

Das Softwarepaket VPLAN

Stefan Körkel

8. Oktober 2002

Inhaltsverzeichnis

1	Überblick	5
2	Arbeitsweise des Softwarepakets VPLAN	6
3	Struktur der Software	12
4	Werkzeug zur automatischen Ableitungserzeugung	23
5	Grafische Benutzeroberflächen	24
	Literaturverzeichnis	27

1 Überblick

Dieser Artikel beschreibt das Softwarepaket VPLAN. Es handelt sich hierbei um das neunte Kapitel meiner Doktorarbeit [9].

Bei Konzeption und Implementierung wurde auf die folgenden Prinzipien Wert gelegt:

- Verwendung bzw. Entwicklung von State-of-the-Art-Numerik,
- Unabhängigkeit von speziellen Anwendungsproblemen,
- Bedienungsfreundlichkeit,
- Plattformunabhängigkeit,
- Performance,
- Stabilität.

Als Programmiersprache wurde C++ [13] gewählt. Dadurch erhält man die nötige Flexibilität, um die komplexen Datenstrukturen von Versuchsplanungsproblemen im Rechner darstellen zu können. Da auch auf vorhandene Numerik-Bibliotheken zugegriffen wird, sind Teile des Programms in Fortran implementiert, die über geeignete Schnittstellen mit den C++-Klassen zusammenwirken.

Es wurde auf die strikte Trennung zwischen dem Programm einerseits und der Beschreibung und Implementation der Anwendungsprobleme andererseits Wert gelegt. So erreichen wir Flexibilität und Problemunabhängigkeit der Software bei gleichzeitigem hohem Bedienungskomfort. Möglich wird dies durch die Verwendung moderner Techniken für automatische Ableitungserzeugung und dynamisches Linken.

Die Anwendungsprobleme werden mit Hilfe von ASCII-Files beschrieben. Dies macht die Anbindung von grafischen Benutzeroberflächen und Modellgeneratoren einfach möglich, zum Beispiel über Client-Server-Techniken. Mit dem Softwarepaket MGAD stellen wir dafür eine Java-Implementierung bereit.

Die Ergebnisausgabe erfolgt ebenfalls mittels ASCII-Files, zur Visualisierung der Daten ist eine Schnittstelle vorhanden.

Durch die Verwendung von Techniken der automatischen Ableitungserzeugung ist die Arbeit mit VPLAN für den Benutzer ableitungsfrei.

Das Softwarepaket VPLAN besteht im wesentlichen aus zwei Programmen: vplan98 und doit. doit („Design Of experiments Initialization Tool“) erledigt die Ableitungsgenerierung und Erzeugung des kompilierten Problemmoduls, vplan98 ist für die Durchführung der numerischen Berechnungen zuständig.

In der augenblicklichen Version benutzt VPLAN die Programmpakete SNOPT [7, 8] und ADIFOR [3, 5, 4] sowie NAG- und LAPACK-Routinen [11, 1]. Außerdem werden DAE-SOL [2] und eine Mehrfachexperimentversion von PARFIT [15] aufgerufen. Die Schnittstellen zu Integrator, Optimierungslöser, Parameterschätzprogramm und Lineare-Algebra-Bibliotheken sind modular gehalten, so daß diese Komponenten gegebenenfalls durch alternative Software ausgetauscht werden können.

Bei der Entwicklung wurde auf Plattformunabhängigkeit Wert gelegt. So läuft VPLAN auf Unix-Betriebssystemen wie Linux, Solaris, Irix und AIX sowie auf Windows NT und 9x. Die Versionsverwaltung der Softwareentwicklung wird mit dem Werkzeug CVS [6] durchgeführt.

VPLAN hat sich inzwischen im professionellen Einsatz an der Universität und in der Industrie bewährt.

In den folgenden Abschnitten wollen wir eine Einführung in die Arbeitsweise und Verwendung des Softwarepakets geben. Außerdem sollen die Konzepte der Implementierung vorgestellt werden.

2 Arbeitsweise des Softwarepakets VPLAN

2.1 Leistungsumfang

Der Leistungsumfang des Programmpakets VPLAN umfaßt verschiedene mathematische Methoden zur Behandlung von Simulations- und Optimierungsaufgaben für Mehrfachexperiment-Systeme mit zugrundeliegenden DAE-Modellen.

Die folgende Übersicht stellt die einzelnen Anwendungsmodi kurz dar:

- *Integration:* Die DAE-Systeme für die Experimente des Versuchsplans werden integriert. Ausgabedaten werden bereitgestellt, anhand derer man die Trajektorien visualisieren kann. Dies benutzt man in der ersten Phase der Modellierung, bevor Nebenbedingungen und Meßverfahren spezifiziert werden.
- *Simulationsumgebung:* Zusätzlich zur Integration werden die Meßfunktionen ausgewertet. Kovarianzmatrix, Gütekriterien und die Näherungen für die Standardabweichungen der Parameter werden berechnet, außerdem die Kosten des Versuchs. Die Nebenbedingungen an Versuchsplanungsgrößen und Zustandsvariablen werden auf Zulässigkeit überprüft. Sind Meßwerte vorhanden, werden Residuen berechnet, pro Meßwert, pro Experiment und für das gesamte Mehrfachexperiment. Die Simulationsumgebung stellt damit ein machtvolles Werkzeug zur Simulation und Beurteilung von Versuchsplänen zur Verfügung. Sie kann, wenn sie interaktiv eingesetzt wird, benutzt werden, um ein intuitives Verständnis für das Verhalten des Modells zu bekommen und verschiedene Konstellationen auszuprobieren.

- *Erzeugen von Pseudo-Meßdaten:* Durch Integration der Experimente, Auswerten der Meßfunktionen und Aufaddieren von „Rauschen“ können simulierte Meßdaten erzeugt werden.
- *Parameterschätzung:* Dieser Modus ermöglicht das Lösen von Mehrfachexperiment-Parameterschätzproblemen durch Aufruf von PARFIT.
- *Versuchsplanung:* Dies ist die zentrale Aufgabe von VPLAN. Die Versuchsplanungsgrößen der variablen Experimente werden so berechnet, daß ein Gütekriterium auf der Kovarianzmatrix optimiert und die Nebenbedingungen erfüllt werden. In die Berechnung der Kovarianzmatrix können neben den zu optimierenden Experimenten auch feste, bereits durchgeführte Experimente eingehen.

2.2 Bedienung des Programmpakets

Die Bedienung von VPLAN ist einfach und benutzerfreundlich. Zunächst muß das Programmpaket installiert und kompiliert werden, dies wird weiter unten in diesem Kapitel beschrieben. Will man nun mit einem Anwendungsproblem arbeiten, muß man die folgenden Schritte durchführen:

1. Anlegen des Problemverzeichnisses,
2. Ablegen von Modell- und Datenfiles im Problemverzeichnis,
3. Aufruf des Programms doit zur automatischen Ableitungsgenerierung und Erzeugung des Problemmoduls,
4. Aufruf von vplan98 zur Durchführung der numerischen Berechnungen, wobei einer der oben beschriebenen Modi gewählt werden kann,
5. Zugriff auf die Ausgabefiles im Problemverzeichnis.

Da die Ausgabefiles die gleiche Struktur wie die Eingabefiles haben, ist auf einfache Weise der wiederholte und sukzessive Aufruf von vplan98 möglich.

2.3 Spezifikation der Ein- und Ausgabedaten

Die Anforderungen zur Beschreibung von Modellen und Daten wurden gemeinsam mit Projektpartnern, Anwendern aus der chemischen Industrie und Informatikern, entwickelt und formuliert. Dabei wurden die folgenden Prinzipien zugrundegelegt:

- Flexibilität,
- Modularität,
- Trennung von Modell und Daten einerseits und Programm andererseits,

- Vermeidung von Redundanz,
- Bedienungsfreundlichkeit.

