SIMUL_R - EINE SIMULATIONSSPRACHE MIT SPEZIELLEN BEFEHLEN ZUR MODELLDARSTELLUNG UND -ANALYSE

Ronald Ruzicka
Technische Universität Wien

Wiedner Hauptstraße 6-10
A-1130 Wien

SIMUL_R ist eine Compiler-orientierte Simulationssprache für kontinuierliche Systeme. Sie bietet unter anderem Möglichkeiten zur Verwendung mehrerer Modelle in einem Programm, discrete-events, Tabellenfunktionen, automatisches Lösen von algebraischen Schleifen und das Abspeichern und Wiederladen des gesamten Systemzustandes oder Teilen davon. Der Runtime-Interpreter besitzt Möglichkeiten zur dynamischen Deklaration von Tabellenfunktionen und Matrizen, diverse Meta-Kommandos, Makros und Unterprogramme (auch rekursiv). Die SIMUL_R-Grafik-Library umfaßt unter anderem 3D- und bewegte Bilder. Das Ende dieser Arbeit nehmen kurze Beschreibungen der Simulations-Preprozessoren BAPS für Bondgraphen und CAPS für Compartements ein.

1. MOTIVATION

Die meisten heutzutage analysierten Systeme besitzen oft sehr komplexe Strukturen, sind sehr rechen-intensiv und fordern ihrem Analysator all seine Raffinessen ab - doch fehlen diesem oft die nötigen Werkzeuge. Dies führt zu erhöhtem Zeitverbrauch und schließlich zur oft beobachteten Kostenexplosion. Moderne Systeme erfordern deshalb Modularisierung, bequeme Analyse- und vielfältige (auch graphische) Darstellungsmethoden für Resultate.

Bei vielen komplexen Systemen ist schon die Modellierung alleine ein großes Problem - man kann nicht gleich *das* richtige Modell finden (falls ein solches überhaupt existiert) und muß mehrere Ansätze testen. Dies wurde bislang in den meisten Fällen durch Übereinanderlegen von Plots und Vergleichen von Zahlenkolonnen durchgeführt.

Andererseits muß man größere Modelle modularisieren, um deren Struktur und eventuell vorhandene Fehler besser und schneller erkennen zu können. ***Warum*** sollte man also nicht mehrere Modelle in einem Simulationsprogramm - nebeneinander und hintereinander - verwenden?

Sehr oft treten Teilmodelle mit impliziter Struktur auf. Diese müssen häufig, soweit möglich, von Hand aus aufgelöst und - mit speziellen numerischen Verfahren umgeben - "ausprogrammiert" werden. ***Warum*** sollte man diesen Vorgang nicht der Simulationssprache überlassen und das System einfach in impliziter Form anschreiben?

Das vierte Argument für die Entwicklung einer neuen Simulationssprache ist die Manigfaltigkeit der heutigen Probleme: man benötigt in diversen Anwendungsgebieten die unterschiedlichsten Methoden - und daraus entstehen oft "handgestrickte" Simulationsumgebungen, die spezifischen Erfordernissen der einzelnen Anwender Rechnung tragen. Die für alle Gebiete benötigten gleichen Grund-Mechanismen der Simulation werden jedoch vernachläßigt. Deshalb benötigt man ein ***offenes*** Simulationssystem, daß dem Anwender die Möglichkeit des Hinzufügens neuer Befehle und Algorithmen gibt.

Dies kann in SIMUL_R einfach durch Entwerfen neuer Unterprogramme, die in der SIMUL_R-Bibliothek gegebene Funktionen zur Analyse einer Kommando-Zeile verwenden, geschehen. Diese Unterprogramme können unter Verwendung der Programmiersprache C (diese ist auch die "Host"-Sprache für SIMUL_R; so wie FORTRAN für andere Simulationssysteme) oder auch FORTRAN oder anderer Hoch-Sprachen programmiert werden. C als Host-Sprache bewährt sich in Hinblick auf die Verwendung in modernen Betriebssystemen, wie z.B. UNIX.

2. DER SIMUL_R-COMPILER

Der SIMUL_R-Compiler übersetzt den Programmtext in ein C-Programm, welches danach mit der SIMUL_R-Runtime-Library zusammengebunden wird.

