GASP-VI: EIN SIMULATIONSPAKET FUER PROZESS-ORIENTIERTE GEMISCHT KONTINUIERLICHE UND DISKRETE SIMULATION

Magnus Rimvall und François E. Cellier, Zürich

Zusammenfassung: GASP-V, der Vorläufer von GASP-VI, wurde zum ersten Mal während des 8. AICA Kongresses vorgestellt, welcher 1976 in Delft durchgeführt worden war. Diese Software wurde schliesslich im Frühjahr 1978 fertiggestellt und befindet sich seither sowohl in Industrie wie in Forschungsvorhaben erfolgreich im Einsatz. GASP-V stellt dem Anwender Modellierungsmechanismen für folgende Aufgaben zur Verfügung: (i) kontinuierliche Simulation (im Zustandsraum), (ii) diskrete Simulation (ereignisorientiert) und (iii) verteilte Simulation (unter Verwendung der Method-of-Lines) sowie beliebige Kombinationen dieser drei Problemklassen (gemischte Simulation). In der Zwischenzeit haben wir uns dazu entschlossen, dieses (FORTRAN-codierte) Softwarepaket um ein Prozessinteraktionskonzept sowohl für diskrete wie auch für kontinuierliche Prozesse zu bereichern. Diese Erweiterung von GASP ist insbesondere im Zusammenhang mit der neuen Simulationssprache COSY von Bedeutung, deren Run-Time-System GASP-VI abgeben soll. Allerdings sind wir der Meinung, dass ein Prozessinteraktionsmechanismus auch unabhängig von COSY seine Berechtigung hat, da er den Benützer bei einer modularen Formulierung von Modellen unterstützt und ihm somit hilft, fehlerfreie Programme zu schreiben. Das in dieser Arbeit vorgestellte System GASP-VI ist vollständig zu GASP-V aufwärts kompatibel und stellt einen solchen Mechanismus zur Verfügung. In diesem Beitrag konzentrieren wir uns auf die innovativen Aspekte dieses Projekts, als da wären: (i) ein neuer Mechanismus zur Listenbearbeitung mit Listenelementen von variabler Länge und (ii) a neuer Resourcenallozierungsmechanismus, welcher erlaubt, die Verwaltung der verfügbaren Resourcen wesentlich besser zu überwachen und welcher ausserdem ermöglicht, Resourcen (wie Transaktionen) mit individuellen Eigenschaften (Attributen) auszustatten.

Summary: GASP-V, the predecessor of GASP-VI, was presented for the first time during the 8th AICA (IMACS) Congress held at Delft in 1976. This software was finally completed in spring 1978, and since then it has been very successfully applied by many (both industrial and university) research groups. GASP-V provides for mechanisms for: (i) continuous simulation (in the state-space domain), (ii) discrete simulation (event oriented), and (iii) distributed simulation (method-of-lines) and any mixture of the three (combined simulation). In the mean time, it was decided that this (FORTRAN-based) software package should be enhanced by adding to it a process interaction mechanism for both continuous and discrete processes. This enhancement is particularly essential in the context of the new simulation language COSY for which GASP-VI is the run-time system. However, even independently of COSY, a process interaction mechanism is very useful as it supports a much more modular programming, and, by these means, helps the user in formulating his models and in coding them error free. The here presented new system GASP-VI is upwards compatible with GASP-V and provides for the above mentioned additional process interaction mechanism. This paper concentrates on the innovative aspects of this new development as there are: (i) a new mechanism for list processing with variable-length list entries, and (ii) a new resource allocation mechanism which allows for a significantly improved monitoring of resource allocations, and which moreover allows to add individual attributes to resources in a resource pool.

1. Einführung

GASP-VI ist eine neue (aufwärts kompatible) Ergänzung zu den bekannten und weit verbreiteten Simulationspaketen GASP-IV und GASP-V [2,3].