Die Beschreibung eines Anwendungsbeispiels umfaßt

- ini-Files zur Problemspezifikation,
- Fortran-Files zur Beschreibung von Modellfunktionen und
- Meßdatenfiles.

Auf Einzelheiten gehen wir unten ein.

Alle Dateien zur Problembeschreibung können natürlich auch von entsprechenden Modellgeneratoren automatisch erzeugt werden. Wir stellen mit MGAD ein solches Tool zur Verfügung, siehe unten. Die Anbindung an kommerzielle Modellierungstools ist durch die Entwicklung von Import- und Export-Filtern leicht zu bewerkstelligen.

ini-Files

Zunächst wollen wir die Files zur Problemspezifikation betrachten. Ihre Syntax wurde an die von ini-Files, wie sie zum Beispiel bei Microsoft Windows verwendet werden, angelehnt. Diese Files werden sowohl zur Spezifikation der Eingaben als auch zur Ausgabe der Programmergebnisse verwendet.

Es gibt zwei Arten von ini-Files:

- **vplan.ini** beschreibt den Gesamtrahmen des Versuchsplanungsproblems. In diesem File werden in verschiedenen Abschnitten definiert: die Aktion, die durchgeführt werden soll (Integration, Simulation, Versuchsplanung, . . .), die Pfade von Problemverzeichnis und Unterverzeichnissen, die Modellparameter, die Experimente, die zusammen den Versuchsplan ergeben, das anzuwendende Gütekriterium, die Namen der Meßdatenfiles und Ausgabefiles sowie die numerischen Optionen für Versuchsplanung und Parameterschätzung. Als Ausgabefile enthält **vplan.ini** außerdem Werte aller Gütekriterien, des Residuums und der Gesamtkosten des Mehrfachexperiments.
- Zur Spezifikation der einzelnen Experimente gibt es jeweils ein **experiment.ini**-File. Es enthält Abschnitte für: das Flag, ob das Experiment festgehalten werden soll oder nicht, das Integrationsintervall, die Namen der Modellfunktionen, differentielle und algebraische Zustandsvariablen, die Beschreibung der dynamischen Nebenbedingungen und des Gitters zur Überprüfung der Nebenbedingungen, Steuergrößen und Steuerfunktionen sowie Nebenbedingungen an die Steuergrößen, die Beschreibung der Meßfunktionen und der Meßstruktur sowie Optionen für die Integration dieses Experiments.

Werden durch die Berechnungen von vplan98 Werte von Variablen verändert, so stehen in den Ausgabe-ini-Files die Ergebnisse an den Stellen, an denen in den Eingabe-ini-Files die Ausgangswerte standen.

Abbildung 1 zeigt exemplarische Ausschnitte aus vplan.ini und experiment.ini.

<pre> ; vplan.ini [Aktion] aktion=Simulationsumgebung [Pfade] problempath=phosphin/ inpath=in/ outpath=out/ messpath=mess/ plotpath=plot/ fortranpath=fortran/ [Parameter] pAnzahl=4 p1=alpha1 1 p2=alpha2 1 p3=ea 0.99768 p4=f1 2.138658 [Versuchsplan] expAnzahl=2 exp1=experiment.ini exp1.ini exp2=experiment.ini exp2.ini [Guetekriterium] Optimierungskriterium=A [OptionenVersuchsplanung] maxit=250 opttol=1e-06 linfeas=1e-07 nlinfeas=0.3 maxitQP=40 realworkspace=30000 integerworkspace=10000 ... </pre>	<pre> ; experiment.ini [Flags] switch=1 ; Experiment fest oder variabel [Integrationsintervall] t0=0 tend=180 [Zustandsvariablen] ; y: differentielle Variablen yAnzahl=2 y1=n1 na1 -1e+10 1e+10 y2=temp temp0 -1e+10 365.16 ; z: algebraische Variablen zAnzahl=3 z1=n2 0 -1e+10 1e+10 z2=n3 0 -1e+10 1e+10 z3=n4 7.64 -1e+10 1e+10 [Steuergroessen] qAnzahl=3 q1=na1 3.74 0.8 20 0 -1 q2=temp0 343 323.16 351.16 0 -1 q3=na4 7.64 3 14 0 -1 [Messungen] tAnzahl=15 t1=12 t1Anzahl=2 t1m1=mfcn1 0.333 1e-06 1 t1m2=mfcn2 0.333 1e-06 1 t1minmax=0 2 ... </pre>
---	---

Abbildung 1: Exemplarische Ausschnitte aus vplan.ini und experiment.ini

Fortran-Files

Alle benutzerdefinierten Subroutinen für Modellfunktionen, Meßfunktionen usw. werden als Fortran-Source-Files abgelegt. Fortran wurde als Beschreibungssprache gewählt, da dafür für die automatische Ableitungserzeugung mit ADIFOR [4] ein leistungsfähiges Werkzeug zur Verfügung steht.

Die folgende Übersicht beschreibt die auftretenden Typen von Fortran-Routinen zur Modellspezifikation mit einer jeweiligen Erläuterung der Aufrufparameter. Jede Subroutine wird in einem eigenen Fortran-File abgelegt, das den Namen der Subroutine und das Suffix `.f` hat.

1. *Modellfunktionen*: rechte Seiten des DAE-System für differentielle und algebraische Gleichungen `ffcn.f` und `gfcn.f`

```
subroutine ffcn( t, x, f, p, q, rwh, iwh, iflag )
```

- `t`: Zeit (Input)
- `x`: Vektor der Zustandsvariablen (Input)
- `f`: Vektor der Funktionswerte von `ffcn` (Output)
- `p`: Vektor der Modellparameter (Input)
- `q`: Vektor der Steuergrößen (Input)
- `rwh`: Hilfsvektor zur Parametrisierung der Steuergrößen (Input)
- `iwh`: Hilfsvektor zur Parametrisierung der Steuergrößen (Input)
- `iflag`: Statusflag (Output)

```
subroutine gfcn( t, x, g, p, q, rwh, iwh, iflag )
```

- `g`: Vektor der Funktionswerte von `gfcn` (Output)

2. *Meßfunktionen* `mfcn.f` und *Sigmafunktionen* `sigma.f`: Modellantwort für Meßwert und Standardabweichung einer Messung

```
subroutine mfcn( t, x, h, p, q, rwh, iwh, iflag )
```

- `h`: Funktionswert von `mfcn` (Output)

```
subroutine sigma( t, x, s, p, q, rwh, iwh, iflag )
```

- `s`: Funktionswert von `sigma` (Output)

3. *Steuerconstraints* `cfcn.f`

```
subroutine cfcn( c, q, iflag )
```

- `c`: Vektor der Funktionswerte von `cfcn` (Output)

4. *Dynamische Nebenbedingungen* `bfcn.f`

```
subroutine bfcn( t, x, b, p, q, rwh, iwh, iflag )
```

- `b`: Vektor der Funktionswerte von `bfcn` (Output)

5. Plotfunktion `plot.f`

```
subroutine plot( t, x, p, q, nga, nt, wron,  
&              ngq, gq, ngaq1, ngaq2, gaq, rwh, iwh )
```

- `nga`: Leading Dimension von `wron` (Input)
- `nt`: Anzahl Spalten von `wron` (Input)
- `wron`: Ableitungsmatrix der Zustände nach den Parametern (Input)
- `ngq`: Leading Dimension von `gq` (Input)
- `gq`: Ableitungsmatrix der Zustände nach den Steuergrößen (Input)
- `ngaq1`: Erste Leading Dimension von `gaq` (Input)
- `ngaq2`: Zweite Leading Dimension von `gaq` (Input)
- `gaq`: Zweite Ableitungsmatrix der Zustände nach Parametern und Steuergrößen (Input)

Meßdaten- und `covmat`-Files

Weitere Dateien zur Bereitstellung von Eingaben oder Ausgaben von VPLAN sind

- *Meßdatenfiles* `mess.dat`. In ihnen werden zeilenweise die aufsteigend geordneten Meßzeitpunkte und dahinter jeweils paarweise Meßwert und Standardabweichung abgelegt.
- *Kovarianzmatrix-File* `covmat.mat` zur Ausgabe der Kovarianzmatrix. Für robuste Versuchsplanung wird diese Datei auch als Inputfile verwendet.