Wie es der CSSL-Standard vorschreibt, ist ein Teilmodell eines SIMUL_R-Programms aus drei Teilen aufgebaut: Anweisungen, die zu Beginn jedes Simulationslaufes durchgeführt werden sollen; die System-definierenden Gleichungen; und Anweisungen, die am Ende jedes Simulationslaufes ausgeführt werden.

Differentialgleichungen - auch in Vektor- und Matrizenform (bis zu 16 Dimensionen) - können genauso definiert werden wie Tabellenfunktionen (mit konstanter oder linearer Interpolation zwischen den Stützstellen) oder diskrete Ereignisse. Letztere können als einmal, periodisch oder zum Eintritt eines bestimmten Ereignisses durchzuführend definiert werden (Zeit-Scheduling ist auch zur Laufzeit möglich).

2.1 Algebraische Schleifen

Wie oben erwähnt, unterstützt SIMUL_R auch die automatische Lösung von impliziten Systemen. Betrachtet man z.B. die Gleichungen (in dieser Form stehen sie auch im SIMUL_R-Programm)

x = x*x + t und y = y*y + x ,

die über x und y für t [1;2] gelöst werden sollen, so kann man den Compiler mit dem Flag "-a" aufrufen und im Runtime-Interpreter die Simulation über [1;2] ablaufen lassen - fertig!

2.2 Mehrere Modelle in einem Programm

Ein SIMUL_R-Programm kann mehr als ein Modell enthalten (mehrere Modelle können auch gemeinsame Variablen besitzen). Sei z.B. ein nichtlineares und ein lineares Modell ein und desselben Systems vorhanden, so können beide in ein SIMUL_R-Programm geschrieben werden. Zur Laufzeit kann man das zu simulierende Modell anwählen (z.B. zuerst das nichtlineare, dann das lineare) und Modellvergleich betreiben.

Man kann auch mehrere Modelle "gleichzeitig" ablaufen lassen, wobei jedes Teilmodell einen Modul der oben geforderten Modularisierung darstellt.

2.3 Meta-Befehle

Der SIMUL_R-Programmtext kann auch Metabefehle enthalten: z.B. "while", "until", "for", "if", macros, include-files. Deren Verwendung erleichtert die Erzeugung von gut les- und überschaubaren Programmen, auch und gerade bei komplexen Systemen.

3. DER SIMUL_R-RUNTIME-INTERPRETER

Wie allgemein bekannt, hängt die Verwendbarkeit einer Compiler-orientierten Simualtionssprache primär von der "Rechen-Stärke" ihres Runtime-Interpreters ab.

Der SIMUL_R-Runtime-Interpreter unterstützt alle üblichen Kommandos, wie "start", "continue",

Verändern und Anzeigen von Variablen und Konstanten (unter Verwendung beliebiger algebraischer Ausdrücke inklusive Funktionen, wie sin, cos, log, exp, ...).

3.1 Schleifen- und Metabefehle

Der Interpreter bietet die gleichen Metabefehle wie der Compiler an. Dadurch können Schleifen und bedingte Anweisungen, Makros und Inkludieren von Files (letzteres kann als das Starten von Batchfiles oder als das Aufrufen von Unterprogrammen gesehen werden!) zur Laufzeit dynamisch programmiert werden. Alle Befehle können verschachtelt und rekursiv verwendet werden. Man kann solche Unterprogramme auch vom Runtime-Interpreter aus mittels Editor neu schreiben und ausbessern.

Des weiteren gibt es einige nützliche Befehle zur Behandlung von Tabellenfunktionen: die gegenseitige Zuweisung von Tabellen und den "assign"-Befehl. Mittels ***assign*** wird einer Tabellenfunktion eine Variable zugeordnet. Während der nächsten Simulationsläufe wird der Verlauf der Variablen in der Tabelle gespeichert und kann dann z.B. in anderen Modellen verwendet werden.

Im übrigen können Tabellenfunktionen und Matrizen zur Laufzeit neu angelegt (also definiert) und wieder gelöscht werden.

Die Meta- und die Tabellenfunktions-Befehle bieten starke Werkzeuge für komplexe Aufgaben, wie Integral-Gleichungen, Totzeit-Probleme, Optimierung von Steuerfunktionen.

