Integration eines rein interaktiven, DOS-basierten Informationssystems in das Internet-Umfeld

am Beispiel des Fahrplanauskunftssystems „RailServer“

Frederik Ramm

Studienarbeit im Wintersemester 95/96

bei Prof. Dr. H. Schmeck

Institut für Angewandte Informatik und Formale Beschreibungsverfahren

Universität Karlsruhe

Inhalt

1 Einleitung 4

2 Grundlagen 5

2.1 Abgrenzung der Begriffe und der Problematiken 5

2.2 Besondere Probleme einer solchen Integration 5

2.3 Verschiedene Zielsetzungen 6

2.4 Verschiedene Möglichkeiten der Integration 6
2.4.1 PC Anywhere 6
2.4.2 HP 500 7
2.4.3 Hardware-Lösungen 7
2.4.4 Prinzipielle Unterschiede 7

3 Das RailServer-Projekt 9

3.1 Was wurde im Rahmen des RailServer-Projekts realisiert? 9

3.2 Die Vorgehensweise bei der Realisierung 10

3.3 Die Reaktion der Internet-Öffentlichkeit auf das Projekt 10

4 Eine universelle Skriptsprache zur „Fernsteuerung“ von DOS-Applikationen 11

4.1 Idee und Funktionsprinzip 11
4.1.1 Kontrolle von Tastatur und Bildschirm 11
4.1.2 „Handlungsvorschrift“ als Skript 12
4.1.3 Speicherresidentes Programm 13

4.2 Realisierung: Virtus-Precompiler und Virtus-Interpreter 15
4.2.1 Der „Precompiler“ 15
4.2.2 Der Virtus-Interpreter 16

4.3 Details über die Virtus-Skriptsprache 18
4.3.1 Variablen und Variablendateien 18
4.3.2 Strukturelemente 18

5 Batch-Betrieb (Offline) 20

5.1 Eingabe einer Anfrage durch den Benutzer 20

5.2 Verarbeitung der Anfrage 23
5.2.1 Homogenisierung der Anfragedaten 24
5.2.2 Einsatz von Virtus und HAFAS zur Erstellung der Auskunft 27
5.2.3 Nachbearbeitung der HAFAS-Ergebnisse durch PROCESS2 31

5.3 Rücksendung der Auskunft an den Netz-Benutzer 33

6 Online-Betrieb 34

6.1 Veränderte Benutzerschnittstelle für Online-Antwendung 34
6.1.1 CGI-Programme 34
6.1.2 Java und JavaScript 35
6.1.3 Übersicht über den veränderten Programmablauf 35

6.2 Technische Anpassungen des Virtus-Ablaufs 36

6.3 Verstärkter Einsatz von Unix-Programmen 36
6.3.1 Homogenisierung der Anfragen 36
6.3.2 Scheduling-Probleme 37
6.3.3 Aufbereitung und Rückgabe der Antwort 37

7 Schlußbetrachtung 38

8 Literaturverzeichnis 39

Anhang A: Alle VIRTUS-Sprachelemente 40

Anhang B: Das RailServer-Skript 44

1 Einleitung

Das Internet mit einer ständig wachsenden Zahl von Benutzern bietet unzählige Möglichkeiten der Verteilung von Information – eMail, News, das inzwischen überholte „Gopher"-System, das boomende World Wide Web (WWW) und dessen Ableger Hyper-G sind nur einige davon[1].

Diesen modernen Informationsverteilmechanismen und den Millionen von informationshungrigen Anwendern steht auf Anbieterseite oft ein Mangel gegenüber. Bestehende Informationssysteme lassen sich nur mit hohem Zeit- und Kostenaufwand in das Internet-Umfeld eingliedern. Solange es sich um statische Informationen – einzelne Texte bzw. Dokumente – handelt, kann eine Umwandlung beispielsweise in HTML vergleichsweise schnell vorgenommen werden. Hat man es jedoch mit dynamischen Informationen zu tun, also Daten, die von einer gewissen Eingabe abhängig sind, so muß man im Zweifel mit den Daten auch das gesamte Informationssystem „portieren" – was in der Regel nur bei der Verfügbarkeit von Quelltext überhaupt möglich ist.

Insbesondere bei Anwendungen, die unter DOS zu bedienen sind, ist eine Schnittstelle zum Internet (oder einer anderen Art der entfernten, automatisierten Abfrage) eine nicht zu unterschätzende Hürde.

In dieser Arbeit soll anhand eines umfassenden Praxisbeispiels ein Weg gezeigt werden, nahezu beliebige DOS-Anwendungen „fernzubedienen" und damit jede unter DOS verfügbare Informationsquelle auch in ein globales Netz wie das Internet einzuspeisen.

Man mag einwenden, daß DOS-Systeme angesichts des großen Markterfolges von Windows im Aussterben begriffen sind. Das ist zwar richtig, aber es gibt gewichtige Gründe, sich dennoch damit auseinanderzusetzen:

- Viele „Altlasten", Software also, die mehrere Jahre alt ist und nicht gewartet wird, liegen in DOS-Version vor;
- eine universelle Anwendungs-Fernsteuerung, wie sie hier vorgestellt wird, ist unter Windows wegen der vielfach größeren Zahl der möglichen Ereignisse schwerer zu realisieren. Hat man – wie im noch zu beschreibenden „RailServer"-Beispiel – die Wahl zwischen einer DOS- und einer Windows-Version, bietet sich die DOS-Version als Arbeitsgrundlage an.

[1] vgl. Dalitz/Heyer 1995 (zu Hyper-G) und Ramm 1995 (zum WWW)

2 Grundlagen

2.1 Abgrenzung der Begriffe und der Problematiken

Im Titel dieser Arbeit ist von einem „rein interaktiven, DOS-basierten Informationsystem“ die Rede. Jede einzelne dieser Angaben ist von Bedeutung für das Problem:

- Unter einer „DOS-basierten“ Lösung ist eine Software zu verstehen, die als ausführbare Datei(en) für das Betriebssystem DOS vorliegt. Wäre die Software auch unter einem netzwerkfähigen Multitasking-Betriebssystem verfügbar, so bräuchte man ja lediglich einen Zugang per Telnet zu ermöglichen, und schon könnten die Informationen überall auf der Welt abgerufen werden.
- Mit „Informationssystem“ wird hier die Verbindung aus Daten und Algorithmen (der Software) bezeichnet, die aus diesen Daten eine zu einer Anfrage passende Antwort generieren. Mindestens einer der beiden Faktoren, Daten oder Algorithmen, liegt in einer unzugänglichen (verschlüsselten, kompilierten) Form vor – denn sonst wäre es ein leichtes, die Daten auf ein Multitasking-Betriebssystem zu übertragen und dort die Algorithmen nachzubilden.
- „Rein interaktiv“ ist ein Programm, das im Betrieb auf Benutzereingaben angewiesen ist und auch die Reaktion des Benutzers auf bestimmte Bildschirmmeldungen erwartet. Das Gegenstück dazu wäre eine Software, die einen Batch-Modus besitzt, in dem sie ohne Eingaben automatisch funktioniert. Eine solche Software wäre relativ einfach (beispielsweise über ein Mail-Gateway) „fernbedienbar“.

Nur wenn es sich um ein geschlossenes (nicht im Quelltext verfügbares) Informationssystem handelt, das unter DOS läuft und interaktiv mit einem Benutzer kommunizieren muß, besteht Bedarf an einer Problemlösung. wie sie hier vorgestellt wird.

2.2 Besondere Probleme einer solchen Integration

Da DOS ein Single-Tasking-Betriebssystem ist, ergeben sich eine Anzahl von Problemen für eine Integration in das Umfeld eines größeren Netzwerkes:

- Es ist stets ein eigens als „Informationsserver“ abgestellter Rechner erforderlich. Man kann die Software nicht, wie das beispielsweise bei Unix möglich wäre, als „Daemon“ im Hintergrund laufen lassen.
- Die Fernwartung ist schwierig bis unmöglich. Während auf einem Multitaskingsystem jederzeit, auch „remote“, sämtliche laufenden Prozesse überwacht und ggf. beeinflußt werden können, müßte eine DOS-Lösung ganz gezielt Wartungsmethoden vorsehen; die „Beobachtung“ des laufenden Systems durch den Administrator ist nur an der Konsole möglich.
- Anfragen aus dem Netz können nie parallel bearbeitet werden, sondern müssen in einer Queue auf ihre Abarbeitung durch das DOS-System warten. Eine solche Warteschlange

kann durch die Bedienung per Mail erreicht werden; es ist jedoch auch möglich, eigene Warteschlangensysteme aufzubauen (vgl. dazu Abschnitt 6.3.2).

2.3 Verschiedene Zielsetzungen

Zunächst ist zu definieren, was man von einer Integration einer DOS-Lösung in das Netz–umfeld erwartet. Ähnlich der Unterscheidung verschiedener Betriebssysteme nach Batch-, Dialog- und Echtzeitsystemen stellt sich die Frage, in welcher Zeit und auf welchem technischen Wege die DOS-Software das Netz „bedienen“ soll.

Als ersten Schritt könnte man sicherlich die prinzipielle Erreichbarkeit des Dienstes über das Netz betrachten. Weitere Schritte könnten dann die Forderung nach einer Minimierung der Antwortzeit oder sogar die Definition einer absoluten Obergrenze für die Antwortzeit sein. Letzteres ist dann vonnöten, wenn eine Online-Abfrage realisiert werden soll, bei der dem Benutzer eine endliche Geduld unterstellt wird.

2.4 Verschiedene Möglichkeiten der Integration

Es gibt bereits eine Anzahl von Produkten, die beliebige PC-Software im Internet verfügbar machen. Dabei ist zwischen reinen Software-Lösungen und solchen Produkten, die mit speziell angepaßter Hardware arbeiten, zu unterscheiden.

Zu den reinen Software-Lösungen zählen das Programm „PC Anywhere“ und das Serversystem „HP500“ von Hewlett-Packard. In beiden Fällen wird ein herkömmlicher PC verwendet, bei dem eine Software zum Einsatz kommt, die die Bildschirmausgabe und Tastatur- bzw. Mauseingabe in das Netz umlenken kann.

2.4.1 PC Anywhere

PC Anywhere ist dabei auf den Textmodus beschränkt. Es beherrscht eine Anzahl verschiedener Terminalemulationen, so daß man beispielsweise von einem VT100-Terminal oder einem Xterm-Fenster aus eine Telnet-Verbindung zum PC öffnen kann. Von da an wird die Bildschirmdarstellung über die Telnet-Verbindung unter Verwendung entsprechender Terminalsteuerzeichen für die Cursorpositionierung usw. umgeleitet; Probleme treten bei Sonderzeichen (Liniensymbole usw.) auf, und auch die Farbdarstellung ist nur bei speziellen Terminalemulationen überhaupt möglich.

Die Tastatureingaben werden ebenfalls über Escape-Sequenzen simuliert (eine Telnet-Verbindung verfügt normalerweise nicht über die Fähigkeit, PC-Sondertasten wie „Alt-F10“ zu übertragen). Um z.B. am PC die Tastenkombination Alt-X zu simulieren, ist auf Netzseite Ctrl-A gefolgt von X zu drücken.

2.4.2 HP 500

Das Server-System HP500 und einige funktionsgleiche Konkurrenzprodukte arbeiten ebenfalls auf Software-Basis, allerdings basieren diese Systeme auf dem Windows-Betriebssystem und können daher auch grafische Anwendungen (i.d.R. mit spezieller Client-Software unter einer grafischen X-Oberfläche) im Netz verfügbar machen. Da unter Windows die Funktionsaufrufe zum Aufbau der Grafik durch spezielle Software abgefangen und umgeleitet werden können, muß nicht ständig der gesamte Inhalt eines hochauflösenden Grafikbildschirms übertragen werden, sondern ein Update nur bei Veränderungen ist möglich.

2.4.3 Hardware-Lösungen

Bei Hardware-Lösungen wird in der Regel ein gewöhnlicher PC verwendet und mit einer speziellen Einsteckkarte aufgerüstet, die zugleich als Grafikadapter und als Netzschnittstelle funktioniert und zuweilen sogar Tastatur und Maus „ersetzt“[2]. Bei anderen Lösungen erfolgt die Tastatursteuerung (wie bei den vorgestellten Software-Lösungen) über ein speicherresidentes Programm und Escape-Sequenzen. Die Bildschirmausgabe muß jedoch nicht aufwendig von einem Programm auf dem PC abgefangen und ins Netz geleitet werden, sondern die „Grafikkarte“ übernimmt das. Der Vorteil gegenüber den reinen Software-Lösungen liegt hierbei darin, daß man dem PC über das Netz auch schon zusehen kann (z.B. den Bootvorgang überwachen), wenn noch keine Software geladen ist.

Auch die Hardware-Lösungen sind allerdings auf den Textmodus beschränkt. Ein regelmäßiges Versenden des Inhalts eines Grafikbildschirms ist zwar technisch möglich, scheitert jedoch an Geschwindigkeitsproblemen, da die Datenmenge sehr hoch ist.

Bei den Hardware-Lösungen gibt es auch die Möglichkeit, einen Rechner mit mehreren Prozessorkarten auszurüsten, so daß sich das Gesamtsystem nach außen wie ein Multitaskingsystem verhält, zumindest so lange, bis alle Prozessoren belegt sind.

