Hierarchische Verhaltensbeschreibung in objekt-orientierten Systemmodellen - eine Grundlage für modellbasiertes Prototyping

Martin Glinz
ABB Informatik AG, CH-5401 Baden

Zusammenfassung

Der Beitrag beschreibt ein objekt-orientiertes Modell für Spezifikation und Entwurf von Systemen, das die Generierung von Prototypen gestattet. Das Verhaltensmodell basiert auf Zustandsautomaten, welche in eine 'Ist-Teil-von'-Hierarchie eingebettet sind. Zusammen mit den Grundprinzipien der objekt-orientierten Modellierung (vor allem Vererbung und Benutzungs-Abstraktion) ergibt sich eine sehr attraktive Kombination von Anschaulichkeit, Ausdruckskraft und Durchgängigkeit der Modellkonzepte einerseits und von Beobachtbarkeit und Erprobbarkeit des Systemverhaltens andererseits.

Ein solches modellbasiertes Prototyping hat gegenüber herkömmlichen Prototypen den Vorteil, daß ein Modell anschaulicher, besser verstehbar und leichter änderbar ist als der Code eines Prototyps.

1 Einleitung

Eine attraktive Variante des Prototyping besteht darin, einen Prototyp nicht zu programmieren, sondern ihn aus der (in Form eines Modells vorliegenden) Spezifikation zu generieren. Modellbasiertes Prototyping hat u.a. folgende Vorteile:

- Ein Modell, welches über geeignete Abstraktionsmechanismen verfügt, eignet sich sehr gut zur Veranschaulichung von Strukturen und Zusammenhängen in einem System. Ein programmierter Prototyp dagegen kann hierzu höchstens indirekt (durch Beobachten und Analysieren seines Verhaltens) Aussagen machen.
- Modelle sind einfacher änderbar als Code, da die Auswirkungen einer Änderung besser überblickbar sind.

Erschienen in: H. Züllighoven / W. Altmann / E.E. Doberkat (Hrsg.):
Requirements Engineering '93: Prototyping, Stuttgart: Teubner 1993.

- Nach Abschluß des Prototyping steht mit dem Modell eine saubere und gut verstehbare Vorgabe für die Realisierung zur Verfügung.
- Häufig wird nur ein Teil eines Systems mit Prototypen erprobt. Bei codierten Prototypen führt dies zu einer oft schwierigen Koexistenz von durch den Prototyp gegebenen Anforderungen und konventionell spezifizierten Anforderungen. Bei modellbasiertem Prototyping dagegen bildet das Modell eine einheitliche Grundlage, in welche Prototypen organisch eingebettet sind.

Dieser Ansatz setzt geeignete Modelle für die Spezifikation des zu erstellenden Systems voraus. Diese müssen (neben einigen anderen Eigenschaften) insbesondere das System*verhalten*, nicht nur die Daten und Operationen modellieren. Die heute existierenden Modelle zur Beschreibung von Systemverhalten basieren meist auf einer Kombination von endlichen Automaten und Strukturierter Analyse mit einer geeignet festgelegten Semantik oder auf Petri-Netzen. Eine wesentliche Voraussetzung für die Benutzbarkeit solcher Modelle für reale Aufgaben aus der Praxis ist die Möglichkeit zur Bildung von Abstraktionen auch für die Verhaltensbeschreibung. Bei Petri-Netzen fehlen die Abstraktionsmöglichkeiten weitgehend. Bei den auf Strukturierter Analyse basierenden Modellen existieren die notwendigen Möglichkeiten zur Verhaltensabstraktion. Diese Modelle haben aber an anderen Orten schwerwiegende Schwachstellen (vgl. Glinz [3]).

Die objekt-orientierte Spezifikation überwindet die wesentlichen Schwachstellen der Strukturierten Analyse. Bisher sind aber die Möglichkeiten zur Verhaltensbeschreibung in objekt-orientierten Spezifikationsmethoden völlig unzureichend:

- Das Verhalten jedes Objekts kann zwar modelliert werden. Das Gesamtverhalten eines Systems ist aber nur durch Zusammensetzung des Verhaltens der einzelnen Objekte erkennbar. Abstraktionen in der Verhaltensbeschreibung fehlen. Dies liegt im wesentlichen daran, daß die heutigen objekt-orientierten Spezifikationsverfahren keine brauchbare 'Ist-Teil-von'-Abstraktion haben. ('Part-of'-Beziehungen ohne Abstraktion, z.B. bei Coad und Yourdon [1], nützen nicht viel.)
- Eine präzise Verhaltensbeschreibung wird ferner dadurch behindert, daß die bisher bekannten Modelle eine nur sehr schwammig definierte Semantik aufweisen.

Dieser Beitrag skizziert ein objekt-orientiertes Systembeschreibungsmodell, welches mit Hilfe der 'Ist-Teil-von'-Abstraktion auch eine Abstraktion der Verhaltensbeschreibung liefert. Die Semantik des Verhaltens ist wo nötig formal definierbar und erlaubt somit die Generierung von Prototypen. Gleichzeitig aber ist das Modell aufgrund seiner Abstraktionsmöglichkeiten wesentlich anschaulicher und leichter verständlich als

die bisherigen objekt-orientierten Ansätze. Das Modell eignet sich gleichermaßen für die Spezifikation wie für den (Architektur-)Entwurf von Systemen. In diesem Beitrag wird es jedoch nur als Spezifikationsmittel verwendet.

2 Grundelemente des Modells

Die Grundelemente des objekt-orientierten Systemmodells sind in Bild 1 anhand eines Ausschnitts aus einem Personalverwaltungs-Systems gezeigt und werden nachfolgend kurz erläutert.

