Anwendung graphischer Konzepte in Simulationssystemen

Prof. Dr. Wolfgang P. Kowalk,
Universität Oldenbur & OFFIS , Escherweg 2, 26121 Oldenburg)

1 Einführung

Simulation kann allgemein als zukunftsorientierte Technik für die Modellierung von Phänomen der realen Welt betrachtet werden. Dieses hat seinen Grund in verschiedenen Beschränkungen, denen andere Modellierungskonzepte unterliegen.
Unter Simulation verstehen wir hier die Nachbildung des dynamischen Verhaltens eines realen Systems. Unter einem dynamischen Modell verstehen wir zum einen eine Beschreibung des (statischen, augenblicklichen) Zustands eines Systems, zum anderen eine vom augenblicklichen Zustand, einer möglichen externen Einwirkung sowie der jeweiligen Zeit abhängigen Änderung dieses Zustands. Simulationsprogramme gestatten es in der Regel, die Zustände und deren Wechsel zu beschreiben und zugleich mit dem Ablauf der Zeit zu verfolgen.
Die Modellierung solcher Systeme, soweit sie nicht trivial sind, erfordert auf jeden Fall eine Möglichkeit der Strukturierung des Modells. Solche Konzepte wurden früher durch Programmiersprachen erreicht, unter denen Simula67 historisch vielleicht das prägnanteste Beispiel ist, da hiermit ein erst später wieder aufgenommenes Programmierparadigma, das objektorientierte, eingeführt wurde. Simula67 erlaubt es dem Programmierer, durch das Konzept der Vererbung die Datenstrukturen und Funktionalitäten, die für die Simulation nötig sind, in die einzelnen Prozesse einzufügen, welches gekapselte Objekte sind, die ihre lokalen Werte schützen. Wenngleich dieses einen großen Fortschritt bedeutet, so erhält man jedoch auch mit dieser Sprache lediglich eine linear notierte Menge von Prozessen, die über gemeinsame Datenspeicher („`set`") miteinander kommunizieren. Auch bei diesem Ansatz gibt es verschiedene Kritikpunkte, die jedoch an dieser Stelle nicht im einzelnen aufgelistet werden sollen.
Eine weitere Entwicklung stellt die Benutzung graphischer Oberflächen dar. Diese kommen natürlich dem menschlichen Auffassungsvermögen wegen der größeren Informationsmenge, die durch die Augen aufgenommen werden, sehr viel mehr entgegen als eine sprachliche Notation. Hier gibt es zwei prinzipielle Konzepte. Das eine verwendete eine Programmiersprache (linear notierter Text), stellt jedoch Mittel zur Verfügung, um das dynamische Verhalten des Systems anschaulich z.B. durch Animation darzustellen. Dieses hilft besonders, ein erstelltes Modell zu validieren, so daß dieses Konzept auf keinen Fall unterschätzt werden sollte. Dennoch ist es mit verschiedenen Problemen behaftet. Die umständliche lineare Notation, die damit verbundene Organisation von Namensräumen sowie die explizite Einbindung der Graphik unterstützt den Modellierer bei der Erstellung seines Simulationsprogramms nur wenig. Außerdem werden häufig Leistungseinbußen festgestellt, da die graphische Funktionalität die Effizienz des Programmablaufs mindert.
Die zweite Konzeption geht von einer graphischen Eingabesprache aus. Hierbei wird das Modell aus seiner graphischen Repräsentation heraus erstellt und gibt somit dem Programmierer die Möglichkeit, weitgehend unabhängig von Namensräumen und zusätzlichen Sprachkonstrukten ein Modell zu erstellen, welches sofort anschaulich seine Struktur sowie seine einzelnen Komponenten wiedergibt. Das ist jedoch auch mit einigen Nachteilen ver-

bunden, die bei der Entwicklung graphischer Simulationssysteme wegen der zentralen Problematik für diese Untersuchungen etwas sorgfältiger analysiert werden sollten. Wir können dieses aus Platzgründen lediglich andeutungsweise in den folgenden Betrachtungen zu einem von uns entwickelten Simulationssystem verfolgen.

