterapi wird dann für alle anderen Rechner mit Ausnahme des lokalen gesperrt. So wird gewährleistet, daß kein Unbefugter sich als Pseudo-Terminal an das System hängt und eigene Buchungen absetzt. Das Tunneln hat den Vorteil, daß Standard-Software verwendet wird, die auch ständig fehlerbereinigt wird. Bei einem proprietären Protokoll würde dieser Aufwand ständig auf die Entwicklung von Interflex zukommen. Dies ist weder praktikabel noch wirtschaftlich sinnvoll.
Als Alternative zum Tunneln über SSH können auch private Netzwerke verwendet werden, die auf IP-Ebene die komplette Kommunikation verschlüsseln. Auch dies erfordert keine Änderungen an der IfMulterapi-Bibliothek.

6.8.1.4 Die Bibliothek „libIfMulterapi“

Würde für jede Softwareterminal-Applikation erneut von Hand die Verbindung zur IfMulterapi programmiert, würde dieselbe Arbeit mehrfach gemacht. Daher ist eine statische Bibliothek mit dem Arbeitstitel „libIfMulterapi“ vorgesehen, die diesen Teil übernimmt.
Sie kennt neben Verbindungsauf- bzw. abbau (`init()` bzw. `exit()`) nur noch Funktionen zum Senden einer Buchung (`sendData()`), lesen der nächsten Buchungsantwort (`receiveAnswer()`), einer Kombination aus beiden (`sendDataWithAnswer()`) sowie einer Abfrage des Verbindungsstatus (`getStatus()`). Das auf TCP aufsetzende Protokoll wird also mit Hilfe der Bibliothek gekapselt und läßt sich unabhängig von der Kenntnis des Protokolls verwenden. Intern wird bei der Initialisierung ein extra Thread generiert, der die Verbindung zum Verteiler bedient. Er erhält die Daten mittels derselben Pufferung wie innerhalb des Verteilers. Die entsprechenden Klassen können hier wiederverwendet werden und brauchen nur einmal gepflegt werden. Außerdem bleiben die Buchungen auch bei einem Abbruch der Verbindung zum Verteiler erhalten und können nach Wiederaufbau nachgesandt werden. So sind alle Teile für sich genommen offline-fähig.

6.8.2 Auswertung mathematischer Ausdrücke

Für das Parsen der mathematischen Ausdrücke konnte ich eine frei verwendbare Bibliothek[SteG1] einsetzen. Sie deckt den geforderten Umfang voll ab und bietet sogar noch mehr Funktionalität.

Die Bibliothek unterstützt die Grundrechenarten (Addition, Subtraktion, Multiplikation, Division) einschließlich richtige Präzedenz, die beiden Vorzeichen, die Konstante π, die Funktionen Sinus, Kosinus, Tangens, Arcus Sinus, Arcus Kosinus, Arcus Tangens, Zehnerlogarithmus, Quadratwurzel, die Operatoren Potenzieren, Integer Division, Integer Modulo sowie die Zahlen im dezimalen, hexadezimalen und binären Format. Auch das Setzen von Klammern wird unterstützt.

6.8.3 Konvertierung UCS2 nach UTF-8

Das Hilfsprogramm „ucs2toutf8“akzeptiert genau einen Kommandozeilenparameter: Eine ganze Zahl, die von der C-Funktion strtol erkannt wird, d.h. normal als Dezimalzahl, beginnend mit „0“ als Oktalzahl oder beginnend mit „0x“ als Hexadezimalzahl. Es gibt dann die zugehörige UTF-8-Sequenz als Folge von mit „h“ abgeschlossenen Hexadezimalzahlen aus.

Die Quelltexte dieses Programms liegen dieser Arbeit bei. Das Verfahren zur Umwandlung von Unicode nach UTF-8 ist im Anhang in 13.3 beschrieben.

6.8.4 Test der seriellen Schnittstelle

Das Hilfsprogramm „serialtest“ öffnet die als Kommandozeilenparameter (als Gerätedatei) angegebene serielle Schnittstelle und gibt alle eingelesenen Zeichen hexadezimal aus.

Das Programm verwendet die interne v24-Bibliothek des Systems 6020, welches die Funktionen zur Kommunikation über die serielle Schnittstelle zur Verfügung stellt.

Die Quelltexte des Programms liegen dieser Arbeit bei.

6.8.5 Aufruf von dynamischen Bibliotheken

Das Hilfsprogramm „sotest“ dient zum Test des Aufrufs dynamischer Bibliotheken mit einer erst zur Laufzeit bekannten Anzahl Parameter. Es erhält als erste beiden Kommandozeilenparameter den Pfad der Bibliothek und den Namen der Funktion (bzw. des Symbols). Alle weiteren Parameter folgen darauf und werden als Long-Werte interpretiert. Der Rückgabewert der Funktion wird als Long-Wert interpretiert und ausgegeben.
Im selben Projekt wird noch eine dynamische Bibliothek erzeugt, die nur eine einzige Funktion exportiert, nämlich die Funktion add. Diese nimmt an, daß der erste Parameter die Anzahl der folgenden angibt und addiert diese folgenden Parameter. Das Ergebnis der Addition wird zurückgegeben.
Die Quelltexte des Programms und der Bibliothek liegen dieser Arbeit bei.

6.9 Erweiterbarkeit

Wie bei vielen Programmen wird es auch beim IfSoftterm in Zukunft zu Erweiterungen des Funktionsumfangs kommen. Aus diesem Grund wird es notwendig, daß, ähnlich wie die bestehenden Aktionen und Steuerelemente implementiert sind, sich auch zusätzliche Aktionen und Steuerelemente hinzuprogrammieren lassen. Eine Dokumentation, die dies beschreibt, wurde erstellt.

6.10 Tests

Für das IfSoftterm wurde eine große Anzahl von Testfällen erstellt, mit deren Hilfe man einen großen Teil der Funktionalität testen kann. Im entsprechenden internen Dokument werden die notwendigen Schritte zur Erstellung der Konfiguration sowie die gewünschte Reaktion des Systems beschrieben.

7 Ausgewählte Probleme der Systemprogrammierung

Während der Arbeit am Projekt IfSoftterm traten in größerem Umfang Probleme aus der Systemprogrammierung auf. Um diesem gerecht zu werden, sind diese hier beschrieben.
Viele der Probleme können auch in anderen Gebieten eine Rolle spielen.

7.1 Gepuffertes Erzeuger-Verbraucher-Problem

Dieses Problem stellt einen Spezialfall des Erzeuger-Verbraucher-Problems dar.
Es wird eine Ressource von zwei Seiten als Warteschlange verwendet. Der Erzeuger schreibt Informationen hinein, die der Verbraucher wieder ausliest und löscht. Damit es kein gegenseitiges Überschreiben der internen Verwaltungsstrukturen der Liste gibt, muß sichergestellt werden, daß immer nur eine Seite eine Änderungsprozedur ausführt. Diese Problematik wird normalerweise durch Standardwerkzeuge wie die Message Queue oder die Semaphore gelöst.
In diesem speziellen Fall kommt hinzu, daß die Warteschlange auf der Festplatte gespeichert werden muß, um mehr Sicherheit bei Stromausfällen oder Programmabstürzen zu haben. Außerdem könnten auf diese Weise alle Daten gepuffert werden, solange keine Verbindung zum Server besteht und sichergestellt werden, daß die einzelnen Datensätze gleich auf Festplatte protokolliert werden.
Eine Lösung hierfür ist, die Daten, statt sie im Speicher zu halten, satzweise ans Ende einer Datei zu schreiben und auf der anderen Seite am Anfang der Datei zu entnehmen. Bereits versandte Dateien können dann im Datensatz gekennzeichnet werden. Bei einem Abbruch des Programms könnte man dann auf den ersten ungelesenen Satz positionieren und so an der richtigen Stelle weitermachen.
Die Verwendung von einfachen Textdateien erwies sich als problematisch. Da alte Sätze irgendwann einmal automatisiert gelöscht werden sollen, müßte die Datei von Hand durchsucht werden, bis ein neuerer als der letzte zu löschen Datensatz gefunden werden kann. Dazu wäre aber die Datei gesperrt und könnte eine gewisse Zeit nicht für die Pufferung verwendet werden.