Obwohl GASP-IV bereits in der Lage war, gemischt kontinuierliche und diskrete Modelle zu bearbeiten, handelte es sich bei diesem Programmprodukt noch um ein ziemlich kleines System, welches auf der diskreten Seite nur einen einfachen Ereignismechanismus unterstützte und auch auf der kontinuierlichen Seite nur wenig ausgebaut war. (So offerierte GASP-IV z.B. nur eine einzige Integrationsmethode (vom Typ Runge-Kutta), welche fest in der Ablaufsteuerung eingebaut war, wodurch die Simulation steifer Systeme von vorneherein ausgeschlossen wurde.)

GASP-V erweiterte dieses Softwaresystem um:

1) eine Bibliothek von Integrationsalgorithmen, welche unter anderem auch die Kahaner-Implementation des GEAR-Algorithmus für steife Systeme beinhaltete,

2) Unterstützung bei der Modellierung verteilter Systeme (durch partielle Differentialgleichungen beschrieben), wobei das Method-of-Lines-Verfahren beigezogen wurde,

3) verbesserte Techniken zur Lokalisierung von Zustandsereignissen (unter Verwendung von invers Hermit'scher Interpolation),

4) eine Bibliothek von diskontinuierlichen Funktionen (in etwa derjenigen von CSMP entsprechend), welche z.B. Hysterese-, Puls- und Schrittfunktionen beinhaltet, wobei allerdings sämtliche Diskontinuitäten intern durch Ereignisse aufgelöst werden, wobei die hinlänglich bekannten numerischen Probleme (Kriechen) bei Verwendung solcher Funktionen in CSMP (oder einer entsprechenden Software) umgangen werden können,

5) einen Datenbankmechanismus, mittels welchem Simulationsdaten während der Simulation weggespeichert werden können; und einen Postprozessor (von DARE-P), welcher erlaubt, vorgängig gespeicherte Daten wieder rückzugewinnen und in verschiedenster Weise zur Darstellung zu bringen, sowie

6) verbesserte Unterstützung bei Optimierungsaufgaben.

Wie man leicht sieht, betreffen alle diese Verbesserungen die kontinuierliche Seite von GASP, während die diskrete Seite unverändert von GASP-IV übernommen wurde.

GASP-VI schliesslich fügt noch einen Prozessinteraktionsmechanismus sowohl für diskrete wie auch für kontinuierliche Prozesse zu. GASP-VI Programme ähneln weitgehend Programmen, welche in GASPPI [10], einem anderen von Pritsker & Associates entwickelten aber nie zur Produktionsreife gebrachten GASP-IV Derivat, wobei allerdings von GASPPI nur Konzepte nicht aber Realisationsmechanismen übernommen wurden.