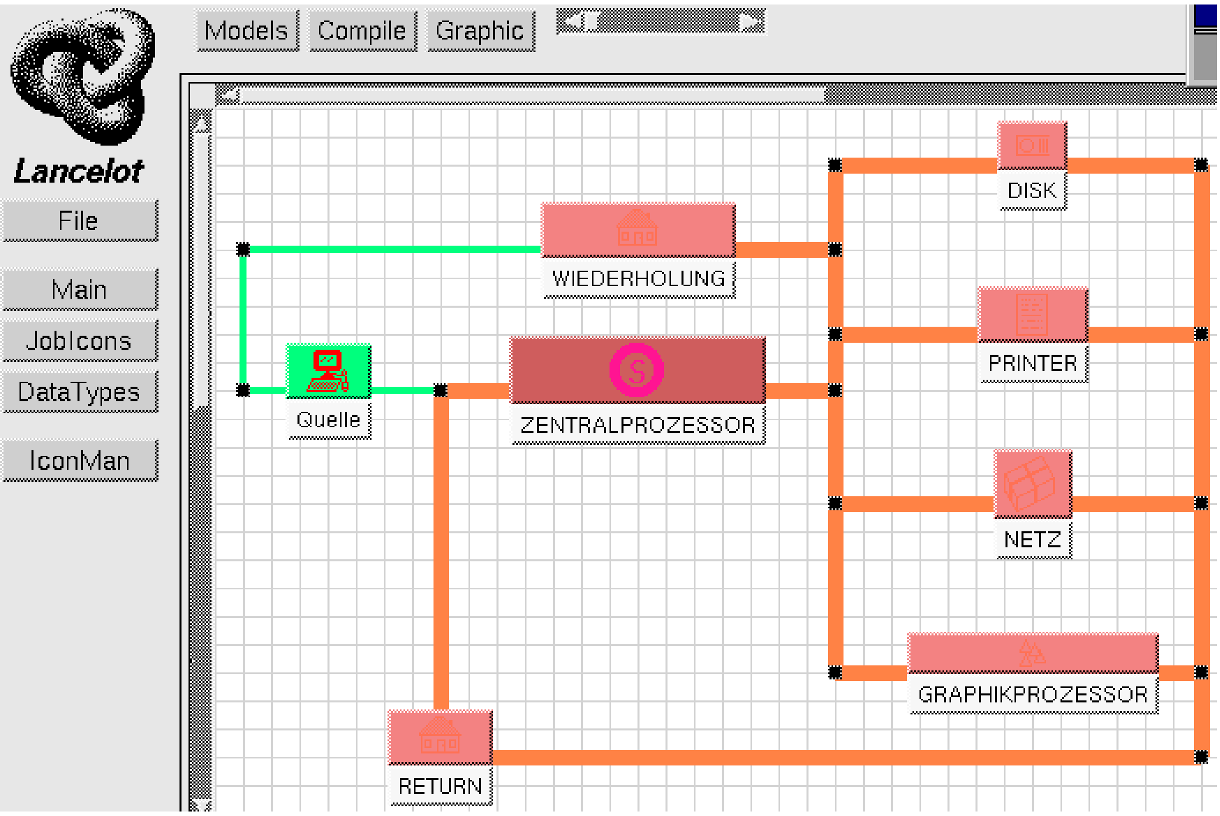

Bild 1-1: Das klassische zentrale Bediensystem (*Central Server*) im Simulationssystem ***Lancelot***

2 Das Simulationssystem *Lancelot*

An der Universität Oldenburg wurde ein Simulationssystem entwickelt, welches primär auf eine graphische Unterstützung abzielt, so daß verschiedene Anforderungen an graphische Simulationskonzepte unterstützt werden. Diese sollen jetzt im einzelnen vorgestellt werden.

2.1 Graphische Unterstützung

Die graphische Unterstützung erhält man in ***Lancelot*** bereits bei der Eingabe des ersten Programmtexts. Nach dem Starten des Systems wird zunächst ein Fenster geöffnet, in welchem die Modelle erstellt werden müssen. Außerdem hat man über dieses Fenster Zugriff zu einer Modelldatenbank, in der bereits erstellte Modelle abgespeichert sind. Diese können wahlweise ganz geladen werden, oder es werden nur Teile von ihnen geladen, indem sie in einem Modell-Manager zwischen gespeichert werden. Aus diesem Modell-Manager können bereits

erstellte Teilmodelle dann in das fertige Modell kopiert werden. Somit stellt die graphische Benutzeroberfläche von ***Lancelot*** dem Programmierer bereits eine einfache und effiziente Möglichkeit zur Verfügung, bereits erstellte Modelle wiederzuverwenden.
Die zunächst leere Oberfläche kann mit Objekten gefüllt werden, wobei es (zur Zeit) im wesentlichen drei verschiedene Arten von Objekten gibt, die Prozesse, die komplexen Objekte und die Warteschlangen. ***Lancelot*** arbeitet also prozeßorientiert, wobei jeder Prozeß eine eigene logische Einheit darstellt, die zwar mit der gleichen Zeit wie andere Prozesse arbeitet, die aber mit anderen Prozessen nur über den Austausch von Nachrichten interagiert. ***Lancelot*** nennt seine Nachrichten „Jobs“, was eher historisch begründet ist; sie können auch „Messages“ oder „Signals“ genannt werden. Eine Nachricht ist ein Datensatz (in C-Syntax), der beliebige Daten speichern kann. Die grundlegende Sprache ist (wegen ihrer großen Verbreitung) im wesentlichen C, mit einigen Ergänzungen.