2.4.4 Prinzipielle Unterschiede

Alle bislang erwähnten Lösungen unterscheiden sich prinzipiell von der, die in dieser Arbeit vorgestellt werden soll:

- Der Zugriff ist universell, d.h., man kann ohne weitere Anpassung jede DOS- oder Windows-Software auf diese Weise aus dem Netz fernbedienen. Die hier vorzustellende Lösung erfordert eine spezielle Anpassung für jede Anwendung.
- Der Zugriff ist interaktiv. Zu jedem Zeitpunkt kann nur ein Benutzer die Software benutzen; will man mehrere Benutzer zulassen, muß man mehrere PCs koppeln und jeden Benutzer über einen automatischen Verteilmechanismus dem nächsten freien Rechner zuweisen. Für die Lösungen, die auf Windows aufsetzen (HP 500), ist zwar prinzipiell ein Mehrbenutzerbetrieb möglich, allerdings sinkt die Leistung des Systems bei starker Belastung

[2] Eine solche Lösung ist z.B. von der Firma Logicraft unter dme Namen Omni-Ware erhältlich.

erheblich, und auch Zugriffsrecht-Probleme treten auf. In der Regel kann der fernsteuernde Benutzer mit dem ferngesteuerten System alles tun, was ihm beliebt. Erst die Verwendung von Windows NT ermöglicht einen vernünftigen Mehrbenutzerbetrieb.

Dagegen schirmt die Lösung, die hier vorgestellt wird, den PC völlig von der Netzseite ab, nimmt Anfragen entgegen und leitet diese an den PC weiter, wo sie mit vordefinierten Verfahren beantwortet werden. Der Benutzer hat keinen direkten Zugang zum PC, sondern kann nur auf die vorprogrammierten Schnittstellen zugreifen.

3 Das RailServer-Projekt

3.1 Was wurde im Rahmen des RailServer-Projekts realisiert?

Im Rahmen des „RailServer“-Projekts, das vom Autor am Rechenzentrum der Universität Karlsruhe durchgeführt wurde, entstand das Beispiel, das hier im Detail vorgestellt werden wird. Der „RailServer“ basiert auf der einer CD-ROM mit Fahrplandaten, die halbjährlich von der Deutschen Bahn AG herausgegeben wird und eine Abfragesoftware für DOS und Microsoft Windows enthält. Seine Funktionsweise ist in Abbildung 1 skizziert.

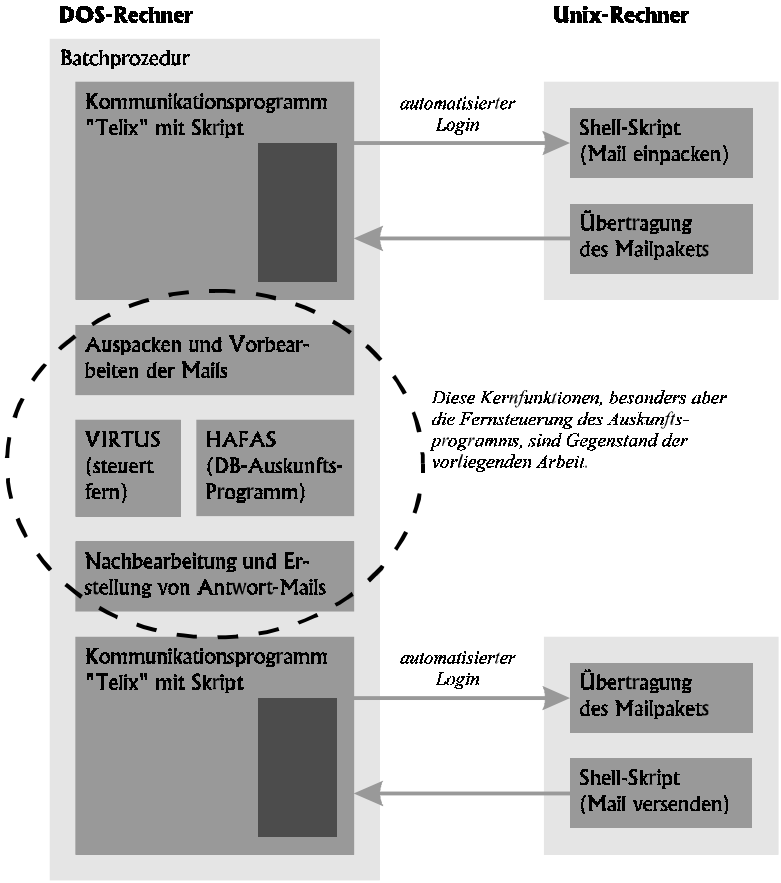

Abbildung 1: Funktionsweise des „RailServers“

Der „RailServer“ nimmt Anfragen, die per Mail oder in Form ausgefüllter WWW-Formulare eingehen, entgegen und verwendet die Daten und die Software der CD-ROM, um die gewünschten Auskünfte zu ermitteln. Die Antworten werden den Anfragern dann per Mail zugestellt.

Neben der hier vorgestellten Fernsteuerungs-Lösung verwendet der RailServer eine Anzahl von Hilfsprogrammen, die die Daten abholen und aufbereiten und die fertigen Antworten versenden; alles in allem handelt es sich um einige Programme in BASIC, um Batchprozeduren unter DOS, Skripts in der Sprache SALT für das Kommunikationsprogramm „Telix“, PERL-Skripts und Shellskripts unter Unix.

3.2 Die Vorgehensweise bei der Realisierung

Im ersten Betriebsjahr lief die RailServer-Software auf einem 486er-PC in der Wohnung des Autors und stellte mittels Modem mehrmals am Tag eine Verbindung zum Uni-Rechner her, auf dem Mail empfangen und versandt wurde. Seit Anfang September 1995 hat das Rechenzentrum der Universität einen eigenen Rechner zur Verfügung gestellt, so daß die Antwortzeit durch ununterbrochenes Abarbeiten der Queue wesentlich verkürzt werden konnte.

Mit Ausnahme der Eintragung eines Mail-Alias wurden für den RailServer im ersten Betriebsjahr keine „außergewöhnlichen“ Ressourcen verwendet; alle zur Anwendung gekommenen Internet-Dienste, -Programme und -Rechner stehen allen Studierenden der Universität offen.

Auf die Funktionsweise des RailServers wird in Kapitel 5 dieser Arbeit genauer eingegangen. Trotz aller Gründlichkeit müssen dabei einzelne (unwesentliche) Details der Implementation unerwähnt bleiben, um den Rahmen dieser Arbeit nicht zu sprengen und das Interesse des Lesers nicht auf eine zu harte Probe zu stellen.

3.3 Die Reaktion der Internet-Öffentlichkeit auf das Projekt

Die Internet-Öffentlichkeit reagierte erfreut auf den Dienst; mit zunehmendem Bekanntheitsgrad steig zwar auch das Beschwerdevolumen an, aber die Anzahl der Anfragen stieg kontinuierlich. (Die meisten Unzufriedenen glaubten, es sei ein offizieller DB-Dienst und beschwerten sich, gewohnt, daß ihnen im Internet stets die Brathähnchen in den Mund fliegen, über die Bearbeitungsdauer von bis zu 14 Stunden.) In den Monaten Juli bis September 1995 wurden im Schnitt 750 Anfragen pro Tag beantwortet, obwohl einige Monate zuvor Monaten ein gewerblicher Provider in Frankfurt/M begonnen hatte, Online-Fahrplanauskünfte anzubieten. Von Oktober bis Dezember stieg das Volumen, vermutlich wegen der verbesserten Antwortzeit, noch einmal auf rund 1.300 Anfragen täglich an.

4 Eine universelle Skriptsprache zur „Fernsteuerung“ von DOS-Applikationen

Um ein DOS-Informationssystem im Internet verfügbar zu machen, bedarf es zweier Hauptbestandteile: Einerseits der eigentlichen „Schnittstelle“ zum und im Internet, die Benutzeranfragen entgegennehmen und in irgendeiner geeigneten Form an den PC weiterleiten kann; andererseits auf PC-Seite eines Systems, das diese Anfragen in das Informationssystem einspeist und von ihm verarbeiten läßt.

In diesem Kapitel wird die Software „Virtus“ vorgestellt, die vom Autor entwickelt wurde, um den RailServer betreiben zu können. Der Name steht für „Virtual User“, und das Programm ist universell einsetzbar. Die genaue Beschreibung der Rolle, die es beim RailServer spielt, folgt im Kapitel 5.

4.1 Idee und Funktionsprinzip

Virtus soll Software – und zwar möglichst beliebige Programme – automatisch steuern. Das mit Abstand wichtigste Eingabegerät ist die Tastatur[3]. Die Ausgaben erfolgen in aller Regel auf dem Bildschirm. Virtus fällt also als erster Problemschwerpunkt die Aufgabe zu, diese beiden Geräte zu bedienen. Ferner wird irgendeine Art von Regel, eine Bedienvorschrift benötigt, nach der das Programm vorzugehen hat. Drittens muß Virtus quasi-parallel zum „fernbedienten“ Programm laufen, um mit diesem interagieren zu können. Diese drei Eckpfeiler des Systems werden nun detaillierter vorgestellt.

4.1.1 Kontrolle von Tastatur und Bildschirm

Die Überschrift klingt trivial – jedes Programm kontrolliert Tastatur und Bildschirm, indem es von der Tastatur liest und auf den Bildschirm schreibt. Rufen wir uns aber ins Gedächtnis zurück, daß Virtus gegenüber einem anderen Programm die Rolle des Benutzers oder der Benutzerin übernehmen soll. Es muß also vom Bildschirm *lesen* und auf die Tastatur *schreiben.* Beides gestaltet sich dennoch vergleichsweise einfach.

Der Bildschirminhalt (Textmodus) kann beim PC direkt aus dem Hauptspeicher ausgelesen werden. Bei den verbreiteten Farbgrafikkarten (VGA, EGA, CGA) handelt es sich um 4000 Byte, beginnend an der Speicherstelle B800 (hex). Für jede der 2.000 Bildschirmpositionen (25 Zeilen zu 80 Zeichen) ist zunächst der ASCII-Code des dargestellten Zeichens und danach ein Farbattribut gespeichert; beide Daten benötigen je ein Byte. Auch die aktuelle Cursorposition kann direkt aus dem DOS-Speicher gelesen werden – für die Fernsteuerung eines Programms kann es ja von Bedeutung sein, welches Eingabefeld gerade aktiv ist, und dann wird in der Regel der Cursor dorthin bewegt.

---

[3] Wir sprechen von DOS, daher ist die Bedeutung der Maus vernachlässigbar.

Die Tastatur ist nicht ganz so leicht zu bewältigen. Es gibt Interrupts, die ein Programm verwenden kann, um von der Tastatur zu lesen bzw. abzufragen, ob überhaupt Zeichen anliegen. Man könnte diese Interrupts auf eine selbstprogrammierte Routine „umbiegen“ und so die gewünschten Zeichen in ein Programm einspeisen. Dieser Weg ist wegen der Interrupt-Routinen mit einem hohen Programmieraufwand verbunden und überdies unsicher, da ein Programm auch auf die Benutzung der Interrupts verzichten und den Tastaturpuffer selbst auslesen könnte. Bei Virtus werden daher die (simulierten) Tastendrücke direkt in den Tastaturpuffer geschrieben, aus dem die besagten Interruptroutinen sich bedienen.

Der Tastaturpuffer ist ein Ringpuffer mit 32 Byte Kapazität. Jeder Tastendruck erzeugt zwei Datenbytes, so daß maximal 16 Zeichen gespeichert werden können. Das erste Byte enthält i.d.R. den ASCII-Wert der gedrückten Taste, das zweite Byte einen Scancode, der die Position der Taste auf der Tastatur angibt (so haben die „1“ in der Ziffernreihe und die „1“ im Nummernblock den gleichen ASCII-Wert, aber unterschiedliche Scancodes).

In Sonderfällen (Tastenkombinationen mit der „Alt“-Taste, Pfeiltasten, Funktionstasten usw.) wird im ersten Byte eine Null und im zweiten ein Schlüssel für die betätigte Taste oder Tastenkombination eingetragen.

Zusätzlich zum Ringpufferinhalt (ab Speicherstelle 41E hex im Segement 0) verwaltet DOS den Zeiger auf den aktuellen „Anfang“, also das erste Zeichen im Puffer (Speicherstelle 41A hex), sowie das Ende (Speicherstelle 41C hex). Will man also einen Tastendruck simulieren, so geht man wie folgt vor:

- Anzahl Zeichen im Puffer prüfen; wenn 16: Kein Schreiben möglich
- Sonst: Aus bekannten Grenzen des Ringpuffers, Offset des „Anfangs“ und Anzahl Zeichen im Puffer die Stelle bestimmen, an die geschrieben werden kann;
- Tastencode eintragen
- Ende-Pointer erhöhen

Daß dieser Vorgang von einem anderen Prozeß unterbrochen wird, der Daten von der Tastatur lesen möchte, wird nicht vorkommen, da die Kontrolle erst nach Ende der Aktion zurückgegeben wird. Es können lediglich dann Konflikte auftreten, wenn während einer solchen Simulationsoperation eine Taste auf der realen Tastatur betätigt wird. Dies löst eine Interruptroutine aus, die ein Zeichen in den Tastaturpuffer einträgt. Man kann diese Routine durch eine eigene ersetzen, die ein globales Flag als Semaphor verwendet und so Konflikte zwischen simulierter und echter Tastatureingabe verhindert.[4]

4.1.2 „Handlungsvorschrift“ als Skript

Als zweiten zentralen Punkt hatte wurde die Notwendigkeit einer Handlungsvorschrift genannt. Eine solche Vorschrift muß das Verhalten des Programmbenutzers algorithmisch beschreiben, so daß Virtus ihn simulieren kann. Der Programmierer der Handlungsvorschrift

[4] Die vorgestellte Version von Virtus macht von dieser Möglichkeit allerdings keinen Gebrauch.

muß also in geeigneter Weise die Umgebungsbedingungen (im wesentlichen: Bildschirmanzeige) abfragen und bestimmte Reaktionen (Tastatureingaben) programmieren können. Einfache Schleifenkonstrukte sind ebenfalls notwendig.