Einer der Hauptgründe für die Entwicklung dieser neuen GASP-Version lag darin begründet, ein adäquates Zielsystem für die Simulationssprache COSY zu schaffen, welche in [2,4,6] beschrieben wurde. Ursprünglich war vorgesehen, COSY als "Frontend" zu GASP-V zu definieren, um das Entwickeln von GASP-Programmen etwas einfacher und weniger fehleranfällig zu gestalten. Während des Entwurfs dieser neuen Sprache sahen wir jedoch, dass COSY wesentlich bessere Möglichkeiten in sich barg, als dies ursprünglich von uns vorausgesehen worden war. Heute betrachten wir COSY als den schweizerischen Beitrag zum Ersatz der (unterdessen etwas demodierten) CSSL Spezifikation [11]. Mit diesem modifizierten Entwicklungsziel im Auge mussten wir feststellen, dass ein alzu starres Anklammern an den Konzepten, welche von GASP-V unterstützt werden, mehr eine Behinderung als eine Hilfe darstellt. So beinhaltet die Sprachdefinition von COSY nun mehrere (uns wesentlich erscheinende) neue Möglichkeiten, die sich nicht ohne weiteres in GASP-V realisieren lassen, während einige Möglichkeiten der GASP-Software in COSY nicht mehr unterstützt werden, da sie durch parallele (aber bessere) Alternativen ersetzt werden konnten. Als COSY schliesslich vollumfänglich definiert war, zeigte es sich, dass GASP-V als Zielsoftware für COSY nicht mehr sehr gut geeignet war. Deshalb wurde nun wiederum auf der GASP-Seite eine Anpassung notwendig, die wir glauben, mit GASP-VI sinnvoll erreicht zu haben. Dennoch haben wir uns entschlossen, in GASP-VI auch solche Komponenten von GASP-V beizubehalten, die in COSY nicht genutzt werden, um GASP-VI voll mit GASP-V aufwärts kompatibel zu belassen, da wir dadurch hoffen, den Einstieg für vormalige GASP-V-Benützer ebenfalls attraktiv zu gestalten. Ein Konflikt, welcher sich bei dieser Vorgehensweise (Aufspaltung in Preprozessor und Ausführungsprogramm) unausweichlich stellt, liegt darin begründet, dass Fehlermeldungen der Software sich immer auf das Quellenprogramm beziehen sollten, d.h. bei COSY Benützern auf ihr COSY-Programm und bei GASP-VI Benützern auf ihr GASP-Programm. Somit sollte sich GASP-VI bei Auftreten eines Fehlers anders verhalten, wenn das fehlerhafte Programm direkt von einem Benützer geschrieben worden ist, als wenn dieses durch Vorübersetzung eines fehlerhaften COSY Programms entstanden ist. Aus diesem Grund ist es notwendig, dass der COSY-Vorübersetzer neben einem GASP-VI Programm auch noch eine (maschinenlesbare) Kreuztabelle von COSY bzw. GASP Variablen und Zeilennummern zusammen mit einem (dem "normalen" GASP-VI Benützer unbekannten) Switch in den GASP-VI Datenkarten erzeugt, welcher dem GASP-System mitteilt, ob es sich beim Ausgangsprogramm um ein COSY oder aber um ein GASP Programm gehandelt hat. Dieses Flag beeinflusst aber ausschliesslich die Fehlerbehandlung sowie das Tracing, wodurch wohl ein

gewisser Overhead an Speicherplatz, nicht jedoch ein Overhead an Ausführungszeit in Kauf genommen werden muss.

GASP-VI, das jüngste Kind in der GASP-Familie, soll in dieser Arbeit kurz vorgestellt werden.

2. Datenstrukturen in GASP-VI

Neben den in FORTRAN standardmässig unterstützten Datentypen (Skalare und Arrays) unterstützt GASP-IV (und somit auch GASP-V) einen weiteren Datentyp, nämlich die vorwärts und rückwärts linear verkettete Liste, welche in GASP-IV als "File" oder "Queue" bezeichnet wird. Jeder Liste zugeordnet ist eines (aus vier verfügbaren) Einreihekriterien (z.B. FIFO (First-in First-out) oder HVF (high value first) angewandt auf ATRIB(5), etc.). Die Manipulation von Listenelementen (Records) geschieht mittels Listenverwaltungsroutinen (z.B. FILEM, um neue Elemente einzutragen oder aber RMOVE, um Elemente aus Listen zu entfernen).

Diese Datenstruktur wird für zwei verschiedene Zwecke verwendet, nämlich:

1) zur Modellierung von Warteschlangen, die in fast allen diskreten Simulationsproblemen vorkommen, und

2) zur Verwaltung des Ereigniskalenders. Diese zweite Verwendungsart ist insoweit speziell, als:

   a) jeder Ereignis-Record zwei Standard-Eigenschaften enthält, nämlich:
      - die Ereigniszeit (Zeit, zu der das Ereignis stattfinden soll): ATRIB(1), sowie
      - die Ereignisart (Typ des Ereignisses, welches stattfinden soll): ATRIB(2);

   b) Ereignisse beim Eintreffen automatisch aus dem Ereigniskalender entfernt werden;

   c) genau ein Ereigniskalender (File Nr. 1) existiert;

   d) die Einreiheregel für diese Liste immer LVF (low value first) angewandt auf ATRIB(1) ist (in GASP-V allenfalls auch HVF angewandt auf ATRIB(1) im Falle von Rückwärtsintegration), wobei eine sekundäre Einreiheregel für gleichzeitig stattfindende Ereignisse spezifiziert werden kann.