Dies läßt sich dadurch vermeiden, daß nicht eine einfache Textdatei verwendet wird, sondern die einfache DISAM-Datenbank[DIS01], die bei Interflex neben Datenbanksystemen wie Oracle oder Microsoft SQL Server eine Option für die Speicherung der Daten im System 6020 darstellt. Alte Sätze können so gelöscht werden, während parallel ein Weiterbetrieb möglich ist.

7.2 Aufruf von Unterprogrammen

7.2.1 Implementierung

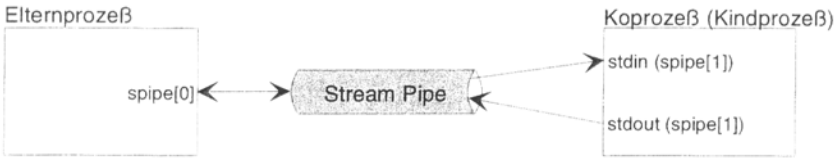

ABBILDUNG 15: STREAM PIPE [HERH1] S.810

Ziel beim Aufruf von Unterprogrammen ist es, diesen Daten an die Standardeingabe zu übergeben und deren Ausgabe einzulesen. Nach [HerH1] Kap. 19.2 läßt sich dies mit einer sog. Stream Pipe zwischen Eltern- und Kindprozeß realisieren. Dabei handelt es sich um ein Paar von Unix Domain Stream Sockets. Dieses kann dann auf beiden Seiten wie eine Datei, auf die sowohl geschrieben als auch gelesen werden kann, verwendet werden.

Der Ablauf ist folgendermaßen: Der Elternprozeß baut ein solches Socketpaar auf. Dann wird der Kindprozeß erzeugt, der die Deskriptoren erbt. Dann schließt jeder der beiden Prozesse den jeweils anderen der beiden Deskriptoren. Der Kindprozeß leitet die Standardein- und -ausgabe auf den noch offenen Deskriptor um. Dann lädt er das Programm und führt es aus.

7.2.2 Probleme

Unter bestimmten Umständen funktioniert dieser Mechanismus nicht. Wenn das Kindprogramm erwartet, daß es Daten von der Standardeingabe lesen soll, bis dieses Device geschlossen ist, muß der Elternprozeß nach dem Schreiben aller Daten die Stream Pipe schließen. Ansonsten würde der Kindprozeß unendlich auf weitere Daten warten. Wird allerdings auf Seiten des Elternprozesses der Deskriptor geschlossen, kann der Elternprozeß von dort auch keine Daten mehr lesen. Alle Ausgaben des Kindprozesses ab diesem Zeitpunkt wäre verloren.

Das genannte Szenario ist realistisch, wie sich am Beispiel des cat-Befehls belegen läßt. Dieser liest die Daten zeilenweise von der Standardeingabe und gibt sie gleich an die Standardausgabe weiter. Er wird erst beendet, wenn die Standardeingabe geschlossen wird bzw. die dort angehängte Datei zu Ende ist.

7.2.3 Lösung

Dieses Problem kann nur gelöst werden, indem es dem Elternprozeß ermöglicht wird, weiter zu lesen, obwohl er den Schreibdeskriptor geschlossen hat. Zu diesem Zweck wird eine zweite Pipe erstellt, so daß jede nur eine Richtung bedient.

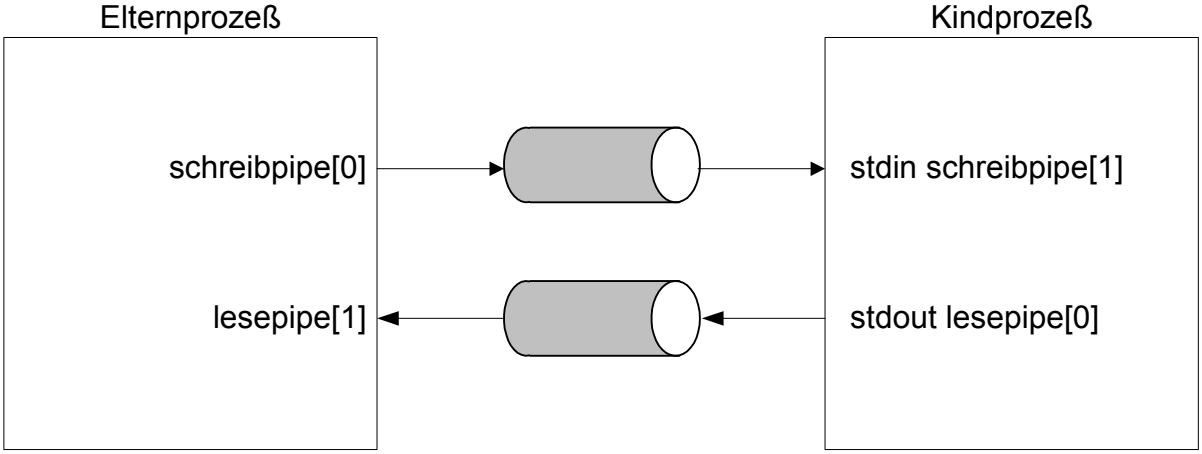

ABBILDUNG 16: KOMMUNIKATION ÜBER ZWEI PIPES

In dieser Konstellation kann jetzt der Elternprozeß, den Deskriptor schließen, mit dessen Hilfe er schreibt, und dennoch weiterhin von der anderen Pipe lesen.

7.3 Kritische Abschnitte

An verschiedensten Stellen in dieser Arbeit trat das Problem auf, daß gewisse Abschnitte nur von einem Thread durchlaufen werden dürfen (beispielsweise wenn gemeinsame Ressourcen verwendet werden). Dies ist auch unter dem Begriff „Kritischer Abschnitt“ bekannt.

Kritische Abschnitte wurden in folgenden Zusammenhängen eingesetzt:
IfSoftterm:

- Sicherstellen, daß der Eingabebildschirm nur einmal gestartet wird,
- Ressourcen einer Signalbehandlungsroutine gegen erneutes Auftreten des Signals schützen,
- Installation externer serieller Geräte schützen, so daß eine neue Installation erst möglich ist, wenn die alte abgeschlossen ist,
- ebenso bei der Installation von Timeout-Aktionen ,
- Warteschlange für Aktionen bzw. Ansichten: Interne Strukturen können bei gleichzeitigem Hinzufügen und Löschen zerstört werden,
- beim Logging muß eine Ausgabe abgeschlossen sein, bevor die nächste beginnen kann,
- Zugriffe auf Variablen dürfen nur hintereinander erfolgen, um die internen Strukturen nicht zu zerstören.

IfMulterapi (Bibliothek und Verteiler):

- Verbindungsaufbau bzw. -abbau muß beendet sein, bevor erneut eine Verbindung auf- bzw. abgebaut werden kann,
- der Zugriff auf den aktuellen Zustand der Verbindung muß synchronisiert werden,
- beim Logging muß eine Ausgabe abgeschlossen sein, bevor die nächste beginnen kann,
- die beiden Seiten des Zugriff auf den Schreib- oder auch Lesepuffer müssen gegeneinander gesperrt werden.

Der Kritische Abschnitt wurde unter Unix mittels eines Mutexes und unter Windows mit Hilfe eines Events implementiert.

7.4 Warten auf Ressource

Oft soll ein Thread solange blockiert werden, bis eine bestimmte Ressource verfügbar ist oder ein Timeout eingetreten ist. Selbst in den Fällen, in denen eigentlich kein Timeout benötigt wird, ist er sinnvoll, da beim Entblocken in die Logdatei geschrieben werden kann, daß das Programm noch korrekt läuft; der Timeout sollte in diesem Fall allerdings so lange gewählt werden, daß weder nennenswert Prozessorlast erzeugt wird, noch das Logfile zu groß und unübersichtlich wird.

