Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen

Bernd Biechele*, Dietmar Ernst*, Frank Houdek**,
Joachim Schmid*, Wolfram Schulte*

Zusammenfassung

Die strukturierte Analyse (SA) definiert eine systematische Vorgehensweise zur Erstellung verständlicher Analysemodelle. In dieser Arbeit vergleichen wir für das Umfeld eingebetteter Systeme zwei SA–Ausprägungen. Um einen praxisrelevanten Vergleich zu ermöglichen, wurde eine realistische Fallstudie herangezogen und aktuelle Werkzeuge, die die betrachteten SA–Ausprägungen unterstützen, mit betrachtet. Diese Arbeit präsentiert einige Ergebnisse einer Studie, die im Rahmen des Softwarelabor Ulm für die Daimler-Benz AG angefertigt wurde. Die vergleichende Betrachtung ergab, daß die Anwendung formaler Methoden und insbesondere eine frühzeitige Simulation sich positiv auf Verständnis und Korrektheit auswirken.

1 Einleitung

Software ist mittlerweile fast allgegenwärtig. So dringt Software nicht nur direkt als eigenständiges Produkt in immer mehr Bereiche vor, sondern auch indirekt in Form eingebetteter Systeme. In diesen stellt die Softwarelösung nur ein Teil der Gesamtlösung dar; sie übernimmt hier Steuerungsfunktionen für die elektrischen und mechanischen Teile des Systems. Eingebettete Systeme steuern mittlerweile oft sicherheitskritische Systeme in der realen Welt. Fehler dürfen in diesem sicherheitsrelevanten Kontext nur selten auftreten und — wenn doch aufgetreten — nur geringe Auswirkung haben.

Vor allem eine klare, eindeutige und widerspruchsfreie Spezifikation des zu entwickelnden Systems als Resultat einer gründlichen Analysephase ist eine notwendige Voraussetzung für Software [Coo91], die den hohen Qualitätsanforderungen genügt, da alle nachfolgenden Aktivitäten der Softwareentwicklung auf dieser aufbauen.

Zur Beschreibung von Anforderungsspezifikationen für eingebettete Systeme sind in den letzten Jahren eine Reihe von Methoden entwickelt worden. Als Hauptvertreter ist dabei die Methode SA/RT (*S*tructured *A*nalysis/*R*eal *T*ime) mit ihren verschiedenen Ausprägungen zu nennen. Weitere bekannte Vertreter sind beispielsweise ROOM [SGW94] und Octopus [AKZ96]. Doch obwohl diese Methoden dasselbe Ziel verfolgen, nämlich eine möglichst präzise Spezifikation von eingebetteten Systemen, unterscheiden sie sich teilweise stark in den zugrundeliegenden Paradigmen. So verfolgt SA/RT nach Hatley und Pirbhai [HP87, HP93] einen semiformalen und auf die Beschreibung sequentieller Vorgänge ausgerichteten Ansatz, währenddessen SA/RT nach Harel [Har87] eine formale Semantik besitzt und die Beschreibung paralleler Vorgänge unterstützt. An dieser Stelle möchten wir jedoch anmerken, daß manche Autoren den Ansatz von Harel nicht als eine Ausprägung von SA/RT, sondern als eine eigenständige Methode ansehen, da sich der Ansatz insbesonders im Hinblick auf die wohldefinierte Semantik stark von anderen SA/RT–Varianten unterscheidet.

Wie in vielen anderen Bereichen zeigt auch bei der systematischen Software–Entwicklung jede Methode in unterschiedlichem Umfeld andere Stärken und Schwächen. Um aber eine Methode entsprechend den Anforderungen der jeweiligen Umgebung auszuwählen, bedarf es Bewertungen, die Aussagen zur Anwendbarkeit und Adäquatheit der verschiedenen Methoden macht. Eine Studie, die eine solche Bewertung im Hinblick auf die SA/RT–Methoden nach Hatley/Pirbhai und Ha-

*Universität Ulm, Fakultät für Informatik, Abt. Programmiermethodik und Compilerbau, 89069 Ulm, e-mail: {bernd, dietmar, joachim, wolfram}@bach.informatik.uni-ulm.de

**Daimler-Benz AG, Forschungszentrum Ulm, Abt. Softwaregestaltung, Postfach 23 60, 89013 Ulm,
e-mail: houdek@dbag.ulm.DaimlerBenz.com

rel vornimmt, wurde von der Daimler-Benz AG beauftragt und im Rahmen des Softwarelabors an der Universität Ulm durchgeführt. Das Softwarelabor ist eine vom Land Baden–Württemberg geförderte Initiative mit dem Ziel, den Wissens- und Technologietransfer zwischen Industrie und Hochschule zu intensivieren. In diesem Dokument beschreiben wir einige Ergebnisse dieser Studie.

Die Motivation der Untersuchung lag vor allem in der Aussage von David Harel [Har92] begründet, die besagt, daß formale Methoden mit Unterstützung der Beschreibung paralleler Vorgänge besonders geeignet für die Spezifikation eingebetteter Systeme seien.

Anhand einer konkreten Spezifikation einer Ampelsteuerung wurde diese Aussage experimentell überprüft, indem mehrere Modellierungen unter Verwendung der betrachteten SA/RT–Ausprägungen erstellt und untersucht wurden. Die Aussage Harels wurde durch diese Untersuchung bestätigt und quantitativ belegt.

Da eine Anwendung der verschiedenen Methoden bei nicht trivialen Anwendungen ohne Werkzeugunterstützung praktisch unmöglich ist, verwendeten wir die folgenden gängigen Werkzeuge: Team*work*[1] (siehe [Cad95]) realisiert eine Umsetzung von SA/RT nach Hatley und Pirbhai (siehe Abschnitt 2.1), und Statemate[2] [i–95] unterstützt SA/RT nach Harel (siehe Abschnitt 2.3).

Bevor wir aber in Abschnitt 3 die durchgeführte Untersuchung sowie in Abschnitt 4 die beobachteten Resultate beschreiben, folgt in Abschnitt 2 zunächst eine Vorstellung der beiden betrachteten SA/RT–Ausprägungen. Anhand eines einfachen Beispiels soll dabei versucht werden, die Gemeinsamkeiten und Unterschiede herauszuarbeiten. Eine Zusammenfassung (Abschnitt 5) rundet den Bericht ab.

2 SA/RT: Modellierungskonzepte und Notationen

Die Spezifikationsmethode SA (*S*tructured *A*nalysis) wurde 1978 von Tom DeMarco entwickelt [DeM78]. Das Ziel dieser Methode ist es, die Anforderungen, die an ein Softwaresystem gestellt werden, durch ein leicht verständliches Modell auszudrücken. Die Sichtweise, die dabei eingenommen wird, ist die des Datenflusses im System. Dieser wird mittels eines Datenflußdiagramms modelliert, das aus Prozessen (funktionale Einheiten des Systems), Datenspeichern (zur Zwischenspeicherung von Datenströmen), Terminatoren (zur Darstellung externer Einheiten) und den eigentlichen Datenflüssen besteht. Neben dieser auf graphischen Elementen basierten Notation gibt es die Möglichkeit, den Aufbau der Datenflüsse sowie die Funktion der Prozesse durch Einträge im Datenlexikon bzw. durch Transformationsbeschreibungen näher zu präzisieren. Außerdem können Datenflußdiagramme hierarchisch verfeinert werden.

Um den unterschiedlichen Anforderungen bei der Entwicklung kommerzieller Informationssysteme auf der einen Seite und hardwarenahen eingebetteten Systemen auf der anderen Seite besser gerecht zu werden, entwickelten sich verschiedene Ausprägungen von SA. So ist die Methode MSA (*M*odern *SA*, siehe [You89]) besonders für die Modellierung großer kommerzieller Informationssysteme geeignet, während SA/RT (*SA* *R*eal–*T*ime) mehr auf die Spezifikation eingebetteter Systeme zugeschnitten ist.

Die Methode SA/RT ist in verschiedenen Ausprägungen in der Literatur bekannt. Ein Ansatz wurde von Ward und Mellor [WM85, WM86] entwickelt, ein weiterer von Hatley und Pirbhai [HP87] und ein dritter von Harel [Har87]. Die letzten beiden Ansätze werden in den Werkzeugen Team*work* bzw. Statemate unterstützt und in den Abschnitten 2.1 bzw. 2.3 näher beschrieben.

Beispiel: Spezifikation einer Drehbank

Um die Beschreibung anschaulicher zu gestalten, werden beide Ansätze anhand eines gemeinsamen Beispiels vorgestellt. Es handelt sich dabei um eine einfache Drehbank, die die folgende Funktionalität aufweisen soll. Mittels eines Startknopfes wird der Motor der Maschine mir langsamer Drehzahl gestartet. Durch zwei Tasten *langsam* und *schnell* kann die Motordrehzahl umgeschaltet werden. Durch Betätigen des Schalters *Bremsen* wird der Motor abgeschaltet und die Bremse aktiviert. Bevor der Motor wieder gestartet werden kann, muß erst ein Knopf für das Ausschalten der Bremse betätigt werden.

[1] Team*work* ist ein eingetragenes Warenzeichen der Firma Cayenne Software, Inc.

[2] Statemate ist ein eingetragenes Warenzeichen der Firma i–Logix Inc.

2.1 Ansatz von Hatley/Pirbhai

Hatley und Pirbhai sehen zur Spezifikation eines Systems im wesentlichen drei Modelle vor, nämlich das *Prozeßmodell*, das *Steuermodell* und das *Informations–* bzw. *Datenmodell*.

Prozeßmodell

Das Prozeßmodell beschreibt die funktionalen Einheiten (die sogenannten *Prozesse*) eines Systems sowie die Beziehungen zwischen diesen Einheiten. Es besteht aus *Datenflußdiagrammen*, *Mini–Spezifikationen* (auch PSPECs genannt) und Einträgen in einem *Datenlexikon*. Das Datenflußdiagramm (DFD) beschreibt den Datenaustausch zwischen Terminatoren und Prozessen sowie zwischen Prozessen. Um eine übersichtliche Darstellung zu ermöglichen, können die beschriebenen Prozesse hierarchisch verfeinert werden. Ist die Verfeinerung hinreichend detailliert, d.h. die Aufgabe der verbleibenden Prozesse ist überschaubar, so werden diese Aufgaben durch Mini–Spezifikationen beschrieben. Eine solche Beschreibung kann sowohl in Prosa als auch unter Verwendung von Pseudocode erfolgen. Im Datenlexikon werden die verwendeten Datenströme hinsichtlich ihres Aufbaus beschrieben. Dazu wird für jeden Datenstrom ein regulärer Ausdruck angegeben.

Informationsmodell

Für die Beschreibung konzeptioneller Datenstrukturen kann das Prozeßmodell mit dem Informationsmodell verknüpft werden, das eine Informationsmodellierung der Datenspeicher in der Entity–Relationship–Notation ermöglicht.

