Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A. Helbling, T.Murer

Spezifizieren und Generieren von integrierten Umgebungen mit GIPSY

TIK-Report
Nr. 12, Mai 1995

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

A. Helbling, T. Murer
Spezifizieren und Generieren von integrierten Umgebungen mit GIPSY
Mai 1995
Version 1
TIK-Report Nr. 12

Computer Engineering and Networks Laboratory,
Swiss Federal Institute of Technology (ETH) Zurich

Institut für Technische Informatik und Kommunikationsnetze,
Eidgenössische Technische Hochschule Zürich

Gloriastrasse 35, ETH-Zentrum, CH-8092 Zürich, Switzerland

---

1 Inhaltsverzeichnis

1 Inhaltsverzeichnis I

2 Einleitung 1

3 Das Werkzeug GIPSY 3

3.1 Scanner-Erzeugung 3
3.1.1 Erzeugter Scanner 3
3.2 Parser-Erzeugung 5
3.2.1 Grammatik-Tests 5
3.2.2 Erzeugter Parser 6
3.3 Hauptmodul 6
3.4 Das GIPSYEnv-Panel 6
3.4.1 Umgebungs-Liste 7
3.4.2 Dokumenten-Liste 8
3.4.3 Sichten- und Werkzeug-Liste 9
3.4.4 GIPSY-Kern 9

4 Die GIPSY-Umgebung 11

4.1 Textdarstellung 13
4.1.1 Menu-Leiste 13
4.1.2 Fehlerelemente 14
4.1.3 Attribut-Gadgets und Attribut-Inspektor 15

5 Sichten und Werkzeuge 16

5.1 Standard-Sichten 16
5.1.1 Token-Liste 16
5.1.2 Struktur-Baum 17
5.2 Programmierung eigener Views 17

5.2.1 Ein Gerüst für eigene Views 18

6 Anhang 20

6.1 Literaturverzeichnis 20

2 Einleitung

Mit dem Werkzeug GIPSY können ausgehend von Spezifikations-Dokumenten (Erweiterbare Attributierte Grammatiken) *integrierte Umgebungen* beschrieben und generiert werden. Eine integrierte Umgebung beinhaltet eine Klasse von Text-Dokumenten, denen allen die in der Umgebungs-Spezifikation definierte Grammatik zugrunde liegt. Mit den in den attributierten Grammatiken aufgeführten Attributierungsprozessen können die Dokumenteninhalte verarbeitet und transformiert werden. Zudem kann jedes Text-Dokument neben der Textdarstellung mit andern Sichten betrachtet werden und besitzt Werkzeuge zum Verändern des Inhaltes.

Eine Grammatik-Hierarchie, die von GIPSY zur Erzeugung einer Umgebung benötigt wird, enthält eine Scanner- und eine Parser-Spezifikation. Die Parser-Spezifikation kann aus mehreren voneinander erweiterten Attributierten Grammatiken bestehen. Aus diesen in der Spezifikationssprache GIPSY/L [Hel95] verfassten Dokumenten erzeugt GIPSY Oberon-Module, die zusammen mit dem umgebungsunabhängigen GIPSY-Kern auf dem Oberon System 3 (Gadgets) zu einer ausführbaren Umgebung übersetzt werden können.

Der Name GIPSY steht sowohl für ein Werkzeug (Compiler-Generator) als auch für eine integrierte Umgebung. Die GIPSY-Umgebung besteht aus Dokumenten, welche die Grammatik von GIPSY/L erfüllen und auf die das Werkzeug GIPSY angewendet werden kann.

Im Gegensatz zu andern Compiler-Werkzeugen (z.B. Coco/R [Möss90]) erzeugt GIPSY einen Syntaxbaum, der auch nach der Compilation noch erhalten bleibt, und benutzt nicht die Technik von rekursiv absteigenden Parse-Prozeduren. Auf dem Syntaxbaum können so jederzeit Attributierungsprozesse durchgeführt werden, ohne dass jedesmal vorher ein Parse-Durchgang nötig ist. Die Traversierung des Baumes ist nicht auf einen Durchgang beschränkt, sondern es stehen Operationen zur Verfügung, mit denen beliebig auf den Knoten des Syntaxbaumes navigiert werden kann.

Ein weiterer Vorteil von GIPSY liegt darin, dass die erzeugten Datenobjekte bereits in eine integrierte Umgebung (GIPSY-Umgebung und GIPSY-Kern) eingebettet sind, ohne dass noch Module ergänzt oder zusätzlich programmiert werden müssen.

Die Wahl eines objektorientierten Ansatzes mit der Programmiersprache und dem Betriebsystem Oberon bei der Implementierung von GIPSY färbt auch auf

die Attributierten Grammatiken ab, die einem Oberon-Modul sehr nahe kommen. Die Nonterminalsymbole entsprechen einer Klasse mit den Attributen als Datenfeldern und den Attributierungsprozeduren als Methoden. Die Technik der Erweiterung wurde auf die Grammatik-Ebene ausgeweitet. So kann die Spezifikation einer integrierten Umgebung aus mehreren voneinander abhängigen Attributierten Grammatiken bestehen. Dies erlaubt einerseits eine Aufteilung der Aufgaben einer komplexen Umgebung im Sinn einer Modularisierung. Andererseits entstehen dadurch erst integrierte Umgebungen. Denn erst durch die Möglichkeit von Erweiterungen können mehrere Werkzeuge auf einer gemeinsamen Basis-Grammatik aufbauen und miteinander integriert werden.

3 Das Werkzeug GIPSY

Das Werkzeug GIPSY überprüft Scanner-Spezifikationen und Attributierte Grammatiken (Parser-Spezifikation) auf Korrektheit und erzeugt anschliessend daraus Oberon-Module für Scanner und Parser, die zusammen mit dem umgebungsunabhängigen GIPSY-Kern eine ausführbare Umgebung ergeben. Zur Bedienung von GIPSY kann das GIPSYEnv-Panel verwendet werden.
Die Eingabedateien, die von GIPSY erwartet werden, müssen in der Spezifikationssprache GIPSY/L [Hel95] verfasst sein. GIPSY erzeugt aus der Scanner-Beschreibung ein Scanner-Modul, aus jeder Attributierten Grammatik ein Parser-Modul, und ein Hauptmodul, das die Umgebung zusammenfügt (vgl. Abb. 1).