Damit Virtus möglichst universell einsetzbar ist, wurde es so gestaltet, daß eine Handlungsvorschrift – der Kürze halber ab jetzt „Skript" – beim Start des Programms aus einer Datei eingelesen werden kann. Die Skripte werden in einer rudimentären, an BASIC angelehnten Sprache geschrieben.[5] Ein Beispiel für ein ganz einfaches Skript ist:

```
LABEL Start
   IF CURSOR AT 10, 8
      SENDKEY "test" #13
      GOTO Weiter
   END IF
   GOTO Start

LABEL Weiter
   IF "Beendet" AT 11, 8
      SENDKEY F10
      HALT
   END IF
   GOTO Weiter
```

Dieses Beispiel enthält zwei Schleifen; in der ersten bleibt das Programm, bis der Cursor an die Position (8,10) bewegt wird. Dann wird das Wort „test", gefolgt von der Enter-Taste, eingegeben, und das Programm wartet so lange, bis an der Bildschirmposition (11,8) das Wort „Beendet" erscheint. Passiert das, wird die Taste F10 „gedrückt", und Virtus beendet sich.

Natürlich bedarf es noch einer Anzahl weiterer Konstrukte, um Virtus sinnvoll einsetzen zu können, aber das Prinzip und die Syntax der Steuersprache sollten dem interessierten Leser nun klar sein. Eine detaillierte Sprachbeschreibung folgt in Abschnitt 4.3 und in Anhang A.

4.1.3 Speicherresidentes Programm

Unter DOS ist kein echtes Multitasking möglich. Daher verwendet man, wenn man zwei (oder mehrere) Programme quasi-parallel ablaufen lassen will, sogenannte TSR-Programme (Terminate-and-stay-resident), die über Interrupts regelmäßig oder bei Eintreten bestimmter Ereignisse aufgerufen werden.

---

[5] Vor dem Einlesen durch Virtus muß ein Skript jedoch „kompiliert" werden, vgl. Abschnitt 4.2.1.

Virtus klinkt sich in den Interrupt 29 ein, der 18,2mal in der Sekunde von DOS aufgerufen wird und eigens als Aufhänger für speicherresidente Programme vorgesehen ist.[6] Bei jedem dieser Aufrufe führt Virtus genau einen Befehl aus dem Skript aus oder sendet eine Taste; das ferngesteuerte Programm sollte möglichst nicht zu lange unterbrochen werden, um eventuell dort aktive Timer-Routinen nicht durcheinanderzubringen.

Wenn Virtus aufgerufen wird und die Initialisierung der Variablen vorgenommen hat, beendet es sich nicht wie ein normales Programm (das danach aus dem Speicher entfernt wird), sondern durch einen Aufruf des DOS-Interrrupts 39 („Keep-Interrupt"). Das Programm wird beendet, bleibt jedoch im Speicher, so daß der zuvor auf eine eigene Routine „umgebogene" Interruptvektor des Interrupts 29 es von nun an regelmäßig aktiviert.

Ein vollwertiges speicherresidentes Programm sollte auch über eine Möglichkeit verfügen, sich selbst nach getaner Arbeit wieder aus dem Speicher zu entfernen. Virtus tut das in der vorliegenden Version nicht; es rekonstruiert lediglich den alten Interruptvektor, so daß es nach einem HALT-Befehl im Skript nicht mehr aufgerufen wird. Im Speicher liegt es dann aber immer noch – man kann eines der verbreiteten Hilfsprogramm wie „RamFrei" benutzen, um es aus dem Speicher zu entfernen.

Ein besonderes Problem stellt der Aufruf von DOS-Funktionen (Interrupts) durch speicherresidente Programme dar; da DOS nicht reentrant ist, würde das System abstürzen, wenn durch den Timer-Interrupt eine DOS-Funktion unterbrochen wird, und Virtus in diesem Interrupt hängend wieder eine DOS-Funktion benutzen möchte. DOS-Funktionen werden beispielsweise dann unbedingt benötigt, wenn man auf das Dateisystem zugreift.

Man kann auch diese Probleme durch ein Semaphorkonzept (in Kombination mit geschicktem Sichern von Teilen des Stacks) umgehen. Bei Virtus wurde anders – einfacher – vorgegangen. Eventuelle Ausgaben werden in einen reservierten Bereich des Hauptspeichers geschrieben. Will man sie nach Ablauf des Skripts auslesen, startet man Virtus erneut von der DOS-Befehlszeile aus; dann werden die Daten in eine Datei kopiert. Dies ist problemlos möglich, weil Virtus in diesem Augenblick ja schon nicht mehr „speicherresident", sondern als ganz normales Programm läuft. Damit sich die speicherresidente und die von der Befehlszeile aufgerufene Instanz von Virtus untereinander über den benutzten Speicherbereich „verständigen" können, verwendet Virtus einige Bytes im Hauptspeicher als globale Variablen – und „global" bedeutet hier, daß die Daten auch zwischen mehreren Programmaufrufen beibehalten werden. DOS stellt einen kleinen Speicherbereich (Segment 0, Offset hex 4F0 bis hex 4F6) eigens für solche Zecke zur Verfügung, und Virtus macht reichlich Gebrauch davon.[7]

---

[6] Die vorgestellte Version ist dabei sogar recht rücksichtslos und sichert den alten Interruptvektor nicht. Speicherresidente Programme, die sich schon vorher an diesen Interrupt geknüpft haben, werden also deaktiviert, solange Virtus läuft. Zu Interrupts vgl. Tischer 1989.

[7] Natürlich hat es fatale Auswirkungen auf die Stabilität des Systems, wenn ein anderes speicherresidentes Programm das gleichfalls tut; ein Abstimmungskonzept existiert nämlich nicht. Da Virtus aber meist auf einem eigens dafür eingerichteten Rechner laufen wird, kann man das in diesem Fall in Kauf nehmen – der Virtus-Benutzer muß sein System eben anpassen.

4.2 Realisierung: Virtus-Precompiler und Virtus-Interpreter

Speicherresidente Programme siedeln sich unter DOS im knappen Speicherbereich diesseits der 640 KB-Grenze an. Ihr Speicherbedarf sollte daher möglichst gering sein, so daß genug RAM für das fernzusteuernde Programm übrigbleibt.

Um den etwas aufwendigen Vorgang des Parsens des Virtus-Skripts (Aufteilung in einzelne Befehle und Argumente – „Tokens“ – und Auffinden von Syntaxfehlern) nicht dem speicherresidenten Virtus-Programm aufzubürden, wurden diese Vorgänge in ein eigenes Programm ausgelagert, den „Precompiler“.

4.2.1 Der „Precompiler“

Ein wirklicher Compiler ist der „Precompiler“ nicht, denn er prüft lediglich die syntaktische Korrektheit und wandelt die Klartext-Befehle in kürzere, binäre Codes um. Dabei entstehen aus einer Source-Datei eine binäre Programmdatei und ein Listing, das die Umsetzung dokumentiert. Das Beispiel aus Abschnitt 4.1.2 wird wie folgt „compiliert“:

```
Offset Bytes                              Source
   1                                      (LABEL Start)
   1   006 010 008                        (IF CURSOR AT 10, 8)
   4   009 005 116 000 101 000 115 000
       116 000 013 000                    (SENDKEY "test" #13)
  16   011 000 000                        (GOTO Weiter)
  19   008                                (END IF)
  20   011 001 000                        (GOTO Start)
  23                                      (LABEL Weiter)
  23   015 007 066 101 101 110 100 101
       116 011 008                        (IF "Beendet" AT 11, 8)
  34   009 001 000 068                    (SENDKEY F10)
  38   012                                (HALT)
  39   008                                (END IF)
  40   011 023 000                        (GOTO Weiter)
```

In der ersten Spalte sieht man die Byte-Postion in der erzeugten Binärdatei, in der zweiten Spalte die Bytes, die aus dem in der „Source“-Spalte stehenden Befehl gebildet wurden. Jedem Befehl ist ein Byte-Code zugeordnet, und zu jedem Befehl gehören eine feste Anzahl Argumente. Zwei Zeilen seien als Beispiele herausgegriffen:

```
4   009 005 116 000 101 000 115 000
    116 000 013 000                  (SENDKEY "test" #13)
```

Der SENDKEY-Befehl hat den Befehlscode 9. Es folgt die Anzahl der Zeichen (5), dann alle Zeichen hintereinander, wobei die Zwei-Byte-Folge angegeben wird, die im Tastaturpuffer

abgelegt werden muß, um dieses Zeichen zu erzeugen. „116 000“ ist beispielsweise ein kleines T (alle „normalen“ Tasten haben als zweite Ziffernfolge 000).

```
23   015 007 066 101 101 110 100 101
     116 011 008                        (IF "Beendet" AT 11, 8)
```

Hier lautet der Befehlscode für das IF...AT-Konstrukt 15. Es folgen ein Zeichenzähler (7), dann sieben Zeichencodes (als einfache ASCII-Werte), schließlich noch die Zeilen- und Spaltenangabe.

Labels werden in der binären Programmdatei nicht gespeichert; stattdessen werden GOTO-Befehle direkt mit der numerischen Sprungadresse versehen. Der Befehl „GOTO Weiter“ (letzte Zeile) enthält nun als Sprungadresse die 23. So kann der Precompiler auch fehlende oder doppelte Sprungmarken auffinden.

Der Precompiler ist in BASIC programmiert. Der Aufruf geschieht einfach mit „virtusc“, danach können die benötigten Daten (Name der Quelldatei, Name der Ausgabedatei) am Prompt eingegeben werden. Eventuelle Fehler werden sofort angezeigt.

4.2.2 Der Virtus-Interpreter

Der Virtus-Interpreter ist nun der eigentliche Kern des Systems, ein speicherresidentes Programm, das eine binäre Programmdatei einliest und ausführt. Die Realisierung erfolgte in Turbo Pascal[8]. Wie bei speicherresidenten Programmen üblich, besteht der VIRTUS-Interpreter aus einem nicht-residenten Teil, der die Initialisierung vornimmt und einen Interrupt-Handler auf den speicherresidetnen Teil setzt, sowie dem speicherresidenten Teil, der die eigentliche Aufgabe erfüllt, das Skript mit jedem Aufruf einen Schritt abzuarbeiten.

„Ein Schritt“ bedeutet hierbei, daß in der Regel bei jedem Aufruf des speicherresidenten Teils ein Befehl aus dem VIRTUS-Skript verarbeitet wird. Es gibt allerdings einige Ausnahmen von dieser Regel: So benötigen einfache Konstruktionselemente wie ELSE und END IF keinen Rechenschritt. Der LABEL-Befehl wird bereits beim Kompilieren reduntant und befindet sich daher gar nicht mehr im binären Skript, und VAR-Befehle am Anfang des Programms, die nur der Definition von Variablen dienen, werden bereits vom Initialisierungsteil des VIRTUS-Interpreters verarbeitet.

Auf der anderen Seite erfordern alle Befehle, die etwas in den Tastaturpuffer schreiben, so viele Schritte, wie sie Zeichen dorthin ausgeben – der Tastaturpuffer wird also nicht en bloc gefüllt, sondern stets Zeichen für Zeichen. Bei einem WAIT-Befehl oder wenn der Tastaturpuffer voll ist und VIRTUS dennoch hineinschreiben soll, wird die Ausführung des Programms entsprechend lange angehalten.

---

[8] Zu speicherresidenten Programmen in Turbo Pascal vgl. Schäpers 1991.

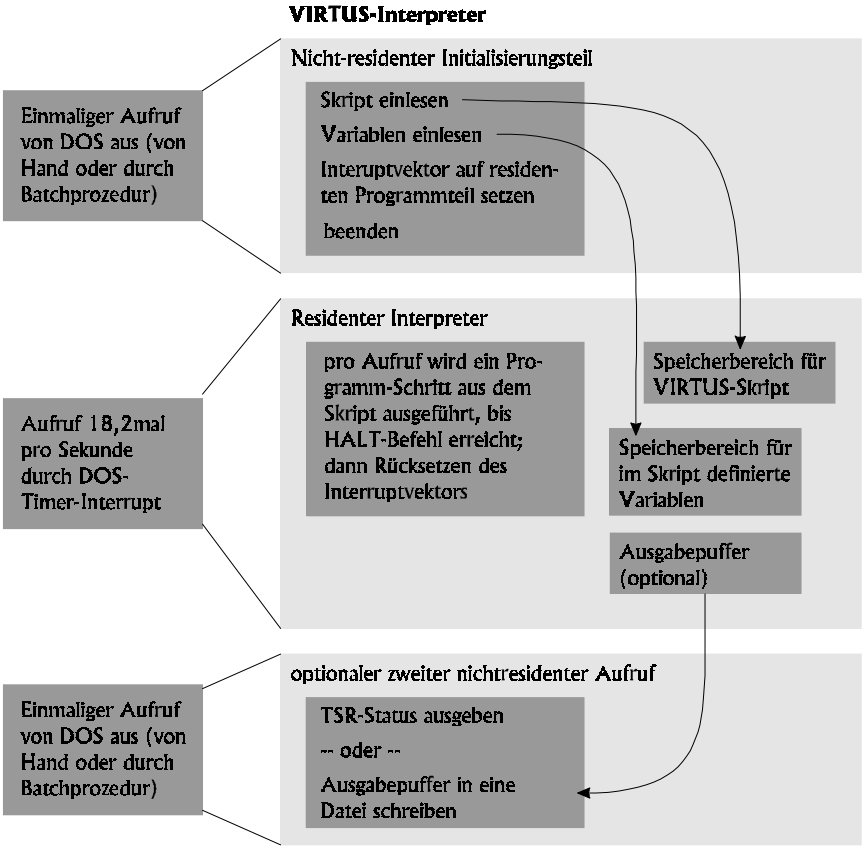

Abbildung 2: Funktionsweise des VIRTUS-Interpreters

Da die Größe des Speicherbereichs, den der VIRTUS-Interpreter für Variablen bereithält, bereits im Quelltext durch Konstanten festgelegt werden muß, ist es für maximale Effizienz erforderlich, den VIRTUS-Interpreter neu zu übersetzen, wenn man die erlaubte Anzahl von Variablen (oder Variablensätzen) verändern möchte.