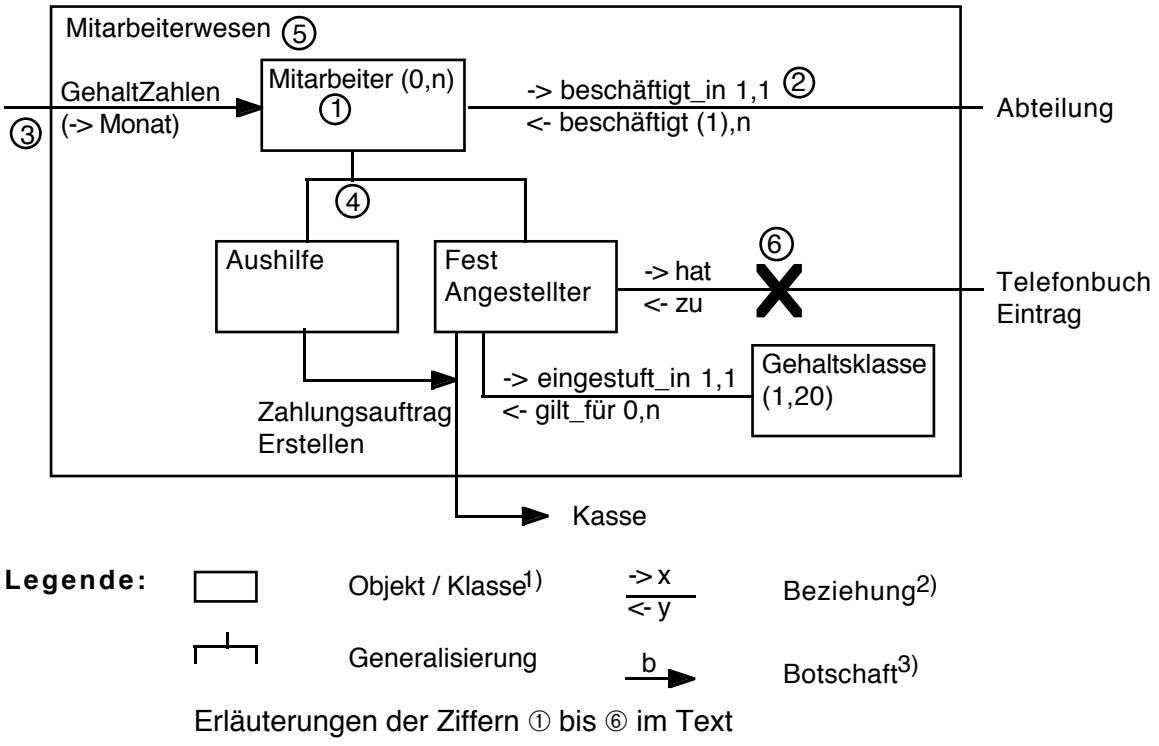

Bild 1: Grundelemente des objekt-orientierten Systemmodells

---

1) Das Zahlenpaar nach dem Namen gibt die Mindest- und Höchstzahl der Exemplare an. Fehlt diese Angabe und wird sie nicht von einer Oberklasse geerbt, so gibt es genau ein Exemplar.

2) Die Beschriftung einer Beziehung gibt an, was die in Beziehung gesetzten Objekte füreinander bedeuten. Die Pfeile geben die Richtung, die Zahlen Kardinalitäten an. Beispiel: Jeder Mitarbeiter ist beschäftigt_in genau einer (mind. 1, max. 1) Abteilung. Die Existenz eines Mitarbeiter-Objekts verlangt zwingend die Existenz einer solchen Beziehung. Umgekehrt beschäftigt jede Abteilung mehrere Mitarbeiter. Die Mindestzahl soll 1 sein, dies wird aber nicht erzwungen.

3) Den Botschaften können aktuelle Parameter mitgegeben werden. Ein Pfeil gibt die Kommunikationsrichtung an (Beispiel: Die Botschaft b (->x, <-y) hat den Eingabeparameter x und den Ausgabeparameter y).

Ein System wird als eine Menge von Objekten ①, die statisch und dynamisch untereinander zusammenhängen, modelliert. Mengen gleichartiger Objekte werden zu Klassen zusammengefaßt. Die Objekte einer Klasse werden durch ein Repräsentanten-Objekt im Modell dargestellt. Ein Objekt verkapselt Daten, Operationen und Zustandsinformationen. Über Beziehungen können statische Zusammenhänge ② zwischen Objekten modelliert werden[1]. Durch das Senden von Botschaften ③ kann ein Objekt dynamisch auf andere Objekte einwirken. Diejenigen Objekte, die mit den dargestellten Objekten in Beziehung stehen oder von ihnen Botschaften empfangen, sind im Diagramm mit ihren Namen aufgeführt. (Damit wird dargestellt, auf welche anderen Objekte die in einem Diagramm modellierten Objekte sich abstützen.) Eine Vererbungshierarchie in den Klassen ermöglicht die Modellierung einer Generalisierungs-Abstraktion ④. (Diese entspricht der Begriffs-Oberbegriffs-Hierarchie im menschlichen Denken und ermöglicht ein realitätsnahes Modellieren.) Das zeitliche Verhalten eines Objekts kann durch Zustandsautomaten beschrieben werden (In Bild 1 nicht vorhanden; wird in den folgenden Kapiteln genauer erläutert).

Spezifikationsmethoden, denen ein Modell mit den Elementen ① bis ④ in dieser oder ähnlicher Art zugrunde liegt, werden objekt-orientiert genannt. Die bekanntesten Ansätze sind die von Coad und Yourdon [1] und von Shlaer und Mellor [7], [8].

Über diese gemeinsamen Grundlagen hinaus enthält das hier vorgestellte Modell Mittel zur Bildung von Strukturabstraktionen (Objekte sind Bestandteil von anderen Objekten ⑤) und zum Information Hiding (ein Teil der Informationen eines Objekts ist von außen nicht zugänglich ⑥; z.B. kein Zugriff vom TelefonbuchEintrag eines Mitarbeiters über die Beziehungen zu und eingestuft_in auf die Gehaltsklasse dieses Mitarbeiters). Die Verhaltensbeschreibung ist im Gegensatz zu den bisherigen Modellen vollständig formalisierbar und dank der 'Ist-Teil-von'-Abstraktion nicht nur auf elementare Objekte beschränkt. Diese formale hierarchische Verhaltensbeschreibung, welche in den folgenden beiden Kapiteln genauer beschrieben wird, bildet den Schlüssel für die Generierung von Prototypen aus dem Modell.

---

1) Ein statischer Zusammenhang ist eine Beziehung, über welche ein Objekt auf ein anderes Objekt direkt zugreifen kann. Alle Attribute, Zustände und eingeschachtelten Objekte eines Objekts, die nicht als privat deklariert sind, können über Beziehungen von anderen Objekten aus gelesen oder verändert werden. Beziehungen zwischen zwei Objekten können einseitig sein oder in beiden Richtungen bestehen.

3 Hierarchische Verhaltensbeschreibung - Einführung anhand eines Beispiels

3.1 Das Beispiel-Problem

Als Beispiel wird die Spezifikation der Steuerung eines einfachen Fahrausweis-Automaten verwendet. Ein solcher Automat verfügt über mehrere Wahltasten für verschiedene Tarifstufen. Es können nacheinander mehrere Wahltasten gedrückt werden. Alle Wahlen werden registriert; der aufsummierte Preis aller Wahlen wird angezeigt. Nach dem Drücken der ersten Wahltaste wird der Münzschlitz entriegelt, und der geforderte Betrag kann bezahlt werden. Nach dem Einwurf der ersten Münze ist keine weitere Wahl mehr möglich. Wenn der ganze Betrag bezahlt ist, druckt das Gerät den (die) Fahrausweis(e) und gibt Wechselgeld. Eine Annulliertaste ermöglicht den Abbruch des Bedienvorgangs. Wird die Bedienung für mehr als 45 s unterbrochen, so löst das Gerät intern eine Annullierung aus (Timeout). Bei Aufbruchversuchen wird für 60 s ein Alarmhorn eingeschaltet. Danach geht das Gerät außer Betrieb. Im Inneren des Geräts gibt es Tasten zur In- und Außerbetriebsetzung des Geräts. Wenn das Gerät außer Betrieb ist, kann der Betreiber ab einem Datenträger Änderungen der im Gerät gespeicherten Tarife vornehmen.

Die Bilder 2 bis 6 zeigen einen Ausschnitt aus der Spezifikation der Software für dieses Gerät. Anhand dieser Bilder werden die Syntax und eine anschauliche Semantik des Modells erläutert.

