EIGNUNG DER SIMULATIONSSPRACHE SLAM-II ZUR MODELLIERUNG UND SIMULATION GROSSER TRANSPORTSYSTEME

André Graber und François E. Cellier, Zürich

Zusammenfassung. Seit 1980 ist das netzwerkorientierte Simulationspaket SLAM erhältlich. Es eignet sich besonders für die diskrete Simulation, ist aber auch für gemischt kontinuierliche und diskrete Systeme geeignet. SLAM wurde seither von vielen Forschungsgruppen für eine Vielzahl verschiedenartiger Simulationsprobleme erfolgreich verwendet. SLAM-II ist die momentan erhältliche Version dieser Software. Dieses Simulationssystem wurde von der Firma Pritsker & Associates, Inc. Indiana USA entwickelt und stellt zweifellos einen Meilenstein in der Entwicklung topologischer Simulationssoftware dar. Diese Studie behandelt die Anwendung von SLAM auf ein sehr grosses Industrieprojekt, die Simulation des internen Transports in einem Lagerhaus. Das Ziel der Arbeit ist die kritische Analyse der Eignung von SLAM für diesen Typus Problem, dessen Vorzüge zu zeigen, sowie auch dessen Beschränkungen aufzudecken. Damit hoffen wir, künftigen Anwendern einige Kriterien für die Softwareevaluation zur Verfügung zu stellen.

Summary. Since 1980, the new network-oriented simulation language SLAM, primarily useful for discrete system simulation but also applicable to combined continuous and discrete system simulation, is available, and since then has been successfully applied by many Research Groups to a large variety of simulation problems. SLAM-II is the currently available version of this software. This simulation system which was developed by Pritsker & Associates, Inc. certainly is a mile stone in topological simulation software development. This paper discusses a very large industrial application of SLAM, namely the simulation of the internal transport in a store house. The aim of this paper is to critically analyse the aptitude of SLAM for the modeling of such types of systems, to show its advantages, and to outline its limitations. By these means, we hope to provide future potential users of SLAM with a decision aid for software selection.

1. Einführung

Dieser Bericht beschreibt die Modellierung des internen Transportsystems eines Lagerhauses, wie es von Digitron AG, einer Schweizer Firma, welche Transportroboter und die dazugehörigen Transportsysteme herstellt, projektiert wurde. Der interne Warenfluss zwischen 8 vollautomatischen Hochregallagern, 6 bemannten Verpackungsstationen, 4 Ausgangs- und einer Eingangsstation wird von ca. 12 Robotern bewältigt, welche einem starren Streckennetz folgen. Dieses Problem wurde mit SLAM-II simuliert.

Die wichtigsten Ziele dieser Simulation waren, zu bestimmen :

1) wie viele Roboter benötigt werden, um einen gegebenen Warenfluss zu bewältigen,
2) welches die optimale Zuordnung der Roboter an die Arbeitsaufträge ist, und
3) wie die Roboter optimal durch das System geleitet werden.

Das Modell kann als ein System mit vier hierarchischen Stufen betrachtet werden (hier in der Reihenfolge von der tiefsten zur höchsten beschrieben):

1) Bewegung der Roboter,
2) Mechanismus der Streckenblockreservation,
3) Zuteilung der Wegstrecken an Roboter,
4) Zuteilung der Roboter an Arbeitsaufträge.

Die unterste Stufe ist anwendungsabhängig und deshalb nicht von allgemeinem Interesse für den Simulationsspezialisten. Hingegen präsentieren sich die Stufen 2 bis 4 recht allgemein und werden darum ausführlicher behandelt. Ersetzt man in der bisherigen Beschreibung das Wort 'Roboter' durch das Wort 'Zug', so kann man unmittelbar sehen, dass die zweite Stufe der Hierarchie eins zu eins auf diese modifizierte Situation übertragen werden kann. Die Stufe 3 wäre zumindest auf grosse Bahnhöfe anwendbar. Hingegen wäre Stufe 4 nicht anwendbar, da diese während der Ausarbeitung des Fahrplanes und somit "off-line" stattfindet. Ersetzt man andererseits das Wort 'Roboter' durch 'Taxi', so ist die zweite Stufe der Hierarchie aus klar ersichtlichen Gründen nicht übertragbar, hingegen sind die Stufen 3 und 4 wieder eins zu eins anwendbar.

