Gefördert vom

bmb+f

Bundesministerium für
Bildung, Wissenschaft,
Forschung und Technologie

Verbmobil

Verbundvorhaben

Verbmobil Integrations- und Testumgebung

Benutzerhandbuch

M. Auerswald, T. Bub, H. Kirchmann,
J. Kroner, J. Schwinn

DFKI GmbH Kaiserslautern

Vm

Report 178
Oktober 1996

Oktober 1996

Marko Auerswald, Thomas Bub, Heinz Kirchmann,
Hans-Jörg Kroner, Johannes Schwinn

DFKI GmbH Kaiserslautern
Erwin-Schrödinger-Str. (Gebäude 57)
Postfach 2080
67608 Kaiserslautern

Tel.: (0631) 205 - 3841
Fax: (0631) 205 - 3210
e-mail: auerswa@dfki.uni-kl.de

Gehört zum Antragsabschnitt: TP 16

Die vorliegende Arbeit wurde im Rahmen des Verbundvorhabens Verbmobil vom Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF) unter dem Förderkennzeichen 413-4001-01 IV 301 4 gefördert. Die Verantwortung für den Inhalt liegt bei den Autoren.

Inhaltsverzeichnis

Kapitel 1

Technischer Überblick über Verbmobil 3

1.1 Fachliche und softwaretechnische Architektur 3
1.2 Architektur und Konzepte der Testumgebung..... 6
1.2.1 Anforderungen an die Testumgebung..... 6
1.2.2 Testbed-Architektur 7
1.2.3 Grundlegende Konzepte 7

Kapitel 2

Konfiguration und Start 10

2.1 Die Environment-Variablen..... 10
2.2 Konfigurationskonzept (offline) 11
2.2.1 Abgrenzung 11
2.2.2 Konfigurations-Dateien und Kompilation 11
2.3 Start des Prototyps..... 14

Kapitel 3

Benutzeroberfläche und Bedienung 15

3.1 Benutzungsoberfläche 15
3.2 Bedienung..... 17
3.2.1 Spracheingabe..... 17
3.2.2 Betriebsmodi..... 17
3.2.3 Konfigurierbarkeit der Module 18
3.2.4 Menü..... 20

Kapitel 4

Daten-Visualisierung 24

4.1 Daten-Archivierung..... 24
4.1.1 Namenskonvention 24
4.1.2 Archivierungsarten 25
4.1.3 Prolog Specials 26
4.2 Visualisierung im Testbed 26
4.2.1 Über Paper-Icons auf der GUI..... 26

4.2.2 Unsichtbare Schnittstellen ... 27
4.2.3 Die Viewer ... 27
4.3 Standalone Visualisierung ... 28

Kapitel 5

Testwerkzeuge **29**
5.1 Einspeisen fachlicher Nachrichten ... 29
5.1.1 Vergabe der Turn-Id ... 30
5.1.2 Nachrichten vom Datentyp „prolog“ oder „v-i-term“ ... 30
5.1.3 Special: String-Eingabe in die SynSem ... 30
5.2 Einspeisen von Steuer-Nachrichten ... 31
5.3 Das Automatische Testmodul ... 31
5.4 Daten-Aufbereitung nach umfangreichen Testläufen ... 31
5.4.1 Extraktion der entstandenen Dateien ... 31
5.4.2 Erzeugung phondat-korrelierter Transliterationsdateien ... 32
5.4.3 Aufarbeitung der Logdatei des Verbmobil-Prototypen ... 32

Kapitel 6

Testen mit dem Automatischen Testmodul (ATM) **33**
6.1 Einführung ... 33
6.1.1 Motivation ... 34
6.1.2 Sendeblock und Konfiguration ... 35
6.1.3 Die Abarbeitung einer Konfiguration ... 36
6.2 Die Konfigurierung des automatischen Testmoduls ... 39
6.2.1 Aufbau einer Konfigurationsdatei ... 39
6.2.2 Aufbau einer Konfiguration ... 39
6.2.3 Aufbau eines Sendeblocks ... 44
6.2.4 Beispiele für Konfigurationsdateien ... 50
6.2.5 Übersicht: Befehle in Konfigurationsdateien ... 52
6.3 Aktivieren des automatischen Testmoduls ... 53
6.3.1 Start durch Kommandozeilenoption ... 53
6.3.2 Start über Visualisierungsmanager (VIM) ... 54
6.3.3 Abbruch der automatischen Verarbeitung ... 55
6.3.4 Ausgabedateien des automatischen Testmoduls ... 55

Kapitel 7

Weitere Entwicklung **56**

Kapitel 8

Literatur **57**

Kapitel 1

Technischer Überblick über Verbmobil

1.1 Fachliche und softwaretechnische Architektur

Das Verbmobil-System ist als eine Menge parallel ablaufender, miteinander kommunizierender UNIX-Prozesse realisiert. In einem solchen Prozeß wird die Software eines oder mehrerer Module abgearbeitet. Bei den einzelnen Software-Paketen unterscheidet man zwischen fachlichen Modulen, die die eigentliche Funktionalität des Verbmobils realisieren und den sogenannten softwaretechnischen Modulen, die durch Hilfsfunktionen die Arbeit der fachlichen Module visualisieren, unterstützen, vereinfachen oder auch überhaupt erst möglich machen. Neben der Kommunikationssoftware PVM und ICE sind als wichtigste softwaretechnische Module der Testbedmanager (TBM), die Benutzeroberfläche (GUI) sowie der Visualisierungsmanager (VIM) zu nennen. Zusammen mit dem automatischen Testmodul

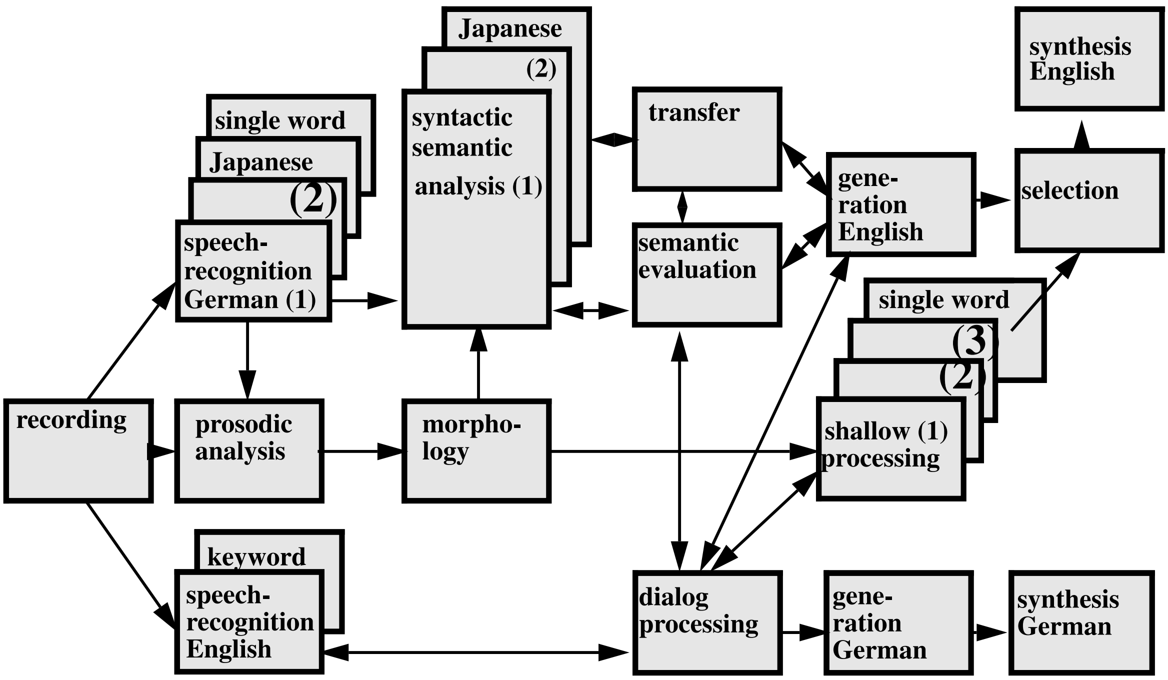


ABBILDUNG 1. Die fachliche Architektur von Verbmobil

(ATM) bilden diese Module das sogenannte Testbed, in das die fachlichen Module integriert werden.

Die Aufgabe des Testbeds besteht darin, die Arbeit des gesamten Verbmobil-Systems zu steuern, zu überwachen und zu visualisieren. Bei der Steuerung durch das Testbed handelt es sich lediglich um eine softwaretechnische Kontrolle der Abläufe im System.

Es gehört nicht zu den Aufgaben des Testbeds, selbstständig fachliche Entscheidungen zu treffen oder eine fachliche Kontrolle über irgendeines der fachlichen Module, oder sogar über das Gesamtsystem auszuüben. Die hierzu notwendigen detaillierten Informationen über den internen Zustand aller fachlichen Module sind im Testbed ebensowenig vorhanden wie das notwendige Wissen, um irgendwelche fachlichen Entscheidungen treffen zu können.

Für die Kommunikation zwischen den einzelnen als Unix-Prozesse realisierten Softwarekomponenten wird das Werkzeug PVM (Parallel Virtual Machine) [Geist et. al.] und darauf aufsetzend ICE (Intarc Communication Environment) [Amtrup 94a,b] eingesetzt. Als technische Module wurden von der Systemgruppe folgende Module entwickelt:

Tabelle 1: ICE Modulnamen der technischen Module

ICE Modulname	**Beschreibung des Moduls**
`automatic_test_module`	Automatisches Testmodul
`gui`	Benutzeroberfläche des Testbeds
`tbm`	Testbedmanager
`vim`	Visualisierungsmanager des Testbeds

Bei den fachlichen Modulen handelt es sich im einzelnen um:

Tabelle 2: ICE Modulnamen der fachlichen Module

ICE Modulname	**Beschreibung des Moduls**
`ali`	Alternative Linguistik
`audiodisp`	Audiodispatcher
`dialog`	Dialog
`flat_gen_b`	Flache Generierung
`flat_gen_sb`	Flache Generierung
`gener_eng`	Generierung Englisch
`gener_ger`	Generierung Deutsch
`keywordspotter`	Keywordspotter Englisch
`morphology`	Morphologie
`prosody`	Prosodie
`recog_eng_uka`	Spracherkenner Englisch
`recog_ger_db`	Spracherkenner Deutsch

Tabelle 2: ICE Modulnamen der fachlichen Module

ICE Modulname	Beschreibung des Moduls
recog_ger_uka	Spracherkenner Deutsch
recog_spell_ger_uka	Buchstabiererkenner
recog_jap_uka	Spracherkenner Japanisch
recog_yesno_ger_db	Ja/Nein-Erkenner
recording	Aufnahme
selection	Linguistisches Auswahlmodul
semeval	Semantische Auswertung
shallow	Flache linguistische Analyse
synsem_chunk	Chunk-Analyse-Modul
synsem_ibm	Syntax - Semantik Konstruktion
synsem_jap	Syntax - Semantik Konstruktion (Japanisch)
synsem_s3	Syntax - Semantik Konstruktion
synth_eng	Sprachsynthese Englisch
synth_ger	Sprachsynthese Deutsch
transfer	Transfer
transif_ger	Schnittstellenkonverter concept-to-speech
wordrecog_ger_db	Einzelworterkenner
wordrecog_ger_uka	Einzelworterkenner
wordtrans	Einzelwortübersetzer

In Abbildung 1 auf Seite 3 ist die fachliche Architektur von Verbmobil dargestellt. Dieselbe Grundarchitektur findet sich sowohl auf der Ebene der Software-Module, als auch auf der Ebene der UNIX-Prozesse wieder.

Trotz zahlreicher Kommunikationskanäle existieren lediglich 4 wesentliche Datentypen als Schnittstellenformate. Im einzelnen sind dies:

- das Audiosignal, zwischen dem Aufnahmemodul und den Spracherkennern,
- der Worthypothesengraph ([Nöth & Plannerer 93]), als Ausgabeformat der Spracherkenner,
- der VIT (Verbmobil Interface Term, [Dorna 96]), zwischen den liguistischen Komponenten, sowie
- Strings, als Eingabe in das Synthesemodul sowie an den sonstigen Daten-Schnittstellen.

Diese Beschränkung auf wenige standardisierte Formate sowie die Durchgängigkeit der Architektur über alle Ebenen der Realisierung sind eine grundlegende Voraussetzung für die Umsetzung der Breadboard-Architektur des Verbmobil-Forschungsprototypen.

1.2 Architektur und Konzepte der Testumgebung

Die Verbmobil-Testumgebung (Testbed) wird realisiert durch die technischen Module TBM (testbed manager), GUI (graphical user interface) und VIM (visualization and interface manipulation manager), sowie optional das Modul ATM (automatic test module).

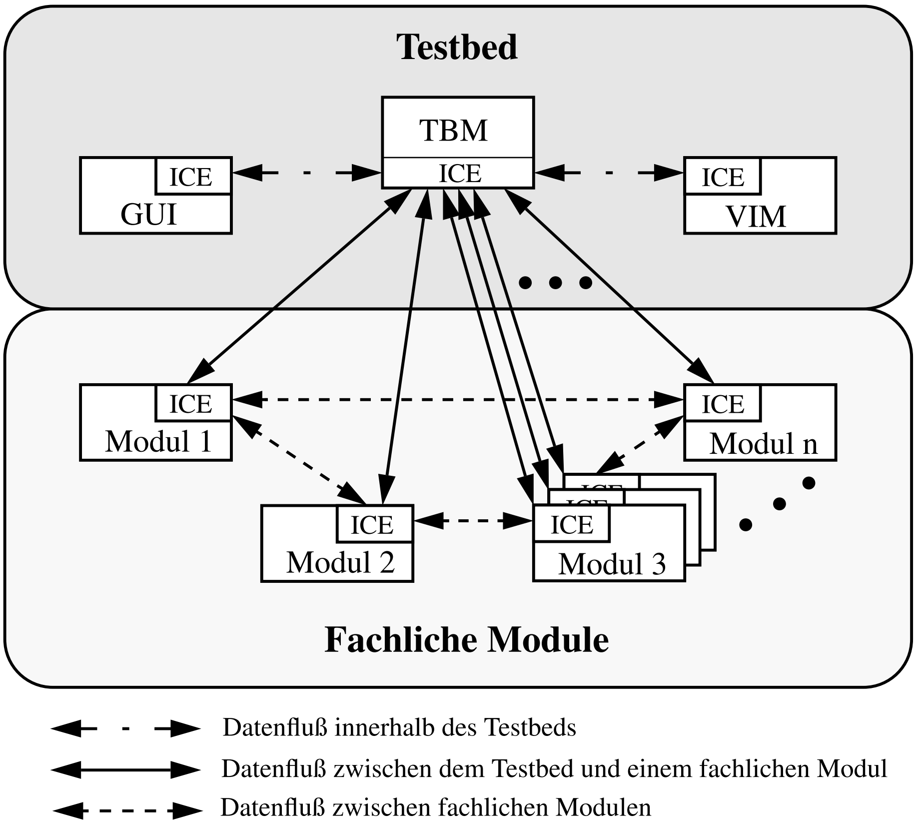


ABBILDUNG 2. Kommunikation im Testbed

1.2.1 Anforderungen an die Testumgebung

Da sich die Testumgebung an eine heterogene Nutzergruppe richtet, entspringen die Anforderungen an sie verschiedenen Quellen. Zunächst ist die Testumgebung ein Integrationsrahmen für ein komplexes System, das durch die Kombination seiner Komponenten sowohl präsentierbar als auch evaluierbar gemacht werden soll. Als Anforderungen an den Integrationsrahmen wären im Detail die folgenden zu nennen:

- Start, Verwaltung, Synchronisation der Module (UNIX-Prozesse),
- Bereitstellung eines Interprozeß-Kommunikationsmediums,
- Bereitstellung einer Benutzerschnittstelle

sowie aus Sicht von Präsentation und Evaluierung:

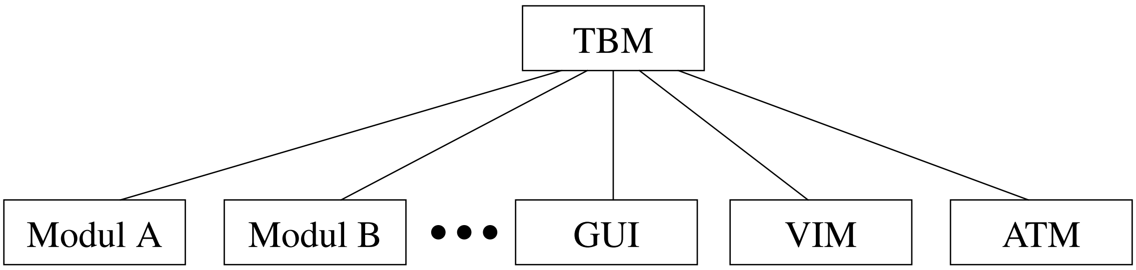
- Verfolgbarkeit der Verarbeitung,
- Visualisierung (zum Teil nur approximativ) von Zwischenergebnissen,
- leichte Konfigurierbarkeit freier Parameter.

Zusätzlich ist die Testumgebung ein Validierungs-Werkzeug für Modulentwickler; es soll also Tests der Module im Kontext des Systems ermöglichen und unterstützen. Daraus ergeben sich als weitere Anforderungen insbesondere:

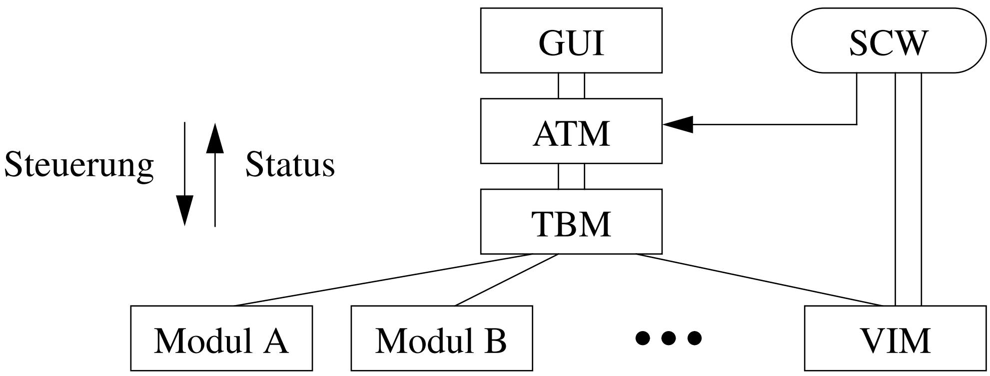
- Möglichkeit der Einspeisung von Testdaten an beliebigen Schnittstellen,
- Archivierung von Daten mit Möglichkeit zur nachträglichen Zuordnung von Eingaben, Zwischenergebnissen und Ausgaben,
- Automatisierbarkeit von Testläufen.

1.2.2 Testbed-Architektur

Die Verwaltung aller UNIX-Prozesse (das bezieht fachliche und technische ein) liegt beim Modul TBM, das die Prozesse startet, ihren Zustand überwacht und sie schließlich auch wieder beendet. Aus Prozeß-Sicht ergibt sich also das folgende Bild:



Die Steuerung des Systems erfolgt durch den Benutzer von der GUI aus bzw. (beim Einspeisen fachlicher Nachrichten) über das Send Control Window (SCW) des Moduls VIM. Wahlweise kann die Steuerung aber auch an das automantische Testmodul (ATM) abgegeben werden. Aus Steuerungssicht ergibt sich also die folgende Hierarchie, in deren vertikaler Richtung Steuer-Nachrichten (von oben nach unten) sowie Status-Nachrichten (von unten nach oben) fließen:



1.2.3 Grundlegende Konzepte

1.2.3.1 Konfigurationskonzept

Die Konfiguration der Module (z.B. Einstellung freier Parameter) erfolgt über modulspezifische Konfigurationsfenster der GUI (online). Zur Konfiguration des Gesamtsystems (beteiligte Module, vorhandene Kommunikationskanäle, Aufrufschnittstellen der Module) wurde eine formale Systembeschreibung erstellt, mit deren Hilfe die Testbed-Module im Wege einer automatischen Kompilation konfiguriert werden. Die System-Konfiguration erfolgt also bequem mittels allgemeiner sowie individueller Konfigurations-Dateien in einer formalen, wohldefinierten Syntax. Näheres zur Systemkonfiguration ist in Abschnitt 2.2 „Konfigurationskonzept (offline)“, Seite 11, zu finden.