Zu diesem Zweck wurde eine Klasse Semaphore erstellt, die dies realisiert. Sie hat zwei Methoden Wait und Post. Wait wartet, bis entweder die Ressource verfügbar ist oder ein Timeout aufgetreten ist. Wait gibt zurück, ob die Ressource verfügbar ist oder nur der Timeout erreicht wurde. Post wird verwendet, um dem wartenden Thread anzuzeigen, daß die Ressource verfügbar ist.

7.4.1 Implementierung

Die beiden Methoden sind unter den verschiedenen unterstützen Plattformen unterschiedlich implementiert, da die verwendeten Mittel der Interprozeßkommunikation jeweils nur auf einem Teil der Systeme zur Verfügung stehen.

Methode Wait:

Betriebssystem	Bibliothek	Vorgehen
Windows	API (kernel32.lib)	WaitForSingleObject, hat Timeout eingebaut
Linux	pthread	sem_wait, für Timeout wird extra Thread gestartet, der mit sleep wartet und dann signalisiert
Solaris	pthread	sema_wait, für Timeout extra Thread
AIX	pthread	pthread_mutex_lock, für Timeout extra Thread
HPUX	pthread (FSU Threads unter HPUX 10.20)	pthread_mutex_lock, für Timeout extra Thread

TABELLE 5: IMPLEMENTIERUNG VON WAIT

Methode Post:

Betriebssystem	Bibliothek	Vorgehen
Windows	API (kernel32.lib)	PulseEvent
Linux	pthread	sem_post
Solaris	pthread	sema_post
AIX	pthread	pthread_mutex_unlock
HPUX	pthread (FSU Threads unter HPUX 10.20)	pthread_mutex_unlock

TABELLE 6: IMPLEMENTIERUNG VON POST

Die beschriebene Implementierung funktioniert unter AIX bis zur Version 4.2 nicht, da die Funktion pthread_mutex_unlock den Fehlercode „EPERM“ zurückgibt. Dieser bedeutet bei der Funktion lt. dessen Manual „The calling thread does not own the mutex lock”[Man01]. Der aufrufende Thread darf den Mutex nicht entblocken, da er nicht Eigentümer ist (und ihn gesetzt hat). Dies unterscheidet die AIX-Implementierung von der gleichaussehenden HPUX-Implementierung. Dieses Problem tritt offenbar nicht auf, wenn der Mutex-Typ PTHREAD_MUTEX_NORMAL ist.[Unb06] Ein Setzen dieses Typs ist aber ab AIX 4.3 möglich. Im Rahmen dieser Arbeit stand kein System mit der Betriebssystemversion 4.3 zur Verfügung. Daher konnte dies nicht überprüft werden.

Thread erzeugen:

Betriebssystem	Bibliothek	Vorgehen
Windows	API (kernel32.lib)	CreateThread
Unix	pthread	pthread_create

TABELLE 7: IMPLEMENTIERUNG DES STARTS EINES THREADS

7.5 Hintereinanderabarbeiten von Aktionen bzw. Ansichten

7.5.1 Überblick

Die Verarbeitung von Aktionen und Ansichten findet grundsätzlich sequentiell statt, d.h. es wird immer nur eine Aktion auf einmal ausgeführt.

Dem folgenden Schaubild kann entnommen werden, wie ein Aufruf weitergereicht wird.

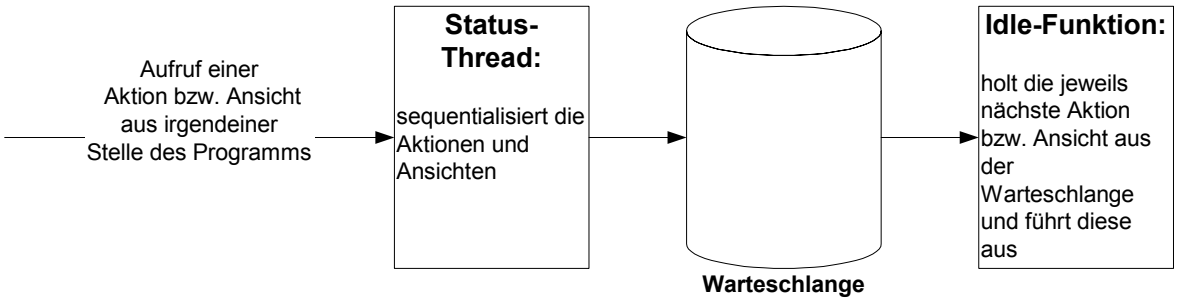

ABBILDUNG 17: AUFRUF VON AKTIONEN BZW. ANSICHTEN

Der Status-Thread dient dazu, die Aktions- bzw. Ansichtsaufrufe, die von verschiedenen Stellen des Programms kommen können, in eine Reihenfolge zu bringen. In dieser gelangen die einzelnen Aktionen bzw. Ansichten in die Warteschlange.

Ursprünglich war vorgesehen, in einem eigenen Thread blockierend aus der Warteschlange zu lesen und die eigentliche Ausführung durchzuführen. Dies hätte zwei Nachteile: Die Ausführung der Aktionen könnte viel Rechenzeit benötigen und die Reaktionen auf Benutzereingaben ausbremsen. Außerdem ist es von Seiten der Gtk-Bibliothek günstiger, wenn möglichst viele Zugriffe vom Hauptthread durchgeführt werden. Die Synchronisierung ist weniger aufwendig und deshalb performanter.

In der Gtk-Bibliothek gibt es die Möglichkeit, Idle-Routinen zu registrieren, die immer dann aufgerufen werden, wenn von der Oberfläche keine Ereignisse zur Abarbeitung vorhanden sind. Dieser Mechanismus wurde schließlich verwendet, um die genannten beiden Probleme zu umgehen.

7.5.2 Sequentialisierung von Ereignissen

Während des Programmablaufs werden viele asynchrone Ereignisse erzeugt. Diese lösen vielfach Aktionen (dies gilt natürlich auch entsprechend für das Aktivieren von Ansichten) aus, bevor die Ausführung der bisherigen beendet ist. Daher muß ein Verfahren angewandt werden, das sicherstellt, daß beliebig oft Aktionen angestoßen werden, wobei der anstoßende Thread nicht blockiert wird und die Aktionen immer hintereinander abgearbeitet werden.

Dies wurde folgendermaßen implementiert:

Die Methode zum Auslösen von neuen Aktionen („activateAction“) startet einen extra Thread mit einer weiteren Methode („activateActionEx“), die das eigentlich Eintragen übernimmt. Ein zusätzlicher Status-Thread über nimmt das Auslesen und die Verarbeitung der Aktionen.

Zur Synchronisation werden zwei Semaphoren benötigt. Für sie wurden jeweils die Dijkstra'schen P- und V-Operationen[HerH1] S. 779 implementiert. Da für die Implementierung von Threads die pthreads-Bibliothek verwendet wurde, wurde die P-Operation mit `sem_wait` und die V-Operation mit `sem_post` implementiert.

Der Algorithmus sieht nun folgendermaßen aus:

Methode ActivateActionEx:

Aktion	Erläuterung
P(#1)	immer nur eine Instanz dieser Funktion darf ausgeführt werden
Aktion eintragen	durch diesen Algorithmus genügt eine einfache Variable für den Namen der Aktion
V(#2)	dem Status-Thread anstoßen

Status Thread:

Aktion	Erläuterung
P(#2)	Blockieren, bis Aktion zur Bearbeitung ansteht
Auslesen und Verarbeiten der Aktion	Variable enthält den Namen, notwendiger Code wird ausgeführt
V(#1)	Nächste Aktion ermöglichen

Die beiden Semaphoren haben also folgenden Sinn:

Aktion	Zustand zu Programmbeginn	Erläuterung
#1	entblockt	Sicherstellen, daß immer nur ein Thread aktivieren kann
#2	geblockt	Blockieren des Status-Thread, solange nichts zu tun ist

8 Embedded PCs

Embedded PCs sind normale Intel-IBM-kompatible PCs mit geringeren Ausmaßen. Die Prozessoren sind i.d.R. etwas langsam als in aktuellen Desktop-Rechnern.
Sie verfügen oft über ein Vielzahl von Anschlußmöglichkeiten direkt auf der Platine.