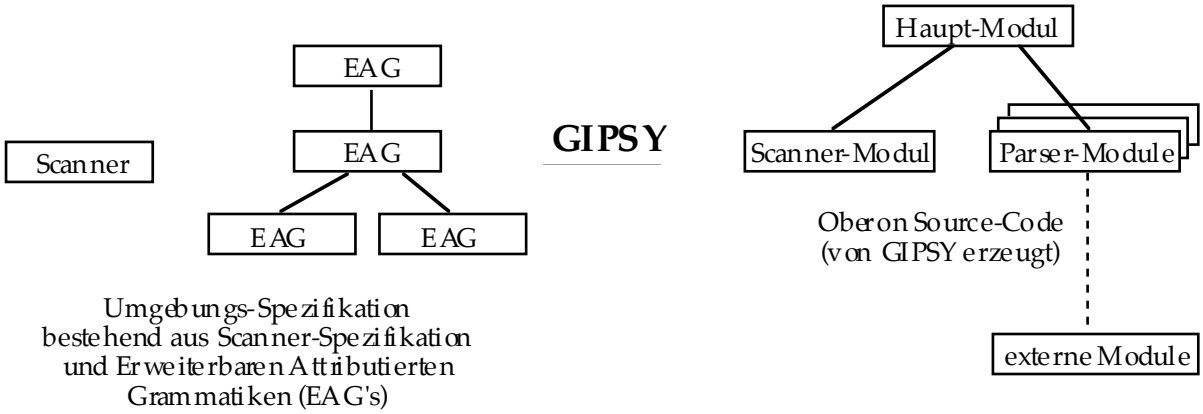

Abb. 1: Entwicklungsablauf beim Generieren einer Umgebung mit GIPSY

Die von GIPSY erzeugten Module

3.1 Scanner-Erzeugung

3.1.1 Erzeugter Scanner

Der GIPSY-Kern bietet einen abstrakten Scanner an, der alle notwendigen Operationen eines Scanners bereits vorsieht und der jedem von GIPSY erzeugten Scanner als Grundlage dient.

Allgemeine Scanner-Schnittstelle

```
DEFINITION Scanners;

   IMPORT GIPSYSets,Objects,SymbolTable,Texts;

   CONST
      EOF = 00X; EOL = 0DX; TAB = 09X; COM = 01X;

   TYPE
```

```
    Scanner = POINTER TO ScannerDesc;
    ScannerDesc = RECORD
        R: Texts.Reader;
        ch-: CHAR;
        ordCh-: INTEGER;
        symPos,symLen-: LONGINT;
        sym-: INTEGER;
        Init-,Get-: ScannerProc
    END;
    ScannerProc = PROCEDURE (S: Scanner);

  PROCEDURE AddTerminal(S: Scanner; ident: ARRAY OF CHAR; sym: INTEGER; isKey: BOOLEAN);
  PROCEDURE AssignSource(S: Scanner; source: Texts.Text);
  PROCEDURE EndOfSource(S: Scanner): BOOLEAN;
  PROCEDURE Init(S: Scanner; Init, Get: ScannerProc);
  PROCEDURE Mark(pos: LONGINT; err: INTEGER);
  PROCEDURE NextChar(S: Scanner);
  PROCEDURE SetPos(S: Scanner; pos: LONGINT);
  PROCEDURE SetSymbol(S: Scanner; sym: INTEGER);
  PROCEDURE SkipComments(S: Scanner);

END Scanners.
```

Spezieller Scanner

Aus der Scanner-Spezifikation erzeugt GIPSY in einem Modul die zwei nicht exportierten Prozeduren Init und Get. Init dient der Initialisierung des Scanners und Get implementiert den deterministischen endlichen Automaten (DFA), der den Eingabestrom in Tokens umwandelt. Wenn nun diese zwei Prozeduren an einen allgemeinen Scanner gebunden werden, entsteht ein konkreter Scanner, der das in der Spezifikation festgelegte Verhalten unterstützt.

Beispiel: Scanner-Definition für die Umgebung *Calc*

```
DEFINITION CalcScanner;

  IMPORT Scanners;

  PROCEDURE New(): Scanners.Scanner;

END CalcScanner.
```

Der Scanner wird in einer New-Prozedur aus einem von GIPSY erzeugten Scanner-Modul angelegt und initialisiert.

```
S:=CalcScanner.New();
```

Mit den Datenfeldern und Prozedurvariablen Init und Get kann der Scanner verwendet und gesteuert werden.

```
S.Init(S)    (* Initialisierung des Scanners *)
S.Get(S)     (* nächstes Symbol scannen *)
```

3.2 Parser-Erzeugung

Bei der Analyse von kontextfreien Grammatiken muss die LL(1)-Bedingung (siehe 3.2.1) erfüllt sein, damit aus der Sprachspezifikation ein Parser erzeugt werden kann.

3.2.1 Grammatik-Tests

Vor der Parser-Generierung testet GIPSY jede Erweiterbare Attributierte Grammatik auf folgende sechs aufeinander aufbauenden Eigenschaften:

1. **Korrekte Syntax**
   Als Grundlage für weitere Grammatiktests muss die Syntax von GIPSY/L eingehalten werden.
2. **Vollständigkeit**
   Zu allen auf der rechten Seite einer Produktion verwendeten Nonterminalsymbolen muss eine Produktion definiert sein.
3. **Erreichbarkeit**
   Jedes in einer Produktion definierte Nonterminalsymbol muss vom Startnonterminalsymbol der Grammatik aus erreichbar sein.
4. **Terminalisierbarkeit**
   Alle Produktionen müssen zu Terminalsymbolen ableitbar sein.
5. **Linksrekursion**
   Linksrekursive Deklarationen (z.B. A ::= A 'a'.) von Nonterminalsymbolen werden entdeckt und werden entdeckt.
6. **LL(1)-Bedingung**
   Für die Definition der LL(1)-Bedingung werden die Begriffe der terminalen Anfänger- und Nachfolger-Menge benötigt:
   Die *Menge der terminalen Anfänger* einer Symbolfolge (First-Menge) ist die Menge aller Terminalsymbole, mit denen die Symbolfolge oder eine daraus abgeleitete Symbolfolge beginnen kann.
   Die *Menge der terminalen Nachfolger* einer Symbolfolge (Follow-Menge) ist die Menge aller Terminalsymbole, die auf die Symbolfolge folgen können.
   Eine Grammatik erfüllt die LL(1)-Bedingung, wenn
   a) für jede ihrer Produktionen die Mengen der terminalen Anfänger aller Alternativen paarweise disjunkt sind und
   b) für Nonterminalsymbole, die sich in eine leere Folge ableiten lassen, alle terminalen Nachfolger des Nonterminalsymbols von den terminalen Anfängern jeder Alternative disjunkt sind.