Das Prozeß- und Informationsmodell beschreiben gemeinsam, welche Funktionen das System ausführt und welche Daten dabei auftreten. Ein für eingebettete Systeme relevanter Punkt, nämlich die Beschreibung, wann und unter welchen Bedingungen diese Funktionen ausgeführt bzw. nicht ausgeführt werden, wird dadurch nicht festgelegt.

Steuermodell

Das Steuermodell setzt sich aus dem Kontrollflußdiagramm, Spezifikation von Steuerungsprozessen, Beschreibung zeitlicher Anforderungen und Einträgen im Datenlexikon zusammen.

Im Kontrollflußdiagramm werden die Steuersignale, die zwischen den verwendeten Prozessen fließen, angegeben. Während mit den Datenflüssen der Datenaustausch zwischen den Prozessen modelliert wird, dienen die Steuersignale der Aktivierung von Prozessen. Da eine hohe Bindung zwischen Daten- und Kontrollflußdiagramm besteht (beide beziehen sich auf dieselben Prozesse), werden diese oft als ein Diagramm gezeichnet.

Die Prozeßsteuerung, die ja das eigentliche Anliegen der Steuermodelle ist, wird durch Spezifikation von *Steuerprozessen* (sogenannte CSPECs) festgelegt. Als Beschreibungsformalismus werden dabei endliche Automaten verwendet, wobei hier zwei Ausprägungen möglich sind: Sie können entweder in graphischer Form als Zustandsübergangsdiagramm angegeben werden oder in Tabellenform als Entscheidungstabelle.

Das Zusammenspiel der verschiedenen Modellkomponenten ist in Abbildung 1 graphisch dargestellt. Auf der einen Seite stehen die Datenflußdiagramme, die hierarchisch verfeinert oder mittels PSPECs spezifiziert werden, und auf der anderen Seite die Kontrollflußdiagramme, deren Funktionalität mittels CSPECs näher beschrieben wird. Aus diesen CSPECs heraus können dann Prozesse im Datenflußdiagramm aktiviert werden. Die umgekehrte Richtung wird über PSPECs realisiert, die ihrerseits über Datenmanipulation auf die Prozeßsteuerung einwirken können.

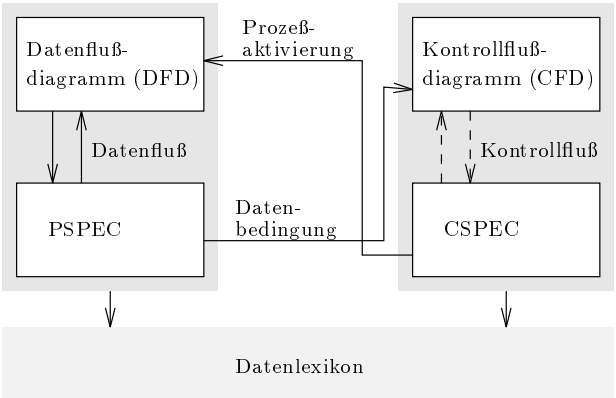

Abb. 1: Zusammenspiel der verschiedenen Modelle.

Eine Modellierung zeitlicher Aspekte wird von Hatley und Pirbhai nur für die Terminatoren, also die externen Einheiten, vorgesehen. Dabei können zwei ver-

schiedene Aspekte modelliert werden, nämlich Wiederholungsraten und Antwortzeiten.

Modellierung des Beispiels

Im folgenden möchten wir anhand einer Modellierung der oben beschriebenen Drehbank den SA/RT–Ansatz nach Hatley/Pirbhai konkretisieren. In Abbildung 2 ist das hierarchisch oberste Datenflußdiagramm (das sogenannte Kontextdiagramm) zusammen mit den Daten- und Kontrollflüssen dargestellt. Hier stellen die Kreise Prozesse dar, während die Rechtecke für Terminatoren stehen. Als solche externe Einheiten werden das Tastenfeld, der Antrieb und die Bremse angesehen. Datenflüsse werden als durchgezogene, Kontrollflüsse durch gestrichelte Linien modelliert. Außerdem wird durch die Zahl hinter dem Namen des Diagramms die Versionsnummer (hier 6) angegeben. In Abbildung 3 finden sich für die Informationsflüsse die zugehörigen Einträge im Datenlexikon. Im Kopf der Definitionen wird dabei nochmals angegeben, ob es sich um einen Kontroll- oder Datenfluß handelt, außerdem besagt der Zusatz *del*, daß es sich bei den Einträgen um diskrete Elemente handelt.

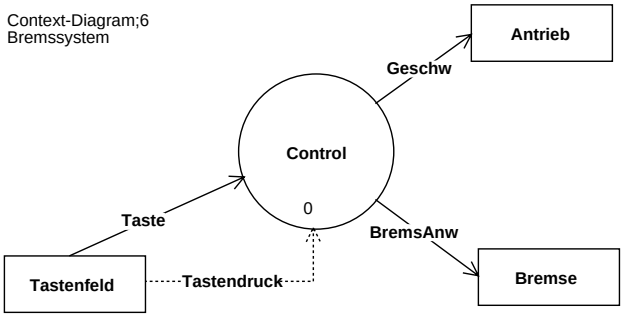

Abb. 2: Daten- und Kontrollfluß–Kontextdiagramm.

Die Drehzahlsteuerung des Motors erfolgt über die Angabe der Betriebsspannung. Die Tastenwerte *G1* und *G2* stehen für niedere bzw. hohe Drehzahl.

In Abbildung 4 findet sich die Verfeinerung des Prozesses 0 (*Control*) aus Abbildung 2, die wiederum Daten- und Kontrollflüsse enthält. Die Zahlenkombinationen am linken oberen Rand der Abbildung (und allen folgenden) gibt an, welcher Prozeß (hier Prozeß 0) genauer spezifiziert ist, und um welche Version (hier 2) es sich handelt.

```
BremsAnw (data flow, del) =
[ "BrEin" | "BrAus" ]

Geschw (data flow, del) =
[ "0V" | "5V" | "7V" ]

Taste (data flow, del) =
["Start" | "G1" | "G2" | "BrEin" | "BrAus" ]

Tastendruck (control flow, del) =
[ "1" | "0" ]
```

Abb. 3: Einträge im Datenlexikon.

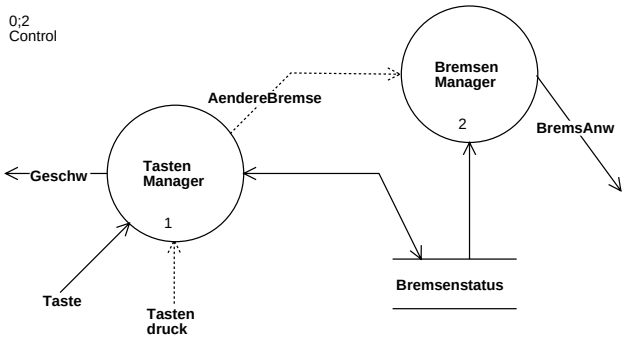

Abb. 4: Verfeinerung des Prozesses *control*.

Der Datenspeicher *Bremsenstatus* kann dabei Einträge vom Typ *BremsAnw* enthalten. Diese Angabe findet sich im Datenlexikon, das hier aber nicht abgebildet ist. Dieser Datenspeicher ist notwendig, damit der Bremsenstatus permanent abgefragt werden kann. Für den Prozeß 1 findet sich in Abbildung 5 eine weitere Verfeinerung.

Für die Prozesse 1.1 (*Setze Geschw*), 1.2 (*Aktiviere Bremse*) und 2 (*Bremsen Manager*) ist eine weitere Verfeinerung nicht mehr sinnvoll, und eine Beschreibung deren Funktionalität mittels Pseudocode ist möglich (siehe Abbildung 6).

Bei dieser Modellierung ergibt sich jedoch das folgende Problem: Werden die Prozesse 1.1 und 1.2 nach dem Drücken einer Taste (*Tastendruck*) in beliebiger Reihenfolge gestartet, so kann es passieren, daß der Motor weiterläuft, obwohl der Bremsschalter betätigt wurde. Um dies zu verhindern, ist es nötig, die Aktivierungsreihenfolge dieser Prozesse zu regeln. Dazu dient die Spezifikation $s1$ eines Steuerprozesses, die in Abbildung 7 angegeben ist. Wird keine Taste (*Ta-*

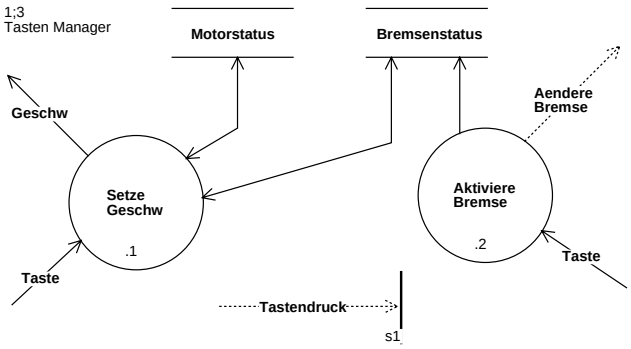

Abb. 5: Verfeinerung des Prozesses *Tasten Manager*.

```
1.1:  if Bremsenstatus = BrAus then
        Case Taste of
          Start: if Motorstatus = Aus
                   then  write An into Motorstatus
                         set Geschw to 5V
          G1:    if Motorstatus = An
                         then set Geschw to 5V
          G2:    if Motorstatus = An
                         then set Geschw to 7V
      else       set Geschw to 0V
                 write Aus into Motorstatus

1.2:  Case Tastendruck of
        1:       write BrAn into Bremsenstatus
        0:       write BrAus into Bremsenstatus
      set Aendere Bremse

2     Case Bremsenstatus of
        BrAn:  set Bremse to BrAn
        BrAus: set Bremse to BrAus
```

Abb. 6: PSPECs der Prozesse 1.1, 1.2 und 2.

Tastendruck	"Setze Geschw"	"Aktiviere Bremse"
1	2	1
0	0	0

Abb. 7: Prozeßaktivierungstabelle für $s1$.

stendruck = 0) gedrückt, so erfolgt auch keine Prozeßaktivierung. Andernfalls wird durch die aufsteigende Numerierung angegeben, daß zuerst Prozeß 1.2 (*Aktiviere Bremse*) aktiviert und nach dessen Beendigung der Prozeß 1.1 (*Setze Geschw*) gestartet wird.

Werkzeug Teamwork

Das Werkzeug Team*work* der Firma Cayenne Software, Inc. (bisher Cadre Technologies Inc., siehe [Cad95]) ermöglicht die Modellierung eines Systems unter Verwendung der eben beschriebenen Ausprägung von SA/RT. Darüberhinaus besteht die Möglichkeit, das Modell zu überprüfen. Diese Überprüfungen beschäftigen sich im wesentlichen mit erweiterten syntaktischen Prüfungen (z.B. Beachtung der Schnittstellen im Zuge der Hierarchisierung, Test auf undefinierte Bezeichner). Außerdem besteht die Möglichkeit, das Modell zu simulieren und zwar in dem Sinne, daß aufgrund der definierten Steuerprozesse ein Prozeßaktivierungsprotokoll erstellt wird. Eine vollständige Simulation ist aber nicht möglich, da die Mini–Spezifikationen keine wohldefinierte Semantik haben und somit nicht ausgeführt werden können.

