

2 Entwurf, Simulation und Programmierung von ereignisgesteuerten Systemen

2.1 Einführung

Systeme lassen sich unter verschiedenen Gesichtspunkten klassifizieren. Eine Variante ist z.B. die Einteilung in lineare und nichtlineare Systeme. Legt man besonderen Wert auf den Echtzeitcharakter, so ist eine Aufteilung in eine **transformatorische** und eine **reaktive** (ereignisgesteuerte) Sicht von Bedeutung.

Ein transformatorisches System führt ein Eingangssignal in ein Ausgangssignal über. Die transformatorische Methode ist gekennzeichnet durch Datenflüsse, Funktionen und Speicherelemente. Daten fließen in Funktionen ein, werden dort über Algorithmen in Ausgangsdaten transformiert oder in Speicherelementen gespeichert. Mit CASE-Tools, die primär mit Datenflüssen arbeiten, lassen sich Zeitabhängigkeiten und Reihenfolgen nur schwer beschreiben.

Echtzeitsysteme sind durch Ereignisse bestimmt, die zu jedem beliebigen Zeitpunkt auftreten können. Dabei ist für die Reaktion des Systems wichtig, in welchem Zustand das System sich zum Zeitpunkt des Ereignisses befindet.

Ein üblicher Ansatz zur Beschreibung von Reihenfolgenabhängigkeiten ist der mittels endlicher Zustandsautomaten und ihrer Zustandübergangsdigrammen (state transition diagram) kurz Zustandsdiagramm (state diagram) genannt.

Endliche Zustandsautomaten werden jedoch zur Spezifikation von komplexen Systemen nur ungerne verwendet. Gründe hierfür sind:

- Zustandsdiagramme sind "flach"; sie enthalten keine Syntax für Hierarchie und Modularität
- Zustandsdiagramme besitzen keinen Prioritätsmechanismus und kein Systemgedächtnis
- In realen Systemen wachsen Zustandsdiagramme exponentiell mit linear wachsender Systembeschreibung
- Zustandsautomaten sind rein sequentiell angelegt. Sie besitzen keine Beschreibungsmöglichkeit für gleichzeitige (parallele) Vorgänge.

Prof. D. Harel und seine Mitarbeiter begannen 1983 mit der Erweiterung der Zustandsdiagramme zu einem leistungsfähigen Werkzeug für den Entwurf zustandsgesteuerter, reaktiver Systeme: den Statecharts. In der Zwischenzeit wird weltweit an Erweiterungen der Statecharts gearbeitet. Zudem gibt es kommerzielle CASE-Tool-Produkte, die mit dieser Technik arbeiten, so z.B. das CASE-Tool STATEMATE von der Firma i-Logix.

2.2 Semantik von Statecharts

2.2.1 Informationselemente und Timing

Zustände werden in der Vorlesung durch Vierecke mit abgerundeten Ecken dargestellt und werden durch einen Bezeichner (in Großbuchstaben) gekennzeichnet. Zustandsübergänge sind mit Pfeil versehen und werden in Kleinbuchstaben bezeichnet. Der aktuelle Zustand wird durch eine braune Schattierung dargestellt (Bild 2-1).



Bild 2-1: Allgemeiner Zustandsübergang

TRANSITION

Ein Übergang (Transition) wird gekennzeichnet durch die Ursache für den Übergang und eventuell durch eine weitergehende Aktion, die aufgrund des Übergangs stattfindet. Ursache und Aktion werden durch einen Schrägstrich getrennt (Bild 2-1).

Ursache für einen Übergang kann (üblicherweise) ein **Event e**, eine **Condition c** oder die Verbindung von e und c sein.

e[c]/a bedeutet: Wenn das Event e zum Zeitpunkt t auftritt und die Condition c zu diesem Zeitpunkt true ist, dann findet ein Zustandswechsel von S1 nach S2 statt. Gleichzeitig wird der Aktionsausdruck a ausgeführt. a kann u.a. wiederum ein Event sein (Bild 2-12). Dies gilt natürlich nur dann, wenn zum Zeitpunkt t die Steuerung sich im Zustand S1 befindet.

EVENT

Das Informationselement Event (Ereignis) ist nur zu einem einzelnen Zeitpunkt wirksam und enthält keinen bestimmten Wert. Die einzige Bedeutung eines Events ist die Tatsache, daß ein Event zu einem speziellen Zeitpunkt t auftritt oder nicht. Man könnte Events als informations-technische "Dirac-Stöße" bezeichnen. Events wirken nicht speichernd.

Wenn (Bild 2-2) die Steuerung sich im Zustand S1 befindet und das Event e23 zum Zeitpunkt t1 auftritt, dann bewirkt dieses Event zu diesem Zeitpunkt keine Zustandsänderung.

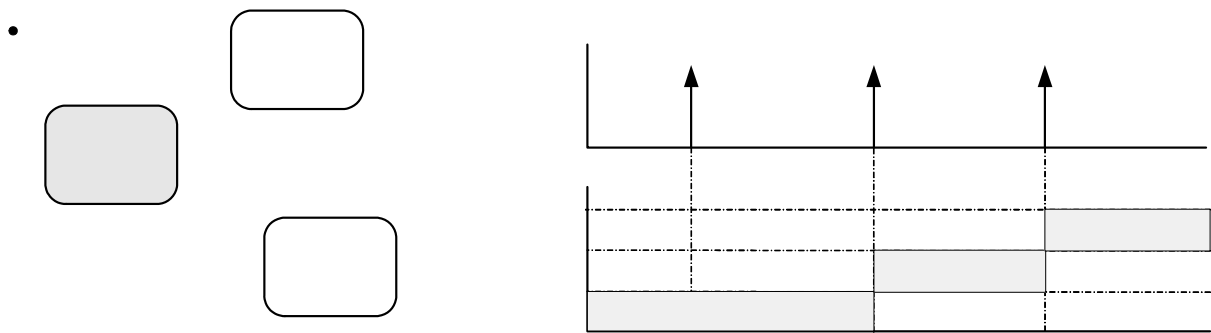


Bild 2-2: Ereignis (Event) als "Dirac-Stoß"

Wenn zu einem späteren Zeitpunkt durch e2 die Steuerung sich im Zustand S2 befindet, ist das Event e3 (vom Zeitpunkt t1) "vergessen". Ein erneutes Auftreten von e3 zum Zeitpunkt t3 bewirkt jetzt den Übergang von Zustand S2 nach Zustand S3.

CONDITION

Das Informationselement Condition (Bedingungen) kann die Werte true und false annehmen. Eine Condition ist level-sensitiv, d.h. sie wirkt speichernd (Bild 2-3).

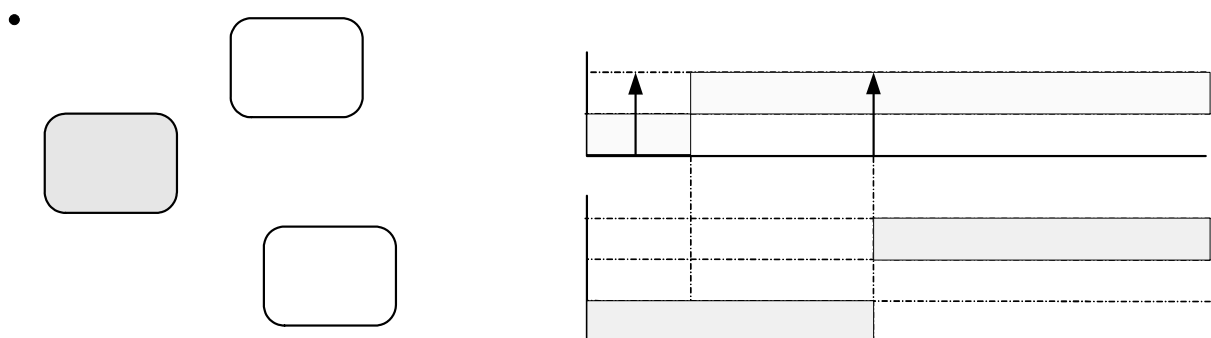


Bild 2-3: Bedingung als Anstoß zum Zustandswechsel

Eine Condition kann aber auch edge-sensitiv sein. Dabei wird die Condition als Parameter in einer Funktion benutzt, die beim Wechsel der Condition von false nach true ein Event erzeugt. Dieses Event wird notiert als true(c) oder kurz tr(c) (Bild 2-4).

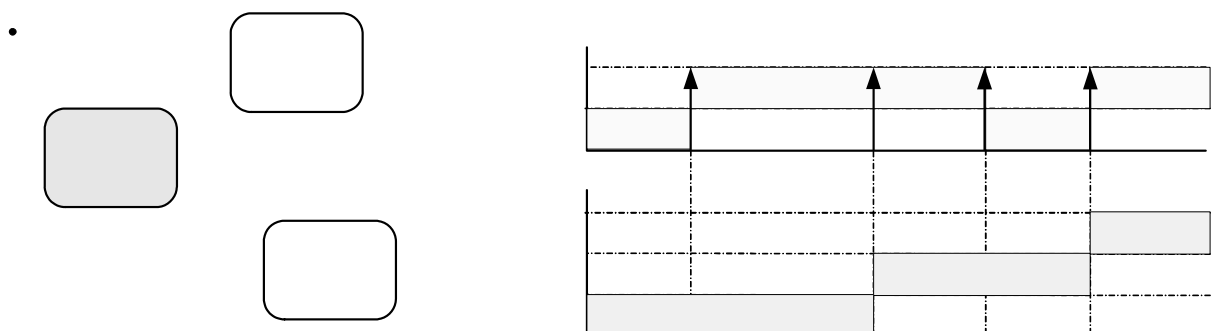


Bild 2-4: Bedingung als Funktionsergebnis

COMPOUND EVENT/CONDITION

Mehrere Events oder Events mit Conditions können verknüpft werden (Compound Event). Die folgende Tabelle zeigt die Syntax:

Syntax	Zustandsübergang tritt ein, wenn:
e[c]	E auftritt und gleichzeitig die Condition c wahr ist
[c]	Condition c wahr ist
not e	E nicht auftritt
e1 and e2	e1 und e2 gleichzeitig auftreten
e1 or e2	e1 oder e2 oder beide gleichzeitig auftreten

Mehrere Conditions können zu einer Condition verknüpft werden (Compound Condition). Die folgende Tabelle zeigt die Syntax:

Syntax	Condition ist true, wenn:
not c	c ist nicht true
c1 and c2	c1 und c2 sind beide true
c1 or c2	c1 oder c2 ist true

DATA ITEM

Das Informationselement Data Item ist vergleichbar mit einer Variablen in einer höheren Programmiersprache. Ein Data Item kann Werte enthalten und besitzt eine Struktur und einen Datentyp (Bild 2-5). Als Strukturen können gewählt werden: Single, Queue und Array. Als Datentypen existieren: Real, Bit, String, Integer, Bit-Array, Record, Union, User-Defined.