2.4 Visualisierungsschnittstelle

Das Programmpaket VPLAN stellt eine Schnittstelle zur Verfügung, die die Visualisierung der berechneten Trajektorien ermöglicht. Grundlage dafür ist die kontinuierliche, fehlerkontrollierte Ausgabe durch den Integrator DAESOL, die durch Auswertung des BDF-Polynoms auf einem äquidistanten, benutzerdefinierten Gitter erfolgt, das nicht von der adaptiv gewählten Integratorschrittweite abhängig ist.

Die kontinuierliche Ausgabe wird somit spezifiziert durch Angabe der Ausgabeschrittweite und die Plotfunktion, die auf diesen Gitterpunkten aufgerufen werden soll.

VPLAN erzeugt dann Ausgabefiles, in denen zeilenweise die Zeitpunkte und die Ausgabewerte stehen. Ausgabewerte können Funktionen der Zustandsvariablen, Parameter, Steuergrößen, Steuerfunktionen und der Sensitivitäten sein.

Diese Datenfiles können leicht mit Visualisierungsprogrammen angezeigt werden. Wir stellen ein Interface zur Ausgabe von Mehrfachexperiment-Plots mit GNU PLOT [17] zur

Verfügung. Mit dem darauf aufbauenden Tool `vplan2html` kann diese Ausgabe automatisch in HTML-Seiten eingebunden werden. Abbildung 2 zeigt ein Beispiel. Eine grafische Darstellung der Trajektorien ist aber zum Beispiel in einfacher Weise auch mit Matlab [10] möglich.

Eine andere Möglichkeit besteht darin, auch während der Iterationen der Optimierung auf die kontinuierliche Ausgabe zuzugreifen und so Zwischenergebnisse und den Verlauf der Optimierung grafisch darzustellen. Die Kommunikation zwischen `vplan98` und dem Visualisierungsprogramm erfolgt dabei über `named pipes`. Ein Screenshot dieser Online-Visualisierung ist in Abbildung 3 dargestellt.

3 Struktur der Software

In diesem Abschnitt wollen wir eine Übersicht über die Implementierung und Struktur des Programmpakets VPLAN geben. Diese Struktur besteht zunächst aus der Organisation der Verzeichnisse, in denen Sourcecode, Programm und Anwendungen abgelegt werden. Der Sourcecode selbst ist in Module gegliedert für die verschiedenen methodischen und softwaretechnischen Teilaspekte des Programms. Die Daten des Programms werden in verschiedenen Klassen verwaltet. Schließlich wird der Informationsfluß der Programmausführung durch eine Reihe von Funktionsaufrufen bewerkstelligt.

3.1 Verzeichnisstruktur

Die Verzeichnisstruktur einer VPLAN-Installation, siehe Abbildung 4, besteht aus drei Unterverzeichnisbäumen unter dem Verzeichnis `vplanroot`. Unter `examples` werden in Problemverzeichnissen die Daten für die Anwendungsprobleme abgelegt. `src` enthält den kompletten Quellcode des Programms. Durch das Build-Tool `make` wird das Programmpaket kompiliert. Durch Aufruf von `doit` werden die Problemmodule erzeugt und kompiliert. Der dadurch erzeugte Binary-Code wird im Verzeichnisbaum `target` abgelegt.

Im Teilbaum `src` befinden sich Unterverzeichnisse für die verschiedenen Programmmodule und für das Programm `doit`, Headerdateien in `include` sowie Installationskripte in `etc`.

Der Teilbaum `examples` enthält Unterverzeichnisse für die einzelnen Anwendungsprobleme. Jedes Problemverzeichnis hat wiederum eine Unterstruktur mit den Verzeichnissen `in` für Input-ini-Files, `out` für Output-ini-Files, `mess` für Meßdatenfiles, `plot` für Output-Plotfiles und `fortran` für Fortran-Files.

Während in `src` und `examples` die plattformunabhängigen ASCII-Files für Sourcecode und Problembeschreibungen stehen, werden unter `target` alle plattformabhängigen, im wesentlichen durch Kompilation erzeugten Binary-Files abgelegt. Das Unterverzeichnis `bin` enthält dabei alle ausführbaren Programme und Skripten des Programmpakets, `lib` die Shared-Library-Files für die Programmmodule, `obj` Unterverzeichnisse für die Objekt-Files der Programmmodule und `modules` Unterverzeichnisse für die Problemmodule.

