Methoden zur Integration von ereignisorientierter Simulation und Presentation-Managern*

Ulrich Freund
TU-Ilmenau†
Fakultät
Informatik und Automatisierung
Fachgebiet
System- und Steuerungstheorie

Timm Grams
FH Fulda ‡
Fachbereich
Elektrotechnik

Zusammenfassung

Ereignisorientierte Systeme lassen sich mit der objektorientierten Programmierung elegant modellieren. Sollen die Simulationsdaten während eines Simulationslaufes angezeigt werden, sind die Simulationsprogramme entsprechend zu entwerfen. Am Beispiel von Telekommunikationsystemen (Token-Bus, FDDI) wird eine Methodik zur Integration von Simulationsmodell und Presentation-Manager vorgeführt.

1 Einleitung

Die ereignisorientierte Simulationstechnik ist ein leistungsfähiges Werkzeug, um das dynamische Verhalten von komplexen technischen Systemen zu studieren bzw. solche Systeme zu dimensionieren. Typische Anwendungen sind Telekommunikations- und Straßenverkehrsnetze, Rechnersysteme und Fertigungsanlagen. Die zu untersuchenden Effekte müssen grafisch darstellbar sowie zeitlich und räumlich beliebig genau auflösbar sein.

2 Ereignisorientierte Simulationsprogramme

Ereignisorientierte Simulationsprogramme lassen sich elegant mittels der objektorientierten Programmierung implementieren. Sie haben als zentralen Bestandteil eine Ereigniswarteschlange. Diese Warteschlange wird von einer Schleife abgearbeitet, welche das jeweils erste Ereignis auskettet, die dem Ereignis zugeordnete Funktion ausführt und die Simulationszeit auf die letzte Ereignisankunftszeit setzt. Ein Beispiel für eine solche Klassenbibliothek ist die Klasse **Simulation** von SIMULA 67.

Wir haben in Turbo-Pascal eine Klassenbibliothek implementiert, mit der sich ereignisorientierte Programme modellieren lassen [?]. Diese Bibliothek besteht aus einer doppelt verketteten Liste, welche sich vielfach verwenden läßt. Von den Elementen, die in die Liste einkettbar sind (Klasse **link**), wird die Klasse **process** abgeleitet. Diese Klasse stellt die Simulationsprozesse

*Als Presentation-Manager werden Programme wie MS-Windows oder Motif bezeichnet
†PSF 327, 98684 Ilmenau
‡Marquardstr. 35, 36039 Fulda

dar, deren Objekte als Ereignisse in die Ereigniswarteschlange eingekettet werden. Das Verhalten eines Prozesses wird durch die virtuelle Methode **Run** beschrieben. Die Schleife des Simulationsprogramms (Methode **Runsim**), welche die Ereigniswarteschlange (in Anlehnung an SIMULA 67 SQS genannt) abarbeitet, hat folgendes aussehen:

```
while not SQS^.empty do
begin
  help:=SQS^.first;
  SQS^.out(help);       {Kette erstes Element der SQS aus}
  simtime:=help^.time;  {Schreibe Simulationszeit fort}
  help^.run;            {Arbeite das Simulationsereignis ab}
end;
```

3 Ein Tokenbus als Beispiel

Man kann ein Token-Netzwerk als verteiltes Wartesystem ansehen. Es besteht aus Warteschlangen und einer Bedienstation. Beim Tokenbus werden die Warteschlangen durch die Stationen dargestellt, das Token ist die Bedienstation mit dem Token-Protokoll als Bedienstrategie.

Das Eintreffen der Nachrichten sind Ereignisse, die als Nachkommen der Klasse **process** definiert sind. Stationen werden in einer Liste verwaltet und vom Grundtyp **link** abgeleitet. Das Token als Bedienstation ist von der Klasse **process** abgeleitet.

