Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE

E. Canver[1]
Abteilung Künstliche Intelligenz
Universität Ulm

J.-T. Gayen, A. Moik
Institut für Eisenbahnwesen und Verkehrssicherung
TU Braunschweig

Ulmer Informatik-Berichte
Nr. 96-01
21. Februar 1996

Zusammenfassung

Die computergesteuerte "elektrisch ortsbediente Weiche (EOW)" ist ein Produkt aus dem Bereich des spurgeführten Verkehrs und wird in Gleisanlagen eingesetzt, die mit nur geringer Geschwindigkeit befahren werden dürfen. Für einen sicheren und ordnungsgemäßen Betrieb werden dennoch hohe Anforderungen an die Steuerungssoftware der EOW gestellt.
Die EOW ist als Industrieprodukt von hoher praktischer Relevanz. Da die Größe der Steuerungssoftware in einem handhabbaren Rahmen liegt, bietet sich mit dieser Anwendung eine geeignete Fallstudie zur Erprobung der in dem System "Verification Support Environment (VSE)" bereitgestellten formalen Methoden, einem Werkzeug zur formalen Spezifikation und Verifikation von sicherheitsrelevanten Softwaresystemen.
Die Fallstudie wurde von einer interdisziplinären Arbeitsgruppe bearbeitet und es wurde versucht, das anwendungsorientierte Vorgehen eines Ingenieurs und die methodikorientierte formale Vorgehensweise eines Softwareentwicklers miteinander zu verknüpfen. Vorliegender Bericht wendet sich insbesondere an diese beiden Lesergruppen.
Zunächst werden die EOW und die Anforderungen an die Steuerungssoftware beschrieben und der mit VSE zu modellierende Softwareausschnitt definiert. Die Abgrenzung und Schnittstellen dieses Ausschnitts zum Gesamtsystem werden festgelegt. Danach ist an diesem Ausschnitt die VSE-Methodik illustriert. Abschließend sind die Erfahrungen und Schlußfolgerungen aus dieser Fallstudie zusammengefaßt.

[1]Die Arbeiten dieses Autors wurden zum Teil vom Bundesamt für Sicherheit in der Informationstechnik (BSI) gefördert.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Vorliegender Bericht ist das Ergebnis einer Pilotstudie zum Einsatz formaler Methoden in der Softwareentwicklung mit Unterstürzung durch VSE. Beim Einsatz formaler Methoden wird versucht, die entwickelten Spezifikationen und Programme einer mathematischen Analyse durch Beweise zugänglich zu machen, um auf diese Weise ein größeres Vertrauen in die entwickelten Systeme zu erlangen. Das VSE-System und die für VSE spezifische Methodik ist in [2] beschrieben. [1, 4] geben jeweils einen kompakten Überblick zu VSE.
Diese Pilotstudie hat die formale Entwicklung der Steuerungssoftware für eine "elektrisch ortsbediente Weiche" (EOW) aus dem Bereich des spurgeführten Verkehrs zum Gegenstand. Dafür mußte zunächst die "IVV Ingenieurgesellschaft für Verkehrsplanung und Verkehrssicherung GmbH" gewonnen werden, die die Unterlagen [5, 6] zur Verfügung stellte.
Aufbauend auf den vorhandenen Dokumenten und einer Vorführung einer in der IVV aufgestellten elektrisch ortsbedienten Weiche wurde die Entwicklung einer formalen Spezifikation der funktionalen Anforderungen und der Sicherheitsbedingungen angegangen und es wurde ein erster Implementierungsschritt angegeben. Diese Arbeiten wurden am "Institut für Eisenbahnwesen und Verkehrssicherung der Technischen Universität Braunschweig" von einer interdisziplinären Arbeitsgruppe bestehend aus Mitarbeitern der TU Braunschweig und der Universität Ulm bearbeitet. Der größte Teil der aus der formalen Spezifikation und Implementierung erzeugten Beweisverpflichtungen wurde anschließend an der Universität Ulm weiter bearbeitet. Dabei konnten auf dem Weg der Beweisführung Fehler in der ersten Fassung der Spezifikation aufgespürt und beseitigt werden.
Der nachfolgende Text ist folgendermaßen gegliedert: Kapitel 2 gibt eine informelle Beschreibung der "elektrisch ortsbedienten Weiche" sowie ihrer Sicherheits- und funktionalen Anforderungen. In Kapitel 3 wird die Vorgehensweise bei der Entwicklung formaler Spezifikationen für diese Anforderungen, insbesondere die Umsetzung der informell formulierten Anforderungen in eine formale Spezifikation dargestellt. Die Entwicklung einer Verfeinerung in Termini einer Programmiersprache wird in Kapitel 4 beschrieben. Das formal entwickelte Programm kann entsprechend der Beschreibung in Kapitel 5 in ein konventionell entwickeltes Rahmensystem eingebunden werden. In Kapitel 6 werden exemplarisch einige Beweisverpflichtungen erläutert und die Rolle der Beweisführung beim Auffinden von Fehlern dargestellt. Kapitel 7 faßt die Ergebnisse zusammen.
Der Anhang enthält eine komplette Auflistung der entwickelten Spezifikationen und Programme. Eine erweiterte Version dieses Berichtes ([3]) enthält in einem zweiten Anhang die Beweisprotokolle.

Kapitel 2

Informelle Beschreibung

2.1 Prinzipieller Aufbau einer elektrisch ortsbedienten Weiche

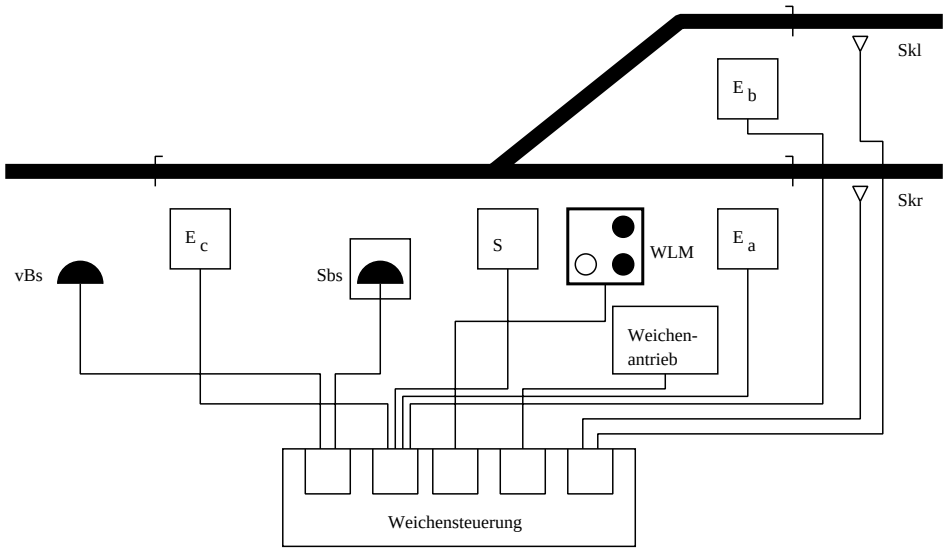


Abbildung 2.1: Elektrisch ortsbediente Weiche

Die "elektrisch ortsbediente Weiche" (im folgenden abgekürzt "EOW") wird in Gleisanlagen eingesetzt, auf denen mit nur geringer Geschwindigkeit (auf Sicht) gefahren wird. Bei der Weiche werden die Verzweigungsrichtungen von der Weichenspitze aus gesehen mit linker und rechter Zweig bezeichnet. In Abb. 2.1 ist also der obere Zweig der linke und der untere der rechte Zweig. Zur Weiche gehören der Weichenantrieb und die Endlagemelder.
Der Weichenbereich wird durch technische Einrichtungen (Gleisstromkreis oder Achszählkreis) frei oder besetzt gemeldet. In der obigen Abbildung sind der Sender (S) und die Empfänger (E_a, E_b, E_c) eines Gleisstromkreises eingezeichnet. Begrenzt wird der Weichenbereich (Gleisstromkreis) durch die eingezeichneten Winkel, die für die Isolierstöße stehen. Ein Weichenlage- und Ordnungsmelder (WLM), der neben der Weiche aufgestellt wird, dient zur Anzeige folgender Weichenzustände:

- rechte oder linke Endlage
- keine Endlage bzw. Weiche gestört (mittlere Lampe blinkt).

Aufträge zum Umstellen der Weiche können erteilt werden von:

- der vorgezogenen Bedienstelle (vBs), einem Schlagtaster, der in genügendem Abstand vor der Weiche aufgestellt ist. Damit wird ein Stellen und Befahren der Weiche ohne vorhergehendes Anhalten ermöglicht.
- der Schlüsselbedienstelle (Sbs) an der Weichenspitze.
- dem linken (Skl) oder rechten (Skr) Schienenkontakt.

2.2 Szenarien

Anhand einiger Beispiele soll nachfolgend die reale Bedienung der EOW veranschaulicht werden.

- Der Weichenlage- und Ordnungsmelder zeigt weithin sichtbar den aktuellen Zustand der Weiche an. Wenn eine Störung oder ein Umlauf angezeigt wird (mittlere Lampe blinkend oder Anzeige aus), darf der Lokomotivführer nicht in den Weichenbereich einfahren.
- Wenn sich ein Zug von der spitzen Seite der Weiche nähert, kann der Lokomotivführer an dem Weichenlage- und Ordnungsmelder ablesen, in welchem Zustand sich die Weiche befindet. Soll die Weiche umgestellt werden, kann der Lokomotivführer diesen Vorgang auslösen, indem er beim Passieren der vorgezogenen Bedienstelle aus dem fahrenden Zug heraus den Schlagtaster betätigt.
- Beim Heranfahren an die Weiche von der stumpfen Seite her geben die Schienenkontakte (Skl, Skr) Signale, damit die Weiche ggf. in die richtige Lage umläuft.
- Die Schlüsselbedienstelle ist an der Weichenspitze aufgestellt und kann von authorisiertem Personal (Schlüssel) dazu verwendet werden, die Weiche selbst in sicherheitskritischen Situationen umzustellen.

2.3 Definition des Modellierungsausschnitts

In dem vorliegenden Anforderungskatalog für die mikroprozessorgesteuerte elektrisch ortsbediente Weiche (MCEOW) für Eisenbahnen des öffentlichen Verkehrs, Eisenbahnen des nichtöffentlichen Verkehrs sowie Stadt-, U- und Straßenbahnen sind die betrieblichen und sicherheitstechnischen Anforderungen niedergelegt. Der Anforderungskatalog gliedert sich in:

- Allgemeine Anforderungen
- Signaltechnische Funktion und Sicherheitsbedingungen
- Anzeigen und Meldungen

Die Mindestanforderungen, die für den Betrieb von Eisenbahn-Stelleinrichtungen gestellt werden, sind in [8, 9, 10, 11] grundsätzlich geregelt. Im Rahmen der formalen Entwicklung wird nur die Softwarekomponente der Weichensteuerung und -sicherung betrachtet. Es findet keine Modellierung des vollständigen Systems Weiche (Hardware und Software) statt.

2.4 Allgemeine Anforderungen an das System

2.4.1 Grundfunktionen

In der Weichensteuerung soll eine Umlaufsteuerung, d.h. eine Steuerung des Motors im Weichenantrieb, und eine kontinuierliche Zustandsüberwachung realisiert werden. Dazu werden an die Steuerung folgende Anforderungen gestellt:

- Übereinstimmung zwischen der Weichenlage, den Lagekriterien der Weichensteuerung und der Lageanzeige
- Selbsttätiges Herstellen der Betriebsbereitschaft nach dem Einschalten/Netzwiederkehr.
- Sichere Anzeige der überwachten Endlage durch einen "Weichenlage- und Ordnungsmelder" in der Außenanlage.
- Zulassungsprüfung der Stellaufträge in Abhängigkeit von der Quelle und dem Systemzustand.
- Steuerung des Weichenumlaufs und Zeitüberwachung.
- Verhinderung unzeitiger Umstellungen.
- Erkennen von Auffahrmeldungen.
- Umstellung der Weichenzungen aus jeder Lage, auch bei nichterreichter Endlage.

2.4.2 Bedieneinrichtungen

Für die Umstellung der Weiche können folgende Einrichtungen vorgesehen werden:

- Schlüsselbedienstelle als Weichenhilfstaste, in der Regel im Weichenlagemelder eingebaut
- Schlagtaster als vorgezogene Bedienstelle vor der Weichenspitze
- Schienenkontakte für die Umstellung von der stumpfen Seite her

2.4.3 Schnittstellen

Es sind Schnittstellen zu folgenden Peripheriegeräten vorzusehen:

- Weichenantrieb und darin enthalten die Weichenendlagemelder
- Weichenlage- und Ordnungsmelder
- Gleisfreimeldekreis bzw. Achszählgruppe

2.5 Signaltechnische Funktion und Sicherheitsbedingungen

2.5.1 Ordnungsstellung

Folgende Voraussetzungen müssen erfüllt sein, damit die Weiche in Ordnungsstellung steht.

- Die Weichenzungen müssen sich in der Endlage innerhalb der vorgeschriebenen Toleranzen und in Übereinstimmung mit den Lagekriterien der Weichensteuerung und den Lageanzeigen befinden.
- Die angeschlossene Gleisfreimeldeeinrichtung muß sich in einem eindeutigen Frei- oder Besetztzustand befinden.

2.5.2 Umstellbedingungen

Stellaufträge von der vorgezogenen Bedienstelle sind nur zulässig, wenn

- der Antrieb sich in Ordnungsstellung befindet und
- die zugehörige Gleisfreimeldung keinen Besetztzustand zeigt.

Ist das Element gestört, aufgefahren oder zeigt es den Besetztzustand an, so ist die Umstellung nur durch Bedienung mit der Schlüsseltaste zulässig.

Stellaufträge von den Schienenkontakten dürfen nur ausgeführt werden, wenn

- der Gleisbereich nicht besetzt gemeldet ist,
- der Antrieb sich im ungestörten Ruhezustand befindet,
- bei Umlaufanforderung aus dem rechten Zweig überwachte Linkslage vorliegt (und umgekehrt).

Stellaufträge von der Schlüsselbedienstelle sind auch in folgenden Fällen zulässig (Umstellen der Weiche unter Beobachtungszwang)

- Gleisbereich besetzt,
- Endlage nicht erreicht oder Umlaufstörung,
- Weiche wurde aufgefahren,
- Reversieren der Umlaufrichtung.

2.5.3 Steuerbefehle

Es werden grundsätzlich nur Einzelkommandos an die Weichensteuerung übertragen.

2.5.4 Signaltechnische Sicherheitsbedingungen bei Ausfällen des Steuerrechners

Der Ausfall des Steuerrechners darf nicht zur Anzeige einer überwachten Endlage oder zum Einschalten des Antriebs führen. Es muß sichergestellt sein, daß der Weichenlage- und Ordnungsmelder blinkt bzw. dunkel bleibt[1].
Nach Wiederherstellung der Funktionsfähigkeit des Steuerrechners muß die Betriebsbereitschaft selbsttätig hergestellt werden, auch wenn zwischenzeitlich die Lage durch Kurbeln geändert wurde.

[1]Diese Anforderung kann natürlich nicht in der Software realisiert werden, sondern muß durch entsprechende Hardwarelösungen erfüllt werden

Kapitel 3

Formale Spezifikation

3.1 Grundzüge der VSE-Methodik

Die Vorgehensweise folgt der in [2] beschriebenen Methodik. Die Grundzüge dieser Methodik sind nachfolgend beschrieben. Die komplette formale Entwicklung großer Software-Systeme wäre sehr komplex und aufwendig und daher in der Praxis nicht durchzuführen. Daher wird zunächst der formal zu entwickelnde Softwareanteil festgelegt. Ein gut mit der VSE-Methodik verträglicher Ansatz ist, nur den sicherheitskritischen Kern der (für Fehler besonders anfälligen) Software formal zu entwickeln. Dazu werden die sicherheitsrelevanten Anforderungen extrahiert und in eine formale Spezifikation umgesetzt, die in VSE Sicherheitsmodell heißt.
Die Funktionalität des festgelegten Ausschnitts wird in einer sogenannten Leistungsspezifikation formal spezifiziert. Die Leistungsspezifikation muß konsistent mit dem Sicherheitsmodell sein. Das wird ausgedrückt, indem beide Spezifikationen in Relation zueinander gesetzt werden. Das VSE-System erzeugt daraus Beweisverpflichtungen, deren Nachweis sicherstellt, daß die Sicherheitsanforderungen auch in der Leistungsspezifikation erfüllt sind.
Die Leistungsspezifikation wird durch ein abstraktes Programm implementiert. Weitere vom VSE-System erzeugte Beweisverpflichtungen dienen dazu, die Korrektheit der Implementierung relativ zur Leistungsspezifikation zu zeigen.
Der nicht sicherheitsrelevante Teil der Software kann mit Hilfe der konventionellen Methoden (z.B. Strukturierte Analyse) entwickelt werden. Am Ende des Entwicklungsprozesses wird das abstrakte Programm in eine konventionelle Programmiersprache umgesetzt und beide Teile, der formal und der konventionell entwickelte, werden wieder zu einem Gesamtsystem zusammengesetzt (siehe Abb. 3.1).
Formale Spezifikationen und abstrakte Programme werden in VSE-SL, der in VSE eingesetzten Sprache, formuliert.

3.2 Vorgehensweise

Ausgangspunkt für die Auswahl des formal zu entwickelnden Ausschnitts sind die informellen Anforderungen aus dem in der Regel meist nur informell verfaßten Lastenheft. Bevor für den mit VSE zu entwickelnden Ausschnitt eine formale Spezifikation entwickelt wird, muß zunächst die Art der Modellierung festgelegt werden. Hierfür bietet VSE zwei Möglichkeiten: funktionale (abstrakter Datentyp) oder zustandsbasierte Modellierung (Zustandsübergangssystem). Die zu treffende Wahl hängt von der Art der Aufgabe und den "Neigungen" des Entwicklers ab. Ein Ingenieur wird für die Steuerung eines elektromechanischen Systems, wie z.B. einer Weiche, in der Regel die zustandsbasierte Modellierung wählen.
Wurde die zustandsbasierte Modellierung gewählt[1], wird als nächstes aus dem Lastenheft

[1] Wir gehen nachfolgend nur auf die zustandsbasierte Modellierung ein.