3.2 Verhaltensbeschreibung in hierarchisch gegliederten Objekt-Diagrammen

Bild 2 beschreibt das Objekt Fahrausweisautomat_Steuerung mit einem Objekt-Diagramm. Dieses Objekt besteht aus dem Objekt Betrieb und den Objekten der Klasse Tarif. Die Beziehung zwischen Betrieb und Tarif beschreibt die Tarife, welche zu den von Betrieb registrierten Auswahlen gehören. Die Botschaft Verwenden dient zum Auf- bzw. Abbau dieser Beziehungen. Die drei Punkte nach dem Namen kennzeichnen Betrieb als ein Objekt, das durch ein weiteres Objekt-Diagramm spezifiziert ist, während Tarif nur noch eine Elementarbeschreibung aufweist.

Zusätzlich ist nun auch das globale Verhalten der Fahrausweisautomat-Steuerung spezifiziert. Hierzu dienen die im Diagramm durch gerundete Rechtecke dargestellten **Zustände** AußerBetrieb und Alarm sowie die durch gestrichelte Pfeile gekennzeichneten

möglichen **Zustandsübergänge**. Da es einen Zustandsübergang zum Objekt Betrieb gibt, repräsentiert dieses gleichzeitig auch einen (globalen) Zustand. Die Einzelheiten dieses Zustands sind in der Beschreibung des Objekts Betrieb (siehe Bild 3 unten) spezifiziert.

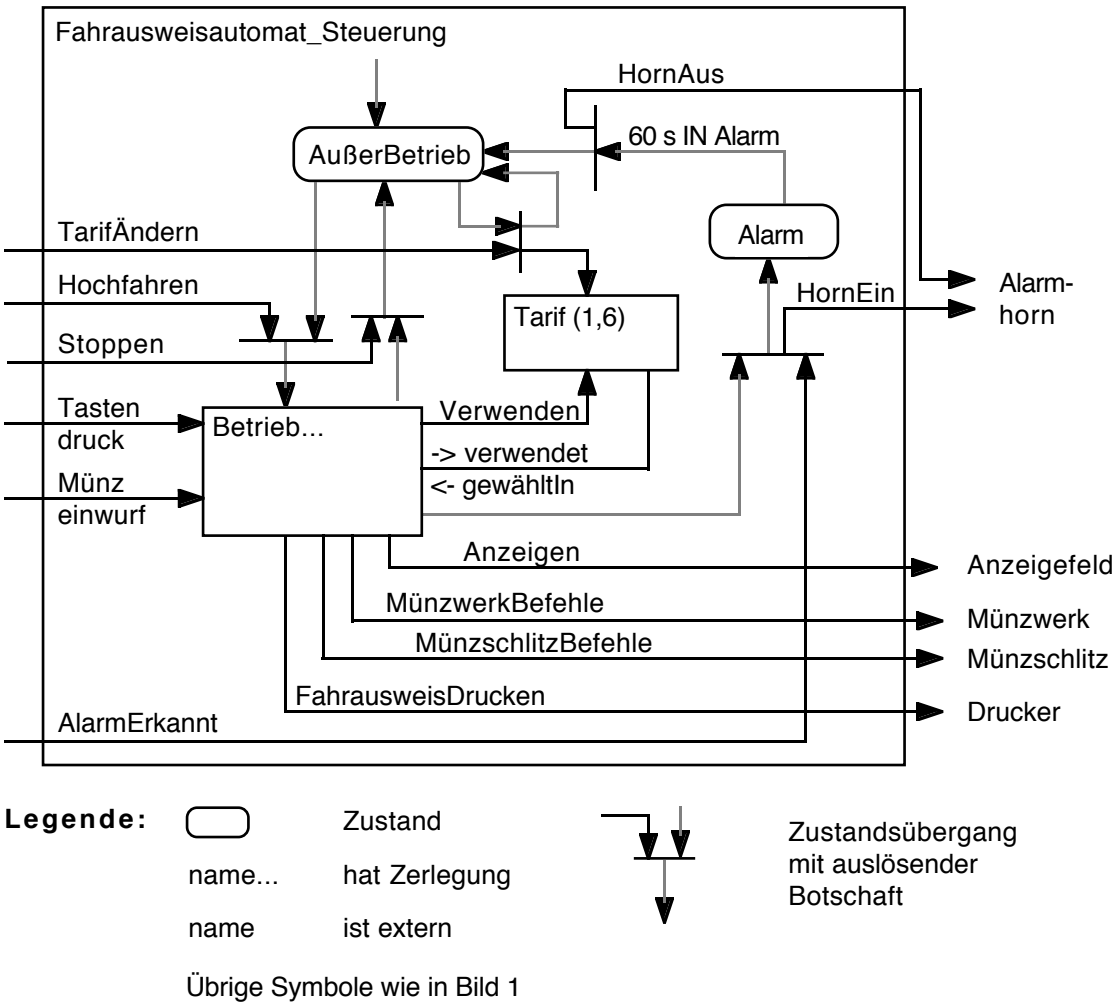

Bild 2: Objekt Fahrausweisautomat_Steuerung (Gesamtmodell der zu spezifizierenden Software)

Die Auslösung eines Zustandsübergangs geschieht durch Botschaften, die in der Graphik auf einen Querbalken im Zustandsübergangspfeil geführt werden. Der Zustandsübergang findet nur statt, wenn das System sich entweder in dem Zustand befindet, von dem der Zustandsübergangspfeil ausgeht, oder wenn es sich in einem Zustand innerhalb desjenigen Objekts befindet, von dem der Zustandsübergangspfeil ausgeht. Botschaften, für die diese Bedingung nicht erfüllt ist, gehen ohne Wirkung verloren. Ein Zustandsübergang kann neue Botschaften erzeugen; diese werden in der Graphik als vom Querbalken abgehende Pfeile dargestellt. Zustandsübergänge sind im Normalfall zeitfrei.

Der vom Diagrammrand kommende gestrichelte Pfeil kennzeichnet AußerBetrieb als Initialzustand. In diesem Zustand wird die Botschaft TarifÄndern akzeptiert und an das beauftragte Tarifobjekt weitergeleitet. Das System bleibt im Zustand AußerBetrieb. Die Botschaften Stoppen, Tastendruck, Münzeinwurf und AlarmErkannt werden in diesem Zustand ignoriert und gehen verloren. Beim Empfang der Botschaft Hochfahren wechselt das System in einen Zustand innerhalb des Objekts Betrieb. Von jetzt an werden die Botschaften TarifÄndern und Hochfahren ignoriert. Auf die Botschaften Tastendruck bzw. Münzeinwurf wird entsprechend der Spezifikation des Objekts Betrieb (siehe Bild 3 unten) reagiert. Eine Zustandsveränderung findet dabei höchstens innerhalb des Objekts Betrieb statt. Hingegen gibt es auf die Botschaften Stoppen bzw. AlarmErkannt eine in Bild 2 spezifizierte globale Reaktion. Bei AlarmErkannt beispielsweise wird das Objekt Betrieb (und damit auch jeder in diesem enthaltene Zustand) verlassen und in den Zustand Alarm übergegangen. Gleichzeitig wird die Botschaft HornEin erzeugt. Der Zustand Alarm wird durch eine Zeitbedingung (nachdem sich das System 60 s in diesem Zustand befunden hat) wieder verlassen. Dabei wird gleichzeitig das Ausschalten des Alarmhorns veranlaßt.