In der Implementation von SLAM-II [9], einem anderen Abkömmling von GASP-IV, haben

die Inauguratoren (ebenfalls Pritsker & Associates) offensichtlich realisiert, dass die oben skizzierte Lösung nicht ideal war, indem:

1) der GASP-IV Benützer immer in Erinnerung behalten musste, dass in der Ereignisliste die ersten zwei Attribute vom System reserviert sind (das System wurde so modifiziert, dass nunmehr die Standardattribute als letzte Attribute aufbewahrt werden), und

2) der GASP-IV Benützer sich immer daran erinnern musste, dass das File Nr. 1 vom System für die Verwaltung der Ereignisse benötigt wurde, wodurch die frei zur Verfügung stehenden Warteschlangen von 2 an durchnummeriert werden müssen (daher bewahrt SLAM-II seine Ereignisse als letzte Liste auf).

Sicherlich stellt diese Modifikation eine leichte Verbesserung gegenüber GASP dar. Dennoch löst auch sie nicht alle Probleme, da bei der Prozessinteraktion zusätzliche Listenarten benötigt werden, um z.B. die Attribute der Transaktionen darin aufzubewahren. Auf Grund der oben erwähnten Beschränkungen bewahrt SLAM-II diese Attribute nicht in Listen auf, die dem Benützer zugänglich sind, wodurch z.B. das Zugreifen auf die Attribute einer "anderen" Transaktion sehr umständlich, wenn nicht unmöglich wird. Ebenso wird es sehr schwierig, in SLAM-II die Attribute einer Transaktion in einem Ereignis zu verändern, ausser wenn das Ereignis dadurch zu Stande kam, das eine Transaktion durch einen EVENT-Knoten passierte.

Es war eines der Entwurfsziele von GASP-VI, dass sowohl Transaktions-Records als auch Resourcen-Records (über welche wir noch ausführlicher sprechen werden) für den Benützer transparent unter Verwendung des normalen Listenmechanismus aufbewahrt werden sollten. Zu diesem Zweck erwies es sich allerdings als notwendig, den Listenverwaltungsmechanismus von GASP-IV namhaft abzuändern und tatsächlich zu grossen Teilen neu zu schreiben.

Betrachten wir nochmals die Transaktions-Records etwas genauer. Für solche Records unterhält GASP-VI 11 transaktionsspezifische Standardattribute (sogenannte "hidden attributes"):

1) Zeiger zu einem Zeitereignis-Record
2) Zeiger auf einen Resourcen-Bedarfs-Record
3) Zeiger zur Liste der allozierten Resourcen
4) Zeiger, der auf einen "Timeout"-Record weist
5) Aktueller Status der Transaktion
6) Prozessnummer
7) Blocknummer

8) Erzeugungszeit der Transaktion
9) Letzte Markierungszeit der Transaktion
10) Priorität
11) Neue Blocknummer (im Falle der Ausführung eines GGOTO-Statements).

Aus dieser Liste sollte klar werden, dass:

a) es dem Benützer weder zugemutet werden kann, seine eigenen applikationsspezifischen Attribute vom Index 12 an aufwärts zu zählen (und wiederum anders für kontinuierliche Prozesse, Resourcen, etc.), noch es sinnvoll ist, den Benützer z.B. die Markierungszeit seiner Transaktion dadurch zugänglich zu machen, dass er zur Anzahl seiner eigenen Attribute einen Offset von 9 dazuaddieren muss. Aus diesem Grunde wurde in GASP-VI eine Lösung gewählt, bei welcher sich der Benützer in GASP-VI ausschliesslich um seine eigenen Attribute zu kümmern braucht, während die Standardattribute ihm jeweils in einem speziellen COMMON-Block bereit gestellt werden, wohin sie das System automatisch kopiert.