1.2.3.2 Split Channels

Mit Hilfe der Split Channels wurde eine Methode geschaffen, Datenschnittstellen - für die Module transparent - zu manipulieren, und zwar sowohl passiv (abhören), als auch aktiv (Umleitung des Datenstroms, Zuführen eines anderen Datenstroms). Dazu wurde das Kanalmodell von ICE ausgenutzt und derart erweitert, daß das Ende eines virtuellen (vom fachlichen Modul bedienten) Kanals mit den Enden mehrerer realer Kanäle identifiziert werden kann. Eine Veranschaulichung dieses Konzeptes findet sich in Abbildung 3.

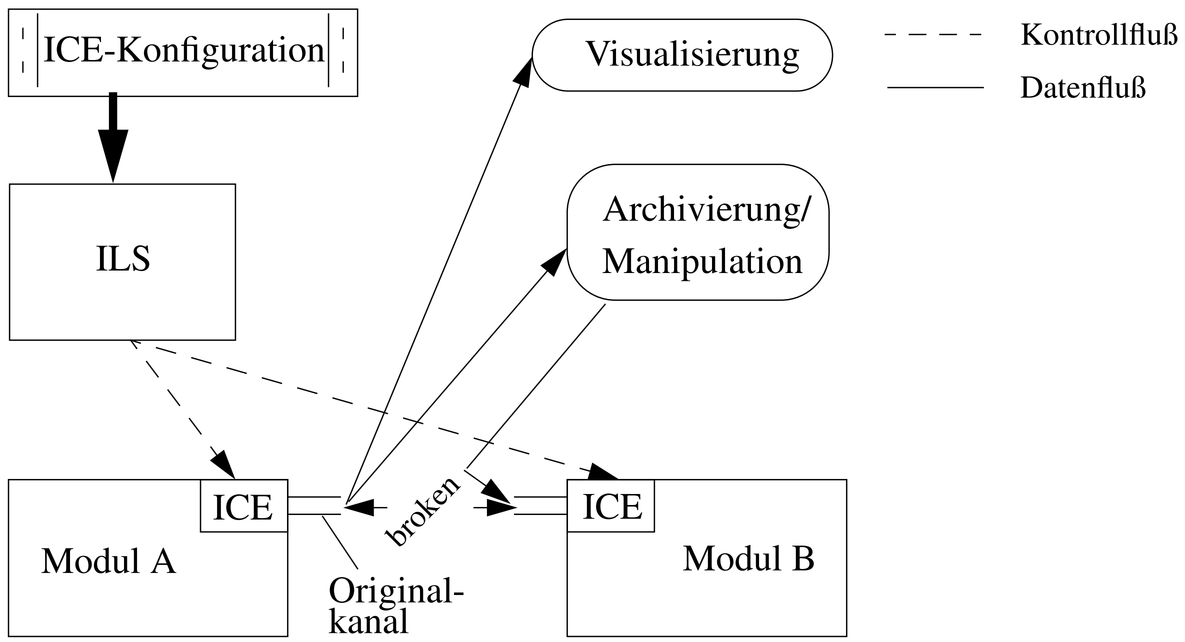


ABBILDUNG 3. Split Channels

1.2.3.3 Einspeisen fachlicher Nachrichten

Unter Ausnutzung des Split Channels Konzepts wurde die Möglichkeit geschaffen, an allen Datenschnittstellen des Systems Daten einzuspeisen (Abbildung 4). Näheres siehe Kapitel 5 „Testwerkzeuge“, Seite 29.

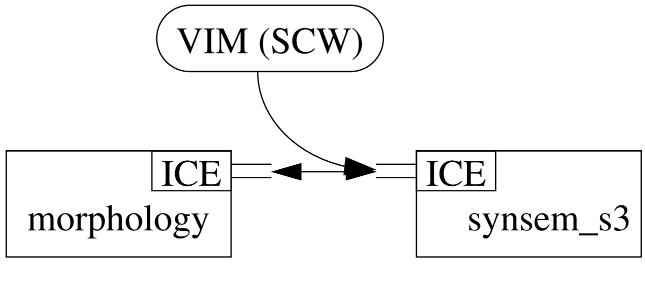


ABBILDUNG 4. Einspeisung von Daten

1.2.3.4 Automatisierte Testläufe

Bei automatischen Testläufen übernimmt das automatische Testmodul die Systemsteuerung sowie die Statuskontrolle. Daten werden über den VIM automatisch in das System eingespeist, und der Systemzustand währenddessen ständig überwacht.

Wenn nötig, werden Aktionen eingeleitet, die einen reibungslosen Testlauf gewährleisten, z.B. Neustart eines Moduls, Abbruch der Verarbeitung oder Wiederholung eines Turns. Sämtliche Ergebnisse und Zwischenergebnisse werden durch den VIM an den Datenschnittstellen abgegriffen und archiviert, so daß sie sich später nach Bedarf aufbereiten lassen.

Kapitel 2

Konfiguration und Start

Dieses Kapitel beschreibt die konfigurierenden Tätigkeiten, die vor dem Start des Verbmobil-Systems durchzuführen sind. Dabei wird vorausgesetzt, daß das System ordnungsgemäß installiert ist.

2.1 Die Environment-Variablen

Das System verwendet die folgenden Environment-Variablen, die defaultmäßig (in dem Skript $VM_HOME/bin/VmFoPro, siehe 2.3 „Start des Prototyps“, Seite 14) auf die folgenden Werte gesetzt werden, sofern sie nicht bereits definiert sind:

```
VM_HOME     /project/verbmobil/VmFoPro.stab
VM_TMP      /tmp/$USER
VM_SAVE     $VM_TMP
```

- **Zur Variablen `VM_HOME`**

Das System kann an jedem beliebigen Ort installiert werden. Der absolute Pfad muß über die Variable `$VM_HOME` spezifiziert werden.

Da in einigen Lisp- und Prolog-Dateien beim Übersetzen absolute Pfade eingetragen werden, müssen diese Teile des Systems dann aber neu übersetzt werden:

```
setenv VM_HOME  ....neuer absoluter Pfad.../VmFoPro
cd $VM_HOME/src
foreach i (*)
   (cd $i; echo „working in Directory $i“; make; make install)
end
```

ACHTUNG: Das geht natürlich nur, wenn der Source Code der Module vorhanden ist. Anderenfalls muß die Installation auf den einkompilierten absoluten Pfaden erfolgen.

- **Zur Variablen `VM_TMP`**

Über die Environment-Variable `$VM_TMP` wird spezifiziert, wo temporäre Dateien des Systems abgelegt werden.

ACHTUNG: Beim Start des Systems wird alles unterhalb `$VM_TMP` gelöscht!

- **Zur Variablen `VM_SAVE`**

Über die Environment Variable `$VM_SAVE` wird spezifiziert, wo Daten des Systems abgelegt werden, die längerfristig erhalten bleiben sollen. Mehr dazu findet man im Kapitel „Benutzeroberfläche und Bedienung“unter 3.2.3.1 „Aufnahme“, Seite 18.

Weitere Environment-Variablen für Suchpfade, die das System zur Laufzeit benutzen kann, werden in $VM_HOME/local/VmFoPro.env definiert.

2.2 Konfigurationskonzept (offline)

2.2.1 Abgrenzung

Bei der Konfiguration des Verbmobil-Systems muß man zunächst zwischen Online- und Offline-Konfiguration unterscheiden.

Modul-Konfiguration (online): Einige Module lassen sich zur Laufzeit mit Hilfe von Konfigurationsnachrichten beeinflussen. Diese Nachrichten erhalten die Module vom TBM. Die meisten Online-Konfigurationsmöglichkeiten werden in Form von Auswahlboxen an die GUI weitergereicht; die Konfiguration kann also komfortabel von der GUI aus erfolgen. Näheres hierzu siehe im Kaptitel „Benutzeroberfläche und Bedienung“ unter 3.2.3 „Konfigurierbarkeit der Module“, Seite 18.

System-Konfiguration (offline): In der Offline-Konfiguration wird festgelegt, welche Module bei einem Verbmobil-Testlauf einbezogen werden sollen, wie und auf welchem Rechner sie gestartet werden und (für die Daten-Archivierung) wie ihre Schnittstellen aussehen.

Im folgenden wird die Offline-Konfiguration des Systems beschrieben.

2.2.2 Konfigurations-Dateien und Kompilation

2.2.2.1 Standard-System-Konfiguration

In dieser Datei sind insbesondere diejenigen Informationen über alle Module festgehalten, die für konkrete integrierte Instanzen der Module fest sind (z.B. die Beschreibung der verwendeten Kommunikationskanäle). Damit soll erreicht werden, daß bei Neuintegration eines Moduls nur an dieser Konfigurationsdatei Änderungen anfallen, und nicht bei den individuellen Konfigurationen.

Zu Veränderungen in der Standard-Konfiguration sollte es nur kommen, wenn Schnittstellen verändert werden. Da dies Auswirkung auf alle beteiligten Module hat, sind solche Änderungen eher langfristiger Natur.

Die Standard-System-Konfiguration existiert nur einmal und steht immer unter

`$VM_HOME/etc/VmConfig`

Aufbau: Für jedes Modul gibt es einen Abschnitt der folgenden Gestalt:

```
1  MODULE recog_jap_uka  /* HK */
2      USED: no
3      STARTUP: „$VM_HOME/bin/VmRecogJapUka"
4      HOST: localhost
5      DATA_CHANNELS
6      { DESCR: recording recording_recog_jap_uka_raw 2
7        RECEIVE: FORMAT: phondat signal_incremental
8        SEND: no }
9      { DESCR: synsem_jap BASE -1
10       RECEIVE: no
11       SEND: FORMAT: whg idl_string }
12     CONTROL_CHANNELS
13     { DESCR: tbm BASE -1
14       RECEIVE: yes
15       SEND: FORMAT: ready active inactive resource_answer }
16 END
```

Bemerkungen:

- Zeile 1 spezifiziert den Modulnamen. Kommentare können wie in C angegeben werden.
- Zeile 2 gibt an, daß das Modul per Default nicht einbezogen werden soll; mögliche Werte: `yes` / `no`.
- In Zeile 3 ist der Startup spezifiziert. Es kann auch `no` eingetragen werden. Dies ist sinnvoll, wenn das Modul zwar beteiligt sein, aber nicht automatisch gestartet werden soll. In diesem Fall wird unter USED: `yes` und unter STARTUP: `no` eingetragen.
  Der Startup wird in Gänsefüßchen spezifiziert; es können beliebig Kommandozeilen-Argumente angefügt werden.
  VORSICHT: Da dieser Startup von PVM, und nicht durch eine Shell, ausgewertet wird, können nicht beliebige Environment-Variablen benutzt werden, sondern nur die in Abschnitt 2.1 „Die Environment-Variablen", Seite 10, angegebenen. Sie werden von dem Script $VM_HOME/bin/VmFoPro (siehe Abschnitt 2.3 „Start des Prototyps", Seite 14) automatisch expandiert.
- In Zeile 4 ist der Rechner angegeben, auf dem das Modul gestartet werden soll. Mögliche Werte:

  a) `localhost`: der Rechner, von dem aus gestartet wird; i.a. der leistungsfähige Server.
  b) `displayhost`: der Rechner, an dem sich Monitor und Audio Device bzw. die GradientBox befindet
  c) `anyhost`: PVM wählt je nach Auslastung einen günstigen Rechner aus.
  d) Alternativ kann direkt ein Host-Name angegeben werden.

- Ab Zeile 5 sind die Daten-Schnittstellen beschrieben. Im Beispiel sind zwei Kanäle deklariert. Auf das Token ‚DESCR:‘ folgt die Spezifikation des Kanals: Kommunikationspartner, Kanal-Name und Flag (näheres hierzu siehe [Amtrup 94b]). Hinter ‚RECEIVE:‘ und ‚SEND:‘ wird angegeben, ob, bzw. in welchem Format, Daten empfangen oder gesendet werden. Mögliche Werte:
  a) `no`
  b) `yes` (Format unspezifiziert)
  c) `FORMAT:` <Datentyp> <Übertragungstyp>
  Mögliche Werte für <Datentyp>:
  `phondat` / `whg` / `plain_text` / `v_i_term` / `prolog` / `lisp_sexpr`
  Mögliche Werte für <Übertragungstyp>:
  `idl_string` / `idl_term` / `idl_longstring` / `signal_incremental` / `file` / `name`
- Ab Zeile 12 sind die Kontroll-Schnittstellen spezifiziert; im Beispiel ein Kanal zum TBM. Im Unterschied zu den Daten-Schnittstellen sind hier hinter ‚FORMAT:‘ die erlaubten Nachrichten angegeben.

2.2.2.2 Individuelle System-Konfiguration

Das Testbed wird problemabhängig in verschiedenen Konstellationen betrieben. Für jede solche problemabhängige Konstellation kann eine individuelle Konfigurationsdatei angelegt werden. Welche individuelle Konfiguration benutzt werden soll, wird in der Environment-Variable `$VM_CONFIG` angegeben, z.B.

```
setenv VM_CONFIG ~/myConfig
```

Eine Kopiervorlage für die individuelle Konfiguration befindet sich unter `$VM_HOME/etc/VmIndividualConfig`.

Der Aufbau ist der gleiche wie bei der Standardkonfiguration; allerdings soll hier nur der Slot `USED`, ggf. auch noch die Slots `STARTUP` und `HOST` spezifiziert werden.

2.2.2.3 Kompilation

Durch das Script `$VM_HOME/bin/VmFoPro` (siehe Abschnitt 2.3 „Start des Prototyps“, Seite 14) werden automatisch Konfigurationen für ICE (Split Channels), den TBM (zu startende Prozesse), den VIM (bediente Kanäle) und die GUI (vorhandene Module) erzeugt.

Dazu werden Standardkonfiguration und individuelle Konfiguration aneinandergehängt und mit entsprechender Option an den Compiler `$VM_HOME/bin/configure` übergeben.

Konsistenz-Check: Die Konsistenz der Konfiguration läßt sich folgendermaßen überprüfen:

```
cat $VM_HOME/etc/VmConfig $VM_CONFIG |

$VM_HOME/bin/configure check
```

Dies sollte immer dann durchgeführt werden, wenn Änderungen an der Standardkonfiguration oder größere Änderungen an der individuellen Konfiguration vorgenommen worden sind. Keine Ausgabe bedeutet: alles in Ordnung; Inkonsistenzen oder Syntax-Fehler führen zu entsprechender Meldung.

2.3 Start des Prototyps

Der Verbmobil-Prototyp wird durch Aufruf des Skripts `$VM_HOME/bin/VmFoPro` gestartet. Die Ausgabe-Ströme der Module (`stdout` und `stderr`) werden sowohl nach `stdout` von `VmFoPro` (in das System-Start-Fenster) als auch in die Log-Datei `$VM_TMP/vm.log` geschrieben.

Das Skript `VmFoPro` versteht die folgenden Kommandozeilen-Argumente (man erhält diese Liste in ausführlicherer Form durch Aufruf von '`$VM_HOME/bin/VmFoPro -.`'):

-c <cfg-file> benutze die individuelle Konfigurationsdatei <cfg-file> anstelle der unter `$VM_CONFIG` spezifizierten

-e <env-file> verwende zur Spezifikation der Environment-Variablen für Suchpfade anstelle von `$VM_HOME/local/VmFoPro.env` die Datei <env-file>

-l <log> schreibe alle Ausgaben in die Datei <log> (und nicht in `$VM_TMP/vm.log`)

-m <cfg> füge die Zeile <cfg> zur individuellen Konfiguration hinzu; nähere Informationen im ausführlichen Hilfstext von `VmFoPro`.

Durch die Eingabe von CTRL-C in das System-Start-Fenster kann das System jederzeit beendet werden („Notbremse“).

Beim Hochfahren des Systems wird ein Fenster für die PVM-Konsole erzeugt (als Icon). Sollte das System nach Beendigung nicht vollständig herunterfahren, so hilft i.d.R. das Eintippen von „halt“ auf der PVM-Konsole.

Sofern die Aufnahme über die Gradient Box erfolgen soll, muß im Suchpfad des Benutzers, der das System startet, die Software der Gradient Box spezifiziert sein. Weiterhin muß die Gradient Box bei Ansteuerung über die Kommandozeile voll funktionsfähig sein. Näheres ist in der Dokumentation der Gradient Desklab 216 beschrieben.

Sollen Prolog-Module verwendet werden, muß auch der Suchpfad für die aktuelle Quintus Release im Environment spezifiziert sein.

Kapitel 3

Benutzeroberfläche und Bedienung

Im folgenden wird die Bedienung von Verbmobil und die Handhabung der Benutzungsoberfläche beschrieben.

Kurz nach dem Start des **Verbmobil-Forschungsprototyp** erscheint eine Dialogbox (siehe Abbildung 5 auf Seite 16), in der auszuwählen ist, ob die **Verbmobil**-Benutzungsoberfläche direkt (*Skip*) oder über das Intro-Fenster (*Show Intro*) (siehe Abbildung 6 auf Seite 16) und den Begrüßungstext gestartet werden soll.

3.1 Benutzungsoberfläche

Die in Abbildung 7 auf Seite 16 dargestellte Benutzungsoberfläche von Verbmobil enthält eine Menüleiste, 14 Schalter, die die fachlichen Module des Programmsystems darstellen, einen Schalter *Sprechen* für die Aktivierung des Programms, einen Schalter *Abbruch* zum Abbruch der Verarbeitung der letzten Äußerung und drei Auswahl-Schalter (Radio-Buttons) für die Auswahl des Programm-Modus (siehe unten). Die Pfeile stellen die Datenflußrichtung zwischen den Modulen dar.

Wenn ein Modul betriebsbereit ist, wird die Schrift auf dem entsprechenden Schalter schwarz dargestellt. Ist ein Modul nicht betriebsbereit (noch bei der Initialisierung oder per Konfiguration nicht gestartet), wird die Schrift in grau dargestellt.

Hinter einigen Schaltern sind u. U. mehrere separate fachliche Module verborgen. Dazu gehören:

- Spracherkennung Deutsch: Die Spracherkenner der Universität Karlsruhe (1) und der Daimler-Benz AG (2). Mit Spracherkenner sind hier sowohl die Einzelworterkenner als auch die Erkenner für kontinuierliche Sprache gemeint. Des weiteren repräsentiert dieser Schalter auch die Spracherkennung Japanisch, ggf. daß der entsprechende Betriebsmodus ausgewählt ist (siehe hierzu 3.2.2 „Betriebsmodi“, Seite 17).
- Flache Analyse & Transfer: Die Linguistik-Module Alternative Linguistik, Shallow und FlatGen-B/SB.
- Syntaxanalyse und Semantikkonstruktion: die tiefe linguistische Analyse für japanische und deutsche Sätze. Es stehen zwei Analyse-Module für die deutsche Sprache zur Verfügung: von Siemens / Universität Stuttgart / Universität Saarbrücken (1), von IBM (2) sowie das Module SynSem-Chunk.