Der VIRTUS-Interpreter kann, wie bereits in Abbildung 2 angedeutet, auf verschiedene Arten von der DOS-Befehlszeile aufgerufen werden:

```
virtus [-p:progname] [-v:varfile] [-n] [h:text]
```

Startet VIRTUS speicherresident; `progname` ist der Name des Programms, das ausgeführt werden soll (bei Fehlen wird zur Eingabe eines Namens aufgefordert). `varfile` ist der Name der Variablen-Datei (vgl. Abschnitt 4.3.1); wird er weggelassen und enthält das Programm dennoch Variablen, wird zur Eingabe der Variableninhalte aufgefordert.

Während ein VIRTUS-Skript läuft, wird in der oberen linken Ecke des Bildschirms ein blinkendes Indikatorzeichen angezeigt; `-n` schaltet diese Funktion aus. `-h` kann benutzt werden, um statt des Indikatorzeichens einen Text am oberen Bildschirmrand anzuzeigen (z.B. „VIRTUS läuft – bitte Programm nicht abbrechen“ o.ä.).

```
virtus -?
```

Zeigt den aktuellen Status des Skripts an („Läuft“, „Beendet“ oder „Abbruch wegen Fehlers“). Ist nur sinnvoll, wenn zuvor gemäß der ersten Syntax ein Skript gestartet wurde.

```
virtus [-w [:outfilename]]
```

Schreibt den Inhalt des Ausgabepuffers, in den ein Skript mit den OUTPUT-Befehlen (vgl. Anhang A) schreiben kann, in eine Datei (wenn kein `outfilename` angegeben, direkt auf den Bildschirm). Dieser Aufruf ist nur sinnvoll, wenn zuvor gemäß der ersten Syntax ein Programm gestartet wurde und dieses inzwischen beendet (beim HALT-Befehl angekommen) ist. Das speicherresidente Programm kann nur auf sehr umständliche Weise Dateizugriffe auslösen, so daß bei der vorliegenden Implementation dieser erneute Aufruf als nicht-residentes Programm dazu nötig ist.

4.3 Details über die Virtus-Skriptsprache

Wie bereits erwähnt, handelt es sich bei der Virtus-Skriptsprache um eine einfache Skriptsprache mit recht beschränkten Möglichkeiten. Prinzipiell sind Erweiterungen der Sprache nicht problematisch; sie blähen jedoch den speicherresidenten Virtus-Interpreter auf, was im Interesse der universellen Verwendbarkeit zu vermeiden ist.

Im Anhang A findet sich eine Beschreibung aller in Virtus verwendbaren Befehle. Zwei wichtige Punkte seien hier bereits herausgegriffen:

4.3.1 Variablen und Variablendateien

Variablen können im Skript definiert und verwendet werden. Die Möglichkeiten bleiben jedoch weit hinter dem zurück, was gewöhnliche prozedurale Programmiersprachen bieten; so ist im Skript keine Zuweisung an Variablen möglich (die Variablenwerte müssen spätestens beim Programmstart feststehen), und der Inhalt einer Variablen kann nicht abgefragt und beispielsweise verglichen werden. Variableninhalte können jedoch an das gerade ferngesteuerte Programm gesandt werden, und es ist möglich, zu prüfen, ob eine Variable überhaupt einen Inhalt hat (vgl. Einträge zu VAR, IF DEF und SENDVAR in Anhang A).

Im Zusammenhang mit Variablendateien besteht die Möglichkeit, mehrere Sätze von Variablen zu definieren. Besitzt ein Programm z.B. fünf Variablen, könnte eine Variablendatei 100 Sätze enthalten, die aus je fünf Zuweisungen bestehen. Das Skript kann dann mittels des SHIFT-Befehls diese Sätze durchlaufen. So wird es beispielsweise im konkreten Fall des RailServers möglich, mehrere Bahnverbindungen hintereinander abzufragen, ohne das ferngesteuerte Programm und VIRTUS für jede einzelne Verbindung neu aufzurufen.

4.3.2 Strukturelemente

VIRTUS unterstützt nicht direkt Schleifenkonstrukte; um eine Schleife zu realisieren, muß eine Sprungmarke definiert und mit GOTO angesprungen werden (LABEL, GOTO). Zusammen mit dem Fehlen von Variablenabfragen und -zuweisungen bedeutet dies, daß eine Schleife mit festgelegter Anzahl Wiederholungen (in vielen Sprachen als FOR-Schleife be-

kannt) hier nicht möglich ist. Nur die Äquivalente zu WHILE- und REPEAT UNTIL-Schleifen lassen sich realisieren.

Der Einbau solcher fortgeschrittenen Schleifenkonstrukte in VIRTUS wäre zwar nicht problematisch; in der Regel werden sie allerdings bei der Definition einer Hadlungsvorschrift zum Fernsteuern eines Programms nicht benötigt.

Die wichtigsten Konstruktionen sind sicherlich die IF-Anweisungen, von denen es in VIRTUS 5 festgelegte Formen zur Abfrage bestimmter Zustände des ferngesteuerten Programms gibt (dem IF folgt nicht, wie sonst oft, ein beliebiger Ausdruck, sondern nur ganz bestimmte Konstruktionen sind zulässig).

Neben GOTO existiert auch das GOSUB/RETURN-Paar zur Realisierung einfacher Prozeduren.

5 Batch-Betrieb (Offline)

Der wichtigste Baustein des RailServers, die Software zur Fernsteuerung einer DOS-Anwendung, wurde nun vorgestellt. Im speziellen Fall des RailServers wird mit dem VIRTUS-Programm das Fahrplanauskunftssystem „HAFAS“, das auf der halbjährlich von der DB AG herausgegebenen Kursbuch-CDROM enthalten ist, gesteuert.

Dieses Kapitel soll jetzt beschreiben, auf welchem Wege genau eine Anfrage eines Internet-Benutzers bearbeitet wird. Der Vorgang der Auskunftserteilung wird dabei untergliedert in die Eingabe der Anfrage und ihre Weiterleitung an den RailServer-PC, ihre Bearbeitung dort und die Rücksendung der fertigen Auskunft.

5.1 Eingabe einer Anfrage durch den Benutzer

Der Internet-Benutzer kann eine Anfrage an den RailServer durch Absenden einer eMail an die Adresse `rail@rz.uni-karlsruhe.de` stellen; diese eMail muß einer bestimmten Form genügen. Ein Beispiel für eine gültige Anfrage wäre:

```
#VON: Karlsruhe Hbf
#NACH: Kleinkleckersdorf
#DATUM: 21.01.96
#ZEIT: 10:20
```

Es gibt dabei noch eine Anzahl weiterer Einstellmöglichkeiten, auf die jedoch an dieser Stelle nicht näher eingegangen werden soll, denn die eMail-Abfrage wird seit dem Boom des World Wide Web kaum noch genutzt. In 90% aller Fälle findet die WWW-Abfrageseite Anwendung, die unter dem URL `http://rail.rz.uni-karlsruhe.de/rail/` erreichbar und in Abbildung 3 dargestellt ist.

Diese Abfrageseite bedient sich einiger einfacher HTML-Eingabeelemente. Beim Betätigen der „Formular Abschicken“-Schaltfläche wird ein kleines Programm auf dem WWW-Server der Uni Karlsruhe aufgerufen, das nichts weiter tut, als aus den Angaben, die der Benutzer in den Eingabefeldern gemacht hat, eine eMail zu generieren, die dem oben geschilderten Format genügt, und diese an die Adresse `rail@rz.uni-karlsruhe.de` abzusenden.

Abbildung 3: Das RailServer-Eingabeformular im World Wide Web.

Hier der Quelltext der Abfrageseite (die <FORM>-Zeile, die den Programmaufruf auslöst, ist grau unterlegt):

```
<HTML>
<HEAD>
<TITLE>RailServer-WWW-Gateway (kurz)</TITLE>
<LINK REV="made" HREF="mailto:ule3@rz.uni-karlsruhe.de">
</HEAD>
<BODY>
<H1><img src="ice-icon.gif"> RailServer-Anfrage absenden</H1>
<HR>
Dieses Formular sendet automatisch eine Anfrage an den RailServer, wenn
Sie die "Formular abschicken"-Schaltfl&auml;che bet&auml;tigen. Sie
m&uuml;ssen
Ihre eMail-Adresse angeben, damit der Server Ihnen antworten kann.
<P>
<FORM ACTION="http://www.rz.uni-karlsruhe.de/cgi-bin/mailto/rail@rz.uni-
karlsruhe.de/AutomaticQuery-D/http://rail.rz.uni-karlsruhe.de/rail/rssnt-
d.html" METHOD=POST>
<TT>Ihre e-Mail-Adresse:</TT> <INPUT NAME="Reply-To" SIZE=45><BR>
<BR>
Finde eine Verbindung: (Bitte ae, oe, ue, ss benutzen)<BR>
```

```
<TT>von (Bahnhof): .....</TT> <INPUT NAME="#FROM" SIZE=22><BR>
<TT>nach (Bahnhof): ....</TT> <INPUT NAME="#TO" SIZE=22><BR>
<TT>Datum (tt.mm.jj): ..</TT> <INPUT NAME="#DATE" SIZE=9><BR>
<TT>Uhrzeit (hh:mm): ...</TT> <INPUT NAME="#TIME" SIZE=7><BR>
<INPUT TYPE=checkbox NAME="#NUR" VALUE="DIREKT">
   Nur Direktverbindungen;
<INPUT TYPE=checkbox NAME="#NUR" VALUE="FAHRRAD">
   nur mit Fahrradbef&ouml;rderung<BR>
   Zuggattungen:
<INPUT TYPE=checkbox NAME="#NICHT" VALUE="ICE"> kein ICE
<INPUT TYPE=checkbox NAME="#NICHT" VALUE="IC"> kein EC/IC
<INPUT TYPE=checkbox NAME="#NICHT" VALUE="IR"> kein IR/FD/D<P>
<INPUT TYPE=submit VALUE="Formular abschicken"><BR>
</FORM>
<A HREF="rsask2-d.html">Mehr Optionen</A> (aufwendigeres Formular)<BR>
<A HREF="railserv.html">Zur RailServer-Hauptseite</A>
</BODY></HTML>
```

Es werden also *sämtliche* Anfragen, ob mit der „alten“ Mail-Methode oder über das neue WWW-Gateway, schließlich unter der Mailadresse `rail@rz.uni-karlsruhe.de` gesammelt. Dort holt der RailServer-PC sie regelmäßig zur Weiterverarbeitung ab; Abbildung 4 verdeutlicht diesen Vorgang.

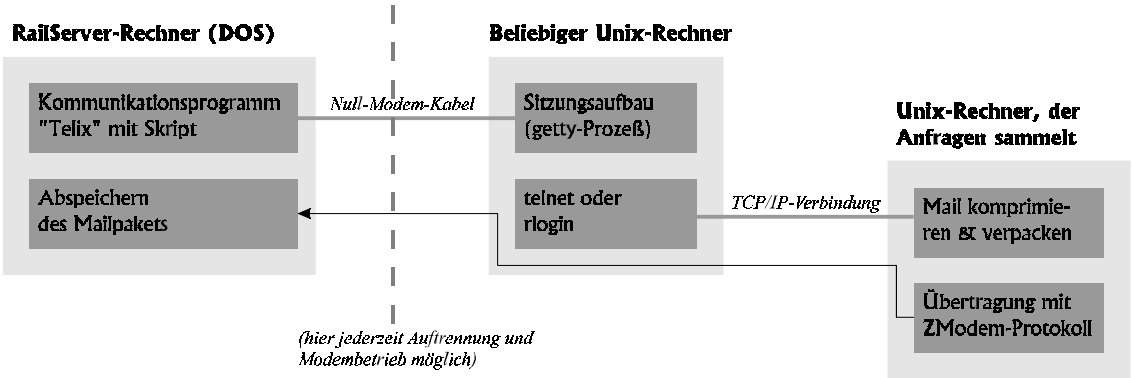

Abbildung 4: Genauer Ablauf der Übertragung der Anfragen

Dazu stellt der PC über ein handelsübliches Terminal-Programm und ein Nullmodem-Kabel die Verbindung zu einem beliebigen Unix-Rechner her, der mit dem Internet verbunden ist. Von diesem Rechner aus kann per telnet oder rlogin eine Internet-Verbindung zu dem Rechner aufgebaut werden, der die Mails an `rail@rz.uni-karlsruhe.de` sammelt. Dort wird ein Skript aufgerufen, das alle unnötigen Verwaltungsinformationen aus den Anfrage-eMails entfernt und das Datenvolumen mittels Kompression auf ein Minimum reduziert. Es entsteht ein Anfrage-Päckchen, das per ZModem-Protokoll über die bestehende Verbindung direkt auf die Festplatte des RailServer-PCs kopiert wird.

Dieses Verfahren mag auf den ersten Blick aufwendig erscheinen, ermöglicht jedoch eine sehr flexible Umgestaltung des Systems. Der RailServer-PC ist völlig ortsungebunden; er kann in räumlicher Nähe (wegen des erforderlichen Nullmodem-Kabels) eines beliebigen Unix-Rechners, der als „Terminalserver“ für den Internet-Zugang fungiert, stehen. Außerdem

– hier offenbart sich die historische Entwicklung des Systems – kann das Nullmodem-Kabel jederzeit durch eine echte Modemstrecke ersetzt werden. Der PC benötigt dann nur noch Zugang zu einem Telefon-Anschluß, und die Internet-Verbindung würde über den normalen Modemserver des Rechenzentrums gewährleistet.