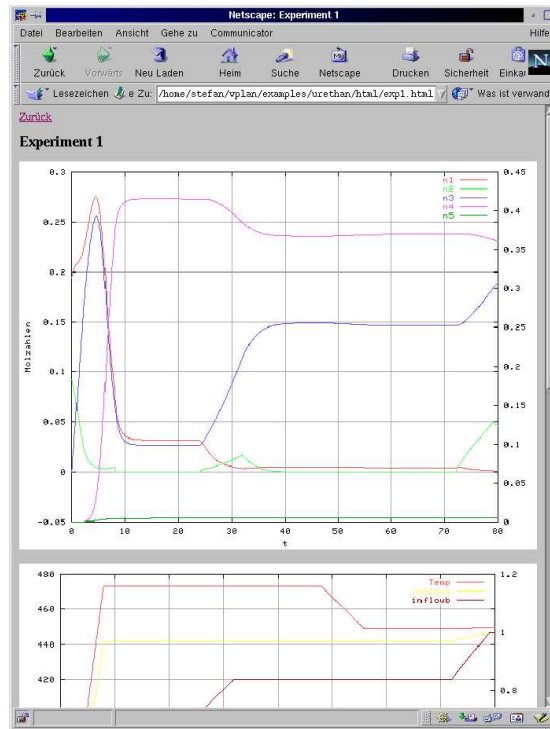


Abbildung 2: HTML-Ausgabe der Ergebnisse mit vplan2html

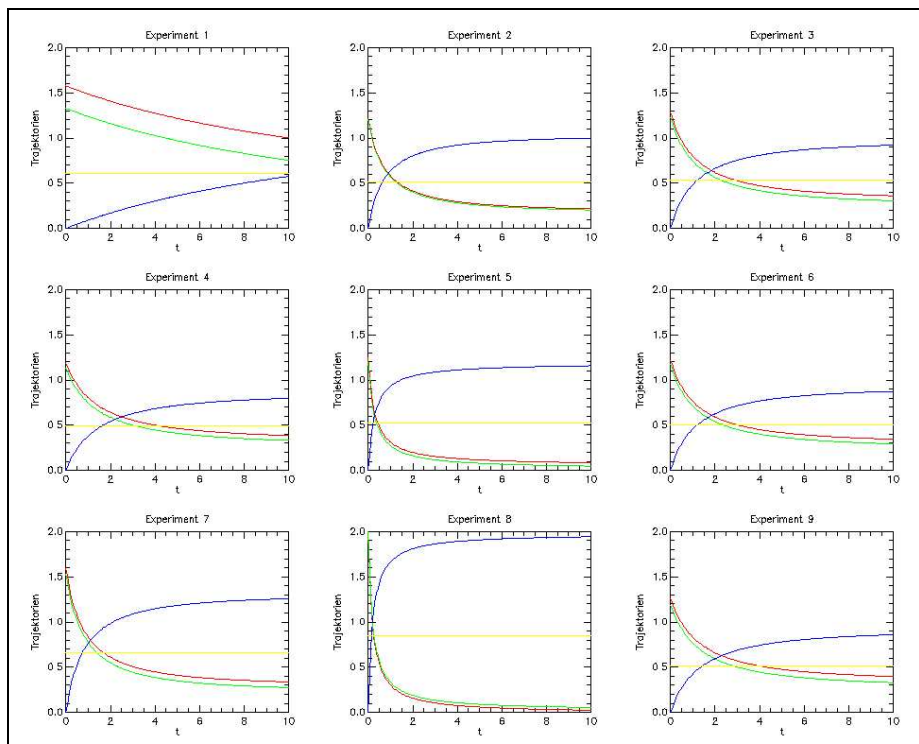


Abbildung 3: Screenshot der Online-Visualisierung

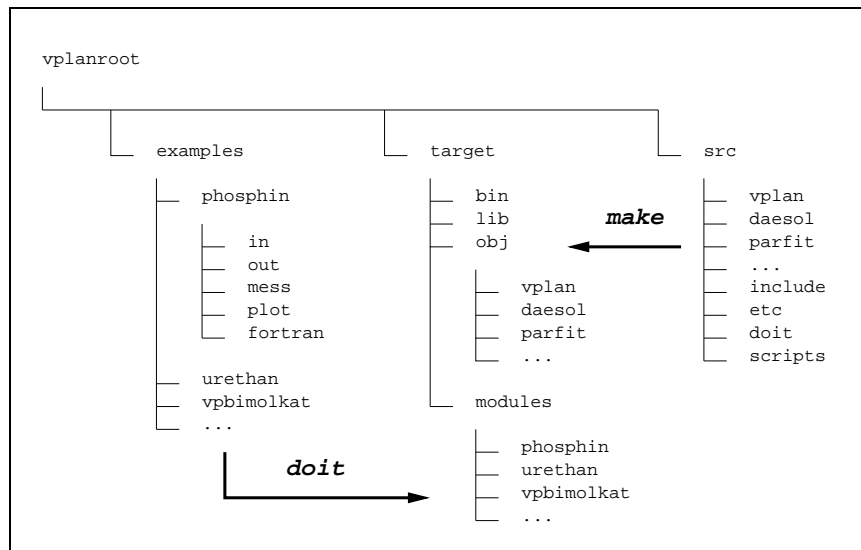


Abbildung 4: Verzeichnisstruktur des Softwarepakets VPLAN

3.2 Module und Kompilation

Programmmodule

Der Code von VPLAN besteht aus mehreren, logisch und softwaretechnisch zusammengehörenden Einheiten. Diese nennen wir Module. Für jedes Modul wird bei der Kompilation eine Shared-Library erzeugt. Jedes Modul selbst setzt sich aus mehreren C++- oder Fortran-Object-Files zusammen.

Im einzelnen gibt es die folgenden Programmmodule:

- *vplan*: Hauptprogramm, Datenstrukturen, Funktionen und Methoden für Versuchsplanung und Simulationsumgebung; in C++ implementiert,
- *tools*: Einlese- und Ausgaberroutinen; C++,
- *sqp*: Schnittstelle zu SQP-Solvern; C++,
- *vplpar*: Schnittstelle zu PARFIT und seinen Datenstrukturen; C++,
- *daesol*: Implementierung des Integrators DAESOL; Fortran,
- *daesupp*: Schnittstelle zwischen DAESOL und den Modellfunktionen und deren Ableitungen; C++,
- *parfit*: Mehrfachexperiment-Version des Parameterschätzprogramms PARFIT; Fortran,
- *stat*: Routinen zur Auswertung statistischer Verteilungen; C++,
- *vplnag*: benötigte Routinen der NAG-Bibliothek; Fortran,

- *vplpack*: benötigte Routinen der LAPACK-Bibliothek; Fortran,
- *doit*: Quelltext des Programms *doit* zur automatischen Ableitungsgenerierung durch Aufruf von ADIFOR und zur Erzeugung der Problemmodule; C++,
- *cov2corr*: Quelltext eines Hilfsprogramms zur Berechnung der Korrelationsmatrix aus der Kovarianzmatrix; C++.

Problemmodule

Für jedes Anwendungsproblem wird durch Aufruf von *doit* ein Problemmodul erzeugt. Im entsprechenden Unterverzeichnis von `vplanroot/target/modules` wird eine Datei `libproblem.so` angelegt. Diese Bibliothek enthält alle Routinen für das Anwendungsbeispiel und kann zur Laufzeit von `vplan98` dynamisch geladen werden, siehe unten. Durch diese Technik wird erreicht, daß Programmcode und Anwendungsbeispiele softwaretechnisch vollständig getrennt sind und erst zur Laufzeit von `vplan98` zusammengeführt werden. Dies gibt dem Programm eine sehr weitgehende Flexibilität.

Kompilation

Die Kompilation des Programmpakets erfolgt einfach durch den Aufruf von `make`. Eine Hierarchie von Makefiles erledigt dabei alle notwendigen Aufgaben. Gegebenenfalls wird das `target`-Verzeichnis angelegt. Die Programmmodule werden kompiliert und zu Dynamic Link Libraries (Shared Objects) gelinkt. Das ausführbare Programm `vplan98` kann so sehr kompakt gehalten werden. Alle Programmmodule werden beim Programmstart von `vplan98` durch den dynamischen Linker geladen.

Die Makefiles sind getrennt in plattformunabhängige Teile `makefile.all` und plattformabhängige Teile `makefile`. Das Verzeichnis `src/etc` enthält plattformabhängige Makefiles mit speziellen Einstellungen für Rechner mit verschiedenen Betriebssystemen wie Linux, Solaris, Irix, AIX, Windows NT, Windows 9x. Durch Aufruf eines Skripts `update.sh` wird jeweils das geeignete `makefile` in den `src`-Unterverzeichnissen installiert. So kann für jede VPLAN-Installation eine optimale Plattform- und Rechneranpassung durchgeführt werden.

Bei der Installation auf den Computern von Anwendern besteht die Möglichkeit, den kompletten Sourcecode nach der Kompilation zu entfernen. Das Programm bleibt dabei selbstverständlich voll lauffähig.

3.3 Speicherung der Daten in VPLAN

Die Programmiersprache C++ erlaubt die Definition adäquater Datenstrukturen für Versuchsplan, Experiment und Dynamik. In diesem Abschnitt wollen wir einen Überblick über diese Klassen geben.

- Die Klasse `data` definiert die umfassende Datenstruktur des Programms `vplan98`.

```

class data
{ ...

    public:
        int Ndynpool;           // Anzahl verschiedener Dynamiken
        DAEynamics *dynpool;   // Array der Dynamiken

        int Nexpool;           // Anzahl verschiedener Experimentbeschreibungen
        experiment *expool;    // Array der Experimentbeschreibungen

        Mexperiment V;         // Mehrfachexperiment-Versuchsplan

        int NP;                 // Anzahl der Parameter
        Matrix *parray;         // Array der Parameter

        PATHSTR problempath;    // Problemverzeichnis
        PATHSTR inpath;         // Unterverzeichnis fuer Inputdateien
        PATHSTR outpath;        // Unterverzeichnis fuer Outputdateien
        PATHSTR messpath;       // Unterverzeichnis fuer Messdatenfiles
        PATHSTR plotpath;       // Unterverzeichnis fuer Plotfiles
        PATHSTR fortranpath;    // Unterverzeichnis fuer Fortran-Files

        ...

        data( int NP_ = 1, int NPSETS_ = 1,
              int Ndynpool_ = 1, int Nexpool_ = 1 ); // Konstruktor
        ~data( void );                               // Destruktor

        int init( int NP_ = 1, int NPSETS_ = 1,
                  int Ndynpool_ = 1,
                  int Nexpool_ = 1 );                // Initialisierung

        ...
};

```

Von dieser Klasse wird das globale Objekt `db` angelegt. So kann auf die Daten auch unterhalb der Fortran-Funktionsaufrufe zugegriffen werden. Konstruktor und Destruktor von `db` erledigen Aufgaben, die am Programmstart bzw. Programmende durchgeführt werden müssen. Dies ermöglicht eine einfache Fehlerbehandlung: Der Destruktor von `db` wird auch im Fehlerfall am Programmende aufgerufen und kann dann alle Aufräumarbeiten erledigen.