Für viele Aufgaben reicht die Struktur Single in Verbindung mit dem Typ Integer. Über den Operator := kann einem Data Item ein Wert zugewiesen werden.

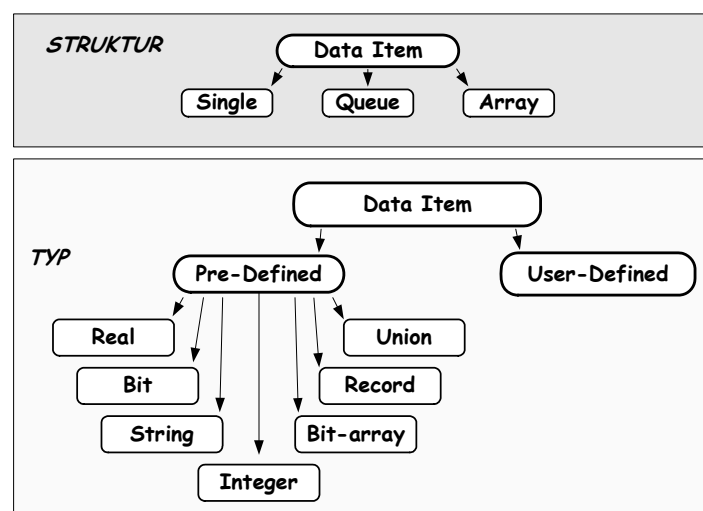


Bild 2-5: Typen und Strukturen zu Data Item

TIMING

Beim Modellieren von Systemen werden häufig Timer, Zeitintervalle zwischen Ereignissen, Wartezeiten zur Time-Out-Überwachung, etc. benötigt. Dafür werden zwei Funktionen angeboten: $tm(e,n)$ und $/sc!(e,n)$.

$tm(e,n)$ ist ein Event, der n Zeiteinheiten nach dem Event e auftritt. $/sc!(e,n)$ ist eine Action, die ein Event e in n Zeiteinheiten auslöst.

Um Wartezeiten zu modellieren, kann die Funktion $tm(e,n)$ zusammen mit der Funktion $entered(S)$ kurz $en(S)$ genutzt werden. $en(S)$ ist ein Event, der beim Eintritt in den State S auftritt. Wird im Ausdruck $tm(e,n)$ das Ereignis e durch das Ereignis $en(S)$ ersetzt, erhält man ein Event $tm(en(S),n)$. Dies bedeutet: n Zeiteinheiten nach dem Eintritt in den Zustand S wird ein Event $tm(...)$ ausgelöst. Dieser Zusammenhang wurde in Bild 2-6 benutzt, um eine Wartezeit zu realisieren. n Zeiteinheiten, nachdem ein Zustand $S1$ erreicht wurde, geht das System automatisch in den Zustand $S2$ über.



Bild 2-6: Wartezeit

Als Zeiteinheit für n kann z.B. die Einheit Sekunde oder Millisekunde gewählt werden.

ACTION

Eine Action (Aktion) ist u.a. ein Event oder die Wertzuweisung an ein Data Item. Actions können auch aufgezählt werden. In diesem Fall werden sie durch ein Semikolon getrennt.

DATA DICTIONARY

Alle Elemente, die in der grafischen Darstellung vorhanden sind, werden auch im Data Dictionary geführt. Hier allerdings in textlicher Form, in Tabellen, etc. Das Data Dictionary ist das Notizbuch zu den einzelnen Statecharts. Im Data Dictionary werden u.a. die Größen wie Event, Condition und Data Item vereinbart. Weiterhin werden in die Data Dictionaries die Static Reactions notiert.

STATIC REACTION

Innerhalb eines Zustands können spezielle Actions, sogenannte Static Reactions definiert werden. Die Actions können u.a. an das Betreten oder das Verlassen des Zustands geknüpft werden. Oder sie hängen von anderen Events im System ab.

Enthält ein Statechart eine Static Reaction (oder mehrere), dann erhält der Statechart-Name automatisch das Präfix <.

Der Eintrag von Static Reactions erfolgt in das Data Dictionary, welches dem Zustand zugeordnet ist. Dafür stehen zwei Spalten zur Verfügung: on event und do action. Unter on event werden die Events notiert, aufgrund derer dann die Actions unter do action ausgeführt werden.

Es können mehrere Static Reactions untereinander gereiht werden. In diesem Fall werden sie durch doppelte Semikolons getrennt:

on event	do action
event1	/action1;;
event2	/action2;;
.....;;;;
eventn	/actionn

Die speziellen Events, die für die Reactionkopplung an das Betreten bzw. Verlassen des Zustands benutzt werden, lauten: entering und exiting, abgekürzt ns und xs. Die Nutzung von ns und xs erzeugen genauso einen Mealy-Automaten, wie bei der Action-Notation an die Transitionspeile (Bild 2-1).

Wird in der on event-Spalte ein Event notiert, der an einer anderen Stelle erzeugt wird, erhalten wir einen Moore-Automaten.

2.2.2 Hierarchie und Struktur

Zustandsdiagramme besitzen eine flache Struktur. Ein einfaches Beispiel zeigt das Bild 2-7. Das Zustandsdiagramm besitzt 2 Zustände AUS und AN. Durch die beiden Ereignisse on und off kann von einem zum anderen Zustand gewechselt werden.

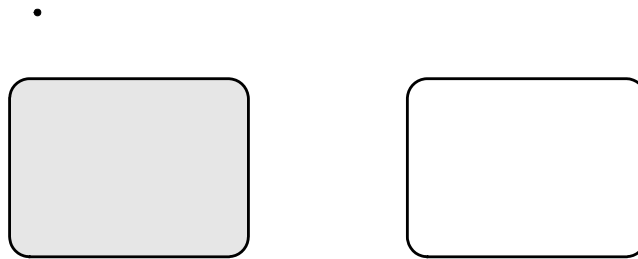


Bild 2-7: Einfaches, flaches Statechart

Bei Statecharts können Zustände ineinander geschachtelt werden. Bei unserem Beispiel werden innerhalb des Zustands AN zwei weitere Zustände angelegt, die sich auf Art des Waschguts beziehen: BUNT und WOLLE (Bild 2-8).

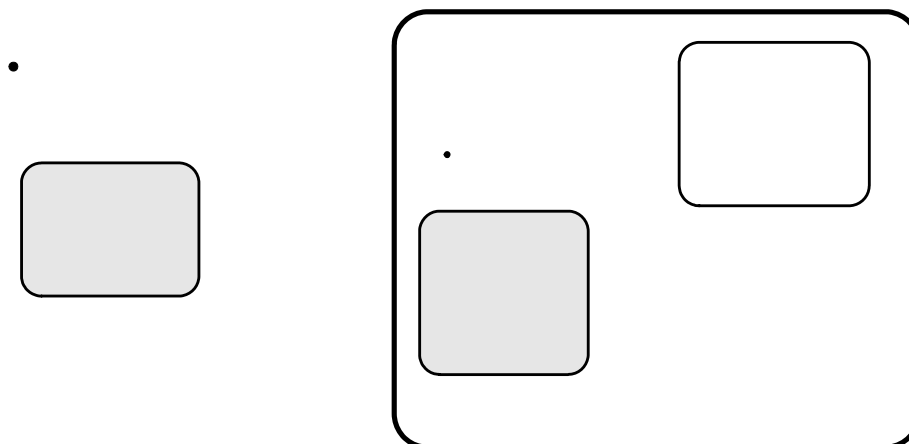


Bild 2-8: Hierarchischer Automat

Den (Super-)Zustand AN nennt man einen OR-Zustand, da entweder der Zustand BUNT oder der Zustand WOLLE eingenommen wird. Zudem wird mit dem gepunkteten Pfeil der Default-Zustand (hier: BUNT) angezeigt, der eingenommen wird, wenn das System in den Zustand AN gelangt. Befindet sich das System im Zustand AN, so befindet es sich gleichzeitig im Zustand BUNT oder WOLLE.

Der Übergang `off` geht vom Rand des Zustand AN ab. Dies bedeutet, daß unabhängig davon in welchem Zustand sich AN befindet (BUNT oder WOLLE), ein Übergang von AN nach AUS stattfindet. Ohne hierarchische Darstellung ergibt sich eine Topologie wie in Bild 2-9 gezeigt.

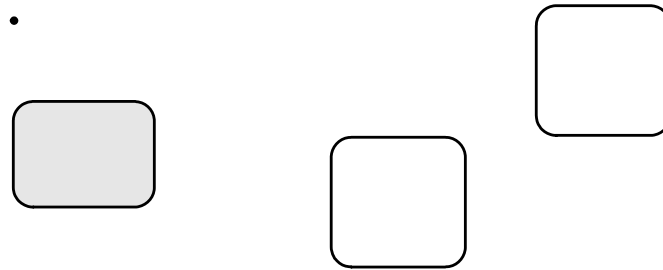


Bild 2-9: Flache Automatenstruktur

Aus Bild 2-9 wiederum kann man durch Einführung eines Superzustands zum Bild 2-8 kommen.