b) wir sehr leicht Situationen antreffen können, bei welchen die eine Liste (welche z.B. Transaktionen beinhaltet) 50 verschiedene Attribute aufweist, während eine andere (z.B. eine Warteschlange) mit einem oder zwei Attributen auskommt. Im Falle des Ereigniskalenders wird es sogar der Normalfall sein, dass ein Rekord sehr viele Einzelattribute aufweist, während ein anderer nur über ganz wenige verfügt. Daraus folgt: Weder ist die GASP-IV Regel, wonach alle Listen genau gleich viele Attribute aufweisen müssen, applikabel, noch diejenige, wonach die Anzahl Attribute die Zahl 25 nicht überschreiten darf. Um dieses Problem zu lösen, unterscheiden wir in GASP-VI zwischen:

   - <u>logischen Rekords</u>, welche eine beliebige Grösse aufweisen können (keine Begrenzung), und welche auch ungleich gross sein dürfen, und

   - <u>physikalische Rekords</u>, welche auch "Slices" genannt werden, welche von gleicher Länge sein müssen, wobei aber auch hier keine Begrenzung für die zulässige Grösse vorgesehen ist.

Jeder logische Rekord enthält jetzt zumindest drei Zeiger, wobei deren zwei (wie in GASP-V) auf den Vorgänger bzw. Nachfolger zeigen, während der dritte neue Zeiger auf den nächsten Slice, der zum logischen Rekord gehört, weist. Drei zusätzliche versteckte Attribute sind <u>allen</u> Rekords zu eigen:

1) die Anzahl der (versteckten sowie der vom Benützer spezifizierten) Attribute des logischen Rekords, wobei automatisch vom System eine Statistik über die durch-

schnittliche Anzahl Attribute erstellt wird, eine Grösse, die fürs optimale "Tuning" nützlich ist,

2) die Art des Rekords, welche benötigt wird, um einen lesbaren "Trace" zu erzeugen, und auch, um einer illegalen Verwendung des Rekords vorzubeugen, sowie

3) der letzte Zeitpunkt, zu welchem der Rekord "bewegt" worden ist, eine Information, welche zwei Verwendungszwecken dient:

   a) Statistiken werden automatisch über die mittlere Zeit erstellt, welche die Rekords in jeder Warteschlange verbracht haben, dies neben der (auch in GASP-V ermittelten) durchschnittlichen Anzahl Rekords in jeder Warteschlange;

   b) GASP-VI verfügt über einen automatischen "Timeout" Mechanismus, welcher erlaubt, "Deadlocks" zu erkennen (wenn auch nicht zu beseitigen).

Die physikalische Grösse jedes Slices (NNATR) muss vom Benützer festgelegt werden, wobei NNATR vom System (wenn nötig) automatisch auf 10 erhöht wird, sobald (kontinuierliche oder diskrete) Prozesse im Modell vorkommen. Diese Grösse hat einen Einfluss auf die Ausführungseffizienz des Programms. Unserer Erfahrung nach sollte NNATR so dimensioniert werden, dass etwas mehr als 50% aller Rekords in einem Slice Platz finden.

3. Resourcen

Zur Illustration der Probleme, welche bei der Resourcenverwaltung auftreten, wollen wir eine Wagenvermietungsfirma betrachten. Turisten (Transaktionen) kommen an und versuchen, einen Wagen zu mieten (eine Resource zu allozieren). Solange Wagen verfügbar sind, erhält der Turist sein Fahrzeug, andernfalls muss er warten, bis ein Fahrzeug zurückgegeben wird, oder aber auf die öffentlichen Verkehrsmittel zurückgreifen.