3.2 Optimierung, Nullstellensuche

SIMUL_R ist ein offenes System. Das bedeutet, daß der Benutzer seine eigenen Runtime-Interpreter-Kommandos entwerfen kann (und zwar nicht nur in Form von Makros oder Batch-Files, sondern als echte Befehle). Einige solcher "neuen Kommandos" sind vordefiniert, wie z.B. Optimierung (mit Parameter-Beschränkungen), Nullstellensuche, Berechnung der Jacobi-Matrix und Ermittlung von Eigenwerten und -vektoren.

Ein einfaches Beispiel kann das Zusammenspiel all dieser Möglichkeiten demonstrieren: man betrachte das oben erwähnte lineare und nichtlineare Modell. Man startet nun mit dem nichtlinearen und rechnet bis zu einem bestimmten Punkt, ermittelt hier die Jacobi-Matrix und verwendet dieselbe im linearen Modell weiter - ohne Verlassen der Simulation und ohne zusätzlichen Programmieraufwand.

SIMUL_R ist auch insoweit ein offenes System, als durch einfaches Angeben des Namens und durch Dazubinden neue Integrations- und Nullstellenalgorithmen verwendet werden können.

Kommandos zum Abspeichern und Lesen von Tabellenfunktionen, Matrizen oder des gesamten Systemzustandes runden die Befehlspalette ab.

4. SIMUL_R-GRAPHIK

Die Grafik-Bibliothek umfaßt nicht nur Funktionen für "normale" Plots mit Achsen-Beschriftungen (in beliebigem Winkel und mit veränderbaren Skalen und Abständen). Es werden auch Möglichkeiten zum Zeichnen von 3D- und bewegten Bildern, von Schichtenlinien und von Tabellenfunktionswerten geboten. Weiters sind Drucker- und Plotter-Treiber zur Ausgabe der Grafiken vorhanden.

Eine Besonderheit sind die sogenannten "cross"- oder "Diagonal"-Plots, bei welchen nicht der Wert einer Variablen über einer anderen, sondern die Werte mehrere Variabler zugleich nebeneinander angezeigt und mit Linien verbunden werden. Man denke z.B. an die Diskretisierung einer partiellen Differentialgleichung für die Wellenbewegung einer schwingenden Saite. Dieses Modell enthält Variablen, die an bestimmten Stellen die momentane Auslenkung der Saite von einer Null-Linie darstellen. Verwendet man nun die genannte Zeichenart, so erhält man eine Vorstellung von der Bewegung der Saite.

Diese Vorstellung kann man noch verbessern, indem man die sich ergebenden Linien nicht auf einmal sondern hintereinander zeichnet - ein Simulations-Film! Eine andere Art der Bewegungsdarstellung ist die in untenstehendem Beispiel (Fadenpendel) anwendbare "Ein-Linien"-Bewegung, bei der von einer Kurve jeweils Punkt und Tangente dargestellt werden.

4.1 Ein SIMUL_R-Beispiel: Fadenpendel mit (Überschlag)

Dieses Beispiel zeigt das Umschalten zwischen Modellen zur Laufzeit. Ein Fadenpendel kann als 2-Modell-System gesehen werden: ein zirkulares und eines mit freiem Fall. Abb. 1 zeigt eine Möglichkeit der Laufzeit-Kommandos, Abb. 2 den Plot y über x.

```
act_mod=pendel_circular;
start;
#while t<tend #
   act_mod = pendel_fall;
   t0=t; x0=x;   y0=y;
        vx0=vx; vy0=vy;
   start;
   act_mod = pendel_circular;
   t0=t; phi0=phi; vphi0=vphi;
   start;
#end
```

Abb. 1 Laufzeit-Kommandos

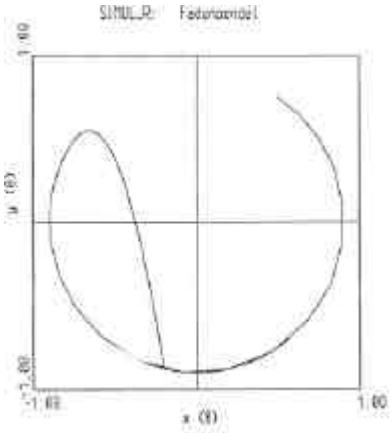

Abb. 2 Fadenpendel

5. PREPROZESSOREN: BAPS, CAPS

Sehr oft sind Systeme nicht explizit in Gleichungsform, sondern implizit in Diagrammen dargestellt. Es ist dann Zeit-aufwendig die Gleichungen abzuleiten, um sie simulieren zu können. Aus diesem Grund wurden Preprozessoren entwickelt, die es gestatten, diese Diagramme in Computer-lesbare Notationen zu bringen, und so dem Anwender, also dem Fach-Wissenschafter und dem Ingenieur, ermöglichen, "seine" gewohnten Schreibweisen zu verwenden.