Allgemein können damit zwei Vorgehensweisen eines Entwurfs gezeigt werden (Bild 2-10).

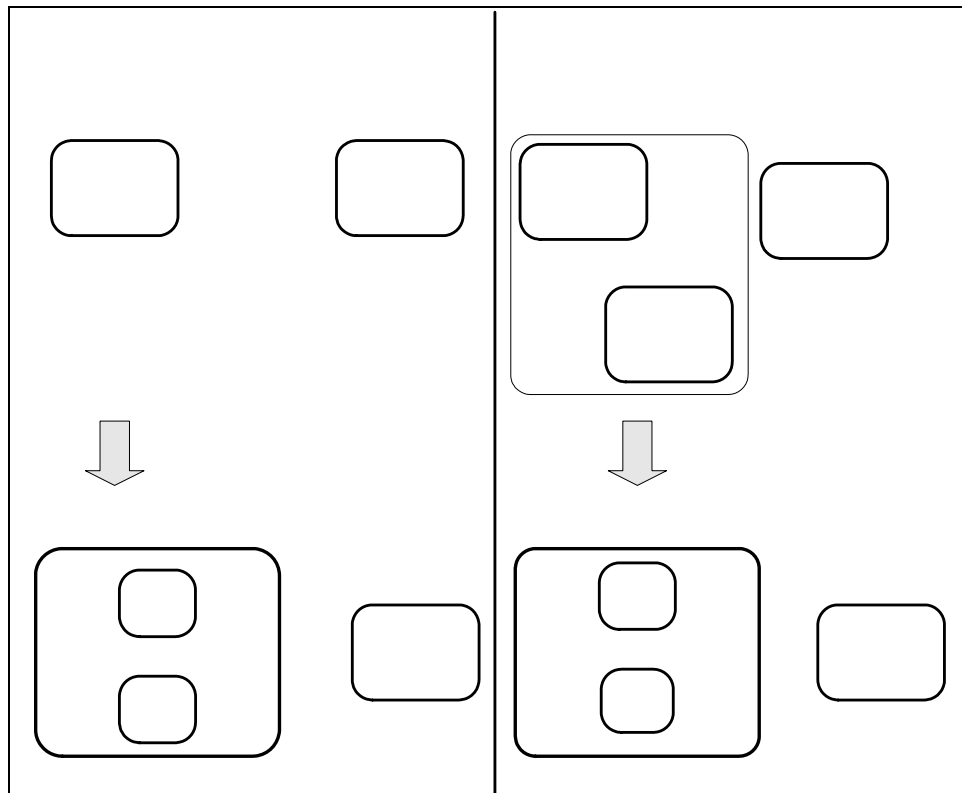


Bild 2-10: Top Down/Bottom Up-Konzept

Übergangslinien können über Hierarchiegrenzen hinweggehen, wie in Bild 2-11 mit dem Übergang off gezeichnet. (Ergibt sich hier ein Problem?)

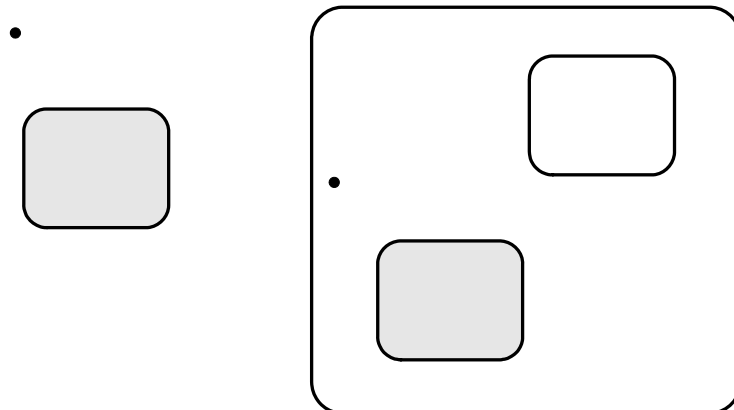


Bild 2-11: Hierarchische Struktur

Box-is-Chart

Je komplexer ein System wird, desto mehr treten hierarchisch aufgebaute Statecharts auf. Um die Übersicht zu behalten, können verschachtelte Statecharts zusammengefaßt in einer Box dargestellt werden. Dadurch wird eine Form erzielt, die ohne überflüssige Details den Überblick über das Gesamtsystem gewährleistet.

Diese Darstellungsart wird Box-is-Chart-Mechanismus genannt. Um zu zeigen, daß in einem einfachen State weitere Subzustände verborgen sind, wird der Name des States in der Box-is-Chart-Darstellung mit einem At-Zeichen (@) begonnen.

Bei einer Bottom-Up-Vorgehensweise kann auf diese Art und Weise ein Teil des Systems modelliert werden. Dieser Teil kann dann in das Gesamtsystem als Box-is-Chart eingefügt werden. Der Name des Teilsystems wird dann durch ein führendes At-Zeichen erweitert.

Über Bedienfunktionen von STATEMATE kann zwischen der Box-is-Chart-Darstellung und der detaillierten Form gewählt werden.

Der Box-is-Chart-Mechanismus bringt auch deshalb Übersichtlichkeit, da eine Bildschirmseite nur eine begrenzte Informationsfülle aufnehmen kann. D.h. betrachtet man das Gesamtsystem auf einer Bildschirmseite, werden Teilsysteme durch den Box-is-Chart-Mechanismus dargestellt. Interessieren Details in einer bestimmten Box, kann diese geöffnet werden. Dadurch erhält man nun das "Innenleben" dieser Box bildschirmfüllend dargestellt, etc.

2.2.3 Parallelität

Eine weitere Erweiterung der Zustandsdiagramme ist die Möglichkeit, parallele Automaten zu definieren. Bei dem Beispiel Waschmaschine können die Zustände SCHLEUDERN und NICHT_SCHLEUDERN in den Zustand AN integriert werden, allerdings unabhängig von den Zuständen BUNT und WOLLE. Dies stellt man in der Statechart-Technik durch zwei parallele (orthogonale) Automaten dar, die durch eine gestrichelte Linie (and-line) getrennt sind. Jeder Automat wird gekennzeichnet (WASCHGUT bzw. NASSZUSTAND), die Bezeichnung AN wird außen angetragen (Bild 2-12).

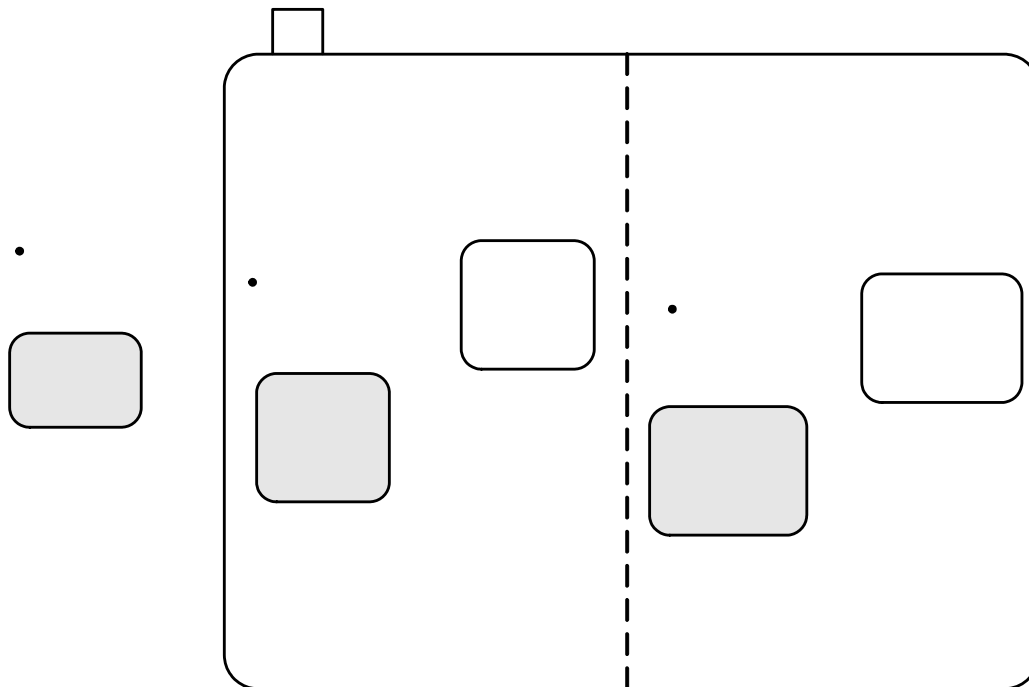


Bild 2-12: Paralleler Automat

Befindet sich die Steuerung im Zustand AN, so werden zwei Subzustände gleichzeitig eingenommen: (BUNT +SCHLEUDERN) | (BUNT +NICHT_SCHLEUDERN) | (WOLLE +SCHLEUDERN) |(WOLLE + NICHT_SCHLEUDERN).

Den Zustand AN nennt man einen AND-Zustand. Einen Zustand kann man in beliebig viele parallele Zustände unterteilen. Wenn ein AND-Zustand betreten wird, werden gleichzeitig sämtliche Subzustände betreten. Dies gilt, wenn der AND-Zustand durch einen Übergang an seine Außengrenze erreicht wird und auch dann, wenn direkt von "Außen" ein Subzustand erreicht wird. Beim Verlassen gilt Ähnliches. Wenn ein AND-Zustand verlassen wird, werden gleichzeitig sämtliche Subzustände verlassen. Dies gilt, wenn der AND-Zustand durch einen Übergang von der Außengrenze aus verlassen wird und auch dann, wenn der AND-Zustand von einem Subzustand aus direkt verlassen wird.

Der Entwurf mit parallelen Automaten reduziert die Anzahl der zu definierenden Zustände ganz enorm. Damit kann ein gewichtiger Nachteil von herkömmlichen Zustandsdiagrammen überwunden werden.