Es ist nicht unsere Absicht, obiges Projekt bis ins Detail zu behandeln, da dies für den Leser nicht von direktem Interesse ist. Hingegen wollen wir die Fähigkeiten und Beschränkungen von SLAM-II aufzeigen, die Erfahrungen beschreiben, die wir beim Modellieren dieses Grossprojektes gewonnen haben. Da diese diskutierte Problematik allen Transportsystemen zu eigen ist, hoffen wir, mit unserem Beitrag künftigen potentiellen Softwarebenützern eine Entscheidungshilfe bei der Frage anbieten zu können, ob sich SLAM-II für die Modellierung ihres eigenen Transportsystems eignet oder nicht.

2. Roboterbewegungen

Da die Roboter mit konstanter Geschwindigkeit herumfahren, war es am einfachsten (und am kostengünstigsten), die Roboterbewegungen diskret zu modellieren (im Gegensatz zu einer Lösung mit einem Satz von Differentialgleichungen). Ein Roboter, der einen Netzwerkknoten verlässt, beginnt eine Aktivität von bestimmter Dauer. Diese

setzt sich zusammen aus einer eventuellen Beschleunigungszeit, der Zeit um die Distanz bis kurz vor den Folgeknoten zurückzulegen, wo entschieden wird, ob der Roboter mit konstanter Geschwindigkeit durchfahren kann oder anhalten muss. Im letzten Fall kommt noch eine Verzögerungszeit dazu. Diese Aktivitätsdauer wurde in FORTRAN in der Funktion USERF (= USER Function) codiert, welche von SLAM-II an den entsprechenden Stellen aufgerufen wird.

3. Blockreservation

Nun wollen wir den einfachsten aller Fälle, den 'TRANSITSTOPP' betrachten. Dies ist ein Knotenpunkt des Streckennetzes ohne irgendwelche Auf- oder Abladeeinrichtung, also ein Punkt, durch den der Roboter nur durchfährt. Im Folgenden wird der vom Auftraggeber eingeführte Begriff 'Stopp' für Knotenpunkte des Transportstreckennetzes verwendet, wobei der Begriff 'Knoten' für die SLAM Netzwerkknoten verwendet wird. Diese Unterscheidung ist sinnvoll, da ein Stopp durch mehrere SLAM-Knoten dargestellt wird. In einem Transitstopp kann ein Roboter zum Anhalten gezwungen werden, falls die nächste zu befahrende Strecke momentan von einem anderen Roboter belegt ist. Fig. 1 zeigt einen solchen Transitstopp, der mit SLAM-II Netzwerkelementen modelliert wurde.

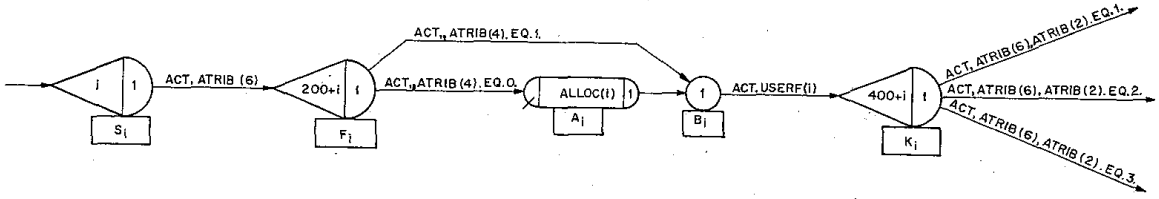

Fig. 1: SLAM Netzwerk für einen Transitstopp

Fig. 2 zeigt den eigentlichen Programmcode für den speziellen Transitstopp Nr. 17. Man beachte insbesondere, wie die Stoppnummer (17) im Code als Teil der Labelbezeichnungen und der Filenummern auftritt, was in der allgemein gehaltenen Zeichnung Fig. 1 durch 'i' dargestellt wurde.

Jeder durch das Netzwerk fahrende Roboter besitzt mehrere Attribute (ATRIB-Vektor), in welchen u.a. der Status (beschäftigt, unbeschäftigt) und der Zielstopp des Roboters gespeichert sind. Zum besseren Verständnis des Beispiels seien hier kurz die Attribute 6, 4 und 2 erklärt. ATRIB(6) enthält die Zeit, die benötigt wird, um das nächste Wegstück zurückzulegen. Mit ATRIB(4) kann festgestellt werden, ob die nächste Wegstrecke reserviert werden konnte (=1), oder ob der Roboter anhalten muss,

um auf die Reservation zu warten. ATRIB(2) enthält den sogenannten Richtungscode. Dieser Code bezeichnet längere Streckenabschnitte und dient bei Kreuzungen als Abzweigungskriterium.