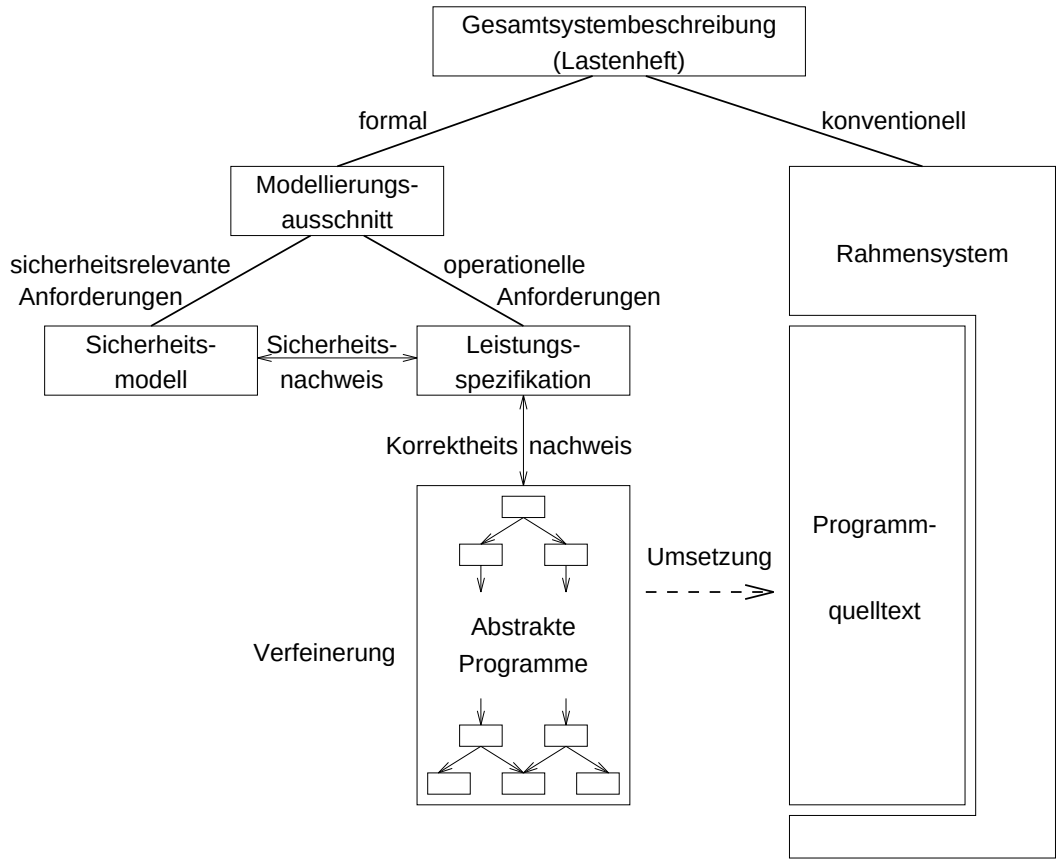


Abbildung 3.1: Modellierung in VSE

das Sicherheitsmodell entwickelt. Dies geschieht durch die Extraktion und Formalisierung der informellen Sicherheitsanforderungen. Dazu sind die folgenden Schritte notwendig:

- Festlegung der Zustandsobjekte und ihrer Wertebereiche (Datenstrukturen)
- Festlegung der Operationen auf diesen Zustandsobjekten
- Umsetzung der Sicherheitsanforderungen in mathematische Formeln, die Systeminvarianten oder Vor- bzw. Nachbedingungen der Operationen beschreiben.

Das Ergebnis ist eine stark strukturierte formale Spezifikation – das Sicherheitsmodell.
Ein System muß neben den Sicherheitsanforderungen auch eine Aufgabe erfüllen. Deswegen werden im nächsten Schritt die Anforderungen an die Funktionalität aus der informellen Beschreibung (Lastenheft) extrahiert. Dadurch kommen in der Regel weitere (nicht sicherheitskritische) Zustandsobjekte hinzu.
Die Beschreibung des gesamten Zustandsübergangssystems erfolgt zuerst konventionell mit Hilfe eines Zustandsübergangsdiagramms, das in einem weiteren Schritt formalisiert wird. Die Umsetzung des Zustandsübergangsdiagramms erfolgt, indem die für die Zustandsübergänge notwendigen Operationen durch mathematische Formeln für Vor- bzw. Nachbedingungen definiert werden. Diese Formeln beschreiben Zustände vor bzw. nach Ausführung der Operationen. Die in diesem Schritt entstandene formale Beschreibung wird Leistungsspezifikation genannt.
Die Konsistenz der Leistungsspezifikation gegenüber dem Sicherheitsmodell muß in einem weiteren Schritt mit mathematischen Methoden (Beweisen) gezeigt werden. Damit wird

sichergestellt, daß die im Sicherheitmodell formulierten Anforderungen von dem in der Leistungsspezifikation beschriebenen System erfüllt werden. Das VSE-System generiert aufgrund der VSE-SL zugrundeliegenden Semantik, [2], alle dafür notwendigen Beweisverpflichtungen.
Folgender Aspekt sollte allerdings beachtet werden: Meist werden in Spezifikationen auch Annahmen über die Umwelt oder Anforderungen an das Rahmensystem formuliert. Wenn z.B. aus physikalischen und betrieblichen Gründen ausgeschlossene Zustände und Zustandsübergänge durch die Formulierung von Vorbedingungen schon am Anfang ausgeschlossen werden, kann dadurch eine formale Beschreibung vereinfacht werden. Dies kann aber auch dazu führen, daß Annahmen über das Rahmensystem gemacht werden, die in VSE nicht formal nachgewiesen werden können, da das Rahmensystem in der Regel nicht formal modelliert wird bzw. in VSE nicht formal modelliert werden kann; dieser Fall tritt ein, wenn die vom abstrakten Programm bereitgestellten Leistungen in einer Endlosschleife[2] genutzt werden sollen. Die Korrektheit der Annahmen muß konventionell validiert werden.
In einem weiteren Schritt wird die (zum großen Teil noch recht abstrakte) Leistungsspezifikation weiter verfeinert. Die Vorgehensweise wird genauer in Kapitel 4 vorgestellt. Das Ergebnis dieser Verfeinerung sind abstrakte Programme, die in ihrer syntaktischen Struktur eng an konventionelle imperative Programmiersprachen angelehnt sind. Diese Programme müssen die Anforderungen der Leistungsspezifikation erfüllen. Auch hierfür generiert das VSE-System die notwendigen Beweisverpflichtungen.
Da somit das abstrakte Programm die Anforderungen der Leistungsspezifikation und die Leistungsspezifikation die Anforderungen des Sicherheitsmodells erfüllen, ist sichergestellt, daß auch das abstrakte Programm den Sicherheitsanforderungen genügt.
Um ein lauffähiges System zu erzeugen, wird im letzten Schritt das abstrakte Programm in den Quellcode einer konventionellen Programmiersprache umgesetzt. Diese Umsetzung kann manuell oder mit dem VSE-System automatisch (zur Zeit Ada) erfolgen. Dieser Schritt wurde in der vorliegenden Fallstudie nicht durchgeführt.

3.3 Modellierung der EOW

Die Steuerungssoftware der EOW wird als ein Zustandsübergangssystem modelliert, das die Weiche kontrolliert und steuert, indem es von außen gelieferte Sensorwerte geeignet auswertet und Ausgangssignale für den Weichenantrieb und den Weichenlage- und Ordnungsmelder setzt. Die Kontrolle erfolgt so, daß immer gewisse Sicherheitsanforderungen eingehalten werden.
Das ursprünglich "endlos" laufende Steuerprogramm der EOW wird in dem in VSE zur Verfügung stehenden Ansatz beschrieben, indem der Effekt eines einzelnen Schleifendurchlaufs als sequentielle Operation spezifiziert wird. Eine weitere Operation dient dazu, die Betriebsbereitschaft nach dem Einschalten herzustellen. Die Formalisierung der Steuerungssoftware hält sich weitestgehend an die in Kapitel 2 gegebene Beschreibung der EOW.

3.4 Sicherheitsmodell der EOW

Im Sicherheitsmodell werden die sicherheitsrelevanten Anforderungen an das Steuerprogramm der EOW festgelegt. Zum einen können Sicherheitsanforderungen häufig als Invarianten formuliert werden, d.h. als Eigenschaften, die in jedem erreichbaren Zustand gelten müssen.

S(0): Weichenlage und Motorantrieb müssen mit dem Ordnungsmelder in Übereinstimmung sein.

[2] An dieser Stelle bietet es sich an, den ersten Teil von Kapitel 5 als Hintergrundinformation und Motivation für die nachfolgenden Abschnitte durchzusehen.

- Übereinstimmung zwischen der Weichenlage und der Lageanzeige.
- Sichere Anzeige der überwachten Endlage durch Weichenlage- und Ordnungsmelder.

Zudem dürfen Änderungen an diesen Komponenten der EOW nur unter ganz bestimmten Bedingungen eintreten (Anforderungen an bestimmte einzelne Zustandsübergänge).

S(1): Einschränkung der Zulässigkeit eines Stellbefehls von der vorgezogenen Bedienstelle. Stellaufträge von dieser Bedienstelle sind nur zulässig, wenn

- der Antrieb sich in Ordnungsstellung befindet und
- die zugehörige Gleisfreimeldung keinen Besetztzustand zeigt.

S(2): Einschränkung der Zulässigkeit eines Stellbefehls von einem der beiden Schienenkontakte. Stellaufträge von den Schienenkontakten dürfen nur ausgeführt werden,wenn

- der Gleisbereich nicht besetzt gemeldet ist
- der Antrieb sich im ungestörten Ruhezustand befindet
- bei Umlaufanforderung aus dem rechten Zweig überwachte Linkslage vorliegt (und umgekehrt).

S(3): In sicherheitskritischen Situationen darf der Motor nur mit der Schlüsselbedienstelle in Gang gesetzt werden. Sicherheitskritische Situationen sind:

- Gleisfreimeldung zeigt Besetztzustand an
- Weichenzunge liegt nicht in einer Endlage (und darin enthalten: Umlaufstörung bzw. Weiche aufgefahren)
- Motor läuft gerade und soll seine Laufrichtung ändern (reversieren).

3.4.1 Zustandsobjekte und ihre Wertebereiche

Anhand obiger Anforderungen können bereits die wichtigsten Zustandskomponenten und ihre Wertebereiche festgelegt werden: die Komponenten *Motor* und *Weichenlage- und Ordnungsmelder* werden angesteuert. Die Steuerungssoftware hält stets intern ein Abbild der *Weichenlage*; es repräsentiert die interne Annahme der Steuerungssoftware über die aktuelle Lage der Weiche, ist für die Steuerung sicherheitsrelevant und wird ständig mit den von außen gelieferten Sensordaten abgeglichen.
Nachdem die sicherheitsrelevanten Komponenten identifiziert sind, können die zugeordneten Wertebereiche festgelegt werden; dies ist hier beispielhaft für den Motorantrieb beschrieben. Der Motor kann sich im `linkslauf`, im `rechtslauf` oder im `stillstand` befinden. Der Wertebereich wird in VSE-SL als abstrakter Datentyp (in einer Spezifikation der Klasse THEORY) definiert:

```
THEORY motorstat
 TYPES : motorstatus =
         FREELY GENERATED BY linkslauf |
                             rechtslauf |
                             stillstand
THEORYEND
```

Das Zustandsobjekt zur Modellierung der Motoransteuerung (`motor`) wird in VSE-SL in der folgenden syntaktischen Form deklariert:

```
OBJECT sm_eow
 ...
 USING: motorstat
 ...
 DATA : motor : motorstatus;
 ...
OBJECTEND
```

Die Zustandsobjekte werden bei der Spezifikation eines Zustandsübergangssystems (Spezifikation der Klasse OBJECT) in der DATA-Klausel festgelegt.
Entsprechend werden auch die Wertebereiche für die anderen Zustandskomponenten sowie die Zustandsobjekte selber spezifiziert. Der Weichenlage- und Ordnungsmelder kann `links`, `rechts` oder `blinkend` anzeigen; die Weichenlage kann sich in den Zuständen `linkslage`, `rechtslage` oder `keine_endlage` befinden.

```
DATA : ...
       anzeige : anzeigestatus;
       weichenlage : weichenlagestatus;
```

Abbildung 3.2 zeigt grafisch die Struktur und Abhängigkeiten zwischen den Spezifikationstexten für das Sicherheitsmodell.

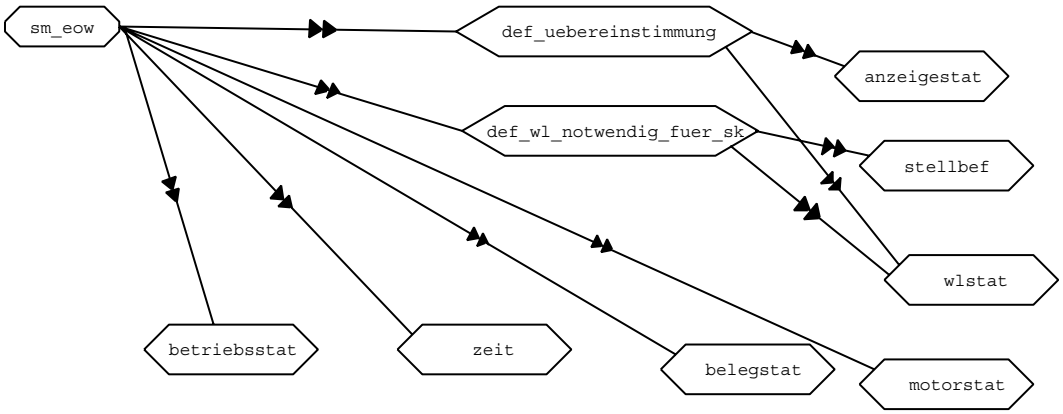


Abbildung 3.2: Struktur des Sicherheitsmodells

3.4.2 Operationen

Die Werte von Zustandsobjekten können sich bei Zustandsübergängen verändern. Diese Übergänge werden durch sogenannte Operationen bewirkt, die aufgrund von Stellbefehl, Weichenlage und den Belegungszustand, die als Sensordaten vorliegen, eine Zustandsänderung vornehmen. Die Operationen sind in dieser Fallstudie so modelliert, daß Sensordaten von einem Rahmensystem über Parameter geliefert werden. Sie werden spezifiziert, indem die Modifikation der Zustandsobjekte beschrieben wird.
Die Sicherheit der EOW wird in diesem Dokument durch eine Invariante (Anforderung an alle Zustände) sowie durch spezifische Anforderungen an bestimmte Operationen formuliert. (Sicherheits-)Eigenschaften, die von allen Operationen und somit in jedem erreichbaren Zustand erfüllt sein müssen, können als Invarianten spezifiziert werden; siehe Abschnitt 3.4.3
Die zentrale Operation der EOW, die auf sicherheitsrelevante Zustandsobjekte modifizierend zugreift, ist eine Operation, die immer wieder zyklisch aufgerufen wird. Diese Operation heißt in der Spezifikation `schalten`. Sicherheitsanforderungen, die speziell die durch

Operation `schalten` bewirkten Zustandsübergänge betreffen, sind unter **S(1)** bis **S(3)** beschrieben.
Direkt nach dem Einschalten des Systems soll ein Abgleich zwischen den internen Daten der Steuerungssoftware und den externen Sensordaten durchgeführt werden. VSE-SL bietet zwar die Möglichkeit, den Anfangszustand eines Zustandsübergangssystems formal zu spezifizieren. Allerdings ist es nicht möglich, dabei Informationen von außen als Parameter einfließen zu lassen. Daher wurde behelfsweise eine weitere Operation eingeführt, die direkt nach dem Einschalten des Systems aufgerufen werden soll und die diesen Abgleich durchführt; diese Operation heißt hier `abgleich`.
Die Operationen werden in zustandsbasierten Spezifikationen in der Klausel OPERATIONS deklariert und spezifiziert.

```
OPERATIONS :
PROC abgleich(wl : weichenlagestatus)
  ...
PROC schalten(quelle : stellbefehl,
              wl : weichenlagestatus,
              belegung : belegungsstatus,
              ...)
  ...
```

Die Wertebereiche der Parameter sind als abstrakte Datentypen definiert. Der Typ `stellbefehl` enthält die Werte `vbs`, `skl`, `skr`, `sbs`, `kein_auftrag` und modelliert somit Stellbefehle von den Bedienstellen/Schienenkontakten bzw. modelliert den Fall, daß im aktuellen Aufruf von `schalten` kein Stellbefehl anliegt.
Die Weichenlage soll einmal zu Beginn über `abgleich` aktualisiert werden. Danach soll nur noch `schalten` aufgerufen werden. Dies wird durch ein weiteres Zustandsobjekt modelliert, das dazu dient, einen bestimmten Kontrollfluß zu definieren. Das System kann sich somit in einem von zwei Betriebsmodi befinden, Einschaltzustand oder Regulärbetrieb.

```
PROC abgleich(wl : weichenlagestatus)
     REQUIRES modus = einschalt_zustand
     ENSURES modus = regulaer;
             weichenlage = wl;
             uebereinstimmung ( wl,anzeige )
```

Die Spezifikation besagt nun, daß `abgleich` nur im Einschaltzustand aufgerufen werden darf (REQUIRES) und daß nach ihrer Ausführung der Regulärbetrieb aktiviert ist (ENSURES `modus = regulaer`) und ein Abgleich mit externen Sensorwerten durchgeführt (`weichenlage = wl`) wurde und der Ordnungsmelder damit in Übereinstimmung steht (`uebereinstimmung( wl,anzeige )`). Die genaue Definition von `uebereinstimmung` ist in eine Spezifikation der Klasse THEORY ausgelagert. Hier wird deutlich, daß auch Spezifikationen selber stark strukturierte Beschreibungsmittel sind, die in einem Top-Down-Entwurf entwickelt werden können.
Die Operation `schalten` darf nur im Regulärbetrieb aufgerufen werden und beläßt das System im Regulärbetrieb:

```
PROC schalten( ... )
     REQUIRES modus = regulaer
     ENSURES modus = regulaer;
```

3.4.3 Invariante und Anfangszustand

Unter **S(0)** ist eine Anforderung beschrieben, die in jedem erreichbaren Zustand gelten muß. Solche Anforderungen können in VSE-SL in einer INVARIANTS-Klausel spezifiziert werden.

Die Umsetzung von **S(0)** in eine mathematische Formel[3] lautet in VSE-SL somit

```
INVARIANTS : (weichenlage = linkslage ->
              anzeige /= rechts) AND
             (weichenlage = rechtslage ->
              anzeige /= links) AND
             (motor /= stillstand OR
              weichenlage = keine_endlage ->
              anzeige = blinkend)
```

Diese Anforderung muß auch direkt nach dem Einschalten und vor Ausführung der (behelfsweise eingeführten) Operation `abgleich` erfüllt sein. Zu diesem Zeitpunkt ist allerdings der Steuerung nicht bekannt, welche Weichenlage in der Realität vorliegt. Um ein möglichst hohes Maß an Sicherheit zu gewinnen, wird zunächst dafür gesorgt, daß der Motor sich im Stillstand befindet und der Weichenlage- und Ordnungsmelder auf blinkend gestellt und somit der Weichenabschnitt gesperrt ist.

```
INITIAL : anzeige = blinkend;
          motor = stillstand;
          modus = einschalt_zustand
```

Zudem wird durch `modus = einschalt_zustand` festgelegt, daß als erste Operation `abgleich` aufgerufen werden muß.