2.2 Das Prozeßkonzept von *Lancelot*

Ein Prozeß besteht aus einem lokalen Variablenteil und einer Folge von Programmstücken, welche die Werte lokaler oder gegebenenfalls auch globaler Variablen verändern. Diese Programmstücke werden „Phasen“ genannt, da sie im Sinne des ereignisorientierten Simulationskonzepts beschreiben, welcher Zustandswechsel in einem Zeitintervall durchgeführt werden soll.
Das Phasenkonzept hat im Gegensatz zu dem vergleichbaren Prozeßkonzept von Simula67 vor allem den Vorteil, strukturierter zu sein. Der Programmierer kann somit nicht mehr an jeder Stelle im Programm durch eine geeignete Anweisung („`hold`“ in Simula67) den Prozeß eine gewisse Systemzeit unterbrechen, sondern muß dieses in eindeutig definierten Abschnitten tun. Dieses verhindert eine gewisse Unübersichtlichkeit, die bei der Anwendung solcher Anweisungen in einem komplexen Ausdruck entstehen kann.
Es wurde bereits erwähnt, daß Prozesse die gleiche Zeitbasis verwenden, d.h. wenn zwei Prozesse mit der gleichen Geschwindigkeit vorangehen, so laufen sie stets synchron. Dieses kann in einem gewissen Sinne verhindert werden, indem ein Prozeß als neue Zeitbasis eingeführt wird; andere Prozesse, die sich von dieser neuen Zeitbasis abhängig machen, laufen nur nacheinander ab und werden dabei durch den Steuerungsprozeß kontrolliert. Dieses sehr allgemeine Konzept kann beispielsweise benutzt werden, um das Zeitteilungsverfahren moderner Rechensysteme (*time-sharing*) zur Realisierung des Mehrprozeßbetriebs nachzubilden, was in anderen Simulationssystemen meistens nur sehr schwierig möglich ist.

2.3 Das Phasenkonzept von *Lancelot*

Die Phasen sind Programmeinheiten, die eine Eingangsbedingung, einen Zustandswechsel sowie eine Nachfolgephase angeben, welche zu einer bestimmten Zeit auszuführen ist.
Ein Zustandswechsel wird durch Zuweisung von Werten an die jeweiligen lokalen oder globalen Variablen durchgeführt. Die in einem Prozeß definierten Werte sind lokal und können in der Regel nur durch diesen Prozeß verändert oder gelesen werden. Die globalen Werte sind allen Prozessen in einer Umgebung gemeinsam bekannt und können dementsprechend von allen geändert oder gelesen werden. Sie können somit auch zur Kommunikation zwischen Prozessen verwendet werden.
Die Bestimmung der Nachfolgephase geschieht einfach durch Benennung dieser Phase. Die Zeit wird explizit angegeben und als differentielle Zeit zur augenblicklichen aufgefaßt. Es gibt besondere Anweisungen, die verwendet werden können, wenn die gleiche Phase gemeint ist,

in der sich der Prozeß befindet, oder wenn die nächste Phase zu einem bestimmten Zeitpunkt ausgeführt werden soll.
Ehe eine Phase ausgeführt wird, kann eine Bedingung getestet werden. Diese definiert verschiedene Umstände, die zutreffen müssen. Zum einen kann eine allgemeine logische Bedingung geprüft werden. Solange diese nicht zutrifft, wird der Prozeß verzögert. In ***Lancelot*** kann durch „`WaitFor(Bedingung);`“ eine solche Funktionalität formuliert werden.
Eine spezielle Bedingung besteht darin, daß ein Prozeß eine Nachricht erwartet. Wird beispielsweise das Eintreffen eines Pakets in einem Kommunikationsprotokoll erwartet, so kann durch die Anweisung „`Await(Packet);`“ am Beginn einer Phase dieser Prozeß solange verzögert werden, bis dieses Paket eingetroffen ist. In manchen Situationen soll jedoch höchstens eine maximale Zeit gewartet werden, beispielsweise wenn es sich bei dem Paket um eine Quittung handelt und u.U. verloren gehen könnte. Dieses wird in ***Lancelot*** durch „`AwaitUntil(Packet, DelayTime);`“ formuliert.