5.2 Verarbeitung der Anfrage

Nach der erfolgreichen Übertragung und Dekompression des Anfrage-Datenpakets an den RailServer-PC liegt nun eine Datei vor, die alle Anfragen hintereinander enthält, getrennt durch einen Separator.

```
☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺
From www@rz.uni-karlsruhe.de Mon Aug  7 15:30:16 1995
Date: Mon, 7 Aug 1995 15:29:55 +0200
From: www@rz.uni-karlsruhe.de (WWW Mail Gateway)
Subject: AutomaticQuery-E
Reply-To: tzf@techprt.co.uk

#FROM: Paris
#TO: Berlin
#DATE: 01-02-96
#TIME: 13:00
☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺
From www@rz.uni-karlsruhe.de Tue Aug  8 09:50:46 1995
Date: Tue, 8 Aug 1995 09:50:35 +0200
From: www@rz.uni-karlsruhe.de (WWW Mail Gateway)
Subject: AutomaticQuery-E
Reply-To: rgaenne@gwdg.de

#FROM: Goettingen
#TO: Heilbronn
#DATE: 10.03.96
#TIME: 12.00
☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺
From www@rz.uni-karlsruhe.de Tue Aug  8 17:12:49 1995
Date: Tue, 8 Aug 1995 17:12:34 +0200
From: www@rz.uni-karlsruhe.de (WWW Mail Gateway)
Subject: AutomaticQuery-D
Reply-To: dotex@zedat.fu-berlin.de

#FROM: Berlin Schoeneweide
#TO: Luebeck Hbf
#DATE: 10.01.96
#TIME: 08:00
#TOTIME: 16:00
```

5.2.1 Homogenisierung der Anfragedaten

Der direkten Einspeisung dieser Daten in das Bahn-Informationsprogramm HAFAS vermittels Virtus steht nun jedoch (neben der notwendigen Formatanpassung) die Tatsache im Wege, daß die Bahnhofsnamen und Datums- und Zeitangaben hier noch völlig unbearbeitet vorliegen, so, wie der Benutzer sie eingegeben hat. Dies eröffnet eine Vielzahl möglicher Fehlerquellen:

- Zeit- und Datumsangaben können ungültig sein,
- das Datum kann außerhalb der aktuellen Fahrplanperiode liegen,
- Bahnhofsnamen können ungültig (=nicht in der Datenbank enthalten sein), z.B. auch aufgrund der Verwendung und fehlerhaften Übertragung von Umlauten
- Bahnhöfe können nicht-eindeutig angegeben sein (so besitzt Berlin fast 100 Bahnhöfe, die Angabe „Berlin“ reicht allein nicht aus).
- Da nicht nur vom WWW-Skript erzeugte, sondern auch „von Hand“ eingesandte Mails hier auflaufen, können durchaus noch Signaturen oder ähnliche nicht relevante Daten in der Mail enthalten sein; zuweilen gehen auch natürlichsprachliche Anfragen ein, die einen Menschen hinter der Mailadresse vermuten.

Die Einspeisung solch fehlerhafter Daten in das HAFAS-Auskunftsprogramm hätte zur Folge, daß das Programm am Schirm mit einer Fehlermeldung reagiert, die umständlich durch Virtus erkannt und angemessen verarbeitet werden müßte.

Daher wird die Anfrage-Datei von einem Programm namens „PROCESS1“ einem ersten Bearbeitungsschritt unterzogen, in dem alle Angaben auf Gültigkeit geprüft und, wo möglich, „zurechtgebogen“ werden, so daß sie ohne Fehler in HAFAS eingespeist werden können. Abbildung 5 veranschaulicht diesen Vorgang.

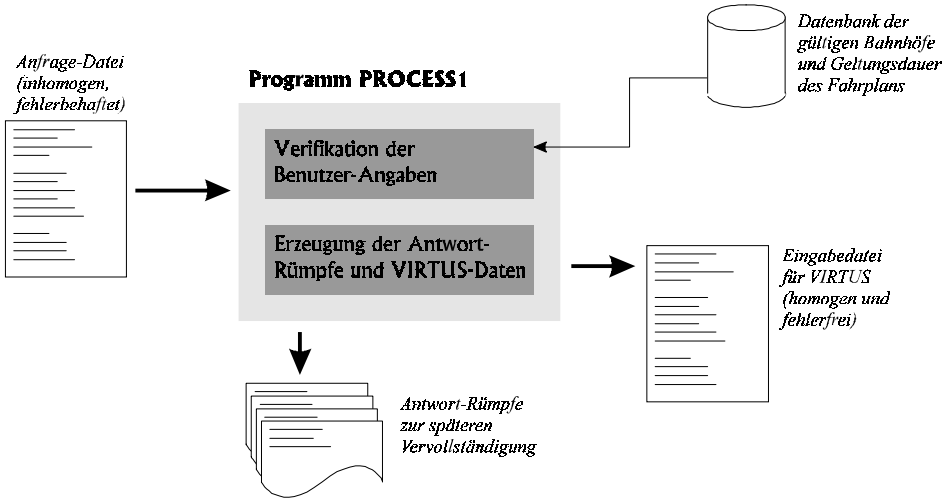

Abbildung 5: Homogenisierung der Anfragedaten mit PROCESS1

PROCESS1 erzeugt also in der Hauptsache die Variablen-Eingabedatei für VIRTUS; sie sieht wie folgt aus (diese Datei wurde aus dem Beispiel von Seite 23 generiert):

```
FROMSTATION=8010041
TOSTATION=8000237
TRAVELDATE=100196
DURATION=0800
OUTFILENAME=1.PRN
TRAVELTIME=0800
DEFAULT=Y
SHIFT
FROMSTATION=8000128
TOSTATION=8000157
TRAVELDATE=100396
DURATION=0200
OUTFILENAME=2.PRN
ENGLISH=YES
TRAVELTIME=1200
DEFAULT=Y
SHIFT
FROMSTATION=8700011
TOSTATION=8010255
TRAVELDATE=010296
DURATION=0200
OUTFILENAME=3.PRN
ENGLISH=YES
TRAVELTIME=1300
DEFAULT=Y
```

Die Bezeichnungen der Variablen („FROMSTATION“ usw.) sind in dem Skript festgelegt, das Virtus später abarbeitet, wenn es mit diesen Daten HAFAS „fernbedient“ (vgl. das Rail-Server-Skript im Anhang B). Man sieht, daß diese Variablendatei ein strenges Format einhält. Insbesondere fällt die Umsetzung der Bahnhofsnamen in einen Nummerncode (die sogenannten „Internationalen Bahnhofsnummern“ oder IBN) auf. Das PROCESS1-Programm verfügt über die Bahnhofsliste, die auch HAFAS verwendet, und kann daher die angegebenen Bahnhofsbezichnungen gleich in diesen Code umsetzen, der von HAFAS als Eingabe akzeptiert wird. Das wäre zwar nicht erforderlich – Bahnhofsnamen, sofern sie eindeutig sind, funktionierten ebenso – aber die Verarbeitung von Nummern ist weniger fehleranfällig.

Diese Datei enthält auch zu jeder Anfrage einen Dateinamen („OUTFILENAME“), unter dem HAFAS später die Ergebnisse der Anfrage speichern wird.

Parallel zu diesen Dateibezeichnungen (1.PRN, 2.PRN usw.) erzeugt PROCESS1 bereits die „Rümpfe“ der später zu versendenden Antwortmitteilungen unter den Namen 1.RES, 2.RES usw. (RES für „Response“). Da PROCESS1 für die Abbildung der vom Benutzer angegebenen Bahnhofsnamen auf die tatsächlich in der Datenbank vorhandenen Einträge zuständig ist, kann es in den entsprechenden Antwort-Rumpf gleich diesbezügliche Mitteilungen an den Benutzer eintragen. In Frage kommen hier:

- Ein Bahnhof ist nicht eindeutig, es wurde ein geeigneter ausgewählt (bei der Eingabe von „Frankfurt“ wird beispielsweise standardmäßig der Hauptbahnhof gewählt).

- Ein Bahnhofsname ist ungültig, konnte aber anhand einer eingebauten Übersetzungstabelle umgesetzt werden („Cologne“ wird zu „Koeln Hbf“).
- Ein Bahnhofsname ist ungültig, aber ein ähnlichklingender konnte gefunden werden (hierzu wird in PROCESS1 eine Ähnlichkeitssuche nach dem Verfahren von Levenshtein[9] eingesetzt).
- Datum und/oder Zeit waren ungültig und werden auf geeignete Werte gesetzt.
- Ein Bahnhofsname fehlt oder ist falsch und konnte nicht auf einen ähnlichen abgebildet werden – in diesem Fall wird nicht nur ein Antwort-Rumpf, sondern gleich eine komplette Antwort erzeugt, und für die betreffende Nummer findet sich kein Eintrag in der Virtus-Datei, da hier keine Auskunft generiert werden kann.

Im bereits mehrfach strapazierten Beispiel werden also folgende drei Antwort-Rümpfe generiert:

(1.res)

```
From: rail@rz.uni-karlsruhe.de (RailServer Germany)
Reply-To: ule3@rz.uni-karlsruhe.de (Frederik Ramm)
Errors-To: null@rz.uni-karlsruhe.de
To: dotex@zedat.fu-berlin.de
Subject: Zugverbindung BERLIN SCHOENEWEIDE-LUEBECK HBF
Message-Id: <rail.1@rz.uni-karlsruhe.de>

Liebe Netzteilnehmerin, lieber Netzteilnehmer,

   hier die Antwort auf Ihre RailServer-Anfrage.

Verbindungen von Berlin Schoeneweide nach Luebeck Hbf
Abfahrt am 10. Januar 1996, innerhalb von ca. 08 Std 00 Min nach 08:00 Uhr
```

(2.res)

```
From: rail@rz.uni-karlsruhe.de (RailServer Germany)
Reply-To: ule3@rz.uni-karlsruhe.de (Frederik Ramm)
Errors-To: null@rz.uni-karlsruhe.de)
To: rgaenne@gwdg.de
Subject: Train Connection GOETTINGEN-HEILBRONN
Message-Id: <rail.2@rz.uni-karlsruhe.de>

Dear Net.Citizen,

   this is the answer to your RailServer query.

TO station (HEILBRONN) ambiguous. Matches found:
         - Heilbronn Hbf [8000157], 39km N of Stuttgart
         - Heilbronn Karlstor [8002697], 40km N of Stuttgart
```

[9] vgl. Ebner 1989.

```
            - Heilbronn Suelmertor [8002699], 41km N of Stuttgart
            (Instead of station names, you may use the number in brackets.)
            The system has selected the TO station for you: Heilbronn Hbf

    Connections from Goettingen to Heilbronn Hbf
    leaving on March 10th, 1996 within approx. 02h 00min after 12:00
```

(3.res)

```
    From: rail@rz.uni-karlsruhe.de (RailServer Germany)
    Reply-To: ule3@rz.uni-karlsruhe.de (Frederik Ramm)
    Errors-To: null@rz.uni-karlsruhe.de
    To:  tzf@techprt.co.uk
    Subject: Train Connection PARIS-BERLIN
    Message-Id: <rail.3@rz.uni-karlsruhe.de>

    Dear Net.Citizen,

       this is the answer to your RailServer query.

    TO station (BERLIN) ambiguous.
            99 stations match.
            Next time add the #VERBOSE line to your query to see them all.
            The system has selected the TO station for you: Berlin Hbf

    Connections from Paris Est to Berlin Hbf
    leaving on February 1st, 1996 within approx. 02h 00min after 13:00
```

Dem aufmerksamen Betrachter fällt auf, daß die Antworten teils auf Deutsch und teils auf Englisch generiert werden – dies wird durch die „Subject"-Zeile in der ursprünglichen Anfragedatei gesteuert. Je nachdem, ob der Teilnehmer die deutsche oder englische WWW-Formularseite ausfüllt, erhält er auch eine deutsche oder englische Antwort.

5.2.2 Einsatz von Virtus und HAFAS zur Erstellung der Auskunft

Nachdem nun eine fehlerfreie, homogenisierte Eingabedatei vorliegt, kann Virtus gestartet werden. Virtus liest nun zunächst das Skript ein, das die Fernbedienung von HAFAS steuert, und danach die von PROCESS1 erzeugte Variablendatei.

Die folgenden Abbildungen zeigen den typischen Verlauf einer Auskunftserteilung durch das HAFAS-Programm unter DOS. Im Anhang B ist das VIRTUS-Skript abgedruckt, das benutzt wird, um diesen Vorgang zu automatisieren.

Abbildung 6: Der HAFAS-Startbildschirm (der Cursor steht im markierten Eingabefeld)

Abbildung 6 zeigt die Startsituation. Sobald HAFAS geladen ist, zeigt es diese Eingabemaske an. Virtus wiederum erkennt die Beendigung des Ladevorgangs und die Eingabebereitschaft von HAFAS daran, daß der Cursor im Eingabefeld für den Startbahnhof steht.

Abbildung 7: Einstellung des Ausgabedateinamens

VIRTUS stellt das HAFAS-Programm als nächstes auf die gewünschte Sprache (deutsch oder englisch; nicht abgebildet) ein und legt dann fest, in welche Datei die Ausgabe geschrieben werden soll (Abbildung 7).

Abbildung 8: Eingabe des Start- und Zielbahnhofs

Dann werden Start-, Ziel- und ggf. auch Via-Bahnhof eingegeben. Virtus füttert HAFAS dabei mit den Bahnhofsnummern, wie in Abbildung 8 ersichtlich; HAFAS zeigt nach Abschluß der Eingabe allerdings nichtsdestotrotz den Klarnamen des Bahnhofs an.

Es folgen die Eingabe von Datum und Uhrzeit (nicht abgebildet). Dabei kann durch Eingabe einer Startzeit und einer Dauer ein Zeitfenster spezifiziert werden, innerhalb dessen Verbindungen gesucht werden sollen.