3.4.4 Sicherheitsanforderungen an einzelne Zustandsübergänge

Anforderungen **S(1)** - **S(3)** beziehen sich auf die Operation `schalten`. Die Umsetzung dieser Anforderungen wird hier nur anhand von **S(3)** erläutert. Die übrigen können analog dazu behandelt werden.
S(3) besagt: In sicherheitskritischen Situationen darf der Motor nur mit der Schlüsselbedienstelle in Gang gesetzt werden. Sicherheitskritische Situationen sind:

- Gleisfreimeldung zeigt Besetztzustand an
- Weichenzunge liegt nicht in einer Endlage (und darin enthalten: Umlaufstörung bzw. Weiche aufgefahren)
- Motor läuft gerade und soll seine Laufrichtung ändern (reversieren)

Um diese Aussage umsetzen zu können, wird sie in ihre Bestandteile zerlegt:

(1) "Motor wird in Gang gesetzt" lautet formal[4]

```
motor /= motor' AND motor /= stillstand
```

(2) "Gleisfreimeldung zeigt Besetztzustand an" wird zu

```
belegung = besetzt
```

(3) "Keine Endlage erreicht" wird ausgedrückt durch

```
wl = keine_endlage
```

[3] In ASCII-Spezifikationstexten wird $\neq$ durch `/=` dargestellt
[4] In der ENSURES-Klausel gelten folgende Konventionen:

`motor'` bezieht sich auf den Wert des Zustandsobjektes `motor` unmittelbar vor Aufruf der Operation `schalten`;
`motor` (ohne Apostroph) dagegen bezieht sich auf den Wert unmittelbar nach Ausführung von `schalten`.

(4) "Motor läuft gerade und soll seine Laufrichtung ändern (reversieren)" wird zu

```
motor' /= stillstand AND motor /= motor' AND motor /= stillstand
```

(5) "Mit Schlüsselbedienstelle" entspricht

```
quelle = sbs
```

Diese Formeln lassen sich so zusammenstellen, daß sich daraus die Aussage **S(3)** ergibt; schematisch entspricht **S(3)** der Aussage:

> Wenn nach Ausführung von `schalten` (1) gilt und dabei (2), (3) oder (4) als Randbedingungen auftreten, dann kann das nur aufgrund (5) erfolgt sein.

In einem weiteren Schritt kann man dies nun als mathematische Formel aufschreiben:

(1) AND ((2) OR (3) OR (4)) $\rightarrow$ (5)

Weiter vereinfacht (`motor /= motor' AND motor /= stillstand` in (4) tritt auch in (1) auf und kann daher in (4) weggelassen werden) ergibt die Umsetzung nach VSE-SL folgende Spezifikation:

```
PROC schalten(quelle : stellbefehl,
              wl : weichenlagestatus,
              belegung : belegungsstatus, ...)
     REQUIRES ...
     ENSURES ...
             motor /= motor' AND motor /= stillstand AND
             (belegung = besetzt OR
              wl = keine_endlage OR
              motor' /= stillstand) ->
              quelle = sbs
```

Das komplette Listing aller Spezifikationen ist in Anhang A abgedruckt.

3.5 Leistungsspezifikation

In der Leistungsspezifikation wird die Funktionalität der Weichensteuerung mit dem Ziel beschrieben, die Zustandsübergänge vollständig anzugeben. In diesem Fall bietet es sich an, ein Zustandsübergangsdiagramm aufzustellen und ausgehend davon die formale Spezifikation weiterzuentwickeln. Zu diesem Zweck müssen die möglichen Zustände sowie die mit ihnen assoziierten Übergänge festgelegt werden.

3.5.1 Zustandsobjekte und ihre Wertebereiche

Zunächst müssen in der Leistungsspezifikation alle Zustandsobjekte auftreten, die auch in dem Sicherheitsmodell enthalten sind: `motor`, `anzeige`, `weichenlage`.
Daneben enthält die Leistungsspezifikation weitere Zustandsobjekte, die keine sicherheitsrelevante Bedeutung haben, die jedoch notwendig sind, um ein bestimmtes operationelles Verhalten der EOW-Steuerung zu beschreiben.
Wenn bei einem Weichenumlauf keine Endlage erreicht wurde, beispielsweise aufgrund eines zwischen Schiene und Weichenzunge eingeklemmten Schottersteins, soll es möglich sein, mit der Schlüsselbedienstelle die Weiche wieder in die vorangegangene Endlage umlaufen zu lassen. Dazu ist es notwendig, die vorangegangene Laufrichtung festzuhalten (Zustandsobjekt `letzterlauf` mit Wertebereich `motorstatus`).
Eine weitere Anforderung besagt, daß der Motor abgeschaltet werden soll, wenn nicht nach Ablauf einer bestimmten Laufzeit eine Endlage erreicht ist. Diese Anforderung ist nicht

sicherheitskritisch, ist jedoch notwendig, um den Weichenantrieb vor einem Motorbrand zu schützen, und stellt somit eine Anforderung dar, die die Betriebsbereitschaft der Weiche aufrechterhalten soll. Ein weiteres Zustandsobjekt (`zeit`) dient hierbei dazu, den Zeitpunkt festzuhalten, zu dem der Weichenantrieb gestartet wird. Der Wertebereich für dieses Zustandsobjekt (Zeitpunkte) wird nicht weiter spezifiziert.

3.5.2 Operationen

Sämtliche Operationen, die in dem Sicherheitsmodell vorkommen, müssen auch in der Leistungsspezifikation auftreten. Somit enthält die Leistungsspezifikation auf jeden Fall die Operationen `abgleich` und `schalten`. Um die Operationalität der Weichensteuerung zu beschreiben sind keine weiteren Operationen notwendig.

Anmerkung: Die Leistungsspezifikation könnte auch weitere Operationen enthalten, die jedoch keine der sicherheitsrelevanten Zustandsobjekte verändern, somit also auf diese höchstens lesend zugreifen dürften.

3.5.3 Anfangszustand

Der Anfangszustand muß laut Leistungsspezifikation der EOW die Anforderung an den Anfangszustand gemäß dem Sicherheitsmodell erfüllen. Weitere operationelle Anforderungen an den Anfangszustand gibt es nicht. Daher ist die INITIAL-Klausel der Leistungsspezifikation identisch mit der des Sicherheitsmodells.

3.5.4 Operationelle Anforderungen an abgleich und schalten

Die operationellen Anforderungen an `abgleich` und `schalten` sind aus dem Lastenheft nicht ohne weiteres direkt ersichtlich. Daher ist in diesem Fall eine analoge Vorgehensweise zu der Enwicklung des Sicherheitsmodells – Extraktion der informellen Anforderungen und Umsetzung in eine formale Spezifikation – schwierig, aufwendig und fehlerträchtig. Problematisch ist insbesondere die Aufstellung einer vollständigen Beschreibung der Zustandsübergänge.
Daher wird hier ein anderer Weg eingeschlagen: als Zwischenergebnis wird ein Zustandsübergangsdiagramm aufgestellt, das als Ausgangspunkt für die weitere Formalisierung der EOW-Spezifikation dient. Ein vollständiges Zustandsübergangsdiagramm wird entwickelt, indem mit dem Anfangszustand beginnend Übergänge zu neuen Zuständen gesucht werden. Zeitaspekte werden später hinzugefügt und sind daher in diesem ersten Schritt nicht enthalten.
Die Zustände werden dabei durch Tupel dargestellt, die aus Zustandsobjekten bestehen. Unterschiedliche Übergänge von einem Zustand aus werden durch unterschiedliche Sensorwertkombinationen bewirkt. Die Sensorwerte finden sich in den Parametern der Operationen wieder.
Damit das Zustandsübergangsdiagramm eine übersichtliche Größe behält, tritt in dem Zustandstupel das Zustandsobjekt `modus` nicht explizit auf. Implizit ist es jedoch enthalten: Im Anfangszstand ist `modus` gerade mit dem Wert `einschalt_zustand` belegt. Nun ist nur die Operation `abgleich` ausführbar wobei der Parameter, der die aktuelle Weichenlage angibt, drei Werte annehmen kann. In jedem Fall wird jedoch `modus` in den Wert `regulaer` geschaltet und danach ist nur noch `schalten` aktiviert; `schalten` beläßt fortan das System im regulären Betriebsmodus.
Somit dient zur Aufstellung des Zustandsübergangsdiagramms ein Zustandstupel bestehend aus den Komponenten:

`anzeige`: Wert des Weichenlage- und Ordnungsmelders

`motor`: Zustand des Weichenantriebs

`weichenlage`: interne Repräsentierung der Weichenlage

`letzterlauf`: interner Speicher für die zuletzt angelegte Motorlaufrichtung

Das Zustandstupel hat folgende Gestalt:

(`anzeige`, `motor`, `weichenlage`, `letzterlauf`)

Die Werte der Komponenten des Zustandstupels sowie die im Zustandsübergangsdiagramm auftretenden Abkürzungen für die Werte sind in folgender Tabelle aufgelistet:

`anzeige`	∈ {`rechts, links, blinkend`	}	(r,l,b)
`letzterlauf, motor`	∈ {`rechtslauf, linkslauf, stillstand`	}	(r,l,s)
`weichenlage`	∈ {`rechts, links, keine_endlage`	}	(r,l,k)

Abgesehen von dem ersten Übergang, der durch `abgleich` bewirkt wird, bestehen die Sensorwertkombinationen für alle übrigen Zustandsübergänge entsprechend der Parameter von `schalten` aus drei Komponenten:

`quelle`: Stellbefehl bzw. kein neuer Auftrag

`wl`: Aktueller Wert von den Weichenlagemeldern

`belegung`: Gleisfreimeldung

Somit können die Sensorwertkombinationen mit Hilfe des folgenden Tupels modelliert werden:

(`quelle`, `wl`, `belegung`)

Diese bestimmen wie der Zustand weitergeschaltet wird.
Die Wertebereiche der Komponenten sind wie folgt definiert:

`quelle`	∈ {`vbs , sbs , skr , skl , kein_auftrag`	}
`wl`	∈ {`linkslage , rechtslage , keine_endlage`	}
`belegung`	∈ {`besetzt , frei`	}

Sensorwertkombinationen werden in dem Zustandsübergangsdiagramm durch Nummern dargestellt, die nach folgendem Schema einander zugeordnet sind:

1:	(`vbs,linkslage,frei`)	6:	(`vbs,linkslage,besetzt`)
2:	(`sbs,linkslage,frei`)	7:	(`sbs,linkslage,besetzt`)
3:	(`skr,linkslage,frei`)	8:	(`skr,linkslage,besetzt`)
4:	(`skl,linkslage,frei`)	9:	(`skl,linkslage,besetzt`)
5:	(`kein_auftrag,linkslage,frei`)	10:	(`kein_auftrag,linkslage, besetzt`)
11:	(`vbs,rechtslage,frei`)		
...			
21:	(`vbs,keine_endlage,frei`)		
...			

Bei der Entwicklung des Zustandsübergangsdiagramms fällt auf, daß gewisse Kombinationen von Werten für Zustandsobjekte und Sensorwerte nicht auftreten können bzw. nicht auftreten sollen. Dabei handelt es sich zum Teil um physikalische oder um betriebliche Gesichtspunkte; zum Teil handelt es sich um Anforderungen an das Rahmensystem, in das diese Spezifikation eingebunden werden soll. Die Randbedingungen sind:

R(1): Weichenlageänderung und Stellbefehl werden nicht gleichzeitig in einem Aufruf der Operation `schalten` behandelt. Diese Kombination ist sehr unwahrscheinlich, kann jedoch nicht vollständig ausgeschlossen werden. Allerdings würde durch die Bearbeitung dieser Kombination ein erheblicher Mehraufwand entstehen, der durch Sequentialisierung der beiden Ereignisse adäquat behandelt werden kann. Daher muß das Rahmensystem in geeigneter Weise, z.B. wie in Kapitel 5 beschrieben, dafür sorgen, daß diese Kombination nicht an die Operation `schalten` weitergegeben wird.

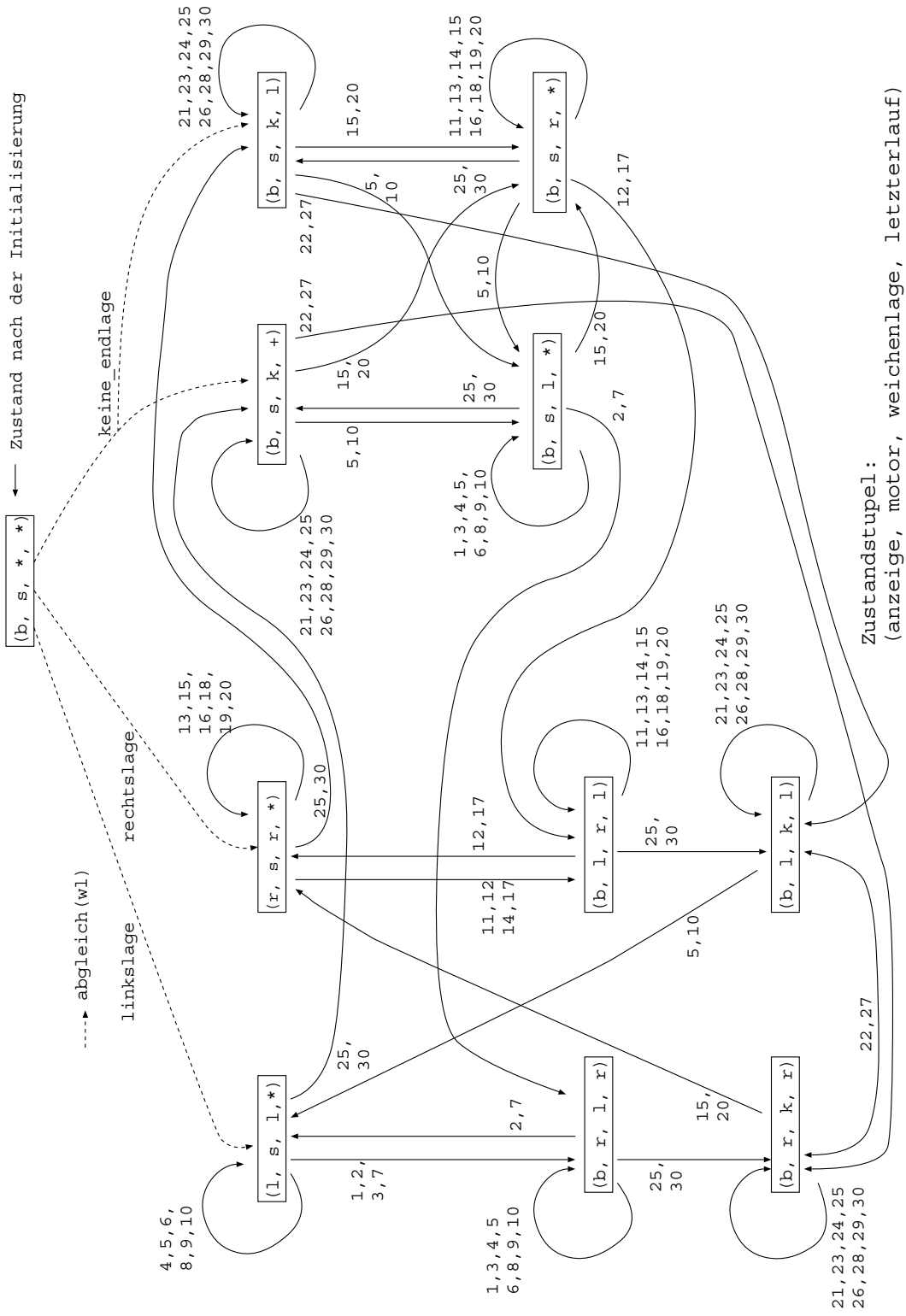


Abbildung 3.3: Zustandsübergangsdiagramm

R(2): Die Weichenlage kann nicht von einer Endlage in die andere Endlage übergehen, ohne daß eine "Zwischenlage" eingenommen wird (physikalisch unmöglich).

R(3): Wenn der Motor nach rechts (links) läuft, kann die Weiche nicht in Linkslage (Rechtslage) ankommen. Hier wird angenommen, daß die Hardware (Weichenantrieb, Mechanik, Endlagemelder, etc.) entsprechend robust und zuverlässig ausgelegt sind.

R(4): Unter betrieblichen Gesichtspunkten wird folgender Fall ausgeschlossen: Der Motor läuft an (aufgrund eines Stellbefehls), die Weiche hat die Endlage noch nicht verlassen und ein neuer Stellbefehl vom Schlüsselschalter liegt an.