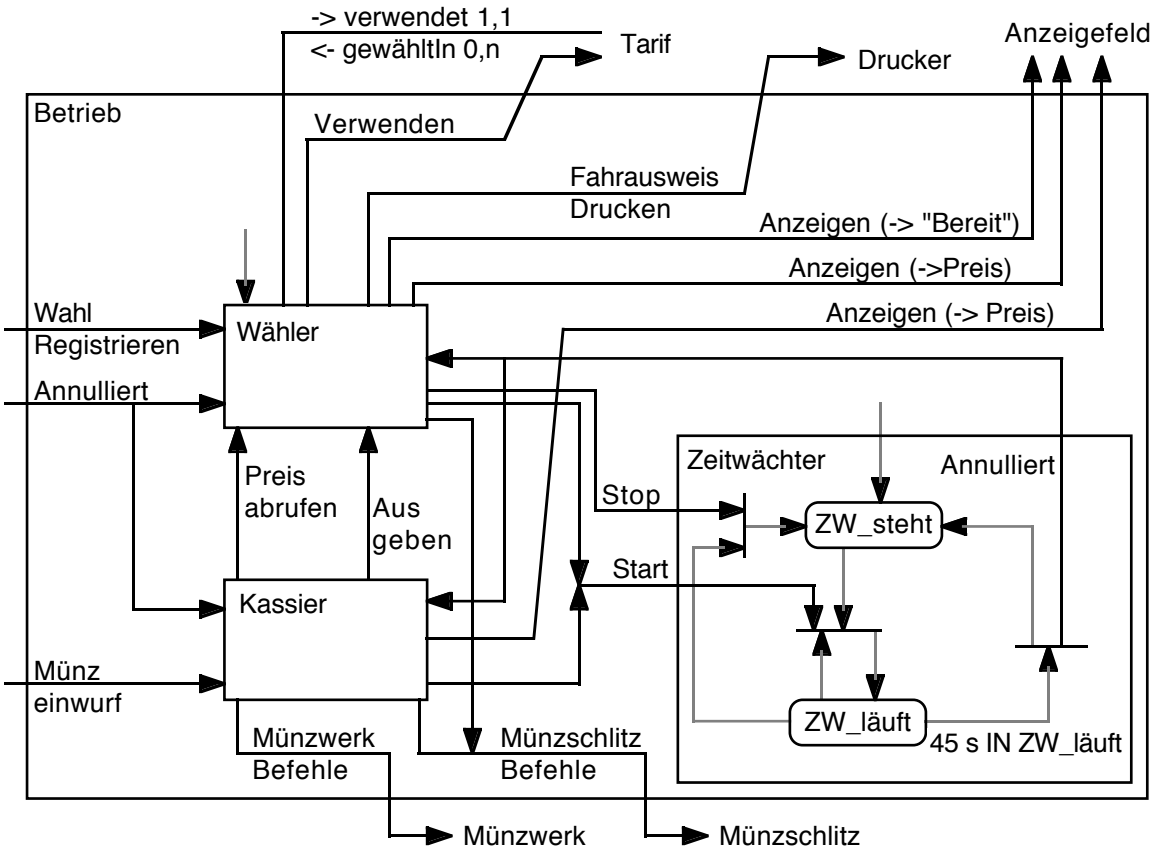

Bild 3: Objekt-Diagramm des Objekts Betrieb

Ein Objekt kann auf zwei Arten genauer spezifiziert werden: a) durch ein Objekt-Diagramm mit Objekten und Zuständen oder b) durch eine Elementarbeschreibung mit Operationen und Daten. Objekt-Diagramme können je nach Komplexität separat oder ineinander verschachtelt gezeichnet werden. Bild 3 zeigt die Spezifikation des Objekts Betrieb mit einem Objekt-Diagramm. Elementarbeschreibungen werden in Abschnitt 3.3 erläutert.

Das Objekt-Diagramm in Bild 3 beschreibt das Objekt Betrieb, welches aus den Objekten Wähler, Kassier und Zeitwächter besteht. Das Diagramm von Zeitwächter ist in dasjenige von Betrieb eingeschachtelt. In Betrieb gibt es zwei Initialzustände: einen Zustand innerhalb von Wähler und den Zustand ZW_steht in Zeitwächter. Dies ist zulässig, solange es im Inneren von Betrieb weder direkt noch transitiv Zustandsübergänge von den beiden Initialzuständen auf einen gemeinsamen Folgezustand gibt. Der Gesamtzustand des Objekts ist das kartesische Produkt der Einzelzustände. Wird das Objekt Betrieb durch die Botschaft Hochfahren (Bild 2) betreten, so gelangt es somit in den Initialzustand (Bereit, ZW_steht), wobei Bereit der Initialzustand innerhalb des Objekts Wähler ist (Bild 5).

Wird ein Objekt durch einen Zustandsübergang verlassen, so werden gleichzeitig und rekursiv alle inneren Zustände des Objekts verlassen. Befindet sich beispielsweise das Objekt Wähler im Zustand InAuswahl (Bild 5) und das Objekt Zeitwächter im Zustand ZW_läuft, so bewirkt die Botschaft AlarmErkannt nicht nur das Verlassen des Objekts Betrieb (Bild 2), sondern gleichzeitig auch das Verlassen der Objekte Wähler und Zeitwächter und damit der Zustände InAuswahl und ZW_läuft. AlarmErkannt bewirkt in dieser Situation also faktisch einen Zustandsübergang von (InAuswahl, ZW_läuft) nach Alarm.

3.3 Verhaltensbeschreibung in Elementarbeschreibungen von Objekten

Die Elementarbeschreibung spezifiziert die Eigenschaften eines Objekts unter den drei Aspekten *Daten* (Attribute, Beziehungen zu anderen Objekten), *Operationen* (welche Operationen gibt es auf den Daten des Objekts und was tun sie) und *Verhalten* (wie sieht der mögliche Lebenslauf eines Objekts aus). Auch auf Stufe Elementarbeschreibung kann ein Objekt noch andere Objekte enthalten. Elementarbeschreibungen erfolgen in der Regel textuell. Bild 4 ist eine Textbeschreibung der Klasse Tarif. Tarif hat kein zustandsabhängiges Verhalten und enthält daher nur Attribute, Beziehungen und Operationen. Vor allem, wenn komplexe Verhaltensabläufe zu spezifizieren sind, kann die textuelle Darstellung durch ein Diagramm ergänzt werden. Die Bilder 5 und 6 zeigen für das Objekt Wähler eine graphische Elementarbeschreibung sowie einen Ausschnitt aus der Textbeschreibung.

```
CLASS Tarif IS
   OBJECTS (1,6);                                  -- min./max. Anzahl von Exemplaren
   OID AUTOMATIC;                                  -- systemgenerierte Objekt-Idents

   TYPE
      Tastencode = 1..6;                           -- Typ-Definitionen für Attribute bzw.
      Rappen = 0..99999;                           -- Parameter

   ATTRIBUTES                                      -- Attributdefinitionen mit Kardinalitä-
      zugeordneteTaste (1,1) Tastencode;           -- ten (min./max. Anzahl Werten pro
      Name (1,1) STRING(20);                       -- Objekt) und Wertebereich
      Preis (1,1) Rappen;

   RELATIONSHIPS                                   -- Beziehungsdefinitionen mit Kardi-
      gewähltIn (0,n) Auswahl;                     -- nalität und Name der referenzierten
                                                   -- Klasse (bzw. des referenz. Objekts)

   FUNCTION TarifÄndern (->neuerCode: Tastencode;
      ->neuerName: Name; ->neuerPreis: Preis) IS
      "Das Objekt mit dem Tastencode <neuerCode>
      wird geändert oder neu erzeugt."             -- informale Definition einer Operation
   END Ändern;

   FUNCTION Abrufen (->Taste: Tastencode;
      -> neueAuswahl: Auswahl; <-gewählterTarif: Tarif) IS
      LET (x IN Tarif WITH x.zugeordneteTaste = Taste);
      PRE                                          -- formale Definition einer Operation
         EXIST x;
      POST
         gewählterTarif = x.OID;                   -- Setzen des Ausgabeparameters
         x.gewähltIn = x.gewähltIn' + {neueAuswahl}; -- Hinzufügen einer Beziehung[1)]
   END Abrufen;

   CLASS FUNCTION VerwendungLöschen IS             -- Botschaft an die Klasse
      PRE   NULL;
      POST
         FORALL x IN Tarif (x.gewähltIn = {});     -- alle Beziehungen sind gelöscht
   END VerwendungLöschen;
END Tarif;
```