Abbildung 9: Zugtyp-Einstellungen in HAFAS

Falls keine „Produktwahl“ getroffen wurde (darunter versteht das HAFAS-Programm die An - oder Abwahl bestimtmer Zuggattungen oder auch die Auswahl „nur Direktverbindungen“, „nur Schlafwagen“ usw.), kann nun gleich mit den Tasten F2 oder F3 die Suche ausgelöst werden; anderenfalls erfolgt zunächst noch ein Sprung in das Produktwahl-Fenster, das in Abbildung 9 dargestellt ist.

Nach einer eventuellen Produktwahl wird durch Druck auf F3 die Vorwärts-Suche ausgelöst (Verbindungen, die zur angegebenen Zeit abfahren); F2 sucht Verbindungen, die zur angegebenen Zeit den Zielbahnhof erreichen. Abbildung 10 zeigt HAFAS beim Suchen einer Verbindung.

Abbildung 10: HAFAS bei der Berechnung einer Verbindung.

Können keine Verbindungen gefunden werden, erscheint eine Fehlermeldung, und Virtus löscht die Eingaben und springt sofort zur nächsten Abfrage (das PROCESS2-Programm erkennt dann an der fehlenden Ausgabedatei, daß keine Verbindung ermittelt werden konnte – vgl. Abschnitt 5.2.3).

Anderenfalls listet HAFAS die Verbindungen auf, und an dem in Spalte 3 der Zeile 7 erscheinenden #-Symbol (in Abbildung 11 noch im Hintergrund zu sehen) erkennt Virtus, daß

Verbindungen gefunden wurden. Es löst dann den Druckbefehl aus, und HAFAS „druckt“ die gefundene Verbindung in die zuvor angegebene PRN-Datei (Abbildung 11).

Abbildung 11: HAFAS druckt die gefundenen Verbindungen

Wenn noch weitere Anfragen vorliegen, löscht Virtus nun alle Eingabefelder und fährt mit der nächsten Anfrage fort; ansonsten wird HAFAS beendet, und für jeden Eintrag in der Virtus-Variablendatei sollte nun die entsprechende PRN-Datei vorhanden sein. Eine solche PRN-Datei sieht wie folgt aus (zur Beispielanfrage Göttingen/Hannover):

```
                    Timetable information Version:3.46
                          Timetable: DB 1995/96

                     from Göttingen to Heilbronn Hbf

 Station              Date     Arr.  Dep.  Train-No  Remarks
==========================================================================
 Göttingen            10.03.96       13:01 ICE  587  RE ZR
 Würzburg Hbf                 14:20 14:40 RE  4013  FB
 Heilbronn Hbf        10.03.96 16:13
 Duration:  3:12, runs always
 Tariffkm: 380; Price: 173,00 DM (1st) 117,00 DM (2nd)
-------
 Göttingen            10.03.96       13:22 ICE  577  RE ZR
 Heidelberg Hbf               15:51 15:57 SE  3387  FB
 Heilbronn Hbf        10.03.96 17:21
 Duration:  3:59, runs always
 Tariffkm: 411; Price: 178,00 DM (1st) 122,00 DM (2nd)
-------
 Göttingen            10.03.96       13:53 ICE  797  RE ZR
 Stuttgart Hbf                17:06 17:15 RE  3856  FB
 Heilbronn Hbf        10.03.96 18:01
 Duration:  4:08, runs always, not 24., 31. Dec
 Tariffkm: 499; Price: 208,00 DM (1st) 142,00 DM (2nd)
-------
 Göttingen            10.03.96       15:01 ICE  589  RE ZR
 Würzburg Hbf                 16:20 16:40 RE  4017  FB
 Heilbronn Hbf        10.03.96 18:13
```

```
 Duration:  3:12, runs always
 Tariffkm: 380; Price: 173,00 DM (1st) 117,00 DM (2nd)

RE: Reservation requested
ZR: Dining car
FB: Accompanying bicycle
```

Damit ist die Errechnung der Verbindung abgeschlossen. Wie der nächste Abschnitt zeigt, sind noch einige Schönheitskorrekturen erforderlich, bevor das Ergebnis als Antwort an den Internet-Benutzer gesandt werden kann.

5.2.3 Nachbearbeitung der HAFAS-Ergebnisse durch PROCESS2

Nach Erstellung der eigentlichen Bahnverbindungen sind noch einige abschließende Schritte notwendig:

- Die PRN-Datei muß von Umlauten, Leerzeilen, Seitenwechseln usw. bereinigt werden;
- einige Fehler, die HAFAS bei der englischen Übersetzung macht, müssen korrigiert werden;
- die Bahnverbindung muß an den passenden Antwort-Rumpf (der bereits von PROCESS1 erzeugt wurde, vgl. Abschnitt 5.2.1) angehängt werden;
- falls HAFAS keine Verbindung fand (also keine PRN-Datei existiert), muß eine entsprechende Fehlermeldung an den Antwort-Rumpf angefügt werden.

All diese Aufgaben übernimmt das Programm PROCESS2.

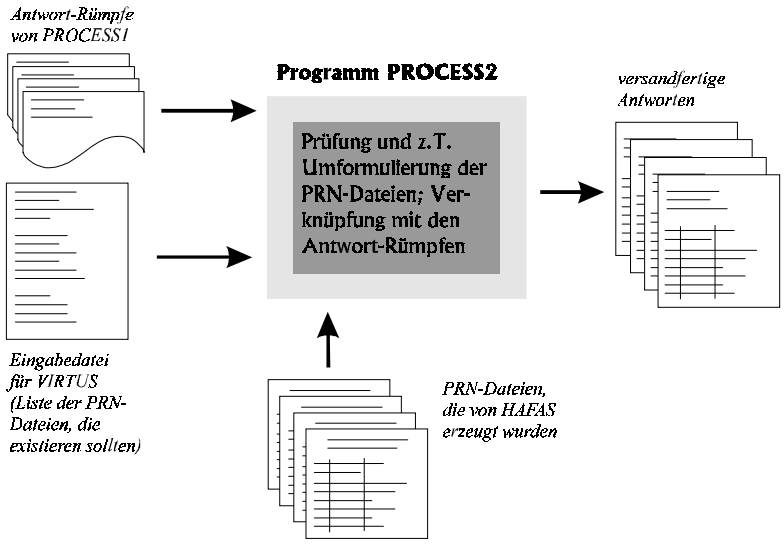

Abbildung 12: Nachbearbeiten der Ergebnisse mit PROCESS2

Nach Ablauf von PROCESS2 liegt nunmehr die fertige Antwort als RES-Datei vor (hier wieder die Göttingen-Anfrage, 2.RES):

```
From: rail@rz.uni-karlsruhe.de (RailServer Germany)
Reply-To: ule3@rz.uni-karlsruhe.de (Frederik Ramm)
Errors-To: null@rz.uni-karlsruhe.de
To: rgaenne@gwdg.de
Subject: Train Connection GOETTINGEN-HEILBRONN
Message-Id: <rail.2@rz.uni-karlsruhe.de>

Dear Net.Citizen,

   this is the answer to your RailServer query.

TO station (HEILBRONN) ambiguous. Matches found:
         - Heilbronn Hbf [8000157], 39km N of Stuttgart
         - Heilbronn Karlstor [8002697], 40km N of Stuttgart
         - Heilbronn Suelmertor [8002699], 41km N of Stuttgart
         (Instead of station names, you may use the number in brackets.)
         The system has selected the TO station for you: Heilbronn Hbf

Connections from Goettingen to Heilbronn Hbf
leaving on March 10th, 1996 within approx. 02h 00min after 12:00

                              | 24h format! |
 Station               (date)  |arrive depart| TrainType Comments
================================================================================
 Goettingen           10.03.96 |       13:01 | ICE  587  RE ZR
 Wuerzburg Hbf                 | 14:20 14:40 | RE  4013  FB
 Heilbronn Hbf        10.03.96 | 16:13       |
 Duration:  3:12, runs always
 2nd class fare: DM 117.00 (1st: 173.00); dist.: 380 km
--------------------------------------------------------------------------------
 Goettingen           10.03.96 |       13:22 | ICE  577  RE ZR
 Heidelberg Hbf                | 15:51 15:57 | SE  3387  FB
 Heilbronn Hbf        10.03.96 | 17:21       |
 Duration:  3:59, runs always
 2nd class fare: DM 122.00 (1st: 178.00); dist.: 411 km
--------------------------------------------------------------------------------
 Goettingen           10.03.96 |       13:53 | ICE  797  RE ZR
 Stuttgart Hbf                 | 17:06 17:15 | RE  3856  FB
 Heilbronn Hbf        10.03.96 | 18:01       |
 Duration:  4:08, runs always, not 24., 31. Dec
 2nd class fare: DM 142.00 (1st: 208.00); dist.: 499 km
--------------------------------------------------------------------------------
 Goettingen           10.03.96 |       15:01 | ICE  589  RE ZR
 Wuerzburg Hbf                 | 16:20 16:40 | RE  4017  FB
 Heilbronn Hbf        10.03.96 | 18:13       |
 Duration:  3:12, runs always
 2nd class fare: DM 117.00 (1st: 173.00); dist.: 380 km

Legend:
 RE: Reservation strongly recommended
```

```
 ZR: Dining car
 FB: Bicycles allowed

--
RailServer ## maintained by Frederik Ramm ## eMail ule3@rz.uni-karlsruhe.de
```

5.3 Rücksendung der Auskunft an den Netz-Benutzer

Als letzter Schritt der Einbindung ist nun noch die Rücksendung der Mail an den anfragenden Netzteilnehmer zu betrachten. Diese gestaltet sich einfach: Alle Antwortdateien (.RES) werden zu einer Archivdatei zusammengefaßt und dabei komprimiert, und die Übertragung erfolgt analog zur Abholung der Anfragen aus dem Netz (Abbildung 13).

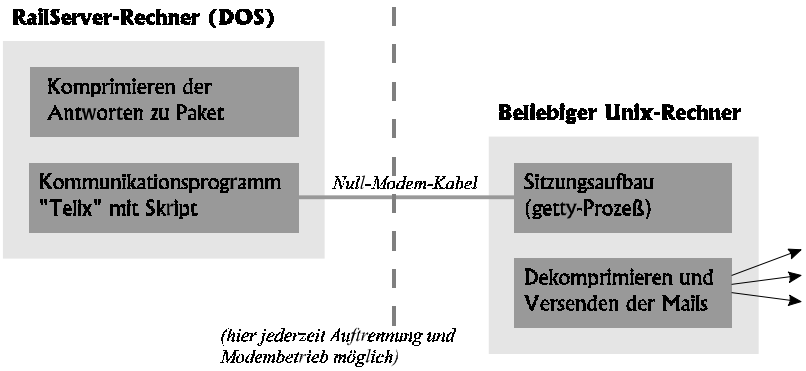

Abbildung 13: Versand der Antworten

Auf dem Unix-Rechner kommt dabei ein einfaches Shell-Skript zum Einsatz, das die eMails unter Verwendung des Unix-Standardprogramms „sendmail“ verschickt[10].

[10] Zu „sendmail“ vgl. Costales 1993.

6 Online-Betrieb

Das Kapitel 5 hat gezeigt, wie der RailServer heute bereits als Offline-Lösung eingesetzt wird („offline“, weil der Benutzer die Antwort nicht sofort im WWW sieht, sondern einige Zeit später per Mail erhält). In diesem Kapitel soll nun ein Weg aufgezeigt werden, wie – ebenfalls unter Verwendung der in Kapitel 4 vorgestellten Virtus-Software – auch ein Online-Einsatz mit sofortiger Antwort möglich ist.[11]

6.1 Veränderte Benutzerschnittstelle für Online-Antwendung

Noch bevor Überlegungen zur Implementierung angestellt werden, kann einiges über die Schnittstelle zum Benutzer gesagt werden. Bei der Offline-Lösung ist der Informationsfluß starr vorgegeben: Der Benutzer schickt eine Anfrage ab, diese wird – u.U. deutlich zeitversetzt – auf einem PC bearbeitet, und die Ergebnisse gehen an den Benutzer zurück, der sie später in seiner Mailbox findet.

Insbesondere ist bei Unklarheiten bezüglich der Bahnhofsnamen usw. keinerlei Rückfrage möglich: Der RailServer muß die Anfragen nach dem „Friß oder stirb“-Prinzip beantworten. Nach Möglichkeit sollte er auch aus den abstrusesten Daten noch eine vernünftige Auskunft generieren. So wurde mit umfangreichen Ähnlichkeitssuche-Algorithmen und mit Übersetzungstabellen versucht, möglichst jede Bahnhofs-Eingabe einem gültigen Bahnhofsnamen zuzuordnen.

6.1.1 CGI-Programme

Ein interaktives System hat da ganz andere Möglichkeiten: Die Eingabe des Benutzers kann sofort an ein CGI-Skript[12] auf dem WWW-Server über tragen werden, das sie prüft und, sofern möglich, gleich verarbeitet.

Bei ungültiger Eingabe eines Bahnhofsnamens wird dem Benutzer einfach eine Liste aller in Frage kommenden (z.B. mit den gleichen Buchstaben beginnenden) Bahnhöfe präsentiert, aus der er dann einen auswählen muß.

Das gleiche gilt prinzipiell auch für ungültige Datumseingaben usw., wobei man jedoch schnell an die technischen Grenzen des WWW stößt. Jede Interaktion zwischen der Software, die auf dem WWW-Server ausgeführt wird, und dem Benutzer, der am „anderen Ende“ seinen Browser bedient, erfordert den Aufbau einer HTTP-Verbindung über das Internet und kann auch bei kleinen Datenmengen zeitaufwendig sein. Daher gilt es, ein ausgewogenes Mittel zu finden zwischen dem Mehr an Komfort, das eine enge Interaktion bringt, und dem

[11] Die Realisierung dieses Ansatzes steht indes – aus Zeitmangel – noch aus.