Einige Transitstopps sind als sogenannte Dispositionsstopps ausgebildet, das sind Orte, in denen unbeschäftigten Robotern ein Arbeitsauftrag zugeteilt werden kann (Stufe 4). Zu jenem Zeitpunkt werden auch die Attribute neu gesetzt (Status, Arbeitsauftrag, Zielstopp u.a.).

```
;
;*****************************************************************
;                          STOPP NR 017                          *
;*****************************************************************
S017 EVENT,17,1;                          BETRITT STOPP 017
       ACT,ATRIB(6);                      FAEHRT ZUM STOPP
F017 EVENT,217,1;                         BEFREIT VORHERIGEN STOPP
       ACT,,ATRIB(4).EQ.1.,B017;          NAECHSTER BLOCK RESERVIERT
       ACT,,ATRIB(4).EQ.0.,A017;          NAECHSTER BLOCK BESETZT
A017 AWAIT(17),ALLOC(17),,1;              WARTET AUF NAECHSTEN BLOCK
B017 GOON,1;
       ACT,USERF(17);                     VERLAESST STOPP 017
K017 EVENT,417,1;                         GIBT STOPP 017 FREI
       ACT,ATRIB(6),ATRIB(2).EQ.1.,S018; GEHT ZU STOPP 018
       ACT,ATRIB(6),ATRIB(2).EQ.2.,S031; GEHT ZU STOPP 031
       ACT,ATRIB(6),ATRIB(2).EQ.3.,S063; GEHT ZU STOPP 063
```

Fig. 2: SLAM Netzwerk Programmsegment für den Transitstopp Nr. 17

Der Roboter erreicht den Stopp im EVENT-Knoten mit dem Label Si (In Fig. 1 für den Knoten 17 heisst dieses Label sinngemäss S017). Zu diesem Zeitpunkt ruft SLAM-II die Subroutine EVENT mit dem Ereigniscode i auf. Dieser EVENT-Knoten stellt das Interface vom Netzwerk mit den hierarchischen Stufen 1 und 2 zu den höheren Stufen 3 und 4 dar.

In der Subroutine EVENT wird die Zeit bestimmt, die der Roboter benötigt, um bis zum Zentrum des Stopps zu fahren, welche in ATRIB(6) gespeichert wird. Sie hängt davon ab, ob die nächste Wegstrecke reserviert werden konnte; wenn nicht, so bedeutet dies, dass der Roboter abbremsen muss.

Nach ATRIB(6) Zeiteinheiten trifft der Roboter im Zentrum des Stopps ein. Im SLAM-II Netzwerk ist dies der Knoten mit dem Label Fi. Nun ruft SLAM-II wieder EVENT auf, diesmal aber mit dem Ereigniscode (200+i). Alle vorgängigen Wegstrecken werden zurückgegeben (möglicherweise mit der Ausnahme des gerade vorherigen Stopps (i-1), der in einigen Fällen reserviert bleiben muss, da einige der Stopps zu nahe beieinander liegen, als dass auf beiden Stopps [(i-1) und i] gleichzeitig je ein Roboter parkiert sein könnte).

Obwohl SLAM-II ein Netzwerkelement FREE enthält, um Resourcen zurückzugeben, muss in unserer Anwendung von EVENT aus die Subroutine FREE aufgerufen werden, um die Wegstrecke zurückzugeben. Der Grund dafür ist, dass die Anzahl zurückzugebender Blöcke variieren kann, und dass das FREE Netzwerkelement nur auf den Namen des Blockes nicht jedoch auf dessen Nummer Bezug nehmen kann, wie dies von uns aus Modularitätsgründen gefordert wurde. Wir wollen nicht näher auf dieses Detail eingehen, als um zu zeigen, dass diese Unschönheit von SLAM-II kein Problem bietet, sondern auf einfache Art und Weise umgangen werden kann. Dies gilt für SLAM-II ganz allgemein.