- Die Klasse `Mexperiment` enthält alle Daten zur Beschreibung eines Mehrfachexperiment-Versuchsplans.

```

class Mexperiment
{ ...

    public:
        int Nex;                // Anzahl der Experimente
        experiment **ex;        // Array mit Zeigern auf Experimentbeschreibungen

        PATHSTR *infile;       // Input-ini-Files

```

```

PATHSTR *outfile;      // Output-ini-Files
PATHSTR *messfile;    // Messdatenfiles
PATHSTR *pltfile;     // Outputfiles fuer Visualisierung

plotfuntype *plotfcn; // Plot-Funktionen fuer kontinuierlichen Output
NAMESTR *plotname;    // Namen
double *plotstep;     // Schrittlängen

int *Nmessvals;       // Anzahl der Messwerte pro Experiment
double **messvals;    // Messwerte
double **sigmavals;   // Standardabweichungen der Messwerte

int *Nmesstimes;     // Anzahl der Messzeitpunkte pro Experiment
double **messtimes;  // Messzeitpunkte

char Aktion;         // auszufuehrende Aktion
char OptKrit;        // Guetekriterium, nach dem optimiert werden soll

PATHSTR covmat;      // Outputfile fuer die Kovarianzmatrix

                                // Numerische Optionen fuer Versuchsplanung:
int maxitSQP;        // Maximale Anzahl der SQP-Iterationen
double opttolSQP;    // Abbruchkriterium

...

Mexperiment( int Nex_ = 1 );    // Konstruktor
virtual ~Mexperiment( void );   // Destruktor

virtual int init( int Nex_ );   // Initialisierung

...
};

```

- In der Klasse `experiment` sind alle Daten zur Experimentbeschreibung abgelegt.

```

class experiment
{ ...

public:
    int swflag;          // Schalter: Experiment festhalten oder nicht

    DAEynamics *dyn;    // Zugrundeliegende Dynamik

    Matrix x0;          // Feste Werte der Anfangswerte
    Matrix q;           // Versuchsplanungsgroessen
    Matrix w;           // Gewichte

    int NU;             // Anzahl der kontinuierlichen Steuerfunktionen
    int *umode;         // Modus der Diskretisierung

    double *t;          // Zeitpunkte
    int *tflag;         // Typ des Zeitpunkts

    int Nmess;          // Anzahl Messfunktionen
    messtype *mess;     // Messfunktionen

```

```

...
experiment( int NQ_ = 1, int NQO_ = 1,
            int NU_ = 1, int NS_ = 1,
            int Nmess_ = 1 );           // Konstruktor
virtual ~experiment( void );           // Destruktor

virtual int init( int NQ_, int NQO_,
                 int NU_, int NS_,
                 int Nmess_,
                 double minmess_,
                 double maxmess_,
                 DAEynamics *DYN );    // Initialisierung
...
};

```

- Die Klasse `dynamic` implementiert die Daten zur Beschreibung jeweils eines dynamischen Modells.

```

class DAEynamics
{ ...

public:
    int nvar;           // Anzahl aller Zustandsvariablen
    int ndiff;         // Anzahl der differentiellen Variablen

    ffcndensefunctype FFCN; // Rechte Seite f der diff. Gleichungen
    gfcndensefunctype GFCN; // Rechte Seite g der alg. Gleichungen

    x_fgfcntype X_FFCN;   // Ableitungen von ffcn
    p_fgfcntype P_FFCN;
    q_fgfcntype Q_FFCN;
    x_x_fgfcntype X_X_FFCN;
    x_p_fgfcntype X_P_FFCN;
    q_x_fgfcntype Q_X_FFCN;
    q_p_fgfcntype Q_P_FFCN;

    ...

    // Numerische Optionen fuer DAESOL:
    double rtol;        // relative Fehlertoleranz
    double atol;        // absolute Fehlertoleranz
    double stepsize;    // Anfangsschrittweite
    int maxorder;       // maximale Ordnung des BDF-Verfahrens
    int maxstepnumber;  // maximale Anzahl von Schritten
    double minstepsize; // minimale Schrittweite
    double maxstepsize; // maximale Schrittweite
    int printlevel;     // Outputlevel

    ...

    DAEynamics( void );           // Konstruktor
    virtual ~DAEynamics( void ); // Destruktor

```

```

...
virtual int integrate( double *pt, double tout, // Aufruf der Integration
    Matrix &x, Matrix &p, Matrix &q,
    int dflag, int startflag, int plotflag,
    Matrix &GA, Matrix &GAdir,
    Matrix &GQ, Matrix &GQdir,
    double *GAQ, int nGAQ1, int nGAQ2,
    double *rwh, int *iwh );
...
};

```

Verschiedene dynamische Modelle und Experimentbeschreibungen werden in den Arrays `expool` und `dynpool` in der Klasse `data` abgespeichert und flexibel durch Zeiger den Experimenten eines Mehrfachexperiment-Versuchsplans zugeordnet, siehe Abbildung 5.

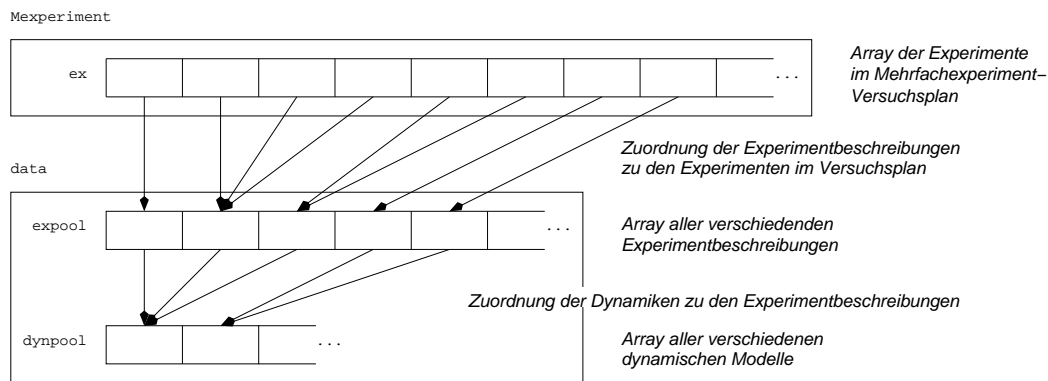


Abbildung 5: Datenstrukturen zur flexiblen Speicherung von Experimentbeschreibungen und dynamischen Modellen und deren Zuordnung zueinander und zu den Experimenten in einem Versuchsplan

3.4 Programmablauf

Der Programmablauf von `vplan98` umfaßt die folgenden Schritte:

1. *Programmstart:* Der Konstruktor des globalen Objekts `db` wird ausgeführt, dabei erfolgt die Initialisierung von Errorhandler und Signalhandler und die Belegung des benötigten Speichers.
2. *Initialisierung:* Das `vplan.ini`-File und die `experiment.ini`-Files werden eingelesen. Das zum Anwendungsbeispiel gehörende Problemmodul wird dynamisch geladen, problemabhängige Funktionspointer werden in die Datenstrukturen eingetragen. Gegebenenfalls werden Meßdaten eingelesen.
3. *Berechnungen:* Wahlweise erfolgt

- der Aufruf der Parameterschätzung,
 - der Aufruf der Integration,
 - der Aufruf der Simulationsumgebung oder
 - der Aufruf der Versuchsplanungsoptimierung.
4. *Ergebnisausgabe*: Die Output-Experiment-Files, das covmat-File und das Output-vplan.ini-File werden geschrieben.
 5. *Programmende*: Der Destruktor des globalen Objekts `db` wird ausgeführt. Er gibt den belegten Speicher wieder frei.