2.4 Das Kommunikationskonzept von *Lancelot*

Lancelot besitzt ein verallgemeinertes asynchrones Kommunikationskonzept, welches zwei oder mehrere Prozesse über einen Puffer miteinander kommunizieren läßt; sobald ein Puffer „voll“ ist, muß jeder sendende Prozeß solange warten, bis ein empfangender Prozeß diesen Puffer leert, d.h. mindestens einen „Job“ aus diesem entfernt.
Ist das Fassungsvermögen des Puffers unbeschränkt, so handelt es sich um eine rein asynchrone Kommunikation. Ist das Fassungsvermögen jedoch null, so müssen sich sendender und empfangender Prozeß auf jeden Fall synchronisieren, ehe sie ihre Nachrichten austauschen und somit weiter arbeiten können. Es gibt jedoch die Möglichkeit, den Prozessen eine Zeit mitzugeben; sobald diese abgelaufen ist, läuft der Prozeß auf jeden Fall weiter, auch wenn er seine Nachricht nicht abgeben konnte bzw. wenn er keine Nachricht empfangen konnte. Dieses kann durch den Zustand der Variablen, die die Nachrichten enthalten, eindeutig erkannt werden. Natürlich kann ein Puffer auch eine endliche Kapazität besitzen, so daß die Prozesse sich nur synchronisieren müssen, wenn der sendende Prozeß sehr viel schneller arbeitet als der empfangende.
Dieses Konzept kann entweder über eine explizite Warteschlange oder über eine implizite, einem Prozeß zugeordnete Warteschlange realisiert werden. Im letzteren Fall kann der sendende Prozeß dem empfangenden Prozeß (ohne die Warteschlange zu benennen) eine Nachricht schicken, so daß die Warteschlange vollständig verborgen bleibt. Dieses Konzept ist dann besonders nützlich, wenn die Verwendung expliziter Warteschlangen nicht gefordert wird. Die Erfahrung zeigt, daß die impliziten Warteschlangen in vielen Anwendungen sehr einfache Programme ergeben.

3 Strukturierungskonzepte in *Lancelot*

Im letzten Abschnitt wurde gezeigt, wie Prozesse, die in einem gewissen Sinne elementare Bauteile eines Simulationsprogramms sind, aufgebaut sind. Diese geben die wesentliche Funktionalität eines Simulationsprogramms an. Prozesse werden üblicherweise einfach auf die graphische Oberfläche von ***Lancelot*** gelegt. Es ist möglich (wenn auch nicht notwendig), Prozesse über ein „Linienobjekt“ miteinander zu verbinden. Wird eine Nachricht zwischen diesen Prozessen übermittelt, so bewegt sich ein (vorbestimmbares) Icon auf dieser Linie entlang. Dadurch kann das Kommunikationsverhalten eines Simulationssystems durch Animation demonstriert werden, was sich bei der Validierung eines Simulationsprogramms als sehr hilfreich erwiesen hat.

3.1 Komplexe Objekte in *Lancelot*

Dennoch lassen sich auf einer beschränkten graphischen Oberfläche nur eine beschränkte Anzahl von Prozessen übersichtlich anordnen, Steigt die Anzahl der Prozesse, und haben u.U. verschiedene solcher Prozeßsysteme die gleiche Struktur (d.h. gleiche Prozesse, die auf die gleiche Weise miteinander kommunizieren), so wäre ein häufiges Kopieren und erneutes Hinschreiben dieser Prozesse relativ umständlich und nicht sehr übersichtlich. Aus diesem Grund stellt ***Lancelot*** ein weiteres Strukturierungskonzept zur Verfügung, welches sehr viel mächtiger ist, das komplexe Objekt.