An dieser Stelle sollen die möglichen Typen von Zuständen nochmals betrachtet werden. Es gibt drei Typen von Zuständen in einem Statechart: OR-Zustände, AND-Zustände und Basic-Zustände. OR-Zustände besitzen Subzustände, die Exklusiv-Oder verknüpft sind. AND-Zustände besitzen orthogonale Subzustände. Basic-Zustände befinden sich "ganz unten" in der Zustandshierarchie und besitzen keine Subzustände.

Der Zustand "ganz oben", der kein Unterzustand ist, wird Root (Wurzelzustand) genannt. Eine Full-Configuration ist eine maximale Menge von Zuständen, in welchen das System sich zur gleichen Zeit befinden kann (bedingt durch OR- und AND-States).

Mit Basic Configuration wird eine maximale Menge von Basic-Zuständen bezeichnet, in denen sich das System zur gleichen Zeit befinden kann.

Zu einer bestimmten Zeit befindet sich das System in mindestens einem Basic-Zustand. Mit maximaler Menge sind zusätzlich alle weiteren Zustände gemeint, in denen sich das System gleichzeitig befinden kann (bedingt durch AND-Zustände bzw. durch übergeordnete Zustände).

2.2.4 Kommunikation (Broadcasting)

In realen Systemen sind parallele Automaten nicht immer unabhängig voneinander, sondern beeinflussen sich gegenseitig. In Statecharts können Ereignisse in einem Automaten erzeugt werden, die auf andere parallele Automaten wirken und dort einen Zustandswechsel hervorrufen. Diesen Vorgang nennt man auch Broadcasting. In Bild 2-13 ist dies durch w/ns ausgedrückt: Wenn das Event w eintritt, wird das Event ns erzeugt. Dadurch geht der Automat NASSZUSTAND in den Zustand NICHT_SCHLEUDERN.

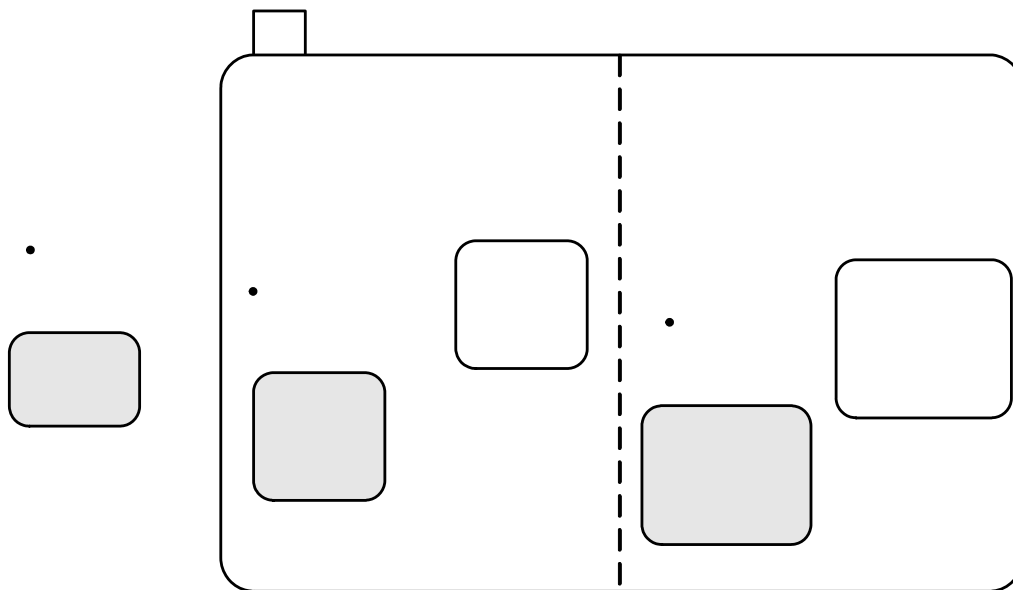


Bild 2-13: Kommunikation zwischen parallelen Automaten

Mit dieser Technik können 1:1 - Beziehungen, aber auch 1:n - Beziehungen aufgebaut werden.

2.2.5 Interruptfähigkeit

Eine Interruptfähigkeit bei Tasksystemen zeigt sich durch folgenden Ablauf:

- ein Interrupt tritt auf
- das aktive Programm wird unterbrochen
- eine Interrupt Service Routine (ISR) wird gestartet und reagiert interruptspezifisch
- nach Beendigung der ISR wird das unterbrochene Programm an der Unterbrechungsstelle fortgesetzt

Besonders wichtig hierbei ist die Programmfortsetzung an der „richtigen Stelle“.

Durch die Hierarchiebildung und die Parallelität der Automaten treten hier ähnliche Situationen auf. In Bild 2-14 ist ein Statechart mit zwei Zuständen gezeigt, wobei ein Zustand ein OR-Zustand ist, d.h. zwei Subzustände enthält. Das Statechart stellt ein Sensorsystem mit Sensordatenverarbeitung dar. Im Zustand ANGESCHLOSSEN kann der Subzustand WARTEN oder der Zustand BERECHNEN aktiv sein. Wenn die Bedingung `sensor_ab` wahr ist, wird der Zustand ANGESCHLOSSEN verlassen, egal in welchem Subzustand sich das System befindet. Wird die Bedingung `sensor_an` wahr, geht das System in den Zustand ANGESCHLOSSEN und gezielt in den Subzustand WARTEN. Die geschieht auch dann, wenn der zeitlich letzte Zustand innerhalb ANGESCHLOSSEN der Zustand BERECHNEN wahr. Das System hat kein „Gedächtnis“.

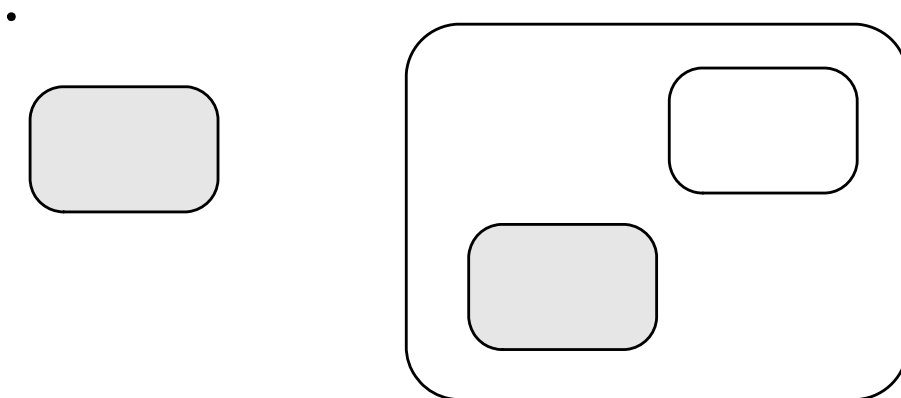


Bild 2-14: System ohne Gedächtnis

Um den Zustand ANGESCHLOSSEN im oben genannten Sinne unterbrechbar zu machen, wird der History Connector eingeführt (Bild 2-15).

Wird der Zustand ANGESCHLOSSEN das erste Mal betreten, dann geht ANGESCHLOSSEN in den Subzustand WARTEN (Default-Übergang). Im weiteren Verlauf jedoch wirkt der History Connector, d.h. der zuletzt eingenommene Subzustand innerhalb ANGESCHLOSSEN wird beim Eintritt eingenommen.

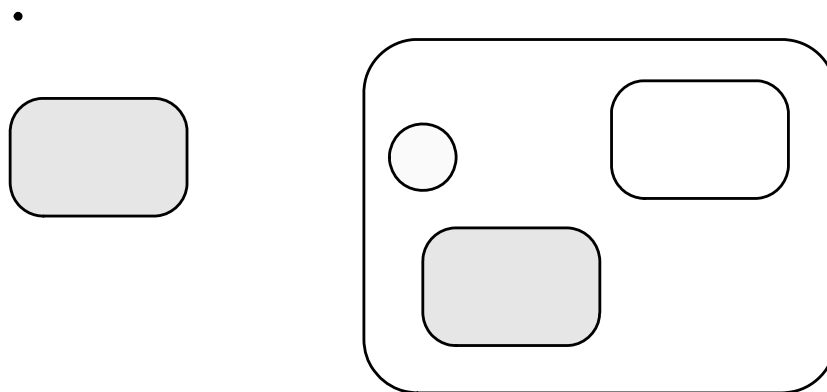


Bild 2-15: System mit Gedächtnis

Wenn der Zustand BERECHNEN weitere Subzustände (z.B. STUFE1 und STUFE2), dann wirkt der History Connector wie in Bild 2-15 benutzt, nicht in diese Subebene hinein. Das „Gedächtnis“ wirkt nur auf der Ebene ANGESCHLOSSEN. Soll das Gedächtnis bis in die tiefste Ebene wirken, dann wird der History Connector wie in Bild 2-16 gezeigt, mit einem Stern versehen (Deep History Connector).