Im folgenden sind wichtige Routinen des Programms zusammengestellt.

- `main`: Hauptprogramm
- `lies_vplan`: Einleseroutine für `vplan.ini`-File
- `lies_experimete`: Einleseroutine für `experiment.ini`-Files
- `schreibe_vplan`: Schreibroutine für `vplan.ini`-File
- `schreibe_experimete`: Schreibroutine für `experiment.ini`-Files
- `create_constraints`: Abspeichern der linearen und nichtlinearen Nebenbedingungen
- `make_consistent`: Lösen der linearen Steuerbeschränkungen
- `ableitungstest`: Funktion zur Überprüfung der Ableitungen
- `constraint_check`: Test der Zulässigkeit der Nebenbedingungen
- `Integration`: Aufruf der Integration
- `Simulationsumgebung`: Aufruf der Simulationsumgebung
- `SQP_algorithm`: Schnittstelle zum Optimierungsverfahren
- `pardri`: Schnittstelle zu PARFIT
- `ADE_Guetekriterium`: Zielfunktion für Versuchsplanungsoptimierung
- `ADE_Guetekriterium_robust`: Zielfunktion für robuste Versuchsplanungsoptimierung
- `delta_Zielfunktion`: Zielfunktion für M-Kriterium
- `Guetekriterium_fuer_einen_Parametersatz`: Berechnung des Gütekriteriums für einen Parametersatz
- `confun`: Funktion zur Auswertung der nichtlinearen Nebenbedingungen

- `eval`: Funktion zur Auswertung eines Experiments: Integration des DAE-Systems, wahlweise Auswertung von Meßfunktionen und Nebenbedingungen, wahlweise mit Bereitstellung der benötigten ersten und zweiten Ableitungsinformationen, wahlweise Erzeugung von kontinuierlicher Ausgabe oder Pseudomeßdaten
- `calculate_measnode`: Auswertung eines Meßpunkts
- `calculate_constrnode`: Auswertung eines Nebenbedingungsgitterpunkts
- `daedrid`: Schnittstelle zu DAESOL
- `AOptKrit`: Berechnung des A-Kriteriums
- `deltaAOptKrit`: Berechnung der Ableitungen des A-Kriteriums
- `DOptKrit`: Berechnung des D-Kriteriums
- `deltaDOptKrit`: Berechnung der Ableitungen des D-Kriteriums
- `BeschrCovMat`: Berechnung der Kovarianzmatrix
- `deltaBeschrCovMat`: Berechnung der Ableitungen der Kovarianzmatrix
- `Nullraummethode`: Auswertung der Nullraummethode

3.5 Dynamisches Linken von Problemmodulen

Um eine hohe Flexibilität zu erzielen, ist der Programmcode von VPLAN problemunabhängig gehalten. Für die Anwendungsprobleme werden Problemmodule erzeugt, die zur Laufzeit von `vplan98` geladen werden. Auf alle problemspezifischen Subroutinen kann damit vom Programm in kompilierter Form zugegriffen werden. Der Benutzer braucht sich um diese Vorgänge nicht zu kümmern, alle notwendigen Schritte werden durch die Angaben in den ini-Files spezifiziert und vom Programm vollautomatisch durchgeführt. Bewirkt wird dies durch den Einsatz von Techniken des dynamischen Linkens, die wir nun genauer beschreiben wollen.

Die Problembeschreibung eines Anwendungsbeispiels ist im jeweiligen Problemverzeichnis abgelegt, siehe Abschnitt 2.3. Daraus wird vom Programm `doit` das Problemmodul erzeugt und in einem Unterverzeichnis von `target/modules` abgespeichert. Das Problemmodul besteht aus allen problemspezifischen Fortran-Subroutinen inklusive der von `doit` mit Hilfe von ADIFOR erzeugten Ableitungen und zusätzlich aus dem Headerfile `problem.hf` und dem Schnittstellenfile `problem.cc`. Dieses implementiert neben der Treiberoutine `problem_interface` drei Routinen zum Eintragen von Zeigern auf die Fortran-Funktionen in die Programmdatenstrukturen: `init_dyn_functions` für die Modellfunktionen für Differentialgleichungen und algebraische Gleichungen, `init_exp_functions` für Meßfunktionen, Sigmafunktionen, Steuerconstraints und dynamische Nebenbedingungen sowie `init_Mexp_functions` für die Plotfunktionen. Von `doit` werden alle diese Source-Files kompiliert und die Objekt-Files dann zu der dynamischen Bibliothek `libproblem.so` gelinkt.

Zur Laufzeit von vplan98 wird `libproblem.so` dynamisch geladen. Dies erfolgt, nachdem der Problemname und die Problemdimensionen eingelesen wurden. Dabei wird durch Aufruf der C-Bibliotheksfunktion `dlopen` das Modul geladen:

```
if ( ( db.handle = dlopen( problemlib, RTLD_NOW ) ) == NULL )
    Fehler("Problemmodul kann nicht geladen werden");
```

Durch `dlsym` wird auf die Treiberfunktion `problem_interface` zugegriffen:

```
if ( ( printerface = (PROBLEM_INTERFACE_TYPE)
      dlsym( db.handle, "problem_interface" ) ) == NULL )
    Fehler("Auf problem_interface kann nicht zugegriffen werden");
```

Der Aufruf von `problem_interface` liefert Zeiger auf die `init`-Funktionen:

```
init_Mexp_functions = (INIT_MEXP_FUNCTIONS_TYPE) printerface(INIT_MEXP);
init_exp_functions = (INIT_EXP_FUNCTIONS_TYPE) printerface(INIT_EXP);
init_dyn_functions = (INIT_DYN_FUNCTIONS_TYPE) printerface(INIT_DYN);
```

Schließlich werden durch Aufruf der `init`-Funktionen Zeiger auf die Modellfunktionen in die Datenstrukturen eingetragen:

```
for ( j = 0; j < db.Ndynpool; j++ )
{
    db.dynpool[j].init_settings();
    if ( ( errstr = init_dyn_functions( &db.dynpool[j] ) ) != NULL )
        Fehler( errstr );
}
```

```
for ( j = 0; j < db.Nexpool; j++ )
    if ( ( errstr = init_exp_functions( &db.expool[j] ) ) != NULL )
        Fehler( errstr );
```

```
if ( ( errstr = init_Mexp_functions( &db.V ) ) != NULL )
    Fehler( errstr );
```

Nun können über diese Zeiger die problemspezifischen Routinen aufgerufen werden.

Am Programmende wird mit `dlclose` das Problemmodul entladen:

```
dlclose( handle );
```

3.6 Weitere Komponenten der Software

Bei der Implementierung von VPLAN wurden weitere Techniken verwendet, die hier kurz erwähnt werden sollen:

- Die Schnittstellen zu SQP-Software, Parameterschätzprogramm, Integrator und Lineare-Algebra-Bibliotheken sind so modular gestaltet, daß es einfach ist, die verwendeten Bibliotheken durch Alternativen zu ersetzen.
- Ein ErrorHandler reagiert auf eventuelle Laufzeitfehler und sorgt für ein kontrolliertes Programmende.
- Ein Signalhandler ermöglicht das vorzeitige Abbrechen der Berechnung mit Zugriff auf die bis dahin erzielten Zwischenergebnisse.
- Zum komfortablen und effizienten Umgang mit Matrizen und Arrays wurden eine Matrixklasse und eine Arrayklasse implementiert, auf deren Objekte mittels Operator Overloading zugegriffen werden kann.