Bild 4: Elementarbeschreibung der Klasse Tarif

In graphischen Elementarbeschreibungen sind Operationen in der Regel Bestandteil von Zustandsübergängen. Das Eintreffen einer Botschaft kann eine Operation nur aktivieren, wenn sie gleichzeitig den zugehörigen Zustandsübergang auslöst. Liegen auf einem Zustandsübergangspfeil mehrere Operationen, so werden diese hintereinander

1) name' ist der Wert von name zum Zeitpunkt des Aufrufs einer Operation

ausgeführt. Mehrere Zustandsübergänge, welche durch die gleiche Botschaft ausgelöst werden, können zusammengefaßt werden. Eine Operation kann auch als Zustandsverzweigung dienen. Der Folgezustand ist dann abhängig vom Ergebnis der Operation.

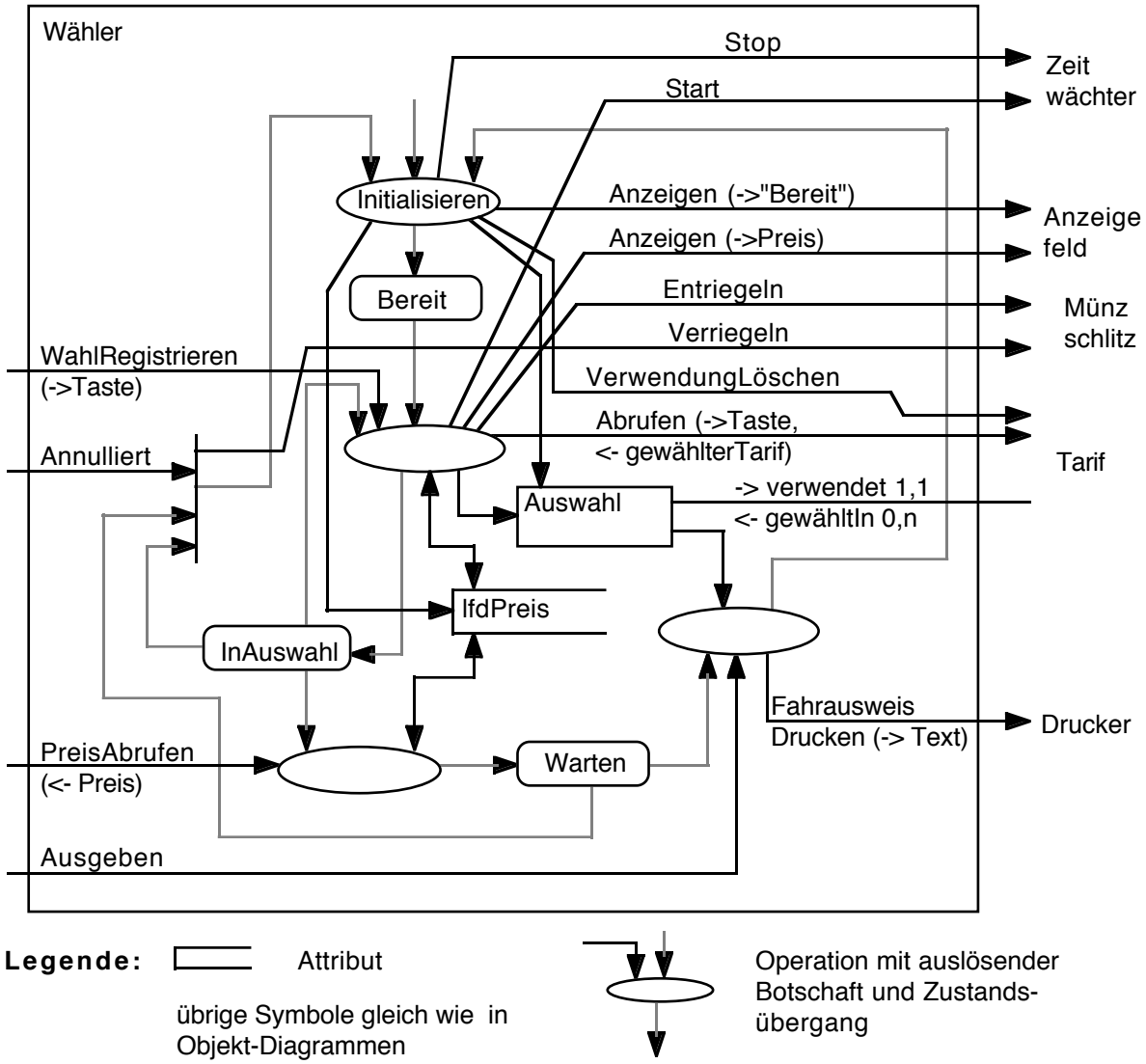

Bild 5: Graphische Elementarbeschreibung des Objekts Wähler

Befindet sich das System beispielsweise innerhalb von Wähler im Zustand Warten (Bild 5), so löst die Botschaft Ausgeben einen Zustandsübergang nach Bereit aus, bei dem zunächst die Operation Ausgeben und dann die Operation Initialisieren zur Ausführung gelangen. Durch die Auslösung der Botschaft Zeitwächter.Stop gibt es dabei auch im Zeitwächter (Bild 3) einen Zustandsübergang, so daß der neue Gesamtzustand des Systems aus dem Paar (Bereit, ZW_steht) besteht.