Hier sollen zwei Preprozessoren zur Bondgraph- und zur Compartement-Verarbeitung vorgestellt werden. Beide sind als Preprozessoren für SIMUL_R, das Simulationssystem HYBSYS und für ACSL vorhanden bzw. geplant.

5.1 BAPS - Bondgraph Analyse und Programm Synthese

BAPS ist ein Preprozessor für Bondgraphen. Diese stellen eine universelle Beschreibungsmethode für elektrotechnische, mechanische, thermodynamische und interdisziplinäre Systeme dar /KARN75/. BAPS übersetzt einen Bondgraphen, beschrieben in einer leicht erlernbaren Eingabe-

Sprache, in z.B. ein SIMUL_R- oder ein HYBSYS-Programm (im übrigen ist es aufgrund der allgemeinen Konzeption von BAPS leicht möglich, in andere Simulationssprachen zu übersetzen).

BAPS verarbeitet auch nichtlineare Bondgraphen /RUZI87/. Diese Nichtlinearitäten können verschiedener Natur sein:

Elementkonstanten (z.B. OHMsche Widerstände oder Massen) können beliebige algebraische Ausdrücke sein; jedes Element kann eine nichtlineare, konstituierende Gleichung aufweisen; logische Schalter können definiert und in Ausdrücken verwendet werden. Die Schalter können auch verwendet werden, um sogenannte ***tdj-junctions*** aufzubauen; dies sind Element-Verbindungen, die sich zur Laufzeit ändern (d.h. man kann die Topologie eines Systems dynamisch verändern!). Ebenso können Tabellenfunktionen verwendet werden.

BAPS analysiert einen Bondgraph und sucht nach algebraischen Schleifen (auch bei den Nichtlinearitäten). Auf Wunsch kann eine Liste der Elemente und Verbindungen, sowie der zugewiesenen Kausalitäten ausgegeben werden.

BAPS unterstützt die Modell-Entwicklung mit der Möglichkeit der Verwendung mehrerer Teilmodelle und noch nicht näher bekannter Abschnitte des Bondgraphen (von letzteren müssen nur die benötigten Eingänge, die erzeugten Ausgänge und die ungefähren funktionellen Abhängigkeiten bekannt sein). Der BAPS-Text kann auch die oben erwähnten Meta-Befehle zur besseren Strukturierung enthalten (Anwendung z.B. bei Diskretisierungen).

5.2 Ein BAPS-Beispiel: Schwingkreis

Als Beispiel sei hier ein einfacher Schwingkreis, bestehend aus einer Spannungsquelle, einem Kondensator und einem Abnehmer (Widerstand), gewählt. Abb. 3 zeigt den Schaltplan, Abb. 4 den entsprechenden Bondgraphen und Abb. 5 das BAPS-Programm.

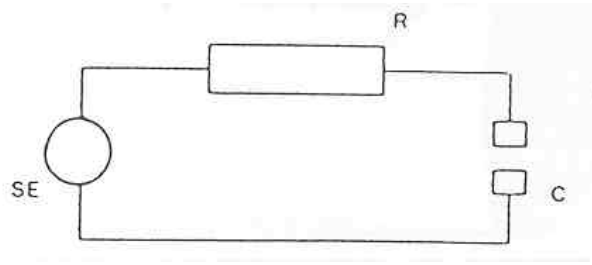

Abb. 3 RC-Kreis

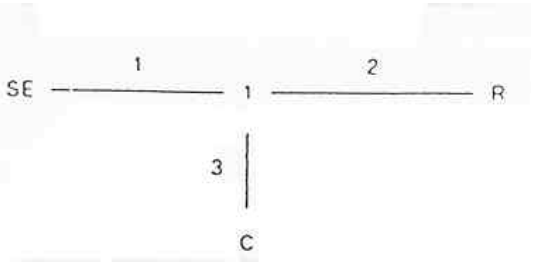

Abb. 4 Bondgraph-Beschreibung

```
RC_Kreis:

SE 1; 1 1 2 3; R 2; C 3;   {

CONSTANT TEND=0.05, f = 50*2*3.1416, ampl=220;
EXTERN SIN(1), t;
TERMINATE t>=TEND;

SE 1: SE1 = ampl*sin(t*f);
R  2: R2  = 100;
C  3: C3  = 0.0001;          q03 = 0;          "Anfangswert für Ladung"        }
```