3.2.2 Erzeugter Parser

Falls die LL(1)-Bedingung gewährleistet ist, kann aus der kontextfreien Grammatik ein Parser erzeugt werden. Die Konstruktion des Parsers beruht auf den Parse-Prozeduren aus dem Modul *Structures* im GIPSY-Kern und ist in [Mar94] beschrieben.

3.3 Hauptmodul

Zusammen mit dem Scanner und Parser wird von GIPSY auch ein Hauptmodul generiert, das die erzeugten Module zu einer ausführbaren Umgebung verbindet und als Schnittstelle zum Oberon System (Gadgets) dient. Der Modulname setzt sich aus dem Grammatiknamen und der Erweiterung 'Env' zusammen.

```
DEFINITION CalcEnv;

   IMPORT Objects,Structures;

   PROCEDURE InitText (T: Structures.StructureText);
   PROCEDURE NewText;
   PROCEDURE TextHandle (T: Objects.Object; VAR M: Objects.ObjMsg);

END CalcEnv.
```

Im Hauptmodul wird für jede Umgebung ein globaler Scanner *gScanner* angelegt. Da nie mehrere Scan-Vorgänge gleichzeitig stattfinden können, ist dieser für alle Umgebungen des gleichen Typs wiederverwendbar, sofern er vor jedem Gebrauch mit *S.Init(S)* wieder zurückgesetzt wird.
Es wird nicht für jede Umgebung ein neuer Typ als Erweiterung von *Structures.StructureText* definiert. Ein Structure-Text besitzt schon alle notwendige Information über einen Text: Textinhalt (Zeichenfolge), Tokenliste und Syntaxbaum. Eine im Hauptmodul implementierte Prozedur zum Starten der Syntax-Überprüfung und Aufbaus des Syntaxbaums (CheckSyntax) wird bei der Erzeugung eines neuen Textes (InitText) als Prozedurvariable installiert.
Ein Text ist ein abstraktes Gadgets, das voll ins Oberon System integriert ist. Der Text-Handler und die Prozedur InitText sind exportiert, um spätere Erweiterungen zu ermöglichen.

3.4 Das GIPSYEnv-Panel

Zur Entwicklung und Verwaltung der von GIPSY erzeugten Umgebungen und des GIPSY-Kerns kann das GIPSYEnv-Panel (Abb. 2) benutzt werden.
Bevor das GIPSYEnv-Panel überhaupt benutzt werden kann, muss das Modul 'GIPSYPanel.Mod', in dem die Funktionalität des Panels enthalten ist, übersetzt werden.

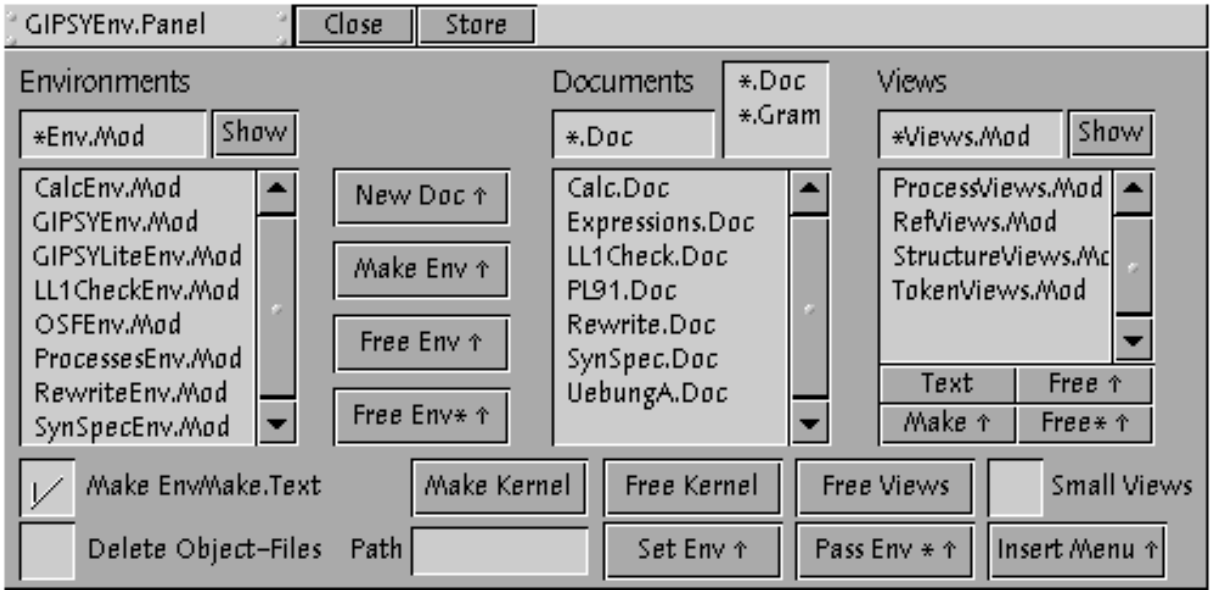

Abb. 2: Panel zum Entwickeln und Verwalten von Umgebungen mit GIPSY

Hauptbestandteil des Panels sind drei Listen (Environments, Documents und Views) und eine Reihe von Knöpfen zur Handhabung des GIPSY-Kerns.

3.4.1 Umgebungs-Liste

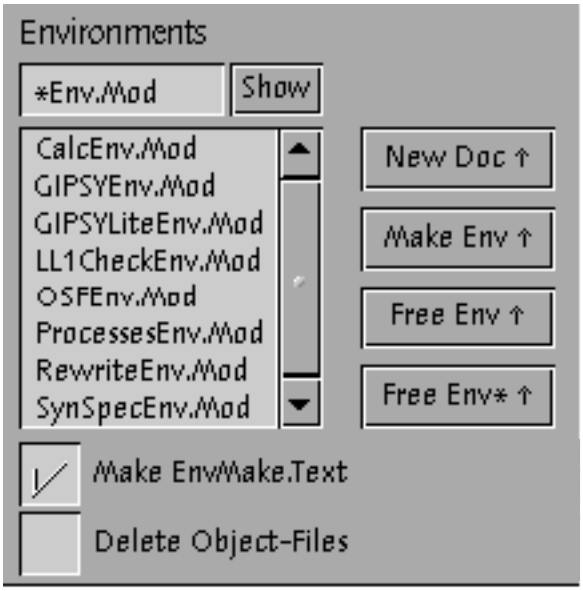

Abb. 3: Umgebungsliste des GIPSYEnv-Panels

In der Umgebungs-Liste **Environments** (Abb. 3) werden alle erzeugten Umgebungen aufgeführt, die im Directory nach Drücken des Knopfes **Show** gefunden werden können.
Durch Selektion einer Umgebung aus der Liste und Drücken von **Make Env** werden die von GIPSY erzeugten Module vom Oberon-Compiler übersetzt. Falls