Die Wirkung einer Operation ist aus der Textform der Elementarbeschreibung ersichtlich. Das Diagramm zeigt nur die Datenzugriffe und die erzeugten Botschaften. Die Operation WahlRegistrieren (Bild 6) beispielsweise hat als Eingabeparameter das Objekt Taste, welches vom Typ Tastencode ist (dieser Typ wird aus der Klasse Tarif importiert). Die Operation ist *privat* im Objekt Betrieb, d.h. sie ist nur innerhalb dieses Objekts zugänglich. PRE beschreibt die Voraussetzung der Operation, nämlich daß das Objekt im Zustand Bereit oder im Zustand InAuswahl sein muß. Unter POST sind die Ergebnis-Zusicherungen aufgeführt: Ein Objekt neueAuswahl der Klasse Auswahl mit einer Beziehung verwendet ist erzeugt. Das Tarifobjekt, zu dem die Beziehung geht, ist durch Verwendung der Operation Abrufen (siehe USES-Angabe in Bild 6 und Definition von Abrufen in Bild 4) bestimmt worden. Der Wert zum Attribut lfdPreis ist erhöht, und zwar um den Wert des Attributs Preis des (über die Beziehung verwendet) zugeordneten Tarifobjekts. Die drei angegebenen Botschaften sind erzeugt. Der neue Zustand ist InAuswahl.

```
FUNCTION WahlRegistrieren (-> Taste: Tastencode) IS
   PRIVATE IN Betrieb;
   PRE
      IN STATE Bereit OR IN STATE InAuswahl;
   POST
      CREATED neueAuswahl WITH neueAuswahl IN Auswahl AND
            neueAuswahl.verwendet = zugehörigerTarif;
      lfdPreis = lfdPreis' + neueAuswahl.verwendet.Preis;
      Anzeigefeld.Anzeigen(->Preis) WITH Preis = lfdPreis;
      Zeitwächter.Start;
      Münzschlitz.Entriegeln;                    -- mehrfaches Entriegeln (bei wieder-
      IN STATE InAuswahl;                        --          holter Wahl) stört nicht
   USES Tarif.Abrufen (->Taste, neueAuswahl, <- zugehörigerTarif);
END WahlRegistrieren;
```

Bild 6: Ausschnitt aus der Textform der Elementarbeschreibung des Objekts Wähler (Operation WahlRegistrieren)

3.4 Der Zusammenhang in der Diagramm-Hierarchie

Werden die Stufen einer Hierarchie in mehreren Diagrammen beschrieben, so müssen die Aussagen dieser Diagramme untereinander konsistent sein. Ist x ein beliebiges Objekt oder eine beliebige Klasse, so müssen die Botschaften und Beziehungen von bzw. zum Diagrammrand im Objekt-Diagramm von x eindeutig mit den Botschaften und Beziehungen von bzw. nach x im übergeordneten Objekt-Diagramm übereinstimmen. Analoges gilt für die Konsistenz zwischen Objekt-Diagrammen und Elementar-Beschreibungen.

Um die abstrakten Modelle, welche durch die übergeordneten Diagramme dargestellt werden, nicht mit Detailinformation zu überladen, muß es möglich sein, auch Botschaften und Beziehungen zu abstrahieren. *Botschaften* können abstrahiert werden, indem zusammengehörige Botschaften zu einer abstrakten Botschaft zusammengefaßt werden, oder indem die Parameter einer Botschaft weggelassen werden. Beispielsweise sind die Botschaften WahlRegistrieren und Annulliert aus Bild 3 in Bild 2 zu einer Botschaft Tastendruck zusammengefaßt. Die Anzeigebotschaften mit verschiedenen aktuellen Parametern aus Bild 3 sind in Bild 2 zu einer Botschaft ohne Parameter zusammengefaßt. *Beziehungen* werden durch Weglassen der Kardinalitäten und durch Zusammenfassen verwandter Beziehungen zu einer abstrakten Beziehung (mit eigenem Namen) abstrahiert.

Werden verschiedene Botschaften oder Beziehungen zusammengefaßt, so braucht dies eine zugehörige Definition, damit die Konsistenz der Darstellung formal überprüft werden kann. Diese Definitionen gehören als Annotation zu den entsprechenden übergeordneten Diagrammen. Beispielsweise muß das Objekt-Diagramm in Bild 2 mit der Definition Tastendruck := WahlRegistrieren | Annulliert annotiert werden.

3.5 Abstraktionen in der Verhaltensbeschreibung

Am Beispiel des Objekts Betrieb in den Bildern oben wird deutlich, wie die **'Ist-Teil-von'-Abstraktion** auch eine Abstraktion der Verhaltensbeschreibung liefert. So gibt z.B. Bild 2 eine Übersicht über das Verhalten von Betrieb, sagt aber nicht, wie die genaue Reaktion auf Tastendruck und Münzeinwurf aussieht. Diese Verhaltensdetails werden erst in den Diagrammen und Textbeschreibungen der tieferen Stufen spezifiziert.

Die **Generalisierungs-Abstraktion** (welche in dem Fahrausweis-Automat-Beispiel nicht vorkommt) ermöglicht es, für die Objekte einer allgemeinen Klasse ein grundsätzliches Verhalten zu spezifizieren und für Objekte abgeleiteter Klassen dann Spezialfälle dieses Verhaltens zu beschreiben.

Die dritte für die Strukturierung objekt-orientierter Systeme wesentliche Abstraktion ist die **'Benutzt'-Abstraktion**, welche ein System in Schichten aufeinander aufbauender Dienstleistungserbringer und Dienstleistungsverwender gliedert. Der Abruf von Dienstleistungen erfolgt über das Senden von Botschaften vom Verwender zum Erbringer. Mit der hier vorgestellten Art der Verhaltensbeschreibung, bei der die Verhaltens-Teilmodelle sich durch Senden und Empfangen von Botschaften beeinflussen, können einzelne Verhaltens-Teilmodelle organisch in eine solche Schichtenstruktur integriert werden.

4 Zur formalen Definition des Verhaltensmodells

Das dynamische Systemverhalten wird durch verallgemeinerte Zustandsautomaten beschrieben. Das Konzept hat Ähnlichkeiten mit den Statecharts von Harel [4] und den davon abgeleiteten Objectcharts von Coleman, Hayes und Bear [2]. Auch Strukturierte Analyse mit Echtzeit-Erweiterungen (Hatley und Pirbhai [5], Ward und Mellor [9]) basiert auf der Grundidee einer hierarchischen Verhaltensbeschreibung mit Zustandsautomaten (allerdings ohne eine genau definierte Semantik). Im Gegensatz zu allen diesen Ansätzen ist im hier vorgestellten Modell die Beschreibung von Funktionalität, Verhalten und Struktur *integriert* und nicht auf verschiedene Diagramm-Typen verteilt.

Ein verallgemeinerter Zustandsautomat unterscheidet sich von einem endlichen Automaten dadurch, daß neben den explizit modellierten Zuständen auch gewöhnliche Daten zur Speicherung eines Teils des gesamten Systemzustands zugelassen sind. Die explizit modellierten Zustände dienen im wesentlichen dazu, das 'Makro'-Verhalten zu beschreiben, d.h. Zeiträume zu charakterisieren, in denen auf bestimmte Botschaften in einer bestimmten Weise reagiert oder auch nicht reagiert wird. Dies wird häufig auch als Objekt-Lebenslauf bezeichnet. Das 'Mikro'-Verhalten dagegen, d.h. unterschiedliche Resultate von Operationen aufgrund der Geschichte der Dateneingaben, wird primär mit Hilfe von Datenspeichern modelliert. Auf diese Weise hat das Gesamtmodell die Mächtigkeit einer Turing-Maschine (natürlich bei jeder Realisierung mit durch den Adreßraum begrenzter Bandlänge). Gleichzeitig wird die beim Spezifizieren mit endlichen Automaten sonst auftretende Zustandsexplosion vermieden.