2.2 Ansatz von Ward und Mellor

Der Ansatz von Ward und Mellor [WM85, WM86] weist große Ähnlichkeiten mit der SA/RT–Ausprägung von Hatley und Pirbhai auf. Während Hatley und Pirbhai zur Beschreibung des Systemverhaltens separate Kontrollflußdiagramme eingeführt haben[3], sehen Ward und Mellor nur ein Diagramm vor. Dieses Diagramm ist ein Datenflußdiagramm, das um für die Prozeßsteuerung relevante Objekte wie Steuerungsprozesse, Puffer und Ereignisflüsse ergänzt ist.

Steuerungsprozesse sind dabei elementare Transformationsprozesse, die eingehende in ausgehende Ereignisflüsse übersetzen. Während Hatley und Pirbhai pro Kontrollflußdiagramm nur einen Steuerungsprozeß vorsehen, ermöglicht der Ansatz von Ward/Mellor die Definition mehrerer Steuerungsprozesse sowie deren graphische Repräsentation.

Die Beschreibung der Ereignisflüsse ist stringenter als die Beschreibung der Kontrollflüsse. Während ein Kontrollfluß einen variablen Inhalt haben kann, beschreibt ein Ereignisfluß nur eine einzelne Information, vergleichbar einem Interrupt oder Impuls. Ward und Mellor unterscheiden drei Arten von Ereignisflüssen, nämlich Signale (Informationen anderer Prozesse ohne Rückkopplung), Aktivierungen und Deaktivierungen anderer Prozesse.

[3] Die Zusammenfassung von Datenfluß- und Kontrollflußdiagrammen, wie sie im Beispiel in Abschnitt 2.1 durchgeführt wurde, ist eine Vereinfachung, die im eigentlichen Ansatz nicht vorgesehen ist.

Unter Puffern werden spezielle Speicher verstanden, die Ereignisse zwischenspeichern können, bis sie von einem Steuerungsprozeß weiterverarbeitet werden.

2.3 Ansatz von Harel

Harel sieht zur Beschreibung eines reaktiven Systems *Activitycharts*, *Statecharts* und *Modulecharts* vor [HP91]. Mit Activitycharts wird, ähnlich wie bei Datenflußdiagrammen nach Hatley und Pirbhai, der Aufbau eines Systems mit Endknoten, Prozessen und Datenflüssen beschrieben. Die verwendeten Informationsflüsse werden auch hier in einem Datenlexikon unter Verwendung regulärer Ausdrücke näher beschrieben.

Die Modulecharts erlauben eine Beschreibung des Systemaufbaus auf einer noch höheren Abstraktionsstufe. Da diese in der Praxis aber wenig Bedeutung haben, werden wir diese im folgenden nicht weiter betrachten.

Prozeßmodell (Activitychart)

Auf die Activitycharts möchten wir hier nur sehr kurz eingehen, da sie den Datenflußdiagrammen von Hatley und Pirbhai stark ähneln. In Abbildung 8 ist das

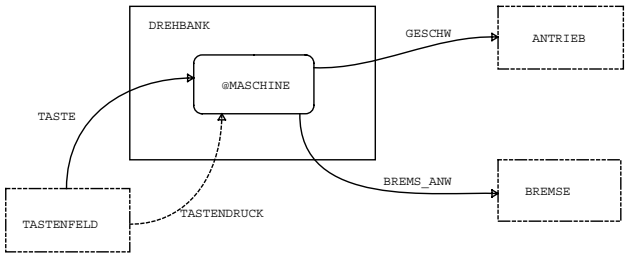

Abb. 8: Das Activitychart für die Drehbank.

Activitychart für die schon weiter oben eingeführte Drehbank zu finden. Man vergleiche diese Abbildung mit Abbildung 2.

Steuermodell (Statechart)

Die wichtigste Beschreibungskomponente beim Ansatz von Harel sind die Statecharts. Diese ermöglichen die Beschreibung des reaktiven Systemverhaltens. Statecharts basieren auf der Idee von Zustandsautomaten, sind aber um die Konzepte Parallelität und Hierarchisierung erweitert. Ein Beispiel für einen solchen parallelen, hierarchischen Zustandsautomaten findet sich in Abbildung 9. Hier wird das Steuerungsverhalten der Drehbank beschrieben.

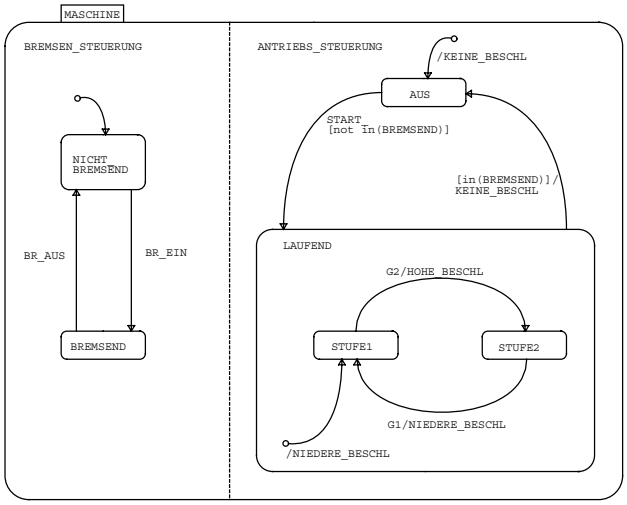

Abb. 9: Hierarchischer und paralleler Zustandsautomat.

An diesem Beispiel lassen sich die wichtigsten Konzepte von Statecharts illustrieren. Der Gesamtautomat *MASCHINE* zerfällt in zwei parallele Teilautomaten *BREMSEN_STEUERUNG* und *ANTRIEBS_STEUERUNG*, die durch eine gestrichelte Linie getrennt werden. Innerhalb von *ANTRIEBS_STEUERUNG* findet sich ein Beispiel für eine Hierarchisierung. Der Zustand *LAUFEND* teilt sich weiter auf in die beiden Zustände *STUFE1* und *STUFE2*. Hier hätte man auch ein neues „Subchart“ zeichnen können, anstatt die beiden Zustände in *LAUFEND* darzustellen.

Übergangsbeschreibung

Die allgemeine Form einer Übergangsbeschreibung, die zwischen zwei Zuständen angegeben wird, lautet

Event[Condition]/Action

wobei *Event* ein Ereignis bezeichnet, das entweder außerhalb oder in einem anderen Teil des Zustandsautomaten erzeugt wurde. Die *Condition* ist eine Bedingung, die erfüllt sein muß, damit dieser Übergang

aktiviert werden kann. *Action* beschreibt, welche Aktionen außer dem Zustandsübergang ausgeführt werden sollen. Dies können entweder Anweisungen an externe Geräte sein, oder aber das Erzeugen eines *Events*, der in einem anderen Teil des Zustandsautomaten verarbeitet wird. Auf diese Weise kann eine Synchronisation zwischen parallelen Teilautomaten erreicht werden. Die einzelnen Teile einer Übergangsbeschreibung sind optional, wobei keine bedingungsfreie Übergänge existieren dürfen, d.h. außer bei Übergängen zu Startzuständen muß wenigstens ein *Event* oder eine *Condition* verwendet werden. Der Ausdruck *[in(BREMSEND)]/KEINE_BESCHL* in Abbildung 9 bedeutet also, daß der entsprechende Übergang vom Zustand *LAUFEND* in den Zustand *AUS* nur dann ausgeführt werden soll, wenn die Bedingung *in(BREMSEND)* wahr ist, d.h. der Automat gerade im Zustand *BREMSEND* ist, und daß bei diesem Zustandsübergang das Ereignis *KEINE_BESCHL* gesendet werden soll.

Neben dem einfachen Generieren von Ereignissen ist es auch möglich, in *Action* komplexere Vorgänge zu beschreiben. Dazu steht eine einfache imperative Programmiersprache zur Verfügung, die auf die Besonderheiten der Zustandsautomaten eingeht. So existieren beispielsweise Funktionen, mit denen das Betreten oder Verlassen eines Zustands überprüft werden. Auch eine Verwendung von Variablen ist möglich.

Werkzeug Statemate

Das Werkzeug Statemate von i–Logix [HLN+90, i–95] ermöglicht eine Modellierung von Systemen auf Basis der soeben beschriebenen SA/RT–Ausprägung nach Harel. Neben der Möglichkeit, Systeme zu modellieren, bietet Statemate aber noch weitergehende Möglichkeiten. Da hier alle beschriebenen Modellelemente und insbesondere auch die verwendeten Programmierkonstrukte eine wohldefinierte Semantik haben, können die erzeugten Modelle vollständig ausgeführt werden.

In der Regel sieht die Modellierung eine Interaktion mit den definierten Endknoten vor. Daher ist es für eine sinnvolle Simulation notwendig, auch deren Verhalten zu spezifizieren. Dazu werden gleichfalls Statecharts verwandt.

Neben der Simulation, die eine sehr gute Möglichkeit bietet, sich von der Adäquatheit der Modellierung zu überzeugen, bietet das System darüber hinaus die Möglichkeit, ablauffähigen Code zu erzeugen und dynamische Tests durchzuführen. Letztere dienen dazu, Sicherheitseigenschaften durch Erreichbarkeitsanalysen automatisch nachweisen zu lassen.

2.4 Vergleich der vorgestellten SA/RT–Ausprägungen

Im folgenden soll versucht werden, die beschriebenen SA/RT–Ausprägungen nach Hatley/Pirbhai und Harel, die auch der in Abschnitt 3 beschriebenen Untersuchung zugrunde liegen, einander gegenüberzustellen und Gemeinsamkeiten sowie Unterschiede herauszuarbeiten. Eine solche Gegenüberstellung soll ein besseres Verständnis der Probleme, die sich im Zuge einer Übertragung von Modellen zwischen diesen beiden Ausprägungen ergeben, ermöglichen.

In Tabelle 1 findet sich eine Gegenüberstellung der beiden Ansätze im Hinblick auf die Aspekte, die bei der Übertragung eine wesentliche Rolle spielen. Im letzten der drei Tabellenabschnitte findet sich auch eine Gegenüberstellung in Bezug auf generelle Spezifikationsprinzipien, wie sie in [Pre92] beschrieben sind.

3 Das Experiment

In Kapitel 2 wurden verschiedene SA/RT–Varianten vorgestellt, insbesondere die von Hatley/Pirbhai und von Harel. Beim Vergleich dieser Varianten in Abschnitt 2.4 haben wir festgestellt, daß eine Spezifikation in Statemate ausführbar und simulierbar ist im Gegensatz zu einer Spezifikation in Team*work*. Da dieser Punkt den Prozeß der Modellierung eines Systems wesentlich vereinfachen und verbessern kann, haben wir in einem Experiment untersucht, ob eine Übertragung einer Team*work*–Spezifikation in eine Statemate–Spezifikation möglich ist, wie groß der Aufwand hierzu ist und welche Vor- und Nachteile eine solche Übertragung bietet.