Ein solcher Mechanismus, der von den meisten prozess-orientierten Sprachen zur Verfügung gestellt wird, wird in allen uns bekannten Systemen (so z.B. in SLAM-II, in GPSS-V und in GPSS-FORTRAN) intern durch einen INTEGER Zähler implementiert, der dekrementiert wird, wenn ein Wagen die Firma verlässt, und inkrementiert, sobald er zurückkommt. Gewöhnlich repräsentiert ein zweiter INTEGER die Kapazität der Resource, so dass wenigstens Kapazitätsüberschreitungen detektiert werden können.

Wird dieser Mechanismus auf diese Weise implementiert, kann ein Turist seinen gemie-

teten Wagen an der nächsten Kreuzung abstellen und nach Hause fliegen, ohne missliebige Folgen gewärtigen zu müssen. Mit anderen Worten kann eine Transaktion jederzeit das System verlassen, ohne ihre momentan allozierten Resourcen wieder freizugeben. Dies ist unvermeidbar, da Resourcen keine Individualität besitzen, und es somit unmöglich ist, nachträglich zu eruieren, welche Resource abhanden gekommen ist. Noch widersinniger: Ein Turist, der die ganzen Ferien hindurch mit öffentlichen Verkehrsmitteln gereist ist, kann ohne weiteres einen nie gemieteten Wagen zurückgeben. Dies kann frühestens dann bemerkt werden, wenn einmal zufällig sämtliche Wagen beim Verleiher sind, da dann allenfalls eine Kapazitätsüberschreitung gemeldet werden kann.

Selbstverständlich ist diese Implementationsart sehr effizient in der Ausführung und funktioniert auch richtig, wenn man davon ausgeht, dass der Benützer keine Fehler macht. Wir sind jedoch der Auffassung, dass es eine der "nobelsten" Aufgaben jeder Software sein muss, dem Benützer auf die Finger zu klopfen, wenn er etwas falsch macht!

In GASP-VI haben wir darum versucht, eine andere Implementation der Resourcenverwaltung zu finden, welche wohl etwas ineffizienter in der Ausführung ist, dabei aber eine wesentlich erhöhte Programüberwachung erlaubt. Dabei wird jede Resource durch einen eigenen Rekord repräsentiert. Alle freien Resourcen sind in einer linear verketteten Liste zusammengefasst (Wagenpark). Sobald eine der Resourcen alloziert wird, wird ihr Rekord aus dieser Liste entfernt und in eine andere Liste umgehängt, welche alle momentan von der Transaktion allozierten Resourcen (beliebigen Typs) enthält. Diese Liste ist als ganzes an den Transaktionsrekord angehängt. Wenn ein Turist vergisst, seinen Wagen, Hotelzimmerschlüssel oder ähnliches zurückzugeben, wird automatisch bei seinem Einchecken im Flugplatz eine Warnung ausgedruckt, und die nicht zurückgegebenen Resourcen werden ihm mitleidlos vom System abgeknöpft. Wird einem Turisten, der einen Wagen gemietet hat, dieser über Nacht von einem anderen Turisten "gestohlen", so muss dieser andere Turist schon ein sehr gewiegter GASP Programmierer sein, wenn es ihm gelingen sollte, den gestohlenen Wagen an die Vermietung zurückzugeben oder sich anderweitig aus dem System wegzustehlen.

Ein Nebeneffekt dieser Methodologie liegt darin, dass automatisch Resourcenverwendungsstatistiken über jede Resource im Pool individuell erstellt werden können.

Ein anderer Nebeneffekt liegt darin, dass jede Resource im Pool ihre eigenen individuellen Attribute mit sich herumtragen kann. So mag z.B. die Vermietungsstelle Wagen aus verschiedenen Klassen zu unterschiedlichen Preisen anbieten, und der Kunde kann entscheiden, ob er irgend einen Wagen will oder aber einen Wagen aus einer bestimmten Preisklasse. Dennoch gibt es nur eine Art Resourcen vom Typ Wagen. Wir Konnen hier nicht weiter auf die (unserer Meinung nach sehr wesentlichen) Vorteile dieser

Möglichkeit eingehen. Diese wurden aber in [6] ausführlich abgehandelt.