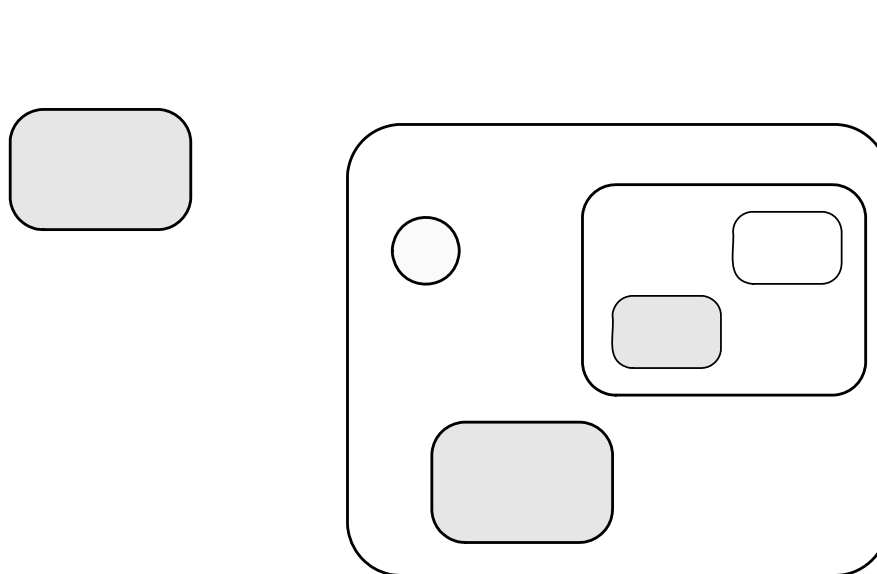


Bild 2-16: Deep History Connector

Um das Gedächtnis auszuschalten, gibt es zwei Actions `history_clear(S)` und `deep_clear(S)`, abgekürzt `hc!(S)` und `dc!(S)`. `hc!(S)` löscht das Gedächtnis des Zustands `S`, so dass das System sich nach `hc!(S)` so verhält, als würde `S` zum ersten Mal betreten. `dc!(S)` löscht das Gedächtnis bis in die letzte Verschachtelungstiefe.

2.2.6 Ausführung des Modells

A Schritte (Steps)

Bei der Ausführung (Simulation) eines Modells werden einzelne Schritte (Steps) durchgeführt. Es ist genau spezifiziert, was in einem Schritt gemacht wird. Es wird davon ausgegangen, dass die Ausführung eines Schritts keine Zeit benötigt. (Der Zustandswechsel in Bild 2-1 vom Zustand S1 nach S2, hervorgerufen durch das Event e und die Condition c, erfolgt in einem Schritt. Im selben Schritt wird die Aktion a ausgeführt.)

Nach einem Schritt befindet sich das System in einem speziellen Systemzustand, zu dem folgende Informationen gehören:

- eine Liste von Zustände, in denen sich das System gleichzeitig befindet
- eine Liste von Activities mit ihren Zuständen
- die aktuellen Werte der Data Items und der Conditions
- eine Liste der Time Out Events mit den Auftrittzeiten
- eine Liste der Scheduled Actions mit den Ausführungszeiten
- eine Liste der Events, die im vorhergehenden Schritt aufgetreten sind
- Informationen, welche die Vergangenheit des Systems (History) betreffen

Ein Schritt wird angestoßen durch externe oder interne Veränderungen. Solche Veränderungen können Actions sein oder auch ein Timeout Event. Ein Schritt ist eine Zwei-Stufen-Prozess mit folgendem Ablauf:

- Schrittanstoß durch Actions und Timeout Events
- Systemreaktion durch die Ausführung von Übergängen, Erzeugung von Events, Durchführung von Static Reactions und Mini-Specs

Generelle Prinzipien sind:

- Reaktionen auf externe und interne Events und Änderungen, die innerhalb eines Schritts stattfinden, werden erst im folgenden Schritt bemerkbar
- Events leben nur für die Dauer eines Schrittes
- Berechnungen innerhalb eines Schritts basieren auf dem Aktivierungsstatus der Zustände und Activities zu Beginn des Schritts und auf den Werten der Data Items und Conditions, die am Anfang eines Schritts vorlagen

Ein Zustand kann im selben Schritt nicht betreten (entered) und verlassen (exited) werden. Wenn in Bild 2-17 a das System sich im Zustand S1 befindet und die Condition c true ist, wird durch das Auftreten von e der Zustandswechsel von S1 nach S2 erfolgen. Der Zustandswechsel von S2 nach S3 erfolgt aber erst im nächsten Schritt. Eine Ausnahme bildet Bild 2-17 b. Hier ist der Ausgangszustand und der Zielzustand identisch, deshalb wird innerhalb eines Schritts der Zustand S verlassen und betreten.

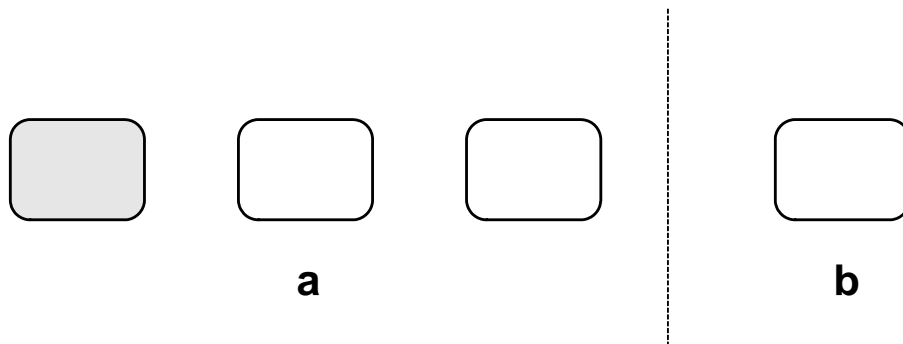


Bild 2-17: Schritt und Zustandswechsel

B Event

Wenn ein Schritt einen Event erzeugt, so ist dieser Event nur während des folgenden Schritts existent und wirksam. Er ist nicht in dem Schritt wirksam, in dem er erzeugt wurde.

In Bild 2-18 wird dies demonstriert. Der folgende Ablauf zeigt die Wirkungsweise:

- Das System befindet sich in den beiden parallelen Zuständen S1 und S3.
- Der Event e tritt auf. Dadurch wird ein Schritt ausgelöst.
- In diesem Schritt wird
 - der Übergang von S1 nach S2 ausgeführt und der Event f erzeugt
 - der Übergang von S3 nach S4 ausgeführt.
- Im folgenden Schritt, initiiert durch f, wird der Übergang von S4 nach S5 ausgeführt.

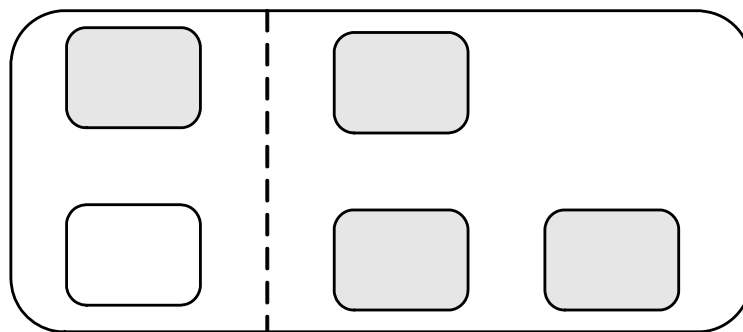


Bild 2-18: Eventerzeugung und Eventwirkung

Ein weiteres Beispiel zeigt Bild 2-19 mit folgendem Ablauf:

- Das System befindet sich in den beiden parallelen Zuständen S1 und S3.
- Der Event e tritt auf. Dadurch wird ein Schritt ausgelöst.
- In diesem Schritt wird
 - der Übergang von S1 nach S2 ausgeführt und der Event f erzeugt
 - der Übergang von S3 nach S4 nicht ausgeführt.
- Im folgenden Schritt wird der Übergang von S3 nach S4 nicht ausgeführt.

Dieser Ablauf kommt daher, weil im ersten Schritt der Event f zwar erzeugt, aber nicht wirksam ist und im darauffolgenden Schritt der Event e nicht mehr existent ist. Deshalb ist die Beziehung e and f falsch und führt zu keinem Event.

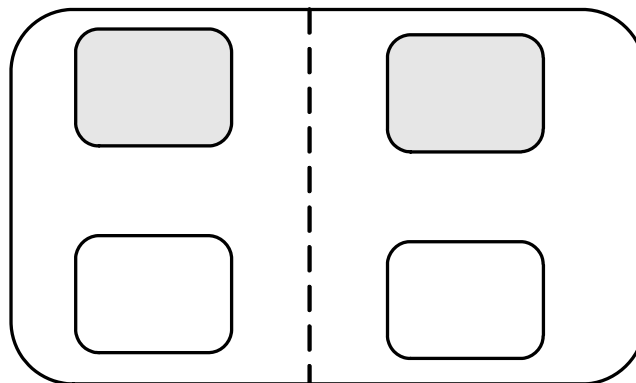


Bild 2-19: Eventerzeugung und Eventwirkung

Eine ausführlichere Beschreibung eines Schritts gibt das Beispiel in Bild 2-20 wieder, welches einen einfachen Zustandsübergang von S1 nach S2 darstellt. Auslöser ist der Event e mit einer verknüpften Action a.



Bild 2-20: Event und Action

Die Transition von S1 nach S2 wird t1 genannt (Syntax gehört nicht zu STATEMATE). Unter der Annahme, daß sich das System in Zustand S1 befindet und das Event e ausgelöst wurde, ergeben sich folgende Systemantworten:

- Die Schaltvoraussetzung von Transition t1 ist erfüllt, weil sich das System im Startzustand der Transition (Zustand S1) befindet und weil der Trigger e ausgelöst wurde. Also wird t1 ausgelöst. Damit wird Zustand S1 verlassen, Zustand S2 wird betreten und Action a wird ausgeführt.
- Die Events $ex(S1)$ und $en(S2)$ werden generiert.
- Die Condition $in(S1)$ wird auf false gesetzt und das Event $fs(in(S1))$ wird generiert; die Condition $in(S2)$ wird auf true gesetzt und das Event $tr(in(S2))$ wird generiert.
- Die Actions, die beim Verlassen des Zustands S1 oder beim Betreten des Zustands S2 ausgeführt werden sollen, werden ausgeführt.
- Da das System vorher im Zustand S war und S auch nicht verlassen wurde während des Schritts, werden die Static Reactions des Zustands S ausgeführt, deren Voraussetzungen

erfüllt sind.

- Alle Activities, die im Zustand S1 als active throughout oder active within definiert sind, werden ausgeschaltet, während die im Zustand B als active throughout (nicht jedoch als die als active within) definierten Activities eingeschaltet werden.