Das im folgenden beschriebene Experiment wurde unter zwei Hypothesen durchgeführt:

1. Analysemodelle, die ausführbar sind, erleichtern deren Validierung.
2. Für eine verständliche Modellierung sollten abstrakte Konzepte verwendet werden.

Die Untersuchung gliederte sich in zwei Teile: Zuerst wurde eine Übertragung einer konkreten Spezifikation von Team*work* nach Statemate durchgeführt,

Aspekt	*Hatley/Pirbhai*	*Harel*
Modellierung des Systemaufbaus	Datenflußdiagramme	Activitycharts
Verwendete Notation	hierarchisches Datenflußdiagramm	hierarchisches Datenflußdiagramm[4]
Beschreibung der Datenflüsse	Definition durch reguläre Ausdrücke im Datenlexikon	
Modellierung des reaktiven Verhaltens	Prozeßmodell	Statecharts
Zugrundeliegende Beschreibungsformalismen	einfache Zustandsautomaten, Prozeßaktivierungstabellen	hierarchische, parallele Zustandsautomaten
Konvention bei der Prozeßaktivierung	Aufrufer wartet, bis Bearbeitung abgeschlossen ist	Aufrufer arbeitet weiter
Beschreibung der Prozesse	informell (z.B. Prosatext oder Pseudocode)	formal (feste Syntax und Semantik)
Trennung von Funktionalität und Implementierung	erfüllt	durch formale Beschreibungssprache für Prozeßbeschreibung Gefahr einer frühen Implementierung
Beschreibung des umgebenden Systems	nur Angabe der Daten- und Kontrollflüsse zu externen Einheiten	Modellierung des umgebenden Systems möglich
Spezifikation ist ausführbar	in Hinblick auf Prozeßaktivierung	vollständig möglich
Spezifikation tolerant gegenüber Unvollständigkeit	erfüllt, da vage Beschreibung von Prozessen möglich	problematisch, falls Spezifikation ausführbar sein soll

Tab. 1: Gegenüberstellung der SA/RT–Ausprägungen nach Hatley/Pirbhai und Harel.

anschließend wurde diese Spezifikation direkt in Statemate modelliert.

In beiden Fällen handelt es sich dabei um das Musterbeispiel für Team*work* von der Firma Cayenne Software — die Spezifikation einer amerikanischen Ampelsteuerung. Die Ausgangsspezifikation deckt das Spektrum einer einfachen Einbahnstraße mit Fußgängerampel bis hin zu einer komplexen Kreuzung, bestehend aus mehreren Fahrbahnen und Sensoren, ab. Durch die Zuordnung von Fahrbahnen zu Sequenzen und deren zyklische Aktivierung ist eine fast beliebige Ampelschaltung denkbar. Neben einer festen Grünphase besteht auch die Möglichkeit, die Anzahl der ankommenden Fahrzeuge zu berücksichtigen, so daß sich bedarfsgesteuert längere Grünphasen ergeben. Die Grünphase kann auch vorzeitig enden, falls alle wartenden Fahrzeuge die Kreuzung passiert haben.

Da die Modellierung in Team*work* sehr detailliert und daher umfangreich ist, können wir hier nicht die gesamte Übertragung beschreiben. Statt dessen gehen wir auf einige wenige Punkte ein, anhand derer wir die allgemeinen Übertragungsmuster vorstellen.

3.1 Übertragung von Teamwork nach Statemate

Die erste Aufgabe war die entsprechende Modellierung der vorgegebenen Team*work*–Spezifikation in Statemate. Da die Systeme Team*work* und Statemate beide auf dem SA/RT–Ansatz beruhen, gingen wir davon aus, daß die Übertragung keine allzugroßen Schwierigkeiten bereiten dürfte. Das Ergebnis bestätigte zwar diese Annahme, es war trotzdem nicht unbedingt zufriedenstellend. Der Grund lag darin, daß die in Statemate vorhandenen mächtigeren Konstrukte, wie die Hierarchisierung und die Parallelisierung, bei der Übersetzung nur sehr eingeschränkt verwendet wurden.

Im folgenden stellen wir die Übertragung eines wesentlichen Teils der Team*work*–Modellierung vor, es handelt sich um die Steuerung der Ampelphasen.

In Abbildung 10 sind die verschiedenen Komponenten der Spezifikation auszugsweise dargestellt. Die

[4] Dieses Konzept wird aber nicht ganz konsequent verfolgt. So ist es beispielsweise nicht möglich, sich auf Datenflüsse an den Schnittstellen zu übergeordneten Diagrammen zu beziehen.

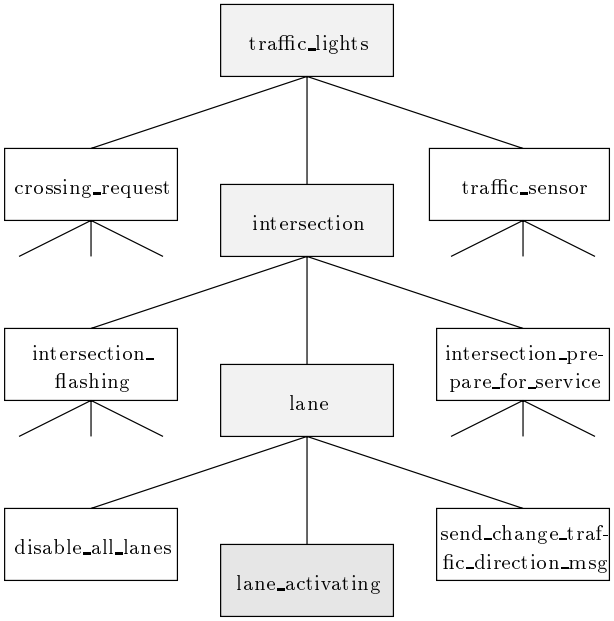

Abb. 10: Die Komponenten der Ampelkreuzung.

Gesamtspezifikation *traffic_lights* zerfällt in drei Teile: Die Komponenten *crossing_request* bzw. *traffic_sensor* regeln die Steuerung der Fußgängerampeln bzw. der Sensoren, die ankommende und abfahrende Fahrzeuge melden; diese beiden Komponenten betrachten wir hier nicht weiter. Die dritte und zentrale Komponente *intersection* regelt den Ampelbetrieb. Dieser Prozeß zerfällt wiederum in drei Subprozesse. Zu Beginn oder in einem Fehlerfall sind die Ampeln einer Kreuzung im Blinkmodus, der durch *intersection_flashing* gesteuert wird. Anschließend werden die Ampeln durch *intersection_prepare_for_service* für den eigentlichen Ampelbetrieb vorbereitet. Jede Ampel einer Fahrbahn wird durch die Komponente *lane* gesteuert. Diese Aufgabe gliedert sich schließlich wiederum in drei Prozesse, dessen wichtigster der Prozeß *lane_activating* ist. Dieser Prozeß, dessen Übertragung wir nun beschreiben, regelt die Farbe der Ampeln nach einer gewissen Logik, die im folgenden vorgestellt wird. Auch äußere Einflüsse, wie beispielsweise die Sensoren, werden dabei berücksichtigt.

Die Grünphasen der Ampeln werden im Normalfall immer zyklisch weitergeschaltet. Da immer dieselben Ampeln gleichzeitig grün anzeigen, werden den einzelnen Ampeln Sequenznummern zugeordnet. Dabei haben die Ampeln, die immer die gleiche Farbe zeigen, auch dieselben Sequenznummern. Zu Beginn zeigen alle Ampeln, denen die Sequenznummer 1 zugeordnet ist, die Farbe grün. Nach Ablauf dieser Grünphase, die durch einen Zeitgeber gesteuert wird, schalten diese Ampeln auf rot und alle Ampeln mit der Sequenznummer 2 auf grün. Dieses Schema setzt sich fort, bis die Ampeln mit der höchsten Sequenznummer grün zeigen, und anschließend wird der ganze Zyklus neu gestartet.

3.1.1 Datenflußdiagramm

Das Team*work*–Datenflußdiagramm aus Abbildung 11 bildet den Rahmen des oben beschriebenen Prozesses. Zu Beginn einer Ampelphase muß bestimmt werden, wie lange eine bestimmte Ampelfarbe aufleuchten soll (Prozeß *determine_duration_of_color*). Der Prozeß benötigt dafür die aktuelle Uhrzeit und Ampelfarbe sowie die Information, um welche Fahrbahn es sich handelt. Diese Information ist in den Datenspeichern *intersection*, *street* und *lane* abgelegt, damit sie je nach Bedarf abgefragt werden kann. Die berechnete Dauer der Ampelphase wird in den Datenspeicher *time_to_interrupt* geschrieben.

Der Prozeß *change_color_of_all_enabled_lanes* steuert die Kommunikation nach außen; er sendet die Umschaltbefehle an die betreffenden Ampeln.

Die Ampelphasen können auch durch äußere Ereignisse beeinflußt werden, einerseits durch Fußgängerampeln, andererseits durch Sensoren, die das Ankommen und Abfahren eines Autos bei einer bestimmten Fahrspur melden. Diese Einflüsse werden durch die Prozesse *service_crossing_request* und *service_threshold_cars_depart_msg* geregelt, auf die hier nicht näher eingegangen wird.

Schließlich wird in dem Zustandsübergangsdiagramm *s1* der generelle Ampelzyklus geregelt, während in der Prozeßaktivierungstabelle *s2* die zu startenden Prozesse bei einem Farbenwechsel einer Ampel angegeben sind. Hierauf werden wir später eingehen.