Abbildung 1: Klassenhierarchie für eine Token-Bus Simulation

4 Anforderungen an eine Simulationsdarstellung

Es ist für den Nutzer wichtig, die von der Simulation gelieferten Daten derart darzustellen, daß die gewünschten Informationen schnell und zielgerichtet erfaßt werden können. Diese

Information sollen in einer für den Benutzer gewohnten Computer-Umgebung erscheinen. Es erscheint daher sinnvoll, einen Presentation-Manager zu verwenden. Weitere Anforderungen sind:

- Eine Simulationsdarstellung muß an beliebiger Stelle vom Benutzer unterbrechbar sein, um auf Maus- oder Tastatureingaben zu reagieren.
- Da ein Mensch nur eine endliche Anzahl von Informationen verarbeiten kann, müssen Möglichkeiten zur Fokussierung der Simulationsparameter gegeben sein, z. B. Aggregation mehrerer Simulationsparameter in einer Darstellung.
- Das simulierte System sollte aus verschiedenen Perspektiven betrachtbar sein. Eine kann z.B. die Darstellung von verschiedenen Systemparametern sein, die andere die Verfolgung von bestimmten Ereignissen
- Um eine kontinuierliche Darstellung der Simulationsparameter zu erreichen, sollte sie unabhängig von der Abarbeitung der Simulationsobjekte sein. Dazu folgende Beispiele:
  - Bei einer FDDI-Simulation soll das Timerverhalten einiger Stationen dargestellt werden. Da z.B. das Ablaufen der Timer durch eine Gerade dargestellt wird, sollte es auch in der Simulationsdarstellung als fortlaufende Gerade erscheinen. [?].
  - Bei einer Simulation von Mobilen-Satellitenkommunikationssystemen ist der Aufenthaltsort der Satelliten im Weltraum ein kontinuierlicher Prozeß - er sollte auch so erscheinen [?].

Ebenso, wie eine physikalische Messung das Meßobjekt nicht beeinflußen sollte (es aber trotzdem in vernachläßigbarem Umfang macht), ist es wünschenswert, wenn die Anweisungen der Darstellung vom Programmkode der Simulation getrennt sind, um den Kode für das Simulationsmodell lesbar zu halten.

5 Arbeitsweise eines Presentation-Managers am Beispiel von Object-Windows

Programme, die auf einem Presentation-Manager aufsetzen, werden auch ereignisorientiert genannt. Der Grund dafür ist die Arbeitsweise von Presentation-Managern. Als Ereignisse werden Tastatureingaben und Mausklicke bezeichnet. Diese werden per Interrupt in ein Record verpackt, welches in eine Eingabewarteschlange eingekettet wird. Der Presentation-Manager verteilt diese Ereignisse nun an die Warteschlange der dem Ereignis zugeordneten Applikation. Die Applikation fragt in einer Schleife (die **Message-Loop**) nach Ereignissen, um danach zu reagieren. Abhängig von der Implementierung werden dabei Case-Anweisungen durchlaufen oder es wird in Tabellen nach den Funktionen gesucht, welche auf das Ereignis reagieren sollen. Diese Funktionen werden **Callback-Functions** genannt.

Klassenbibliotheken für Presentation-Manager haben oft folgendes Aussehen:

- Initialisierungen und die Message-Loop werden in eine Klasse verpackt. Bei Object-Windows heißt diese Klasse **Tapplication**. Die Adresse des Objektes sind dem Programm zugänglich.
- Das Hauptfenster einer Applikation wird in einer virtuellen Methode der Klasse Tapplication initialisiert. Der Anwender überschreibt diese Methode, um sein eigenes Fenster zu initialisieren.

- Hat eine Applikation mehrere Unterfenster, so werden diese in einer Liste verwaltet.

Da MS-Windows ein kooperratives "Betriebssystem" ist, läßt es kein echtes Multitasking zu. Wird eine Callback-Function abgearbeitet, hat der Anwender keine Möglichkeit mehr, irgendetwas anderes zu machen. Dies ist erst nach der Abarbeitung wieder möglich. Es wird von Microsoft daher die Empfehlung gegeben, die Callback-Functions möglichst kurz zu halten.

6 Die Integration von ereignisorientierter Simulation und Presentation- Managern

6.1 Probleme...

Das Simulationsprogramm besteht aus einer großen Schleife, ebenso das Programm eines Presentation-Managers. Zur Integration von beiden Schleifen bieten sich zwei Möglichkeiten an:

- Die Message-Loop wird modifiziert und arbeitet - solange kein Ereignis von der Tastatur oder der Maus vorliegt - Simulationsereignisse ab. Maus- oder Tastaturereignisse werden vorrangig bearbeitet.
- Die Simulation wird in einer Callback-Schleife abgearbeitet.