4.1 Semantik hierarchisch verschachtelter Zustandsautomaten

Die Semantik der Zustände und Zustandsübergänge im hierarchischen Verhaltensmodell ist durch eine Abbildung auf eine Gesamt-Zustandsübergangsmatrix eindeutig definiert. Dazu wird zunächst die Menge der Elementarzustände bestimmt: Überall, wo es innerhalb eines Objekts parallele Zustandsfolgen gibt, wird die Menge aller geordneten Tupel der parallelen Zustände gebildet. Bei verschachtelten Objekten erfolgt die Tupelbildung rekursiv von innen nach außen. Die Menge der Elementarzustände besteht aus allen so gebildeten Zustandstupeln sowie allen Zuständen des Modells, die in keinem dieser Tupel vorkommen. Die Elementarzustände sind die *Zeilenindizes* der Gesamt-Zustandsübergangsmatrix. Die *Spaltenindizes* werden durch die Menge der Auslöser für Zustandsübergänge gebildet. Dies sind alle Botschaften, welche irgendwo im Modell einen Zustandsübergang auslösen, alle zeitlichen Bedingungen (wie z.B. 60 s IN Alarm; vgl. Bild

2) und alle Hilfsbotschaften bei Verzweigungen[1]. Tabelle 1 beschreibt die Abbildung vom Modell auf die Gesamt-Zustandsübergangsmatrix. Tabelle 2 zeigt als Beispiel die Matrix des Fahrausweis-Automaten (vgl. Bilder 2, 3 und 5).

Tabelle 1: Abbildung vom Modell auf Gesamt-Zustandsübergangsmatrix

Modell			**Gesamt-Zustandsübergangsmatrix**		
von	**Auslöser**	**nach**	**Zeilenindex i**	**Spalten-index j**	**Eintrag m_{ij}**
z_1	b	z_2	Alle Tupel der Art $(x_1,...,z_1,...,x_n)$	b	$(x_1,...,z_2,...,x_n)$
x_1	b	z_2	Alle Tupel der Art $(x_1,...,z_{s1},...,z_{sm},...,x_n)$ mit z_{si} Zustand in x_1	b	$(x_1,...,z_2,...,x_n)$
z_1	b	x_2	Alle Tupel der Art $(x_1,...,z_1,...,x_n)$	b	$(x_1,...,z_{j1},...,z_{jk},...x_n)$ mit $z_{j1},...z_{jk}$ = Tupel der Initialzustände von x_2
x_1	b	x_2	Alle Tupel der Art $(x_1,...,z_{s1},...,z_{sm},...,x_n)$ mit z_{si} Zustand in x_1	b	$(x_1,...,z_{j1},...,z_{jk},...x_n)$ mit $z_{j1},...z_{jk}$ = Tupel der Initialzustände von x_2
Legende: z_1, z_2: Zustände x_1, x_2: Objekte					

Tabelle 2: Gesamt-Zustandsübergangsmatrix des Fahrausweis-Automaten

Zeile: alter Zustand **Spalte:** Auslöser **Zelle:** neuer Zustand	Tarif-Ändern	Hochfahren	Stoppen	Wahl Registrieren	Annulliert	Preis abrufen	Ausgeben	Start	Stop	Alarm Erkannt	60 s IN Alarm	45 s IN ZW_läuft
1: AußerBetrieb	1	2	-	-	-	-	-	-	-	-	-	-
2: (Bereit, ZW_steht)	-	-	1	4	-	-	-	3	-	8	-	-
3: (Bereit, ZW_läuft)	-	-	1	5	-	-	-	3	2	8	-	2
4: (InAuswahl, ZW_steht)	-	-	1	4	2	6	-	5	-	8	-	-
5: (InAuswahl, ZW_läuft)	-	-	1	5	3	7	-	5	4	8	-	4
6: (Warten, ZW_steht)	-	-	1	-	2	-	2	7	-	8	-	-
7: (Warten, ZW_Läuft)	-	-	1	-	3	-	3	7	6	8	-	6
8: Alarm	-	-	-	-	-	-	-	-	-	-	1	-

Werden bei einem Zustandsübergang eine oder mehrere Botschaften erzeugt, welche ihrerseits Zustandsübergänge auslösen können, so wird zunächst der erste Zustandsübergang durchgeführt und vom Folgezustand aus dann untersucht, was die neu erzeugten

[1] Verzweigt ein Zustandsübergang abhängig vom Ergebnis einer Operation auf mehrere mögliche Folgezustände, so wird die Verzweigung vorgängig beseitigt: Der Zustandsübergang geht zunächst auf einen Hilfs-Zwischenzustand. Von der steuernden Operation erzeugte Hilfsbotschaften bewirken den Übergang vom Zwischenzustand in den gewünschten Folgezustand.

Botschaften bewirken. Sind dabei mehrere verschiedene Zustandsübergänge möglich, so gewinnt zufällig einer.

4.2 Synchronität und Zeit

Dem ganzen Modell liegt die Annahme einer zeitsynchronen Parallelität mit Zustandsübergängen der Dauer null zugrunde. Dadurch ist sichergestellt, daß das System sich zu jedem Zeitpunkt in genau einem der in der Gesamt-Zustandsübergangsmatrix aufgeführten Zustände befindet. Die Einführung von Dauern für Operationen (was zu Zustandsübergängen mit Dauern führt), ist von der Theorieseite her kein Problem: Ein Zustandsübergang von z_1 nach z_2 mit einer Operation f, welche n Zeiteinheiten dauert, wird für die Bestimmung der Semantik ersetzt durch einen neuen Zustand z_f mit Zustandsübergängen z_1 nach z_f und z_f nach z_2. Der erste Übergang wird ausgelöst durch die Bedingung, welche bisher den Übergang von z_1 nach z_2 ausgelöst hat; der zweite durch die Zeitbedingung "n Zeiteinheiten IN z_f".

Asynchrone Parallelität zwischen zwei Teilen des Modells ist dann möglich, wenn es zwischen den Modellteilen keine Zustandsübergänge und keine wechselseitige zeitbezogene Inspektion von Zuständen gibt.

4.3 Verhaltens-Spezialisierung

Wird von einer Klasse X eine Spezialisierung X' abgeleitet, so muß sich jedes Objekt aus X' auch wie der Spezialfall eines Objekts aus X verhalten. Dazu gehört, daß die Invarianten von X auch Invarianten von X' sind und daß für jede Operation b' in X', welche eine Operation gleichen Namens in X spezialisiert, gilt $\mathsf{PRE}\ b' \subseteq \mathsf{PRE}\ b$ und $\mathsf{POST}\ b' \supseteq \mathsf{POST}\ b$, d.h. eine Spezialisierung darf Voraussetzungen nicht verstärken und Ergebnis-Zusicherungen nicht abschwächen (vgl. Meyer [6]). Zusätzlich aber gibt es Bedingungen für das Verhalten von spezialisierten Objekten: Wird ein Objekt x' aus X' gleich behandelt wie ein gleichartiges Objekt x aus X, so muß es sich auch gleich wie x verhalten. Welches die genauen formalen Bedingungen für diese Verhaltenseinbettung spezialisierter Objekte sind, ist noch nicht untersucht.