Übersetzung. Betrachten wir nun das dem vorgestellten Datenflußdiagramm entsprechende Statemate–Activitychart in Abbildung 12. Durch den Vergleich der beiden Abbildungen erkennt man, daß die Datenflußdiagramme in Team*work* und Statemate im strukturellen Aufbau ähnlich sind. Außer den folgenden Unterschieden kann die Übertragung 1:1 erfolgen,

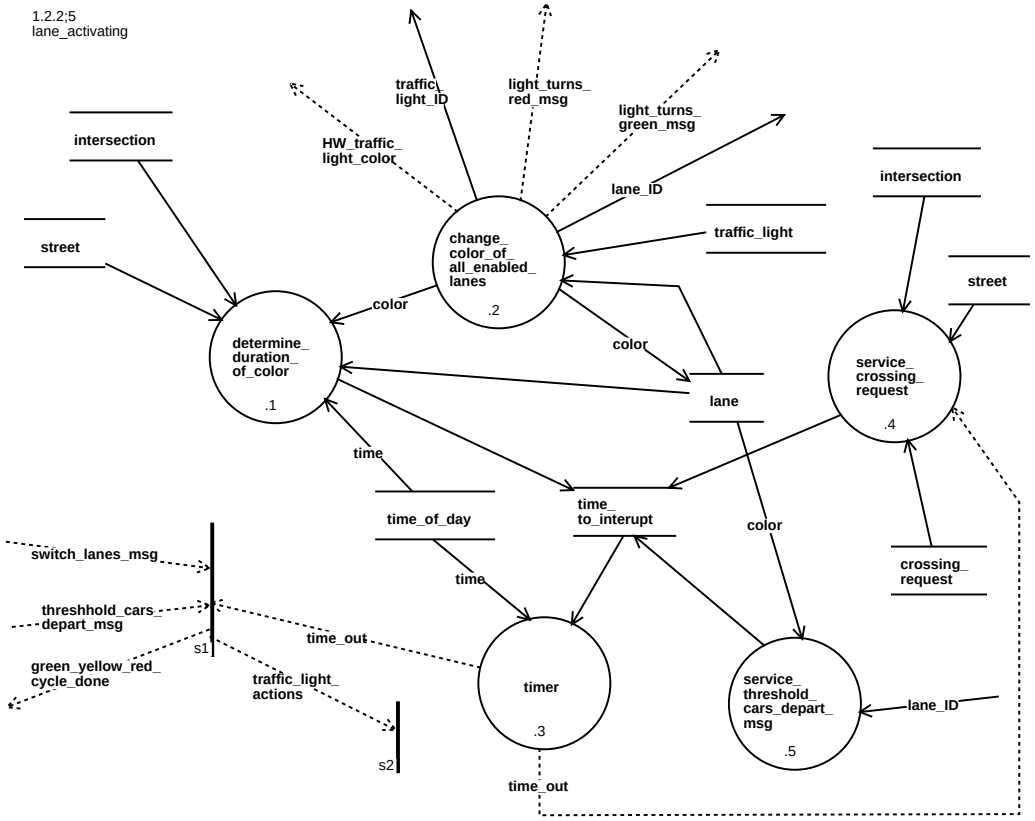

Abb. 11: Datenflußdiagramm in Team*work* für das Schalten der Ampelphasen.

indem die Team*work*–Konstrukte durch die entsprechenden Statemate–Elemente modelliert werden:

- Die sogenannten *offpage*–Flüsse[5] von Team*work* gibt es in Statemate nicht. Sie können entfallen, da diese eine redundante Information darstellen: Im übergeordneten Activitychart ist genau diese Information zu finden.

- Es fällt auf, daß im Datenflußdiagramm qualitativ gleichartige Prozesse im Activitychart unterschiedlich modelliert sind (vergleiche hierzu die Prozesse *determine_duration_of_color* und *service_crossing_request*). Dies ist in der Tatsache begründet, daß die zugehörigen Mini–Spezifikationen sich teilweise direkt übertragen lassen und teilweise Statecharts als Hilfskonstruktion benötigen (siehe Abschnitt 3.1.2).

- Ein Datenspeicher kann zwecks übersichtlicherer Darstellung in Team*work* mehrmals aufgeführt sein. Da dies in Statemate nicht möglich ist, müssen alle Instanzen eines Datenspeichers zu einer Ausprägung vereinigt werden. Dies wurde zum Beispiel für die Datenspeicher *INTERSECTION* und *STREET* durchgeführt.

- Ein Problem ergibt sich bei den Zustandsübergangsdiagrammen: Während in Team*work* ein Datenflußdiagramm beliebig viele Zustandsübergangsdiagramme enthalten kann, darf ein Activitychart in Statemate höchstens ein Statechart enthalten. Das Problem wird dadurch gelöst, daß alle Zustandsübergangsdiagramme in ein Statechart zusammengefaßt werden, das aus den einzelnen parallelen Diagrammen besteht. Auf diesen Punkt gehen wir in Abschnitt 3.1.3 noch ge-

[5] Dies sind Daten– oder Kontrollflüsse zwischen Prozessen eines Diagramms und dessen übergeordnetem Diagramm; im Diagramm haben die entsprechenden Pfeile keine Quelle bzw. kein Ziel.

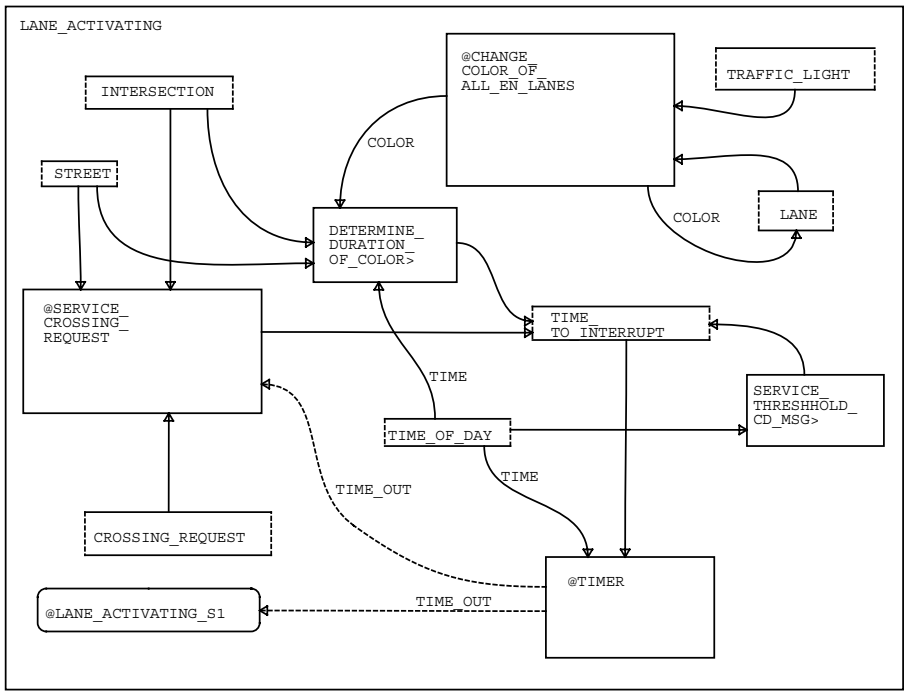

Abb. 12: Statemate–Activitychart für das Schalten der Ampelphasen.

nauer ein.

- Auf weitere kleine Unterschiede, wie beispielsweise die Aufteilung eines Doppelpfeils in zwei einzelne Pfeile oder der unterschiedliche Namensraum für Bezeichner, möchten wir hier nicht näher eingehen.

3.1.2 Mini–Spezifikationen

Im allgemeinen kann für die Übertragung von Mini–Spezifikationen (PSPECs) aus Team*work* nach Statemate kein Übersetzungsschema angegeben werden, da PSPECs in der Regel informell abgefaßt sind.

So auch in unserem Beispiel. Dort wurde für die PSPECs eine Art Pseudocode verwendet. Bei ihrer manuellen Übertragung stießen wir auf das folgende Problem: Eine Minispec muß in Statemate in einem Schritt ausgeführt werden. Deshalb ist es nicht möglich, beispielsweise in einer Schleife mehrere Ereignisse sequentiell zu versenden. Aus diesem Grund haben wir PSPECs, in denen eine solche Schleife oder andere entsprechende Konstrukte vorhanden sind, in ein Statechart übersetzt, in dem die Schleife durch zyklische Zustandsübergänge dargestellt ist.

Beispielsweise wurde dieser Mechanismus für die P-SPEC *change_color_of_all_enabled_lanes* (siehe Abbildung 13) angewandt. Hier wird die angezeigte Ampelfarbe für alle aktiven Fahrbahnen geändert. Welche Fahrbahnen gerade aktiv sind, wird durch eine einfache Schleife abgefragt, und innerhalb dieser Schleife wird einer bestimmten Ampel die neue Farbe zugewiesen.

Die entsprechende Modellierung in Statecharts ist in Abbildung 14 zu sehen: In der äußeren Schleife (*FOR EACH lane*), die durch die Indexvariable *L_IDX* gesteuert wird, wird jede Fahrbahn abgefragt und, falls diese aktiv ist, die Ampelfarbe neu gesetzt. In der inneren Schleife (*FOR EACH traffic_light*), die die Variable *T_IDX* verwendet, wird einer bestimmten Ampel die neue Farbe zugewiesen. Nach Beendigung dieser Zuweisungen muß dieser Prozeß beendet werden; dies wird durch einen sogenannten Terminierungspunkt gekennzeichnet.

Diese Vorgehensweise konnte für alle Schleifen, die in einer PSPEC verwendet wurden, angewandt werden.

```
NAME:
1.2.2.2

TITLE:
change_color_of_all_enabled_lanes

INPUT/OUTPUT:
traffic_light : data_in
traffic_light_ID : data_out
HW_traffic_light_color : control_out
light_turns_green_msg : control_out
light_turns_red_msg : control_out
lane_ID : data_out
color : data_out
lane : data_in

BODY:

FOR EACH lane WHERE lane.<lane>enabled = "TRUE" DO
  CASE lane.<current>color OF
    "green" : SET lane.<current>color = "yellow";
    "yellow": SET lane.<current>color = "red";
    "red"   : SET lane.<current>color = "green";
  ENDCASE;

  FOR EACH traffic_light WHERE traffic_light.lane_ID =
                               lane.lane_ID DO
      SET traffic_light_ID = traffic_light.traffic_light_ID;
      SET HW_traffic_light_color = lane.<current>color;
      SEND HW_traffic_light_color;
  ENDFOR;

  lane_ID = lane.lane_ID;
  CASE lane.<current>color OF
    "green"  : SET light_turns_green_msg  = "TRUE";
    "red"    : SET light_turns_red_msg = "TRUE";
    "yellow" : ;
  ENDCASE;
ENDFOR;
```

Abb. 13: Die PSPEC *change_color_of_all_enabled_lanes* in Team*work*.

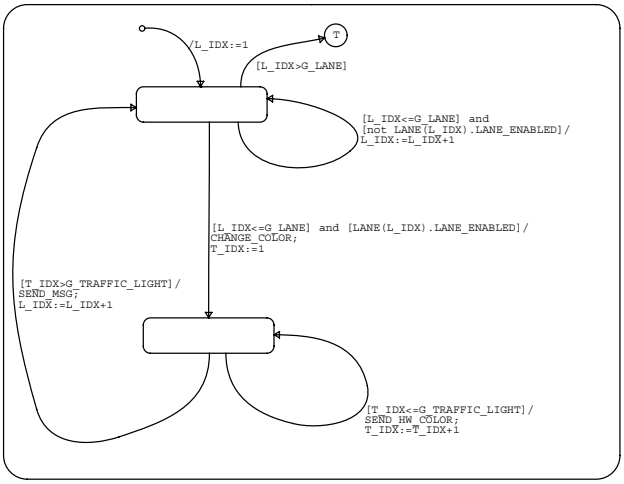

Abb. 14: Die Statemate–Modellierung der PSPEC aus Abbildung 13.

Eine allgemeine Verfahrensweise läßt sich nur dann ableiten, falls der in den PSPECs verwendete Pseudocode hinreichend eindeutig und konsistent eingesetzt ist.