Wie jede andere Lösung, weist auch diese Implementation ihre Schwächen auf. Dies kann z.B. an Hand einer Rechnersimulation aufgezeigt werden. Soll nämlich der Kernspeicher eines Rechners durch eine Resource dargestellt werden, scheint es kaum sinnvoll, 4 MByte Speicher durch 4 Millionen individueller Resourcenrekords darzustellen! Es ist daher bei dieser Implementation wichtig, die Anzahl individueller Resourcen in einem Pool in Grenzen zu halten. Aus diesem Grund offeriert GASP-VI noch einen zweiten Resourcenverwaltungsmechanismus, bei welchem ein Resourcenpool durch einen einzigen Rekord dargestellt wird, welcher die Anzahl verfügbarer Resourcen als verstecktes Attribut mit sich führt. Wird ein Teil dieser Resourcen belegt, muss dieser Rekord kopiert werden, wobei in der alten Kopie die Anzahl nun noch verfügbarer Resourcen aufbewahrt wird, während die neue Kopie,welche der Liste der allozierten Resourcen der betroffenen Transaktion angehängt wird, die Anzahl neu allozierter Resourcen mit sich trägt. Selbstverständlich können solche Resourcen keine individuellen Attribute aufweisen. Dennoch erweist sich auch dieser Mechanismus immer noch bei weitem sicherer gegenüber Programmierungsfehlern als der üblicherweise verwendete. Auch dazu finden sich weitere Erläuterungen in [6].

4. Status der Implementation

Momentan ist die Implementation soweit gediehen, dass:

a) die Sprache COSY (mit Ausnahme einiger sekundärer Details) vollständig entworfen ist, wobei ein allgemein verwendbarer Parser zum Einsatz gelangte, welchen wir in [1] beschrieben haben;

b) mehrere recht umfangreiche Anwendungsprogramme sind in COSY codiert und (unter Verwendung des Parsers) auf ihre Korrektheit überprüft worden, dies, um die Eignung der neuen Sprache zur Modellierung grosser Systeme zu verifizieren;

c) die neuen Datenstrukturen von GASP-VI sowie die neuen Ueberwachungs- und Testroutinen ("Monitoring", "Tracing", "Debugging") sind geschrieben und ausgetestet worden.

d) ein erster Satz von Prozessinteraktionsroutinen ist codiert worden, der sich im Augenblick in der Testphase befindet, wobei zusätzliche Routinen bereits spezifiziert aber noch nicht ausprogrammiert worden sind.

Es ist vorgesehen, die Entwicklung von GASP-VI (inklusive Gebrauchsanleitung) noch in diesem Jahr abzuschliessen. Andrerseits muss die Implementation von COSY warten,

bis GASP-VI vollständig verfügbar und ausgetestet ist.