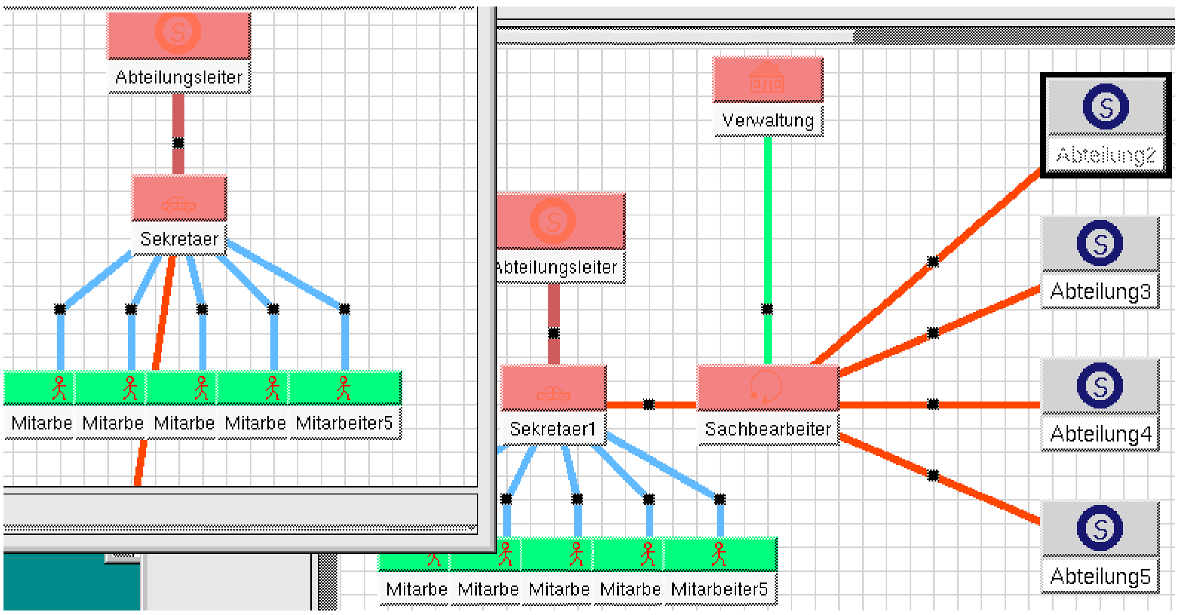

Bild 3-1: Modellhierarchie in einem komplexen Objekt

Ein komplexes Objekt kann selbst eine beliebig große Anzahl von Prozessen sowie deren Verbindungsstruktur enthalten. Es kann jedoch zugleich auch selbst wieder komplexe Objekte enthalten. Da ein komplexes Objekte eine eigene lokale Umgebung darstellt, können die komplexen Objekte eine beliebige Hierarchie erzeugen, so daß auf diese Weise auch sehr komplexe Systeme modelliert werden können. Dabei kann die Modellierung „Top-Down" oder „Bottom-Up" durchgeführt werden, d.h. entweder zunächst die grobe Struktur festgelegt und die einzelnen komplexen Objekte dann jeweils verfeinert werden, oder umgekehrt die einfachsten Prozesse programmiert und dann zu komplexen Objekten zusammengefaßt werden. Ebenso kann natürlich auch eine mittlere Position eingenommen, also eine „Middle-Up-Down"-Programmierung durchgeführt werden, was wegen der Wiederverwendbarkeit gegebener Prozesse oder komplexer Objekte eine naheliegende Strategie sein dürfte.
Wurden bereits Prozesse oder komplexe Objekte definiert, so können diese jederzeit in dem Objekt-Manager abgelegt und wiederverwendet werden. Dieses hat sich besonders bewährt,

um Ausgabe- oder Statistikprozesse an verschiedenen Stellen zu benutzen, bzw. abhängig vom Fortschritt der Programmierung auch leicht wieder zu entfernen. Das Bild 3-1 zeigt ein Beispiel, welches einen Geschäftsprozeß modelliert. Dargestellt sind verschiedene Abteilungen, die sich Information, in der Regel Geschäftsunterlagen, zusenden. Bei einer gegebenen Bearbeitungszeit an den einzelnen Standorten kann die Belastung der Sachbearbeiter ermittelt werden. Dieses kann mit der bei der Modellierung erstellten Graphik leicht präsentiert und sofort veranschaulicht werden.

Das Beispiel zeigt jedoch recht deutlich, daß die große Anzahl von Mitarbeitern eine übersichtliche Darstellung kaum möglich macht, zumal diese auch kaum erforderlich sein dürfte. Daher genügt es, ganze Abteilungen durch ein Icon zu repräsentieren, welches bei Bedarf natürlich beliebig „aufgeklickt“ werden kann, wie im Bild 1-1 für eine Abteilung (Abteilung2) beispielhaft gezeigt. Es dürfte klar sein, daß auf diese Weise Modelle sehr viel strukturierter entwickelt und natürlich auch gewartet werden können, als wenn alle Prozesse flach auf der gleichen Ebene liegen würden.

3.2 Namensräume in komplexen Objekten von *Lancelot*

Die Prozesse in den einzelnen komplexen Objekten werden jeweils durch ein geeignetes Namensschema angesprochen, indem die Namen der komplexen Objekte bzw. der Prozesse konkateniert werden. Innerhalb eines komplexen Objekts herrscht ein eigener Namensraum, so daß die Prozesse sich hier direkt ansprechen können.

Zusätzlich können in komplexen Objekten Warteschlangen und „lokale“ Variable angelegt werden, so daß die Prozesse implizit stets auf diese Namen zugreifen. Da jede Umgebung ihren eigenen Namensraum besitzt, entsteht eine Hierarchie von Umgebungen, ähnlich den Blockumgebungen in Programmiersprachen wie Algol60 oder C. Der Unterschied besteht natürlich darin, daß diese Sprachen die aktuellen Speicherbereiche ständig dynamisch auf- und abbauen, während in einer Simulationssprache diese ständig aktiv sein müssen.

Durch die Einführung geschützter Umgebungen wird die Vervielfältigung von komplexen Objekten ermöglicht, ohne Namenskonflikte zu erzeugen. Dadurch kann die Instanz eines komplexen Objekts beliebig vervielfältigt werden, ohne irgendwelche Umbenennungen von Prozeß-, Warteschlangen- oder Variablennamen durchführen zu müssen. Dieses ermöglicht es somit sehr einfach, bereits vorhandene komplexe Objekte, die beispielsweise im Objekt-Manager gespeichert wurden, wiederzuverwenden.

Eine weitere Möglichkeit, Prozesse zu vervielfältigen, besteht darin, zu einem Prozeß sogenannte „Childs“ zu erzeugen, die die gleiche Struktur wie der „Vater-Prozeß“ haben, jedoch eine eigene Instanz darstellen. Auf diese Prozesse kann durch eine Indizierung, also einen berechneten Namen zugegriffen werden, ähnlich einem Feldzugriff, so daß die Anzahl von Prozessen auf diese Weise dynamisch verändert werden kann. Diese Prozesse werden übrigens nicht in der graphischen Oberfläche angezeigt, so daß dieses deren Handhabung stark vereinfacht.