3.1.3 Zustandsübergangsdiagramme

Die Zustandsübergangsdiagramme in Team*work* entsprechen im wesentlichen den Statecharts in Statemate. Betrachten wir hierzu das Zustandsübergangsdiagramm, das den Zyklus der Ampelfarben steuert (siehe Abbildung 15). Zuerst zeigt eine Ampel die Farbe grün. Nach einem Timeout schaltet sie auf gelb, nach einem weiteren Timeout auf rot (Zustand *red_both_ways*). Erst nach einer gewissen Zeit schaltet die Ampel auf den eigentlichen Rot–Zustand und sendet das Ereignis, daß ein Zyklus abgelaufen ist (*green_yellow_red_cycle_done* = "TRUE"). Damit wird gewährleistet, daß die nächsten Ampeln nicht sofort auf grün schalten, sondern zuerst eine gemeinsame

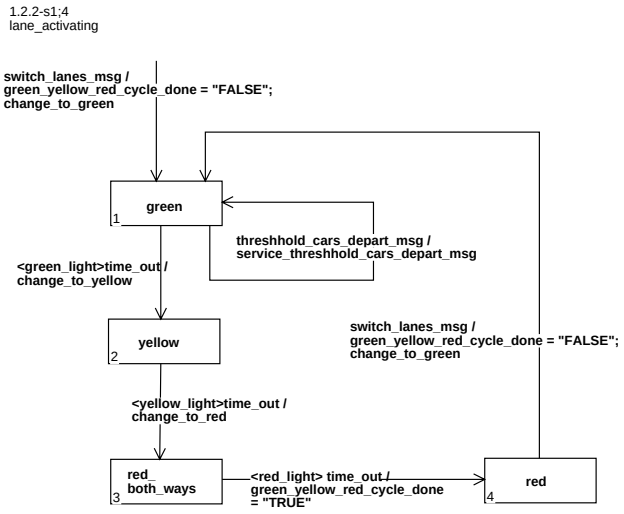

Abb. 15: Zustandsübergangsdiagramm in Team*work*.

Rotphase besitzen. Erst nachdem die Ampel wieder das Signal *switch_lanes_msg* erhält, schaltet sie auf grün.

Übersetzung. Statecharts sind eine Verallgemeinerung von Zustandsübergangdiagrammen. Mit Ausnahme der folgenden Aspekte ist daher eine 1:1–Übersetzung möglich.

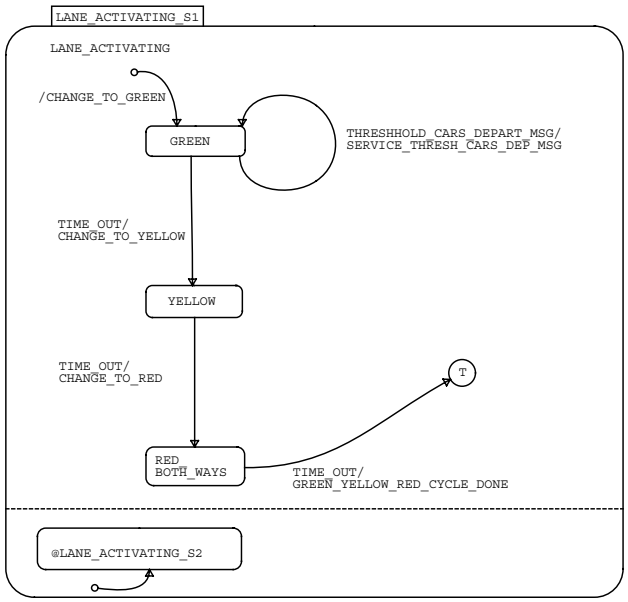

Abb. 16: Zustandsübergangsdiagramm in Statemate.

- Bei einem Vergleich des Team*work*–Zustandsübergangsdiagramms mit dem entsprechenden Statechart (siehe Abbildung 16) wird ersichtlich, daß bei letzterem der Zustandsübergang von *red* nach *green* fehlt. Dies liegt daran, daß im übergeordneten Activitychart (siehe Abbildung 12) der Ampelzyklus neu gestartet wird. Folglich muß er durch das Statechart durch eine Transition zu einem Terminierungspunkt beendet werden, da sonst dieser Prozeß mehrmals aktiviert wäre.
- Weiter fällt auf, daß in diesem Statechart auch das zweite Zustandsübergangsdiagramm aus dem Activitychart mit aufgenommen ist. Wie bereits erwähnt, kann ein Activitychart in Statemate nur ein Statechart enthalten, das deshalb die beiden aus Team*work* notwendigen Zustandsübergangsdiagramme in zwei parallelen Subcharts enthält.

3.1.4 Prozeßaktivierungstabelle

Für Prozeßaktivierungstabellen, die in Team*work* die Reihenfolge der Aktivierung von Prozessen steuern, findet sich in Statemate keine direkt entsprechende Beschreibungsmöglichkeit. Eine mögliche Übersetzung in Statecharts sieht daher folgendermaßen aus.

Übersetzung. In einer Prozeßaktivierungstabelle ist angegeben, welche Aktionen in welcher Reihenfolge unter bestimmten Bedingungen ausgeführt werden sollen. Diese Tabelle kann in Statemate wie folgt über Statecharts realisiert werden:

- Zu Beginn ist dieses Statechart in einem *IDLE*–Zustand. Falls eine Bedingung erfüllt ist, werden all diejenigen Aktionen (in Statemate Aktivitäten) gestartet, die in der Tabelle mit der Zahl 1 versehen sind. Der nächste Zustandsübergang ist mit der Bedingung beschriftet, daß all diese Aktivitäten beendet sind. Erst dann können bei diesem Zustandsübergang die nächsten Aktivitäten gestartet werden.

In Abbildung 17 und 18 finden sich eine Prozeßaktivierungstabelle und das entsprechende Statechart. Betrachten wir zur Erläuterung des Vorgehens bei der Übersetzung die Zeile (†) in Abbildung 17 und die obere Schleife in Abbildung 18.

Falls die Bedingung *change_to_green* wahr ist, muß als erstes der Prozeß *change_color_of_all_enabled_lanes* gestartet werden, da dieser in der Prozeßaktivierungstabelle den Eintrag 1 besitzt. In Statemate wird das Starten eines Prozesses mit der Aktion *st!(process)* ausgelöst. Erst wenn dieser Prozeß beendet ist — ausgedrückt durch *sp(process)* — wird der Prozeß mit der Aktivierungsnummer 2, hier *determine_duration_of_color*, gestartet. Anschließend müssen zwei Prozesse parallel gestartet werden, da diese in der Tabelle die gleiche Aktivierungsnummer besitzen. Danach kehrt das System wieder in den *IDLE*–Zustand zurück.

Es ist zu sehen, daß für die drei Bedingungen *change_to_green*, *change_to_yellow* und *change_to_red* die ausgelösten Prozesse identisch sind. Deshalb wäre es auch möglich, diese drei Bedingungen im Statechart zusammenzufassen, was eine übersichtlichere Darstellung zur Folge hätte.

Falls die Bedingung *service_threshold_cars_depart_msg* wahr ist, muß nur der entsprechende Prozeß gestartet werden. Nach dessen Beendigung kehrt das System wiederum in den *IDLE*–Zustand zurück (siehe Zeile (‡) in Abbildung 17 und die rechte Schleife in Abbildung 18).

3.2 Direktmodellierung in Statemate

Im vorigen Abschnitt stellten wir Übertragungsmuster dar, die von einer Team*work* SA/RT–Spezifikation

1.2.2-s2;1
lane_activating

change_ to_ green	change_ to_ yellow	change_ to_ red	service_ threshhold_ cars_ depart_ msg	1	2	3	4	5
"TRUE"				2	1	3	3	
	"TRUE"			2	1	3	3	
		"TRUE"		2	1	3	3	
			"TRUE"					1
				*determine duration of color *	*change color of all enabled lanes*	*timer*	*service crossing request msg *	*service threshold cars depart msg *

(†) (‡)

Abb. 17: Beispiel einer Prozeßaktivierungstabelle.

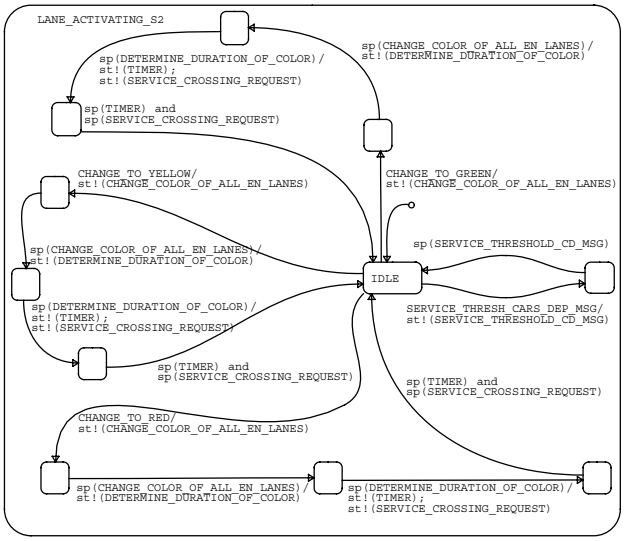

Abb. 18: Übersetzung der Prozeßaktivierungstabelle von Abbildung 17.

zu einer identischen Statemate–Modellierung führten. Wir beobachteten dabei, daß

- sowohl die Ausgangsspezifikation als auch das Übertragungsergebnis unübersichtlich und kompliziert waren,
- einige Teile der Team*work*–Spezifikation nur mit einiger Mühe übertragen werden konnten (z.B. Mini–Spezifikationen),
- die Ausdrucksmöglichkeiten von Statemate teilweise ungenutzt blieben (z.B. Parallelität, Hierarchisierung).

Wir versuchten also eine Direktmodellierung in Statemate mit dem Ziel, eine einfachere, verständlichere und übersichtlichere Spezifikation zu erstellen. Dies sollte insbesonders durch die Ausnutzung von Parallelität und Hierarchisierung in den Zustandsübergangsdiagrammen gelingen.

Dabei sollte natürlich die Funktionalität der Ampelsteuerung erhalten bleiben, wie das freie Konfigurieren der Kreuzungsgeometrie.

Im Umgebungsmodell (siehe Abbildung 19) werden die Datenflüsse außerhalb der eigentlichen Ampelsteuerung beschrieben. Da die Umgebung des Systems, d.h. die Hardwarekomponenten (hier im wesentlichen Ampeln und Sensoren), unverändert ist, entspricht auch das Umgebungsmodell im wesentlichen dem Kontextdiagramm der gegebenen Team*work*–Modellierung.

Anders als bei dem SA/RT–Ansatz nach Hatley/Pirbhai, der Team*work* zugrundeliegt, orientiert sich die Zerlegung bei SA/RT nach Harel am zustandsgesteuerten Verhalten des Systems.