4 Werkzeug zur automatischen Ableitungserzeugung

Das Programm `doit` erzeugt vollautomatisch die für Simulation, Parameterschätzung und Versuchsplanung benötigten Ableitungen aller problemspezifischen Modellfunktionen eines Anwendungsbeispiels. Alle Fortran-Dateien, die gegeben und die erzeugten, werden gemeinsam mit der Treiberroutine zum Problemmodul `libproblem.so` zusammengelinkt.

Ein Programmlauf von `doit` führt im wesentlichen die folgenden Arbeiten durch:

1. Aus den ini-Files werden die benötigten Informationen über das Anwendungsbeispiel eingelesen.
2. Wenn noch nicht vorhanden, wird unter `target/modules` das Problemmodulverzeichnis angelegt.
3. Die gegebenenfalls beim letzten Aufruf von `doit` verwendeten und ins Problemmodulverzeichnis kopierten Fortran-Files werden mit den aktuellen verglichen. Nur bei Änderungen müssen die Ableitungen neu erzeugt werden.
4. Der C-Präprozessor wird auf die Fortran-Files angewendet.
5. Zur Parametrisierung der Steuerfunktionen wurden vom Benutzer DISCRETIZE-Kommandos in die Fortran-Files eingetragen. `doit` ersetzt diese durch den konkreten Sourcecode für den Zugriff auf die Parametrisierungsvariablen.
6. Für Modell-, Meß- und Constraintfunktionen werden die benötigten Ableitungen erzeugt. Dies geschieht durch Aufruf von ADIFOR. `doit` generiert alle von ADIFOR benötigten Input- und Treiberfiles. Da ADIFOR zweite Ableitungen nicht direkt

bereitstellen kann, generieren wir diese in einer zweistufigen Prozedur: Zunächst erzeugen wir die ersten Ableitungen und leiten diese nochmals mit ADIFOR ab, um die zweiten Ableitungen zu erhalten.

7. `doit` erzeugt außerdem die Files `problem.cc` und `problem.hf` und ein `makefile`.
8. Zuletzt ruft `doit make` zur Erzeugung von `libproblem.so` auf.

Zum Löschen des Problemmodulverzeichnis steht das Skript `undoit` zur Verfügung.

5 Grafische Benutzeroberflächen

5.1 minigui

`minigui` ist ein kleines, zu Demonstrationszwecken entwickeltes Frontend, implementiert in der objektorientierten Skriptsprache Python [14]. Es ist vor allem zur Vorführung der Simulationsumgebung gedacht. Mit Slidern können die Werte der Steuergrößen interaktiv verändert werden, die Auswirkungen werden auf Knopfdruck angezeigt. Abbildung 6 zeigt einen Screenshot.

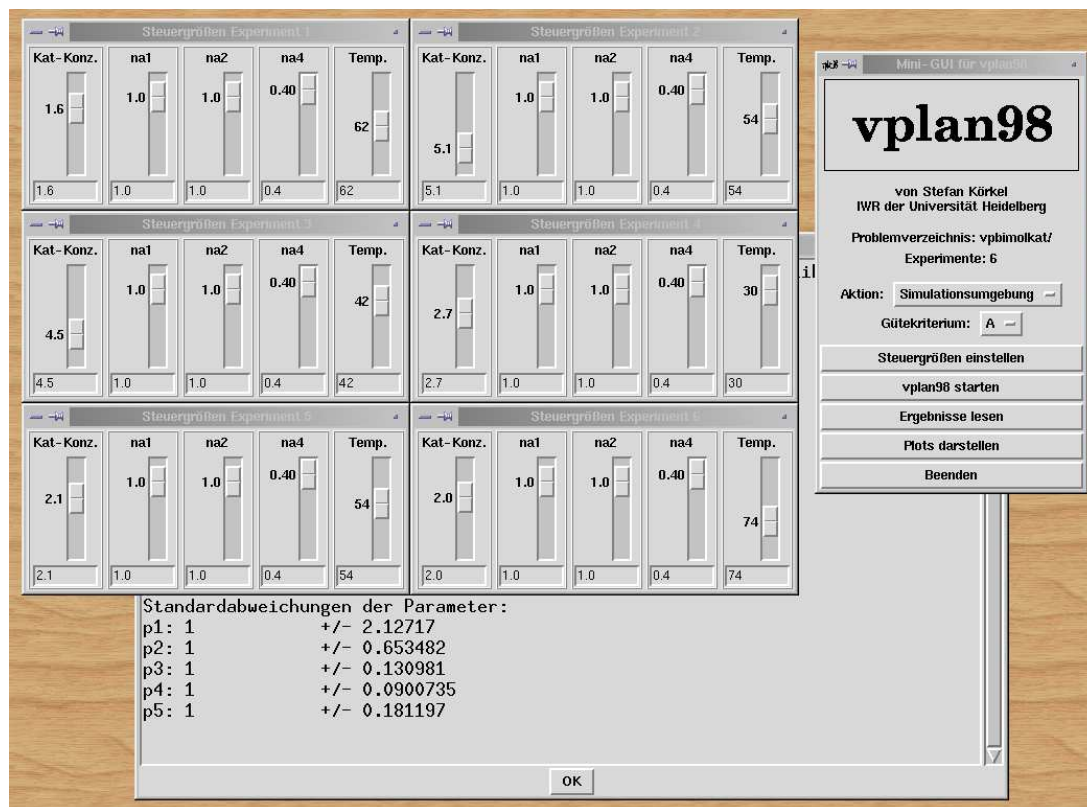


Abbildung 6: Screenshot der minigui-Oberfläche für VPLAN

5.2 MGAD

MGAD ist eine unter Betreuung des Autors im Rahmen der Diplomarbeit von Rucker [12] und mehrerer Softwarepraktikumsprojekte entwickelte Java-Oberfläche für VPLAN.

Sie wurde in einer Client-Server-Architektur realisiert.

Auf Client-Seite soll die Benutzeroberfläche plattformunabhängig und sehr flexibel einsetzbar sein. Deshalb wurde sie als Java-Applet implementiert. Damit kann sie aus jedem Java-fähigen, mit dem Internet verbundenen Browser heraus ausgeführt werden ohne weitere Softwareinstallation oder Systemvoraussetzungen.

Die Benutzeroberfläche ist ein grafisches Frontend, implementiert unter Benutzung der Java-Klassenbibliothek Swing [16]. Es ermöglicht die Spezifikation von Modellen für chemische Reaktionssysteme und die Beschreibung chemischer Prozesse. Aus diesen Eingabedaten erzeugt ein Modellgenerator die mathematische Problembeschreibung, also die rechten Seiten des DAE-Systems, in Form von Fortran- oder C-Source-Files und generiert mit automatischer Differentiation die Source-Files zur Berechnung der benötigten ersten und zweiten Ableitungen. In weiteren Eingabemasken werden die Problemdata und Werte der Variablen eingegeben. Dann können die Simulations- und Optimierungsmethoden von VPLAN aufgerufen werden.

Die Daten werden nun serialisiert und über das Internet zum Web-Server transferiert, der hier als Application-Server dient. Dort nimmt ein Java-Servlet die Anfragen entgegen, legt ein Problemverzeichnis und darin die problemspezifischen ini- und Fortran-Files an und ruft dort zur Erzeugung des Problemmoduls und vplan98 zur numerischen Berechnung auf. Der Client kann per Statusanfrage feststellen, ob die Berechnung beendet ist (Polling), und dann die Ergebnisdaten abholen. Zu deren Visualisierung stellt die Benutzeroberfläche Werkzeuge bereit.