4 Graphische Darstellung von Ereignissen in *Lancelot*

Es wurde bereits darauf hingewiesen, daß ***Lancelot*** gewisse Ereignisse an der Oberfläche optional anzeigt, so daß beispielsweise während der Entwicklung eines Simulationsmodells einfach überprüft werden kann, ob sich das Programm erwartungsgemäß verhält. Dieses kann jedoch jederzeit ausgeschaltet werden, ohne daß im Programmtext irgendeine Veränderung durchgeführt werden müßte, so daß dieses keine mühseligen und fehleranfälligen Eingriffe im Programm nach sich ziehen würde; entsprechend kann die Animation jederzeit wieder eingeschaltet werden, so daß für spätere Entwicklungen einfach erneut Überprüfungen oder Präsentationen vorgenommen werden können.

4.1 Animation, Icons und Farben in *Lancelot*

Die Animation bewirkt die Bewegung von frei wählbaren Icons entlang der Verbindungslinien während des Sendens einer Nachricht von einem Prozeß zu einem anderen; die Geschwindigkeit dieser Bewegung kann während der Laufzeit verändert oder die Animation ganz abgeschaltet werden. Außerdem kann der Zustand der Prozesse durch eine farbige Markierung dargestellt werden. Auch hierdurch wird der „innere" Zustand eines Prozesses einfach erkennbar, und es kann beispielsweise festgestellt werden, ob ein bestimmter Prozeß überhaupt jemals aktiv wird.
Da die Prozesse mit individuellen Icons und Farben versehen werden können, sind diese leicht voneinander, außer durch ihren Namen, zu unterscheiden. Die Verbindungsobjekte können zum einen farblich, zum anderen aber auch in ihrer Form unterschiedlich dargestellt werden. So gibt es unterschiedlich breite Linien, oder es können Strichmuster definiert werden. Die Verbindungsobjekte werden übrigens durchgehend in ein komplexes Objekt geführt, und animierte Icons, die in ein komplexes Objekt laufen, werden entsprechend im inneren des komplexen Objekts weitergeführt. Dadurch kann der Ablauf einer Simulation sehr detailliert verfolgt werden.

4.2 Graphische Ausgabe in *Lancelot*

Während der Simulation ist es möglich, verschiedene Information über den Zustand eines Prozesses zu erlangen. So können beispielsweise gewisse Werte definiert werden, welche beim Anklicken eines Prozeß-Icons mit der mittleren Maustaste angezeigt werden. Außerdem kann der Programmierer ein eigenes Graphikfenster öffnen, in das er Icons zeichnen und diese bewegen kann, und darüber hinaus eine Reihe anderer geometrischer Objekte wie Linien, Kurven, Kreise usw. darstellen. Diese können natürlich auch farbig differenziert werden. Auf diese Weise lassen sich beispielsweise bewegte Objekte darstellen.
Eine weitere Möglichkeit der graphischen Darstellung insbesondere quantitativer Ergebnisse besteht darin, statistische Ausgabefunktionen zu verwenden. Hier gibt es die sogenannten Zeitreihen, die eine oder mehrere Wertefolgen in einem geeigneten Diagramm darstellen. Da dieses dynamisch während der Laufzeit geschieht, kann der Programmierer anhand dieser Werte sehr schnell feststellen, ob sein Programm die gewünschten Ergebnisse produziert. Beispielsweise lassen sich hiermit die Werte, die ein Zufallszahlengenerator erzeugt, anzeigen oder die Wartezeiten in einer Warteschlange. Während des Simulationslaufs besteht darüber hinaus auch die Möglichkeit, die Wertedarstellung zu verändern und beispielsweise Histogramme (siehe Bild 5-1) oder Linienverläufe in unterschiedlichen Farben auszugeben. Dieses erlaubt eine sehr detaillierte Anzeige der Ergebnisse, die auch zur Präsentation vorteilhaft verwendet werden können.
Natürlich gibt es auch die üblichen Standardverfahren zur statistischen Analyse, wie die Aufnahme von Mittelwerten oder anderen Momenten, oder die Ermittlung einer Verteilungsfunktion (siehe Bild 5-2). Die statistischen Verfahren von ***Lancelot*** bieten darüber hinaus die

Möglichkeit, mit Hilfe eines Teiltestverfahrens Konfidenzintervalle zu ermitteln, so daß auch über die Genauigkeit der Ergebnisse eine Aussage gemacht werden kann. Beispielsweise könnte nach jedem Teiltest überprüft werden, ob die gewünschte Genauigkeit für die Ermittlung einer statistischen Größe erreicht ist und gegebenenfalls die Simulation beendet werden. Diese Konzepte sollen hier jedoch nicht im Detail dargestellt werden, da sie im wesentlichen dem Stand der Technik entsprechen.