8.1 Hardware

Der im Rahmen der Diplomarbeit verwendete Embedded PC verfügt über folgende Anschlüsse auf der Platine:

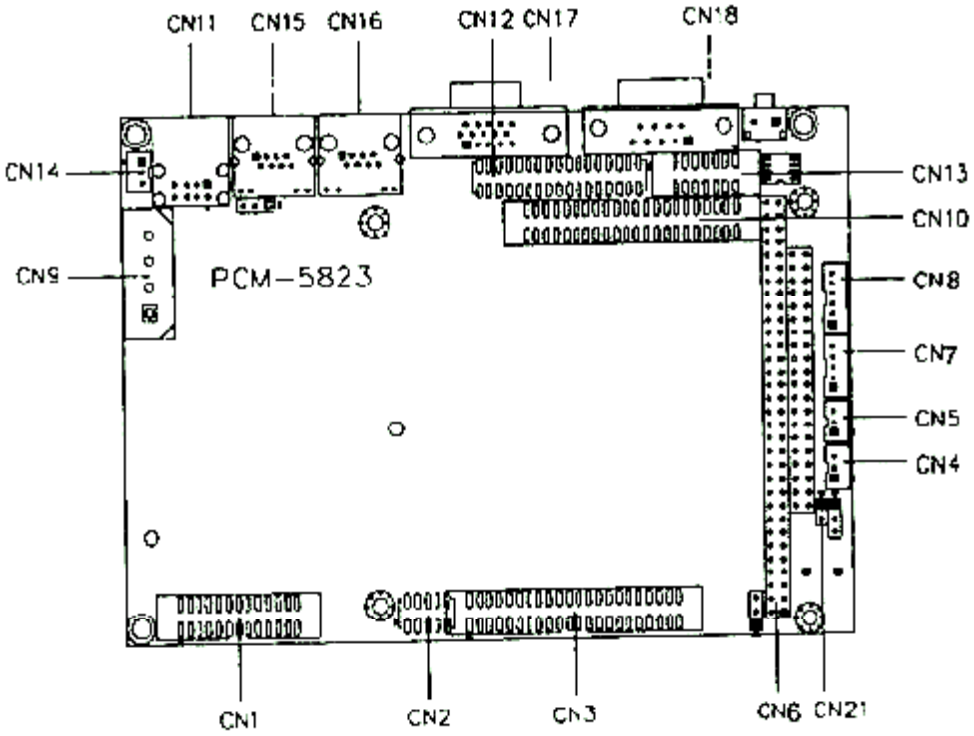

ABBILDUNG 18: EMBEDDED PC[PCM01]

In diesem Fall sind dies: Parallele Schnittstelle, AC97 Audio Connector, LDC Verbindung, Hilfsstromversorgung, CPU Stromversorgung, PC/104 Verbindung, Infrarot-Verbindung, PS2-Anschluß, Stromversorgung, IDE Controller, USB-Anschluß, Disketten-Controller, Serielle Schnittstelle, ATX Standby-Verbindung, 2 Ethernet-Anschlüsse und Monitor-Ausgang. Eine Unterstützung für LCD-Displays ist ebenfalls vorhanden.
Bezüglich der Schnittstellen können solche Rechner sicherlich mit normalen PCs mithalten. Als Prozessor wird ein Pentium-MMX-kompatibler Embedded-Prozessor von Cyrix mit 300 MHz verwendet.

Die Entwicklung fand auf einem normalen PC (1 GHz-Athlon mit 512 MB RAM und 40 GB Festplatte) statt. Der Embedded PC wurde nur zum Testen verwendet.
Darüber hinaus stand für die Tests mit externen seriellen Geräten der Interflex Berechtigungsleser P60/USB/RS232/RS485[IFi04] zur Verfügung. Er wurde über die serielle Schnittstelle des Rechners betrieben, da USB-Treiber unter Linux noch nicht zur Verfügung stehen.

8.2 Betriebssystem und Laufzeitumgebung

Auf dem Testsystem wurde eine SuSE-Linux-Distribution 7.3 mit dem dem Window-Manager KDE 2.2.1 verwendet. Die letzten Tests wurden mit dem Window-Manager fvwm95 durchgeführt, der weniger Overhead erzeugt. Darüber hinaus wurden die Pakete für das Gtk 2.0, der Xerces-XML-Parser 1.7 sowie einige Standard-Bibliotheken des System 6020 und die im Rahmen dieser Arbeit entstandene LibIfMulterapi nachinstalliert.
Um die Buchungsschnittstelle IfMulterapi zu verwenden, muß ein 6020-Server auf einem per TCP erreichbaren Rechner installiert sein. Auf einem beliebigen, per TCP erreichbaren, Rechner muß die Verteilerapplikation der IfMulterapi laufen. Der 6020-Server muß so konfiguriert sein, daß er sich auf den Port des Verteilers verbindet.
Für die Verwendung der 6020 API muß diese auf dem lokalen System laut deren Dokumentation installiert sein. Die IfSoftterm-Anwendung muß unter demselben Benutzer laufen, wie die 6020 API installiert wurde. Außerdem müssen die bei der Installation eingetragenen Umgebungsvariablen verfügbar sein. Als Pfad der Bibliothek wird dann der Pfad eingetragen, in dem die Datei „lib6020api.so“ installiert ist.

9 Zusammenfassung und Bewertung

Für die Programmierung von konfigurierbaren Clients für Client-Server-Systeme gibt es keine allgemein verbindlichen Regeln. Wie im Sprichwort führen auch in der Softwareentwicklung viele Wege nach Rom. Genau so viele Möglichkeiten gibt es auch, solche Clients zu programmieren.

Im Rahmen dieser Arbeit ist mit dem Interface-Action-Config-Modell ein Denkansatz entwickelt worden, mit dessen Hilfe eine solche Programmierung ermöglicht werden soll. Dieser wurde dann am praktischen Beispiel im Projekt IfSoftterm angewandt und erprobt. Er wurde den Anforderungen gerecht: Es entstand ein lauffähiger Prototyp zur Konfiguration von Benutzerschnittstellen. Die notwendigen Schnittstellen (dynamische Bibliotheken, ausführbare Programme, Buchungsschnittstelle ifterapi und 6020 API) sind vorhanden.

Das Projekt IfSoftterm ist auch für künftige Erweiterungen gut gerüstet. Die Dokumentation enthält Anleitungen, mit deren Hilfe zusätzliche Aktionstypen und Steuerelemente hinzugefügt werden können.

Ein sinnvolle Erweiterung des Projekts wäre die Entwicklung einer graphischen Oberfläche, mit deren Hilfe sich die Konfiguration durchführen ließe.

Alles in allem ist das Projekt IfSoftterm ein Anknüpfungspunkt für zukünftige Produkte der Firma Interflex, die derartige Funktionalität benötigen.

10 Abbildungsverzeichnis

Abbildung 1: Model-View-Controller-Modell[Unb01] 19
Abbildung 2: Model-View-Presenter-Modell[Unb02] 20
Abbildung 3: Interface-Action-Config-Modell 21
Abbildung 4: Beispiel für derartige Abläufe 23
Abbildung 5: Automat für das Zusammensetzen von Strings 43
Abbildung 6: Automat für das Parsen des Formatierungsstrings 45
Abbildung 7: Automat für ganze Zahlen „%i%“ 45
Abbildung 8: Automat für Kommazahlen „%f%“ 46
Abbildung 9: Ablauf für Buchstabenfolgen „%z%“ 46
Abbildung 10: Ablauf für Zeichenketten „%s%“ 47
Abbildung 11: Formatieren von Zahlen 48
Abbildung 12: Aufbau der Verteilerapplikation lfMulterapi 52
Abbildung 13: Anschluß eines Terminals an das System 6020 52
Abbildung 14: Tunneln von TCP-Verbindungen über SSH 53
Abbildung 15: Stream Pipe [HerH1] S.810 58
Abbildung 16: Kommunikation über zwei Pipes 59
Abbildung 17: Aufruf von Aktionen bzw. Ansichten 63
Abbildung 18: Embedded PC[PCM01] 66
Abbildung 19: Beispiel für einen möglichen Ablauf 81
Abbildung 20: Screen-Shot „Info“ 84
Abbildung 21: Screen-Shot „StartView“ 85
Abbildung 22: Screen-Shot „AnswerView“ 85
Abbildung 23: Screen-Shot „OfflineView“ 85