5 Generierung von Prototypen

Der Schwerpunkt dieses Beitrags liegt in der Einführung und Beschreibung eines hierarchischen Verhaltensmodells als *Grundlage* für modellbasiertes Prototyping. Das Vorgehen bei der Generierung eines Prototyps wird daher hier nur grob skizziert.

Das Ausführungsmodell basiert auf einer Menge zeitsynchron ablaufender Prozesse. Jedes Objekt wird durch einen Prozeß betrieben[1]. Zur Erzeugung dieser Prozesse hat der Prototyp-Generator folgende Basis-Arbeit zu leisten:

- Aus den Attributen und Beziehungen jedes Objekts generiert er entsprechende Datenstrukturen (am besten unter Verwendung einer objekt-orientierten Datenbank).
- Für jede Operation generiert er eine Methode. Für den dazu notwendigen Code gibt es drei mögliche Quellen: (1) die Spezifizierer annotieren die Definition der Operation im Modell mit einem entsprechenden Codestück, (2) sie geben auf eine entsprechende Anfrage des Prototyp-Generators hin ein Codestück oder feste Resultatwerte an, (3) der Prototyp-Generator generiert eine Laufzeit-Anfrage für die zu verwendenden Operations-Resultate.
- Er generiert die Zustandsmaschine für das Objekt (siehe unten).
- Er generiert einen einfachen Objektrahmen (Bild 7), welcher mit Hilfe der Objekt-Zustandsmaschine das Objektverhalten steuert.

```
LOOP
   "Warten auf Botschaft";
   Zustandsmaschine_von_x.Fortschalten (->empfangeneBotschaft);
END LOOP;
```

Bild 7: Coderahmen für ein Objekt x

Jeder Prozeß, der ein zustandsbehaftetes Objekt x betreibt, enthält eine Objekt-Zustandsmaschine für x. Dies ist ein Objekt einer Klasse Zustandsmaschine, welches den Automaten betreibt, der im Verhaltensmodell von x definiert ist. Auf diesem Automaten sind drei Operationen definiert: Init, Exit und Fortschalten(->auslösendeBotschaft) (siehe Bild 8). Init und Exit werden innerhalb von Fortschalten verwendet, um Zustandsübergänge in eingeschachtelte Objekte hinein oder aus solchen heraus zu veranlassen. Beispiel: Beim

[1] Die Anzahl der benötigten Prozesse läßt sich reduzieren, wenn alle Objekt-Prozesse, die nie parallel ausgeführt werden können, jeweils zu einem Prozeß zusammengefaßt werden.

Zustandsübergang von AußerBetrieb nach Betrieb (vgl. Bild 2) wird Zustandsmaschine_von_Betrieb.Init aufgerufen, welche rekursiv Zustandsmaschine_von_Wähler.Init und Zustandsmaschine_von_Zeitwächter.Init aufruft.

```
Operation Init IS
   PRE
      "Der aktuelle Zustand ist nicht innerhalb des Objekts";
   POST
      "Der Initialzustand des Objekts ist gesetzt. Enthält ein Objekt eingeschachtelte Objekte, so
      gilt diese Zusicherung rekursiv auch für diese Objekte.";
END Init;

Operation Exit IS
   PRE   NULL;
   POST
      "Alle Zustände des Objekts (einschließlich aller eingeschachtelten Objekte) sind verlassen.";
END Exit;

Operation Fortschalten (->auslösendeBotschaft: Botschaft) IS
   PRE   NULL;
   POST
      IF "auslösende Botschaft wird in aktuellem Zustand akzeptiert"
      TRUE:   "Botschaft ist an Methode gebunden und ausgeführt, Folgeoperationen auf dem
              gleichen Zustandsübergang sind (falls vorhanden) reihenfolgerichtig ausgeführt,
              Folgezustand ist eingenommen (ggf. unter Verwendung von Init bzw. Exit).";
      FALSE:  "Falls vorhanden, ist Ergebnisparameter für Mißerfolg in Botschaft gesetzt,
              Zustand bleibt unverändert.";
      FI;
END Fortschalten;
```

Bild 8: Operationen auf der Objekt-Zustandsmaschine

Schlußendlich wird der erzeugte Prototyp in eine Umgebung eingebunden, welche die benötigten externen Botschaften erzeugt und die Botschaften an externe Objekte entgegennimmt. Diese Umgebung kann sein: (1) die Hardware und Betriebssoftware des Zielsystems, (2) vorhandene Software (vor allem bei Erweiterungen und Änderungen bestehender Software) oder (3) ein Simulator für das Zielsystem.

6 Schlußbemerkung

Meine Arbeit an objekt-orientierten Systemmodellen mit hierarchischer Verhaltensbeschreibung steht noch am Anfang. Die Sprache ist grob definiert; die Möglichkeiten

zu einer vollständigen, sauberen Definition sind erkennbar. Eine Methodik für den Einsatz der Sprache existiert in Ansätzen; wird aber in diesem Beitrag nicht behandelt. Werkzeuge fehlen noch vollständig, und zwar sowohl für die Modell-Erstellung und -Bearbeitung, als auch für die Generierung von Prototypen.

Das, was heute vorhanden ist, zeigt aber deutlich, daß mit einem solchen Modell die wesentlichen Schwachstellen bisheriger Ansätze zur objekt-orientierten-Spezifikation überwunden werden können. Durch die Zusammenhänge in der Verhaltensbeschreibung wird objekt-orientiertes Spezifizieren auch für Echtzeitsysteme anwendbar. Die 'Ist-Teil-von'-Hierarchie mit den zugehörigen Abstraktionsmechanismen erlaubt eine natürliche Modellierung komplexer, mehrstufig ineinander eingebetteter Systeme. Die Generierung von Prototypen aus dem Modell ermöglicht ein organisches Zusammenwirken von Modellen und Prototypen in der Spezifikation von Systemen.

Literatur

1. Coad, P., E. Yourdon (1991). *Object-Oriented-Analysis*. Prentice-Hall, Englewood Cliffs, N.J.
2. Coleman, D., F. Hayes, S. Bear (1992). *Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design*. IEEE Transactions on Software Engineering, **18**, 1 (Jan 1992), 9-18.
3. Glinz, M. (1991). *Probleme und Schwachstellen der Strukturierten Analyse*. GI-Fachtagung RE'91, Marburg, Informatik-Fachberichte Nr. 273, Springer Verlag.
4. Harel, D. (1987). *Statecharts: A Visual Formalism for Complex Systems*. Sci. Computer Program. **8** (1987), 231-274.
5. Hatley, D.J., I.A. Pirbhai (1988). *Strategies for Real-Time System Specification*. Dorset House, New York.
6. Meyer, B. (1992). *Applying "Design by Contract"*. IEEE Computer 25, 10 (Oct 1992), 40-51.
7. Shlaer, S., S.J. Mellor (1988). *Object-Oriented Systems Analysis: Modeling the World in Data*. Prentice-Hall, Englewood Cliffs, N.J.
8. Shlaer, S., S.J. Mellor (1992). *Object Lifecycles: Modeling the World in States*. Prentice-Hall, Englewood Cliffs, N.J.
9. Ward, P.T., S.J. Mellor (1985). *Structured Development for Real-Time Systems*, Vol. I-III. Prentice-Hall, Englewood Cliffs, N.J.