zur Umgebung noch externe Module benötigt werden, müssen diese separat compiliert werden. Wenn diese Module hingegen in einem Text mit dem Namen **MyEnvMake.Text** (*MyEnv* steht dabei für den Namen der entsprechenden Umgebung) aufgelistet und abgespeichert sind, werden sie mit der Operation Make Env ebenfalls übersetzt, sofern dies in der Checkbox **Make EnvMake.Text** angegeben ist.
Falls beim Übersetzen auch die Option **Delete Object-Files** eingestellt ist, werden zuerst alle Object-Files gelöscht und dann vom Oberon-Compiler neu generiert. Dies kann von Nutzen sein, wenn in einer verteilten Anwendung Object-Files von verschiedenen Plattformen miteinander benutzt werden.
Ein neues Dokument zu einer Umgebung wird geöffnet, indem die gewünschte Umgebung selektiert und **New Doc** gedrückt wird, oder durch direktes Drücken auf den entsprechenden Listeneintrag.
Mit **Free Env** werden die Module der selektierten Umgebung freigegeben. **Free Env** * versucht, neben der Umgebung auch noch den ganzen GIPSY-Kern zu entladen.

3.4.2 Dokumenten-Liste

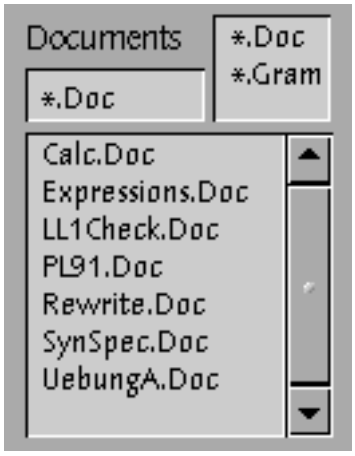

Abb. 4: Dokumenten-Liste des GIPSYEnv-Panels

In der Liste **Documents** (Abb. 4) können die zu den Umgebungen geschriebenen Dokumente gesammelt werden. Die Suchkriterien sind frei wählbar und können in einer eigenen Liste festgehalten werden. GIPSY-Dokumente sind nicht an eine bestimmte Filename-Extension gebunden.
Ein Dokument aus dieser Liste kann geöffnet werden, wenn sein Eintrag angeklickt wird.

3.4.3 Sichten- und Werkzeug-Liste

Abb. 5: Sichten- und Werkzeug-Liste des GIPSYEnv-Panels

In der Liste **Views** (Abb. 5) werden die Standard-Sichten (Token-Liste, Struktur-Baum) und die umgebungs-spezifischen Sichten (in Abb. 5: RefViews und ProcessViews) aufgelistet. Diese Sichten können mit **Make** und **Free** bzw. **Free** * übersetzt oder freigegeben werden.
Durch Drücken des entsprechenden Listeneintrages wird zu einem markierten oder selektierten Dokument die gewünschte Sicht angezeigt.
Text öffnet zu einer bestehenden, markierten oder selektierten Sicht ein Text-Dokument.

3.4.4 GIPSY-Kern

Der restliche Teil des Panels (Abb. 6) bietet allgemeine Einstellungen und Operationen, die den GIPSY-Kern betreffen, an.

Abb. 6: Operationen für den GIPSY-Kern

Allgemeine Einstellungen

Im Feld **Path** kann ein Pfadname angegeben werden, unter dem der Oberon-Compiler die zu übersetzenden Module sucht.
Mit **Small Views** kann eingestellt werden, ob die Sichten in normaler oder verkleinerter (für kleinere Bildschirme) Grösse dargestellt werden.

GIPSY-Kern

Mit dem Knopf **Make Kernel** kann der umgebungsunabhängige GIPSY-Kern übersetzt werden. Dies ist im allgemeinen nur bei einer Installation von GIPSY nötig.

Free Kernel gibt die Module des ganzen GIPSY-Kerns frei, was aber nur dann erfolgreich durchgeführt werden kann, wenn im Oberon-System keine Umgebung geladen ist.

Free Views versucht, die zum GIPSY-Kern gehörenden Sichten freizugeben.

Set Env entspricht dem Oberon-Command *Gadgets.Link SelectedEnv.NewText* und übergibt dem selektierten Gadget einen Text als Modell. Der Typ der Umgebung, zu der der neue Text erzeugt wird, wird dabei in der Umgebungs-Liste ausgewählt.

Pass Env übergibt allen selektierten Gadgets einen bestehenden markierten Struktur-Text als Modell.

Insert Menu fügt ein zum Text-Modell eines selektierten Gadgets passendes Menu, mit dem die Attributierungsprozeduren ausgeführt werden können, an die Caret-Position ein.

4 Die GIPSY-Umgebung

Das Werkzeug GIPSY ist wie jede andere von ihm erzeugte Umgebung auch in die GIPSY-Umgebung (GIPSY-Kern) eingebettet, in der alle umgebungsunabhängigen Bestandteile enthalten sind.

- Scanner-Gerüst
- Parser-Prozeduren
- GIPSY-Dokumente
- Textdarstellung und Standard-Sichten
- Modul GIPSYSpecs als Schnittstelle zu den einzelnen Umgebungen.

Eine erzeugte Umgebung besteht wie aus Abb. 7 ersichtlich aus dem umgebungsunabhängigen GIPSY-Kern und den vom Werkzeug GIPSY aus der Spezifikation 'G' erzeugten Modulen.