Das vollständige Zustandsübergangsdiagramm ist in Abb. 3.3 dargestellt. Ein "*" in dem Tupel bedeutet, daß der entsprechende Wert keine Rolle spielt. Das Zeichen "+" in einem der Tupel faßt die Werte `stillstand` und `rechtslauf` zusammen. Dieses komplexe Diagramm dient nun als Grundlage für die weitere Formalisierung der EOW-Steuerung. Dabei werden Übergänge vereinfacht und gemeinsam behandelt, sofern dies aus dem Diagramm direkt zu ersehen ist. Um die Umsetzung übersichtlich zu halten, werden die Übergänge getrennt für jedes Zustandsobjekt spezifiziert.
Die Spezifikation von `abgleich` ist direkt aus diesem Diagramm abzulesen; die oben aufgestellten Rahmenbedingungen wirken sich auf `abgleich` nicht aus. Die einzige Anforderung an `abgleich` ist die aus dem Sicherheitsmodell übernommene Anforderung an den Betriebsmodus.

```
PROC abgleich(wl : weichenlagestatus)
  MODIFIES anzeige, weichenlage, modus
     REQUIRES modus = einschalt_zustand
     ENSURES  modus = regulaer;
              weichenlage = wl;
              IF wl = linkslage
              THEN anzeige = links
              ELSE IF wl = rechtslage
                   THEN anzeige = rechts
                   ELSE anzeige = blinkend
                   FI
              FI
```

Die Spezifikation von `schalten` ist komplizierter. Eine Vorbedingung ist die aus dem Sicherheitsmodell übernommene Anforderung an den Bertriebszustand. Daneben gehen in die Spezifikation der Operation `schalten` die oben aufgestellten Rahmenbedingungen als Vorbedingung ein.

```
PROC schalten( quelle   : stellbefehl,
               wl       : weichenlagestatus,
               belegung : belegungsstatus,
               clk : uhr)
     REQUIRES
     Betriebszustand[5]: modus = regulaer;

       /* R(1) */  NOT (weichenlage /= wl AND quelle /= kein_auftrag);


       /* R(2) */  NOT (wl /= weichenlage AND wl /= keine_endlage AND
```

[5] In VSE-SL ist es möglich, Formeln mit einem Namen zu versehen, indem ein Bezeichner der Formel vorangestellt wird.

```
                weichenlage /= keine_endlage);

      /* R(3) */  NOT (weichenlage = keine_endlage AND
                  motor /= stillstand AND
                  wl    /= keine_endlage AND
                  wl    /= erwartete_endlage ( motor ));

      /* R(4) */  NOT (motor /= stillstand AND
                  erwartete_endlage ( motor ) /= wl AND
                  wl    /= keine_endlage AND
                  quelle = sbs)
```

Der Betriebszustand, in dem Zustandsübergangsdiagramm implizit enthalten, wird von `schalten` nicht verändert.

```
PROC schalten( ... )
     ...
     ENSURES   modus = modus';
```

Aus dem Diagramm ist zu ersehen, daß nach jedem Aufruf der Operation `schalten` der Wert von `weichenlage` mit dem Parameterwert `wl` übereinstimmen muß:

```
     ENSURES   ...
       weichenlage = wl;
```

Jedesmal, wenn der Motor um- oder eingeschaltet wird, wird `letzterlauf` aktualisiert; ansonsten wird der Wert beibehalten.

```
     ENSURES   ...
       IF motor /= stillstand
       THEN letzterlauf = motor
       ELSE letzterlauf = letzterlauf'
       FI
```

Das Weiterschalten von `anzeige` und `motor` nimmt einen breiteren Raum ein. Daher werden die konkreten Details ausgelagert und die Veränderung der zugehörigen Werte wird über separat spezifizierte Funktionen ausgedrückt.

```
     ENSURES   ...
               anzeige = anzeige_next( ... )
               motor = motor_next( ... )
```

Die Funktionen `anzeige_next` und `motor_next` werden in Spezifikationen der Klasse THEORY beschrieben, die in der Spezifikation `eow` sichtbar sind:

```
OBJECT eow
 ...
 USING : ...
         def_anzeige_next;
         def_motor_next
 ...
OBJECTEND
```

Die Anzeige wird über die Funktion `anzeige_next` weitergeschaltet. Man kann beobachten, daß für die Mehrheit der möglichen Kombinationen von Zustandswerten und Sensorwerten ein neuer Zustand entsteht, in dem der Weichenlage- und Ordnungsmelder blinkt.
Entsprechend ist `anzeige_next` so spezifiziert, daß zunächst überprüft wird, ob eine Endlage angezeigt werden darf. In Abhängigkeit davon werden die konkreten Signale berechnet:

```
THEORY def_anzeige_next
 ...
 VARS : as : anzeigestatus; /* bisheriger Wert */
        ms : motorstatus;
        sb : stellbefehl;
        ws_alt, ws_aktuell : weichenlagestatus;
        bs : belegungsstatus
 ALGORITHMS : ...
        DEFFUNC anzeige_next(as, ms, ws_alt, sb, ws_aktuell, bs) =
          IF anf_fuer_endlageanzeige(as,ms,ws_alt,sb,ws_aktuell,bs)
          THEN IF ws_aktuell = rechtslage
               THEN rechts
               ELSE links
               FI
          ELSE blinkend
          FI
THEORYEND
```

Das Prädikat `anf_fuer_endlageanzeige`, das überprüft, ob die Anforderungen für die Anzeige einer Endlage vorliegen, ist ebenfalls in `def_anzeige_next` spezifiziert:

```
THEORY def_anzeige_next
 ...
   DEFPRED anf_fuer_endlageanzeige(as,ms,ws_alt,sb,ws_aktuell,bs) <->
               IF ms = stillstand
               THEN /* Motor steht still und die Weiche befindet
                       sich und bleibt in einer Endlage.
                       Die weiteren Anforderungen ergeben sich aus
                       dem Zustandsübergangsdiagramm  */
                    ws_aktuell /= keine_endlage AND
                    ws_aktuell = ws_alt AND
                    as /= blinkend AND
                   (sb = kein_auftrag OR
                    sb /= sbs AND bs = besetzt OR
                    sb = skl AND ws_aktuell = linkslage OR
                    sb = skr AND ws_aktuell = rechtslage)

               ELSE /* Fall: Weiche war im Umlauf und
                       hat nun eine Endlage erreicht;
                       die anderen Anforderungen ergeben sich wieder
                       aus dem Zustandsübergangsdiagramm. */
                    ws_alt = keine_endlage AND
                    ws_aktuell /= keine_endlage AND
                    as = blinkend AND
                    sb = kein_auftrag
               FI;
 ...
THEORYEND
```

Die Spezifikation der Funktion `motor_next` ist etwas aufwendiger. Die Vorgehensweise ist jedoch analog zu der Spezifikation von `anzeige_next`, wobei hier beobachtet werden kann, daß die meisten Übergänge zu einem Stillstand des Motors führen. Daher wird in der Formalisierung von `motor_next` ein Prädikat verwendet, das überprüft, ob der Motor im nächsten Zustand laufen darf/soll; `motor_next` ist in Spezifikation `def_motor_next` definiert. Abbildung 3.4 zeigt grafisch die Struktur und Abhängigkeiten zwischen den Spezifikationstexten

für die Leistungsspezifikation. Das komplette Listing der Leistungsspezifikation ist in Anhang A abgedruckt.

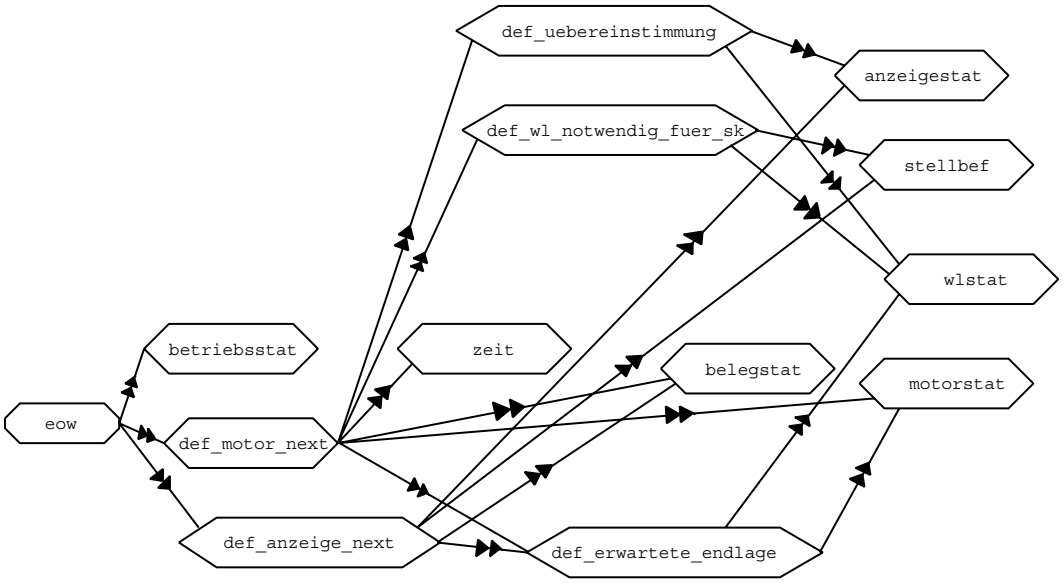


Abbildung 3.4: Struktur der Leistungsspezifikation

3.5.5 Behandlung von Zeitaspekten

Zeitaspekte lassen sich in einem gewissen eingeschränkten Rahmen ebenfalls modellieren. Dabei werden jedoch starke Annahmen an die Umgebung gemacht, die zudem nicht formal spezifiziert sind.
Eine Zeitanforderung besagt, daß der Motor nach Ablauf einer bestimmten Laufzeit abgeschaltet werden muß. Grundlage für die Modellierung dieser Anforderung ist, daß die aktuelle Zeit bei jedem Aufruf von `schalten` als Parameter mitgeliefert wird.
Beim Einschalten des Motors wird die aktuelle Zeit in einem internen Zustandsobjekt festgehalten.

```
PROC schalten( ..., clk: uhr)
     REQUIRES ...
     ENSURES ...
             einschaltzeit_merken : IF motor' = stillstand AND
                                       motor /= stillstand
                                    THEN start = clk
                                    ELSE start = start'
                                    FI;
     ...
```

Bei jeder Ausführung von `schalten` wird überprüft, ob die Motorlaufzeit noch innerhalb der zulässigen Schranken liegt. Ist diese Zeitbeschränkung überschritten, wird der Motor ausgeschaltet.

```
THEORY def_motor_next
 ...
 ALGORITHMS : ...
      DEFFUNC motor_next(..., clk_motor_start, clk_aktuell) =
              IF motor_timeout ( ... ,clk_motor_start, clk_aktuell)
              THEN stillstand
              ELSE /* Formalisierung des
                       Zustandsübergangsdiagramms */
              FI
THEORYEND
```

Die Funktion `motor_timeout` beruht auf dem Prädikat `timeout` aus THEORY `zeit`:

```
THEORY zeit
  TYPES : uhr
  PREDICATES : timeout : uhr,uhr
THEORYEND
```

Diese Theorie dient der Bereitstellung eines Typs für die Behandlung von Zeit sowie eines Prädikats `timeout`. Das Prädikat `timeout` erhält als Parameter eine Startzeit und eine Endzeit. Falls die Differenz zwischen der Startzeit und der Endzeit eine bestimmte Schranke, die Laufzeit, überschreitet, soll `timeout` *true* liefern und ansonsten *false*. Die Zeitschranke ist in dieser Modellierung an das Prädikat gebunden, ist damit eine Konstante, tritt aber nicht explizit in Erscheinung.
Sowohl der Typ als auch das Prädikat werden nicht weiter spezifiziert, und es bleibt als informelle Anforderung, daß die Implementierung dieser Theorie die reale Umwelt adäquat umsetzt. Davon hängt auch ab, ob Formeln, in denen `uhr` und `timeout` auftreten, die echte Welt möglichst realitätsgetreu beschreiben.
Zu einer adäquaten Modellierung von Zeit gehören jedoch noch zwei weitere informelle Anforderungen. Zum einen muß extern im Rahmensystem sichergestellt sein, daß bei jedem Aufruf von `schalten` die aktuelle Zeit übergeben wird. Zum anderen muß dafür gesorgt werden, daß die Operation mit einer gewissen Regelmäßigkeit aufgerufen wird. Diese Anforderungen sind für ein angemessenes "Echtzeitverhalten" wichtig, können jedoch in dem zugrundegelegten Formalismus nicht formal behandelt werden. Die formale Behandlung erstreckt sich daher lediglich auf den Aspekt der Abschaltung des Motors, wenn ein "Timeout" vorliegt.

Kapitel 4

Abstrakte Programmierung

Die Leistungsspezifikation ist der Ausgangspunkt für die Entwicklung der Steuerungssoftware. Dabei können einzelne Spezifikationen (OBJECT, THEORY) unabhängig voneinander durch abstrakte Programme (AP) verfeinert werden.
Die Vorgehensweise entspricht einem top-down Entwurf, wobei Spezifikationen schrittweise verfeinert werden. Abstrakte Programme verwenden Konzepte aus formalen (Import-) Spezifikationen, die ihrerseits in gleicher Weise wie die Exportspezifikation verfeinert werden können. Die Syntax abstrakter Programme ist an die Syntax konventioneller imperativer Programmiersprachen, wie z.B. Pascal angelehnt. In den abstrakten Programmen treten globale Programmvariablen auf. Operationen der zu implementierenden Spezifikation (Export) werden durch AP-Prozeduren realisiert, die ihrerseits lokale Variablen und Kontrollstrukturen enthalten.
Daneben können diese Prozeduren auch Funktionen und Operationen aus den Importspezifikationen aufrufen, die nur abstrakt spezifiziert vorliegen; daher sprechen wir von "abstrakten Programmen". Siehe Abb. 4.1.

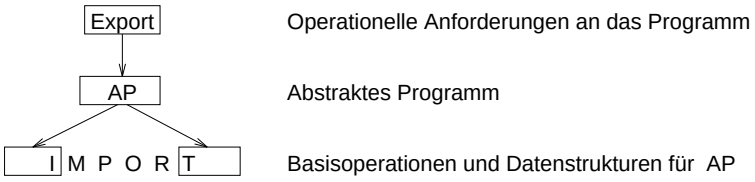


Abbildung 4.1: Modularisierungskonzept

In der Regel wird die Exportspezifikation selber gegliedert vorliegen. Eine empfohlene Vorgehensweise ist, die Verfeinerung der Top-Level-Spezifikation innerhalb dieser Gliederung in Termini der von ihr genutzten Spezifikationen zu formulieren; siehe Abb. 4.2.

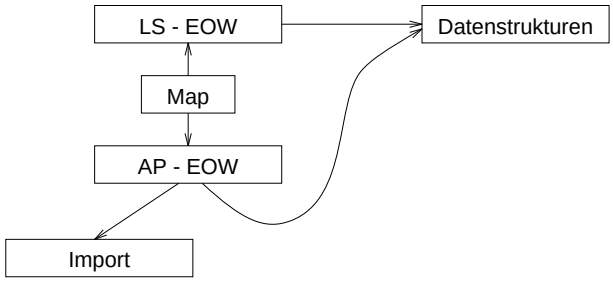


Abbildung 4.2: Verfeinerungsmethodik

Hierbei dienen diese Spezifikationen dem abstrakten Programm als Importspezifikation.

Wie die Exportspezifikation und das abstrakte Programm inhaltlich miteinander zusammenhängen, wird in einem Mapping angegeben, das zusammengehörende Bestandteile (Programmvariablen - Zustandsobjekte, Prozeduren - Operationen) aufeinander abbildet.
Das abstrakte Programm besteht in VSE-SL aus einem Rahmen (MODUL) in dem Programmvariablen und Prozeduren enthalten sind. Zudem enthält ein MODUL Referenzen auf die von ihm verwendeten Importspezifikationen.
Aufgrund der begrenzten Bearbeitungszeit wurde im Rahmen dieser Pilotstudie keine vollständige Verfeinerung angegeben, sondern nur ein Verfeinerungsschritt für einen begrenzten Ausschnitt (`OBJECT eow`) der Steuerungssoftware durchgeführt. Zudem ist die Verfeinerung inhaltlich ziemlich trivial; sie dient in erster Linie der Illustration der allgemeinen Vorgehensweise.
Die Spezifikation `eow` enthält Zustandsobjekte und Operationen auf diesen Zustandsobjekten. In einem VSE-Verfeinerungsschritt für `eow` müssen alle darin enthaltenen Zustandsobjekte und Operationen durch globale Programmvariablen bzw. Zustandsobjekte der Importschicht und Prozeduren verfeinert werden. Dabei können die Verfeinerungen weiterhin einen sehr abstrakten Charakter haben.

4.1 Verfeinerung der Zustandsobjekte

Zustandsobjekte können auf unterschiedliche Weise verfeinert werden. Verfeinerungen können zum einen in Termini globaler Programmvariablen vorgenommen werden, zum anderen können auch Zustandsobjekte der Importspezifikation diesem Zweck dienen. Eine weitere Möglichkeit ist die Verfeinerung eines einzigen Zustandsobjektes durch mehrere Zustandsobjekte/Programmvariablen der Verfeinerungsschicht. Rein formal stehen alle Möglichkeiten offen, allerdings ist es, wie unten beschieben, in manchen Fällen günstig, ein bestimmtes Verfahren auszuwählen.
In der Spezifikation `eow` sind folgende Zustandsobjekte enthalten:

```
DATA : motor       : motorstatus;
       letzterlauf : motorstatus;
       anzeige     : anzeigestatus;
       weichenlage : weichenlagestatus;
       modus       : betriebsstatus;
       start       : uhr
```

Hierbei dienen `motor` bzw. `anzeige` konzeptionell zur Ansteuerung des Weichenantriebs bzw. des Weichenlage- und Ordnungsmelders. Eine Verfeinerung dieser Zustandsobjekte durch Programmvariablen ist zwar theoretisch möglich, würde jedoch nicht der Realität entsprechen, da globale Programmvariablen keine direkte Auswirkung auf externe Hardware-Komponenten eines Systems haben. Zu reinen Modellierungszwecken ist eine Verfeinerung durch Programmvariablen durchaus geeignet, allerdings ist die Zielsetzung in dem hier verfolgten Ansatz die Verfeinerung durch abstrakte Programme, die weitgehend maschinell in ablauffähige Software umgesetzt werden können. Daher sollten `motor` und `anzeige` durch Zustandsobjekte implementiert werden, die in der Importschicht der Verfeinerung auftreten. Die Ansteuerung dieser Komponenten kann als separates Teilproblem in Importspezifikationen ausgelagert und durch konventionelle Programme implementiert werden. Beispielsweise modelliert folgende zustandsbasierte Spezifikation die Ansteuerung des Weichenlage- und Ordnungsmelders.

```
OBJECT anzeigesteuerung
 PURPOSE :
 " Softwarekomponente zur Ansteuerung der Anzeige."
 USING : anzeigestat
 DATA  : /* Weichenlage- und Ordnungsmelder */
```

```
        anzeige : anzeigestatus
 OPERATIONS :
 PROC wert : anzeigestatus
      ENSURES anzeige = RESULT
 PROC ansteuern(neuer_wert : anzeigestatus)
      MODIFIES anzeige
      ENSURES neuer_wert = anzeige
 INITIAL : anzeige = blinkend
OBJECTEND
```

Die Operation `wert` dient der Ermittlung des aktuellen Signals, das an der Anzeige anliegt; die Operation `ansteuern` legt das über den Parameter übergebene Signal an der Anzeige an.
Die Motorsteuerung kann ganz analog hierzu spezifiziert werden und ist in Anhang A abgedruckt.
Die übrigen oben aufgelisteten Zustandsobjekte dienen dazu, bestimmte Werte über Aufrufe von Operationen hinweg festzuhalten. Zudem handelt es sich bei diesen um relativ einfach strukturierte Daten, die nicht weiter verfeinert werden brauchen. Diese sollten daher direkt durch Programmvariablen verfeinert werden.
In der Verfeinerung (`MODULE i_eow`) drücken sich die bisher beschriebenen Konzepte in folgender Weise aus:

```
MODULE i_eow
 IMPORTSPEC : def_motor_next; def_anzeige_next; betriebsstat;
              anzeigesteuerung; motorsteuerung
 DATA : modus : betriebsstatus;
        weichenlage : weichenlagestatus;
        letzterlauf : motorstatus;
        start : uhr
 ...
MODULEEND
```

4.2 Zustandsoperationen und ihre Verfeinerung

Zustandsoperationen werden durch Prozeduren des abstrakten Programms verfeinert. Somit muß das abstrakte Programm Verfeinerungen für die Operationen `abgleich` und `schalten` bieten. Diese treten in `i_eow` auf: die Prozedur `i_abgleich`[1] verfeinert die Operation `abgleich` und `i_schalten` verfeinert `schalten`.
In zustandsbasierten Spezifikationen gibt es zudem eine Klausel, in der Bedingungen an den Anfangszustand spezifiziert sind und die von der Verfeinerung erfüllt werden müssen. Dazu muß eine (parameterfreie) Prozedur angegeben werden, die den Anfangszustand initialisiert. Diese Prozedur hat hier den (frei gewählten) Namen `i_init`. In VSE-SL werden die Namen der Prozeduren festgelegt durch:

```
MODULE i_eow
 ...
 ELEMENTS : i_abgleich, i_schalten
 MAIN : i_init
MODULEEND
```

Die zugehörigen Inhalte der Prozeduren werden textuell separat hiervon entwickelt.