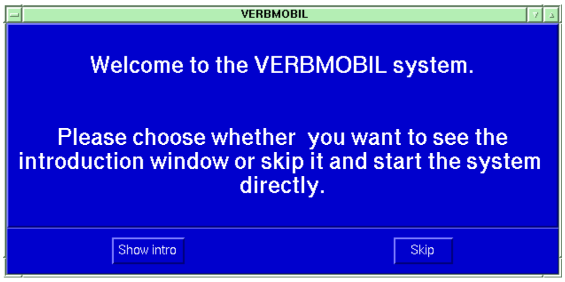


ABBILDUNG 5. Dialogbox zur Auswahl Intro-Fenster / Verbmobil-Oberfläche

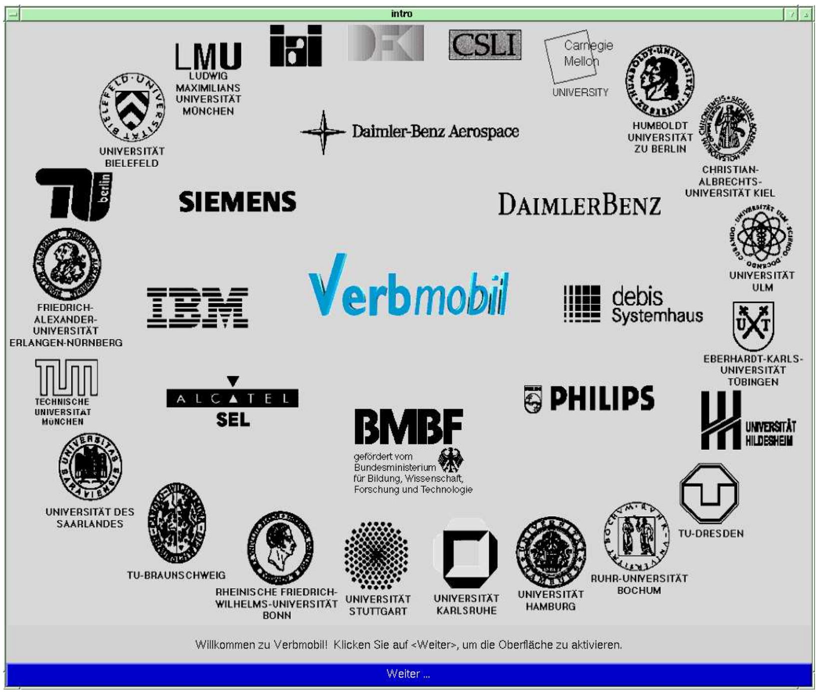


ABBILDUNG 6. Intro-Fenster

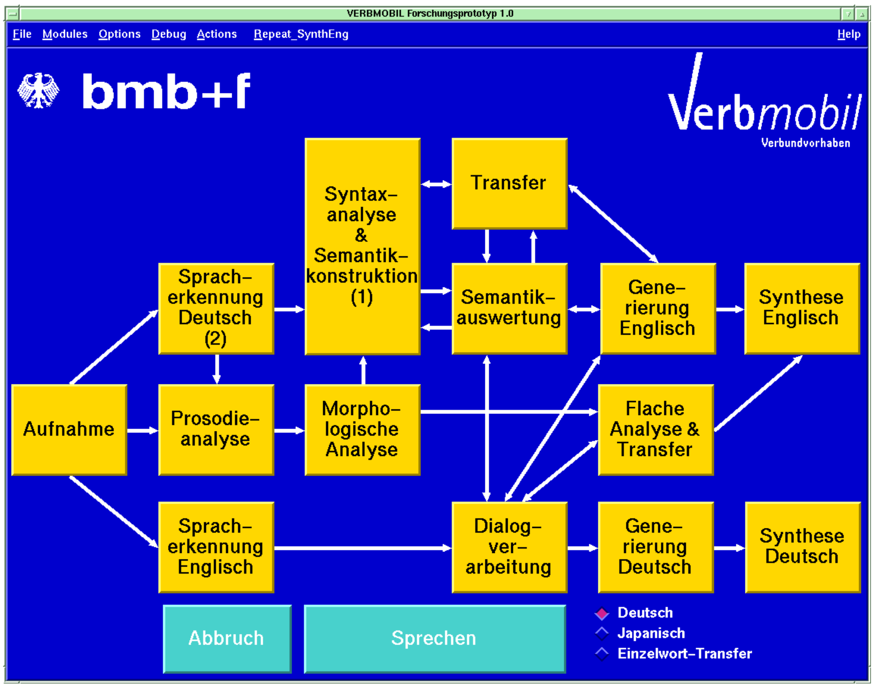


ABBILDUNG 7. Verbmobil-Benutzungsoberfläche

- Spracherkennung Englisch: Keywordspotting und Spracherkennung Englisch, Ja/Nein-Erkenner.
- Transfer: Transfer-Modul und Einzelwortübersetzer.
- Synthese Englisch: Selection und Synthese Englisch

3.2 Bedienung

3.2.1 Spracheingabe

Für eine Spracheingabe muß man den Schalter *Sprechen* mit der linken Maustaste gedrückt halten und die zu übersetzende Äußerung in das Mikrofon sprechen. Die Maustaste muß solange gedrückt werden, bis die Spracheingabe beendet wurde. Während der Eingabe sollten die Schalter *Sprechen* und *Aufnahme* ihre Farbe wechseln und in rot dargestellt sein. Wenn man nun die Maustaste losläßt (d.h. den Schalter *Sprechen* nicht mehr gedrückt hält), dann verfärben sich die Schalter *Aufnahme* und *Sprechen* wieder in gelb. Solange im weiteren Verlauf der Verarbeitung ein Modul aktiv ist, wird der entsprechende Schalter rot dargestellt. Der Ablauf der Verarbeitung einer Spracheingabe läßt sich also anhand der in rot dargestellten Schalter verfolgen. Das Endergebnis wird zuletzt an das Modul *Synthese Englisch* übergeben, das die ermittelte Übersetzung akustisch ausgibt.

Wenn während der Übersetzung eines Satzes Daten zwischen den einzelnen Modulen übergeben werden, erscheint an den entsprechenden Schnittstellen (dargestellt durch Pfeile zwischen den Schaltern) das Symbol eines Papierbogens. Durch Anklicken eines solchen Symbols werden die entsprechenden Daten in einem eigenen Fenster visualisiert (näheres siehe Kapitel 4 „Daten-Visualisierung“, Seite 24).

Wenn man einen Pfeil mit der mittleren Maustaste anklickt, so wird ein Dialog geöffnet, mit dem man Daten direkt an ein fachliches Modul senden kann. Mehr hierzu ist in Kapitel 5 „Testwerkzeuge“, Seite 29, zu finden.

Unter dem Menüpunkt *Help - Example Sentences* findet man Beispiel-Äußerungen aus der Domäne.

Sollte bei der Spracheingabe ein Fehler aufgetreten sein - wurde zum Beispiel zu laut oder zu leise gesprochen -, so wird dies als Fehlermeldung von der *Synthese Deutsch* ausgegeben. Ebenso wird eine akustische Meldung ausgegeben, wenn Probleme bei der Verarbeitung auftraten und keine Übersetzung gefunden werden konnte.

Soll die Verarbeitung einer Äußerung vorläufig abgebrochen werden, so kann man das erreichen, indem man den Schalter *Abbruch* links neben dem Schalter *Sprechen* mit der linken Maustaste anklickt.

Mit den Auswahlschaltern rechts neben dem Schalter *Sprechen* kann zwischen den einzelnen Betriebsmodi umgeschaltet werden.

3.2.2 Betriebsmodi

Für **Verbmobil** wurden drei verschiedene Betriebsmodi implementiert: *Deutsch, Japanisch* und *Einzelwort-Transfer*. Diese Modi lassen sich durch die Auswahlschalter rechts neben dem

Schalter *Sprechen* auswählen.

- Mit der Einstellung ***Deutsch*** werden die Sprachsignale von der Aufnahme an einen der deutschen Spracherkenner weitergeleitet. Wie unter Abschnitt 3.2.3.2 auf Seite 19 beschrieben ist, kann der Benutzer dabei zwischen verschiedenen Erkennern wählen.
- Mit der Einstellung ***Japanisch*** werden die Sprachsignale an den japanischen Spracherkenner weitergeleitet und zur Analyse die nun zuständigen linguistischen Module aktiviert.
- Mit der Einstellung ***Einzelwort-Transfer*** kann **Verbmobil** wie ein Lexikon benutzt werden. Als Spracheingabe wird nun ein einzelnes deutsches Wort erwartet. Die Übersetzung für das erkannte Wort wird von Verbmobil dann (akustisch) ausgegeben.

3.2.3 Konfigurierbarkeit der Module

Beim Anklicken folgender Schalter erhält man eine Dialogbox zur Konfiguration des betreffenden Moduls:

Aufnahme

Dialogverarbeitung

Generierung Englisch

Spracherkennung Deutsch

Spracherkennung Englisch

Syntaxanalyse & Semantikkonstruktion

Synthese Deutsch

Synthese Englisch

Semantikauswertung

Flache Analyse & Transfer

Die Dialogboxen enthalten eine Reihe von Steuerelementen zur Eingabe der Konfigurationsdaten und zwei Schalter **Ok** und **Cancel**. Bei dem Anklicken des **Ok**-Schalters werden die Einstellungen für das Modul bestätigt. Die aktuellen Änderungen der Einstellungen werden verworfen, wenn man den **Cancel**-Schalter anklickt. Was in den oben angegebenen Modulen zur Laufzeit eingestellt werden kann, wird im folgenden beschrieben.

3.2.3.1 Aufnahme

Bei dem Anklicken des Modul-Schalters *Aufnahme* erscheint ein Dialog mit den Auswahlschaltern

Erkenner GUI: GUI anzeigen: Aktivierung der Graphischen Oberfläche

Eingabe über Mikrophon

Eingabe aus Datei

Audiodaten permanent speichern

Wenn man *Eingabe aus Datei* ausgewählt hat, so muß man den Namen der Datei, die als

Eingabe dienen soll, in dem Textfeld in diesem Dialog eingeben. Den Dateinamen kann man auch aus einem File-Dialog auswählen, der geöffnet wird, wenn man den Schalter *Search* anklickt.

3.2.3.2 Spracherkennung Deutsch

In der Dialogbox für die Konfiguration der *Spracherkennung Deutsch* stehen folgende Auswahlschalter zur Verfügung:

Generator GUI - GUI anzeigen

Erkenner No. 1 (uka)

Erkenner No. 2 (db)

Mit den Schaltern *Erkenner No 1 / 2* wählt man den entsprechenden Erkenner aus. Der Schalter *GUI anzeigen* ist nur relevant, wenn der ausgewählte Erkenner über die entsprechende Funktionalität verfügt. Wenn diese Option ausgewählt ist, wird während der Verarbeitung (Suche) dynamisch die gegenwärtig beste Hypothese in einem Fenster angezeigt.

3.2.3.3 Spracherkennung Englisch

Hier kann ausgewählt werden, ob das Modul *Keywordspotting* oder *Spracherkennung Englisch* aktiviert werden soll.

3.2.3.4 Syntaxanalyse & Semantikkonstruktion

In der Dialogbox für die Konfiguration des Moduls *Syntaxanalyse & Semantikkonstruktion* können die entsprechenden Analysen ausgewählt werden. Wenn das Modul *synsem_s3 (1)* ausgewählt wurde, dann kann für dieses auch unter *Language Model* eine Option (*Bigram / Trigram / kein Language Model*) ausgewählt werden.

3.2.3.5 Semantikauswertung

Für dieses Modul kann die Option *Dialogakt vorhersagen* ausgewählt werden.

3.2.3.6 Dialogverarbeitung

Hier kann ausgewählt werden, ob für folgende Phänomene Klärungsdialoge aktiviert werden sollen:

Ähnliche Wörter

Lexikalische Ambiguitäten

Unbekannte Wörter

Inkonsistente Termine

Des weiteren kann hier die Anzeige der Grafischen Benutzungsoberfläche der Dialogverarbeitung aktiviert werden.

3.2.3.7 Generierung Englisch

Hier kann die Anzeige der Grafischen Oberfläche der englischen Generierung aktiviert werden.

3.2.3.8 Flache Analyse & Transfer

Hier kann die Anzeige der Grafischen Oberfläche für das Modul *Shallow* aktiviert werden.

3.2.3.9 Synthese Englisch

Der Schalter Synthese-Englisch repräsentiert auch das Modul *selection.* Deshalb gibt es in der Konfigurations-Dialogbox der englischen Synthese zwei Seiten für die Einstellungen: **Selection** und **Synthese Englisch**.

Auf der Seite **Selection** gibt es folgende Bereiche:

- *Eingabe*: hier kann eingestellt werden, von welchem Analyse/Transfer-Modul die Übersetzung an die Synthese Englisch weitergeleitet werden soll. Wird die Option *Automatische Auswahl* angewählt, so wird das Ergebnis mit der höchsten Bewertung übernommen.
- Selection GUI: GUI anzeigen

Auf der Seite **Synthese Englisch** können folgende Einstellungen gemacht werden:

- *Stimme: Grundfrequenz[Hz]*; hier kann die Höhe der ausgegebenen Stimme bestimmt werden.
- *Synthese: Ausgabe durch*: durch Auswahl eines in der Auswahlliste enthaltenen Programmnamens kann bestimmt werden, durch welches Programm die Ausgabe der englischen Synthese durchgeführt wird. Welches Programm benutzt werden kann, hängt von der Hardware-Konfiguration des Rechnersystems ab. Informationen hierzu sind in der Installations-Anweisung zu finden.
- *Audio Dispatcher: AudioDispatcher verwenden*: hier kann ausgewählt werden, ob die englische Synthese das Modul **AudioDispatcher** zur Koordinierung der Sprachausgabe benutzt.
- *Einstellungen (nur Truetalk):* bei Benutzung der Synthese **Truetalk** kann hier eingestellt werden, ob die Ausgabe auf Lautsprecher (*Speaker*), Kopfhörer (*Headphone*) oder externen Anschluß (*Line*) erfolgt, mit welcher Lautstärke die Ausgabe erfolgen soll, und welche Sampling-Rate benutzt werden soll.

3.2.3.10 Synthese Deutsch

Wie bei der **Synthese Englisch** kann hier der *Ausgabe*-Kanal (*Line*, *Headphone*, *Speaker*) ausgewählt, die *Lautstärke* und die *Sampling-Rate* eingestellt werden.

3.2.4 Menü

In den folgenden Abschnitten werden die Menü-Punkte der Verbmobil-Oberfläche beschrieben.

3.2.4.1 File - Exit Verbmobil

Mit dem Menüpunkt *File - Exit Verbmobil* wird das **Verbmobil**-System beendet. Es gibt auch stärkere Mittel, um das System zu beenden; siehe hierzu Abschnitt 2.3 „Start des Prototyps“, Seite 14.

3.2.4.2 Modules

In diesem Menü sind die einzelnen fachlichen Module von **Verbmobil** als Menüpunkte aufgelistet. Hier können alle Module ebenso wie an ihren Schaltern konfiguriert werden (siehe Abschnitt 3.2.3).

3.2.4.3 Options - Error Messages

Bei Auswahl dieses Menüpunktes erhält man eine Dialogbox zur Behandlung von Fehlermeldungen (*Show Error Messages*). Es ist konfigurierbar, ob bzw. ab welchem Level (1-10) die Fehler angezeigt werden sollen.

3.2.4.4 Options - Language

Unter diesem Menüpunkt kann die Beschriftung der Grafischen Benutzungsoberfläche zwischen deutsch- und englischsprachig umgestellt werden.

3.2.4.5 Options - Buttons - Drag ‘n Drop

Wenn dieser Punkt markiert ist, können die Schalter der fachlichen Module wahlweise verschoben werden. Die Position eines Schalters kann verändert werden, indem man den betreffenden Schalter mittels der rechten Maustaste an die gewünschte Position zieht.

3.2.4.6 Options - Buttons - Autom. Arrow Positioning

Wenn dieser Punkt markiert ist, werden die Datenfluß-Pfeile auf der Verbmobil-Oberfläche automatisch repositioniert.

3.2.4.7 Options - Buttons - Save Button Position

Eine neu gewählte Position der Schalter kann hier abgespeichert werden.

3.2.4.8 Options - Buttons - Load Button Position

Die abgespeicherte Einstellung von Schalter-Positionen kann hier wieder geladen werden.

3.2.4.9 Options - Show TAZ

Mit dieser Option kann getestet werden, ob die verwendete Hardware Verbmobil-tauglich ist. Läuft die ausgelöste Animation nicht ruckelfrei, so wird dringlichst empfohlen, auf leistungsfähigere Hardware umzusteigen.

3.2.4.10 Options - Show Result Windows

Bei der Markierung dieses Punktes werden während der Verarbeitung einer Äußerung die Ergebnisse der einzelnen Module jeweils in einem separaten Fenster angezeigt. Vorsicht: nur für Demonstrationszwecke geeignet.

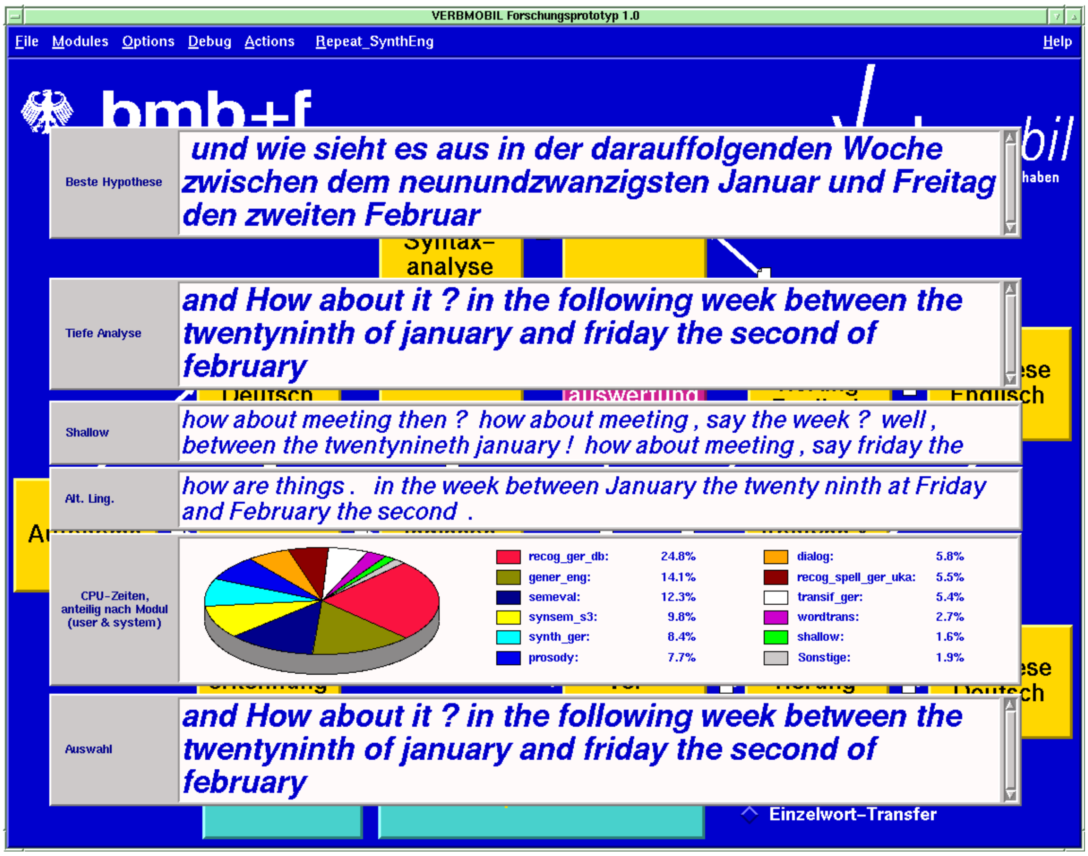


ABBILDUNG 8. Result-Windows / Resource-Window

3.2.4.11 Options - Show Resource Window

Über diesen Menüpunkt kann ein Fenster zur Anzeige der relativen Anteile der einzelnen Module an der Gesamtverarbeitungs-CPU-Zeit aktiviert werden (siehe ABBILDUNG 8.).