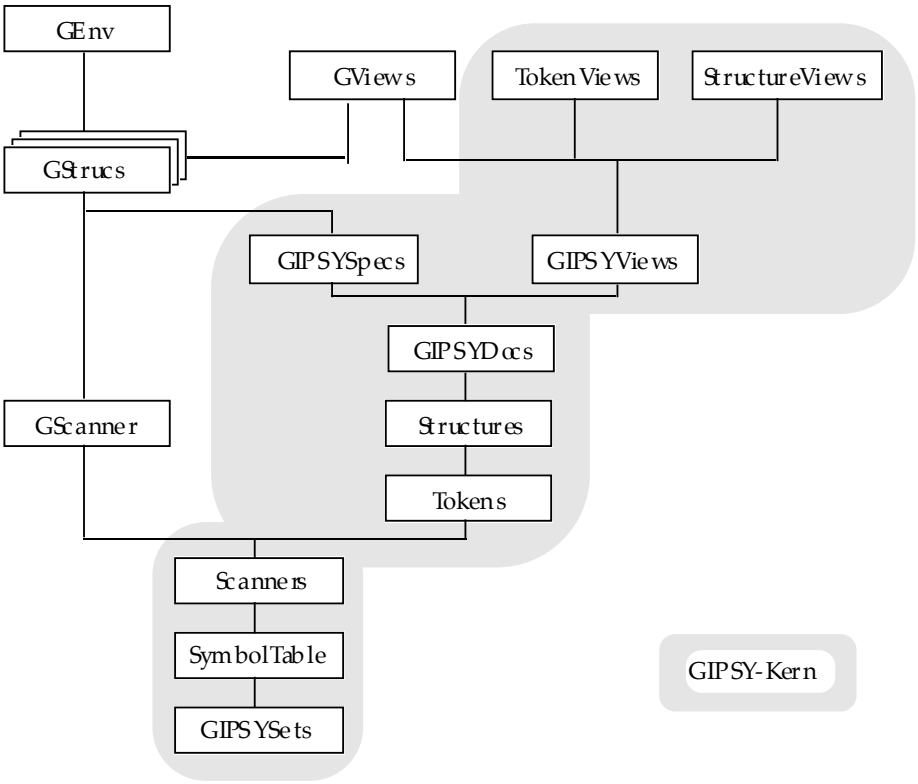

Abb. 7: GIPSY-Modulhierarchie (GIPSY-Kern und erzeugte Umgebung 'G')

Die aus Abb. 7 ersichtlichen Schnittstellen-Module zwischen dem Kern und einer erzeugten Umgebung sind in Tab. 1 aufgelistet.

Modul	Schnittstelle zu …
GIPSYSpecs	Attributierte Grammatiken
Scanners	Scanner der Umgebung
GIPSYViews	Sichten/Werkzeuge
GIPSYDocs	Textdarstellung

Tab. 1: Schnittstellen-Module des GIPSY-Kerns

Jede Hierarchie von Attributierten Grammatiken definiert eine eigenständige Umgebung und gleichzeitig auch eine Klasse von GIPSY-Dokumenten, die folgende Eigenschafen aufweisen:

- Jedes Dokument besitzt einen Oberon-Text zur Erfassung des Dokumenteninhaltes. Sobald durch einen Parse-Vorgang die Syntax des Inhaltes fehlerlos überprüft werden konnte, besitzt das Dokument eine Token-Liste und einen Struktur-Baum, die mit den entsprechenden Werkzeugen visualisiert werden können.
- Zum Verändern des Dokumentinhaltes steht ein integrierter Editor zur Verfügung.
- Das Dokument kennt alle Attributierungs-Operationen (Commands), die auf ihm ausgeführt werden können, und bietet ein Menu zur Ausführung dieser Operationen an.
- Verschiedenste Sichten, mit denen die Attribute eingesehen und verändert werden können, sind vorhanden (Token-View, Struktur-View) oder können selber programmiert und integriert werden (Abb. 8).

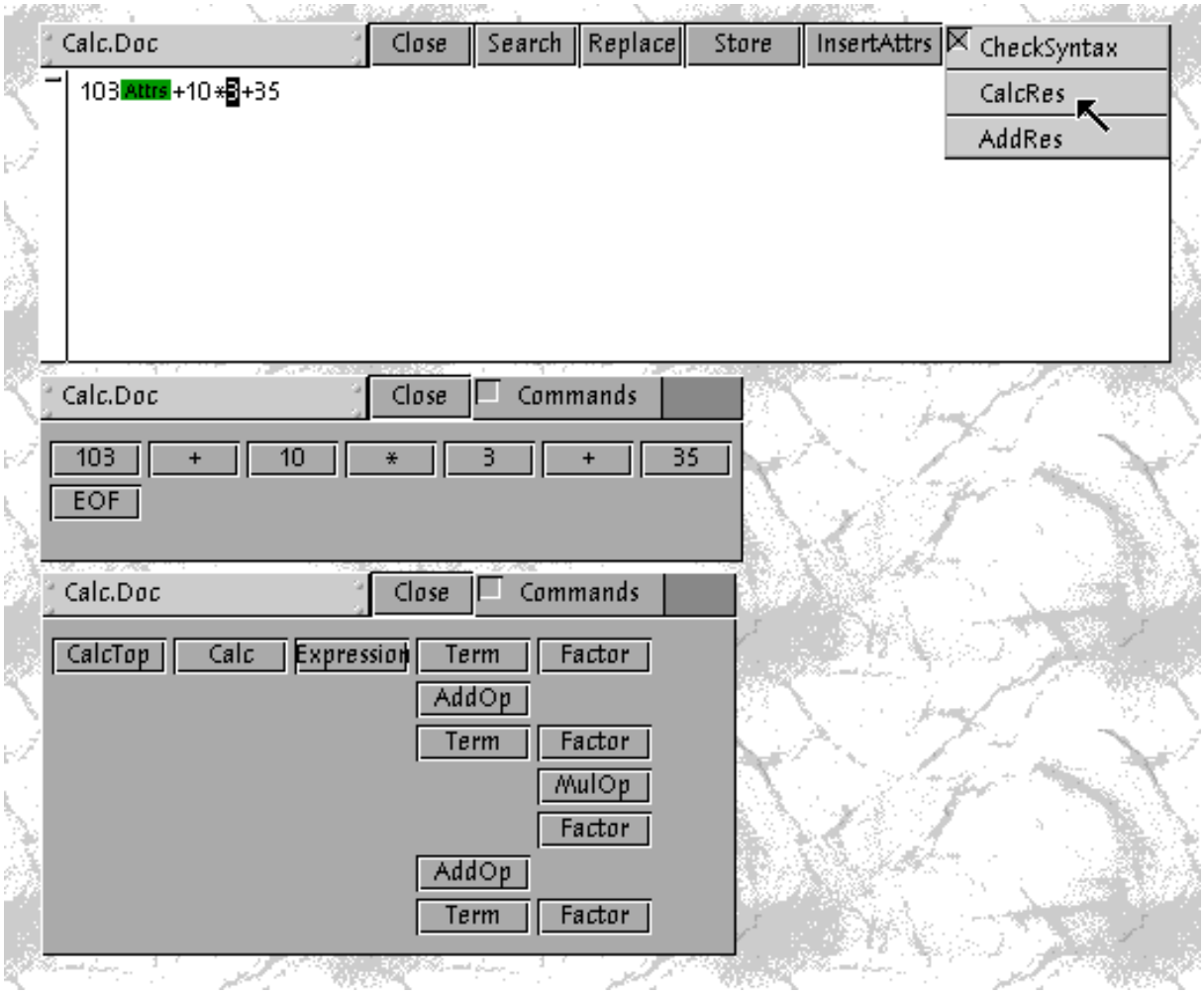

Abb. 8: GIPSY-Dokument mit integriertem Editor,
Operationen (Menu-Gadget) und Sichten (Werkzeuge)

4.1 Textdarstellung

Als Editor für die Textdarstellung von GIPSY-Dokumenten wird eine leicht abgeänderte Version des Standard-Text-Editors des Oberon Systems eingesetzt. Die Textdarstellung von GIPSY-Dokumenten unterscheidet sich von Oberon-Texten in der Hintergrundfarbe und in der erweiterten Menu-Leiste.