Beide Methoden haben Nachteile. Bei der ersten geht eine Trennung von Simulation und Darstellung verloren. Nicht bei jedem Presentation-Manager ist es möglich, die Message-Loop zu verändern. Bei MS-Windows wird die Programmkontrolle an eine andere Applikation weitergegeben, wenn keine Maus- oder Tastaturereignisse vorliegen. Die zweite Methode verstößt gegen die Empfehlung von Microsoft, nur kurze Callback-Functions zu schreiben. Wird eine Callback-Routine abgearbeitet, können Maus- oder Tastaturereignisse nicht abgearbeitet werden - ein Veränderung der Simulationsdarstellung wäre so unter MS-Windows nicht möglich. Es kommt noch ein anderes Problem hinzu, nämlich das der Ausgabe des Simulationszustandes. Es könnte innerhalb eines jeden Simulationsobjektes der jeweilige Simulationszustand durch eine dem Presentation-Manager angepaßte Ausgabefunktion dargestellt werden. Diese Vorgehensweise hat viele Nachteile:

- Es ist nur eine endliche Anzahl von Simulationsdarstellungen möglich.
- Der Wert der Simulationsparameter ist von der jeweiligen Platzierung der Ausgabeanweisung abhängig.
- Die Basisklassen der Simulation müßten geändert werden, da sowohl Klassen des Presentation-Managers als auch die Simulationsklassen von eigenen Basisklassen abgeleitet werden. (Mehrfacherbung würde zwar dieses Problem lösen - es träten dann aber andere Probleme auf).
- Für die Änderung des Presentation-Managers müßte der gesamte Simulationskode neu entwickelt werden.
- Das Programm wird schwer lesbar und damit wartungsunfreundlich.

6.2 ...und Lösungen

Wir schlagen eine Trennung von Simulation und Darstellung vor. Jedem Simulationsobjekt, welches dargestellt werden soll, wird ein Darstellungsobjekt zugeordnet. Die Darstellungsobjekte sind Objekte des Presentation-Managers. Die Darstellung von Simulationszuständen wird als Simulationsview bezeichnet.

Das Simulationsprogramm wird aufgeteilt in Prozeduren für die Initialisierung, der eigentlichen Simulation und dem Vernichten der Simulationsobjekte nach Simulationsende. Diese Prozeduren werden Callback-Functions zugeordnet. Um die Callback-Funktion für die eigentliche Simulation - die Abarbeitung der Message-Loop - nicht zu lang werden zu lassen, wird die Schleife modifiziert. Es wird bei den Basisklassen der Simulation in der Run-Methode der Simulation eine virtuelle Funktion eingeführt, welche feststellt, ob ein Maus- oder Tastatureignis vorliegt. Ist das der Fall, liefert diese Funktion den Wert **true** und die Abarbeitung der Simulationsereignisse wird unterbrochen. In der Basisklasse liefert diese Funktion immer den Wert **false**, da dort keine Darstellung vorgenommen wird. Die while-Schleife der **Runsim**-Methode hat nun folgendes aussehen:

```
while (not SQS^.empty) and (not simstop) do
```

Bleibt noch die Frage, wie die Verbindung zwischen Presentation-Manager und der Simulation vorgenommen wird. Es wird ein spezieller Simulationsprozeß eingeführt, dessen einzige Aufgabe es ist, den Presentation-Manager zur Ausgabe zu veranlassen.

Dieses Ereignis, ein Nachkomme der Klasse **process** mit einer Referenz auf das Hauptfenster des Presentation-Managers, wird in bestimmten Abständen in die Simulation eingeplant. Bei seiner Abarbeitung wird die Draw-Methode des Hauptfensters aufgerufen. Die Draw-Methode des Hauptfensters ruft nacheinander die Draw-Methoden der Unterfenster auf. Diese Unterfenster sind jeweils einem Simulationsobjekt zugeordenet und benutzen beim Neuzeichnen die aktuellen Parameterwerte der Simulationsobjekte. Das folgende Bild zeigt die Trennung zwischen Simulation und Darstellung.