Das Statechart *CONTROL* (siehe Abbildung 20) ist das globale und damit das allgemeinste Diagramm in der hierarchischen Struktur von Zustandsübergangsdiagrammen. Am Anfang befindet sich das System im Ruhezustand *IDLE*. Beim Eintritt des Ereignisses *HW_POWER* wechselt die Steuerung in

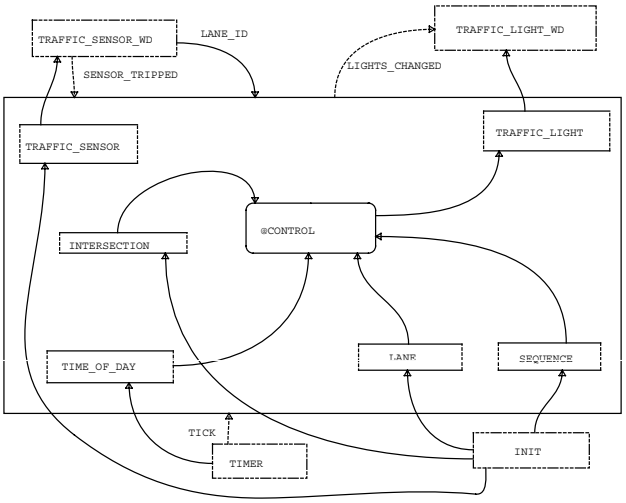

Abb. 19: Das Activitychart für die Ampelkreuzung.

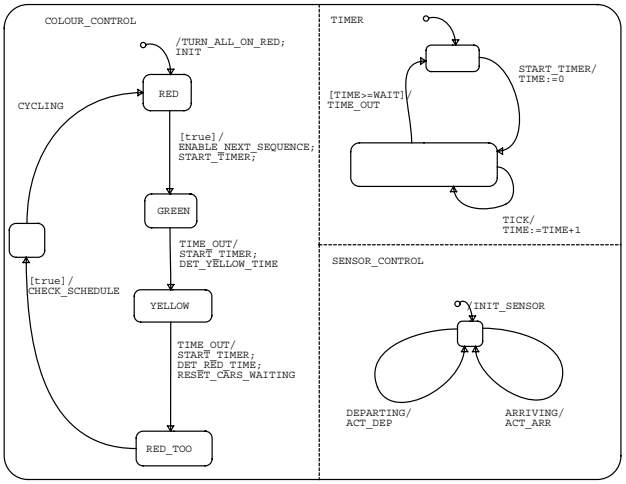

Abb. 21: Die Funktionsweise der Ampel im Normalbetrieb.

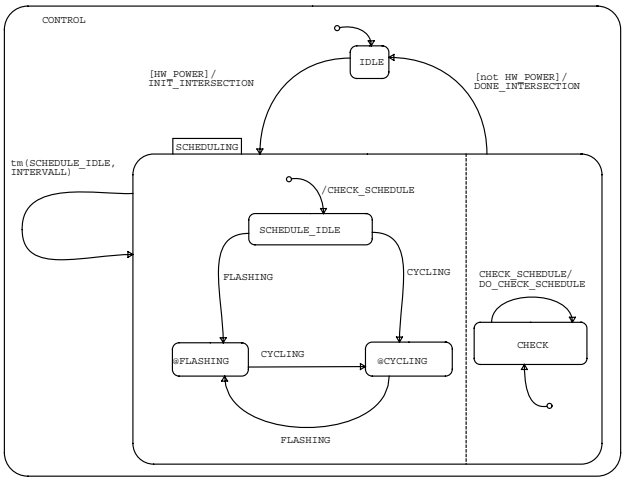

Abb. 20: Das globale Steuerungsmodell.

den Zustand *SCHEDULING*. In diesem werden periodisch die von der Initialisierung her vorgeschriebenen Operationsmodi *SCHEDULE_IDLE*, *CYCLING*, *FLASHING* der Ampeln aktiviert.

Das Subchart *FLASHING* regelt den Ampelbetrieb, wenn die Ampel im Blinkmodus ist. Dieser Fall soll hier nicht näher betrachtet werden.

Interessanter ist das Statechart *CYCLING* (siehe Abbildung 21), das die Steuerung der Ampelleuchten im Normalbetrieb übernimmt. Es besteht aus drei parallelen Teilautomaten. Im Automat *COLOUR_CONTROL* ist die Funktionsweise des Ampelzyklus modelliert: Zunächst werden alle Ampeln auf rot geschaltet. Danach wird die nächste Sequenz ermittelt und die betroffenen Ampeln auf grün geschaltet. Dies leistet die Minispec aus Abbildung 22.

Gleichzeitig wird der Zeitgeber *TIMER* aktiviert, der nach der Zeitdauer für die Grünphase das Ereignis *TIME_OUT* sendet. Anschließend werden die Ampeln auf gelb geschaltet und schließlich auf rot. Die entsprechenden Zeitdauern werden durch die Aktionen *DET_YELLOW_TIME* und *DET_RED_TIME* berechnet und der Variablen *WAIT* zugewiesen.

Parallel zu all dem reagiert die Ampelsteuerung in *SENSOR_CONTROL* auf die Sensoren für ankommende und abfahrende Wagen. Trifft das Ereignis *ARRIVING* bzw. *DEPARTING* ein, so werden die Minispecs *ACT_ARR* bzw. *ACT_DEP* ausgeführt, in denen die Länge der augenblicklichen Ampelphase entsprechend verkürzt oder verlängert wird.

Mit den eben beschriebenen Komponenten ist das Verhalten der vorgegebenen Ampelsteuerung vollständig nachmodelliert, wie auch durch eine Simulation nachgewiesen werden kann (siehe Kapitel 4). Der deutlich geringere Umfang der Direktmodellierung resultiert einerseits aus dem Verständnis, das bei der intensiven Beschäftigung mit der gegebenen Spezifikation gewonnen wurde, und andererseits aus den mäch-

```
ENABLE_NEXT_SEQUENCE (Action) defined in chart: CYCLING
Definition:  $SEQUENCE:=MOD(INTERSECTION.SEQUENCE_NUMBER+1,
                           INTERSECTION.MAX_SEQUENCE_NUMBER);
             $FOUND:=-1;
             $MAXDIFF:=INTERSECTION.MAX_SEQUENCE_NUMBER;
             for $I in 1 to G_LANE loop
              if (LANE($I).CARS_WAITING>0) then
               if LANE($I).SEQUENCE_NUMBER<=INTERSECTION.SEQUENCE_
             NUMBER then
                 $DIFF:=LANE($I).SEQUENCE_NUMBER+
                          INTERSECTION.MAX_SEQUENCE_NUMBER-
                          INTERSECTION.SEQUENCE_NUMBER;
                else
                 $DIFF:=LANE($I).SEQUENCE_NUMBER-
                          INTERSECTION.SEQUENCE_NUMBER;
               end if;
               if ($DIFF<$MAXDIFF) then
                 $MAXDIFF:=$DIFF;
                 $FOUND:=LANE($I).SEQUENCE_NUMBER;
               end if;
              end if;
             end loop;
             if ($FOUND>=0) then $SEQUENCE:=$FOUND; end if;
             WAIT:=SEQUENCE($SEQUENCE).GREEN_TIME;
             INTERSECTION.SEQUENCE_NUMBER:=$SEQUENCE;
             for $I in 1 to G_TRAFFIC_LIGHT
             loop
              if LANE(TRAFFIC_LIGHT($I).LANE).SEQUENCE_NUMBER=
             $SEQUENCE then
               fs!(TRAFFIC_LIGHT($I).RED);
               tr!(TRAFFIC_LIGHT($I).GREEN);
               if LANE(TRAFFIC_LIGHT($I).LANE).CARS_WAITING>=
                  LANE(TRAFFIC_LIGHT($I).LANE).CARS_THR then
                 WAIT:=SEQUENCE($SEQUENCE).GREEN_THR;
               end if;
              end if;
             end loop;
             LIGHTS_CHANGED;
```

Abb. 22: Die Minispec von *ENABLE_NEXT_SEQUENCE*.

tigeren Sprachmitteln, die bei Statemate zur Verfügung stehen und hier eingesetzt wurden.

4 Bewertung des Experiments

In diesem Kapitel wollen wir die Ergebnisse und Beobachtungen der beiden Untersuchungen näher beschreiben, wodurch wir zu den Hypothesen (siehe Seite 7), die dem Experiment zugrundeliegen, Stellung nehmen.

4.1 Übertragung

Die Hauptfrage, die sich bei der beschriebenen Übertragung stellt, ist: Lohnt sich der Aufwand? Bevor wir diese Frage beantworten, wollen wir die Modellierung unter drei Gesichtspunkten näher beleuchten.

Größe

Die Größe der Spezifikation blieb im wesentlichen gleich (siehe Tabelle 2). Unterschiede ergaben sich hauptsächlich bei den PSPECs, die, wie erwähnt, in Statecharts aufgelöst werden müssen, sobald sie eine Schleife enthalten, in der zyklisch bestimmte Aktionen oder Ereignisse ausgelöst werden. Falls die Schleifenrümpfe einen größeren Umfang hatten, wurden diese wieder in einer Minispec realisiert. Dadurch kam die größere Zahl an Komponenten im Steuerungsmodell zustande.

Korrektheit

Die Spezifikation der Ampelsteuerung in Team*work* ist bereits so umfangreich, daß es uns nicht möglich war, die Funktionsweise im Detail zu verstehen. Erschwerend kam hinzu, daß die Spezifikation sehr allgemein gehalten ist. Prinzipiell kann damit eine beliebige Kreuzung mit einer unterschiedlichen Anzahl der Fahrbahnen beschrieben werden. Erst durch die Füllung der Datenspeicher mit konkreten Werten wird die Modellierung auf eine bestimmte Kreuzung zugeschnitten. Eine genaue Beschreibung der Bedeutung dieser Datenspeicher und deren Initialwerte fehlte jedoch. Auch lagen uns dafür keine Beispieldaten vor.

Ein Nachvollziehen der Arbeitsweise der Ampelsteuerung war erst durch die Modellierung in Statemate und der Nutzung der dort möglichen Simulation möglich. Die benötigten Initialwerte wurden dabei durch experimentelles Vorgehen ermittelt. Die Simulation in Statemate brachte einen weiteren Vorteil mit sich: Sie bestätigte den bei der Übertragung bereits vermuteten Verdacht, nämlich daß sich in der gegebenen Team*work*–Modellierung Fehler befanden. Zwei Fehler möchten wir an dieser Stelle beschreiben:

1. Der Prozeß *lane* führt die notwendigen Aktionen beim Wechseln der Ampelfarbe durch. Beim zugehörigen Statechart wird zuerst auf das Ereignis *change_traffic_direction_msg* gewartet. In der übergeordneten Prozeßaktivierungstabelle wird jedoch zuerst dieses Ereignis generiert und dann der Prozeß *lane* gestartet, was zur Folge hat, daß das Ereignis beim Starten des Prozesses nicht mehr anliegt. An dieser Stelle würde also das gesamte System stehenbleiben, da das betreffende Ereignis erst wieder im nächsten Zyklus generiert wird, wozu aber eine Abarbeitung von *lane* notwendig ist. In der Statemate–Version haben wir das Ereignis und den Prozeß parallel gestartet, somit kann der Prozeß auf das Ereignis reagieren.