4.1.1 Menu-Leiste

Die Menu-Leiste von GIPSY-Dokumenten enthält neben den Standard-Buttons auch einen Button für das Einfügen von Attribut-Gadgets (siehe 4.1.3) und ein Popup-Menu (Abb. 9) für die Ausführung der Attributierungs-Operationen (Commands).

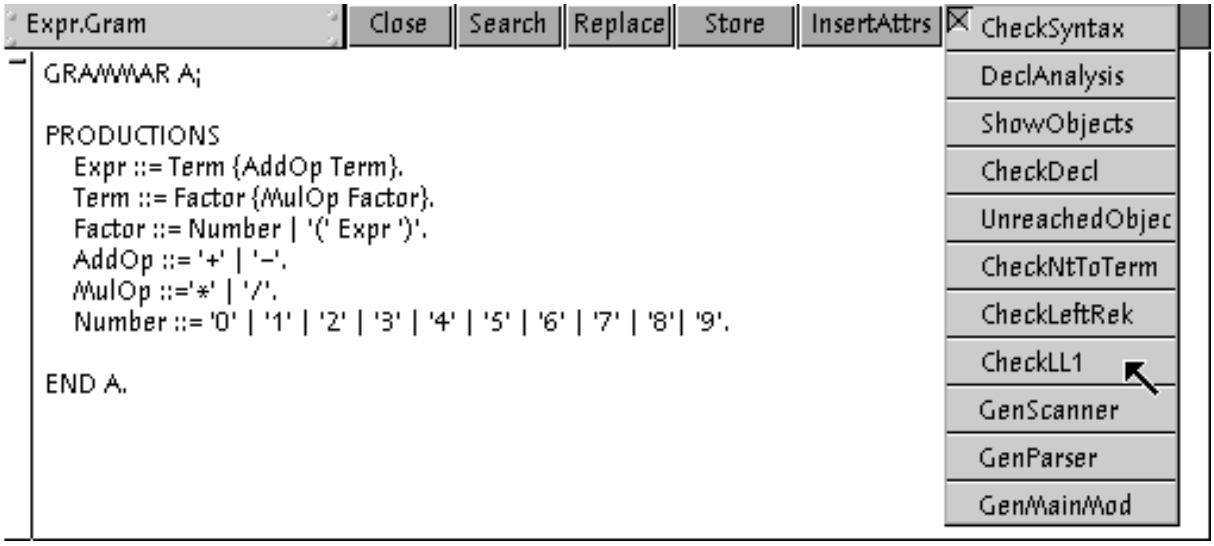

Abb. 9: Menu-Gadget für Attributierungs-Operationen

Für jede Attributierungsprozedur des Start-Nonterminalsymbols einer Grammatik wird ein Eintrag ins Menu-Gadget generiert und kann mit der Maus ausgewählt werden.
Zusätzlich ist das Menu auch über das GIPSYEnv-Panel (siehe 3.4.4) als eigenständiges Gadget erhältlich.
Die Attributierungs-Operationen einer Grammatik können auch unabhängig vom Menu-Gadget mit dem Oberon-Command

```
GIPSYDocs.ExecuteCmd CmdName
```

auf das markierte oder selektierte GIPSY-Dokument angewandt werden. Als 'CmdName' muss hier der Name der Attributierungs-Operation angegeben werden.

4.1.2 Fehlerelemente

Für jeden Fehler, der beim Überprüfen der Syntax (syntaktische Fehler) oder durch die Standardfunktion RULE [Hel95] in Attributierungsprozeduren (semantische Fehler) in einem GIPSY-Dokument auftaucht, wird im Text an der Fehlerposition ein Error-Gadget eingefügt.
Die Error-Gadgets sind mit der Fehlernummer beschriftet, die in den zur Grammatik gehörenden Fehlertext verweist. Die zugehörige Fehlermeldung wird durch einen Mausklick auf das Error-Gadget aus dem Fehlertext herausgelesen und im Oberon-Log ausgegeben (Abb. 10).

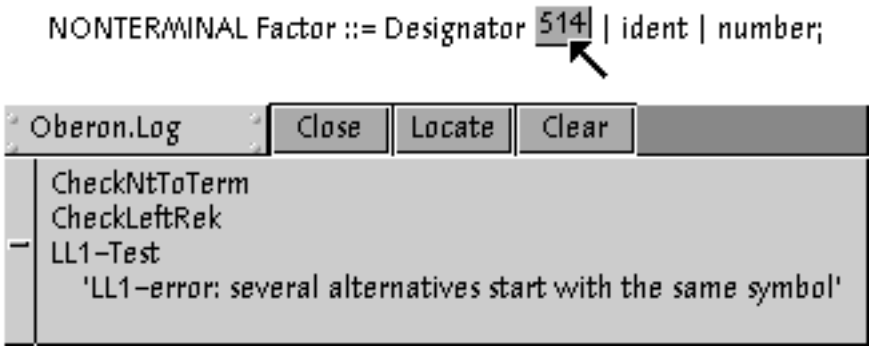

Abb. 10: Error-Gadget mit Ausgabe der Fehlermeldung im Oberon-Log

4.1.3 Attribut-Gadgets und Attribut-Inspektor

In der Spezifizierungssprache GIPSY/L [Hel95] können zu Nonterminalsymbolen Attribute deklariert werden. Für einige Standard-Typen von Attributen besteht die Möglichkeit, Attribut-Gadgets in den Text einfliessen zu lassen (Abb. 11).
Die Attribut-Gadgets können mit dem Button InsertAttrs in der Menu-Leiste oder mit dem Oberon-Command

GIPSYAttributes.InsertAttributes

eingefügt werden. Beide Operationen werden auf ein markiertes GIPSY-Dokument oder einen selektierten Textausschnitt angewendet.