11 Tabellenverzeichnis

Tabelle 1: Zuordnung Steuerelemente zu Gtk-Widgets 30
Tabelle 2: Bereichsangaben für Formatierungsstrings 44
Tabelle 3: Mögliche Formatierungskonstanten für Formatierungsstrings 44
Tabelle 4: Escape-Sequenzen 49
Tabelle 5: Implementierung von Wait 61
Tabelle 6: Implementierung von Post 61
Tabelle 7: Implementierung des Starts eines Threads 62
Tabelle 8: Umwandlung Unicode nach UTF-8 86

12 Quellen

Im Rahmen dieser Arbeit wurden folgende Quellen verwendet:

12.1 Bücher und Loseblattsammlungen:

[AdaB1] Bernhard Adamski
Die Organisation der computergesteuerten Zeitwirtschaft
Datakontext-Verlag, Köln, 1995

[AdaB2] Bernhard Adamski
Praktisches Arbeitszeitmanagement, 2. Auflage
Datakontext-Verlag, Köln, 2000

[BauW1] Wolfgang Bauer
Skript Nr. 358 Systemprogrammierung 2
Eigendruck FH Furtwangen

[BooG1] Grady Booch, Jim Rumbaugh, Ivar Jacobson
Das UML-Benutzerhandbuch, 2. Auflage
Addison-Wesley-Longman, Bonn, 1999

[BöhH1] Hans Böhme, Wilfried Butzer, Matthias Herold
Computerunterstützte Urlaubsanspruchsermittlung
Bibliomed Med. Verl.-Ges., Melsungen, 2001

[DIS01] nicht bekannt
BYTE DESIGNS D-ISAM User Manual, June 14, 1993
Byte Designs Ltd., Langley/BC/Kanada, 1993

[GamE1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
Entwurfsmuster, Elemente wiederverwendbarer objektorientierter Software
Addison-Wesley-Longman, Bonn, 1996

[GolA1] Adele Goldberg, David Robson
Smalltalk-80: the language
Addison-Wesley, Reading/MA/USA, 1989

[GolA2] Goldberg, Katzman, Rubin
Object-Oriented Methology Workshop, Course Materials Version 4.0
ParcPlace Systems Inc., Sunnyvale/CA/USA 1992

[HerH1] Helmut Herold
Linux-Unix-Systemprogrammierung, 2. Auflage
Addison Wesley, München, 1999

[JunJ1] Jürgen Junghans, Karlheinz Roschmann
Zeit- und Betriebsdatenerfassung
Verlag Moderne Industrie, Landsberg/Lech, 1993

[KopH1] Herbert Kopp
Skript Compilerbau
Vorlesungsunterlagen Prof. Schwegler

[LalW1] Wilf R. Lalonde, John R. Pugh
Inside Smalltalk, Volume I
Prentice-Hall Inc., Englewood Cliffs/NJ/USA, 1990

[LalW2] Wilf R. Lalonde, John R. Pugh
Inside Smalltalk, Volume II
Prentice-Hall, Inc., Englewood Cliffs/NJ/USA, 1991

[SchH1] Herbert Schild
C++ from the ground up, Second Edition;
Osborne/McGraw-Hill, Berkeley/CA/USA, 1998

[SteR1] W. Richard Stevens
Unix Network Programming
Prentice-Hall International, Englewood Cliffs/NJ/USA, 1990

[SteR2] W. Richard Stevens
Advanced Programming in the UNIX Environment
Addison Wesley, Reading/MA/USA, 1992

[StrB1] Bjarne Stroustrup
Die C++ Programmiersprache
Addison Wesley, München, 2000

[ZeiA1] A. Zeidler, R. Zellner
Software-Ergonomie Techniken der Dialoggestaltung
R. Oldenbourg Verlag, München Wien, 1992

[ZieJ1] J. Ziegler, R. Ilg (Hrsg.)
Benutzergerechte Software-Gestaltung
R. Oldenbourg Verlag, München Wien, 1993

12.2 Internetseiten

12.2.1 Einzelne Seiten zu einem speziellen Thema

[BraT1] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen
XML 1.0
http://www.w3.org/TR/1998/REC-xml-19980210
04.06.2002

[BraT2] Tim Bray, Jean Paoli, C.M. Sperberg, McQueen, Eve Maler
XML 1.0 (Second Edition)
http://www.w3.org/TR/2000/REC-xml-20001006
06.03.2002

[CzyR1] Roman Czyborra
The ISO 8859 Alphabet Soup
http://czyborra.com/charsets/iso8859.html
10.05.2002

[GroM1] Mathias Grohe
Programmieren in der Skriptsprache Tcl/Tk
http://www.uni-muenster.de/ZIV/Mitarbeiter/MathiasGrote/tcl/Tcl.html
06.03.2002

[JelD1] Dejan Jelović
Using namespaces properly C++
http://www.codeproject.com/cpp/using_namespaces.asp
21.06.2002

[RicN1] Norbert Richter
DCE Glossar
http://www.fujitsu-siemens.es/servers/dce/dce_de/dcegloss/dcegl_m.htm
18.07.2002

[RicN2] Norbert Richter
DCE Glossar
http://www.fujitsu-siemens.es/servers/dce/dce_de/dcegloss/dcegl_t.htm
18.07.2002

[Unb01] nicht bekannt:
Model View Controller
http://www.object-arts.com/EducationCentre/Overviews/MVC.htm
07.03.2002

[Unb02] nicht bekannt:
Model-View-Presenter Framework
http://www.object-arts.com/EducationCentre/Overviews/ModelViewPresenter.htm
07.03.2002

[Unb03] nicht bekannt
Unicode Code Charts
http://www.unicode.org/charts/
24.04.2002

[Unb04] nicht bekannt
Client-Server-Architektur
http://gd.tuwien.ac.at/study/hrh-glossar/1-2_17.htm
22.05.2002

[Unb05] nicht bekannt
Robolab
http://www.lego.com/dacta/robolab/default.htm
22.05.2002

[Unb06] nicht bekannt
AIX Version 4.3 Base Operating System and Extensions Technical Reference, Volume 1
http://www.llnl.gov/computing/tutorials/workshops/workshop/pthreads/man/pthread_mutex_lock.html
16.07.2002

[WhiJ1] E. James Whitehead: jun., Murata Makoto
RFC 2376: XML Media Types
http://www.ietf.org/rfc/rfc2376.txt
27.06.2002

[YerF1] F. Yergeau
RFC 2279: UTF-8, a transformation format of ISO 10646
http://www.ietf.org/rfc/rfc2279.txt
10.05.2002

12.2.2 Referenzseiten zu größeren Themenbereichen

[Ref01] nicht bekannt
TCL/TK
http://www.tcl-tk.net
08.04.2002

[Ref02] nicht bekannt
Perl
http://www.perl.com
08.04.2002

[Ref03] nicht bekannt
Python
http://www.python.org
08.04.2002

[Ref04] nicht bekannt
XML
http://www.w3.org/XML
23.04.2002

[Ref05] nicht bekannt
XML
http://www.w3.org/TR/1998/REC-xml-19980210.html
04.06.2002

[Ref06] nicht bekannt
Apache XML Project, u.a. Xerces
http://xml.apache.org
05.06.2002

[Ref07] nicht bekannt
DTD Tutorial
http://www.w3schools.com/dtd/
07.06.2002

[Ref08] nicht bekannt
Gtk
http://www.gtk.org/
07.06.2002

[Ref09] nicht bekannt
QT
http://doc.trolltech.com/3.0/
07.06.2002

[Ref10] Stefan Münz
HTML-Dateien selbst erstellen
http://selfhtml.teamone.de
10.06.2002

[Ref11] nicht bekannt
Unicode
http://www.unicode.org
27.06.2002

12.3 Dokumente

[HeeD1] Dimitri van Heesch
doxygen - Manual for version 1.2.9.1

[IFi01] Core-Team Interflex
Beschreibung eines Anwendungsszenarios für easy@production