3.2.4.12 Debug - Show communication

Bei der Auswahl dieses Menüpunktes wird ein Fenster geöffnet, in dem die ausgetauschten Nachrichten in dem Programmsystem zwischen den vom TBM gesteuerten Modulen und dem Modul `TBM` angezeigt werden. Zusätzlich enthält das Fenster ein Textfeld, in dem spezielle Nachrichten versenden werden können. Das allgemeine Format der Nachrichten ist

<Empfänger> <Nachricht> <Parameter>

Das Setzen des Eingabe-Dateinamens für die Aufnahme, würde zum Beispiel so aussehen:

recording audio_input file /project/verbmobil/VmProto/Beispiel/datei.dat

Dieser Befehl setzt die Eingabe auf die Datei /project/verbmobil/VmProto/Beispiel/datei.dat.

3.2.4.13 Debug - Show Module States

Bei der Auswahl dieses Menüpunktes wird ein Fenster aktiviert, das die Zustände der einzelnen gestarteten fachlichen Module anzeigt. Dabei bedeutet der Zustand *unknown*, daß ein Modul die Initialisierung (Startphase) noch nicht abgeschlossen hat, der Zustand *ready*, daß ein Modul betriebsbereit aber nicht aktiv ist und der Zustand *active*, daß ein Modul gerade aktiv ist.

3.2.4.14 Actions - Restart Verbmobil

Bei Auswahl dieses Menüpunktes wird eine Dialogbox geöffnet, in der bestimmt werden kann, ob und welche Module heruntergefahren und noch einmal neu gestartet werden sollen. Die Module, die in dieser Dialogbox in dem linken Listenfenster stehen, werden bei einem Mausklick auf den Schalter *Restart* lediglich die Nachricht `restart` erhalten, während die Module in dem rechten Listenfenster neu gestartet werden. Die Module, die neu gestartet werden sollen, müssen in dem linken Listenfenster angeklickt werden.

3.2.4.15 Actions - Reset Configuration

Die Einstellungen, die beim Start von Verbmobil als Default eingestellt sind, werden wieder gesetzt, dadurch werden alle manuell vorgenommenen Veränderungen der Einstellungen zurückgesetzt.

3.2.4.16 Actions - Resource Query

Wird dieser Menüpunkt angewählt, so bestimmen alle Module ihre verbrauchten System- und Benutzer-Zeiten und (optional) ihre Speicher-Resourcen. Diese Werte werden nach Beendigung von Verbmobil in einer Statistik-Datei zu finden sein. Zusätzlich wird das Ressourcen-Diagramm (siehe 3.2.4.11 „Options - Show Resource Window“, Seite 22) aktualisiert.

3.2.4.17 Repeat_SynthEngl

Das Modul Synthese Englisch erhält den Befehl `repeat`, worauf es den zuletzt gesprochenen Satz wiederholt.

3.2.4.18 Help - Example Sentences

Es wird ein Fenster mit Beispiel-Äusserungen aus der Domäne geöffnet.

3.2.4.19 Help - Consortium

Das *Intro-Fenster wird* erneut angezeigt. Dazu wird die **Verbmobil**-Benutzungsoberfläche geschlossen. Um diese wieder zu aktivieren, muß der Schalter am unteren Rand des *Intro-Fensters* angeklickt werden.

3.2.4.20 Help - Consortium + Welcome

Bei dieser Auswahl wird wie bei *Help - Consortium* das Intro-Fenster geöffnet Zusätzlich wird beim Wechsel in die **Verbmobil**-Benutzungsoberfläche der Begrüßungstext gesprochen (wie beim Start von **Verbmobil**).

3.2.4.21 Help - Say Info Text

Bei der Auswahl dieses Menüpunktes wird von dem Modul *Synthese Deutsch* ein Informationstext über **Verbmobil** ausgegeben.

Kapitel 4

Daten-Visualisierung

4.1 Daten-Archivierung

Die vom VIM an den Daten-Schnittstellen des Systems abgezweigten Nachrichten werden unter

$VM_TMP/VIM_Archiv/

archiviert. Die Art der Archivierung und die Namensbausteine der archivierten Pakete sind in der Standardkonfiguration $VM_HOME/etc/VmConfig unter „MODULE vim“ jeweils in den „COMMENT:“-Slots der Daten-Kanäle spezifiziert.

4.1.1 Namenskonvention

Der im o.g. Slot enthaltene Namens-Token ist der *Grundbaustein* für die Namen der Archiv-Dateien. Er hat z.B. die Gestalt

`synsem__transfer` oder
`dialog_Z_shallow`,

spezifiziert also (u.U. verkürzt auf einen mehrdeutigen Namen) die verbunden Module. Befindet sich ein „Z“ zwischen den beiden Unterstrichen, so werden unter gleichem Namen Nachrichten in beiden Richtungen entlang des Kanals gespeichert; anderenfalls handelt es sich um eine unidirektionale Archivierung.

4.1.1.1 Namen der archivierten Dateien

Die Namen der im VIM-Archiv angelegten Archiv-Dateien haben die folgende allgemeine Form:

<Grundbaustein>[<Formats-Suffix>][<Archiv-Suffix>]

Erläuterungen:

- Grundbaustein: s.o.
- Formats-Suffix: ist in der Regel leer, nur bei der Archivierung von Prolog-Termen wird ggf. das Suffix „.txt“ oder „.pp“ angehängt; siehe unter 4.1.3 „Prolog Specials“, Seite 26.
- Archiv-Suffix: hierfür gibt es folgende Varianten:

kein Suffix: alle archivierten Nachrichten der Schnittstelle sind hier (getrennt durch Zeilenumbruch) abgespeichert.

„.toc“: es handelt sich um ein Inhaltsverzeichnis zu dem gleichnamigen File ohne dieses Suffix; es enthält für jede Nachricht einen einzeiligen Eintrag bestehend aus dem Nachrichten-Namen für die Extraktion (siehe unter 4.1.1.2 „Namenskonvention für die Nachrichtenextraktion“, Seite 25), der Start- und Stop-Position sowie weiteren Informationen zu der Nachricht.

„.app“: bei der Archivierungsart `append_to` wird zusätzlich dieses File angelegt, in dem sowohl die Nachrichten selbst, als auch Informationen zu den Nachrichten abgespeichert werden.

4.1.1.2 Namenskonvention für die Nachrichtenextraktion

Aus den Sammelarchiven lassen sich bei Bedarf später wieder einzelne Nachrichtenpakete extrahieren. Die Namen dieser extrahierten Pakete haben die folgende Form:

<Grundbaustein><Nachrichten-Suffix>[<Formats-Suffix>]

Das Nachrichten-Suffix hat die Form „_TurnId_ModuleId“, ist also ein eindeutiger Spezifizierer der Nachricht; Grundbaustein und Formats-Suffix wie oben.

Zur Extraktion von Nachrichten aus Archiv-Dateien steht das Werkzeug *arex* mit dem Aufruf:

$VM_HOME/opt/global/bin/arex [-t <turn_id_1> ... -t <turn_id_k>] <File_1> ... <File_n>

zur Verfügung. *File_1* bis *File_n* sind Archiv- oder „.toc“-Dateien. Es können alle Nachrichten extrahiert werden oder nur die Nachrichten zu bestimmten Turns.

4.1.2 Archivierungsarten

Jeder Schnittstelle ist eine der folgenden Archivierungsarten zugeordnet:

1. `write_to`: Normale Archivierung. Es entsteht ein File ohne Archiv-Suffix, das alle Nachrichten enthält, sowie ein Inhaltsverzeichnis mit dem Suffix „.toc“.

2. `append_to`: Zusätzlich entsteht ein File mit dem Suffix „.app“, in dem die eigentlichen Nachrichten mit zusätzlichen Informationen annotiert sind.

3. Spezialfall: Für Phondat-Signale entsteht nur ein File ohne Archiv-Suffix; es enthält zur Schonung der Speicherkapazitäten immer nur das letzte übertragene Audio-Signal.

4.1.3 Prolog Specials

Bei der Archivierung von Prolog-Termen werden Dateien mit verschiedenen Suffixen anglegt.

4.1.3.1 Übertragung mit idl_term

Formats-Suffix	Inhalt
keines	VIM-internes Format mit expliziten Koreferenzen
.pp	Pretty Print durch VIT-ADT
.txt	writeq-Ausgabe-Fromat

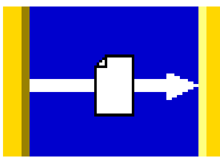
4.1.3.2 Übertragung von idl_string oder idl_longstring

Formats-Suffix	Inhalt
keines	Originalstring
.pp, .txt	wie oben

4.2 Visualisierung im Testbed

4.2.1 Über Paper-Icons auf der GUI

Wenn Daten über eine Schnittstelle geflossen sind, so erscheint auf der GUI das Icon:



Durch Anklicken mit der linken Maustaste werden die Daten in einem geeigneten Viewer dargestellt. Wenn die Daten einer Schnittstelle dargestellt worden sind, so verfärbt sich das Icon (ursprünglich weiß) nach blau. Es werden immer nur Daten zum letzten an der Schnittstelle beobachteten Turn angezeigt.

4.2.1.1 Visualisierung bei mehreren Segmenten / Nachrichten in einem Turn

Wenn mehrere getrennt gespeicherte Nachrichten zu einem Turn existieren (z.B. mehrere VITs für einzelne Segmente), so wird beim ersten Anklicken die erste Nachricht / das erste Segment angezeigt, beim zweiten Anklicken die / das zweite u.s.w.

Sind noch weitere Nachrichten / Segmente vorhanden, so bleibt das Icon weiß; es verfärbt sich erst dann blau, wenn alle Nachrichten / Segmente gesehen wurden.

4.2.2 Unsichtbare Schnittstellen

Schnittstellen, für die kein Pfeil auf der GUI vorhanden ist, können über das Communication Info der GUI (siehe Kapitel 3 „Benutzeroberfläche und Bedienung“, unter 3.2.4.12 „Debug - Show communication“, Seite 22) angezeigt werden, mittels der Nachricht

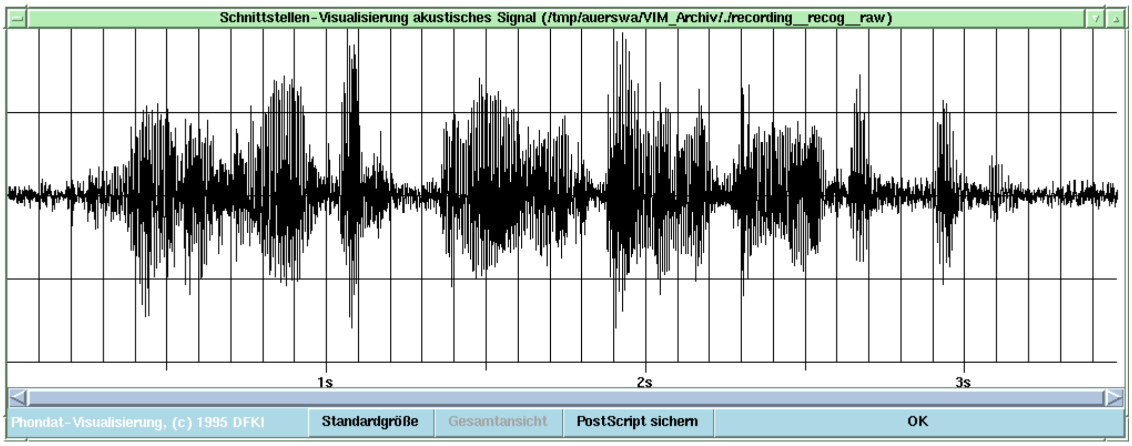
vim show_data <from> <to>

Die jeweils gültigen Identifier für <from> und <to> sind der Standardkonfiguration $VM_HOME/etc/VmConfig unter „PARAMETERS vim“ oder den „data_available“-Nachrichten im Communication Info zu entnehmen.

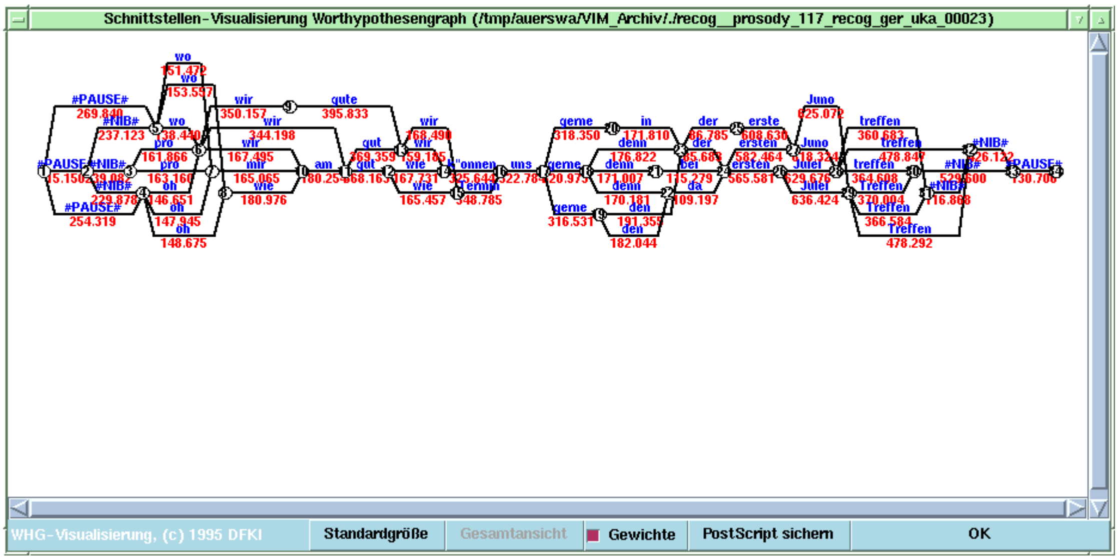
4.2.3 Die Viewer

In das Verbmobil-Testbed sind die folgenden Viewer integriert:

4.2.3.1 Viewer für Phondat-Audio-Signale

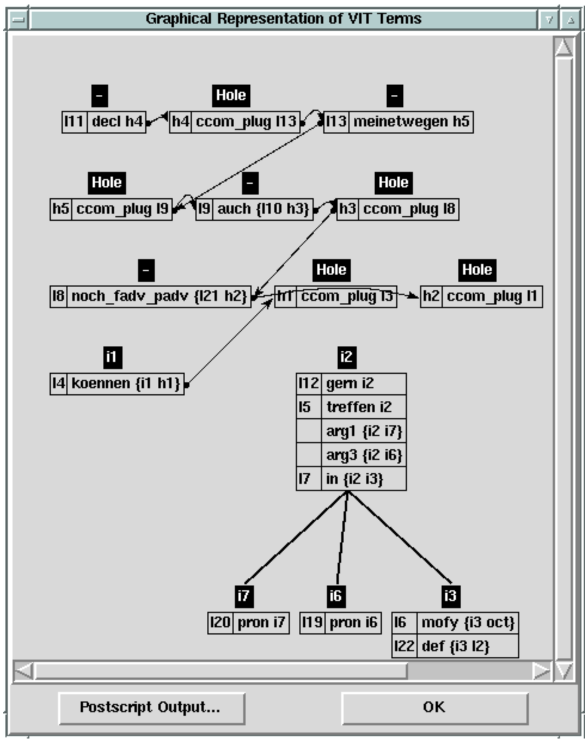


4.2.3.2 Viewer für Worthypothesengraphen

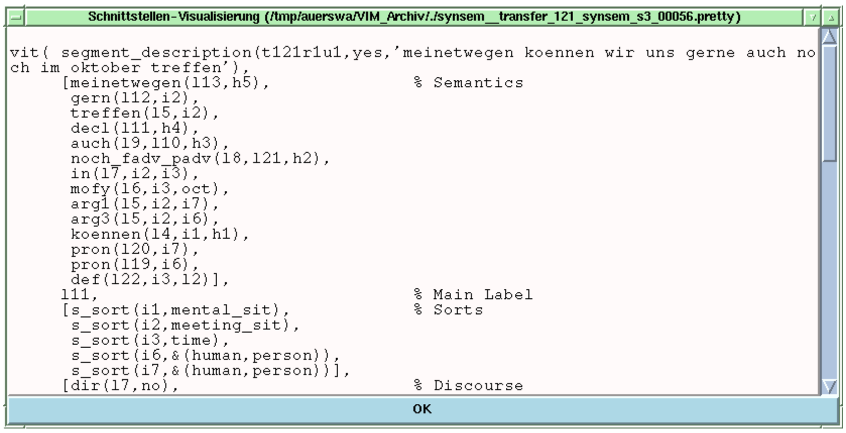


4.2.3.3 Viewer für Verbmobil-Interface-Terme (VITs)

graphisch:



als Pretty Print unter Verwendung des VIT-ADT




```
vit( segment_description(t121r1u1,yes,'meinetwegen koennen wir uns gerne auch no
ch im oktober treffen'),
      [meinetwegen(l13,h5),              % Semantics
       gern(l12,i2),
       treffen(l5,i2),
       decl(l11,h4),
       auch(l9,l10,h3),
       noch_fadv_padv(l8,l21,h2),
       in(l7,i2,i3),
       mofy(l6,i3,oct),
       arg1(l5,i2,i7),
       arg3(l5,i2,i6),
       koennen(l4,i1,h1),
       pron(l20,i7),
       pron(l19,i6),
       def(l22,i3,l2)],
      l11,                               % Main Label
      [s_sort(i1,mental_sit),            % Sorts
       s_sort(i2,meeting_sit),
       s_sort(i3,time),
       s_sort(i6,&(human,person)),
       s_sort(i7,&(human,person))],
      [dir(l7,no),                       % Discourse
```


4.2.3.4 Viewer für sonstige Nachrichten

Nachrichten in anderen Formaten werden textuell in einem Textfenster ausgegeben.

4.3 Standalone Visualisierung

Der Viewer „whgvis“ für Worthypothesengraphen ist als Standalone-Tool unter $VM_HOME/opt/global/bin installiert. Aufruf:

$VM_HOME/opt/global/bin/whgvis <whg_file_name>

Kapitel 5

Testwerkzeuge

5.1 Einspeisen fachlicher Nachrichten

Der VIM bietet die Möglichkeit, Nachrichten in die Daten-Schnittstellen des Systems einzuspeisen. Dazu klickt man mit der mittleren Maustaste auf einen Pfeil auf der GUI, wodurch das „Send Control Window“ des VIM mit Vorauswahl des Kanals erscheint:

Es kann nur an denjenigen Schnittstellen eingespeist werden, die in der Standard-Konfiguration $VM_HOME/etc/VmConfig entsprechend deklariert sind. Eine Auswahl aller erlaubten Schnittstellen erhält man, wenn man auf den „Clear“-Button klickt.

In den Listen „From“ und „To“ können das Modul, dessen Ausgabe simuliert werden soll, sowie das Modul, an das gesendet werden soll, eingestellt werden. Gibt es zwischen den Modulen mehrere Kanäle, so kann die Auswahl in der „Name“-Liste getroffen werden.

Der „Swap“-Button vertauscht die voreingestellten Einträge im „Channel Selection Filter“; dies kann bei bidirektionalen Kanälen sinnvoll sein. Der Daten- und Übertragungstyp wird normalerweise automatisch eingestellt; sollte er durch die Kanal-Auswahl nicht eindeutig spezifiziert sein, so kann eine Nachspezifikation erfolgen.