Kann die folgende Wegstrecke nicht reserviert werden, so wird der Roboter unverzüglich (der Platz für die Aktivitätsdauer ist leer) in den AWAIT Knoten mit dem Label Ai transferiert. Der Roboter wartet in diesem Knoten bis alle benötigten Wegstrecken verfügbar sind. Dies geschieht durch wiederholtes Aufrufen der Subroutine ALLOC (allocate), und zwar bis alle Wegstrecken gleichzeitig verfügbar sind; erst dann werden sie reserviert. Der Aufruf von ALLOC ist in dem Fall unvermeidbar, wo mehrere Strecken gleichzeitig beansprucht werden. Dies kann nicht sequentiell erfolgen, da sonst Deadlocksituationen auftreten können.

Wurden einem Roboter seine benötigten Wegstrecken zugeteilt, so fährt er durch den GOON Knoten (dieser entspricht einem FORTRAN CONTINUE). Die Dauer der folgenden Aktivität wird in der Funktion USERF gerechnet und steht in dem Feld, wo bis anhin ATRIB(6) stand. Der EVENT Knoten mit dem Label Ki und dem Ereignisknoten (400+i) befindet sich 25 cm nach dem Zentrum des Stopps; es ist der Ort, wo der eben verlassene Stopp freigegeben wird.

Die Fahrzeit zum nächsten Knoten zu steht wieder in ATRIB(6), und der Richtungscode ATRIB(2) dient als Verzweigungskriterium. Der Roboter wird nun in den nächsten Stopp zum EVENT Knoten mit dem Label A(i+1) fahren, womit wir wieder am Anfang sind.

Dies ist kurz zusammengefasst die Funktionsbeschreibung des einfachen Transitstopps. Die Lade-, Entlade- und Hochregallagerstopps sind bei gleicher Struktur etwas komplexer. Sie enthalten zwischen den Knoten mit den Labels Fi und Ai einige zusätzliche Zweige.

Auch die bemannten Verpackungsstationen konnten problemlos durch SLAM-II Netzwerkelemente modelliert werden.

Das Netzwerk besteht aus ca. 120 Stopps, was ungefähr 3000 Zeilen Netzwerkbeschrei-

bung entspricht.

4. Zuteilung der Fahrtwege an die Roboter

Das Dispositionsproblem enthält viel sequentielle Logik, welche durch die SLAM-II Netzwerksprache nur schlecht ausgedrückt werden kann. Aus diesem Grunde wurde die Dispositionslogik in FORTRAN programmiert. Das Interface von dieser zum Netzwerk bilden die EVENT Knoten mit den Labels Si.

Der Richtungscode, den jeder Roboter als Attribut [ATRIB(2)] mit sich trägt, muss zumindest beim Zuordnen der Arbeitsaufträge neu gesetzt werden. Dies ergibt den kürzesten Weg zum Zielstopp. Wenn diese Wegstrecke aber sehr stark befahren wird, kann ein längerer Weg zeit-kürzer sein. Dazu muss dann aber in jedem Stopp der Richtungscode neu bestimmt werden. Dies geschieht durch Auslesen des Richtungscodes aus einem topologischen Datenfile, das die Netzwerkstruktur enthält, so dass zwischen zwei beliebigen Stopps der Richtungscode neu bestimmt werden kann.

Die Disposition ist zum gegenwärtigen Zeitpunkt nocht statisch, d.h. die Dichte des Verkehrs wird bei der Suche nach alternativen Wegen noch nicht berücksichtigt. Wohl aber ist jede dazu notwendige Information vorhanden. Das Programm ist ausserdem so modular aufgebaut, dass die jetzt verwendete Dispositionssubroutine einfach durch eine intelligentere ausgetauscht werden kann.

5. Zuordnung von Robotern an Arbeitsaufträge

Es gibt grundsätzlich zwei verschiedene Zuordnungsalgorithmen. Der eine besteht darin, dass ein Arbeitsauftrag einen unbeschäftigen Roboter sucht, während der andere die unbeschäftigten Roboter Arbeit suchen lässt. Der zweite Algorithmus wurde von uns verwendet, da dieser auch in den realen Anlagen zur Anwendung gelangt. Die untätigen Roboter fahren andauernd in einem zentralen Kreis um die Stopps, bei denen Arbeit anfallen kann, und fragen in den Dispositionsstopps nach Arbeit. Das Interface von der Arbeitszuweisung zum Netzwerk ist wiederum durch die EVENT Knoten gegeben. Da die Arbeitszuweisung auch in einer separaten (zum jetzigen Zeitpunkt noch sehr einfach gehaltenen) Subroutine erfolgt, kann diese wie im oben diskutierten Dispositionsproblem einfach durch eine komplexere Subroutine ersetzt werden.