[1] Die Namen der Prozeduren sind frei gewählt; man hätte auch die Namen der Operationen verwenden können.

4.2.1 Prozedur i_init

Die Prozedur `i_init` muß die Anforderung aus der INITIAL-Klausel der Spezifikation `eow` sicherstellen; diese ist nachfolgend noch einmal abgedruckt:

```
INITIAL : anzeige = blinkend;
          motor = stillstand;
          modus = einschalt_zustand
```

Ein Prozedur, die diese Anforderung erfüllt, könnte folgendermaßen aussehen:

```
PROCEDURE i_init
 PURPOSE : " Anfangszustand der Steuerung herstellen."
 BODY :
        ansteuern(blinkend);
        abschalten;
        modus := einschalt_zustand
PROCEDUREEND
```

Hierbei sind `ansteuern(...)` und `abschalten` Aufrufe von Operationen aus den Importspezifikationen `anzeigesteuerung` und `motorsteuerung`.
Allerdings sind diese beiden Anweisungen redundant, da beide Spezifikationen entsprechende INITIAL-Klauseln enthalten, in denen die Zustandsobjekte `anzeige` und `motor` bereits korrekt initialisiert vorliegen. Daher ist es ausreichend in der Prozedur `i_init` nur `modus` zu initialisieren:

```
PROCEDURE i_init
 PURPOSE : " Anfangszustand der Steuerung herstellen."
 BODY :
        modus := einschalt_zustand
PROCEDUREEND
```

4.2.2 Verfeinerung der Operationen abgleich und schalten

Die Prozedur `i_abgleich` soll die Anforderungen an die Operation `abgleich` erfüllen; Operation `abgleich` ist spezifiziert durch

```
 PROC abgleich(wl : weichenlagestatus)
      MODIFIES anzeige,weichenlage,modus
      REQUIRES modus = einschalt_zustand
      ENSURES modus = regulaer;
              weichenlage = wl;
              IF wl = linkslage
              THEN anzeige = links
              ELSE IF wl = rechtslage
                   THEN anzeige = rechts
                   ELSE anzeige = blinkend
                   FI
              FI
```

Die Formel nach REQUIRES ist nun eine Zusicherung, auf die in der Verfeinerung aufgebaut werden kann, sofern dies notwendig ist. Nach ENSURES stehen die Anforderungen an die Verfeinerung der Operation.
Eine Verfeinerung, die sehr stark an die Struktur der Spezifikation angelehnt ist, ist durch folgende Prozedur gegeben:

```
PROCEDURE i_abgleich
 PURPOSE :
 " In modus einschalt_zustand werden die internen Daten mit
  den externen Sensordaten abgeglichen."
 PARAMS : wl : IN weichenlagestatus
 BODY : modus := regulaer;
        weichenlage := wl;
        IF wl = linkslage
        THEN ansteuern(links)
        ELSE IF wl = keine_endlage
             THEN ansteuern(blinkend)
             ELSE ansteuern(rechts)
             FI
        FI
PROCEDUREEND
```

Die in der Spezifikation durch Gleichungen ausgedrückten Anforderungen sind in der Implementierung ersetzt durch Zuweisungen und Aufrufe von Operationen, die die entsprechenden Zustandsobjekte modifizieren. Nicht immer sind Verfeinerungen von Operationen derart eng an ihre Spezifikationen gebunden; siehe dazu auch die Prozedur `i_schalten` (siehe Anhang A), die die Operation `schalten` verfeinert. Der Abstand zwischen Spezifikation und abstraktem Programm kann auch wesentlich größer sein.

4.3 Mapping

Das Mapping `m_eow` stellt den formalen Zusammenhang zwischen der Leistungsspezifikation `eow` (EXPORTSPEC-Klausel) und dem abstrakten Programm `i_eow` (IMPLEMENTATION-Klausel) her.

```
MAPPING m_eow
 EXPORTSPEC : eow
 IMPLEMENTATION : i_eow
 MAPS : motorsteuerung.motor IMPLEMENTS eow.motor;
        anzeigesteuerung.anzeige IMPLEMENTS eow.anzeige;
        letzterlauf IMPLEMENTS letzterlauf;
        weichenlage IMPLEMENTS weichenlage;
        modus IMPLEMENTS modus;
        start IMPLEMENTS start;

        i_abgleich IMPLEMENTS abgleich;
        i_schalten IMPLEMENTS schalten
MAPPINGEND
```

Abbildung 4.3 zeigt den Ausschnitt aus dem Entwicklungsgraphen, der die Verfeinerung der Leistungsspezifikation `eow` graphisch darstellt. `eow` ist die Leistungsspezifikation und `m_eow` ist das Mapping. Das abstrakte Programm setzt sich aus `i_eow`, `i_init`, `i_abgleich` und `i_schalten` zusammen. `anzeigesteuerung` und `motorsteuerung` sind zusätzliche Importspezifikationen; die übrigen Textbausteine beschreiben die gemeinsam von `eow` und `i_eow` genutzten Datenstrukturen.
In der MAPS-Klausel wird explizit angegeben, welche konkreten Objekte zur Verfeinerung abstrakter Objekte herangezogen werden. In der obigen MAPS-Klausel könnte der erste Block (`motor` bis `start`) weggelassen werden, da die abstrakten Objekte und ihre Implementierungen den gleichen Namen tragen.

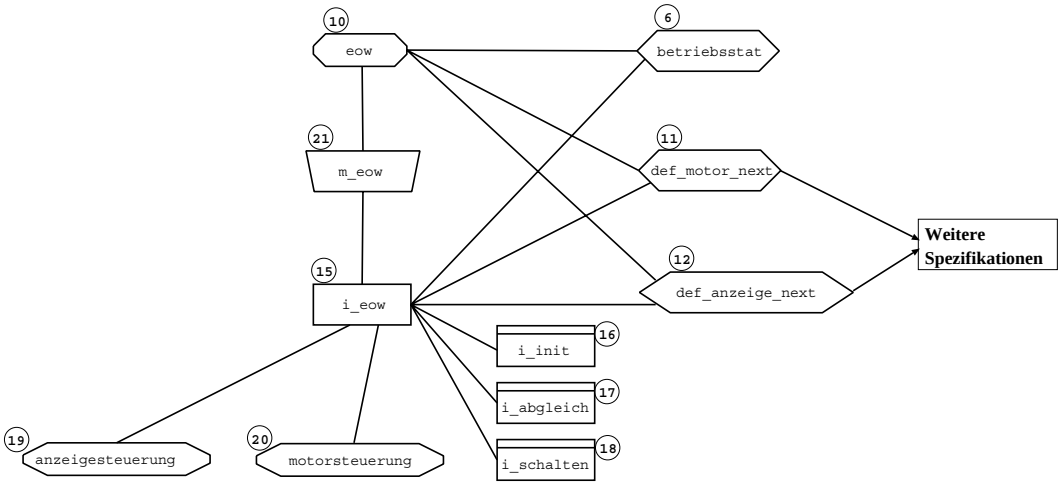


Abbildung 4.3: Verfeinerung der EOW

Die beiden Abbildungen (`abgleich`, `schalten`) am Ende müssen dagegen angegeben weden, da die abstrakt spezifizierten Operationen und ihre Implementierungen unterschiedlich benannt sind.

Kapitel 5

Konventionelle Anbindung

In VSE formal entwickelte Softwarekomponenten bestehen im wesentlichen aus einer Sammlung von Prozeduren, die in einen konventionell zu entwickelnden Rahmensystem eingebunden werden können (bzw. sollen).
Dieses Rahmensystem besteht zum einen aus einem "Kontrollprogramm", aus dem die formal entwickelten Prozeduren aufgerufen werden, und zum anderen um Implementierungen von sogenannten Basispezifikationen, die vom abstrakten Programm als Importspezifikation genutzt werden, selber aber nicht weiter formal verfeinert wurden.
Das "Kontrollprogramm" könnte beispielsweise aus einer Schleife bestehen, die ständig Sensorwerte abfragt und die entsprechenden formal entwickelten Prozeduren aufruft. Eine sehr einfache Kontrollschleife, die diesen Zweck erfüllt, illustriert das folgende Beispiel:

```
wl := weichenlage();        /* Weichenlage einlesen */
abgleich(wl);               /* Aufruf der formal entwickelten Prozedur */
LOOP
  sb := stellbefehl();      /* Stellbefehl einlesen */
  bm := besetztmeldung();   /* Besetztmeldung einlesen */
  cl := uhrzeit();          /* Aktuelle Uhrzeit einlesen */
  schalten(sb,wl,bm,cl)     /* Aufruf der formal entwickelten Prozedur */

  wl := weichenlage();      /* Weichenlage einlesen */
  bm := besetztmeldung();   /* Besetztmeldung einlesen */
  cl := uhrzeit();          /* Aktuelle Uhrzeit einlesen */
  schalten(sb,wl,bm,cl)     /* Aufruf der formal entwickelten Prozedur */
ENDLOOP
```

Die Schleife enthält zwei Blöcke; im ersten Block wird ein neuer Stellbefehl eingelesen, im zweiten wird die Weichenlage eingelesen. Dies dient dazu, Anforderung **R(1)** zu erfüllen.
In der Regel besteht die Basis der formalen SW-Entwicklung aus vielen jedoch sehr einfach aufgebauten Importspezifikationen. Handelt es sich bei diesen Importspezifikationen um in VSE-SL vordefinierte Datentypen, beispielsweise um eine Spezifikation der ganzen Zahlen (INTEGER), so kann das VSE-System dafür automatisch Programm-Code erzeugen (zur Zeit Ada), der mit einem Compiler weiter übersetzt werden kann. In allen anderen Fällen müssen diese Importspezifikationen "von Hand" konventionell implementiert werden.
Dies trifft z.B. auf die Spezifikationen `anzeigesteuerung` und `motorsteuerung` zu, die so implementiert werden müssen, daß dadurch tatsächlich externe Hardware-Komponenten angesteuert werden.
Formal entwickelte Softwarekomponenten werden somit sowohl nach unten (Basis-Importspezifikationen) als auch nach oben (häufig: zyklisch ablaufende Kontrollschleife) in ein konventionell entwickeltes Rahmensystem eingebunden.

Kapitel 6

Verifikation

Beweisverpflichtungen ergeben sich in dem bearbeiteten Ausschnitt aus zwei Aspekten heraus. Zum einen entstehen Beweisverpflichtungen aus der Beziehung zwischen Sicherheitsmodell und Leistungsspezifikation, d.h. die Leistungsspezifikation muß die Sicherheitsanforderungen erfüllen, zum anderen werden aus der Verfeinerung Beweisverpflichtungen generiert, die sicherstellen sollen, daß die Verfeinerung relativ zu der Leistungsspezifikation korrekt ist.

6.1 Sicherheitsanforderungen

Die Beziehung zwischen Leistungsspezifikation und Sicherheitsmodell wird in VSE-SL syntaktisch durch die Angabe einer SATISFIES-Klausel in der Leistungsspezifikation festgelegt. In dieser Fallstudie enthält die Spezifikation `sm_eow` das Sicherheitsmodell, und die Leistungsspezifikation ist durch `eow` gegeben:

```
OBJECT eow
 ...
 SATISFIES : sm_eow
OBJECTEND
```

Dies besagt, daß die Leistungsspezifikation `eow` die Anforderungen des Sicherheitsmodells `sm_eow` erfüllen muß. Dabei werden zusammengehörende Komponenten in `sm_eow` und `eow` über die Namensgleichheit identifiziert.
Falls unterschiedliche Namen verwendet würden, könnte man den Bezug zwischen Zustandsobjekten und Operationen des Sicherheitsmodells und der Leistungsspezifikation, in ähnlicher Form wie oben für die Verfeinerung beschrieben (Mapping) auch hier explizit angeben.
Aus der SATISFIES-Relation entstehen Beweisverpflichtungen; die formalen Grundlagen hierfür sind in der Sprachbeschreibung zu VSE-SL, [2], erläutert. Die Beweisverpflichtungen werden vom System automatisch erzeugt.
In der Regel wird man nicht ohne weiteres nachweisen können, daß diese Beweisverpflichtungen erfüllt sind, sondern wird als Hilfskonstrukt eine Beweisinvariante finden müssen, die zum einen selber beweisbar sein muß und die auf der anderen Seite dazu dient, den Nachweis zu führen, daß die übrigen Beweisverpflichtungen erfüllt sind.

Aus der Bearbeitung der Beweisverpflichtungen wurde folgende Beweisinvariante[1] als ausreichend festgestellt:

```
OBJECT eow
 ...
 INVARIANTS : anzeige = blinkend OR
              ((weichenlage = linkslage OR
                weichenlage = rechtslage) AND
               motor = stillstand);
              modus = einschalt_zustand -> motor = stillstand;
              motor /= stillstand -> letzterlauf = motor;
              motor /= stillstand -> weichenlage /= erwartete_endlage(motor)
 ...
OBJECTEND
```

Hier soll ein einfaches Beispiel den Charakter der Beweisverpflichtungen erläutern. Das Sicherheitsmodell enthält als Anforderung eine Invariante.

```
OBJECT sm_eow
 ...
 INVARIANTS : (weichenlage = linkslage ->
               anzeige /= rechts) AND
              (weichenlage = rechtslage ->
               anzeige /= links) AND
              (motor /= stillstand OR
               weichenlage = keine_endlage ->
               anzeige = blinkend)
OBJECTEND
```

Diese Invariante muß unter anderem auch in dem Anfangszustand der Leistungsspezifikation erfüllt sein. Der Anfangszustand der Leistungsspezifikation ist gegeben durch

```
OBJECT eow
 ...
 INITIAL : anzeige = blinkend;
           motor = stillstand;
           modus = einschalt_zustand
 ...
OBJECTEND
```

Aus der INVARIANTS- und der INITIAL-Klausel von `eow` (Prämissen) muß die INVARIANTS-Klausel von `sm_eow` (Konklusion) ableitbar sein. Schematisch wird diese Beweisverpflichtung dargestellt durch

$$\text{INITIAL(eow)}, \text{INVARIANTS(eow)} \vdash \text{INVARIANTS(sm\_eow)}$$

Die konkrete Beweisverpflichtung lautet somit

```
anzeige = blinkend,
motor = stillstand,
modus = einschalt_zustand,
anzeige = blinkend OR ((weichenlage = linkslage OR
weichenlage = rechtslage) AND motor = stillstand),
```

[1] Invarianten dienen zum einen der Spezifikation von Eigenschaften und zum anderen sind sie Hilsmittel in der Beweisführung. VSE trennt diese beiden Konzepte derzeit noch nicht vollständig voneinander; beide Konzepte werden in der INVARIANTS-Klausel ausgedrückt.

```
modus = einschalt_zustand -> motor = stillstand,
motor /= stillstand -> letzterlauf = motor,
motor /= stillstand -> weichenlage /= erwartete_endlage(motor)
⊢
(weichenlage = linkslage -> anzeige /= rechts) AND
(weichenlage = rechtslage -> anzeige /= links) AND
(motor /= stillstand OR weichenlage = keine_endlage -> anzeige = blinkend)
```

6.2 Korrektheitsnachweise

Korrektheitsnachweise stellen sicher, daß die in der abstrakten Programmiersprache formulierte Verfeinerung, die an sie in der (Export-)Spezifikation gestellten Anforderungen erfüllt. Der Bezug zwischen Exportspezifikation und Verfeinerung wird in VSE-SL syntaktisch über das Mapping beschrieben. Aus Mappings resultieren Beweisverpflichtungen für die Korrektheit. Diese werden von dem VSE-System ebenfalls automatisch erzeugt. Die formalen Grundlagen für die Beweisverpflichtungen sind in der Sprachbeschreibung erläutert. Hier gehen wir exemplarisch auf eine Beweisverpflichtung ein, die direkt an die oben (SATISFIES) beschriebene Beweisaufgabe anschließt.
Eine der erzeugten Beweisverpflichtungen besagt informell, daß nach Ausführung der Initialisierungsprozedur `i_init` die Invariante der Leistungsspezifikation erfüllt sein muß.

```
OBJECT eow
 ...
 INVARIANTS : anzeige = blinkend OR
              ((weichenlage = linkslage OR
                weichenlage = rechtslage) AND
               motor = stillstand);
              modus = einschalt_zustand -> motor = stillstand;
              motor /= stillstand -> letzterlauf = motor;
              motor /= stillstand -> weichenlage /= erwartete_endlage(motor)
 ...
OBJECTEND
```

Zudem beruht das abstrakte Programm auf den Importspezifikationen `anzeigesteuerung` und `motorsteuerung`, die ihrerseits bestimmte Zusicherungen an den Anfangszustand enthalten:

```
OBJECT anzeigesteuerung
 ...
 INITIAL : anzeige = blinkend
OBJECTEND

OBJECT motorsteuerung
 ...
 INITIAL : motor = stillstand
OBJECTEND
```

Diese Zusicherungen gehen als Prämissen in die generierte Beweisverpflichtung ein und man erhält als Aufgabe

$$\text{INITIAL(anzeigesteuerung)}, \text{INITIAL(motorsteuerung)} \vdash \langle \text{i\_init}() \rangle \text{ INVARIANTS(eow)}$$

Diese Beweisverpflichtung ist folgendermaßen zu lesen:

Wenn als Prämissen die Anfangsbedingungen der Importspezifikationen zugrundegelegt sind, dann terminiert die Prozedur `i_init` *und nach ihrer Ausführung gilt als Nachbedingung die Invariante der Leistungsspezifikation.*

Die explizite Formulierung dieser Beweisverpflichtung lautet somit:

```
anzeige = blinkend, motor = stillstand
⊢
<i_init()> anzeige = blinkend OR
    ((weichenlage = linkslage OR
    weichenlage = rechtslage) AND
    motor = stillstand),
    modus = einschalt_zustand -> motor = stillstand,
    motor /= stillstand -> letzterlauf = motor,
    motor /= stillstand -> weichenlage /= erwartete_endlage(motor)
```

Eine weitere Beweisverpflichtung, die als Nachbedingung statt der Invarianten die Anforderung aus der INITIAL-Klausel von `eow` enthält, besagt, daß von `i_init` die Initialisierungsbedingung aus der Leistungsspezifikation erfüllt werden muß.