	Team*work*	Statemate
Prozeßmodell	1 Kontextdiagramm	1 Activitychart
	8 Datenflußdiagramme	8 Activitycharts
Steuerungsmodell	6 Prozeßaktivierungstabellen	6 Statecharts
	24 PSPECs	17 Statecharts und 20 Minispecs

Tab. 2: Die Größe der Spezifikation in Team*work* und Statemate.

2. Zu Beginn des Ampelbetriebs blinken alle Ampeln gelb. Wenn nun auf den regulären Ampelbetrieb umgeschaltet werden soll, wird sofort das Ereignis *change_traffic_direction* gesendet. Dies bewirkt in diesem Fall, daß alle Ampeln mit der Sequenznummer 1 auf grün geschaltet werden, also noch während die anderen Ampeln gelb blinken. Wir haben bei der Übertragung dieses Problem dadurch gelöst, daß beim Übergang in den regulären Ampelbetrieb durch eine zusätzliche Aktion zuerst alle Ampeln auf rot geschaltet werden.

Desweiteren wurden die Fußgängerampeln nur mangelhaft berücksichtigt. Insbesondere fehlte eine konsistente und durchgängige Verbindung der Fußgängerampeln mit der übrigen Ampelsteuerung.

Mit Hilfe der Simulation war es uns jedoch möglich, bis auf die Einbindung der Fußgängerampeln alle Fehler, Inkonsistenzen und Unvollständigkeiten zu korrigieren. Ein formaler Nachweis der Korrektheit durch dynamische Tests konnte aufgrund der Größe der Spezifikation allerdings nicht erbracht werden.

Die Verwendung der Simulation, die auf einem ausführbaren Analysemodell basiert, hat sich als wichtiger Schritt bei der Validierung erwiesen und bestätigt somit unsere erste Hypothese.

Verständnis

Grundsätzlich haben wir den Eindruck gewonnen, daß die Team*work*–Spezifikation zu detailliert ist. Es war deshalb nicht möglich, einen kompletten Überblick über die Modellierung und die zugrundeliegenden Ideen zu gewinnen. Verstärkt wurde dieser Effekt noch durch die Tatsache, daß die Umgebung in Team*work* nur unzureichend mitmodelliert wird. Es werden nur die externen Ereignisse und Datenflüsse angegeben, aber es wird nicht spezifiziert, wie die Umgebung auf diese reagieren soll.

Diese Unzulänglichkeiten hatten auch zur Folge, daß wir die vermuteten Fehler zuerst ignorierten, da diese nicht eindeutig identifiziert werden konnten. Erst die Simulation in Statemate bestätigte diese Fehler.

An dieser Stelle sei auch angemerkt, daß eine Beschreibung der Modellierung und der dabei getroffenen Entscheidungen unerläßlich ist. Diese wäre bei Schwierigkeiten beim Verständnis der Spezifikation sehr nützlich gewesen.

Zur Beantwortung der Frage, ob sich der Aufwand lohnt, müssen die Aufwände und Nutzen einander gegenübergestellt werden. Der Aufwand für die manuelle Übertragung betrug ca. 160 Stunden. Dabei ist die Zeit für Einarbeitung in Statemate und Extraktion der Übersetzungsmuster einbezogen.

Der Nutzen ist diffiziler zu bewerten. Ein Gewinn ist zweifelsohne das bessere Verständnis, das durch eine wohldefinierte Semantik einerseits und der dadurch möglichen Simulation andererseits ermöglicht wird. Gerade aber dieses Verständnis ist notwendig, um die späteren Phasen der Softwareentwicklung sinnvoll durchführen zu können.

Ein weiterer Vorteil ist, daß durch die Simulation eine Art Referenzmodell geschaffen wird, mit dem später das Programm verglichen werden kann.

4.2 Direktmodellierung

Die Direktmodellierung in Statemate hat eine einfachere und verständlichere Spezifikation zum Ergebnis als die zuvor vorgestellte Übertragung. Durch Ausnutzung von Parallelität und Hierarchisierung ergibt sich eine übersichtliche Struktur, was gleichzeitig auch unsere zweite Hypothese untermauert.

Die übersichtliche Struktur spiegelt sich auch in der Größe der Spezifikation wider: Im Gegensatz zur ursprünglichen Modellierung (siehe Tabelle 2) besteht das Ergebnis der Direktmodellierung aus nur einem Activitychart, 3 Statecharts und 13 Minispecs — die Spezifikation ist also auf etwa 20 % des ursprünglichen Umfangs geschrumpft.

Durch die geringere Größe wird insbesondere ein schnelles Verstehen der Spezifikation ermöglicht. Etwaige Fehler können dadurch leichter gefunden werden. Auch die Möglichkeiten zur Verifikation des Modells sind deutlich besser als im ursprünglichen Modell. Während in der Spezifikation, die bei der Übertragung entstanden ist, aufgrund der Größe keinerlei dynamischen Tests möglich waren, konnten in der direkt entwickelten Spezifikation mehrere Sicherheitseigenschaften nachgewiesen werden. So haben wir beispielsweise nachgewiesen, daß die Ampeln zweier konkurrierender Fahrspuren nicht gleichzeitig grün werden können. Allerdings sind wir auch hier relativ rasch an die Grenzen des Statemate–Systems gestoßen.

Die Verwendung von Parallelität hat zwar die schon erwähnten Vorteile, ist aber nicht völlig unkritisch. Es ist zu bedenken, daß es in ungünstigen Situationen zu Nichtdeterminismus und zu gleichzeitigem Variablenzugriff führen kann. Beides kann aber durch die Simulation erkannt und eliminiert werden.

5 Zusammenfassung

In dieser Arbeit haben wir die zwei in der Industrie am weitesten verbreiteten SA/RT–Methoden, nämlich SA/RT nach Hatley/Pirbhai und nach Harel, sowie ihre unterstützenden Werkzeuge Team*work* und Statemate anhand eines realistischen Fallbeispiels miteinander verglichen und bewertet. Folgende Punkte haben sich herausgestellt:

- Große Systeme sind ohne Werkzeugeinsatz nicht kontrolliert zu erstellen, da durch das Werkzeug Konsistenz und Vollständigkeit gesichert werden kann.
- Für eingebettete Systeme ist die Verwendung der Beschreibungsmächtigkeit von hierarchischen und parallelen Zustandsautomaten geeignet und führt zu kürzeren und verständlicheren Darstellungen.
- Für komplexe Systeme ist eine frühzeitige Simulation, die eine weitreichende Formalisierung der Analysemodelle notwendig macht, zu empfehlen, da bei der Simulation Inkonsistenzen und Fehler zuverlässiger entdeckt werden können.

Bei Neuentwicklungen hat also ein formaler Ansatz, wie SA/RT nach Harel, deutliche Vorteile gegenüber semiformalen Ansätzen. Freilich birgt eine solche Formalisierung die Gefahr einer zu frühen Implementierung in sich, da Minispecs durch programmiersprachenähnliche Konstrukte spezifiziert werden. Es gibt jedoch bereits Ansätze, die dies vermeiden, indem zur Beschreibung der Minispecs semantisch höhere Konstrukte wie beispielsweise algebraische Spezifikationen eingesetzt werden [Web96].

Eine Übertragung bereits bestehender Team*work*–Modelle nach Statemate ergibt genauere Spezifikationen, da diese semantisch eindeutig sind. Durch Simulation kann man sich zudem von deren Adäquatheit überzeugen. Der damit verbundene Aufwand ist vor allem dann zu rechtfertigen, wenn — insbesondere bei sicherheitsrelevanten Systeme — die bestehenden Team*work*–Modelle unzureichend (d.h. fehlerhaft, ungenau oder schwer verständlich) sind.

Literatur

[AKZ96] M. Awad, J. Kuusela und J. Ziegler. *Object–oriented Technology for Real–Time Systems: A Practical Approach Using OMT and Fusion.* Prentice–Hall, 1996.

[Cad95] Cadre Technologies Inc. *Teamwork User's Guide, Release 6*, 1995.

[Coo91] J.E. Cooling. *Software Design for Real–time Systems.* Chapman and Hall, 1991.

[DeM78] T. DeMarco. *Structured Analysis and System Specification.* Yourdon Press, Prentice–Hall, Englewood Cliffs, NJ, 1978.

[Har87] D. Harel. *Statecharts: A Visual Formalism for Complex Systems.* Science of Computer Programming, 8:231–274, 1987.

[Har92] D. Harel. *Biting the Silver Bullet: Toward a Brighter Future for System Development.* COMPUTER, Seiten 8–20, Januar 1992.

[HLN+90] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring und M. Trakhtenbrot. *STATEMATE: A Working Environment for the Development of Complex Reactive Systems.* IEEE Transactions on Software Engineering, 16(4):403–414, April 1990.

[HP87] D.J. Hatley und I.A. Pirbhai. *Strategies for Real–time System Specification*. Dorset House Publishing, 1987.

[HP91] D. Harel und A. Pnueli. *The Languages of STATEMATE*. Technischer Bericht i-Logix, Inc., 3 Riverside Drive, Andover, MA 01810, 1991.

[HP93] D.J. Hatley und I.A. Pirbhai. *Strategien für die Echtzeit–Programmierung*. Carl Hanser Verlag, 1993.

[Hum90] W.S. Humphrey. *Managing the software process*. Addison–Wesley, 1990.

[i–95] i–Logix Inc.. *Statemate User Manual, Release 6.1*, 1995.

[Pre92] R.S. Pressman. *Software engineering: A practioner's approach*. McGraw–Hill, 1992.

[SGW94] B. Selic, G. Gullekson und P.T. Ward. *Real–Time Object-Oriented Modeling*. John Wiley & Sons, 1994.

[SN92] F. Schönthaler und T. Németh. *Software–Entwicklungswerkzeuge: Methodische Grundlagen*. B. G. Teubner Stuttgart, 1992.

[Web96] M. Weber. *Combining Statecharts and Z for the Design of Safety–Critical Control Systems*. In: M. Gaudel und J. Woodcock (Herausgeber): *FME '96: Industrial Benefit and Advances in Formal Methods, Third International Symposium of Formal Methods Europe, Proceedings*, Band 1051 der Reihe *Lecture Notes in Computer Science*, Seiten 307–326. Springer–Verlag, 1996.

[WM85] P.T. Ward und S.J. Mellor. *Structured Development for Real–Time Systems*, Band 1–2. Yourdon Press, Prentice–Hall, Englewood Cliffs, NJ, 1985.

[WM86] P.T. Ward und S.J. Mellor. *Structured Development for Real–Time Systems*, Band 3. Yourdon Press, Prentice–Hall, Englewood Cliffs, NJ, 1986.

[You89] E. Yourdon. *Modern structured analysis*. Prentice–Hall, Englewood Cliffs, NJ, 1989.