6. Kritische Beurteilung von SLAM II

Array Begrenzungen: SLAM-II ist in FORTRAN programmiert. Daraus resultiert eine Begrenzung der Grösse von Systemparametern (z.B. Anzahl Zustandsvariabeln, Resourcen

u.a.). Da diese Begrenzungen in keinem der uns zur Verfügung stehenden Dokumente aufgeführt sind (wir mussten sie aus dem Programmlisting herauslesen!), haben wir uns entschlossen, die Grenzen in Tab. 1 aufzuführen.

SLAM-II Menge	Grösse
Anzahl Differentialgleichungen	100
Anzahl Zustandsbedingungen	25
Anzahl ENTRY Knoten	25
Anzahl Gates	25
Anzahl Recourcen	75
Anzahl Histogramme	50
Anzahl Zellen (in Histo.)	500
Anzahl COLCT Statistiken	50
Anzahl zeitintegrierter Stat.	50
Anzahl Attribute	100
Anzahl Files	100
Anzahl Plots	10
Anzahl abhängiger Var. pro Plot	10
Anzahl Zufallszahlengeneratoren	10
Anzahl Aktivitäten	100
Anzahl Knoten	500
Anzahl Serviceaktivitäten	50
Anzahl globaler Variabeln	100

Tab. 1: Grössenbegrenzungen von SLAM-II

Für unsere Anwendung mussten wir 7 der 18 Bereiche vergrössern (z.B. benötigten wir 750 Aktivitäten). Es war eine positive Ueberraschung zu sehen, dass diese Begrenzungen im SLAM Sourceprogramm nur in Form von Arraydimensionierungen auftreten, während im übrigen der Wert der Begrenzung in einer Variabeln abgespeichert ist, welche nur an einer Stelle im Programm ein Wert zugewiesen erhält (keine expliziten Referenzen in DO-loops, etc.). Auf diese Weise konnten die notwendigen Aenderungen in nur wenigen Stunden vorgenommen werden, und die vergrösserte Version von SLAM-II lief auf Anhieb.

Ausdruckskraft der Sprache: Die Simulation wurde von einem Studenten in den acht Wochen durchgeführt, die ihm für seine Diplomarbeit zur Verfügung standen. Die Tatsache, dass ein so grosses Projekt von einem Studenten, der keine Erfahrung mit SLAM hatte, in so kurzer Zeit erfolgreich bearbeitet werden konnte, spricht besser als irgend eine theoretische Analyse für die Leichtigkeit, mit der SLAM-II zur Modellierung verwendet werden kann. Wir begegneten keinerlei Problemen, die Komponenten unseres Transportsystems durch SLAM-II Elemente auszudrücken. Dies wäre bei Verwendung eines Softwaresystems wie Q-GERT oder GPSS, welches nicht über solch ein flexibles FORTRAN Interface verfügt, bestimmt nicht der Fall gewesen.

Fehlersuche: Es war zu erwarten, dass ein Programm dieser Grössenordnung zunächst einige Fehler aufweist. Netzwerkfehler waren dank dem gut verständlichen Tracing sehr einfach zu finden, und wurden ausnahmslos durch den Studenten selbst eruiert. Hingegen waren die FORTRAN Fehler sehr viel schwieriger zu beheben und bedurften einiger Nachteinsätze des betreuenden Assistenten (!). Einige Ueberraschungen boten die von FORTRAN-Teilen aus aufgerufenen Prozessinteraktionssubroutinen, die zuweilen schwer verständliche Einträge ins Tracing erzeugten.