[12] Common Gateway Interface, ein standardisiertes Interface für Programme, die auf WWW-Servern ausgeführt werden, um Benutzereingaben zu verarbeiten. Vgl. Ramm 1995.

Verlust an Geschwindigkeit im Vergleich zu einer einfachen Seite mit Eingabefeldern, wie sie in Abbildung 3 zu sehen war.

6.1.2 Java und JavaScript

Die Programmiersprache Java ist ein plattformunabhängiges Derivat von C++ und wurde speziell für den Einsatz im WWW angepaßt. Im Gegensatz zu CGI-Programmen, die stets auf dem WWW-Server ausgeführt werden, werden Java-Programme zusammen mit einer HTML-Seite an den Browser übertragen und dann von diesem ausgeführt. Auf diese Weise kann unnötiger Netzverkehr vermieden werden.

Java-Programme können beispielsweise eingesetzt werden, um ein eingegebenes Datum sofort auf Gültigkeit zu prüfen oder dem Benutzer komfortable Eingabemöglichkeiten zu bieten, die über die Standard-HTML-Bedienelemente hinausgehen.

Rein theoretisch wäre es sogar möglich, per Java-Programm die gesamte Liste der gültigen Bahnhöfe an den Browser zu übertragen, wo die Benutzereingabe samt eventuell erforderlicher Ähnlichkeitssuche usw. durchgeführt wird. Das scheitert jedoch wiederum an der großen Datenmenge – die Bahnhofsliste muß auf dem Server bleiben.

JavaScript ist eine von der Firma Netscape entwickelte vereinfachte Version von Java, die ohne ein eigenes Entwicklungssystem auskommt und ebenfalls geeignet ist, eine verbesserte Benutzeroberfläche zu entwickeln.

6.1.3 Übersicht über den veränderten Programmablauf

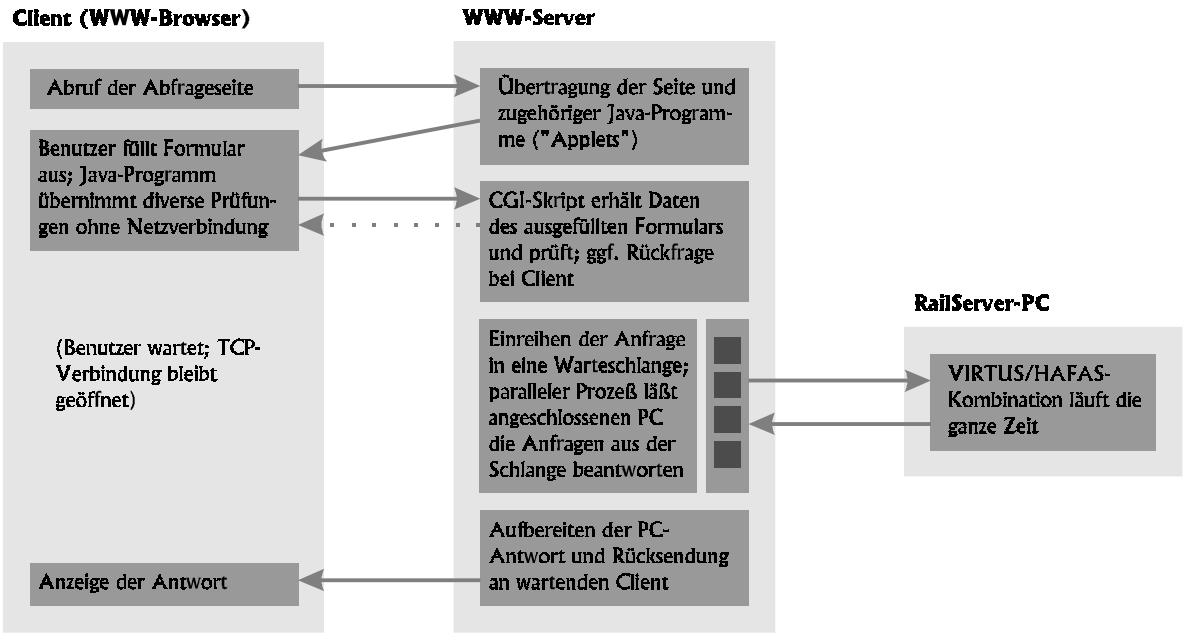

Abbildung 14: Schematische Darstellung einer möglichen Online-Lösung

Abbildung 14 zeigt einen vereinfachten Überblick über die Funktionsweise einer möglichen RailServer-Online-Lösung. Die folgenden Abschnitte beschreiben genauer, welche Veränderungen im Vergleich zur Offline-Funktion erforderlich sind.

6.2 Technische Anpassungen des Virtus-Ablaufs

Bei einer Online-Lösung darf auf dem PC keine Sekunde Zeit verschenkt werden; am anderen Ende wartet ja ein Benutzer auf seine Antwort, und womöglich waren schon einige andere in der Warteschlange vor ihm dran.

Der PC muß sich daher auf das konzentrieren, was nicht auf das Unix-System ausgelagert werden kann: Die Ermittlung einer Auskunft mittels der Virtus/HAFAS-Programmkombination. Da HAFAS etwa eine Minute zum Start benötigt, ist es nicht möglich, HAFAS für jede Anfrage neu zu starten. Stattdessen müssen HAFAS und das fernsteuernde Virtus die ganze Zeit aktiv sein und mit der Außenwelt kommunizieren.

Dies kann man durch eine Erweiterung von Virtus um die Möglichkeit, Variablensätze über die serielle Schnittstelle des PCs einzulesen, erreichen. Im Ruhezustand wartet Virtus auf Daten an der seriellen Schnittstelle; sobald ein vollständiger Variablensatz eintrifft, wird dieser ganz analog zur Offline-Lösung in HAFAS eingespeist. HAFAS wird so konfiguriert, daß die Ausgabe nicht in eine Datei, sondern auf die serielle Schnittstelle „gedruckt“ wird.

Auf diese Weise kann das HAFAS-Program mittels Virtus sehr direkt vom Unix-Rechner aus fernbedient werden; kommuniziert wird einzig über die serielle Schnittstelle.

Bei dieser Lösung muß sorgfältig darauf geachtet werden, daß der PC nicht aufgrund von nicht vorhergesehenen Eingaben aus dem Takt gerät und womöglich keine weiteren Eingaben mehr entgegennimmt; denkbar ist eine kleine Hardware-Schaltung, die es dem Unix-Rechner ermöglicht, am PC einen Kaltstart auszulösen, wenn dieser eine gewisse Zeit nicht reagiert.

6.3 Verstärkter Einsatz von Unix-Programmen

Da der PC bei der skizzierten Online-Lösung nur noch das reine HAFAS-Programm betreibt, müssen alle anderen Aufgaben auf dem WWW-Server unter Unix gelöst werden.

6.3.1 Homogenisierung der Anfragen

Die Überprüfung der Benutzereingaben (bei der Offline-Lösung Aufgabe des PROCESS1-Programms) wird hier größtenteils im Vorfeld abgewickelt, Unstimmigkeiten können im Dialog mit dem Benutzer geklärt werden. Da dieser Programmteil völlig unabhängig vom PC funktionieren kann, ist er trivial; es wird eine dem PROCESS1-Programm vergleichbare Lösung auf das Unix-System übertragen.

6.3.2 Scheduling-Probleme

Komplikationen treten da auf, wo die Schnittstelle zum PC bedient werden muß, um die eigentliche Auskunft zu erhalten. Das CGI-Skript, das gestartet wird, wenn der Benutzer eine Anfrage abschickt, kann nicht einfach auf die serielle Schnittstelle schreiben, da wegen des Multitasking mehrere parallele Anfragen zur Verstümmelung der Daten führen könnten.

Ein einfaches Semaphorkonzept reicht hier ebenfalls nicht aus, da die Anfragen auf jeden Fall in der Reihenfolge ihres Eingangs beantwortet werden sollen, um die Wartezeit für den einzelnen Benutzer so gering wie möglich zu halten.

Es muß also – beispielsweise mit System V IPC Message Queues[13] – ein Scheduler-Prozeß laufen, der alle Anfragen, die u. U. gleichzeitig eintreffen, sammelt und geordnet an den PC weiterleitet, auf dessen Antwort wartet und dies dann wiederum dem anfragenden Prozeß zuführt.

Der Scheduler formatiert die Anfrage so, daß Virtus auf der anderen Seite sie verstehen kann, übersendet sie an den PC und wartet auf Antwort. Erhält er in einer vordefinierten Zeit keine Antwort, muß am PC ein Neustart der (vermutlich abgestürzten) Software ausgelöst werden; hierzu wird wohl eine Hardware-Schaltung erforderlich sein. Kommt die Antwort hingegen wie erwartet zurück, kann sie unbearbeitet an das CGI-Skript, von dem die Anfrage kam, zurückgegeben werden; dort wird dann die weitere Formatierung erledigt. So ist sichergestellt, daß der Scheduler am Flaschenhals PC-Schnittstelle einen optimalen Durchsatz erzielt – alles, was nicht mit dem PC zu tun hat, kann ja im Multitasking-Betrieb erledigt werden.

Der Scheduler muß auch über die Fähigkeit verfügen, zu erkennen, wenn ein Benutzer des Wartens auf die Antwort überdrüssig ist und seinen Browser beendet (oder auf den „Stop“-Knopf drückt). In diesem Fall muß die Anfrage, sofern sie sich noch in der Warteschlange befindet, sofort entfernt werden, um nicht unnötig Rechenzeit zu verschwenden.

6.3.3 Aufbereitung und Rückgabe der Antwort

Nach Ermittlung der Antwort fallen dieselben Aufgaben an, die bei der Offline-Lösung durch das PROCESS2-Programm bewerkstelligt wurden; in der Hauptsache geht es um die Formatierung der HAFAS-Ausgabe.

Da es sich in diesem Fall nicht um die Ausgabe im nur-Text-Format (wie bei den Mails, die die Offline-Lösung verschickte) handelt, kann auch von den HTML-Textgestaltungsmöglichkeiten HTML Gebrauch gemacht werden, um beispielsweise zentrierte Überschriften, in Tabellen angeordnete Texte usw. darzustellen.

Die fertige Antwort wird über die seit dem Eintreffen der Anfrage aufrechterhaltene TCP-Verbindung zum WWW-Browser zurückgesandt und dort dargestellt.

---

[13] Details hierzu in Stevens, 1990.

7 Schlußbetrachtung

Mit dem Programm „Virtus“ und der damit verbundenen Skriptsprache wurde in dieser Arbeit ein universell anwendbares Verfahren zur automatisierten Bedienung von nahezu beliebigen DOS-Applikationen vorgestellt[14].

Neben anderen Einsatzzwecken ist es unter Zuhilfenahme von Virtus möglich, ein Informationssystem, das unter DOS läuft und eigentlich für Interaktionen mit einem Benutzer vorgesehen ist, im Internet verfügbar zu machen. Je nach Art des Informationssystems sind neben dem Informationssystem selbst und Virtus weitere, speziell angepaßte Komponenten erforderlich:

- die Benutzerschnittstelle – beispielsweise in Form von WWW-Eingabeseiten;
- Hilfsprogramme, die die vom Benutzer eingegebenen Daten in einer geeigneten Art und Weise an das auf einem PC laufende Virtus weiterleiten;
- Hilfsprogramme, die die Ausgabe des von Virtus gesteuerten Informationssystems an den Benutzer zurückgeben,

Anhand eines umfassenden Beispiels – des seit über einem Jahr im Betrieb befindlichen „RailServer“-Auskunftsdienstes für Bahnverbindungen – wurden alle diese Komponenten und auch der konkrete Einsatz von VIRTUS detailliert geschildert.

Ebenfalls bezugnehmend auf den „RailServer“ wurden Überlegungen angestellt, welche Modifikationen an den vorgestellten Komponenten erforderlich sind, um vom derzeitigen zeitversetzen Offline-Betrieb auf eine Online-Lösung umzustellen. Da sich diese Überlegungen größtenteils auf die Benutzerschnittstelle und auf Scheduling-Probleme an der Schnittstelle zwischen einem (Multitasking-) Unix-Sytem und dem (Singletasking-) PC beziehen, besitzen sie über das konkrete Beispiel hinaus Gültigkeit.

Durch den Vergleich mit einigen handelsüblichen Lösungen zur Fernsteuerung von PCs wurde gezeigt, daß die vorgestellte Lösung, wenn auch als „Nischenprodukt“, durchaus ihre Berechtigung hat und insbesondere bei der „stapelweisen“ Verarbeitung von Anfragen durch keines der üblichen Produkte ersetzbar ist.

[14] Quelltexte und auch die benötigten Binärdateien sind im Internet unter dem URL http://rail.rz.uni-karlsruhe.de/rail/virtus/ abrufbar.

8 Literaturverzeichnis

COSTALES, Bryan: Sendmail. O'Reilly, Sebastopol, 1993

DALITZ, Wolfgang und HEYER, Gernot: Hyper-G. Das Internet-Informationssystem der 2. Generation. dpunkt, Heidelberg, 1995

EBNER, G.: Wort-Arithmetik. Phoentische Ähnlichkeiten mit der Levenshtein-Distanz errechnet, c't 7/89, S. 192-208, Heise Verlag, Hannover, 1989

HEWLETT-PACKARD INC.: HP 500 Windows Application Server. Zu finden im WWW unter `http://hpcc998.external.hp.com/pad/hp500.html`.

LOGICRAFT INC.: Omni-Ware Manual. Release 2.01, 1992.

NETSCAPE CORPORATION: JavaScript Manual (under development). – Ständig in Entwicklung, zu finden im WWW unter `http://home.netscape.com/`.