5 Technische Angaben zu Lancelot

5.1 Ausführungstechnik und Programmiersprache

Lancelot verwendet für die sprachlichen Konstrukte die bereits früher entwickelte Simulationssprache C-SIM. Ein in Lancelot formuliertes Simulationsprogramm wird in Standard-C übersetzt und dieses dann compiliert. Die Ausführung einer Phase dauert also im Prinzip nicht länger als die Ausführung folgender C-Anweisungen:

- ein Befehl `switch(PhasenNummer){...`
- Ausführung aller C-Anweisungen in der Phase,
- Einsortierung des Prozesses in die Zeitliste.

Wird eine animierte Darstellung verwendet, so verlangsamt sich die Zeit der Ausführung natürlich, da die Animation der Inspektion des Programmverhaltens durch den Programmierer dient. Die Animation kann interaktiv oder wahlweise vor der Compilierung ausgeschaltet werden, so daß die reine Laufzeit eines compilierten Standard-C-Programms übrig bleibt.

5.2 Ausführungszeiten

Um die Leistungsfähigkeit des Simulationssystems zu bewerten, wurden die folgenden Meßdaten erhoben, wobei die Simulation auf einem 133 MHz-Pentium-Prozessor mit 40 Mbyte Hauptspeicher unter einem Linux-Betriebssystem durchgeführt wurde.
Ohne Graphik und Statistik wurde für ein einfaches M/M/1-Wartesystem eine Laufzeit von 10 s für 1 Million Aufträge benötigt (erzeugen, versenden an Server, bedienen im Server, vernichten, also drei Phasen); das sind etwa 100.000 Aufträge pro Sekunde. Wird eine einfache Mittelwertstatistik (1. und 2. Momente) durchgeführt, so erhöht sich die Laufzeit auf 20 s; also erreichen wir 50.000 Aufträge pro Sekunde. Schließlich wurde eine Statistik mit der Erhebung der Verteilungsfunktion (Dichte der Wartezeiten) durchgeführt (siehe Bild 5-2). Dazu wurde eine Teilteststatistik erhoben mit jeweils 10.000 Aufträgen je Teiltest sowie 200.000 Sekunden Simulationszeit, also ca. 20 Teiltests. Wir kommen hierbei auf etwa 20.000 Aufträge pro Sekunde, was auf die aufwendigen Berechnungen sowie die aufwendige graphische Ausgabe während der Simulation zurückzuführen ist.
Um den Einfluß der Anzahl der Prozesse auf die Ausführungszeiten zu messen, verwenden wir mehrere einfache Bediensysteme, durch welche die Aufträge sequentiell hindurchlaufen. Bei 10 Bedienern bearbeitet Lancelot ca. 42.000 Aufträge pro Sekunde, bei 50 Bedienern ca. 7.700 Aufträge pro Sekunde und bei 100 Bediener ca. 3.200 Aufträge pro Sekunde.
Insgesamt ist Lancelot damit ausreichend schnell für eine große Anzahl von Anwendungen.

5.3 Ein- und Ausgabe von Lancelot

Die Eingabe von Parametern geschieht in der Regel durch Initialisierung der jeweiligen Variablen, oder durch eigene Ein- Ausgabeboxen für Abfragen, Auswahl, File-Menüs usw. Die Ausgabe kann durch normale Textausgabe auf die Kontrolloberfläche geschehen, durch automatisches Anhängen der Ergebnisse an eine Ausgabedatei oder durch graphische Ausgaben in eigene Fenster. Im Bild 5-2 ist als Beispiel die Ausgabe einer Verteilungsfunktion für Wartezeiten angegeben, die während einer Simulation ausgegeben werden kann; Bild 5-1 zeigt entsprechend die Ausgabe einer Folge von Werten in Form eines Histogramms. Am Ende der Simulation werden automatisch ähnliche Diagramme ausgegeben, welche Konfidenzintervalle für die Verteilungsfunktion sowie die Dichte (als komplementäre Dichtefunktion) angeben. Sämtliche Daten können in vielfach veränderter Form dargestellt werden, insbesondere mit logarithmischer Skalierung, mit anderen Farben oder Strichelungen usw., so daß auch sehr unterschiedliche Wertebereiche und Bedeutungen der Werte sehr verständlich veranschaulicht werden können.