6.3 Zusammenhang zwischen Sicherheitsmodell und Verfeinerung

Die in Abschnitt 6.2 beschriebenen Beweisaufgaben dienen dem Nachweis

Das abstrakte Programm ERFÜLLT die Leistungsspezifikation

Beweisverpflichtungen in Abschnitt 6.1 dienen dem Nachweis

Die Leistungsspezifikation ERFÜLLT das Sicherheitsmodell.

Beides zusammengenommen bietet in einem Kettenschluß die Aussage

Das abstrakte Programm ERFÜLLT das Sicherheitsmodell

Als Beispiel sei hier wieder der in den Abschnitten 6.1 und 6.2 betrachtete Aspekt herangezogen. `i_init` erfüllt sowohl die INITIAL- als auch die INVARIANTS-Klausel der Leistungsspezifikation; aus der INITIAL- und der INVARIANTS-Klausel der Leistungsspezifikation läßt sich die Invariante des Sicherheitsmodells ableiten. Somit erfüllt die Prozedur `i_init` auch die INVARIANTS-Klausel des Sicherheitsmodells.
Dieses Schema wird auch über mehrere Verfeinerungsschritte hinweg beibehalten, so daß auf diese Weise sichergestellt ist, daß das entwickelte Programm insgesammt die Sicherheitsanforderungen und die Leistungsspezifikation erfüllt.

6.4 Beweisführung und Fehlersuche

Die Durchführung von Beweisen dient zum einen dem Nachweis, daß alle formal aufgestellten Anforderungen tatsächlich erfüllt sind. Auf der anderen Seite werden in der Regel beim ersten Versuch nicht alle Beweise gelingen, da häufig Fehler auch in der formalen Spezifikation vorhanden sind.
Entsprechend ist auch hier im ersten Anlauf der Beweis verschiedener Beweisverpflichtungen nicht gelungen. Dies kann aus unterschiedlichen Gründen geschehen: zum einen aus Fehlern, die noch in der formalen Spezifikation enthalten sind und zum anderen spielen die Fähigkeiten der "beweisführenden" Person eine entscheidende Rolle. Interessant ist in diesem Zusammenhang, ob und in welchem Maß das System bei der Fehlersuche Unterstützung bietet. Dies ist nachfolgend anhand der bei der Beweisführung aufgetretenen Probleme dargestellt.

6.4.1 Beweisinvariante – Teil 1

Eine nicht-beweisbare Aussage ergab sich aus dem Beweisversuch von op-12 der Sicherheitsanforderungen. In der Invarianten von `eow` fehlte zunächst die Formel

```
motor /= stillstand -> weichenlage /= erwartete_endlage(motor)
```

Nach einer Abfolge von Beweisregeln, die nach der Korrektur zum Ziel führte, konnte das letzte Beweisziel nicht nachgewiesen werden. Der interaktive Aufruf von INKA mit diesem Beweisziel schlägt fehl und im Beweisbaum von INKA kann das folgende nicht-widerlegbare[2] Teilziel als ein Grund hierfür identifiziert werden:

```
(NOT Zeit.Timeout (_EOW.START', _EOW.SCHALTEN-CLK')
 AND def_motor_next.motor_next (anzeigestat.blinkend,
                                motorstat.linkslauf,
                                motorstat.stillstand,
                                wlstat.linkslage,
                                stellbef.skr,
                                wlstat.linkslage,
                                belegstat.frei,
                                _EOW.START',
                                _EOW.SCHALTEN-CLK')
     = motorstat.rechtslauf)
```

Ein Aufruf von `motor_next` mit obigen Parametern kann während der Laufzeit der Steuerungssoftware nicht eintreten. Insbesondere kann nicht eintreten, daß der Motor (im `linkslauf` – zweiter Parameter) weiterläuft, obwohl die Weiche bereits die entsprechende Endlage (`linkslage` – vierter Parameter) erreicht hat. Um den Beweis mit dem VSE-System führen zu können, muß die Beweisinvariante die Formel

```
motor /= stillstand -> weichenlage /= erwartete_endlage(motor)
```

enthalten.

6.4.2 Auffahrerkennung

Bei dem Versuch Beweisverpflichtung op-9 der Sicherheitseigenschaften nachzuweisen, konnte auch ein Fehler in der Modellierung der Sicherheitseigenschaften entdeckt werden. Der Beweis schlug fehl, obwohl in der Leistungsspezifikation hierfür kein Grund zu sehen war. Das Beweisziel von op-9 ist die Auffahrerkennung.

```
schalten(...)
  ...
  ENSURES
    NOT wl = weichenlage' AND
    motor = stillstand ->
    anzeige = blinkend;
    ...
```

Der Fehler konnte durch genaue Betrachtung der Formalisierung dieser Sicherheitsanforderung erkannt werden: In der Formel muß das Zustandsobjekt `motor` mit einem Apostroph versehen werden, denn die Weiche gilt als aufgefahren, wenn sich die Weichenlage ändert, obwohl *vor* Ausführung der Operation `schalten` der Motor stillsteht (`motor' = stillstand`). Zudem wurde bei dieser Gelegenheit die Anforderung insgesamt überdacht und strenger formuliert.

[2] Widerspruchsbeweis, Resolutionskalkül.

```
schalten(...)
  ...
  ENSURES
    NOT wl = weichenlage' AND
    motor' = stillstand ->
    anzeige = blinkend AND motor = stillstand;
    ...
```

Wenn die Weiche aufgefahren wird, soll das dazu führen, daß im Folgezustand neben dem Weichenlage- und Ordnungsmelder auch der Motor in den sicheren Zustand `stillstand` geschaltet wird.

6.4.3 Prädikat `anf_fuer_rechtslauf`

Das Prädikat `anf_fuer_rechtslauf` wird in der Spezifikation von `motor_next` verwendet und soll *true* liefern, wenn der Motor in Rechtslauf geschaltet werden darf. Die fehlerhafte Spezifikation lautete:

```
DEFPRED anf_fuer_rechtslauf(ms, ll, sb, ws_aktuell) <->
                (sb /= sbs AND ms = rechtslauf) OR
                (ws_aktuell = keine_endlage AND ll = linkslauf) OR
                ws_aktuell = linkslage;
```

Der ursprünglich enthaltene Fehler trat wieder in Erscheinung als versucht wurde, die Sicherheitsanforderung op-12 zu beweisen. Nach einer Abfolge von Beweisregeln, mit denen später der Beweis von op-12 gelang, konnte das letzte Beweisziel nicht nachgewiesen werden. INKA liefert als einen Grund das nicht-widerlegbare Teilziel

```
(NOT Zeit.Timeout (_EOW.START', _EOW.SCHALTEN-CLK')
 AND NOT wlstat.keine_endlage
         = def_erwartete_endlage.erwartete_endlage (def_motor_next.motor_next (
anzeigestat.blinkend,
motorstat.linkslauf,
motorstat.linkslauf,
wlstat.keine_endlage,
stellbef.kein_auftrag,
wlstat.keine_endlage,
belegstat.frei,
_EOW.START',
_EOW.SCHALTEN-CLK'))
 AND NOT def_motor_next.motor_next (anzeigestat.blinkend,
                                    motorstat.linkslauf,
                                    motorstat.linkslauf,
                                    wlstat.keine_endlage,
                                    stellbef.kein_auftrag,
                                    wlstat.keine_endlage,
                                    belegstat.frei,
                                    _EOW.START',
                                    _EOW.SCHALTEN-CLK')
         = motorstat.linkslauf
 AND NOT def_motor_next.motor_next (anzeigestat.blinkend,
                                    motorstat.linkslauf,
                                    motorstat.linkslauf,
                                    wlstat.keine_endlage,
                                    stellbef.kein_auftrag,
```

```
                                    wlstat.keine_endlage,
                                    belegstat.frei,
                                    _EOW.START',
                                    _EOW.SCHALTEN-CLK')
        = motorstat.stillstand)
```

Eigentlich sollte `motor_next` bei Aufruf mit obigen Parametern als Resultat `linkslauf` liefern. Dieses Teilziel kann nicht widerlegt werden, da mit der obigen (falschen) Definition von `anf_fuer_rechtslauf` gilt:

```
motor_next(...)  = rechtslauf
```

Die Teilformel `ws_aktuell = keine_endlage AND ll = linkslauf` in der Definition von `anf_fuer_rechtslauf` soll eigentlich das Reversieren der Laufrichtung von `linkslauf` in `rechtslauf` beschreiben. Allerdings fehlt hier noch die Einschränkung, daß das nur von der Schlüsselbedienstelle aus erfolgen kann, d.h. das Konjunkt `sb = sbs` fehlt. Die korrigierte Spezifikation lautet somit

```
DEFPRED anf_fuer_rechtslauf(ms, ll, sb, ws_aktuell) <->
            (sb /= sbs AND ms = rechtslauf) OR
            (sb = sbs AND ws_aktuell = keine_endlage AND ll = linkslauf) OR
            ws_aktuell = linkslage;
```

6.4.4 Beweisinvariante – Teil 2

Informell besagt die Beweisverpflichtung inv-1 aus den Korrektheitsanforderungen, daß Operation `abgleich` die Invariante erfüllen muß. Für den Korrektheitsbeweis von inv-1 war es allerdings notwendig, die Invariante der Leistungsspezifikation zum Zweck der Beweisführung zu erweitern. Dies kann aus einem offenen Beweisziel bei der Beweisführung mit KIV gefolgert werden[3]:

---

8) open
$\ldots$,
$motor = motor_0$,
$links = anzeige$,
$\neg$ $((anzeige = blinkend \vee (i\_weichenlage = linkslage \vee i\_weichenlage = rechtslage) \wedge motor = stillstand) \wedge \ldots)$
$\vdash$

Dieser Beweis gelingt, wenn $motor_0 = stillstand$ gilt. $motor_0$ bezeichnet den Wert des Motors vor Aufruf der Operation `abgleich`, d.h. wenn `modus = einschalt_zustand` ist. Daher wurde die Implikation

```
modus = einschalt_zustand -> motor = stillstand
```

in die Invariante der Leistungsspezifikation aufgenommen.

6.4.5 Beweisinvariante – Teil 3

Der Beweis der Korrektheitsanforderung op-2, die besagt, daß die Operation `schalten` die an sie gestellten funktionalen Anforderungen erfüllen muß, führt ähnlich dem letzten Beispiel zu einem offenen Beweisziel:

[3] Das abgebildete Beweisziel ist dem Beweisprotokoll entnommen.

106) open
$motor = motor\_next(anzeige', motor', letzterlauf, weichenlage', schalten-quelle,$
$schalten-wl, schalten-belegung, start, schalten-clk),$
$letzterlauf = letzterlauf',$
$motor = motor'$
$\vdash$
$motor \neq stillstand \rightarrow letzterlauf = motor$

Die Konklusion

```
motor /= stillstand -> letzterlauf = motor
```

ist gerade die Formel, die in die Invariante aufgenommen werden muß, damit der Beweis gelingt.

6.4.6 Prädikat `anf_fuer_endlageanzeige`

Beim Beweis der Korrektheitsanforderung inv-2 war in der ursprünglichen Spezifikation ein Beweisziel nicht erfüllbar. Insbesondere konnte mit INKA das folgende Teilziel nicht widerlegt werden:

```
((belegstat.besetzt = belegstat.frei OR belegstat.besetzt = belegstat.besetzt)
 AND NOT def_anzeige_next.anzeige_next (anzeigestat.links,
                                        motorstat.stillstand,
                                        wlstat.linkslage,
                                        stellbef.sbs,
                                        wlstat.linkslage,
                                        belegstat.besetzt)
         = anzeigestat.blinkend)
```

Da ein Stellbefehl von der Schlüsselbedienstelle anliegt und somit der Motor eingeschaltet wird, müßte die Funktion `anzeige_next(...)` als Resultat `blinkend` liefern. `anzeige_next` basiert auf dem Hilfsprädikat `anf_fuer_endlageanzeige`, das *true* liefern soll, wenn eine Endlage auf dem Weichenlage- und Ordnungsmelder angezeigt werden soll. Daher sollte also `anf_fuer_endlageanzeige(links,stillstand,linkslage,sbs,linkslage,besetzt)` *false* ergeben. `anf_fuer_endlageanzeige` war zunächst definiert durch

```
DEFPRED anf_fuer_endlageanzeige(as, ms, ws_alt, sb, ws_aktuell, bs) <->
      IF ms = stillstand
      THEN
           ws_aktuell /= keine_endlage AND ws_aktuell = ws_alt AND
           as /= blinkend AND
           (sb = kein_auftrag OR
            bs = besetzt OR
            sb = skl AND ws_aktuell = linkslage OR
            sb = skr AND ws_aktuell = rechtslage)
      ELSE
           ...
      FI;
```

Die in der Definition enthaltene Gleichung `bs = besetzt` beschreibt den Fall, daß auch weiterhin eine Endlage angezeigt werden soll, wenn der Schienenabschnitt besetzt ist, da nun kein Stellbefehl angenommen wird. Dies ist jedoch nur dann richtig, wenn der Stellbefehl nicht von der Schlüsselbedienstelle kommt (`sb/= sbs`). Die korrekte Definition des Hilfsprädikats lautet

```
DEFPRED anf_fuer_endlageanzeige(as, ms, ws_alt, sb, ws_aktuell, bs) <->
      IF ms = stillstand
      THEN
           ws_aktuell /= keine_endlage AND ws_aktuell = ws_alt AND
           as /= blinkend AND
           (sb = kein_auftrag OR
            sb /= sbs AND bs = besetzt OR
            sb = skl AND ws_aktuell = linkslage OR
            sb = skr AND ws_aktuell = rechtslage)
      ELSE
           ...
      FI;
```

6.4.7 "Korrekt" und doch nicht richtig

Eine Spezifikation kann auch Fehler enthalten, die nicht durch Beweisversuche aufgedeckt werden können. In der vorliegenden Fallstudie wurde Sicherheitsanforderung **S(0)** zunächst unvollständig in die formale Spezifikation umgesetzt:

```
INVARIANTS : motor /= stillstand OR
             weichenlage = keine_endlage ->
             anzeige = blinkend
```

Die Forderung, daß bei Vorliegen einer Endlage die andere Endlage nicht angezeigt werden darf, wurde übersehen. Die Bearbeitung der Beweisverpflichtungen offenbarte hierfür natürlich keinen Fehler; ein formaler Fehler lag schließlich nicht vor. Das eigentliche Problem betraf die Umsetzung der informellen Anforderungen in eine formale Spezifikation. Die korrekte Umsetzung lautet

```
INVARIANTS : (weichenlage = linkslage ->
              anzeige /= rechts) AND
             (weichenlage = rechtslage ->
              anzeige /= links) AND
             (motor /= stillstand OR
              weichenlage = keine_endlage ->
              anzeige = blinkend)
```

Nach Korrektur der Sicherheitsanforderung konnte nachgewiesen werden, daß die Leistungsspezifikation unverändert auch die neuen Sicherheitsanforderungen erfüllt. Dies ist in dieser Fallstudie nicht weiter erstaunlich, da die fehlende Sicherheitsanforderung indirekt auch eine maßgebliche Anforderung an die Funktionalität darstellt: Wenn sich die Weiche nicht in einer sicherheitskritischen Situation befindet, soll der Weichenlage- und Ordnungsmelder die gerade vorliegende Lage der Weichenzunge anzeigen.