Zuordnung von Resourcen: Die ursprüngliche Version von SLAM, wie in [4] beschrieben, erlaubt keine gleichzeitige Zuordnung von Resourcen in einem Knoten. Eine solche Möglichkeit ist aber essentiell, wenn Deadlocksituationen vermieden werden sollen. Nehmen wir an, dass zwei Resourcen mit Kapazität 1 gleichzeitig von zwei Transaktionen benötigt werden. Wenn jede Transaktion eine der Resourcen alloziert hat, dann werden beide Transaktionen für ewige Zeit auf die zweite Resource warten. In unserem Projekt trat dieser Fall dann auf, wenn aus verschiedenen Richtungen kommende Roboter gleichzeitig eine komplexe Kreuzung reservieren wollten, welche durch mehrere Resourcen modelliert wurde. Die einzige Lösung dieses Problems lag darin, mit einem einzigen, ununterbrechbaren Befehl mehrere Resourcen zu belegen. SLAM-II [3] beinhaltet diese Möglichkeit, indem von einem AWAIT Knoten aus die in FORTRAN zu schreibende Subroutine ALLOC aufgerufen werden kann, in welcher der Benützer spezifiziert, welche Resourcen gleichzeitig reserviert werden sollen. Dabei wird keine Resource reserviert, solange nicht alle benötigten Resourcen gleichzeitig verfügbar sind. Neben diesem in SLAM nun gelösten Problem der Resourcenzuteilung bleiben die "üblichen" Unschönheiten der Resourcenverwaltung, wie diese im begleitenden Bericht [5] diskutiert werden. So können Resourcen freigesetzt werden, die nie zuvor reserviert worden sind; auch können Transaktionen das System verlassen ohne ihre Resourcen zurückzugeben (eine nette Art, sterbende Millionäre zu simulieren, die ihr Geld auf einem Nummernkonto einer Schweizer Bank hinterlassen!).

Ausführungseffizienz: Es ist noch zu früh, die Effizienz von SLAM-II mit derjenigen anderer Sprachen zu vergleichen, da das Projekt noch nicht weit genug fortgeschritten ist. Eines kann aber jetzt schon gesagt werden: Die Kompilationszeit wächst enorm mit anwachsender Netzwerksgrösse. In unserem Beispiel mit ca. 3000 Zeilen Netzwerkbeschreibung wurden in einer CDC CYBER Maschine über 120 Sec. CPU Zeit zur Kompilation des Netzwerkes benötigt. Dies wäre akzeptabel, wenn es eine Möglichkeit gäbe, das kompilierte Netzwerk zu speichern, wie z.B.:

```
SAVE,8*
```

um die Netzwerkbeschreibung auf Band 8 zu speichern, und

LOAD,8*

um sie wieder von Band 8 zu lesen. Man bedenke, dass die Dispositionslogik in FORTRAN geschrieben ist. Es wäre somit durchaus sinnvoll, an ein und demselben Netzwerk verschiedene Dispositionsstrategien auszutesten. Leider ist diese Möglichkeit zur Zeit nicht gegeben.

Macro Möglichkeit: Im Gegensatz zu Q-GERT verfügt SLAM bis heute nicht über einen Macromechanismus. In grossen Netzwerken ist aber diese Möglichkeit von einiger Bedeutung, da Netzwerke sehr oft aus vielen gleichartigen, komplexen Elementen bestehen, wie in unserer Anwendung die Stopps. Eine solche Möglichkeit erlaubt, wenn sie sorgfältig ausgeführt ist (was z.B. in Q-GERT nicht der Fall ist, da Macros in jener Sprache keine formalen Parameter aufweisen können), eine modularere hierarchische Strukturierung des Models. In unserem Falle entschieden wir uns zur Entwicklung eines (relativ kurzen) PASCAL Programms, welches aus 4 Netzwerkelementen (wie dem diskutierten Transitstopp) sowie einem Datenfile, das die topologische Netzwerkstruktur enthält, die Netzwerkbeschreibung generiert. Dies schien uns weniger aufwendig als die Verwendung einer allgemeinen Macrosprache (wie z.B. ML/1 [1]), die oft eine schwerfällige Macrodefinition erfordert. (Eine solche Problemlösung wäre dennoch ebenfalls gangbar gewesen, wie z.B. in [2] diskutiert.)