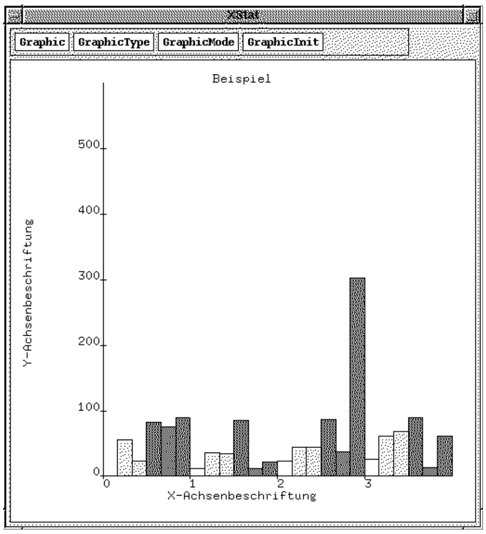

Bild 5-1: Ausgabe von zeitlichen Folgen in ***Lancelot***

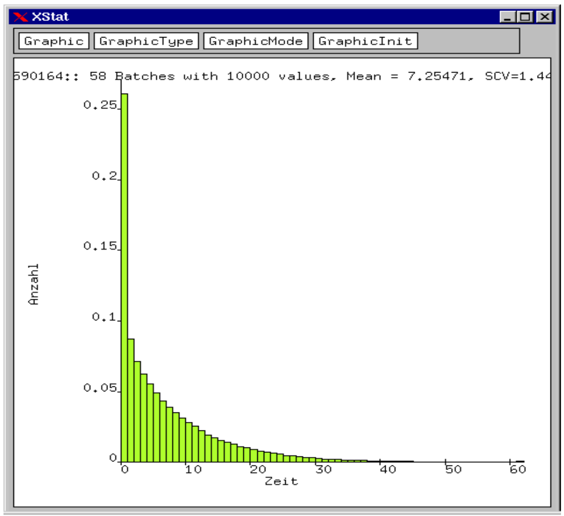

Bild 5-2: Ausgabe einer Verteilungsfunktion während der Simulation

5.4 Einsatz von C-Sim und Lancelot

C-Sim wurde bereits in mehreren Projekten eingesetzt, u.a. in einer Simulationsstudie für den DFN-Verein (3) sowie Dissertationen. In diesen Projekten wurden Modelle mit mehr als eintausend Prozessen erstellt, die unter C-Sim noch mit guten Laufzeiten simuliert und ausgewertet werden konnten. Tatsächlich sind die statistischen Bewertungen (Berechnung von Konfidenzinterfallen aus einer Teilteststatistik) in der Regel aufwendiger als die reinen Simulationszeiten.

Gegenwärtig wird Lancelot im Rahmen eine Disseration zum Performance-Management von ATM-Netzen eingesetzt sowie für die Modellierung eingebetteter Kontrollsysteme.

5.5 Entwicklungsumgebung

Das System Lancelot wurde auf Linux entwickelt und benötigt eine X-Windows-Oberfläche sowie einen C-Compiler. Es kann mit entsprechenden Bibliotheken auch auf anderen UNIX-basierten Plattformen laufen.

6 Schlußbetrachtungen

Obgleich ***Lancelot*** als Simulationskonzept entwickelt wurde, eignet es sich auch als prozeßorientiertes Programmierkonzept mit objektorientierten Ergänzungen. Eine Analyse graphischer Oberflächen, die in diesem Paper aus Platzmangel nicht wiedergegeben werden konnte, zeigt, daß ***Lancelot*** viele Anforderungen an graphische Benutzeroberflächen, insbesondere die Strukturierung komplexer Information durch komplexe Objekte, erfüllt. Durch die Aufnahme weiterer Konzepte, wie die kontinuierliche Simulation, ist ***Lancelot*** auch imstande, in sehr viel weitergehenden Anwendungen eingesetzt zu werden.
Ein weiterer Vorteil von ***Lancelot*** ist ein sehr einfaches Laufzeitsystem, so daß in dieser Sprache erstellte Simulationsprogramme leicht auf jeden anderen Rechner portiert werden können, ohne daß zusätzliche Anpassungsarbeiten nötig wären. Daher eignet sich ***Lancelot*** auch sehr gut dazu, eingebettete Kontrollsysteme zu entwickeln, zumal hier die gleiche Oberfläche für das zu regelnde System wie auch für das Regelsystem verwendet kann, was die Entwicklung solcher Systeme sicherlich sehr vereinfacht.

7 Danksagungen

Ich möchte mich insbesondere bei den Herren Torsten Sterzenbach and Gerd Block bedanken, die die Oberfläche (Sterzenbach) und den Simulationscompiler (Block) im Rahmen ihrer Diplomarbeiten entwickelt haben, und auch viel zum Konzept beigetragen haben.

8 Referenzen und Literatur

[1] W.P.Kowalk, 1992: „Das Simulationssystem C-Sim.“ Internal report, University of Oldenburg, Oldenburg.
[2] W.P.Kowalk, 1993: „Simulation in C-Sim. Principles of modeling, simulation techniques, and simulation tools.“ Internal report, University of Oldenburg, Oldenburg.
[3] W.P.Kowalk:, 1994 „X.25-PLP in höheren Geschwindigkeitsklassen“ DFN-Mitteilungen 1994.
[4] W.P.Kowalk:, 1996 „High-Precision simulation technique“ Internal report of the University of Oldenburg.
[5] W.P.Kowalk:, 1997 „Distributed Development of Embedded Systems“ CEE97 (1997)..
[6] Sterzenbach, und G. Block, 1996: „Das Simulationssystem ***Lancelot.***“ Internal report, University of Oldenburg, Oldenburg.