C Superstep

Ein Zustandswechsel kann durch externe Stimuli oder durch interne Stimuli erzeugt werden. Tritt z.B. ein externer Event auf, so kann als Folge nicht nur ein Schritt ausgeführt werden, sondern eine Reihe von Schritten, für die aber interne Stimuli verantwortlich sind. Sind keine internen Ursachen für Zustandswechsel mehr vorhanden, nimmt das System einen sogenannten stabilen Zustand ein. Weiter Zustandsänderungen werden wieder durch externe Stimuli verursacht.

Der Abschnitt von einem stabilen Zustand zum nächsten nennt man Superstep. In Bild 2-21 wird ein Beispiel gezeigt mit folgendem Ablauf:

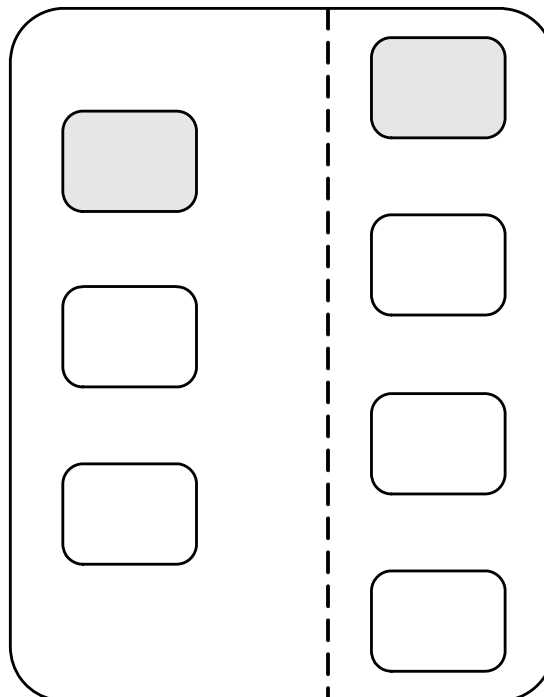


Bild 2-21: Beispiel zu Superstep

- Die Zustände S1 und S4 sind stabile Zustände.
- Der externe Event e wird erzeugt.
- Die Zustände S2 und S5 werden eingenommen und der interne Event f wird erzeugt. S2 ist ein stabiler Zustand.
- Im nächsten Schritt wird der Zustand S6 eingenommen bedingt durch f. S6 ist ein stabiler Zustand.
- Weitere Zustände (S3 bzw. S7) werden erst eingenommen, wenn der Event G oder der Time Out Event nach 5 Zeiteinheiten auftritt. Die Sequenz von S1 bis S2 und die Sequenz S4 bis S6 nennt man einen Superstep.

D Schritt und Zeit

Bei der Modellsimulation wird zwischen einem Simulationsschritt und einem Zeitschritt unterschieden. Für die Beziehung zwischen Schritt und Zeit gibt es zwei unterschiedliche Modi:

- asynchroner Modus (Schritt-unabhängig)
- synchroner Modus (Schritt-abhängig)

In beiden Modi benötigen die Zustandswechsel sowie die Static Reactions innerhalb der Zustände keine Zeit.

Beim asynchronen Modus gibt es keine Beziehung zwischen Schritt und Zeit. Mehrere Schritte können hintereinander ausgeführt werden, ohne dass die Zeit weiterläuft. Die Zeit läuft erst weiter, wenn das System sich in einem stabilen Zustand befindet. Anders ausgedrückt, beim asynchronen Modus wird die Zeit nach jedem Superstep weitergezählt und nicht nach jedem Schritt.

Da Schritt und Zeit unabhängig voneinander sind, gibt es Situationen, in denen sich eine unendliche Abfolge von Schritten ergibt, ohne dass die Zeit sich verändert. In Bild 2-22 ist ein Beispiel gezeigt.



Bild 2-22: Unendliche Schrittfolge "ohne Zeit"

Um diese Fälle zu vermeiden, wird ein Schrittlimit angegeben (z.B. 1000 Schritte). Durch dieses Limit ist auch ein Superstep begrenzt. Der asynchrone Modus wird gewählt, wenn die Simulation in Zusammenhang mit externen Vorgängen steht.

Beim synchronen Modus sind Schritt und Zeit identisch. Die Zeit wird mit jedem Schritt um eine Einheit weitergezählt unabhängig von externen Einflüssen (Stimuli).

2.2.7 Konfliktsituationen Nondeterminismus und Racing

Bei der Modellierung von Steuerungssystemen mit Statecharts können Konfliktsituationen in zweierlei Ausprägung auftreten: Nondeterminismus und Racing.

Ein Nondeterminismus kann dann auftreten, wenn im System zwei oder mehr Übergangsbedingungen zur gleichen Zeit erfüllt sind, die das System in unterschiedliche Zustände überführen würden.

In Bild 2-23 ist die Situation ungeklärt, wenn t_1 und t_2 gleichzeitig auftreten. Zustand S_{11} wird in jedem Fall verlassen, aber es kann nicht entschieden werden, ob in den Zustand S_{12} oder S_{13} übergegangen werden soll. Dies ist ein Fall von Nondeterminismus.

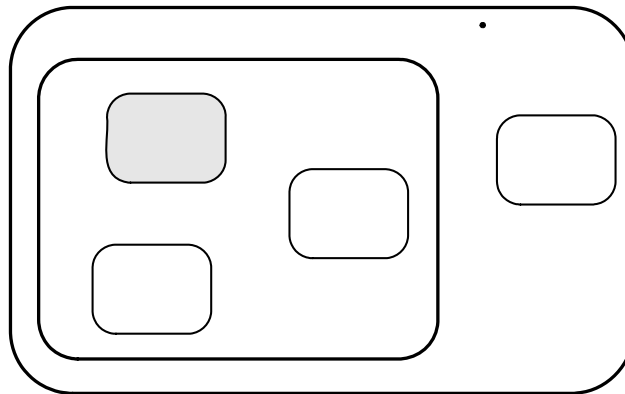


Bild 2-23: Konfliktsituation Nondeterminismus

Anders sieht es aus, wenn t_1 und t_4 gleichzeitig auftreten. Durch den hierarchischen Aufbau der States ergeben sich Bereiche mit verschiedener Priorität und dadurch ergeben sich verschiedenen Prioritäten für die Zustandsübergänge. Der Bereich S hat eine höhere Priorität als der Bereich S_1 , deshalb hat der Übergang t_4 eine höhere Priorität als der Übergang t_1 . D.h. durch die Prioritätsregel ergibt sich beim gleichzeitigen Auftreten von t_1 und t_4 kein Nondeterminismus. Der Übergang t_4 wird in diesem Fall ausgeführt.

Ist eine Modellierung mit t_1 und t_2 wie in Bild 2-23 dargestellt erforderlich, kann eine Eindeutigkeit dadurch erreicht werden, dass z.B. t_1 den Vorrang vor t_2 bekommt. Bild 2-24 zeigt eine Lösung.

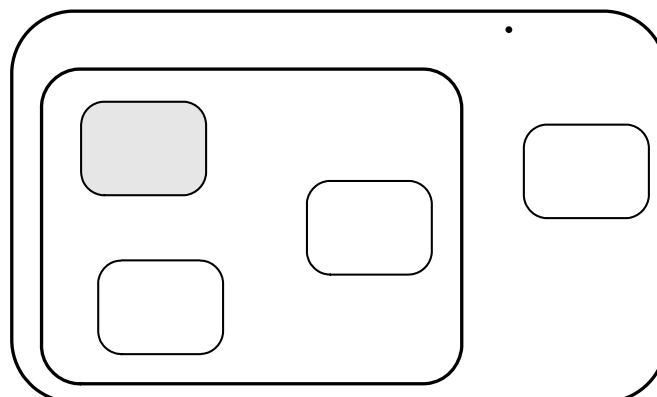


Bild 2-24: Priorisierung zur Vermeidung von Nondeterminismus

Der Fall Racing tritt auf, wenn zur selben Zeit eine Condition oder ein Data Item mehr als einmal verändert wird. Im Bild 2-25 spricht man von Read-Write Racing, im Bild 2-26 von Write-Write Racing.

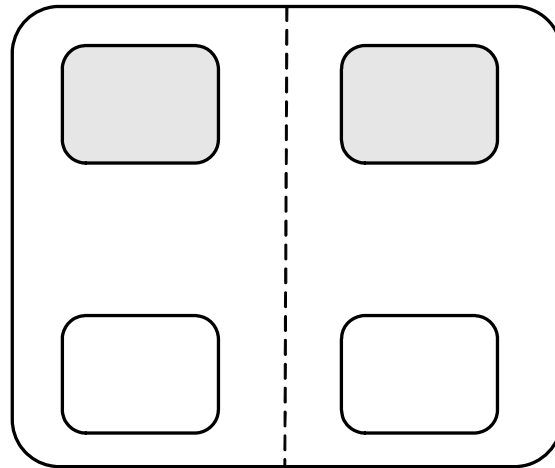


Bild 2-25: Konfliktsituation Read-Write Racing

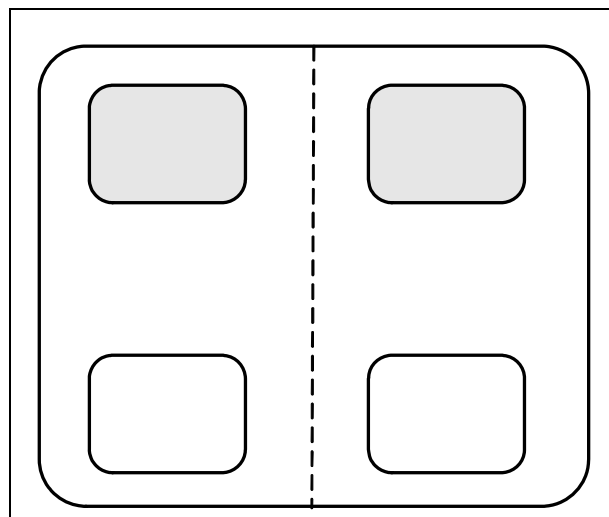


Bild 2-26: Konfliktsituation Write-Write Racing

2.3 Entwurf und Simulation mit STATEMATE

Systeme werden können unter verschiedenen Gesichtspunkten betrachtet und beschrieben werden. In STATEMATE sind folgende Betrachtungsweisen implementiert:

- **Funktionale Sicht**
- **Verhaltenssicht**
- **Struktursicht**

Mit der funktionalen Sicht wird das System in Funktionen und Prozesse aufgeteilt. Zwischen den Funktionen werden Informationen ausgetauscht. Jede Funktion hat Eingangsinformationen, die bearbeitet werden, und erzeugt als Ergebnis der Bearbeitung Ausgangsinformationen.