Den Ablauf der Kommunikation zwischen Client und Server veranschaulicht Abbildung 7.

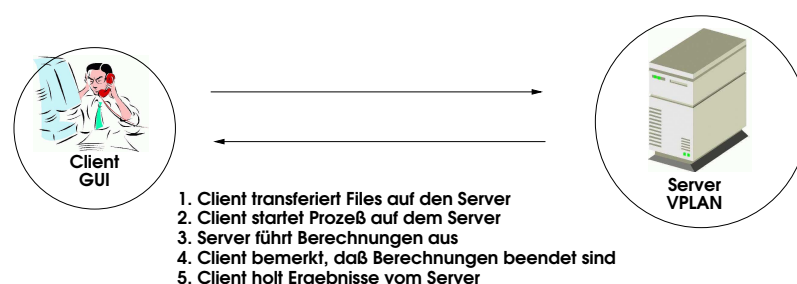
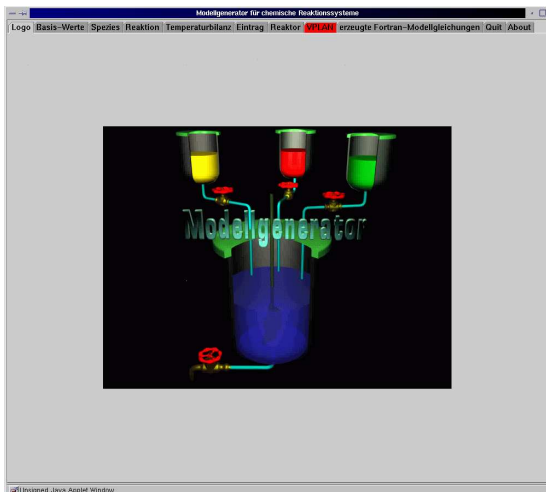
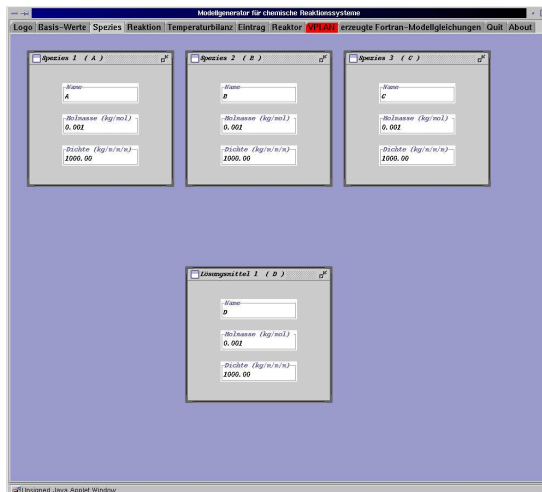


Abbildung 7: Kommunikation zwischen Client und Server

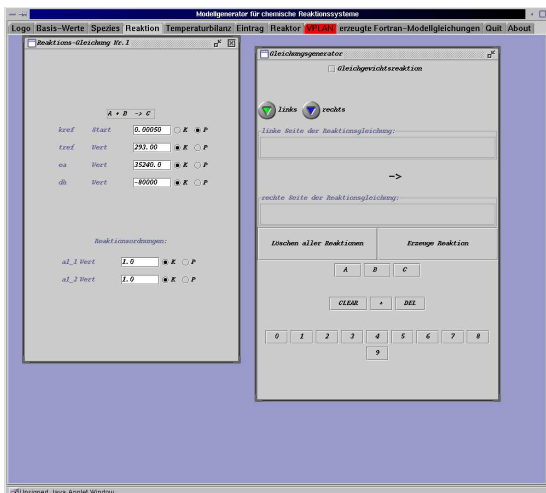
Abbildung 8 zeigt Screenshots der grafischen Benutzeroberfläche von MGAD.



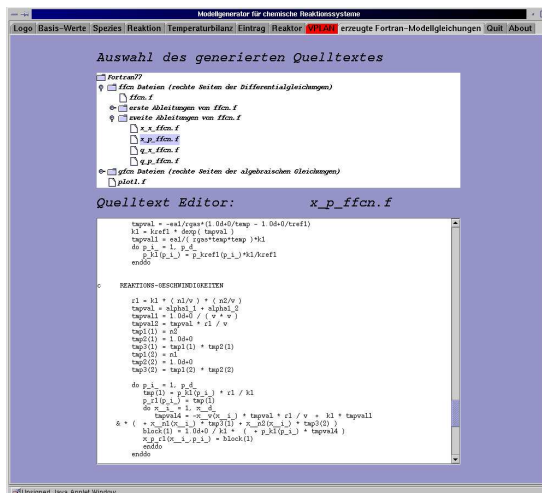
Titelseite



Spezifikation der chemischen Spezies



Eingabe der chemischen Reaktionen



Anzeige der erzeugten Source-Files

Abbildung 8: Screenshots der MGAD-Oberfläche für VPLAN

Literatur

- [1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide Third Edition*. SIAM, 1999.
- [2] I. Bauer, H. G. Bock, and J. P. Schlöder. DAESOL — a BDF code for the numerical solution of differential-algebraic equations. Preprint, IWR der Universität Heidelberg, SFB 359, November 1999.
- [3] C. Bischof, A. Carle, G. Corliss, A. Griewank, and P. Hovland. ADIFOR - generating derivative codes from fortran programs. *Scientific Computing*, 1(1):1–29, 1992.
- [4] C. Bischof, A. Carle, P. Hovland, P. Khademi, and A. Mauer. ADIFOR 2.0 user's guide (revision D),. Technical Report CRPC-TR95516-S, Center for Research on Parallel Computation, Rice University, Houston, TX, 1995, revised 1998.
- [5] C. Bischof, A. Carle, P. Khademi, and A. Mauer. The ADIFOR 2.0 system for the automatic differentiation of fortran 77 programs. Technical Report CRPC-TR94491, Center for Research on Parallel Computation, Rice University, Houston, TX, 1994.
- [6] P. Cederqvist et al. *Version Management with CVS*. Signum Support AB, available online: <http://www.cvshome.org/docs/manual/>, 1993.
- [7] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. Report NA 97-2, Dept. of Mathematics, University of California, San Diego, 1997.
- [8] P. E. Gill, W. Murray, and M. A. Saunders. *User's Guide For SNOPT 5.3: A FORTRAN Package for Large-Scale Nonlinear Programming*, December 1998.
- [9] S. Körkel. *Numerische Methoden für Optimale Versuchsplanungsprobleme bei nichtlinearen DAE-Modellen*. PhD thesis, Universität Heidelberg, available at <http://www.ub.uni-heidelberg.de/archiv/2980>, 2002.
- [10] The Math Works, Inc. *Using Matlab*, 1998.
- [11] The Numerical Algorithms Group Limited. *NAG Fortran Library Manual*.
- [12] G. Rücker. Automatisches Differenzieren mit Anwendung in der Optimierung bei chemischen Reaktionssystemen. Diplomarbeit, Universität Heidelberg, Dezember 1999.
- [13] Bjarne Stroustrup. *The C++ Programming Language (3rd Edition)*. Addison-Wesley, 1997.
- [14] M. von Löwis and N. Fischbeck. *Das Python-Buch*. Addison-Wesley, 1997.
- [15] M. von Schwerin. *Numerische Methoden zur Schätzung von Reaktionsgeschwindigkeiten bei der katalytischen Methankonversion und Optimierung von Essigsäure- und Methanprozessen*. Dissertation, Universität Heidelberg, 1998.

- [16] K. Walrath and M. Campione. *The JFC Swing Tutorial: A Guide to Constructing GUIs*. Addison-Wesley, 1999.
- [17] T. Williams and C. Kelley. *gnuplot — An Interactive Plotting Program*. Online Documentation <http://www.ucc.ie/gnuplot/gnuplot.html>, 1998.