Kapitel 7

Ergebnisse und Erfahrungen

Der sicherheitskritische Kern der Steuerungssoftware für eine elektrisch ortsbediente Weiche wurde formal spezifiziert. Hierbei wurde das Vorgehen zur Bestimmung des Modellierungsausschnitts dargestellt. Die Umsetzung informeller Sicherheitsanforderungen in Formeln wurde an einzelnen Beispielen im Detail beschrieben und die Entwicklung der Leistungsspezifikation erfolgte anhand einer semiformalen Zwischenlösung, dem Zustandsübergangsdiagramm. Das Zustandsübergangsdiagramm wurde "von Hand" in VSE-SL umgesetzt. Die Entwicklung der Leistungsspezifikation würde jedoch erheblich vereinfacht, wenn die Möglichkeit geboten wäre, Systeme direkt mittels solcher Zustandsübergangsdiagramme bzw. in tabellarischer Form zu spezifizieren.
Daneben wurde ein erster Verfeinerungsschritt angegeben, und die Einbindung formal entwickelter Programme in einen konventionell entwickelten Rahmen wurde aufgezeigt.
Die entwickelten abstrakten Programme sind hier sehr eng an die zugehörigen Spezifikationstexte angelehnt; der Abstand zwischen Spezifikation und abstraktem Programm kann auch weitaus größer sein. Dann muß jedoch mit einem höheren Aufwand bei der Verifikation gerechnet werden.
Aus der formalen Beziehung zwischen Leistungsspezifikation und Sicherheitsmodell sowie aus dem formalen Zusammenhang zwischen abstrakten Programmen und Leistungsspezifikation entstehen Beweisverpflichtungen. Anhand einfacher Beispiele wurde versucht, die Art der auftretenden Beweisverpflichtungen darzustellen.
Das Führen von Beweisen dient zwei Zielen. Neben dem Nachweis, daß die formalen Anforderungen erfüllt sind, ist die Bearbeitung von Beweisverpflichtungen hilfreich beim Auffinden von Fehlern. Es wurde gezeigt, wie fehlgeschlagene Beweise dazu verwendet werden können, Fehler zu entdecken und zu beseitigen.
Fehlgeschlagene Beweise resultieren jedoch nicht ausschließlich aus Fehlern in den zugrundegelegten Spezifikationen/abstrakten Programmen, sondern können auch in einer ungeschickten Beweisführung begründet sein. Zu einer erfolgreichen Beweisführung gehört unter anderem auch die Formulierung einer Beweisinvariante. Offene Beweisziele können auch beim Auffinden der Beweisinvariante hilfreich sein.
Allerdings verläuft die Beweisführung häufig nicht so glatt und zielgerichtet, wie es aus der vorangegangenen Beschreibung erscheinen mag. Es werden in Beweisen auch Pfade eingeschlagen, die nicht zum Ziel führen. Die Beschreibung in Kapitel 6 ist insofern aufbereitet, um die gebotene Unterstützung bei der Fehler- und Invariantenfindung hervorzuheben.
Die während der Beweisführung auftretenden Zwischenziele können sehr umfangreich sein[1]. Das in Abschnitt 6.4.4 offene Beweisziel ist entsprechend gekürzt, um den Blick auf die wesentlichen Bestandteile zu lenken. Das Aufspüren dieser Bestandteile ist zeitaufwendig und bedarf zudem guter Kenntnisse beim Umgang mit dem Beweiser.
Daneben werden nach jeder Korrektur, d.h. Veränderung an den Spezifikationen bzw. ab-

[1] Über ftp ist eine erweiterte Version [3] dieses Berichtes erhältlich, die Beweisprotokolle einschließt.

strakten Programmen, die bisher geführten Beweise ungültig und müssen erneut (zeitaufwendig) durchgeführt werden. Im Idealfall können die ungültigen Beweise nachgespielt werden. Mit den in VSE enthaltenen Beweisern dauert allein das Nachspielen der gelungenen Korrektheitsbeweise auf einer sehr leistungsfähigen Workstation (Sparc20) ca.45 Minuten — diese Zeit wurde manuell ermittelt und schließt die in diesem Fall noch immer benötigte Benutzerinteraktion ein.
Schlägt ein Beweisversuch fehl, ist auch nicht in jedem Fall klar ersichtlich, ob ein Fehler oder eine zu schwach formulierte Beweisinvariante der Grund für einen mißlungenen Beweis ist. Der in Abschnitt 6.4.1 beschriebene Beweisversuch führte anfänglich zu der Vermutung, daß ein Fehler in der Spezifikation der Funktion `motor_next` vorliegt. Zunächst wurde dieser vermeintliche Fehler in `motor_next` "korrigiert". Eine gegen Abschluß dieser Arbeiten parallel durchgeführte Modellierung und Überprüfung der EOW-Steuerungssoftware mittels Verfahren aus dem "model-checking"[2] — hierbei werden keine expliziten Invarianten benötigt — erbrachte, daß auch die "fehlerhafte" `motor_next`-Variante sämtliche Sicherheitsanforderungen erfüllt. Dies lieferte den Hinweis, in der VSE-Modellierung der EOW-Steuerung nach einer stärkeren Beweisinvariante zu suchen.
Bei der EOW-Steuerung handelt es sich im Wesentlichen um ein System mit endlichem Zustandsraum, der zudem noch relativ klein ist. Hier können model-checking-Verfahren ihre Stärken ausspielen. Auf der anderen Seite haben model-checker nur einen sehr stark eingegrenzten Einsatzbereich. Im Gegensatz dazu sind die Beweissysteme, die in VSE derzeit enthalten sind, für eine wesentlich größere Klasse von Problemstellungen geeignet. Daher erscheint es sinnvoll, unterschiedliche Beweisverfahren in einem System miteinander zu integrieren.
Man darf sich jedoch nicht der Täuschung hingeben, daß ein System, das formal entwickelt wurde und für das alle Sicherheitsbeweise und Korrektheitsbeweise durchgeführt wurden, auch sicher und korrekt im Sinne der informellen Anforderungen sei. In der vorliegenden Fallstudie wurde dies an der Sicherheitsanforderung **S(0)** deutlich. Den Hinweis auf diesen Fehler lieferte eine an den vorangegangenen Tätigkeiten zur Formalisierung nicht beteiligte Person, die die ihm vorgelegten informellen Sicherheitsanforderungen mit den formalen Spezifikationen verglich und dabei die Unvollständigkeit entdeckte.
Die Leistungsspezifikation war in der vorliegenden Fallstudie so gefaßt, daß davon auch die vervollständigte Sicherheitsanforderung erfüllt wurde, d.h. auch in dem ursprünglich spezifizierten System lag keine durch diese Unvollständigkeit bedingte Sicherheitslücke vor. Allerdings ist dies in der bearbeiteten Aufgabenstellung begründet und dürfte ein Ausnahmefall sein. Im allgemeinen ist davon auszugehen, daß unvollständige Sicherheitsanforderungen mit Sicherheitslücken im entwickelten System einhergehen können.
Da keine streng formalen Mittel bereitstehen, um zu prüfen, ob informelle Anforderungen korrekt umgesetzt wurden, ist es unbedingt notwendig, die aus einem solchen Schritt entstandenen formalen Spezifikationen mit anderen Mitteln zu validieren. Daher sollten auch Untersuchungen zur Integration konventioneller Validierungstechniken mit formalen Beweismethoden geführt werden.

[2]Verwendet wurde dazu das System SMV, siehe [7]

Anhang A

Quelltexte der Spezifikationen

Nachfolgend sind die Spezifikation der EOW und die entwickelten Verfeinerungen abgedruckt. Abbildung A.1 veranschaulicht anhand eines Entwicklungsgraphs die Zusammenhänge zwischen den Textbausteinen der formalen Spezifikation. Die Nummern im Graphen entsprechen der Nummerierung im nachfolgenden Listing und sind eine gute Hilfe beim schnellen Auffinden der zugehörigen Spezifikationstexte.

A.1 anzeigestat

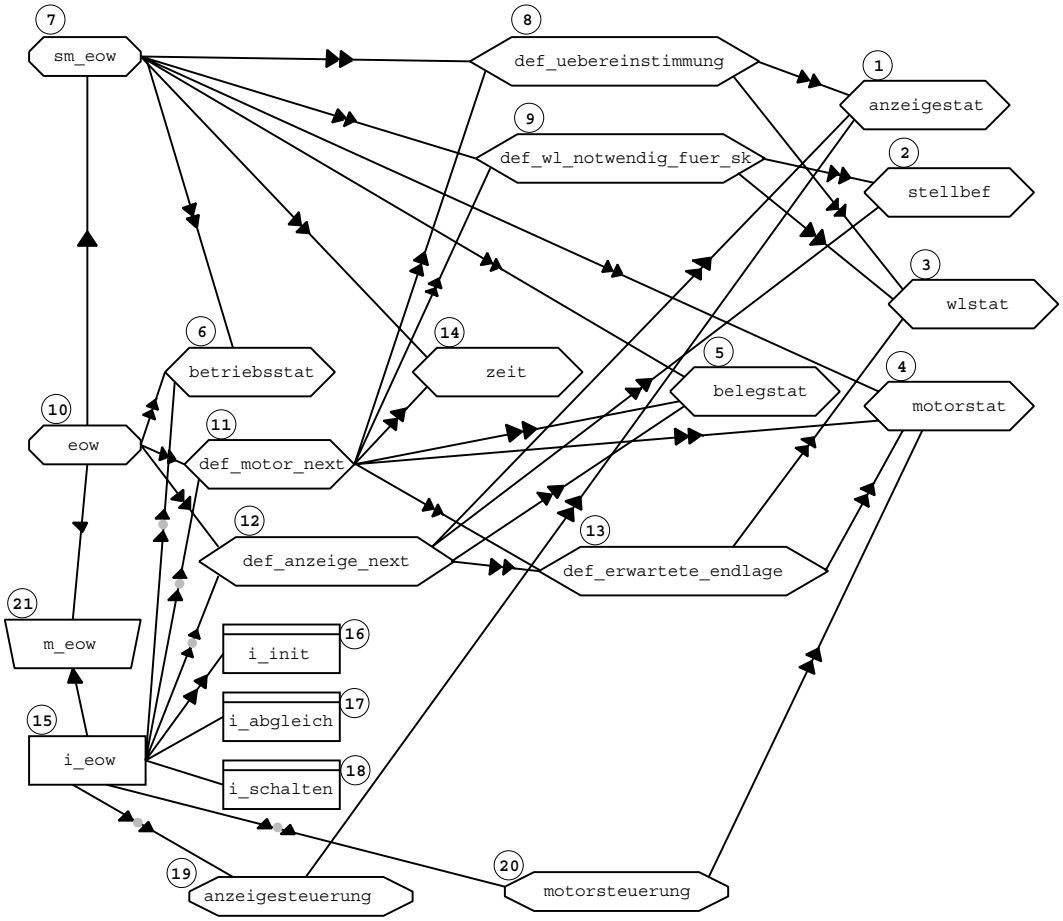
```
THEORY anzeigestat
 TYPES : anzeigestatus =
         FREELY GENERATED BY rechts |
                             links |
                             blinkend
THEORYEND
```

A.2 stellbef

```
THEORY stellbef
 TYPES : stellbefehl =
         FREELY GENERATED BY vbs |
                             sbs |
                             skr |
                             skl |
                             kein_auftrag
THEORYEND
```

A.3 wlstat

```
THEORY wlstat
 TYPES : weichenlagestatus =
         FREELY GENERATED BY rechtslage |
                             linkslage |
                             keine_endlage
THEORYEND
```



Die Pfeile stehen für unterschiedliche Beziehungen zwischen den Spezifikationstexten: Doppelpfeil für USING, Doppelpfeil mit Kreis für IMPORTSPEC, einfacher Pfeil für alle anderen Relationen.

Abbildung A.1: Entwicklungsgraph

A.4 motorstat

```
THEORY motorstat
 TYPES : motorstatus =
         FREELY GENERATED BY linkslauf |
                             rechtslauf |
                             stillstand
THEORYEND
```

A.5 belegstat

```
THEORY belegstat
 TYPES : belegungsstatus =
         FREELY GENERATED BY frei |
                             besetzt
THEORYEND
```

A.6 betriebsstat

```
THEORY betriebsstat
 TYPES : betriebsstatus =
         FREELY GENERATED BY einschalt_zustand |
                          regulaer
THEORYEND
```

A.7 sm_eow

```
OBJECT sm_eow
 PURPOSE :
 " Die Spezifikation sm_eow beschreibt die Sicherheitsanforderungen an die
   Steuerungssoftware der EOW. Die globale Sicherheitsanforderung besagt,
   dass der Weichenlage- und Ordnungsmelder mit dem Motorzustand
   und der Weichenlage uebereinstimmen muss (Invariante). Weitere
   Sicherheitanforderungen sind an einzelne Zustandsuebergaenge gebunden."
 USING : betriebsstat;
         belegstat;
         def_uebereinstimmung;
         def_wl_notwendig_fuer_sk;
         motorstat;
         zeit
 DATA : motor : motorstatus;
        anzeige : anzeigestatus;
        /* interne Annahme über die aktuelle Lage der Weiche */
        weichenlage : weichenlagestatus;
        /* aktueller Zustand der Steuerungssoftware:
             Initialisierung (einschalt_zustand) /laufender Betrieb (regulaer) */
        modus : betriebsstatus
 OPERATIONS :
 /* Abgleich des Systems mit den Daten aus der Umwelt direkt
    nach dem Einschalten des Systems. Selbstaetiges Herstellen der
    Betriebsbereitschaft nach dem Einschalten/Netzwiederkehr.
    */
 PROC abgleich(wl : weichenlagestatus)
      REQUIRES modus = einschalt_zustand
      ENSURES modus = regulaer;
              weichenlage = wl;
              uebereinstimmung ( wl,anzeige )
 /* Hauptroutine zur Abarbeitung der eingelesenen Sensordaten und Stellbefehle.
    Muss von aussen zyklisch aufgerufen werden (schalten terminiert und
    die Vor- / Nachbedingungen beschreiben Anforderungen an dei erreichbaren Zustaende)
    Im Sicherheitsmodell ist die Zeit irrelevant wird aber in der Parameterliste
    aufgefuehrt, da sie in der Leistungsspezifikation verwendet wird. */
 PROC schalten(quelle : stellbefehl,wl : weichenlagestatus,belegung : belegungsstatus,clk : uhr)
      REQUIRES modus = regulaer
      ENSURES modus = regulaer;
              /* Abgleich der externen Sensordaten mit der internen Repraesentation der Weichenlage. */
              intern_extern : weichenlage = wl;
              /* Erkennung, dass die Weiche aufgefahren wurde. */
              auffahrerkennung : NOT wl = weichenlage' AND
                                 motor' = stillstand ->
                                 anzeige = blinkend AND motor = stillstand;
              /* Einschraenkung der Zulaessigkeit eines Stellbefehls von der vorgezogenen Bedienstelle.
                 Stellauftraege von der oertlichen Bedienstelle sind nur zulaessig, wenn
                 - der Antrieb sich in Ordnungsstellung befindet und
                 - die zugehoerige Gleisfreimeldung keinen Besetztzustand zeigt.
                 */
              aktuator_vbs : quelle = vbs AND
                             motor' = stillstand AND
                             NOT motor = stillstand ->
                             NOT wl = keine_endlage AND
                             uebereinstimmung ( wl,anzeige' ) AND
                             belegung = frei;
              /* Einschraenkung der Zulaessigkeit eines Stellbefehls von einem der beiden Schienenkontakte.
```

```
               Stellauftraege von den Schienenkontakten duerfen nur ausgefuehrt werden wenn
               - der Gleisbereich nicht besetzt gemeldet ist
               - der Antrieb sich im ungestoerten Ruhezustand befindet
               - bei Umlaufanforderung aus Plusrichtung ueberwachte Minuslage vorliegt (und umgekehrt)
               */
            aktuator_sk : (quelle = skl OR
                           quelle = skr) AND
                          motor' = stillstand AND
                          NOT motor = stillstand ->
                          belegung = frei AND
                          NOT wl = keine_endlage AND
                          uebereinstimmung ( wl,anzeige' ) AND
                          wl_notwendig_fuer_sk ( wl,quelle );
            /* In sicherheitskritischen Situationen darf der Motor nur mit der Schluesselbedienstelle
               in Gang gesetzt werden. Sicherheitskritische Situationen:
               - Gleisfreimeldung zeigt Besetztzustand an
               - keine Endlage erreicht (und darin enthalten: Umlaufstoerung bzw. Weiche aufgefahren)
               - der Motor laeuft gerade und soll seine Laufrichtung aendern (reversieren)
               */
            aktuator_sbs : NOT motor = motor' AND
                           NOT motor = stillstand AND
                           (belegung = besetzt OR
                            wl = keine_endlage OR
                            NOT motor' = stillstand) ->
                           quelle = sbs
 /* Herstellung eines sicheren Anfangszustands unmittelbar nach dem
    Einschalten und vor Herstellung der Betriebsbereitschaft. */
 INITIAL : anzeige = blinkend;
           motor = stillstand;
           modus = einschalt_zustand
 INVARIANTS :
            /* - Uebereinstimmung zwischen der Weichenlage und der Lageanzeige.
               - Sichere Anzeige der ueberwachten Endlage durch Wichenlage- und Ordnungsmelder.
               */
            wl_motor_anzeige : (weichenlage = linkslage ->
                                anzeige /= rechts) AND
                               (weichenlage = rechtslage ->
                                anzeige /= links) AND
                               (motor /= stillstand OR
                                weichenlage = keine_endlage ->
                                anzeige = blinkend)
OBJECTEND
```

A.8 def_uebereinstimmung

```
THEORY def_uebereinstimmung
 USING : wlstat;
         anzeigestat
 PREDICATES : uebereinstimmung : weichenlagestatus,anzeigestatus
 VARS : w : weichenlagestatus;
        a : anzeigestatus
 ALGORITHMS : DEFPRED uebereinstimmung(w, a) <->
              SWITCH w IN
              CASE rechtslage : a = rechts
              CASE linkslage : a = links
              OUT : a = blinkend
              NI
THEORYEND
```

A.9 def_wl_notwendig_fuer_sk

```
THEORY def_wl_notwendig_fuer_sk
 USING : wlstat;
         stellbef
 PREDICATES :
              /* wl_notwendig_fuer_sk beschreibt die notwendige Bedingung,
                 damit ein Stellbefehl von einem Schienenkontakt angenommen werden kann */
              wl_notwendig_fuer_sk : weichenlagestatus,stellbefehl
 VARS : wl : weichenlagestatus;
        sb : stellbefehl
 AXIOMS : wl_notwendig_fuer_sk ( wl,sb ) <->
          wl = rechtslage AND
          sb = skl OR
          wl = linkslage AND
          sb = skr
THEORYEND
```