Hinter dem Select-Button verbirgt sich eine File Selection Box für die einfache Auswahl von Daten Files.

Achtung: Wird die Nachricht durch Editieren im „Message Window“ erzeugt, so ist zu beachten, daß alle eingegebenen Zeilenumbrüche erhalten bleiben und mit der Nachricht verschickt werden.

5.1.1 Vergabe der Turn-Id

Die Vergabe der Turn-Id erfolgt bei einem normalen Lauf des Gesamtsystems durch das Modul Aufnahme („recording“). Werden jedoch Nachrichten im „Send Control Window“ (SCW) verschickt, so übernimmt dadurch der Benutzer / Tester die Kontrolle über die Turn-Id-Vergabe.

Das „Turn Id“-Feld im SCW wird automatisch voreingestellt auf die bisher höchste Turn-Id einer fachlichen Nachricht im System. Für einen neuen Turn muß die Turn-Id vom Benutzer stets erhöht werden, da sonst ggf. die Verarbeitung der als veraltet angenommenen Nachricht von einigen Modulen verweigert wird (häufige Fehlerquelle beim manuellen Testen).

Es kann jederzeit wieder in den normalen Betriebsmodus (Einsprechen über das Mikrofon) zurückgewechselt werden. Die Kontrolle über die Vergabe der Turn-Id wird automatisch an das Modul Aufnahme abgegeben; die erforderliche Synchronisation zwischen den Modulen VIM und Aufnahme erfolgt transparent.

5.1.2 Nachrichten vom Datentyp „prolog“ oder „v-i-term“

Wenn der Datentyp der zu verschickenden Nachricht „prolog“ oder „v-i-term“ ist, so sollte der Daten-Term im Standard-Prolog-Format vorliegen bzw. eingegeben werden, also so, wie er von „writeq“ ausgegeben würde (großgeschriebene Atome in Apostroph eingeschlossen), **ohne** einen sog. Full Stop am Ende.

5.1.3 Special: String-Eingabe in die SynSem

Die Eingabe für die SynSem besteht normalerweise in einem Worthypothesengraphen. Wird an der Schnittstelle Morphologie->SynSem eine Nachricht eingespeist, so wird automatisch detektiert, ob es sich um einen WHG handelt. Liegt kein WHG vor, so wird die Nachricht für String-Eingabe gehalten.

String-Eingabe wird automatisch in einen linearen WHG mit geeigneten Zeit- und Gewichtparametern überführt, so daß die SynSem stets einen WHG als Eingabe erhält.

Die gleiche automatische String-WHG-Konvertierung findet übrigens an allen WHG-Schnittstellen statt. Die dabei vorgenommene prosodische Annotierung des WHG ist nicht an allen diesen Schnittstellen sinnvoll (z.B. recog_ger_*->prosody); sie führt aber auch nicht zu Fehlverhalten, da sie vom Modul ‚prosody‘ ignoriert wird.

5.1.3.1 Details zur String-WHG-Konvertierung

Der Eingabe-String soll die Wörter enthalten, die auch in dem Wörtergitter stehen sollen. Weitere Eingaben gemäß der Transliterationen (Satzabbruch o. ä.) können nicht aufgelöst werden, außer es handelt sich um die Satzzeichen „.,?“ oder das Symbol <G3>. Ein Punkt (.) wird in dem Wörtergitter als eine harte Grenze an dem vorangehenden Wort markiert, ein Komma als eine „weiche“ Grenze. Ein Fragezeichen bewirkt die Einführung einer harten Grenze und Annotation mit dem Fragemodus. Das Symbol <G3> wird genauso wie ein Punkt behandelt. Die Satzzeichen können separat stehen oder direkt folgend auf das letzte Wort. Die drei Strings

```
,das ist ein Punkt.‘
,das ist ein Punkt . ,
,das ist ein Punkt <G3>‘
```

würden zu dem gleichen Ergebnis führen.

Die Unter- und Obergrenze der Kosten werden mit den Eigenschaften MinCosts und MaxCosts durch einen Zufallszahlen-Generator berechnet. Per Default sind diese Werte 0 und 150.

5.2 Einspeisen von Steuer-Nachrichten

Steuer-Nachrichten, die vom TBM an die Module gesendet werden sollen, lassen sich im Communication Info der GUI verschicken (siehe Kapitel 3 „Benutzeroberfläche und Bedienung“, unter 3.2.4.12 „Debug - Show communication“, Seite 22).

5.3 Das Automatische Testmodul

Mit Hilfe des automatischen Testmoduls lassen sich kontrolliert große Datenmengen in das System einspeisen und große Mengen von Testdaten generieren. Das Einspeisen der Nachrichten erfolgt dabei über den VIM, so daß die Bemerkungen in Kapitel 5.1 zu beachten sind. Das automatische Testmodul ist ausführlich in Kapitel 6 „Testen mit dem Automatischen Testmodul (ATM)“, Seite 33, beschrieben.

5.4 Daten-Aufbereitung nach umfangreichen Testläufen

Folgende Perl-Skripte befinden sich unter $VM_HOME/opt/global/bin. Die Aufrufstruktur der Skripte (Parameter, Flags...) kann mit Hilfe des Aufrufs ‘<Skript_Name> ---’ abgefragt werden.

5.4.1 Extraktion der entstandenen Dateien

Das Perl-Skript ‘extractResults’ dient der nachträglichen Strukturierung der bei einem automatischen Testlauf entstehenden Daten. Diese sind im Verzeichnis $VM_HOME/VIM_Archiv gespeichert, und zwar existiert für jede Schnittstelle ein Datenfile <xyz> und ein zugehöriges Inhaltsverzeichnis <xyz>.toc. Aus diesen beiden Dateien wird für jeden Turn während eines Testlaufs eine Datei erzeugt, in der alle Daten enthalten sind, die während dieses einen Turns an der Schnittstelle geflossen sind.

Zur besseren Strukturierung der (zahlreich) entstehenden Dateien kann der während des automatischen Testens vergebene symbolische Name herangezogen werden. Dieser spiegelt i.a. den kompletten Pfad-Namen der Eingabedatei wieder, so daß die entstandenen Daten entsprechend der Eingabestruktur angeordnet werden können. Zu diesem Zweck wird die vom VIM erzeugte Datei 'statistics.vim' benötigt, die im Verzeichnis $VM_TMP automatisch angelegt wird.

Aufruf:
extractResults <vim_statistics_datei> <toc_datei>+ -to <ziel_verzeichnis>

Die Ergebnisse werden im Zielverzeichnis entsprechend den symbolischen Namen der Eingabedateien abgelegt. So werden z.B. die Daten, die bei der Verarbeitung des Turns nps1k008.a16 aus dem Dialog n023k geflossen sind, in folgendem Verzeichnis abgelegt:
<ziel_verzeichnis>/n023k/nps1k008.a16/<datei_name>

5.4.2 Erzeugung phondat-korrelierter Transliterationsdateien

Die Zuordnung von Transliterationen zu den zugehörigen Audio-Dateien ist nicht immer ganz einfach. Insbesondere bei der automatischen Verarbeitung führte die Verwendung der Original-Transliterationen daher zu Schwierigkeiten. Um diese zu beheben, gibt es das Skript 'createPhondatTrlFiles', mit dessen Hilfe Transliterationsdateien erstellt werden können, deren Index genau dem Namen der ursprünglichen Audio-Datei entspricht. Wir sprechen in diesem Zusammenhang dann von phondat-korrelierten Transliterationen.

Aufruf:
createPhondatTrlFiles <dialog>+ -o <ausgabe_verzeichnis> -i <eingabe_verzeichnis>

Hierbei ist <dialog> ein Verzeichnis, in dem sich die Audio-Dateien für den entsprechenden Dialog befinden. Für jeden so spezifizierten Dialog muß sich im Eingabeverzeichnis eine Datei <dialog>.trl mit den Original-Transliterationen befinden. Erzeugt wird eine Datei namens <dialog> im Ausgabeverzeichnis, die phondat-korrelierte Transliterationen enthält.

5.4.3 Aufarbeitung der Logdatei des Verbmobil-Prototypen

Die Ausgaben aller Module des Prototypen werden während der Verarbeitung in die Datei vm.log im Verzeichnis $VM_TMP geschrieben. Da diese Logdatei durch die Vielzahl der Ausgaben i.a. sehr unübersichtlich wird, gibt es die Möglichkeit, sie mit Hilfe des Skripts 'splitLog' in ihre Einzelteile aufzuteilen.

Aufruf:
splitLog

Erwartet wird eine Datei vm.log im aktuellen Verzeichnis. Falls diese nicht existiert, wird im Verzeichnis $VM_TMP gesucht. Erzeugt wird ein Verzeichnis 'singleLogs', in dem die Ausgaben der einzelnen Module enthalten sind (pro Modul eine Datei). Dabei werden die Zeilennummern aus der ursprünglichen Datei mit übernommen, so daß beliebige Modulkombinationen über den Befehl 'sort -m <file_1> ... <file_n>' erstellt werden können.

Kapitel 6

Testen mit dem Automatischen Testmodul (ATM)

6.1 Einführung

Das automatische Testmodul stellt ein Werkzeug zur Verarbeitung großer Datenmengen innerhalb des Verbmobil-Systems dar. Es ist Teil der Testumgebung und dazu gedacht, das gesamte System, oder auch einzelne Module bzw. Modulgruppen unabhängig von anderen fachlichen Modulen, zu testen.

Das in der ICE-Kommunikation realisierte Konzept der ‘splitted channels’ ist eine wesentliche Voraussetzung für das automatische Testmodul, da es ein problemloses An- und Abschalten der Testfunktionalität ermöglicht (ein Eintrag in der Konfigurationsdatei genügt).

Innerhalb des Testbeds hört das automatische Testmodul den Kanal zwischen GUI und Testbedmanager ab. Dort fließen einerseits Konfigurationsnachrichten vom Modul ‘gui’ an die fachlichen Module, in der anderen Richtung die Statusmeldungen der einzelnen Module.

Daher kann das automatische Testmodul:

i) Konfigurationsnachrichten in das System einspeisen

ii) Auf bestimmte Systemzustände reagieren (Absturz eines Moduls, ‘turnend’ etc.)

Abbildung 6-1 zeigt, wie das automatische Testmodul in das Gesamtsystem integriert ist. Da der Kanal zwischen Testbedmanager und graphischer Oberfläche vollständig umgeleitet wird, können zusätzlich Nachrichten vom Modul ‘gui’ an einzelne Module herausgefiltert werden (z.B. der Befehl an das Modul ‘recording’, mit der Aufnahme von Daten zu beginnen).

Die Arbeitsweise des automatischen Testmoduls wird über Konfigurationsdateien gesteuert. Diese Dokumentation soll die Arbeitsweise des automatischen Testmoduls erläutern und dem Benutzer seinen Gebrauch ermöglichen.

Das wesentliche Ziel dieses Dokuments ist es, die Erstellung von Konfigurationsdateien zu erläutern. Bedauerlicherweise leiden formale Beschreibungen oft unter einem Mangel an Verständlichkeit, umgangssprachliche Erklärungen und Beispiele hingegen lassen manchmal einige Details vermissen. Daher findet sich in dieser Anleitung beides.

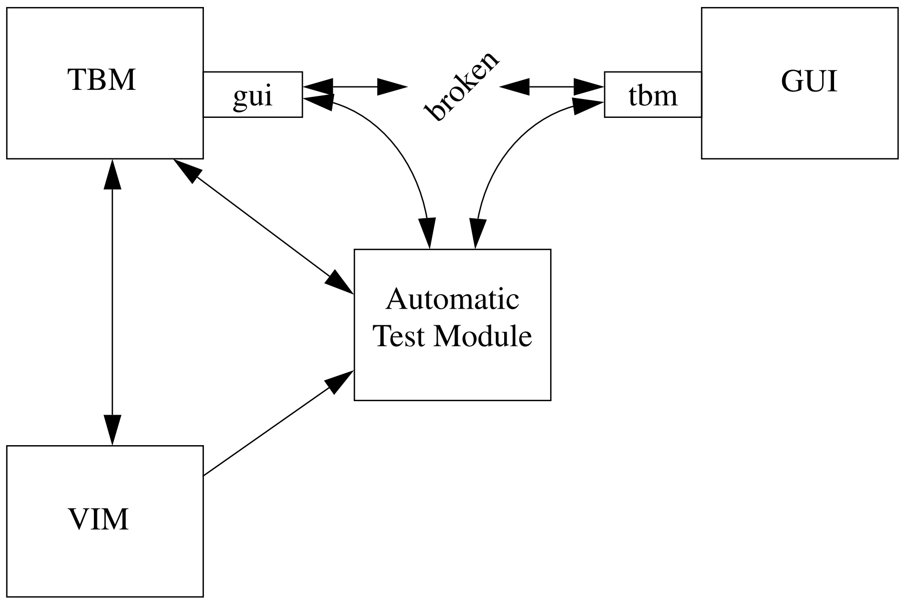


Abbildung 6-1: Automatische Verarbeitung im Testbed

Empfehlenswert ist es, zunächst die Grundbegriffe (z.B.: Sendeblock, Konfiguration) zu verstehen. Danach sollte es anhand der Beispiele möglich sein, relativ problemlos eigene Konfigurationsdateien zu erstellen. Zur Klärung von Detailfragen dienen die formalen Beschreibungen.

6.1.1 Motivation

Die Durchführung regelmäßiger Testläufe während der (Weiter-)Entwicklung von Software ist unerläßlich. Das Ziel dieser Tests ist die Beurteilung des momentanen Entwicklungsstandes (z.B.: funktionale Abdeckung oder Robustheit) der Programme:

- was ist neu dazugekommen
- was ist geblieben
- was ist (evtl.) verlorengegangen

Daraus ergibt sich die erste Anforderung an eine automatische Testkomponente: Die durchgeführten Tests müssen leicht wiederholbar sein. Die Erstellung einer Konfigurationsdatei, die immer wieder verwendet werden kann, bietet diese Funktionalität: Die einzugebenden Daten und die Steuerung des Verarbeitungsablaufs werden hierin festgelegt.

Ein automatischer Testlauf muß auch robust sein. Der Absturz eines fachlichen Moduls sollte zu einem Neustart dieses Moduls führen. Anschließend ist die Verarbeitung fortzusetzen. Auch diese Funktionalität ist in der vorliegenden Version implementiert.

Weiterhin muß die Spezifikation eines Tests einfach und flexibel sein. Zu diesem Zweck wurde folgendes Konzept verwirklicht:

Ein Test besteht aus einer oder mehreren Konfigurationen, die weiter unterteilt sind in verschiedene Sendeblöcke.

Innerhalb der Konfiguration wird festgelegt, welche fachlichen Module im Verbmobil-Prototyp zu verwenden sind (z.B. 'synsem_s3' oder 'synsem_ibm').
Wie bereits erwähnt, werden die Statusnachrichten einzelner Module (z.B.: 'active', 'inactive') vom Testbedmanager an die Oberfläche mit abgehört. Daher können Statusmeldungen als Bedingungen angegeben werden, anhand derer das automatische Testmodul die Verarbeitung eines Turns (d.h. eines Konfigurationsschrittes) als abgeschlossen ansieht.

Innerhalb eines Sendeblocks werden sämtliche Daten spezifiziert, die an *einer* Schnittstelle während der Abarbeitung einer Konfiguration (d.h. während mehrerer Turns) verschickt werden sollen. Die Daten werden Schritt für Schritt abgearbeitet, in jedem Turn wird eine Dateneinheit verschickt. Zusätzlich können Bedingungen festgelegt werden, die eine Abarbeitung des nächsten Sendeblocks bewirken. Es werden also dann weitere Daten an *anderen* Schnittstellen innerhalb *eines* Turns verschickt.

Sicherlich wird dieses Konzept nicht alle Anforderungen befriedigen können. Manche werden umständlich in Szene gesetzt werden müssen, manche überhaupt nicht realisierbar sein. Es hat sich jedoch gezeigt, daß viele verschiedene Testarten problemlos durchgeführt werden können. Zudem wurde im vorliegenden Ansatz darauf geachtet, daß eine Erweiterung (z.B. um zusätzliche Sendeblock*typen*) möglichst problemfrei vorgenommen werden kann.

6.1.2 Sendeblock und Konfiguration

Ein **Sendeblock** beinhaltet alle Daten, die von *einem* bestimmten Sender an *einen* bestimmten Empfänger geschickt werden sollen. Somit enthält er im wesentlichen die Namen der beiden beteiligten Module sowie die Spezifikation von Dateneinheiten (z.B. Dateinamen oder Strings). Diese zu verschickenden Daten sind sequentiell als Liste angeordnet und ein **Verarbeitungsschritt** besteht im Verschicken der ersten Dateneinheit dieser Liste.

Bei der **Aktualisierung** eines Sendeblocks wird die erste Dateneinheit der Liste entfernt. Ein Sendeblock ist vollständig abgearbeitet, wenn die Liste der Dateneinheiten leer ist.

Eine **Konfiguration** faßt verschiedene Sendeblöcke zu einer logischen Einheit zusammen. Die Sendeblöcke sind als sequentiell abzuarbeitende Liste organisiert. Abbildung 6-2 zeigt den Ablauf während der Abarbeitung einer Konfiguration.

Hinweis: Nicht alle Sendeblöcke müssen dieselbe Menge von Dateneinheiten enthalten. Falls ein Block vollständig abgearbeitet ist, wird er aus der Konfiguration entfernt, die Verarbeitung geht jedoch weiter, solange noch mindestens ein nichtleerer Sendeblock existiert.

Innerhalb eines **Konfigurationsschrittes** wird für alle in der Konfiguration enthaltenen Sendeblöcke (in der Reihenfolge ihres Auftretens) *ein* Verarbeitungsschritt durchgeführt. Im allgemeinen entspricht ein Konfigurationsschritt *einem* Turn.

Eine Konfiguration ist vollständig abgearbeitet, wenn alle enthaltenen Sendeblöcke vollständig abgearbeitet sind.

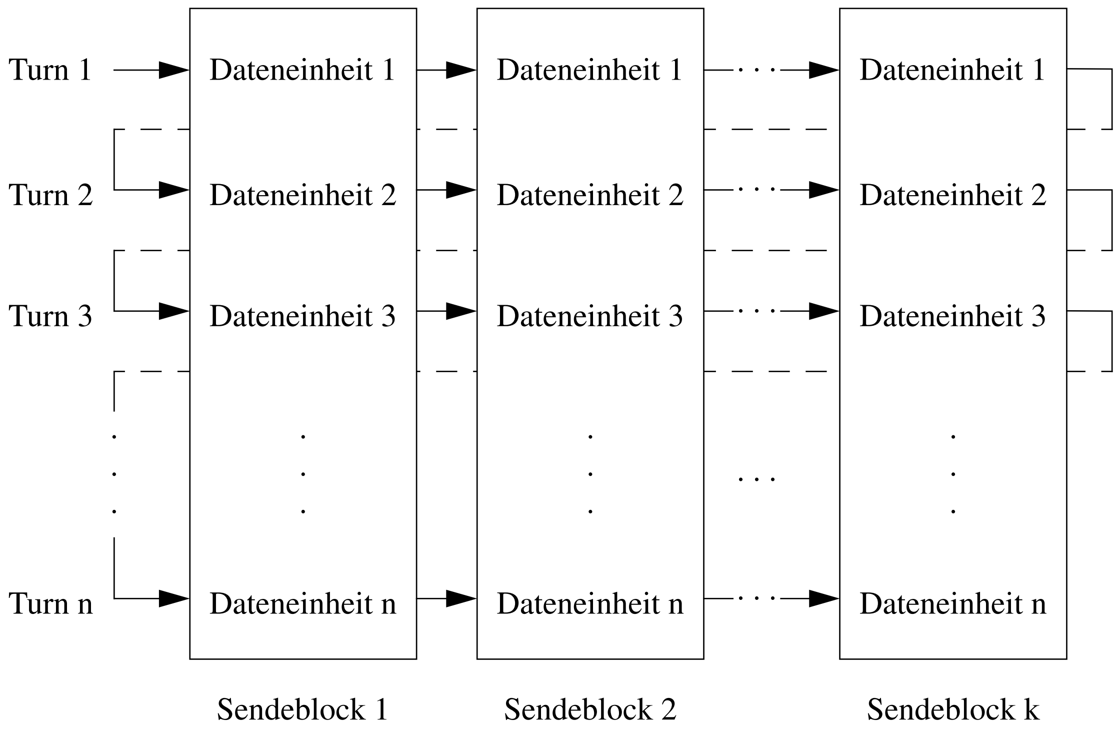


Abbildung 6-2: Abarbeitung einer Konfiguration

6.1.3 Die Abarbeitung einer Konfiguration

Die Abarbeitung einer Konfiguration beinhaltet, wie bereits kurz erläutert wurde, die Versendung aller in den Sendeblöcken spezifizierten Dateneinheiten. Der Informationsfluß wird durch sogenannte Fortsetzungsbedingungen gesteuert: Timeout und Trigger-Nachrichten. Wir unterscheiden dann jeweils noch einmal zwischen lokalen und globalen Bedingungen. Diese Begriffe werden unten anhand der Durchführung eines Konfigurationsschrittes erläutert.