[IFi02] EVA-Team Interflex
Memorandum easy@production

[IFi03] Interflex
Werkerdialoge

[IFi04] Interflex
OEM-Handbuch für Berechtigungsleser P60/USB/RS 232/485, Stand 4/2002

[IFi05] Interflex
Modul 928: Buchungen Online von Fremdterminals lesen und verarbeiten

[IFi06] Peter Gabrian
6020 API
Application Programming Interface für System 6020 Version V1.51, Stand 4/2002

[IFi07] Valérie Bory et al.
Datenbank: Grundlagen für IF-Produkte
Dictonary German-English/Wörterbuch Deutsch-Englisch

[PCM01] nicht bekannt
PCM-5823 Startup Manual

12.4 Quellen für Programmcode

[SteG1] Gerhard Stephan
Parsen + Auswerten von mathematischen Ausdrücken
Quelle: http://www.ad-factum.de/tips/fparser/
19.03.2002

[Unb07] nicht bekannt
Kommandozeilenparameter (getopt_long für Plattformen, die diese Funktion nicht haben)
Quelle: http://lottery.merseyworld.com/Wheel/
06.05.2002

12.5 Sonstiges

[IFd01] Lucia Zeitler et al.
Datenbank: Grundlagen für IF-Produkte
Dokument: PEP-Begriffe -> K

[Man01] nicht bekannt
Manual: pthread_mutex_unlock Subroutine (mit dem AIX-Betriebssystem mitgelieferte manpage)

[GieT1] Thomas Giese (lt. Quelle)
Quelle: http://www.zugernet.ch/users/f.schori/komment.html
23.04.2002

13 Anhang

13.1 Glossar

6020

Die “6020” oder das “System 6020“ ist ein Zeiterfassungs- und Zutrittssystem der Firma Interflex.

6020 API

Programmierschnittstelle zum System 6020. Sie steht unter Windows in Form einer DLL, unter den verschiedenen Unix-Derivaten als Shared Object zur Verfügung.

ABU

Abholen alter Buchungen; Verfahren zur Erhöhung der Ausfallsicherheit eines Gesamtzeiterfassungssystems. Dabei werden alle Buchungen auf dem Terminal gespeichert. Fällt der Server aus, kann dieser ja mit dem Stand der letzten Datensicherung versehen werden. Dann werden einfach alle Buchungen ab diesem Zeitpunkt erneut von den Terminals angefordert, so daß wieder alles im System vorhanden ist.

BDE

Betriebsdatenerfassung, dazu gehören Bereiche wie Maschinen-datenerfassung, mobile Datenerfassung, Prozeßdatenerfassung, Qualitäts-datenerfassung, Zeitdatenerfassung, Auftragsdatenerfassung und Lohndaten-erfassung.

Control

Steuerelement; einzelner Baustein, aus denen die graphische Oberfläche eines Programms zusammengesetzt wird.

Dialog

Ein Dialog stellt den komplexen Teil der Benutzeroberfläche dar, den der Benutzer zu einem bestimmten Zeitpunkt sieht und mit dem er interagiert.

DOM-Parser

Bestandteil der Xerces-XML-Bibliotheken von Apache; Parser, für XML-Dateien

Embedded PC

Von den Ausmaßen her kleinerer PC, der im Prinzip dieselbe Architektur verwendet wie IBM-Intel-kompatible Systeme, aber weniger Platz benötigt und auch viele Komponenten statt als Einsteckkarten bereits auf der Hauptplatine hat. Oft besitzt er zusätzliche Anschlüsse wie für LCD-Touchscreen-Displays.

Errorlevel

Wert den ein Programm an das Betriebssystem zurückgibt; wird oft für die Anzeige von Fehlern verwendet.

Erzeuger-Verbraucher-Problem

Beim Erzeuger-Verbraucher-Problem gibt es zwei verschiedene Klassen von Prozessen bzw. Threads, den Erzeugern, die Daten erzeugen und in einem Puffer ablegen und den Verbrauchern, die Daten aus dem Puffer entnehmen und verarbeiten. Dabei muß der Erzeuger blockiert werden, wenn Puffer voll ist und der Verbraucher solange der Puffer leer ist. Diese Blockade darf nur solange aufrecht erhalten werden, bis die Bedingung nicht mehr vorhanden ist. Der Zugriff auf den Puffer selbst ist ein kritischer Abschnitt. Zur Lösung dieses Problems bedarf es i.d.R. mehrerer Synchronisationsmechanismen. *nach* **[BauW1]** *S. 52*

fork

Mit diesem API-Befehl wird unter Unix ein neuer Prozeß gestartet, der i.w. eine Kopie des aktuellen Prozesses ist.

Gdk

Beim Gdk handelt es sich um die auf tieferen Ebenen angesiedelten Funktionen des Gtk

Gtk

Das Gtk (GNOME Tool Kit) ist eine Bibliothek, mit deren Hilfe Benutzeroberflächen für das X11-System programmiert werden können

GUI

Graphical User Interface; die Oberfläche, die der Benutzer wahrnimmt

Ifterapi

Schnittstelle zum Anschluß von Fremdterminals an das System 6020

IT-Term-Protokoll

Proprietäres Protokoll der Firma Interflex für die Kommunikation zwischen dem System 6020 und den einzelnen echten Terminals.

Kritischer Abschnitt

Ein kritischer Abschnitt besteht aus den Aktionen, die jeweils nur von einem einzelnen Prozeß oder durchlaufen werden dürfen. Ein Beispiel ist der exklusive Zugriff auf Betriebsmittel. *nach* **[BauW1]** *S. 52*

Message-Queue

Nachrichtenwarteschlange; verschiedenste Prozesse bzw. Threads können Nachrichten eintragen, die dann zur Abarbeitung einzeln wieder ausgelesen werden.

Mutex

Mutex ist aus „mutal exclusion“ (zu deutsch gegenseitiger Ausschluß) zusammengezogen. Man versteht darunter „ein Synchronisationsobjekt, durch das sich Threads gegenseitig ausschließen können. Ein Mutex wird oft dazu verwendet sicherzustellen, daß gemeinsam benutzte Variablen von anderen Threads immer in einem konsistenten Zustand gesehen werden. Ein Mutex ist vergleichbar mit einem Semaphor oder Lock.“**[RicN1]**

Oberflächeneigenschaften

Diejenigen Attribute eines Steuerelements, die für den Benutzer sichtbar sind. Dazu gehören z. B. Position, Größe, Farbe oder Schriftart.

PEP

Personaleinsatzplanung; Anpassung der Arbeitszeiten der einzelnen Mitarbeiter an den Personalbedarf

PLI – Programmierbare Listen

einfache Skriptsprache für das System 6020 zum Programmieren von Abrechnungen

Steuerelement

einzelner Baustein, aus denen die graphische Oberfläche eines Programms zusammengesetzt wird.

SSH-Tunnel

Einbetten von TCP-Verbindungen in eine andere verschlüsselte Verbindung

system

Mit diesem API-Befehl wird unter Unix ein anderes Programm wie in einer Kommandozeile der Shell gestartet

Terminal

Bedienungseinheit im Zeiterfassungs- und Zutrittssystem. Es wird zwischen Hardwareterminals (Mikrocontroller mit wenigen Tasten und mehr oder weniger stark beschränkter Ausgabe) und Softterminals (auf PCs laufende Applikation mit ähnlicher Funktionalität) unterschieden.

Thread

Unter einem Thread (zu deutsch Faden) versteht man „einer von mehreren quasi gleichzeitigen Programmabläufen innerhalb einer Aufgabe (Prozesses). Ein Thread wird auch als nebenläufiger oder leichtgewichtiger Prozeß bezeichnet. Der Einsatz von Threads entspricht einem Multiprocessing, das vom Programm selbst verwaltet wird und in derselben Task (Prozeß) abläuft wie das Hauptprogramm.“[RicN2]

Touchscreen

Bildschirm, der auf Berührung reagiert. Von Seiten des Betriebssystems wird der Touchscreen wie ein Maus verwendet.

UCS

UCS steht für „Universal Character Set“ und ist auch unter Unicode bekannt. Es gibt UCS-2 (mit 16 bit) und UCS-4 (mit 32 bit).