Die funktionale Sicht möchte die Frage: Was passiert im System? beantworten. Das Entwurfsmittel für die Sicht in STATEMATE sind die Activities (siehe Kapitel 2.3.1).

Aus der Verhaltenssicht heraus wird das zeitliche und dynamische Verhalten des Systems beschrieben. Das Entwurfsmittel sind die bereits betrachteten Statecharts. Aus Verhaltenssicht wird gefragt: Wann passiert etwas?

Die Struktursicht beschreibt das System durch die physikalisch vorhandenen Module. Hier wird das Mittel der Modulcharts eingesetzt und gefragt: Wie wird das System physikalisch realisiert? Diese Sicht der Modellierung werden wir in der Vorlesung zunächst nicht betrachten.

Zwischen den drei Sichten bestehen vielfältige Beziehungen. Durch diese unterschiedlichen Ansichten entsteht ein Gesamtmodell eines Systems, dessen Verständnis durch die Vielfältigkeit des Betrachtens erleichtert wird.

Häufig erweist es sich als vorteilhaft, die Modellierung aus funktionaler Sicht zu beginnen und anschließend das Verhalten zu beschreiben. Zuletzt kann die Realisierung des Systems durch die Gerätestruktur spezifiziert werden.

2.3.1 Systembeschreibung durch Activities

Mit dem Einsatz von Activities erhält man eine Methode, ein System durch funktionale Zerlegung (Decomposition) zu beschreiben. Aus dieser Sicht heraus wird das gesamte System als eine Sammlung von Funktionen angesehen, die miteinander verbunden sind und hierarchisch organisiert sind. Zwischen den Funktionen werden Informationen ausgetauscht.

Jede Activity kann in Subactivities zerlegt werden. Diese können wiederum weiter zerlegt werden, etc. Activities, die keine weitere Zerlegung benötigen, nennt man Basic Activities. Activities, die weitere Subactivities enthalten, nennt man Nonbasic Activities.

Activities werden in STATEMATE durch Rechtecke dargestellt.

Bei der Modellierung eines Systems wird das Gesamtsystem als eine Activity (Top Level Activity) beschrieben. Funktionen, die außerhalb des betrachteten Systems liegen, werden durch gestrichelte Rechtecke dargestellt und externe Activities (external activities) genannt.

Im Bild 2-27 sind die Activities GLÄTTUNG und VERGLEICH Basic Activities. Die Activity DATENERFASSUNG ist die Top Level Activity und gleichzeitig eine Nonbasic Activity. Die Activities SENSOR und ANZEIGE sind externe Activities und bilden die Systemumgebung (environment).

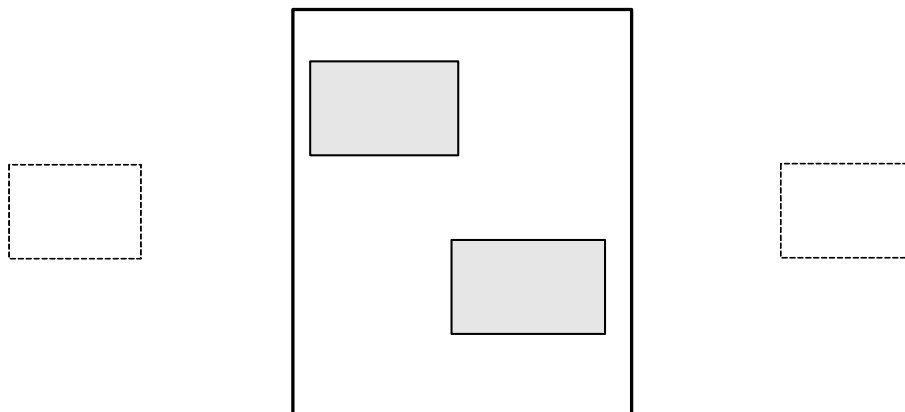


Bild 2-27: Systemdekomposition durch Activities

Zwischen den Activities findet ein Informationsfluß statt. Mit Activities hat man daher eine Methode, das System durch ein Datenflußdiagramm (Data Flow Diagram) zu beschreiben.

Eine Activity beschreibt die Überführung einer Eingangsgröße in eine Ausgangsgröße. Dazwischen führt eine Activity eine Informationsbearbeitung durch. Die Bearbeitung wird durch die Aktivierung (Starten) einer Activity veranlaßt. Durch die Deaktivierung (Stoppen) einer Activity wird die weitere Bearbeitung unterbunden.

INFORMATIONSFLUSS

Der Informationsfluß von einer Activity zur anderen wird durch Linien mit Pfeil am Liniende dargestellt. Geht der Pfeil von der Activity A1 zur Activity A2, so ist A1 der Informationserzeuger und A2 der Informationsverbraucher. Der Pfeil wird mit einem Namen (Label) beschriftet. Mit dem Namen wird ein einzelnes Informationselement (Event, Condition, Data Item) bezeichnet. Weiterhin ist es möglich, an eine Pfeillinie mehrere unterschiedliche Informationselemente anzuschreiben. Der Eintrag der Elemente in diesem Fall geschieht in eine Liste. Diese Linie wird explizit Information Flow bezeichnet. Durch dieses Vorgehen werden die Flußlinien reduziert und die Topologie wird übersichtlicher.

Es existieren zwei unterschiedliche Informationslinientypen:

- Durchgezogene Linie (Data Flow Line)
wird für Informationen benutzt, die innerhalb von Rechenalgorithmen benutzt werden (meist Data Items)
- Gestrichelte Linie (Control Flow Line)
wird für Informationen benutzt, die bei Entscheidungen benutzt werden (meist Conditions)

Diese Unterscheidung wird aber von Statemate nicht weiter verwendet. Man kann also die Data Flow Line benutzen, um den Fluß einer Condition-Größe zu kennzeichnen und umgekehrt. Die verschiedenen Linientypen dienen ausschließlich zur besseren Unterscheidung für die Leser der Statemate-Darstellungen.

In Bild 2-28 sind verschiedenen Situationen von Informationsflüssen dargestellt.

- Information z1 geht vom Rand der Nonbasic Activity A1 zum Rand der Nonbasic Activity A2. Dies bedeutet, daß z1 von einem oder mehreren der Subactivities von A1 erzeugt wird und von einem oder mehreren der Subactivities von A2 benutzt wird.
- Information z2 geht vom Rand von A1 direkt zur Subactivity A21. Dies bedeutet, daß z2 nur von A21 gebraucht wird und z.B. nicht von A22.
- Information z3 geht direkt von Subactivity A12 zum von Rand von A2. Dies bedeutet, daß z3 exakt nur von A12 erzeugt wird und sonst von keiner anderen Activity.
- Information z4 geht genau von Subactivity A12 zur Subactivity A22.
- Bei der Information z5 ist die Informationsquelle noch nicht spezifiziert. Dies wird durch einen kleinen Querstrich am Ausgangspunkt von z5 ausgedrückt.
- Bei der Information z6 ist das Ziel noch nicht exakt spezifiziert. Dies wird durch einen kleinen Querstrich am Ende von z6 ausgedrückt.

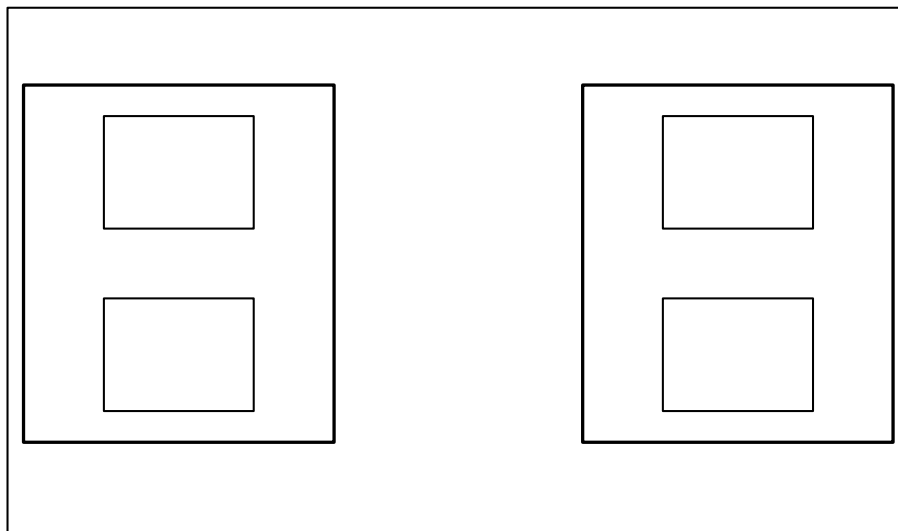


Bild 2-28: Informationsflußlinien

Verbindung zwischen Statecharts und Activity Charts: Control Activity

Mit Activities ist es möglich, ein System aus funktionaler Sicht zu modellieren. Eine Top Level Activity als Repräsentant des Gesamtsystems enthält Subactivities, zwischen denen Informationen fließen. Die Subactivities können wiederum in Subactivities zerlegt werden, etc.