Innerhalb eines Konfigurationsschrittes wird für alle Sendeblöcke die erste spezifizierte Dateneinheit verschickt. Der Übergang von einem Sendeblock zum nächsten wird durch lokale Bedingungen gesteuert, der von einem Konfigurationsschritt zum nächsten durch globale Bedingungen. Daher werden globale Bedingungen innerhalb der Konfiguration spezifiziert, lokale innerhalb der einzelnen Sendeblöcke.

Abbildung 6-3 illustriert (etwas vereinfacht) die Abläufe während der Durchführung eines Konfigurationsschrittes:

(1) Solange die Liste der zu verarbeitenden Sendeblöcke ('to-do-Liste') nicht leer ist:

> Verschicke die aktuelle Dateneinheit des ersten Sendeblocks aus der to-do-Liste
>
> Falls innerhalb dieses Sendeblockes eine lokale Bedingung angegeben ist, erfolgt der Übergang zum nächsten Sendeblock erst dann, wenn diese Bedingung eingetreten ist: beim Eintreffen einer lokalen Trigger-Nachricht bzw. nach Ablauf einer festgelegten Zeit (lokaler Timeout).
> Im Gegensatz hierzu erfolgt der Übergang zum nächsten Sendeblock *sofort*, falls inner-

halb eines Sendeblocks *keine* lokale Bedingung angegeben ist.
Entferne den aktuellen Sendeblock aus der to-do-Liste und füge ihn zur Liste der bereits abgearbeiteten Sendeblöcke (‘done-Liste’) hinzu.

Bei Eintreten einer globalen Bedingung wird sofort zum nächsten Konfigurationsschritt übergegangen (Schritt 2): Alle im aktuellen Konfigurationsschritt spezifizierten Sendeblöcke werden als erledigt angesehen, insbesondere auch die, deren aktuelle Dateneinheit noch nicht verschickt wurde.

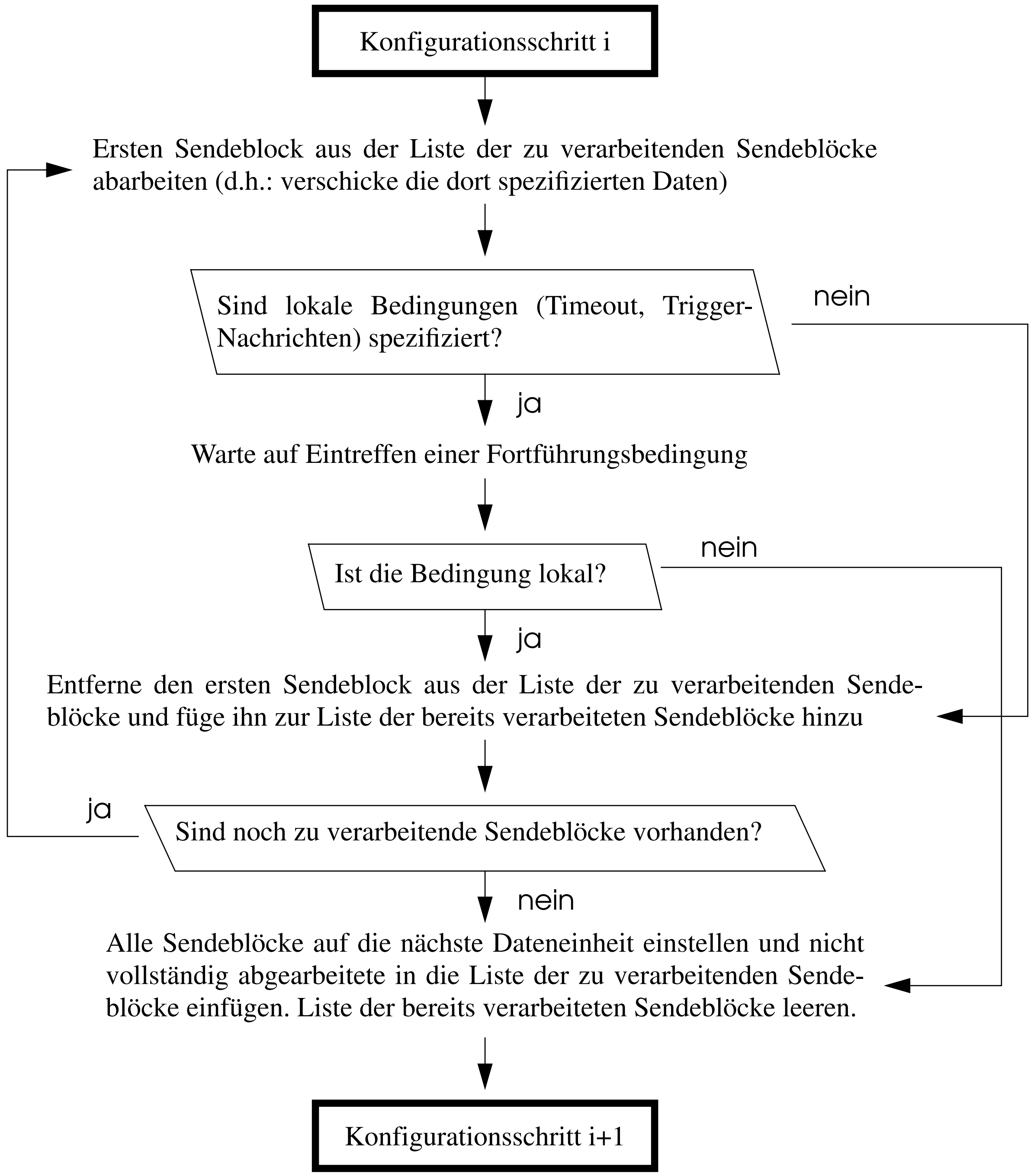


Abbildung 6-3: Ausführung eines Konfigurationsschrittes

(2) Entferne alle (eventuell übrig gebliebenen) Sendeblöcke aus der to-do-Liste, füge sie zur done-Liste hinzu und aktualisiere sie (Einstellen auf die nächste Dateneinheit). Fahre mit dem nächsten Konfigurationsschritt fort.

Hinweis: Die Vielzahl der anfallenden Daten während eines Testlaufs sind erfahrungsgemäß für den Menschen sehr unübersichtlich. Daher schickt das automatische Testmodul während der Verarbeitung einen symbolischen Namen an den Visualisierungsmanager, damit anhand dieser symbolischen Turn-Id eine spätere Sortierung vorgenommen werden kann. Wie dieser Name gewählt wird, ist vom ersten Sendeblock der Konfiguration abhängig und kann den entsprechenden Kapiteln zu den verschiedenen Sendeblock*typen* entnommen werden.

6.1.3.1 Kriterien für die *erfolgreiche* Durchführung eines Schrittes

Bei Eintreffen einer globalen Trigger-Nachricht wird der aktuelle Konfigurationsschritt automatisch als ‘korrekt beendet’ angesehen.

Anders ist die Situation bei einem globalen Timeout: hier besteht die Möglichkeit, daß ein oder mehrere Module während der Datenverarbeitung ‘stecken geblieben’ sind. Daher ergibt sich (in Abweichung zur vereinfachten Darstellung in Abbildung 6-3) folgende Vorgehensweise bei globalem Timeout:

1) Überprüfe, ob noch irgendein Modul aktiv gemeldet ist.

2) Falls ‘nein’ wird der Konfigurationsschritt als erfolgreich angesehen und eine entsprechende Ausgabe gemacht.

3) Falls ‘ja’, muß davon ausgegangen werden, daß erstens die Verarbeitung nicht erfolgreich war und zweitens das System sich unter Umständen in einem inkonsistenten Zustand befindet: Es wird ein ‘cancel’-Befehl an alle Module geschickt.

    3.1) Falls maximal zwanzig Sekunden später alle Module inaktiv gemeldet sind, kann mit der Verarbeitung fortgefahren werden (nächster Konfigurationsschritt).

    3.2) Andernfalls werden alle Module, die noch aktiv gemeldet sind, durch einen Befehl an den Testbedmanager beendet und dann wieder neu gestartet. Alle anderen Module erhalten den ‘restart’-Befehl. Anschließend wird erst dann mit der Verarbeitung fortgefahren, wenn *erneut* eine globale Fortsetzungsbedingung erfüllt ist!

Das automatische Testmodul registriert auch die außerplanmäßige Beendung eines fachlichen Moduls (‘Absturz’). Auch in diesem Fall wird der aktuelle Konfigurationsschritt als nicht erfolgreich angesehen. Das beendete Modul wird via Testbedmanager neu gestartet, alle anderen Module erhalten den ‘restart’-Befehl. Die Verarbeitung wird erst fortgesetzt, wenn eine globale Fortsetzungsbedingung eintritt.

6.2 Die Konfigurierung des automatischen Testmoduls

Dieses Kapitel stellt alles Wissen zusammen, das zum Erstellen einer Konfigurationsdatei notwendig ist. Hierbei wurde Wert auf Vollständigkeit gelegt. Wer nicht an den einzelnen Details interessiert ist, kann direkt zu den Beispielen übergehen und bei Bedarf nachblättern.

6.2.1 Aufbau einer Konfigurationsdatei

Das automatische Testmodul handelt beliebig viele, vom Benutzer spezifizierte, Konfigurationen ab. Die Spezifikation der Konfigurationen erfolgt in Dateien, die dem automatischen Testmodul entweder beim Aufruf als Parameter übergeben werden oder während einer Verbmobil-Sitzung mittels einer speziellen Nachricht. Jede angegebene Datei kann beliebig viele Konfigurationen enthalten. Eine Kommentarzeile ist eine beliebige Zeile, deren erstes nicht-leeres Zeichen ein '#' ist.

Der Aufbau einer Konfigurationsdatei sieht folgendermaßen aus:

<optional: Leer- oder Kommentarzeilen>
KONFIGURATION
<optional: Leer- oder Kommentarzeilen>
KONFIGURATION
<optional: Leer- oder Kommentarzeilen>
...
KONFIGURATION
<optional: Leer- oder Kommentarzeilen>

Eine Konfigurationsspezifikation selbst darf Leerzeilen und Kommentarzeilen enthalten. Jede Zeile kann beliebig weit eingerückt sein. Die Zeilenlänge ist auf 32K (32768 Zeichen) beschränkt.

6.2.2 Aufbau einer Konfiguration

Eine Konfiguration wird folgendermaßen spezifiziert:

CONFIGURATION_BEGIN
System-Konfigurationsnachrichten
Globaler Timeout
Globale Trigger-Nachrichten
Zusätzliche Befehle
Sendeblöcke
CONFIGURATION_END

Der Aufbau der Elemente einer Konfiguration wird in den nächsten Unterkapiteln erläutert. Ein Sendeblock ist so komplex, daß ihm ein eigenes Kapitel gewidmet wird.

Bemerkungen:

- Alle Elemente einer Konfiguration sind optional, insbesondere ist die leere Konfiguration
  CONFIGURATION_BEGIN
  CONFIGURATION_END
  syntaktisch korrekt (wenn auch nicht sonderlich sinnvoll).

- Bei mehrfacher Spezifikation gilt: Timeout und zusätzliche Befehle werden überschrieben (letztes Vorkommen zählt). Alle anderen Elemente müssen, von Ihrer Idee her, mehrfach spezifizierbar sein.

6.2.2.1 System-Konfigurationsnachrichten

System-Konfigurationsnachrichten dienen dazu, das Verbmobil-System für die automatische Verarbeitung einzurichten. Manche der Einstellungen, die normalerweise auf der Oberfläche vorgenommen werden, können dadurch automatisch angepaßt werden (z.B.: welcher kontinuierliche Spracherkenner oder welche ‘synsem’ zu benutzen ist).

Aufbau:
SYSTEM_CONFIGURATION "Nachricht_1" ... "Nachricht_n"

Die Nachrichten müssen in doppelte Anführungszeichen eingeschlossen sein und dürfen selbst keine doppelten Anführungszeichen enthalten. Auf das Schlüsselwort SYSTEM_CONFIGURATION muß mindestens *eine* Nachricht folgen.

Das Schlüsselwort SYSTEM_CONFIGURATION darf innerhalb einer Konfigurationsbeschreibung beliebig oft vorkommen. Es gibt keinen Unterschied zwischen den Spezifikationen

SYSTEM_CONFIGURATION "Nachricht_1" "Nachricht_2"
und
SYSTEM_CONFIGURATION "Nachricht_1"
SYSTEM_CONFIGURATION "Nachricht_2"

Erlaubte System-Konfigurationsnachrichten sind:
recogleft recog_ger_uka
recogleft recog_ger_db
recogleft wordrecog_ger_uka
recogleft wordrecog_ger_db
recogleft recog_jap_uka
recogright keywordspotter
recogright recog_eng_uks
synsem synsem_ibm
synsem synsem_s3

Die korrekte Angabe von System-Konfigurationsnachrichten wird auch in den Beispielen erläutert. Grundsätzlich ist es vorteilhaft, System-Konfigurationsnachrichten in jede Konfiguration mit aufzunehmen. Damit ist gewährleistet, daß die erstellte Konfigurationsdatei auch nach einer eventuellen Umstellung der Default-Werte im Verbmobil-Prototyp einwandfrei funktioniert.

6.2.2.2 Globaler Timeout

Bewirkt den sofortigen Übergang zum nächsten Konfigurationsschritt. Ein globaler Timeout soll sicherstellen, daß ein automatischer Testlauf nicht dadurch unterbrochen wird, daß ein Modul während der Verarbeitung unbemerkt "steckenbleibt" (und damit auch keine eventuell spezifizierten Trigger-Nachrichten mehr sendet).

Aufbau:
TIMEOUT <n>

Hierbei steht <n> für die Anzahl der Sekunden, nach denen der Timeout wirksam werden soll.

Wie bereits angesprochen, wird ein Konfigurationsschritt, der durch einen globalen Timeout beendet wird, nur dann als erfolgreich angesehen, wenn zum Zeitpunkt des Timeout kein Modul mehr aktiv war (für weitere Information siehe Kapitel 6.1.3.1). Darin unterscheidet er sich wesentlich von einem lokalen Timeout.

6.2.2.3 Globale Trigger-Nachrichten

Das Eintreten einer globalen Trigger-Bedingung bewirkt den sofortigen Übergang zum nächsten Konfigurationsschritt:
Alle Sendeblöcke (auch die noch nicht abgearbeiteten!) werden als erledigt angesehen und auf die nachfolgende Dateneinheit eingestellt.

Aufbau:
a) TRIGGER "Nachricht"
oder
b) TRIGGER "Nachricht_1" ... "Nachricht_n"

Im Fall b) wird eine Konjunktion von Trigger-Nachrichten spezifiziert: Erst wenn alle angegebenen Nachrichten während der Verarbeitung eintreffen, gilt die globale Trigger-Bedingung als erfüllt.
Beachte:

- Eine Konjunktion gilt nur dann als erfüllt, wenn alle in ihr spezifizierten Nachrichten mit der gleichen Turn-Id eintreffen.

- Die Angaben

  TRIGGER "Nachricht_1"
  TRIGGER "Nachricht_2"
  und
  TRIGGER "Nachricht_1" "Nachricht_2"

  unterscheiden sich wesentlich voneinander! Im ersten Fall wird beim Eintreffen *einer* der Nachrichten mit dem nächsten Konfigurationsschritt fortgefahren, im zweiten Fall nur dann, wenn "Nachricht_1" *und* "Nachricht_2" eintreffen.

Zum Thema ‘Vorrangigkeit von globalen vor lokalen Trigger-Nachrichten’ siehe Kapitel 6.2.3.6.

6.2.2.4 Zusätzliche Befehle

Die zusätzlichen Befehle dienen dazu, dem Benutzer einige weitere Konfigurierungsmöglichkeiten zugänglich zu machen.

Aufbau:
ADDITIONAL <ORDER>

Momentan als <ORDER> erlaubt:

DOUBLE_PROCESSING:
Falls ein Konfigurationsschritt fehlschlägt (Tod eines Moduls, Module noch aktiv bei globalem Timeout), wird er noch genau einmal wiederholt.
Idee: Manche Module neigen zum Absturz, wenn sie viele Daten verarbeitet haben. Wenn sie dann neu gestartet werden, kann es passieren, daß dieselben Daten, die vorher zum Absturz führten, nun verarbeitet werden können.

SUBDIR <directory_name>:
Die bei der automatischen Verarbeitung entstehenden Daten werden nicht in $VM_TMP/VIM_Archiv, sondern dort im Unterverzeichn in ‘directory_name’ abgelegt.
Idee: Die Vielzahl der entstehenden Daten führt zu einer für den Menschen recht unübersichtlichen Struktur. Dieser Befehl bietet bei Verarbeitung mehrerer Konfigurationen die Möglichkeit, zumindest deren Daten klar voneinander abzugrenzen.

STARTING_POINT <suffix>:
Mit diesem Befehl kann ein String angegeben werden. Alle Dateneinheiten des ersten spezifizierten Sendeblocks werden daraufhin entfernt, bis der spezifizierte String ein Suffix der ersten Dateneinheit ist. Danach werden in allen weiteren Sendeblöcken (falls es solche gibt) ebenso viele Dateneinheiten eliminiert, wie im ersten Block.
Idee: Falls die automatische Verarbeitung aus irgendwelchen Gründen unterbrochen wurde, kann man das automatische Testmodul mit derselben Konfiguration starten; es muß nur zusätzlich das erste zu verarbeitende Element angegeben werden.
Der Startpunkt wird als Suffix spezifiziert, damit er so kurz wie möglich angegeben werden kann. Das bedeutet, daß der Startpunkt und die Dateneinheiten des ersten Sendeblocks *rechtsbündig* aneinandergelegt werden. Dann werden so viele Dateneinheiten übersprungen, bis eine Übereinstimmung mit dem Startpunkt (match) vorliegt.
Wenn etwa alle Dateien von der Form <xyz>000 bis <uvw>999 sind, reicht die Angabe 157 als Startpunkt. Falls aber z.B. die Dialoge ‘g071a’ und ‘g073a’ verarbeitet werden sollen und der Startpunkt die Datei ‘gtisa002.a16’ aus ‘g073a’ sein soll, so muß mindestens das Suffix "3a/gtisa002.a16" angegeben werden, da eine Datei mit Namen ‘gtisa002.a16’ auch in ‘g071a’ vorkommt.
Falls der erste Sendeblock vom Typ ‘STRINGS’ ist (siehe hierzu Kapitel 6.2.3.3), wird kein Suffix angegeben, sondern die Nummer des ersten zu bearbeitenden Strings (>0).