Unicode

Unicode[Ref11] ist eine universelle Repräsentation von Zeichen mit Hilfe eines plattformunabhängigen Standards. Es ist eine Implementierung der ISO-10646-Norm.

UTF-8

UTF-8 codiert lt. RFC 2279[YerF1] UCS-Zeichen so, daß sie in Umgebungen, die nur 8-Bit-Zeichen zulassen, verwendet werden können. Die einzelnen UCS-Zeichen werden dann entweder auf ein oder eine Folge von 8-Bit-Zeichen abgebildet.

VB

Visual Basic; Entwicklungsumgebung und Programmiersprache der Firma Microsoft

VBA

Visual Basic for Applications; mit VB verwandte Entwicklungsumgebung und Programmiersprache, die in anderen Applikationen integriert ist. Beispielsweise läßt sich mit ihr Code in die Programme des Office-Pakets von Microsoft integrieren. Auch das System 6020 der Firma Interflex besitzt eine solche Integration.

Widget

Das Wort Widget setzt sich aus “window” und ”gadget“ zusammen und bedeutet einen Baustein einer graphischen Oberfläche. Der Begriff stammt aus der Unix/X11-Welt; unter Windows wird eher Steuerelement bzw. Control verwendet.

Xerces

Xerces ist eine Bibliothek, die zum Parsen von XML-Dateien verwendet wird. Sie ist nach der ersten Schmetterlingsart benannt, die in Amerika durch Wirken des Menschen ausgestorben ist.

XML

XML[WhiJ1] steht für Extensible Markup Language. XML ist eine Weiterentwicklung des SGML-Standards (Standard Generalized Markup Language; Standard für die Definition von Dokumentenbeschreibungssprachen). Sie besteht i.w. aus in spitzen Klammern eingeschlossenen Tags mit zugehörigen Attributen sowie Texten, die zusammen eine Baumstruktur abbilden.

13.2 XML-Beispielkonfiguration

Zur Illustration der Verwendung des IfSoftterm wird der folgende Ablauf in eine Konfiguration umgesetzt:

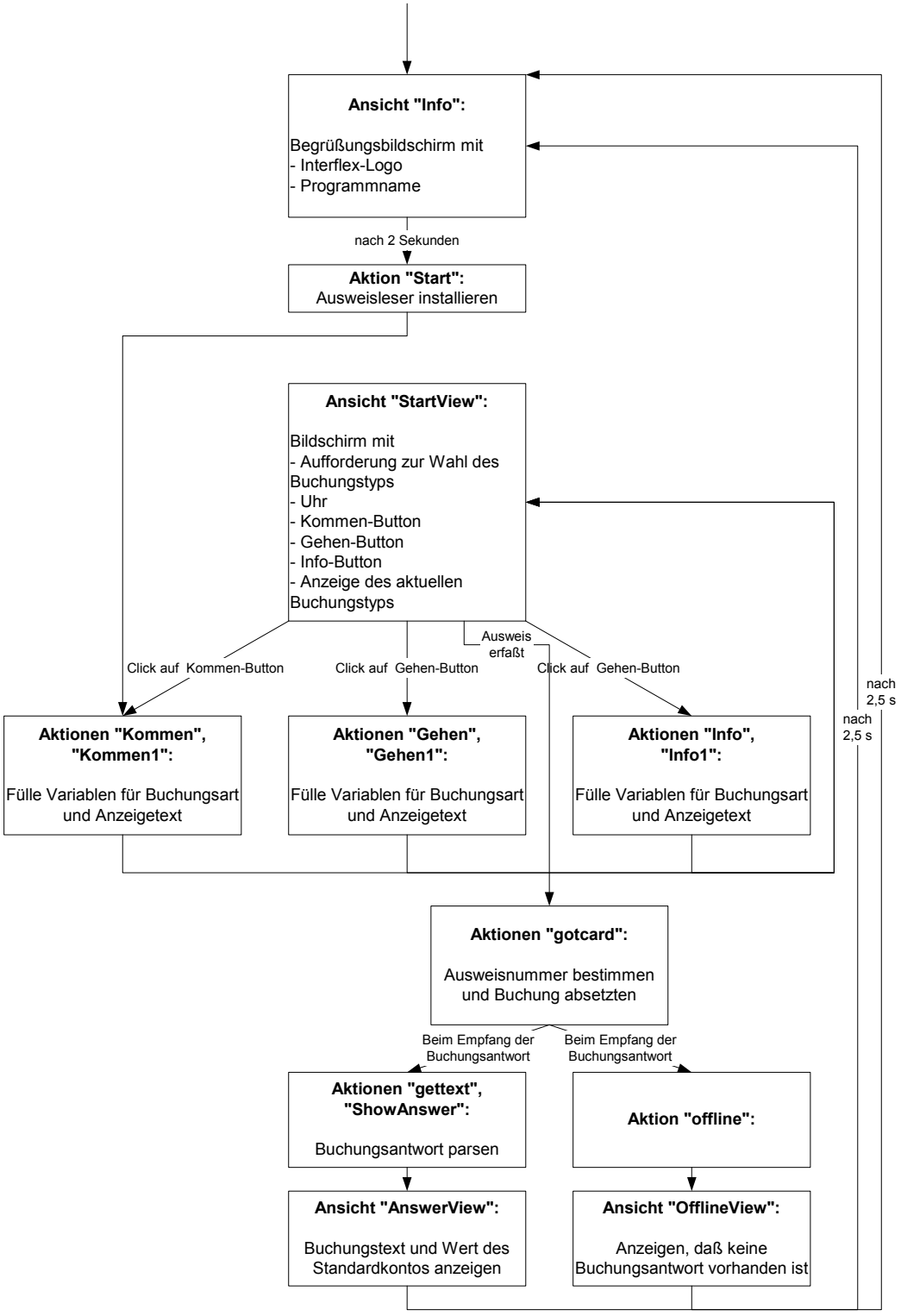

ABBILDUNG 19: BEISPIEL FÜR EINEN MÖGLICHEN ABLAUF

Die Konfiguration sieht dann folgendermaßen aus:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE prog SYSTEM "ifsoftterm.dtd">
<prog>
  <view id="Info" onidletime="2000"
    onidleaction="Start,Kommen,Kommen1">
    <image x="10" y="10" width="620" height="350"
      picture="/home/ddreher/lxembterm/ifHintergrund.jpeg"
      visible="1" />
    <label x="10" y="360" width="620" height="60"
      font="Sans bold 64" align="center"
      text="IfSoftterm v.1.0.2" color="0000FF" visible="1" />
    <label x="10" y="420" width="620" height="50"
      font="Sans bold 16" align="center"
      text="\a9h 2002 Interflex Datensysteme GmbH & Co. KG"
      visible="1" />
  </view>
  <view id="StartView">
    <label x="10" width="620" y="10" height="30" font="Sans 14"
      text="Bitte wählen Sie zunächst über einen der Kn\f6hpfe"
      visible="1" align="left" />
    <label x="10" width="620" y="40" height="30" font="Sans 14"
      text="die gew\fchnschte Aktion und buchen Sie dann mit
      Ihrem Ausweis." visible="1" align="left" />
    <datetime x="10" width="620" y="140" height="50"
      font="Sans bold 16" format="%weekday%, den %d%. %month%
      %y% %h%:%0m%:%0s% Uhr %tz%" visible="1" align="left" />
    <label x="10" width="620" y="200" height="60"
      font="Sans bold 24" text="$Buchungstext$" visible="1"
      align="left" />
    <button x="10" width="200" y="270" height="200"
      label="Kommen" onclick="Kommen,Kommen1" visible="1" />
    <button x="220" width="200" y="270" height="200"
      label="Gehen" onclick="Gehen,Gehen1" visible="1" />
    <button x="430" width="200" y="270" height="200"
      label="Info" onclick="Info,Info1" visible="1" />
  </view>
  <view id="AnswerView" onidleview="StartView" onidletime="2500">
    <label x="10" y="10" width="620" height="460"
      font="Sans bold 24" align="center"
      text="$BuchungsantwortLang$" visible="1" />
```