Die Vorteile dieser Vorgehensweise sind:

- Simulation und Darstellung sind voneinander entkoppelt.
- Die Programmierung der Simulation ist entlastet von Darstellungsdetails.
- Der Presentation-Manager ist austauschbar.
- Es lassen sich beliebige Simulationsviews problemlos während eines Simulationslaufes hinzufügen.

6.3 Praktische Anwendung

Wir haben mit dieser Methodik das Timed-Token-Protokoll eines FDDI-Rings entwickelt. Diese Simulation demonstriert das Timerverhalten einer FDDI-Station in Abhängigkeit vom Netzzustand. Diese Simulation wird sowohl in der Lehre als auch in kommerziellen Schulungseinrichtungen eingesetzt.

7 Ausblick

Bei der nach beschriebener Methode entwickelten FDDI-Ring-Simulation sind die Möglichkeiten des Eingriffs in die Simulation sowie Art und Anzahl der Darstellungen vom Entwickler fest vorgegeben.

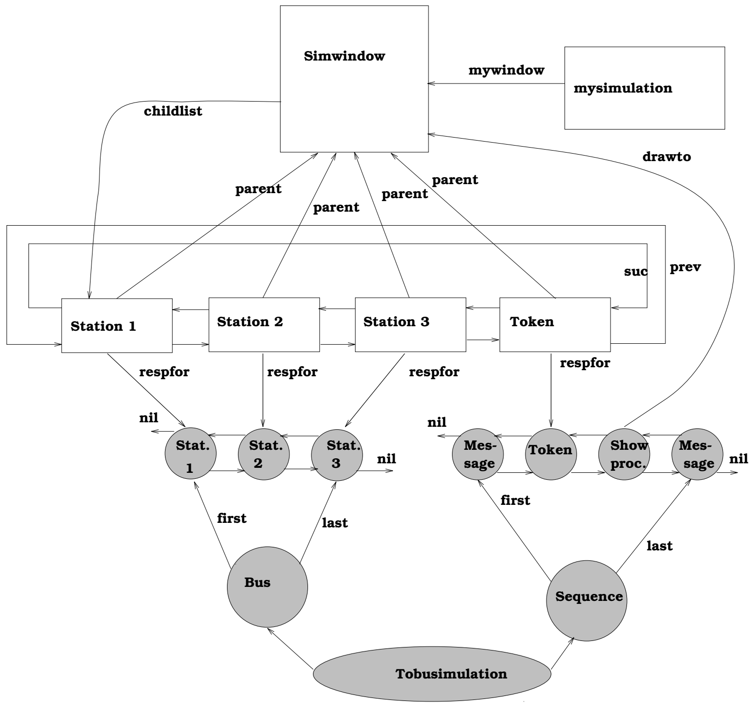

Abbildung 2: Das Gesamtsystem für einen Tokenbus. Es werden die Wartelschlangengröße in den Stationen und die Aufenthaltstation des Tokens angezeigt

Um eine größere Flexibilität in Darstellung und Manipulation von Simulationsdaten erreichen, wären für den Anwender zugängliche Schnittstellen zwischen Simulationsmodell und Darstellungssystem wünschenswert. Eine solche Schnittstelle wird Parameterraum genannt. Dort sind die Daten des Simulationsmodells zugänglich und manipulierbar. Neben definierten Zugriffsverfahren, welche Simulationsdaten in einen solchen Parameterraum schreiben, sollten Werkzeuge vorhanden sein, die in einer einfachen Sprache beliebige Darstellungen der Simulationsdaten ermöglichen.

Literatur

[1] Timm Grams. Simulation - strukturiert und objektorientiert programmiert, BI-Wissenschaftsverlag, Mannheim 1992

[2] Ulrich Freund,Simulation von Hochgeschwindigkeits-Kommunikationsprotokollen durch objektorientierte CBT-Software, Diplom-Arbeit FH Fulda, 1992

[3] Horst Salzwedel, BONeS-Satlab User's Guide ,Commdisco System Foster-City California, 1992

[4] Wedekind,Unterrichtsmedium Computersimulation Lexikaverlag, Weil der Stadt 1981