Abb. 5 BAPS-Programm

5.3 CAPS - Compartement Analyse und Programm Synthese

CAPS ist ein Preprozessor für Compartements /SAUB87/. Es wandelt eine Compartement-Modellbeschreibung in den Sourcetext einer Simulationssprache (z.B. ACSL) um. CAPS verarbeitet nichtlineare und lineare Modelle und bietet Möglichkeiten der Optimierung und der statistischen Analyse. Eine weitere Besonderheit von CAPS ist die automatische Umwandlung von Compartement- in Volterra-Systeme.

Compartements werden zur Beschreibung biologischer Systeme, chemischer Prozesse, in Pharmakokinetik und -kinematik, genauso wie zur Modellierung von Epidemien und Wachstums-Vorgängen verwendet.

Ein CAPS-Programm kann Konstanten-Definitionen, Tabellen, Compartements, nichtlineare Blöcke, Infusionen und Raten enthalten. Ebenso können Makros und Meta-sprachliche Befehle verwendet werden.

5.4. Ein CAPS-Beispiel: Tollwut bei Füchsen

Das untenstehende Modell der Ausbreitung von Tollwut bei Füchsen kann als Netzwerk von Knoten verstanden werden, wobei jeder Knoten eine Region in einem Land darstellt. Jeder Knoten (Abb. 6) enthält ein Compartement für gesunde (x), kranke (y) und tote (z) Füchse, mit gewissen Übergangsraten. Abb. 7 zeigt einen Makro für einen Knoten. Das Wachstum der Population kann hier auf zwei Arten beschrieben werden: kontinuierlich im Jahr (dann wird inc1.cap inkludiert) oder einmal (z.B. im Frühjahr, inkludieren von inc2.cap) /TIMI84/.

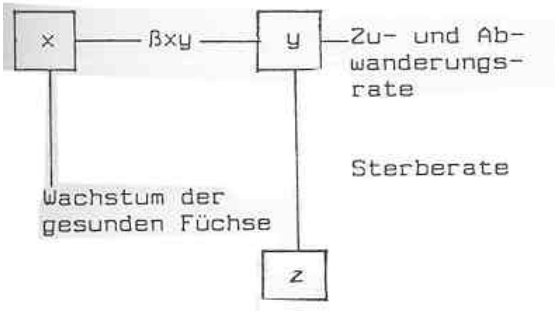

Abb. 6 Compartement-System für Tollwut bei Füchsen

```
MACRO knod(x,y,xicw,yicw)
CONST xic = xicw, yic = yicw;
COMP x(xic), y(yic);
BLOCK inf_x;
inf_x = beta*x*y;
RATE xout, yin, yout;
xout = x, inf_x, 1;
yin  = inf_x, y, 1;
yout = y, EMPTY, gamma;
IF inc1 INSERT <inc1.cap>;
IF inc2 INSERT <inc2.cap>;
END;
```

Abb. 7 CAPS-Programm

6. LITERATUR

/KARN75/ D. Karnopp, R. Rosenberg, ***System Dynamics: A Unified Approach***, Willey-Interscience, New York, 1975, 402 Seiten

/RUZI87/R. Ruzicka, ***Eine Methode zur theoretischen und praktischen Darstellung und Analyse von Bondgraphen unter besonderer Beachtung von Nichtlinearitäten***, Dissertation TU Wien, 1987, 273 Seiten

/SAUB87/A. Sauberer, ***Darstellung, Analyse und Simulation von Kompartmentsystemen unter Berücksichtigung von Nichtlinearitäten***, Dissertation TU Wien, 1987, 320 Seiten

/TIMI84/W. Timischl, "Influence of Landscape on the Spread of an Infection", ***Bull. of Mathematical Biology***, Vol. 46, No. 5/6, Pergamon Press, 1984, pp 869-877