Speicherplatzbedarf: Auch eine Macromöglichkeit würde dennoch nicht alle Probleme lösen. Zur Ausführung unseres Programms benötigten wir 240000 Speicherplätze (oktal) in unserer CYBER Maschine, womit wir beinahe an der oberen Limite des zur Verfügung stehenden Speichers arbeiten mussten. (ECS hätte verwendet werden können, hätte aber weitgehende Aenderungen der SLAM-II Software bedingt.) Das Betriebssystem unserer CYBER ist zweifellos veraltet, da es über keine virtuelle Speicherplatzzuteilung verfügt. (Weder auf unserer VAX noch auf der IBM 3033 wären wir auf solche Beschränkungen gestossen.) Um eine gute Ausführungseffizienz zu erhalten, könnte es aber auch dort sinnvoll sein, den Speicherplatzbedarf zu reduzieren. Dies könnte u.a. dadurch geschehen, dass man einen Roboter immer wieder durch denselben Stopp fahren liesse. Allerdings müsste man dazu noch grössere Teile des Netzwerkes in FORTRAN programmieren, und könnte im Extremfall die Netzwerkbeschreibung ganz weglassen. (Man beachte, dass schon in der gegenwärtigen Lösung die meisten Netzwerkelemente Schnittstellen zu FORTRAN Subroutinen sind.) Dies bedeutet aber nicht, dass dadurch die Netzwerkbeschreibung unnütz würde. Auch in diesem Fall hätte uns SLAM-II durch seine Netzwerkstruktur geholfen, die Software zu strukturieren. Diese Unterstützung bei der Modellierung darf nicht unterschätzt werden. Wir sind überzeugt, dass der Student ohne das Hilfmittel SLAM-II keine Chance gehabt hätte, in der kurzen Zeit von 8 Wochen auch nur irgenwohin zu gelangen.

Hierarchische Strukturen: Wie in Fig. 1 ersichtlich, ist es in diesem Model nicht einfach, die hierarchischen Stufen (1 und 2) voneinander zu trennen. Auch die Stufen 3 und 4 wurden in zwei parallelen statt in hierarchisch strukturierten Subroutinen implementiert. Die Tendenz, hierarchische Strukturen zu verschmieren, ist der Problemlösung mit Netzwerkmodellierung eigen. In einem anderen Projekt wurde die prozessorientierte Sprache MODULA-II (eine PASCAL Erweiterung) verwendet, um eine Modelleisenbahnanlage zu steuern. Dabei konnte die hierarchische Struktur der Prozesse: Zug, Fahrplan, Blockreservation und Disposition in der Software erhalten bleiben.

Allgemeiner Ueberblick: Alle bisher erhältlichen transaktionsorientierten Simulationspakete (wie z.B. GPSS-V oder Q-GERT) sind viel zu unflexibel, um komplexe Systeme, wie man sie in der Industrie antrifft, klar und verständlich zu modellieren. Von SLAM-II glauben wir, dass es das erste System ist, das flexibel genug ist, um darin grosse Industrieprozesse aller Art überzeugend modellieren zu können. Auch wenn die endgültige Software (wie in unserem Fall) auf grosse Teile in FORTRAN programmierter Subroutinen zurückgreift, rechtfertigt die Unterstützung, die SLAM-II bei der Modellierung bietet, bei weitem den Overhead, der einem Allzweckprogramm notgedrungenermassen zu eigen ist.

Verdankungen

Wir möchten dem Auftraggeber (Digitron AG) für die hoch interessante praktische Anwendung und für die gute Unterstützung während der Ausführung des Projektes unseren Dank aussprechen. Auch sind wir den Herren Dr. Hazeghi und Benninger vom Institut für Operations Research an der ETH für ihre Unterstützung und Zusammenarbeit bei diesem Projekt verbunden.

REFERENZEN

[1] Brown P. J.: "Macro Processors and Techniques for Portable Software", John Wiley, 1975.

[2] Cellier F. E.: "Macro-Handler for Simulation Packages Using ML/I", Proceedings des achten AICA Kongresses über Simulation of Systems, (Ed: L. Dekker), North-Holland Publishing Company, S. 515 - 521, (1976).

[3] Duket S. D., O'Reilly J. J. und Hannan R. J.: "SLAM-II, Enhanced Simulation

Capabilities", Pritsker and Associates, Inc., P. O. Box 2413, West Lafayette, IN 47906, 1981.

[4] Pritsker A. A. B. und Pedgen C. D.: "Introduction to Simulation and SLAM", Halsted Press (John Wiley) und Systems Publishing Corporation, 1979.

[5] Rimvall M. und Cellier F. E.: "GASP-VI: Ein Simulationspaket für prozess-orientierte gemischt kontinuierliche und diskrete Simulation", (gleicher Band).

[6] Wirth N.: "MODULA-II". Interner Bericht Nr. 36, Institut für Informatik, ETH-Zentrum, CH-8092 Zürich, Schweiz.