CONTINUE_IF_ALL_INACTIVE <n>:

Wenn alle Module für mindestens <n> Sekunden inaktiv gemeldet sind, wird der Konfigurationsschritt als beendet angesehen. Voraussetzung ist, daß sich mindestens ein Modul während eines Konfigurationsschrittes aktiv meldet.

<u>Idee</u>: Es kann unter Umständen schwierig sein, vorherzusehen, ob eine bestimmte Nachricht, die als Trigger-Nachricht fungieren soll, während der Verarbeitung gesendet wird oder nicht. Daher gibt es diesen Befehl als zusätzliche Möglichkeit, eine globale Fortsetzungsbedingung zu definieren.

STARTUP_TIMEOUT <n>:

Beim Hochfahren des Verbmobil-Systems und nach einem ‘restart’ wird maximal <n> Sekunden gewartet, bis die Verarbeitung begonnen bzw. fortgesetzt wird. Falls dieser zusätzliche Befehl nicht spezifiziert ist, wird der Wert des globalen Timeout auch als Startup-Timeout benutzt.

<u>Idee</u>: Manche Module brauchen sehr viel Zeit zum Hochfahren, zur Verarbeitung eines Turns jedoch vergleichsweise wenig. Dann kann der globale Timeout dennoch relativ klein gehalten werden (um eine hohe Verarbeitungsquote zu garantieren), ohne daß nach einem ‘restart’ Daten verschickt werden, bevor alle Module bereit sind.

Weiterhin ist die Verwendung eines Timeout beim (Wieder-)Hochfahren unter Umständen unerwünscht. Um das zu erreichen kann man <n> auf sehr große Werte setzen (z.B.: 1.000.000), ohne während der Verarbeitung eines Turns auf einen Timeout verzichten zu müssen.

KILL_ACTIVE_MODULES <n>:

Das automatische Testmodul sieht einen Turn u.a. dann als korrekt beendet an, wenn eine globale Trigger-Nachricht eintrifft. Normalerweise wird dann sofort mit dem nächsten Konfigurationsschritt fortgefahren. Falls eine Konfiguration aber diesen zusätzlichen Befehl enthält, wird nach einer globalen Trigger-Nachricht bis zu <n> Sekunden gewartet. Alle Module die dann noch aktiv sind werden beendet und neu gestartet, die Verarbeitung wird unterbrochen bis das System wieder einsatzbereit ist.

<u>Idee</u>: Es kommt vor, daß bestimmte Module, die nur unterstützende Aufgaben haben, während der Verarbeitung hängenbleiben. Dies läßt sich auf diese Weise schnell und relativ sicher erkennen und beheben. Es ist jedoch darauf zu achten, daß der Wert für <n> nicht zu klein gewählt wird, da manche Module unter Umständen einige Sekunden Nachlaufzeit benötigen.

6.2.2.5 Beispiele für den grundsätzlichen Aufbau von Konfigurationen

Beispiel 1:

Leere Konfiguration. Syntaktisch korrekt, ansonsten überflüssig.

```
CONFIGURATION_BEGIN
CONFIGURATION_END
```

Beispiel 2:

Keine Besonderheit.

```
CONFIGURATION_BEGIN
    SYSTEM_CONFIGURATION "recogleft recog_ger_uka" "synsem synsem_s3"
    <SENDEBLOCK>
    TIMEOUT 900
    TRIGGER "tbm turnend"
    TRIGGER "tbm all_ready"
CONFIGURATION_END
```

Beispiel 3:

Mehrfachspezifikation des globalen Timeouts: die letzte Angabe ist gültig.

```
CONFIGURATION_BEGIN
    SYSTEM_CONFIGURATION "recogleft recog_ger_uka"
    SYSTEM_CONFIGURATION "synsem synsem_s3"
    <SENDEBLOCK>
    TIMEOUT 200
    TIMEOUT 12
    TIMEOUT 900
    TRIGGER "tbm turnend"
    TRIGGER "tbm all_ready"
CONFIGURATION_END
```

Beispiel 4:

Benutzung einer Konjunktion von Trigger-Nachrichten.

```
CONFIGURATION_BEGIN
    SYSTEM_CONFIGURATION "recogleft recog_ger_uka"
    <SENDEBLOCK>
    TIMEOUT 900
    TRIGGER "shallow inactive" "ali inactive"
CONFIGURATION_END
```

Beachte: Beispiel 2 und Beispiel 3 spezifizieren die gleiche Konfiguration: Zum nächsten Konfigurationsschritt wird gegangen, wenn *eine* der beiden Nachrichten "tbm turnend" oder "tbm all_ready" verschickt wird. Anders ist es in Beispiel 4: Hier werden die Nachrichten "shallow inactive" und "ali inactive" als Konjunktion angesehen. Der Übergang zum nächsten Konfigurationsschritt erfolgt erst, wenn *beide* Nachrichten verschickt wurden.

6.2.3 Aufbau eines Sendeblocks

Ein Sendeblock beinhaltet die Spezifikation, welche Daten an welcher fachlichen Schnittstelle des Verbmobil-Systems verschickt werden sollen. Zusätzlich wird festgelegt, durch welche (lokalen) Bedingungen die Verarbeitung des nächsten Sendeblocks innerhalb der aktuellen Konfiguration angestoßen wird. Falls keine lokalen Bedingungen angegeben werden, erfolgt der Übergang zum nächsten Sendeblock sofort.

SENDING_BLOCK_BEGIN TYPE <Sending_Block_Type>
    Beteiligte Module
    Datenspezifikation
    Lokaler Timeout
    Lokale Trigger-Nachrichten
    Zusätzliche Befehle
SENDING_BLOCK_END

Gültige Werte für <Sending_Block_Type> sind ‘FILES’, ‘STRINGS’ und ‘FILE_CONTENT’. Die entsprechenden Sendeblöcke unterscheiden sich lediglich in der Datenspezifikation. Wie diese in den einzelnen Fällen auszusehen hat, ist in den entsprechenden Unterkapiteln beschrieben.

Bemerkungen:

- Alle Elemente eines Sendeblocks sind optional, insbesondere ist
  SENDING_BLOCK_BEGIN TYPE (FILES | STRINGS | FILE_CONTENT)
  SENDING_BLOCK_END
  syntaktisch korrekt.
  Ein Sendeblock ohne Datenspezifikation macht nicht viel Sinn: er wird bereits vor dem ersten Konfigurationsschritt eliminiert werden.
  Ein Sendeblock, bei dem keine beteiligten Module spezifiziert sind, ist lediglich von geringer Bedeutung: Ein solcher Sendeblock wird keinen Einfluß auf den Testverlauf nehmen, wohl aber u.U. auf die Vergabe der symbolischen Turn-Id.

- Bei mehrfacher Spezifikation gilt: Timeout, zusätzliche Befehle, beteiligte Module und Datenspezifikationen vom Typ ‘FILE_CONTENT’ werden überschrieben (letztes Vorkommen zählt). Die Datenspezifikationen der anderen Typen (‘FILES’ und ‘STRINGS’) sind mehrfach erlaubt (die Dateneinheiten zusätzlicher Spezifikationen werden zu denen vorhergehender mit hinzugenommen). Trigger-Nachrichten müssen, von ihrer Idee her, mehrfach spezifizierbar sein.

6.2.3.1 Die Spezifikation der beteiligten Module

Zunächst gilt es anzugeben, an welcher Schnittstelle des Verbmobil-Systems die Daten eingespeist werden sollen: sendendes Modul, empfangendes Modul, Name des ICE-Kanals.

Aufbau:
    PARTICIPANTS <module_from> <module_to> <channel_name>

Der Kanalname ist i.a. einfach das Wort ‘BASE’ (der Basiskanal zwischen den beteiligten Modulen). Eine Ausnahme bilden alle Kanäle, die zum Versenden von Audio-Rohdaten (Typ: IDL_AUDIO_RAW) gedacht sind: Hier gilt für den Kanalnamen die Konvention
    "module_from"_"module_to"_raw.
Das gilt für alle Kanäle von der Aufnahme zu einem der Erkenner oder zur Prosodie.

Bsp.: recording_recog_ger_uka_raw ist der Name des Kanals von der Aufnahme zum kontinuierlichen Karslruher Spracherkenner.

6.2.3.2 Sendeblock vom Typ ‘FILES’

Ein Sendeblock vom Typ ‘FILES’ kommt zum Einsatz, wenn die zu verschickenden Daten in Form von Dateien vorliegen.

Aufbau der Datenspezifikation:
SEND_FILES <pattern>+

<pattern> ist ein Muster, aus dem eine Menge von Namen erstellt wird nach den Regeln des ‘pattern matching’ in der standard shell (sh), also im Wesentlichen:

*	ersetzt einen String beliebiger Länge (auch Länge 0), ersetzt nicht ein dot (‘.’) am Anfang eines Namens.
?	ersetzt genau ein Zeichen.
[...]	ersetzt jedes der innerhalb der Klammern angegebenen Zeichen. Ein durch ein ‘-’ getrenntes Buchstaben- oder Ziffernpaar ersetzt alle Buchstaben bzw. Ziffern, die lexikalisch zwischen den beiden Zeichen (inklusive) liegen. Falls das erste Zeichen innerhalb der Klammern ein ‘!’ ist, werden alle Zeichen ersetzt, die *nicht* in Klammern eingeschlossen sind.

Bsp.:	n00[129]k	matcht n001k, n002k und n009k
	n0*k	matcht alle Namen, die mit ‘n0’ starten und mit ‘k’ enden
	[!n]*	matcht alle Namen, die nicht mit ‘n’ starten
	?001k	matcht alle Namen der Länge 5, die auf ‘001k’ enden

Für weitere Informationen siehe manual pages für ‘sh’.

Der Sendeblock erzeugt aus den angegebenen "pattern" die dazugehörigen Pfadnamen nach obigen Regeln. Dann wird eine Liste L mit allen *flachen* Dateien erstellt, die später verschickt werden. Zu diesem Zweck wird jeder der erhaltenen Pfadnamen p_name folgendermaßen weiterverarbeitet:

1) p_name ist eine flache Datei und beginnt nicht mit einem dot (‘.’): Füge p_name zu L hinzu

2) p_name ist ein Verzeichnis und der Name eines Dialogs: Ermittle alle Dateinamen die auf CD im entsprechenden Dialog zu finden sind und durchsuche p_name gezielt nach flachen Dateien dieser Namen. Falls eine entsprechende Datei gefunden wird, wird sie zu L hinzugefügt, andernfalls erfolgt die Ausgabe einer Warnung.

3) p_name ist ein Verzeichnis und nicht Name eines Dialogs: Alle enthaltenen Dateien werden alphabetisch sortiert aufgelistet und dann gemäß Fall 1 bis 3 rekursiv weiter verarbeitet.

Anmerkung zu Fall 2:
Motivation: Die alphabetische Sortierung spiegelt vornehmlich auf CD1.0.3 *nicht immer* den Dialogverlauf wider. Da es beim Testen auf die Reihenfolge der Turns innerhalb eines Dialogs ankommen kann, versucht das automatische Testmodul eben diese Reihenfolge beizubehalten, wenn ein ganzer Dialog verarbeitet wird.
Zu diesem Zweck sucht es unter einem festen Pfad ‘TRL_PATH’ nach Dialognamen. Alle dort befindlichen flachen Dateien werden als eine Art Transliterationsdatei (ohne

Zusatz .trl) für einen Dialog desselben Namens angesehen, müssen aber statt der üblichen Indizes (z.B. TIS000) die Namen der dazugehörigen Dateien auf CD enthalten (z.B. gtisa000.a16).
Für den ‘TRL_PATH’ gilt:

- der default Wert ist "/IDoNotExist", kann aber mit der Kommandozeilenoption ‘-T <newPath>’ auf jeden beliebigen Wert gesetzt werden.
- jeder Sendeblock kann einen eigenen Wert für den Transliterationspfad festlegen (siehe Kapitel 2.3.7).
- falls dieser Wert auf ein nichtexistierendes Verzeichnis zeigt, tritt immer Fall 3 ein (alphabetische Sortierung).

Anstelle der Verwendung von Transliterationsdateien bleibt natürlich immer noch die Erstellung eines Verzeichnisses, in dem die alphabetische Reihenfolge der Dateien die Reihenfolge im Dialog widerspiegelt. Das kann erreicht werden, indem man die Originaldateien kopiert und umbenennt. Möglich ist auch die Verwendung von ‘soft links’.

Bemerkung zur Wahl der symbolischen Turn-Id:
Falls ein Sendeblock vom Typ ‘FILES’ der erste Block einer Konfiguration ist, wird der absolute Pfadname der jeweils verschickten Datei auch als symbolische Turn-Id gewählt (vgl.: Hinweis in Kapitel 6.1.3).
Für spätere Sortierungszwecke ist es daher vorteilhaft, die vom Visualisierungsmodul archivierten Daten durch entsprechende Unterstrukturen zu gliedern (siehe auch unter ’ADDITIONAL SUBDIR’ in Abschnitt 6.2.2.4 „Zusätzliche Befehle“). Dies gilt insbesondere, wenn während der Abarbeitung verschiedener Konfigurationen manche Dateien mehrfach verschickt werden (z.B. einmal an den Karlsruher Erkenner und einmal an den von Daimler-Benz).

6.2.3.3 Sendeblock vom Typ ‘STRINGS’

Ein Sendeblock vom Typ ‘STRINGS’ dient dazu, Strings zu verschicken, die innerhalb des Sendeblocks definiert werden.

Aufbau der Datenspezifikation:
SEND_DATA "<string_1>" ... "<string_k>"

Jeder der (in doppelten Anführungszeichen!) spezifizierten Strings bildet in diesem Fall eine Dateneinheit. Der String selbst darf keine doppelten Anführungsstriche enthalten. Sendeblöcke dieses Typs sind dazu gedacht, kleinere Datenmengen zu verschicken. Zu beachten ist auch, daß die Strings nicht durch Zeilenumbrüche getrennt werden dürfen. Mehrere Datenspezifikationen innerhalb eines Sendeblocks sind jedoch erlaubt.

Bemerkung zur Wahl der symbolischen Turn-Id:
Falls ein Sendeblock vom Typ ‘STRINGS’ der erste Block einer Konfiguration ist, wird als symbolische Turn-Id (vgl.: Hinweis in Kapitel 1.2) der Name ‘String_#i’ verwendet, wobei i kontinuierlich hochgezählt wird.

6.2.3.4 Sendeblock vom Typ ‘FILE_CONTENT’

Ein Sendeblock vom Typ ‘FILE_CONTENT’ dient dazu, Strings zu verschicken, die in einer Datei definiert werden.

Aufbau der Datenspezifikation:
SEND_CONTENT <content_file_name>

Hierbei gibt <content_file_name> den Namen der Datei an, die die zu verschickenden Daten enthält. Eine Dateneinheit ist ein Block, d.h. der Inhalt dieser Datei wird blockweise verschickt, jeder Block sequentiell Zeile für Zeile.

Ein Block ist eine Menge von aufeinanderfolgenden Zeilen, die alle nicht leer sind. Verschiedene Blöcke werden durch eine oder mehrere Leerzeilen getrennt. Zu beachten ist, daß eine Zeile nicht länger als 32K (32768 Zeichen) sein darf. Weiterhin kann jeder Block beliebig viele Kommentarzeilen enthalten (Zeilen, deren erstes nichtleeres Zeichen ein '#' ist). Ein Block, der ausschließlich aus Kommentarzeilen besteht, gilt als leerer Block, d.h. es wird in diesem Turn an dieser Schnittstelle keine Dateneinheit verschickt.

Bemerkung zur Wahl der symbolischen Turn-Id:
Falls ein Sendeblock vom Typ 'FILE_CONTENT' der erste Block einer Konfiguration ist, wird als symbolische Turn-Id (vgl.: Hinweis in Kapitel 1.2) der Name "<content_file_name>_block_#i" verwendet, wobei i angibt, der wieviel-te Block aus der Datei gerade verschickt wurde.

6.2.3.5 Lokaler Timeout

Aufbau:
TIMEOUT <n>

Hierbei steht <n> für die Anzahl der Sekunden, nach denen der Timeout wirksam werden soll.

Wie bereits erwähnt, steuert ein lokaler Timeout den Übergang zum nächsten spezifizierten Sendeblock.

Lokaler und globaler Timeout arbeiten unabhängig voneinander. Wenn die Summe aller lokalen Timeouts den Wert des globalen Timeouts unterschreitet, wird der globale Timeout niemals aktiviert werden. Andernfalls *kann* der globale Timeout wirksam werden, er muß es aber nicht. Es kommt dann darauf an, *wann welche* Trigger-Nachrichten empfangen werden.

Das automatische Testmodul sieht einen Konfigurationsschritt, der durch lokalen Timeout beendet wird, immer als erfolgreich abgeschlossen an. Das ist ein wesentlicher Unterschied zum globalen Timeout. Ein Konfigurationsschritt kann nur dann durch einen lokalen Timeout beendet werden, wenn im *letzten* Sendeblock der Konfiguration ein lokaler Timeout definiert ist.

6.2.3.6 Lokale Trigger-Nachrichten

Lokale Trigger-Nachrichten steuern den Übergang zum nächsten spezifizierten Sendeblock.

Aufbau:

a) TRIGGER "Nachricht"

oder

b) TRIGGER "Nachricht_1" ... "Nachricht_n"

oder

c) PRIORITY_TRIGGER "Nachricht"

oder

d) PRIORITY_TRIGGER "Nachricht_1" ... "Nachricht_n"

Wie im globalen Fall (vgl.: 6.2.2.3) gilt: in b) und d) wird eine Konjunktion von Trigger-Nachrichten spezifiziert: Erst wenn alle angegebenen Nachrichten (mit derselben Turn-Id) während der Verarbeitung eintreffen, gilt die Trigger-Bedingung als erfüllt.

Ist eine Nachricht als globale *und* lokale Trigger-Nachricht (mit dem Schlüsselwort ‘TRIGGER’) spezifiziert, so hat der globale Fall Vorrang, d.h. es erfolgt der Übergang zum nächsten Konfigurationsschritt.

Anders liegt der Fall bei lokalen Trigger-Nachrichten, die mit dem Schlüsselwort ‘PRIORITY_TRIGGER’ spezifiziert sind: hier hat die lokale Nachricht Vorrang vor der globalen.

Wenn im letzten Sendeblock einer Konfiguration lokale Trigger-Bedingungen definiert sind, so erfolgt bei deren Eintreffen der Übergang zum nächsten Konfigurationsschritt.

6.2.3.7 Zusätzliche Befehle

Die zusätzlichen Befehle dienen dazu, dem Benutzer die Verwendung einiger zusätzlicher Optionen zu ermöglichen.

Aufbau:

ADDITIONAL <ORDER>

Momentan als <ORDER> erlaubt:

TRL_PATH <directory_name>:

Benutze zur Sortierung der Dateien eines Dialogs (vgl.: Kapitel 6.2.3.2) und zur automatischen Erkennung englischer Sprachdaten (siehe unten) die Dateien im Verzeichnis <directory_name>.

Idee: Einbeziehung von Informationen aus Transliterationsdateien in die automatische Verarbeitung. Wesentliche Voraussetzung hierfür ist, daß die Indizes der Transliterationsdateien den Namen der zugehörigen Phondat-Dateien entsprechen müssen!