Die Steuerung der Activities übernehmen die sogenannten Control Activities, die durch ein abgerundetes Rechteck dargestellt werden. Durch diese Darstellung wird implizit darauf hingewiesen, daß eine Control Activity ein Statechart darstellt. Die Beziehung zwischen der Control Activity und dem zugehörigen Statechart wird durch den Box-is-Chart-Mechanismus erreicht. Wird das Statechart mit der Bezeichnung NAME benannt, so erhält die Control Activity automatisch die Bezeichnung @NAME und umgekehrt. Weiterhin ist es möglich, der Control Activity einen eigenen Namen (z.B. NAME1) zu geben. Die vollständige Bezeichnung der Control Activity lautet dann NAME1@NAME.

Jede Activity kann maximal eine Control Activity enthalten. Eine Control Activity übernimmt die Steuerung des Informationsflusses und startet bzw. stoppt Activities durch die Actions start(A) bzw. stop(A). A sei der Name der Activity. Für eine Control Activity gilt:

- Control Activities beeinflussen alle ihre Nachbar-Activities, d.h. die Activities auf der selben Hierarchieebene
- Wird eine Activity ohne Control Activity gestartet bzw. gestoppt, werden alle Subactivities gestartet bzw. gestoppt
- Wird eine Activity mit Control Activity gestartet bzw. gestoppt, so wird die Control Activity und damit das zugehörige Statechart gestartet bzw. gestoppt.

Für eine Activity existieren verschiedene Typen, die sich durch ihr Verhalten beim Stoppen der Activity unterscheiden:

- Reactive-controlled Termination Type:

Wenn die Activity aktiv ist, führt sie Schritte aus, reagiert auf Events, etc. Sie muß von außen gestoppt werden.

- Reactive-self Termination Type:

Wenn die Activity aktiv ist, führt sie Schritte aus, reagiert auf Events, etc. Sie kann gestoppt werden

- von außen
- durch Stopp innerhalb des Mini Spec
- durch Erreichen des Termination Connectors (T-Connector)

- Procedure-like Termination Type

Die Activity ist nur während eines Schritts aktiv. Enthält die Activity eine Mini Spec, so wird diese während des Schritts ausgeführt.

Die Activity muß eine Basic Activity sein.

Mini-Specs in Activities

Üblicherweise enthalten Activities Control Activities, welche das Verhalten der Activities steuern. Wenn in einer Activity ausschließlich Daten manipuliert werden sollen, kann dies z.B. durch eine Control Activity mit zugeordnetem Statechart erfolgen, welches die Datenmanipulationen als Static Reactions enthält.

Für diesen Fall ist es aber übersichtlicher die Datenmanipulationen direkt an die Activity zu binden als sogenannte Mini Spec. Die Mini Spec wird ähnlich wie die Static Reactions von Statecharts in das Data Dictionary der Activities eingetragen. Der Name der Activities erhält in diesem Fall ein > angehängt. Ist die Activity vom Typ Procedure-like Termination, so wird die Mini Spec pro Start der Activities genau einmal durchgeführt. Dies entspricht dem Aufruf einer Funktion in einem C-Programm.

Innerhalb einer Mini Spec dürfen keine Bezüge zu Statecharts oder Activities verwendet werden. So sind z.B. keine Conditions wie `in(S)` oder `active(A)` erlaubt.

Modellierungsstrukturen

Beim Entwurf eines Steuerungssystems wird zunächst das zu modellierende System zu seiner Außenwelt abgegrenzt. Die Fragen sind: Was gehört zum System? Welche Komponenten sind außerhalb des Systems (externe Komponenten)? Welche Informationen werden zwischen dem System und den externen Komponenten ausgetauscht?

Der erste Schritt wird die Darstellung des Systems durch eine Top Level Activity sein. Weiterhin werden die externen Komponenten dargestellt und die Informationsflüsse zwischen System und externen Komponenten gezeichnet (Bild 2-29).

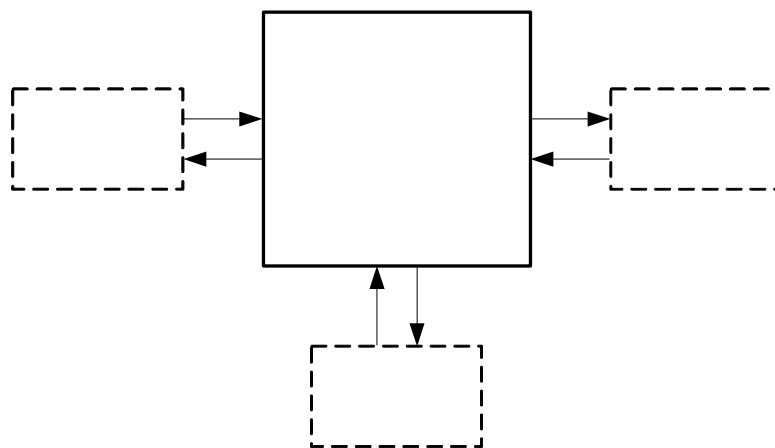


Bild 2-29: Ausgangsstruktur für einen Entwurf

Der Entwurf des Systems selbst kann durch verschiedene Strukturen realisiert werden. Der gängigste Ansatz ist die Zerlegung des gesamten Systems in Subactivities bis hinunter zu Basic Activities. Jede Activity enthält in der Regel eine Control Activity, bzw. eine Mini Spec (Bild 2-30). Die Top Level Activity enthält ebenfalls eine Control Activity, welches die Subactivities steuert.

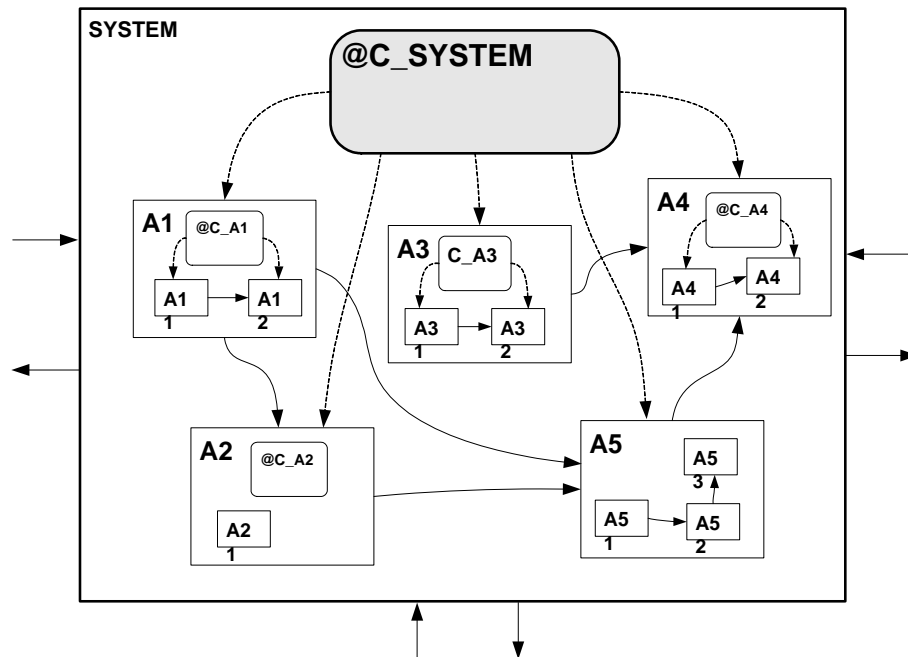


Bild 2-30: Modellierung mit Funktionsdekomposition

Eine weitere Entwurfstruktur ist die Zerlegung des Gesamtsystems ebenfalls in Subactivities. Die Top Level Activity enthält aber keine Control Activity. Dies bedeutet, daß beim Starten der Simulation alle Activities gemeinsam aktiv sind. Die Steuerung der Activities wird durch Events ermöglicht: Eine Activity konsumiert Eingangsinformationen, manipuliert diese Informationen und produziert daraus Ausgangsinformationen. Leitet man aus der Erzeugung der Ausgangsinformationen ein Event ab, so kann dieses Event die Control Activities der "nächsten" Activity anstoßen. Ein Event könnte sein: `wr(data item)` (Bild 2-31).

Für kleinere Systeme kann eventuell auf eine funktionale Dekomposition verzichtet werden. In diesem Fall besteht die Entwurfsstruktur nur aus einer Top Level Activity mit einer Control Activity. Die gesamte Funktionalität wird in die Control Activity verlagert. Vorteil von dieser Struktur ist ein kompakter Auto-C-Code. Nachteil die Unübersichtlichkeit der Struktur.

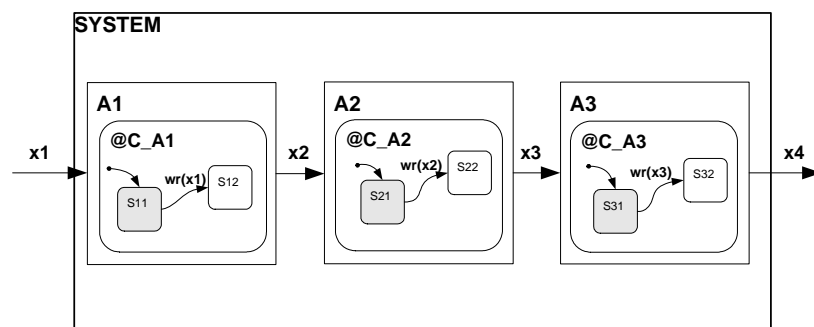


Bild 2-31: Entwurfsstruktur ohne Top Level Control Activity

2.3.2 Oberflächen durch Panels

Zur Visualisierung eines Modells kann eine Oberfläche gestaltet werden, die Bedien- und Anzeigeelemente enthält. Die Schnittstelle zwischen Oberfläche und Modell wird durch Modellvariablen hergestellt. So gibt es z.B. als Visualisierungselement einen Push Button, der mit einem Event des Modells verbunden ist. Wird über den Mauszeiger der Push Button gedrückt, wird der Event ausgelöst und wirkt somit innerhalb des Modells.

Entsprechend arbeiten Anzeigeelemente.