5. Vergleich von GASP-VI mit SLAM-II und mit COSY

Man mag sich fragen, ob die Entwicklung des neuen Systems GASP-VI wirklich gerechtfertigt war. Eine Alternative könnte allenfalls darin gelegen haben, die erweiterten Möglichkeiten von GASP-V (gegenüber GASP-IV) ins System SLAM-II einzubauen, was sicherlich mit wesentlich kleinerem Aufwand verbunden gewesen wäre. SLAM-II, wozu eine Anwendung im begleitenden Bericht [7] beschrieben wurde, ist ein weiterer Abkömmling von GASP-IV, dessen kontinuierliche Seite sich mit GASP-IV völlig deckt, während seine diskrete Seite um einen Netzwerkbeschreibungsmechanismus ergänzt wurde. Für kleinere Schulbeispiele (wie z.B. Joe's Friseurladen), ist es sehr einfach zu zeigen, dass eine solche Netzwerkbeschreibung einer Prozessinteraktionsbeschreibung äquivalent ist. Leider trifft dies bei komplexeren Anwendungen nicht mehr zu. Die Aequivalenz gilt nur für sehr simple Prozesse, die durch eine sequentielle Hintereinanderreihung von Aufrufen von Prozessinteraktionsroutinen beschrieben werden können (z.B. Erzeugung einer neuen Transaktion, gefolgt von der Allokation einer Resource, gefolgt von einer Aktivität, etc.). Sobald jedoch diese Aufrufe mit sequentieller Logik durchsetzt sind, wie z.B. beim Aufzugproblem, für welches eine COSY Lösung in [5] präsentiert wurde, müsste eine Netzwerkbeschreibung, wie sie in SLAM-II propagiert wird, beinahe ausschliesslich aus einer Hintereinanderreihung von Ereignisknoten bestehen, was weder zu schreibbaren noch zu lesbaren Programmen führt, und wodurch sich diese Vorgehensweise weder als attraktiv noch als effizient erweist. Dies bedeutet: Für einfache Prozesse sind COSY und SLAM-II in etwa äquivalent, wobei sich SLAM-II ohne jeden Zweifel sowohl bezüglich Länge des vom Benützer zu erstellenden Programms wie auch bezüglich Fehlerwahrscheinlichkeit als GASP-VI bei weitem überlegen erweist. Sobald jedoch ein Problem nach extensiver Logik ruft, erhält GASP-VI einen gewissen Vorsprung vor SLAM-II, während COSY bei weitem die attraktivste Variante darstellt. Auf jeden Fall jedoch eignet sich GASP-VI in solchen Fällen wesentlich besser zur Zielsprache für COSY als SLAM-II.

REFERENZEN

[1] Bongulielmi A. P. und Cellier F. E.: "On the Usefulness of Deterministic Grammars for Simulation Languages". Proc. der Sorrento Arbeitstagung über International Standardization of Simulation Languages (SWISSL), Sorrento, Italien, 19. + 20. Sept., (1979).

[2] Cellier F. E.: "Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools". Dissertation, Diss ETH No 6483, Eidgenössische Technische Hochschule, Zürich, 1979.

[3] Cellier F. E. und Blitz A. E.: "GASP-V: A Universal Simulation Package". Proc. des achten AICA Kongresses über Simulation of Systems, Delft, Niederlande, 23. - 28. August 1976. Publiziert durch North-Holland Publishing Company (Ed.: L. Dekker); S. 391 - 402, (1976).

[4] Cellier F. E. und Bongulielmi A. P.: "The COSY Simulation Language". Proc. des neunten IMACS Kongresses über Simulation of Systems, Sorrento, Italien, 23. - 27. Sept. 1979. Publiziert durch North-Holland Publishing Company (Ed: L. Dekker, G. Savastano und G. C. Vansteenkiste); S. 271 - 281, (1979).

[5] Cellier F. E., Bongulielmi A. P. und Rimvall M.: "Comments of the Swiss TC3 Group on the 'Outline Proposal for a New Standard for Continuous-System Simulation Languages (CSSL 81)'". In: TC3 of IMACS, Committee on Simulation Software, Committee Newsletter, Nr. 10, Sept. 1981, Appendix 3 (Ed: R. E. Crosbie und F. E. Cellier).

[6] Cellier F. E., Rimvall M. und Bongulielmi A. P.: "Discrete Processes in COSY". Proc. des European Simulation Meetings über Simulation Methodology, gehalten in Cosenza, Italien, 9. - 11. April 1981, (Ed: F. Maceri), (1981).

[7] Graber A. und Cellier F. E.: "Eignung der Simulationssprache SLAM-II zur Modellierung und Simulation grosser Transportsysteme". (Gleicher Band)

[8] Pritsker A. A. B.: "The GASP-IV Simulation Language", John Wiley, 1974.

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

[10] Washam W. B. und Pritsker A. A. B.: "Introduction to GASPPI". Unpubliziertes Dokument, Pritsker & Assoc., Inc., 1976.

[11] "The SCi Continuous System Simulation Language (CSSL)". Simulation, Band 9, Nr. 6, Dez. 1967; S. 281 - 303, (1967).