RAMM, Frederik: Recherchieren und Publizieren im World Wide Web. Vieweg, Braunschweig/Wiesbaden, 1995 *(World Wide Web, HTML, CGI)*

SCHÄPERS, Arne: Turbo Pascal 6.0. Konzepte, Analysen, Tips & Tricks. Addison-Wesley, Bonn/München, 1991 *(Speicherresidente Programme unter Turbo Pascal)*

STEVENS, W. Richard: UNIX network programming. Prentice Hall, Englewood Cliffs, 1990 *(System V IPC, Message-Queueing unter Unix)*

SUN MICROSYSTEMS: Java Manual. – Ständig in Entwicklung, zu finden im WWW unter `http://java.sun.com/`.

TISCHER, Michael: PC intern. Systemprogrammierung. 4. Auflage. Data Becker, Düsseldorf 1989 *(Speicherresidente Programme; Interrupts; reservierte Speicherbereiche beim PC)*

WALL, Larry und SCHWARTZ, Randal L.: Programming Perl. O'Reilly, Sebastopol, 1992.

Anhang A: Alle VIRTUS-Sprachelemente

In diesem Anhang findet sich eine alphabetische Liste aller Sprachelemente, die VIRTUS verarbeiten kann. Ein anschauliches Beispiel hierzu ist das Skript, das für den RailServer eingesetzt wird und im Anhang B abgedruckt ist.

' <text>

Das Apostroph-Zeichen wird benutzt, um einen Kommentar einzuleiten; <text> wird von VIRTUS ignoriert.

Beispiel:

```
' Hier folgt die Reaktion auf eine Fehlermeldung:
```

GOTO <marke>

Springt zu der angegebenen Sprungmarke.

Beispiel:

```
GOTO EndProgram
```

GOSUB <marke>

Dient dem gleichen Zweck wie GOTO, doch wird hier die aktuelle Ausführungsposition auf einen Stack gelegt, so daß mit RETURN ein Rücksprung hinter das GOSUB möglich ist.

Beispiel:

```
GOSUB SubProgram
..
LABEL SubProgram
   IF "Error" AT 1,1
     GOTO EndProgram
   END IF
   RETURN
```

HALT

Beendet die Ausführung des Skripts. VIRTUS bleibt im Speicher, setzt jedoch den Interruptvektor zurück, so daß die volle Rechenleistung wieder anderen Programmen zur Verfügung steht.

IF CURSOR AT <row>,<col>

Führt alles zwischen dem IF und dem zugehörigen END IF (oder ELSE) nur dann aus, wenn der Cursor sich gerade an der angegebenen Position befindet. Diese Methode wird i.d.R. verwendet, um zu prüfen, ob ein bestimmtes Eingabefeld gerade aktiv ist. Die obere linke Ecke ist (1,1).

Beisipiel:

```
IF CURSOR AT 10,50
   SENDKEY "Q"
END IF
```

IF "<text>" AT <row>, <col>

Führt den folgenden Block nur aus, wenn der angegebene <text> sich an der angegebenen Position auf dem Bildschirm befindet. Wird i.d.R. benutzt, um auf bestimmte Meldungen des ferngesteuerten Programms reagieren zu können.

Beispiel:

```
IF "ERROR" AT 20, 10
  SENDKEY "Q"
END IF
```

IF DEF <varname>

Führt den folgenden Block nur aus, wenn die angegebene Variable definiert ist. Auf diese Weise kann man logische Variablen einsetzen, die den Programmmfluß steuern. „Definiert“ (der tatsächliche Inhalt ist dann egal) entspricht dem TRUE, und undefiniert ist FALSE.

Beispiel:

```
IF DEF useLog
  SENDKEY "L"
END IF
```

IF EOT

Führt den folgenden Block nur dann aus, wenn bereits alle Variablensätze in der Variabendatei abgearbeitet wurden (ein SHIFT – siehe dort – also fehlschlagen würde).

Beispiel:

```
IF EOT
   GOTO EndProgram
END IF
```

Zu jedem IF gehört ein abschließendes END IF; ELSE-Blöcke können optional verwendet werden.

LABEL <labelname>

Definiert eine neue Sprunggmarke, die mit GOTO oder GOSUB zusammen verwendet werden kann. Die Groß- und Kleinschreibung spielt keine Rolle.

Beispiel:

```
LABEL Ausdruck
```

OUTPUT SCREEN <row1>, <col1>, <row2>, <col2>

Sendet den durch die Koordinaten der Eckpunkte beschriebenen rechteckigen Bildschirmbereich an den Ausgabepuffer. Dieser Puffer kann später durch einen Aufruf von VIRTUS mit dem -o-Parameter in eine Datei geschrieben werden.

Beispiel:

```
OUTPUT SCREEN 10,10,25,80
```

OUTPUT <varname>

Sendet den Inhalt der Variablen <varname> an den Ausgabepuffer.

Beispiel:

```
OUTPUT QueryNumber
```

OUTPUT "<text>"

Sendet <text> an den Ausgabepuffer.

Beispiel:

```
OUTPUT "***Programmausgabe folgt:"
```

SENDKEY <keylist>

Sendet die angegebenen Tastendrücke über den Tastaturpuffer an das fernzusteuernde Programm. Es können ASCII-Codes (z.B. #13 für die Return-Taste oder #27 für ESC), Texte in Anführungszeichen (z.B. "Q") und einige festgelegte Tastenbezeichnungen (F1 bis F10, UP, DOWN, PGUP, PGDN, LEFT, RIGHT, HOME, END) benutzt werden. (Diese Angaben werden bereits m Precompiler VIRTUSC ausgewertet und in die im Tastaturpuffer zu verwendenden Kombinationen aus Key- und Scancode umgewandelt.)

Beispiel:

```
SENDKEY "Q" #13 F10 #13
```

SENDVAR <varname>

Sendet den Inhalt der angegebenen Variablen über den Tastaturpffer an das fernzusteuernde Programm.

Beispiel:

```
SENDVAR TextEingabe
```

SHIFT

Schaltet zum nächsten Vriablensatz um; sämtliche Variableninhalte ändern sich. Hierzu muß beim Programmstart mit der -v-Option eine Variablendatei mit mehr als einem Variablensatz eingelesen worden sein. Mit IF EOF kann abgefragt werden, ob überhaupt noch weitere Variablensätze vorliegen.

SPEED <zahl>

Legt fest, wie schnell das VIRTUS-Skript abgearbeitet wird. 1 bedeutet keine Wartezyklen, also 18 Schritte[15] pro Sekunde (pro Interruptaufruf einer); 18 würde beispielsweise 17 Wartezyklen (nur einen Programmschritt pro Sekunde) bewirken.

Beispiel:

```
SPEED 1
```

VAR <varname> "<text>"

Definiert eine neue Variable <varname>. Muß am Anfang des Programms stehen. Groß- und Kleinschreibung des Variablennamens sind unbedeutend. Wenn beim Aufruf von VIRTUS eine Variablendatei angegeben wurde, wird der Inhalt der Variablen von dort gelesen (oder, bei ihrem Fehlen, die Variable auf „undefiniert“ gesetzt). Sonst würde der angegebene <text> direkt nach Aufruf von VIRTUS am Schirm angezeigt, und das Programm würde auf die Eingabe des Inhalts für diese Variable warten. (Diese Möglichkeit kommt i.d.R. nur zu Debug-Zwecken zur Anwendung; im Betrieb arbeitet man mit Variablendateien, da VIRTUS ja gerade dazu da ist, die Interaktivität zu umgehen.)

Beispiel:

```
VAR Startbahnhof "Bitte Startbahnhof eingeben:"
```

WAIT <laenge>

Hält das Programm für <laenge> Achtzehntelsekunden an.

---

[15] Zur Bedeutung eines Programmschrittes vgl. Abschnitt 4.2.2.

Anhang B: Das RailServer-Skript

Das hier abgedruckte Skript wird für den RailServer eingesetzt. Zum besseren Verständnis werden die bereits in Abschnitt 5.2.2 enthaltenen Abbildungen hier noch einmal wiederholt; sie sind genau an den Stellen plaziert, an denen das Skript die jeweils abgebildeten Situationen verarbeitet.

```
' Definition der Variablen
VAR FROMSTATION "Startbahnhof"
VAR TOSTATION "Zielbahnhof"
VAR VIASTATION "ueber Bahnhof (Enter = direkt)"
VAR TRAVELDATE "Reisedatum (MMTTJJ)"
VAR ARRIVAL "Soll die Ankunftszeit eingegeben werden?"
VAR ENGLISH "Englische Bedienung?"
VAR TRAVELTIME "Zeit (SSMM)"
VAR DURATION "Bereich (SSMM)"
VAR OUTFILENAME "Name fuer Ausgabedatei"

' Zug-Ausschluesse. Wenn DEFAULT gesetzt ist, wird das Produktwahlfenster
' gar nicht erst aufgeblendet (geht schneller)
VAR DEFAULT "Keine Einschraenkung"
VAR EX-ICE "Kein ICE"
VAR EX-IC "Kein IC/EC"
VAR EX-IR "Kein IR"
VAR EX-E "Keine RSB/E"
VAR EX-SCHIFF "Keine Schiffe"
VAR EX-N "Kein Nahverkehr"
VAR EX-S "Keine S-Bahn"
VAR NUR-DIREKT "Nur direkt"
VAR NUR-RAD "Nur mit Fahrrad"
VAR NUR-SCHLAF "Nur Schlafwagen"
VAR NUR-LIEGE "Nur Liegewagen"
SPEED 1
LABEL START
```

Abbildung 15: Der HAFAS-Startbildschirm (der Cursor steht im markierten Eingabefeld)

```
' Warten, bis der Cursor an Position 3,10 - d.h., das erste Eingabefeld
```

```
    ` wird aktiv (umittelbar nach Programmstart)
    IF CURSOR AT 3, 10
       ` Sprache einstellen und Druck-Ausgabedatei aktivieren
       IF DEF ENGLISH
          SENDKEY F10 "O" #13 DOWN #13
          SENDKEY F10 "P"
       ELSE
          SENDKEY F10 "O" #13 #13
          SENDKEY F10 "D"
       END IF
       SENDKEY UP UP #13 UP #13 #8 #8 #8 #8 #8 #8 #8 #8 #8 #8 #8 #8 #8
```

Abbildung 16: Einstellung des Ausgabedateinamens

```
       SENDVAR OUTFILENAME
       SENDKEY #13 #27

       LABEL StartBahnhof

' Startbahnhof
       SENDVAR FROMSTATION
       SENDKEY #13
```

Abbildung 17: Eingabe des Start- und Zielbahnhofs

```
' Zielbahnhof
       SENDVAR TOSTATION
       SENDKEY #13
```

```
' Umwegbahnhof
      IF DEF VIASTATION
         SENDVAR VIASTATION
      ELSE
         SENDKEY #32 #8
      END IF
      SENDKEY #13

      SENDVAR TRAVELDATE
      SENDKEY #13
      SENDVAR TRAVELTIME
      SENDKEY #13
      SENDVAR DURATION

      IF DEF DEFAULT
         GOTO Default
      END IF

' Zugtyp-Einstellungen
```

Abbildung 18: Zugtyp-Einstellungen in HAFAS

```
SENDKEY DOWN DOWN DOWN DOWN
      IF DEF EX-ICE
         SENDKEY " "
      END IF
      SENDKEY DOWN

      IF DEF EX-IC
         SENDKEY " "
      END IF
      SENDKEY DOWN

      IF DEF EX-IR
         SENDKEY " "
      END IF
      SENDKEY DOWN

      IF DEF EX-E
```

```
         SENDKEY " "
      END IF
      SENDKEY DOWN

      IF DEF EX-SCHIFF
         SENDKEY " "
      END IF
      SENDKEY DOWN

      IF DEF EX-N
         SENDKEY " "
      END IF
      SENDKEY DOWN

      IF DEF EX-S
         SENDKEY " "
      END IF
      SENDKEY DOWN

      IF DEF NUR-DIREKT
         SENDKEY " "
      END IF
      SENDKEY DOWN


      IF DEF NUR-SCHLAF
         SENDKEY " "
      END IF
      SENDKEY DOWN

      IF DEF NUR-LIEGE
         SENDKEY " "
      END IF
      SENDKEY DOWN

      IF DEF NUR-RAD
         SENDKEY " "
      END IF

LABEL Default
      IF DEF ARRIVAL
         SENDKEY #13 F2
      ELSE
         SENDKEY #13 F3
      END IF

      WAIT 10

' Warteschleife, bis Verbindung errechnet oder Fehlermeldung erscheint
      LABEL LOOP
```

```
        IF "#" AT 7, 3
```

Abbildung 19: HAFAS bei der Berechnung einer Verbindung. Wenn fertig, erscheint ein # an Position 7,3

```
' Befehl zum Ausdruck der angezeigten Verbindung
LABEL Drucken
         IF DEF ENGLISH
           SENDKEY F10 "P" #13
         ELSE
           SENDKEY F10 "D" #13
         END IF
         WAIT 20
         IF "•" AT 8,20
            IF DEF ENGLISH
               SENDKEY "O"
            ELSE
               SENDKEY "Ü"
            END IF
         END IF
         GOTO NEXTVAR
      ELSE
         ' Fehlermeldung des HAFAS-Programms
         IF DEF ENGLISH
            IF "Unsuccessful" AT 10, 30
               GOTO Drucken
            END IF
         ELSE
            IF "ergebnislos" AT 10, 38
               GOTO Drucken
            END IF
         END IF
         IF "•" AT 8, 20
            GOTO NEXTVAR
         END IF
         GOTO LOOP
      END IF
   ELSE
      GOTO START
   END IF
```

```
LABEL NEXTVAR
   IF EOT
      GOTO ENDE
   ELSE
      SHIFT
      SENDKEY #27 #27 #27 F8 #9 F8
      GOTO START
   END IF

LABEL ENDE
   ' Programm verlassen
   IF DEF ENGLISH
      SENDKEY #27 F10 "Q" #13
   ELSE
      SENDKEY #27 F10 "E" #13
   END IF
   HALT
```