```
</view>
<view id="OfflineView" onidleview="StartView" onidletime="2500">
  <label x="10" y="10" width="620" height="460"
    font="Sans bold 24" align="center"
    text="Buchung registriert. Verbindung offline."
    visible="1" />
</view>
<action id="Start" resetbefore="$*$" action="gettext,gotcard"
  type="externaldevice" device="kartenleser" cardvar="$CARD$"
  datavar="$DATA$" customervar="$CUSTOMER$"
  versionvar="$VERSION$" />
<action id="Kommen" type="concat" format="Kommen"
  dest="$Buchungstext$" />
<action id="Kommen1" type="concat" format="0100"
  dest="$Buchungsart$" view="StartView" />
<action id="Gehen" type="concat" format="Gehen"
  dest="$Buchungstext$" />
<action id="Gehen1" type="concat" format="0101"
  dest="$Buchungsart$" view="StartView" />
<action id="Info" type="concat" format="Info"
  dest="$Buchungstext$" />
<action id="Info1" type="concat" format="0110"
  dest="$Buchungsart$" view="StartView" />
<action id="gettext" type="scan"
  format="%s%~2102%s$BuchungsantwortKonto$%~%s%">
  $CompleteAnswer$</action>
<action id="gotcard" type="ifterapi"
  data="$Buchungsart$~200117~2002$@DATE$~2003$@TIME$~2000$CARD$$
  VERSION$~" name="server" onanswer="gettext,ShowAnswer"
  action="offline" timeout="4000" answervar="$Buchungsantwort$"
  completeanswervar="$CompleteAnswer$" />
<action id="ShowAnswer" view="AnswerView" type="concat"
  dest="$BuchungsantwortLang$"
  format="$Buchungsantwort$ ($BuchungsantwortKonto$)" />
<action id="offline" view="OfflineView" />
<general mousevisible="0">
  <device name="kartenleser" stx="\x2" etx="\x3"
    dev="/dev/ttyS0" baudrate="9600" parity="n" stopbits="1"
    bits="8" setlength="14" />
  <ifterapi id="server" ip="172.18.21.98" port="2003"
    terminalno="17" />
```

```
    <startview>Info</startview>
    <text id="inputok">Eingabe in Ordung</text>
    <text id="inputnotok">Eingabe fehlerhaft</text>
    <text id="fieldname">Feld</text>
    <text id="Help_Back">zurück</text>
    <msgbox>
      <label x="0" y="0" width="640" height="400"
        color="0000FF" backcolor="CCCCCC" font="sans bold 24"
        visible="1" align="left" />
      <okbutton x="0" y="400" width="640" height="80"
        color="FF0000" backcolor="333333" font="sans bold 24"
        label="okay" visible="1" />
    </msgbox>
  </general>
</prog>
```

Dies ergibt dann die folgenden Bildschirme:

ABBILDUNG 20: SCREEN-SHOT „INFO“

ABBILDUNG 21: SCREEN-SHOT „STARTVIEW“

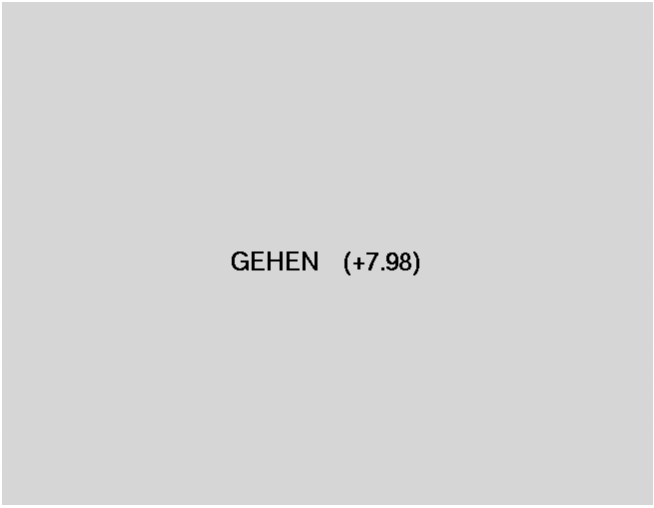

ABBILDUNG 22: SCREEN-SHOT „ANSWERVIEW“

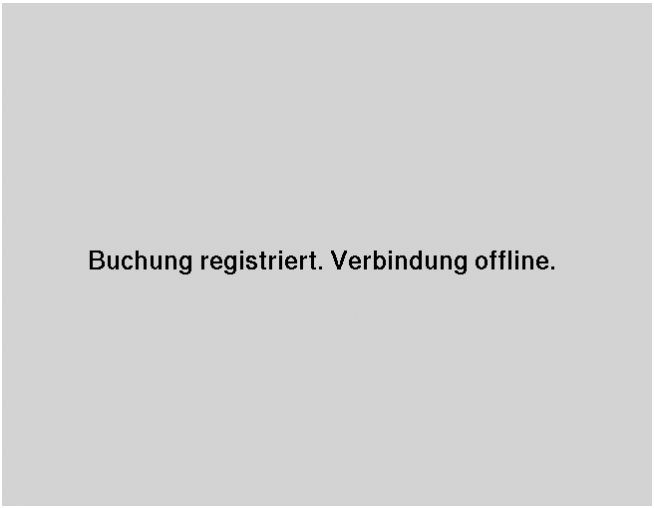

ABBILDUNG 23: SCREEN-SHOT „OFFLINEVIEW“

13.3 UTF-8-Codierung

Die UTF-8 codiert Unicode-Zeichen (UCS) so, daß sie mit normalen 8-bit-Zeichen dargestellt werden können. Eine Beschreibung des Unicode-Zeichensatzes kann bei [Unb03] heruntergeladen werden.

Die Umwandlung von Unicode nach UTF-8 und zurück ist in RFC-2279 [YerF1] folgendermaßen definiert:

Unicode-Zeichen	UTF-8-Sequenz
0000 0000_h-0000 $007F_h$	$0xxxxxxx_b$
0000 0080_h-0000 $07FF_h$	$110xxxxx_b$ $10xxxxxx_b$
0000 0800_h-0000 $FFFF_h$	$1110xxxx_b$ $10xxxxxx_b$ $10xxxxxx_b$
0001 0000_h-001F $FFFF_h$	$11110xxx_b$ $10xxxxxx_b$ $10xxxxxx_b$ $10xxxxxx_b$
0020 0000_h-03FF $FFFF_h$	$111110xx_b$ $10xxxxxx_b$ $10xxxxxx_b$ $10xxxxxx_b$ $10xxxxxx_b$
0400 0000_h-7FFF $FFFF_h$	$1111110x_b$ $10xxxxxx_b$ $10xxxxxx_b$ $10xxxxxx_b$ $10xxxxxx_b$ $10xxxxxx_b$

TABELLE 8: UMWANDLUNG UNICODE NACH UTF-8

Das erste Byte bestimmt also, wie viele Bytes folgen (die alle mit binär 10 beginnen). Die Bits des Unicode-Zeichens werden hier in die mit x markierten Bits der UTF-8-Sequenz verteilt, wobei immer vom am wenigsten signifikanten zum meist signifikantesten Bit vorgegangen wird.

Bei der Umwandlung von UTF-8 wird umgekehrt vorgegangen: Zunächst wird das erste Byte ausgewertet und die Anzahl der Folgebytes berechnet. Dann werden die mit x markierten Bits der Reihe nach auf das Unicode-Zeichen aufgerechnet.

Als Beispiel wird hier das Euro-Zeichen „**€**“ von Unicode nach UTF-8 umgewandelt:

Das Euro-Zeichen wird mit $20ac_h$ im Unicode codiert.[Unb03]

Dies entspricht 00000000 0000000 00100000 10101100_b.

Es liegt im Intervall $[0000\ 0800_h;0000\ FFFF_h]$, wird also durch drei Zeichen mit folgendem Muster codiert: $1110xxxx_b$ $10xxxxxx_b$ $10xxxxxx_b$

Das Einfüllen der Bits von rechts nach links ergibt dann:

11100010_b 10000011_b 10101100_b oder $E2_h$ 83_h AC_h.