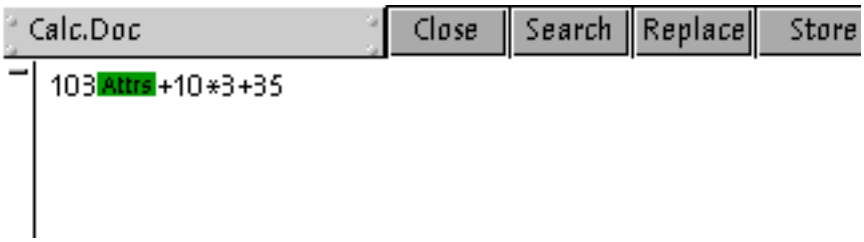

Abb. 11: Attribut-Gadget

Bei einem Mausklick auf ein Attribut-Gadget öffnet sich der Attribut-Inspektor (Abb. 12), mit dem die einzelnen Attribute eingesehen (Inspect) und verändert (Apply) werden können. Der Close-Button schliesst den Inspektor, ohne eine Änderung vorzunehmen.

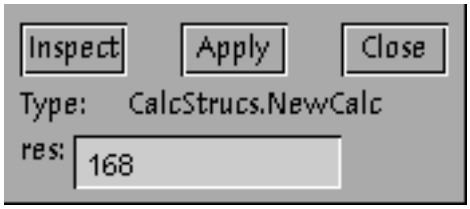

Abb. 12: Attribut-Inspektor

5 Sichten und Werkzeuge

Im GIPSY-Kern werden die Standard-Sichten *TokenView* und *StructureView* angeboten. Eigene Views können mit Hilfe des Moduls *GIPSYViews* einfach dazuprogrammiert werden.

5.1 Standard-Sichten

Die Standard-Sichten (Token- und Structure-View) veranschaulichen die Token-Liste bzw. den Struktur-Baum eines GIPSY-Dokumentes. Sie sind beide Bestandteile des GIPSY-Kerns, da sie nicht von einer bestimmten Umgebung abhängig sind.

Die Sichten können mit dem GIPSYEnv-Panel geöffnet werden.

Abb. 13 zeigt ein GIPSY-Dokument der Umgebung *Calc*, zu dem in den nachfolgenden Abschnitten die Token-Liste und der Struktur-Baum angegeben sind.

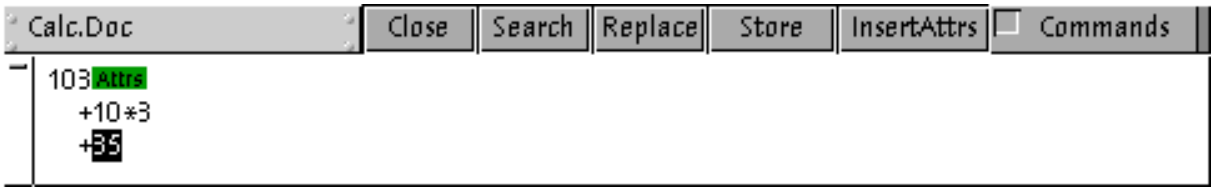

Abb. 13: GIPSY-Dokument

5.1.1 Token-Liste

Die Token-Liste stellt den Dokumenteninhalt als Folge von Tokens dar, wie sie vom Scanner geliefert werden (Abb. 14). Bei einem Mausklick auf ein Token wird in allen zugehörigen Text-Dokumenten der Textausschnitt selektiert, der das Token im Text einnimmt.

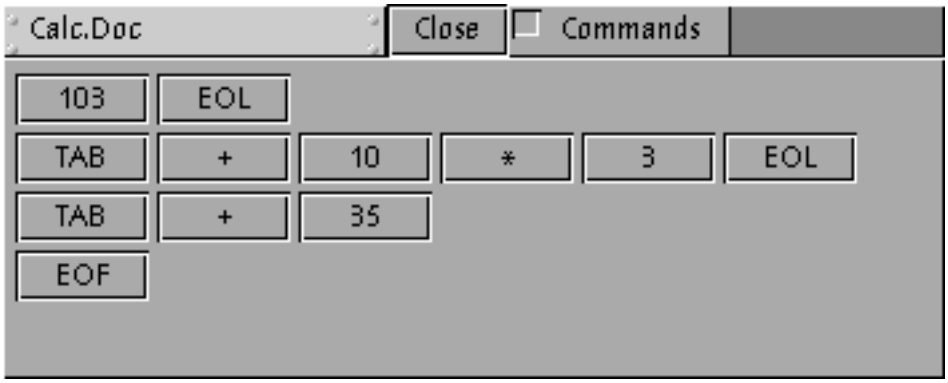

Abb. 14: Token-Liste (TokenView) für das Dokument aus Abb. 13

5.1.2 Struktur-Baum

Der Struktur-Baum zeigt den Dokumenteninhalt in Form eines Baumes mit den Nonterminalsymbolen als Knoten (Abb. 15). In der Horizontalrichtung ist die Sub-Knoten-Beziehung und in der Vertikalrichtung die Next-Knoten-Beziehung dargestellt.

Wie bei der Token-Liste kann auch hier der zu einem Knoten gehörende Textausschnitt mit einem Mausklick auf den entsprechenden Knoten selektiert werden.

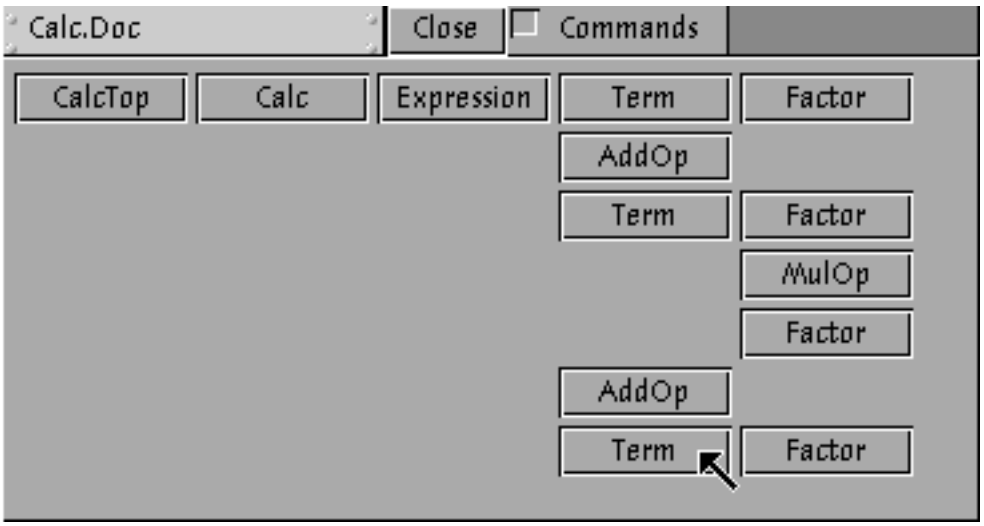

Abb. 15: Struktur-Baum (StructureView) für das Dokument aus Abb. 13

5.2 Programmierung eigener Views

Im Modul *GIPSYViews* sind die allgemeinen Aufgaben einer View implementiert. Dazu gehören:

- Notifikation von Commands
- Löschen und Nachführen der Darstellung
- Einbettung in ein GIPSY-Dokument

Die Hauptaufgabe bei der Programmierung neuer Views liegt somit nur noch im Überschreiben der Darstellungs-Prozedur *ShowView*, die den Syntaxbaum in der gewünschten Form in einem Panel visualisiert.

```
DEFINITION GIPSYViews;

  TYPE
    Panel = POINTER TO PanelDesc;
    PanelDesc = RECORD(Panels.PanelDesc)
      ShowCmd: ARRAY 32 OF CHAR;
      ShowView: PROCEDURE (P: Panel)
    END;
```

```
    PROCEDURE InitPanel(P: Panel);
    PROCEDURE NewPanel;
    PROCEDURE PanelHandler(obj: Objects.Object; VAR M: Objects.ObjMsg);
    PROCEDURE RemoveFrames(P: Panel);
    PROCEDURE Show; (* ^ GIPSYViews.PanelNewProc DocName *)
    PROCEDURE ShowDoc(P: Panel);

END GIPSYViews.
```

Im Feld *ShowCmd* eines Panels kann ein Command angegeben werden, der als Vorbedingung für die Darstellung der Sicht nötig ist. Falls nur das Vorhandensein des Strukturbaumes gefordert wird, genügt es, wenn in *ShowCmd* der Wert *Structures.CheckSyntax* eingetragen ist.

Eine View kann auf zwei Arten geöffnet werden:

1. Mit dem Aufruf der Prozedur *ShowDoc*, der das Panel als Parameter mitgegeben wird
2. Mit dem Command *GIPSYViews.Show*, dem eine Panel-New-Prozedur angefügt wird z.B.

   ```
   GIPSYViews.Show TokenViews.NewPanel [Name.Doc]
   ```

Im ersten Fall muss das Text-Modell (StructureText) vor dem Aufruf an das Panel gebunden werden. Im zweiten Fall wird das optional im Command angegebene Dokument geladen und dem Panel als Modell übergeben. Falls kein Dokument angegeben ist, wird versucht, beim Aufruf einen markierten bzw. selektierten Text an das Panel zu binden.

5.2.1 Ein Gerüst für eigene Views

```
MODULE SkeletonViews;

    IMPORT GIPSYViews,Objects,Structures;

    PROCEDURE ShowView(P: GIPSYViews.Panel);
        VAR L: Objects.LinkMsg;
    BEGIN
        L.name:='Model'; L.id:=Objects.get; L.res:=-1;
        P.handle(P,L);
        IF (L.res=0) & (L.obj#NIL) & (L.obj IS Structures.StructureText) THEN
            (* draw view *)
        END
    END ShowView;

    PROCEDURE InitPanel*(P: GIPSYViews.Panel);
    BEGIN
        IF P#NIL THEN
        Panels.Freeze(P,TRUE); (* verhindert die Manipulation von Gadgets durch den Benutzer *)
            P.ShowCmd:='aCmd';      (* z.B. Structures.CheckSyntax *)
            P.ShowView:=ShowView
        END
```

```
    END InitPanel;

    PROCEDURE NewPanel*;
        VAR obj: Objects.Object;
    BEGIN
        obj:=Gadgets.CreateObject('GIPSYViews.NewPanel');
        IF obj#NIL THEN
            InitPanel(obj(GIPSYViews.Panel))
        END
    END NewPanel;

END SkeletonViews.
```

6 Anhang

6.1 Literaturverzeichnis

[ASU86] Aho, A., Sethi, R., Ullman, J.: „Compilers: Principles, Techniques and Tools“. Addison-Wesley, 1986.

[Hel95] Helbling, A., Murer, T.: „Die Spezifizierungssprache GIPSY/L“. TIK-Report Nr. 13, Institut für Technische Informatik und Kommunikationsnetze, ETH Zürich, Mai 1995.

[Mar94] Marti, R.: „GIPSY: Ein Ansatz zum Entwurf integrierter Softwareentwicklungssysteme“. ETH Dissertation Nr. 10463. Verlag der Fachvereine, Zürich, 1994.

[MM92] Marti, R., Murer, T.: „Extensible Attribute Grammars“. TIK-Report Nr. 6, Institut für Technische Informatik und Kommunikationsnetze, ETH Zürich, Dezember 1992.

[Möss86] Mössenböck, H.P., Rechenberg, P.: „Ein Compiler-Generator für Mikrocomputer“. Carl Hanser Verlag München Wien, 1985.

[Möss90] Mössenböck, H. P.: „Coco/R: A Generator for Fast Compiler Front-Ends“. Departement Informatik, Institut für Computersysteme, ETH Zürich, Februar 1990.

Report 1 - A Relational Data Base Design for an X.500 Directory System Agent
(F. Perruchoud, C. Lanz, B. Plattner, July 1990)

Report 2 - Model and Functionality Definition for the Collaborative Editing Conferencing System Multim ETH.
(H. Lubich, July 1990)

Report 3 - X.400 Security Capabilities: Evaluation and Constructive Criticism
(M. Müller, August 1990)

Report 4 - CPU Evaluation for ADAM
(Schibli, M. Tadjan, February 1992)

Report 5 - Aspekte computergestützter Kooperation - Schriftliches Material eines Seminars an der ETH Zürich
(Hannes Lubich, Januar 1993)

Report 6 - Extensible Attribute Grammars
(R. Marti, T. Murer, December 1992)

Report 8 - Test Case Validation — TTCN Test Case Validation Against SDL Specifications
(F. Kristoffersen, T. Walter, May 1994)

Report 9 - Conformance and Interoperability - A critical assessment
(T. Walter, B. Plattner, September 1994)

Report 10 - OOP-Softwarearchitektur für Multimediakommunikation
(Serge Hoffmann, März 1995)

Report 11 - A Comparison of Selection Schemes used in Genetic Algorithms
(Tobias Blickle, Lothar Thiele, April 1995)

Report 12 - Spezifizieren und Generieren von integrierten Umgebungen mit GIPSY
(A. Helbling, T. Murer, Mai 1995)

Report 13 - Die Spezifizierungssprache GIPSY/L
(A. Helbling, T. Murer, Mai 1995)