AUTO_INCREMENT:

Bevor eine Dateneinheit dieses Sendeblocks verschickt wird, erfolgt eine Erhöhung der Turn-Id um 1.

Idee: Obwohl ein Konfigurationsschritt normalerweise einem Turn entspricht und daher die Turn-Id nicht innerhalb der Sendeblöcke erhöht werden sollte, kann dieser Befehl in Ausnahmefällen doch sinnvoll sein. Wenn z.B. während eines Turns ein Klärungsdialog angestoßen wird, sollte dieser auch automatisch verarbeitet werden können (etwa durch Verschicken einer Phondat-Datei an den Ja-Nein-Erkenner). Zu diesem Zweck muß jedoch im Verbmobil-System eine neue Turn-Id vergeben werden.

6.2.4 Beispiele für Konfigurationsdateien

Beispiel 1:

Die Dialoge ‘n001k,’ ‘n002k’ und ‘n009k’ sollen das Verbmobil-System von der Erkennung bis zur Morphologie durchlaufen. Als Erkenner soll das Modul von Daimler-Benz verwendet werden. Wenn nach 600 Sekunden kein Ergebnis vorliegt, ist mit dem nächsten Schritt fortzufahren.
Die Einträge in der privaten Systemkonfigurationsdatei des Verbmobil Prototyps sind so vorzunehmen, daß (neben ‘automatic_test_module’, ‘gui’ und ‘vim’) die Module ‘recog_ger_db’, ‘prosody’ und ‘morphology’ gestartet werden. Das zusätzliche Starten des Karlsruher Erkenners hätte keinen Einfluß auf den Testverlauf, wohl aber das eines der ‘synsem’-Module (eine zusätzliche linguistische Analyse des Satzes wäre die Folge)!
Zu beachten ist, daß die Sprachdaten nicht nur an den Erkenner, sondern auch an die Prosodie geschickt werden müssen.

```
CONFIGURATION_BEGIN

    SYSTEM_CONFIGURATION "recogleft recog_ger_db"
    TIMEOUT 600
    TRIGGER "tbm turnend"
      # Die nächste Trigger-Nachricht wird verschickt, wenn das System
      # nach dem Start oder nach einem Restart einsatzbereit ist.
    TRIGGER "tbm all_ready"
    TRIGGER "morphology inactive"

      # Zunächst der Sendeblock für den Erkenner
    SENDING_BLOCK_BEGIN TYPE FILES
    PARTICIPANTS recording recog_ger_db recording_recog_ger_db_raw
    SEND_FILES /cdRom/n00[129]k
    SENDING_BLOCK_END

     #  Die Sprachdaten müssen auch an die Prosodie
     #  geschickt werden
    SENDING_BLOCK_BEGIN TYPE FILES
    PARTICIPANTS recording prosody recording_prosody_raw
    SEND_FILES /cdRom/n00[129]k
    SENDING_BLOCK_END
```

CONFIGURATION_END

Beispiel 2:

Für die Dialoge ‘n001k,’ ‘n002k’ und ‘n009k’ liegen Worthypothesengraphen von einem Erkenner-Modul vor. Sie sollen prosodisch annotiert werden. Zu beachten ist, daß die Prosodie neben den Wörtergittern auch die Sprachdaten benötigt.
Voraussetzung für einen ordnungsgemäßen Testverlauf ist, daß entweder die Worthypothesengraphen dieselben Namen haben wie die zugehörigen Sprachdaten (falls die Sortierung aus der entsprechenden Transliterationsdatei verwendet werden soll), oder sie zumindest lexikographisch gleich sortiert werden (‘TRL_PATH’ sollte in diesem Fall auf ein nichtexistierendes Verzeichnis zeigen).

```
CONFIGURATION_BEGIN

    SYSTEM_CONFIGURATION "recogleft recog_ger_uka"
    TIMEOUT 500
    TRIGGER "tbm turnend"
    TRIGGER "tbm all_ready"
      # Nächster Konfigurationsschritt, wenn sich die Prosodie zum zweiten
      # mal ‘inactive’ meldet
    TRIGGER "prosody inactive"

      # Zunächst die Sprachdaten. Nach deren Abarbeitung meldet sich
      # die Prosodie ‘inactive’ und das Wörtergitter kann geschickt werden.
    SENDING_BLOCK_BEGIN TYPE FILES
        # Durch Verwendung einer lokalen Trigger-Nachricht mit Priorität
        # wird erreicht, daß beim Eintreffen der Nachricht "prosody inactive"
        # zum nächsten Sendeblock übergegangen wird, obwohl diese
        # auch als globale Trigger-Nachricht spezifiziert ist.
        PRIORITY_TRIGGER "prosody inactive"
        PARTICIPANTS recording prosody recording_prosody_raw
        SEND_FILES /cdRom/n00[129]k
    SENDING_BLOCK_END

     #  Sendeblock zum Verschicken der Wörtergitter
    SENDING_BLOCK_BEGIN TYPE FILES
        PARTICIPANTS recog_ger_uka prosody BASE
        SEND_FILES /wordLattices/n00[129]k
    SENDING_BLOCK_END

CONFIGURATION_END
```

Beispiel 3:

Es stehen Wörtergitter der Dialoge ‘n001k’, ‘n002k’ und ‘n009k’ zur Verfügung, die linguistisch verarbeitet werden sollen. Die Wörtergitter liegen im Verzeichnis ‘/wordLattices’ unter denselben Namen wie auf der Verbmobil-CD-Rom. Die zugehörigen Transliterationen wurden mit den Phondat-Namen als Indizes versehen und stehen als trl/phondatCorrelated/ {n001k,n002k,n009k} zur Verfügung. Die Abarbeitungsreihenfolge der einzelnen Turns soll dem natürlichen Gesprächsverlauf der Dialoge entsprechen. Es werden, neben dem Modul ‘selection’, alle Module auf dem Pfad der tiefen Analyse in den Testlauf einbezogen.

```
CONFIGURATION_BEGIN

    SYSTEM_CONFIGURATION "synsem synsem_s3" "recogleft recog_ger_uka"
    TIMEOUT 900
    TRIGGER "tbm turnend"
    TRIGGER "tbm all_ready"

      # Sendeblock zum Verschicken der Wörtergitter
    SENDING_BLOCK_BEGIN TYPE FILES
        PARTICIPANTS morphology synsem_s3 BASE
        SEND_FILES /wordLattices/n00[129]k
        ADDITIONAL TRL_PATH /project/verbmobil/trl/phondatCorrelated
    SENDING_BLOCK_END

CONFIGURATION_END
```

6.2.5 Übersicht: Befehle in Konfigurationsdateien

```
CONFIGURATION_BEGIN
    SYSTEM_CONFIGURATION   "<system configuration msg>"+
    TIMEOUT                <n>
    TRIGGER                "<msg>"+  (* Trigger Nachricht oder Konjunktion
                                        von Trigger Nachrichten *)
    ADDITIONAL             DOUBLE_PROCESSING
    ADDITIONAL             SUBDIR <directory_name>
    ADDITIONAL             STARTING_POINT <suffix>
    ADDITIONAL             CONTINUE_IF_ALL_INACTIVE <n>
    ADDITIONAL             STARTUP_TIMEOUT <n>
    ADDITIONAL             KILL_ACTIVE_MODULES <n>
    <SENDING_BLOCK>*
CONFIGURATION_END

SENDING_BLOCK_BEGIN TYPE [ FILES | STRINGS | FILE_CONTENT ]
    PARTICIPANTS       <sender> <receiver> <channel name>
    SEND_FILES         <pattern>+
          oder
    SEND_DATA          "<string>"+
          oder
    SEND_CONTENT       <filename>
    TIMEOUT            <n>
    TRIGGER            "<msg>"+     (*  Trigger Nachricht oder Konjunktion
                                        von Trigger Nachrichten *)
    PRIORITY_TRIGGER   "<msg>"+     (*  Trigger Nachricht oder Konjunktion
                                        von Trigger Nachrichten *)
    ADDITIONAL         TRL_PATH <directory_name>
    ADDITIONAL         AUTO_INCREMENT
SENDING_BLOCK_END
```

6.3 Aktivieren des automatischen Testmoduls

Das automatische Testmodul ist, wie alle Module im Verbmobil-Prototyp einfach durch einen Eintrag in der privaten Systemkonfigurationsdatei an- und abschaltbar. Es genügt eine Spezifikation der Art:

```
MODULE automatic_test_module
      USED: yes
      STARTUP: "$VM_HOME/bin/VmAutoTestModule <options>"
END
```

Folgende Kommandozeilenoptionen (<options>) sind erlaubt:

-f <filename>	wird im nächsten Kapitel beschrieben
-T <pathname>	setzt den default Wert für den ‘TRL_PATH’
-t <trace level>	steuert die Anzahl der Kontrollausgaben. Erlaubte Werte: zwischen 0 (keine Ausgaben) und 99999.

Während der automatischen Verarbeitung ist es verboten, Spracheingaben zu machen. Es ist *nicht* verboten, Systemeinstellungen zu verändern, aber man sollte bedenken, daß etwa eine Änderung des verwendeten ‘Erkenner-Moduls’ einen reibungslosen Ablauf des Tests verhindern kann. Die Umstellung zwischen den verwendeten Erkennern sollte innerhalb der Konfiguration vorgenommen werden.

Nach Abarbeitung aller Konfigurationsdateien kehrt das System in den ‘normalen’ Ausführungsmodus zurück. Jetzt kann das System ebenso bedient werden wie ohne automatischen Tester. Insbesondere sind Spracheingaben wieder erlaubt.

6.3.1 Start durch Kommandozeilenoption

Ein Testlauf kann durch Verwendung einer Kommandozeilenoption gestartet werden. Zu diesem Zweck muß ein Eintrag in die private Systemkonfigurationsdatei folgendermaßen aussehen:

```
MODULE automatic_test_module
      USED: yes
      STARTUP: "$VM_HOME/bin/VmAutoTestModule [ -f <name> ]*"
END
```

Hierbei ist <name> der Name einer Konfigurationsdatei, die den syntaktischen Vorgaben zur Spezifikation einer Konfigurationsdatei für das automatische Testmodul genügt.

Es ist zu beachten, daß das automatische Testmodul mit dem Versenden der Dateneinheiten erst beginnt, wenn eine globale Fortführungsbedingung eintritt (Timeout oder Trigger-Nachricht). Dies gilt nur für die *erste* spezifizierte Konfiguration. Sobald deren Daten vollständig verarbeitet wurden, wird sofort mit der nächsten Konfiguration fortgefahren (falls es eine nächste Konfiguration gibt).

Grund: Mit der automatischen Verarbeitung sollte erst begonnen werden, wenn das System einsatzbereit ist. Zu diesem Zweck spezifiziert man in der ersten verwendeten Konfiguration einen hinreichend großen globalen Timeout (bzw. Startup-Timeout) oder die globale Trigger-Nachricht "tbm all_ready".

6.3.2 Start über Visualisierungsmanager (VIM)

Die zweite Möglichkeit, einen Testlauf zu starten, ist über das Send Control Window (SCW) des Visualisierungsmanagers (Maus-Mittelklick auf einem beliebigen Pfeil an einer Systemschnittstelle).

Dort ist die Auswahl:

From: vim
To: automatic_test_module

zu treffen. Dann können dem automatischen Testmodul einige bestimmte Befehle über das ‘Message Window’ gesandt werden. Folgende Befehle sind, je nach Status des automatischen Testmoduls, erlaubt:

1) Im ‘running mode’: Das automatische Testmodul ist aktiv, d.h. es wird momentan eine Konfiguration verarbeitet:

   ‘interrupt_processing’ : die automatische Verarbeitung wird sofort unterbrochen, Übergang zum ‘interrupt mode’.

2) Im ‘interrupt mode’: Das automatische Testmodul wurde vom Benutzer während der Verarbeitung unterbrochen:

   ‘resume’ : Die automatische Verarbeitung wird dort fortgesetzt, wo sie unterbrochen wurde (die letzte bearbeitete Dateneinheit wird nochmals verschickt). Übergang zum ‘running mode’.

3) Im ‘idle mode’: Das automatische Testmodul ist nicht beschäftigt, d.h. es wird momentan keine Konfiguration verarbeitet und der Zustand ist nicht ‘interrupt mode’:

   ‘start’: Die Verarbeitung einer (noch gespeicherten oder zuvor mit ‘load_configuration’ spezifizierten) Konfiguration wird sofort aufgenommen. Übergang zum ‘running mode’.

   ‘start_delayed’: Die Verarbeitung einer (noch gespeicherten oder zuvor mit ‘load_configuration’ zu spezifizierenden) Konfiguration wird aufgenommen, sobald eine globale Fortsetzungsbedingung eintritt. Übergang zum ‘running mode’.

   ‘install_configuration <filename>’: Entspricht der Befehlsfolge
   ‘load_configuration <filename>’ ‘start’

4) Im ‘interrupt mode’ *und* ‘idle mode’ sind folgende Befehle erlaubt:

   ‘load_configuration <filename>’: Die mit <filename> bezeichnete Datei wird *hinten* in die Liste aller zu verarbeitenden Konfigurationsdateien eingefügt. Der augenblickliche Zustand (‘mode’) ändert sich nicht.

   ‘delete_configuration’: Die aktuelle Konfiguration wird vernichtet. Momentan gibt es nur im ‘interrupt mode’ eine aktuelle Konfiguration: eben die unterbrochene. Übergang zum ‘idle mode’.

‘delete_all’: Alle gespeicherten Konfigurationen und die eventuell existierende aktuelle Konfiguration werden vernichtet. Übergang zum ‘idle mode’.

6.3.3 Abbruch der automatischen Verarbeitung

Ein automatischer Testlauf kann durch Anklicken des Buttons ‘Abbruch’ auf der Oberfläche unterbrochen werden. Der Effekt ist derselbe, wie wenn über das Send Control Window der Befehl ‘interrupt_processing’ verschickt wird (siehe Kapitel 6.3.2).

6.3.4 Ausgabedateien des automatischen Testmoduls

Folgende Dateien werden im Verzeichnis ‘$VM_TMP’ angelegt:

autotestData.log: Sobald eine Konfiguration installiert ist, wird ihre genaue Beschreibung hier abgelegt: Timeout, Trigger-Nachrichten und spezifizierte Sendeblöcke.

autotestError.log: Alle während der Verarbeitung festgestellten Unregelmäßigkeiten (Fehler und Warnungen) werden in dieser Datei dokumentiert.

autotestProcessing.log: Die wesentlichen Vorkommnisse während der automatischen Verarbeitung werden (mit Zeitstempel) in dieser Datei protokolliert: Spezifikation der verschickten Daten, Timeout, Empfang von Trigger-Nachrichten, Modulabstürze und jeder Übergang zu einem neuen Konfigurationsschritt.

Falls ein automatischer Testlauf nicht das tut, was von ihm erwartet wird, empfiehlt es sich, zunächst in diesen Dateien auf Fehlersuche zu gehen:

- autotestData.log gibt z.B. darüber Auskunft, ob die zu verarbeitenden Dateien richtig spezifiziert waren (Schreibfehler: ‘m001k’ statt ‘n001k’ ...).
- autotestError.log dokumentiert z.B. Fehler beim Lesen der Konfigurationsdatei (falscher Dateiname, fehlerhafter Eintrag ...).
- autotestProcessing.log kann hilfreich sein, semantisch falsch spezifizierte Konfigurationen zu entdecken (falsche Trigger-Nachricht ...).

Zu beachten ist, daß die Konfigurationen erst bei ihrer Installation in die Datei ‘autotestData.log’ geschrieben werden. Wenn eine Konfigurationsdatei mehrere Konfigurationen enthält, werden die Daten der zweiten Konfiguration erst dann sichtbar, wenn die erste Konfiguration vollständig abgearbeitet ist.

Kapitel 7

Weitere Entwicklung

Wegen des äußerst dynamischen Charakters des Projektes Verbmobil entwickelt sich das System ständig weiter. Weitere Versionen dieses Handbuches werden folgen. Trotzdem haben sich die grundsätzlichen Architekturkonzepte seit mehr als zwei Jahren bewährt und werden auch in Zukunft erhalten bleiben. Dazu gehören:

- die Durchgängigkeit der fachlichen Architektur bis hin zur Realisierung als kommunizierende Prozesse,
- die Beschränkung aller Schnittstellenformate auf vier Datentypen und deren adäquate grafische Visualisierung,
- die Unterstützung der Modulentwicklung durch komfortable Testwerkzeuge (split channels, automatisches Testmodul).

Verbmobil ist ein konfigurierbares und skalierbares System. In dem Integrationsrahmen Testbed können unterschiedliche Systemausprägungen eingebettet werden. Wegen des klaren Konzepts der kommunizierenden Module kann das Testbed auch für andere Anwendungen als Integrationsrahmen dienen.

Kapitel 8

Literatur

[Amtrup 94a] Jan Willers Amtrup: „ICE-Intarc Communication Environment: Design and Specification“. Verbmobil Memo 48, Universität Hamburg, September 1994.

[Amtrup 94b] Jan W. Amtrup: „ICE: INTARC Communication Environment. User‘s Guide and Reference Manual“. Verbmobil Technisches Dokument 14, Universität Hamburg, November 1994.

[Auerswald & Amtrup 94] Marko Auerswald, Jan Amtrup: „Kommunikationsarchitektur für den Demonstrator“. Verbmobil Technisches Dokument 15, DFKI Kaiserslautern, Universität Hamburg, Dezember 1994.

[Benra 95a] Joerg Benra: „Standards der Software-Integration im Verbmobil Teilprojekt 16“. Verbmobil Memo 91, DFKI Kaiserslautern, August 1995.

[Benra 95b] Joerg Benra: „Kommunikation im Verbmobil Forschungsprototyp“. Verbmobil Memo 92, DFKI Kaiserslautern, Oktober 1995.

[Bub & Schwinn 96] Thomas Bub, Johannes Schwinn: „Verbmobil: The Evolution of a Complex Large Speech-to-Speech Translation System“. In: Proceedings of the Fourth International Conference on Spoken Language Processing (ICSLP 96), Philadelphia, PA, Oktober 1996.

[Dorna 96] Michael Dorna: „The ADT Package for the Verbmobil Interface Term“. Verbmobil Report 104, Universität Stuttgart, April 1996.

[Geist et. al.] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam: „PVM 3 Users’s Guide and Reference Manual“. Oak Ridge National Laboratory.

[Görz & Menzel 94] Günther Görz, Wolfgang Menzel: „Diskussionsbeitrag zur Systemarchitektur von Verbmobil“. Verbmobil Memo 45, Universität Erlangen-Nürnberg, August 94.

[Kesseler 94] Marcus Kesseler: „Distributed Control in Verbmobil“. Verbmobil Report 24, Universität Erlangen Nürnberg, August 1994.

[Nöth & Plannerer 93] Elmar Nöth, Bernd Plannerer: „Schnittstellendefinition für den Worthypothesengraphen“. Verbmobil Memo 2, Universität Erlangen-Nürnberg, Dezember 1993.

[Preuß 96] Wiebke Preuß: „Qualitätsanforderungen an ein dolmetschendes Computersystem. Eine empirische Analyse der Erwartungen potentieller Benutzer“. Verbmobil Report 150, Universität Hamburg, August 1996.

[Turk & Geißler 95] Andreas Turk, Stefan Geißler: „Integration alternativer Komponenten für die Sprachverarbeitung im Verbmobil-Demonstrator“. Verbmobil-Report 67, DFKI Kaiserslautern, IBM Heidelberg, April 1995.

[Wahlster & Engelkamp 92] Wolfgang Wahlster, Judith Engelkamp (Hrsg.): „Wissenschaftliche Ziele und Netzpläne für das VERBMOBIL-Projekt“. Saarbrücken, April 1992.