A.10 eow

```
OBJECT eow
 PURPOSE :
 "  Beschreibung der funktionalen Anforderungen an die Steuerung der EOW.
    Die Spezifikation stellt zwei Operationen bereit, die von einem darueberliegenden
    (konventionell entwickelten) Rahmensystem aufgerufen werden koennen.
    Das Rahmensystem traegt die Verantwortung dafuer, dass bei jedem Aufruf die
    aktuellen Sensordaten sowie die aktuelle Zeit als Parameter uebergeben werden.
    Um eine funktionsfaehige Steuerung zu entwickeln, ist es notwendig, das hier
    beschriebene Zustandsuebergangssystem zu beginn durch einen Aufruf der Operation
    'abgleich' mit den Sensordaten aus der Umwelt abzugleichen. Danach muss die
    Operation 'schalten' in einer Schleife des Rahmensystems zyklisch aufgerufen werden."
 USING : betriebsstat;
         def_anzeige_next;
         def_motor_next
 DATA : motor : motorstatus;
        letzterlauf : motorstatus;
        anzeige : anzeigestatus;
        /* interne Annahme über die aktuelle Lage der Weiche */
        weichenlage : weichenlagestatus;
        /* aktueller Zustand der Steuerungssoftware:
           Initialisierung (einschalt_zustand) /laufender Betrieb (regulaer) */
        modus : betriebsstatus;
        start : uhr
 OPERATIONS :
 /*   Abgleich des Systems mit den Daten aus der Umwelt direkt
      nach dem Einschalten des Systems */
 PROC abgleich(wl : weichenlagestatus)
      MODIFIES anzeige,weichenlage,modus
      REQUIRES modus = einschalt_zustand
      ENSURES modus = regulaer;
              weichenlage = wl;
              IF wl = linkslage
              THEN anzeige = links
              ELSE IF wl = rechtslage
                   THEN anzeige = rechts
                   ELSE anzeige = blinkend
                   FI
              FI
 /*
      Hauptroutine zur Abarbeitung der eingelesenen Sensordaten und Stellbefehle.
      Muss von aussen zyklisch aufgerufen werden (schalten terminiert und
      die Vor- / Nachbedingungen beschreiben Anforderungen an die erreichbaren Zustaende)
     */
 PROC schalten(quelle : stellbefehl,wl : weichenlagestatus,belegung : belegungsstatus,clk : uhr)
      /* Anforderungen an bzw. Annahmen ueber die Aussenwelt und somit
         gleichzeitig Zusicherungen an die Prozedur 'schalten'. */
```

```
      REQUIRES modus = regulaer;
               /* Weichenlageaenderung und Stellbefehl treten nicht gleichzeitig ein. */
               NOT (weichenlage /= wl AND
                    quelle /= kein_auftrag);
               /* Die Weichenlage kann sich nicht von einer Endlage in die andere
                  Endlage verlagern, ohne dass eine ``Zwischenlage'' eingenommen
                  wird (physikalisch unmoeglich). */
               NOT (wl /= weichenlage AND
                    wl /= keine_endlage AND
                    weichenlage /= keine_endlage);
               /* Wenn der Motor nach rechts (links) laeuft (in `` keine_endlage''
                  beginnend) kann die Weiche physikalisch nicht in linkslage
                  (rechtslage) ankommen. */
               NOT (weichenlage = keine_endlage AND
                    motor /= stillstand AND
                    wl /= keine_endlage AND
                    wl /= erwartete_endlage ( motor ));
               /* Unter betrieblichen Gesichtspunkten wird folgender Fall ausgeschlossen:
                  Der Motor laeuft an (aufgrund eines Stellbefehls), die Weiche
                  hat die Endlage noch nicht verlassen und ein neuer Stellbefehl
                  vom Schluesselschalter kommt an. */
               NOT (motor /= stillstand AND
                    erwartete_endlage ( motor ) /= wl AND
                    wl /= keine_endlage AND
                    quelle = sbs)
      ENSURES modus_beibehalten : modus = modus';
              wl_schalten : weichenlage = wl;
              /* letzte ``echte'' Motorbewegung merken (d.h. stillstand wird nicht festgehalten) */
              letztenlauf_merken : IF motor /= stillstand
                                   THEN letzterlauf = motor
                                   ELSE letzterlauf = letzterlauf'
                                   FI;
              /* Zeitpunkt zu dem der Motor eingeschaltet wurde merken */
              einschaltzeit_merken : IF motor' = stillstand AND
                                        motor /= stillstand
                                     THEN start = clk
                                     ELSE start = start'
                                     FI;
              /* Aktualisierung des Weichenlage- und Ordnungsmelders. */
              anzeige_schalten : anzeige = anzeige_next ( anzeige',motor',weichenlage',
                                                          quelle,wl,belegung );
              /* Aktualisierung des Motorantriebs. */
              motor_schalten : motor = motor_next ( anzeige',motor',letzterlauf',weichenlage',
                                                    quelle,wl,belegung,start',clk )
 INITIAL : anzeige = blinkend;
           motor = stillstand;
           modus = einschalt_zustand
 /* Beweisinvariante (Hilfsinvariante) um die Anforderungen aus dem
    Sicherheitsmodell nachweisen zu koennen. Sie beschreiben keine zusaetzlichen
    funktionalen Anforderungen an die Weichensteuerung. */
 INVARIANTS : anzeige = blinkend OR
              ((weichenlage = linkslage OR
                weichenlage = rechtslage) AND
               motor = stillstand);
              modus = einschalt_zustand -> motor = stillstand;
              motor /= stillstand -> letzterlauf = motor;
              motor /= stillstand -> weichenlage /= erwartete_endlage(motor)
 /* Das hier spezifizierte System muss die Anforderungen aus dem Sicherheitsmodell
    `sm_eow' erfuellen. */
 SATISFIES : sm_eow
OBJECTEND
```

A.11 def_motor_next

```
THEORY def_motor_next
 USING : def_wl_notwendig_fuer_sk;
         def_erwartete_endlage;
         motorstat;
         belegstat;
         zeit;
         def_uebereinstimmung
 FUNCTIONS :
            /* motor_next liefert den naechsten Zustand fuer die Motorsteuerung.
                  Die ersten vier Parameter beschreiben ``alte'' Zustandswerte,
                  der ertse ``motorstatus'' ist der tatsaechliche Motorstatus
                  der zweite ``motorstatus'' ist die zuletzt gemerkte Laufrichtung
                  die restlichen sind aktuell eingelesene Werte */
            motor_next : anzeigestatus,motorstatus,motorstatus,weichenlagestatus,
                         stellbefehl,weichenlagestatus,belegungsstatus,uhr,uhr -> motorstatus
 PREDICATES :
             /* Uberpruefung der Zulaessigkeit des Motorlaufs */
             anf_fuer_motorlauf : anzeigestatus,motorstatus,weichenlagestatus,
                                  stellbefehl,weichenlagestatus,belegungsstatus;
             /* Ueberpruefung der Zulaessigkeit des Motorlaufs nach rechts unter
                der Voraussetzung, dass ein Motorlauf zulaessig ist. */
             anf_fuer_rechtslauf : motorstatus,motorstatus,stellbefehl,weichenlagestatus;
             /* Hilfspraedikat zur Ueberpruefung, ob die zulaessige Laufzeit
                fuer den Motor ueberschritten wurde.*/
             motor_timeout : motorstatus,uhr,uhr
 VARS : as : anzeigestatus;
        ms, ll : motorstatus;
        ws_alt, ws_aktuell : weichenlagestatus;
        sb : stellbefehl;
        bs : belegungsstatus;
        clk_motor_start, clk_aktuell : uhr
 ALGORITHMS : DEFPRED anf_fuer_motorlauf(as, ms, ws_alt, sb, ws_aktuell, bs) <->
              SWITCH sb IN
              /* Stellbefehl von der Schluesselbedienstelle wird immer angenommen
                 und fuehrt zu einem Anlaufen des Motors */
              CASE sbs : TRUE
              /* Wenn kein Stellbefehl kommt, darf der Motor im naechsten Zustand laufen,
                 wenn der Motor zuvor bereits in Bewegung war
                 und die Anzeige in Uebereinstimmung damit ist (d.h. blinkt)
                 und wenn die Weichenlage noch nicht die erwartete Endlage
                 erreicht hat.
                 */
              CASE kein_auftrag : as = blinkend AND ms /= stillstand AND
                          (ws_alt = ws_aktuell OR ws_aktuell = keine_endlage)
              /* Fall: Ein Stellbefehl kommt von ``vbs'' oder von ``skl'' oder
                 von ``skr''. Hierbei kann es auf zwei Arten dazu kommen,
                 dass der Motor im naechsten Zustand in Bewegung ist:
                 1. Der Motor soll aufgrund eines Stellbefehls in Bewegung gesetzt werden.
                 2. Er war bereits in Bewegung und hat noch nicht die erwartete Endlage erreicht;
                    der Stellbefehl wird ignoriert.
                 */
              OUT : IF ms = stillstand
                    THEN uebereinstimmung ( ws_aktuell,as ) AND ws_aktuell /= keine_endlage AND
                         bs = frei AND (sb = vbs OR wl_notwendig_fuer_sk ( ws_aktuell,sb ))
                    ELSE as = blinkend
                    FI
              NI;
              DEFPRED anf_fuer_rechtslauf(ms, ll, sb, ws_aktuell) <->
              /* ``rechtslauf'' ist zulaessig, wenn der Motor sich bereits im
                 ``rechtslauf'' befindet und die Richtung nicht geaendert werden soll
                 oder wenn die Weiche sich in keiner Endlage befindet und
                 die zuletzt gemerkte Laufrichtung ``linkslauf'' war
                 oder wenn sich die Weiche in ``linkslage'' befindet.
                 */
              (sb /= sbs AND ms = rechtslauf) OR
```

```
            (sb = sbs AND ws_aktuell = keine_endlage AND ll = linkslauf) OR
            ws_aktuell = linkslage;
            /* Der Motor muss nach Ueberschreiten der von dem Praedikat ``timeout''
               vorgegebenen Zeitschranke abgeschaltet werden. */
            DEFPRED motor_timeout(ms, clk_motor_start, clk_aktuell) <->
            timeout ( clk_motor_start,clk_aktuell ) AND ms /= stillstand;
            DEFFUNC motor_next(as, ms, ll, ws_alt, sb, ws_aktuell, bs, clk_motor_start, clk_aktuell) =
            IF motor_timeout ( ms,clk_motor_start,clk_aktuell )
            THEN
                 /* Wenn der Motor zu lange laeuft (``timeout'') wird er abgeschaltet. */
                 stillstand
            ELSE IF
                    /* Ueberpruefung der Voraussetzungen um den Motor
                       in Betrieb zu setzen bzw. in Betrieb zu belassen */
                    anf_fuer_motorlauf ( as,ms,ws_alt,sb,ws_aktuell,bs )
                 THEN IF
                         /* Ueberpruefung der Voraussetzung um den Motor in
                            ``rechtslauf'' zu versetzen bzw. zu belassen */
                         anf_fuer_rechtslauf ( ms,ll,sb,ws_aktuell )
                      THEN rechtslauf
                      ELSE linkslauf
                      FI
                 ELSE stillstand
                 FI
            FI
THEORYEND
```

A.12 def_anzeige_next

```
THEORY def_anzeige_next
 USING : stellbef;
         anzeigestat;
         def_erwartete_endlage;
         belegstat
 FUNCTIONS :
             /* anzeige_next liefert den neuen Wert fuer den Weichenlage- und Ordnungsmelder.
                Die ersten drei Parameter geben die ``alten'' Werte an,
                die restlichen sind die aktuell von aussen eingelesenen Werte */
             anzeige_next : anzeigestatus,motorstatus,weichenlagestatus,stellbefehl,
                            weichenlagestatus,belegungsstatus -> anzeigestatus
 PREDICATES :
              /* anf_fuer_endlageanzeige ist ein Hilfspraedikat, welches die Bedingungen festlegt,
                 um auf dem Weichnlage- und Ordnungsmelder eine Endlage anzuzeigen.
                 Die ersten drei Parameter geben die ``alten'' Werte an,
                 die restlichen sind die aktuell von aussen eingelesenen Werte */
              anf_fuer_endlageanzeige : anzeigestatus,motorstatus,weichenlagestatus,
                                        stellbefehl,weichenlagestatus,belegungsstatus
 VARS : as : anzeigestatus;
        ms : motorstatus;
        sb : stellbefehl;
        ws_alt, ws_aktuell : weichenlagestatus;
        bs : belegungsstatus
 ALGORITHMS : DEFPRED anf_fuer_endlageanzeige(as, ms, ws_alt, sb, ws_aktuell, bs) <->
              IF ms = stillstand
              THEN
                   /* Motor steht still und die Weiche befindet sich
                      und bleibt in einer Endlage */
                   ws_aktuell /= keine_endlage AND ws_aktuell = ws_alt AND
                   as /= blinkend AND
                   (sb = kein_auftrag OR
                    sb /= sbs AND bs = besetzt OR
                    sb = skl AND ws_aktuell = linkslage OR
                    sb = skr AND ws_aktuell = rechtslage)
              ELSE
                   /* Fall: Weiche war im Umlauf und hat nun eine Endlage erreicht */
                   ws_alt = keine_endlage AND as = blinkend AND
```

```
                sb = kein_auftrag AND ws_aktuell /= keine_endlage
            FI;
            DEFFUNC anzeige_next(as, ms, ws_alt, sb, ws_aktuell, bs) =
            IF anf_fuer_endlageanzeige ( as,ms,ws_alt,sb,ws_aktuell,bs )
            THEN IF ws_aktuell = rechtslage
                 THEN rechts
                 ELSE links
                 FI
            ELSE blinkend
            FI
THEORYEND
```

A.13 def_erwartete_endlage

```
THEORY def_erwartete_endlage
 USING : motorstat;
         wlstat
 FUNCTIONS : erwartete_endlage : motorstatus -> weichenlagestatus
 AXIOMS : erwartete_endlage ( rechtslauf ) = rechtslage AND
          erwartete_endlage ( linkslauf ) = linkslage
THEORYEND
```

A.14 zeit

```
THEORY zeit
 PURPOSE :
 " Diese Theorie dient der Bereitstellung eines Typs fuer die Behandlung von Zeit sowie
   eines Praedikats, das bei Ueberschreiten einer bestimmten Differenz zwischen zwei
   Zeitpunkten TRUE und ansonsten FALSE liefert.
   Sowohl der Typ als auch das Praedikat werden nicht weiter spezifiziert und es bleibt
   als informelle Anforderung, dass die Implementierung dieser Theorie die reale Umwelt
   adaequat umsetzt. Davon haengt auch ab, ob Formeln, in denen ``uhr'' und ``timeout''
   auftreten die reale Welt korrekt beschreiben."
 TYPES : uhr
 PREDICATES :
            /* timeout erhaelt als Parameter eine Startzeit und eine Endzeit.
                 Falls die Differenz zwischen der Startzeit und der Endzeit eine
                 gewisse Schranke ueberschreitet, soll timeout TRUE liefern
                 und ansonsten FALSE.
                 Die Zeitschranke ist in dieser Modellierung hart in das Praedikat
                 eingebunden.
                */
            timeout : uhr,uhr
THEORYEND
```

A.15 i_eow

```
MODULE i_eow
 PURPOSE :
 " Erster Verfeinerungsschritt: Implementierung der Spezifikation eow
   durch ein abstraktes Programm."
 IMPORTSPEC : def_motor_next;def_anzeige_next;betriebsstat;anzeigesteuerung;motorsteuerung
 DATA :
        /* Aktueller Zustand der Steuerungssoftware */
        modus : betriebsstatus;
        /* Interne Variable zur Repraesentierung der Weichenlage */
        weichenlage : weichenlagestatus;
        /* Variable, in der die zuletzt am Motor angelegte Laufrichtung
             gespeichert wird. */
        letzterlauf : motorstatus;
        /* Variable, um den Einschaltzeitpunkt des Motors festzuhalten. */
        start : uhr
 ELEMENTS :
            /* Impementiertung der Operation 'abgleich' */
            i_abgleich,
                        /* Implementierung der Operation 'schalten' */
                        i_schalten
 /* Die MAIN-Prozedur muss die Anfordeungen der INITIAL-Klausel erfuellen */
 MAIN : i_init
MODULEEND
```

A.16 i_init

```
PROCEDURE i_init
 PURPOSE :
 " Anfangszustand des Steuerungssystems herstellen."
 BODY :
        /* Die folgenden Anweisungen sind redundant;
            siehe INITIAL-Klausel in anzeigesteuerung und motorsteuerung

            ansteuern(blinkend);
            abschalten;
            */
        modus := einschalt_zustand
PROCEDUREEND
```

A.17 i_abgleich

```
PROCEDURE i_abgleich
 PURPOSE : " Im einschalt_zustand werden die internen Daten mit
             den externen Sensordaten abgeglichen."
 PARAMS : wl : IN weichenlagestatus
 BODY : modus := regulaer;
        weichenlage := wl;
        IF wl = linkslage
        THEN ansteuern(links)
        ELSE IF wl = keine_endlage
             THEN ansteuern(blinkend)
             ELSE ansteuern(rechts)
             FI
        FI
PROCEDUREEND
```

A.18 i_schalten

```
PROCEDURE i_schalten
 PURPOSE : " Impelementierung der Operation 'schalten'.
  Diese Operation muss zyklisch mit den aktuellen Sensordaten aufgerufen werden.
  Sie dient der Ansteuerung des Motors und des Weichenlage- und Ordnungsmelders."
 PARAMS : quelle : IN stellbefehl;
          wl : IN weichenlagestatus;
          belegung : IN belegungsstatus;
          clk : IN uhr
 BODY : DECLARE
        /* Alten Wert der Anzeige zwischenspeichern */
        anz_zuvor: anzeigestatus:=anzeigesteuerung.wert,
        /* Alten Wert der Motorlaufrichtung zwischenspeichern */
        motor_zuvor: motorstatus:=motorsteuerung.wert,
        /* Der Wert, den die Motorsteuerung nach Ausfuehrung der Prozedur annehmen soll. */
        motor_nachher: motorstatus:=motor_next ( anzeigesteuerung.wert,motorsteuerung.wert,
                                                 letzterlauf,weichenlage,quelle,wl,belegung,start,clk );
        /* Anforderung: anzeige_schalten */
        ansteuern(anzeige_next ( anz_zuvor,motor_zuvor,weichenlage,quelle,wl,belegung ));
        /* Anforderungen: motor_schalten, letztenlauf_merken, einschaltzeit_merken */
        IF motor_zuvor /= motor_nachher
        THEN IF motor_nachher = stillstand
             THEN abschalten
             ELSE IF motor_nachher = rechtslauf
                  THEN einschalten_rl
                  ELSE einschalten_ll
                  FI;
                  /* Anforderung: einschaltzeit_merken */
                  IF motor_zuvor = stillstand
                  THEN start := clk
                  FI;
                  /* Anforderung: motorlauf_merken */
                  letzterlauf := motor_nachher
             FI
        FI;
        /* Anforderung: wl_schalten */
        weichenlage := wl
PROCEDUREEND
```

A.19 anzeigesteuerung

```
OBJECT anzeigesteuerung
 PURPOSE :
 " Spezifikation der Softwarekomponente zur Ansteuerung der Anzeige."
 USING : anzeigestat
 DATA :
        /* Zustandsobjekt zur Ansteuerung des Weichenlage und Ordnungsmelders */
        anzeige : anzeigestatus
 OPERATIONS :
 PROC wert : anzeigestatus
      ENSURES anzeige = RESULT
 PROC ansteuern(neuer_wert : anzeigestatus)
      MODIFIES anzeige
      ENSURES neuer_wert = anzeige
 INITIAL : anzeige = blinkend
OBJECTEND
```

A.20 motorsteuerung

```
OBJECT motorsteuerung
 PURPOSE :
 " Spezifikation der Softwarekomponente zur Ansteuerung des Motors."
 USING : motorstat
 DATA : motor : motorstatus
 OPERATIONS :
 PROC wert : motorstatus
      ENSURES RESULT = motor
 PROC abschalten
      MODIFIES motor
      ENSURES motor = stillstand
 PROC einschalten_rl
      MODIFIES motor
      ENSURES motor = rechtslauf
 PROC einschalten_ll
      MODIFIES motor
      ENSURES motor = linkslauf
 INITIAL : motor = stillstand
OBJECTEND
```

A.21 m_eow

```
MAPPING m_eow
 PURPOSE :
 " Das Mapping stellt den formalen Zusammenhang zwischen der Leistungsspezifikation eow
   und dem abstrakten Programm i_eow."
 EXPORTSPEC : eow
 IMPLEMENTATION : i_eow
 MAPS :
        /* Folgende Maps sind redundant, da die Namen gleich sind. */
        motorsteuerung.motor IMPLEMENTS eow.motor;
        anzeigesteuerung.anzeige IMPLEMENTS eow.anzeige;
        letzterlauf IMPLEMENTS letzterlauf;
        weichenlage IMPLEMENTS weichenlage;
        modus IMPLEMENTS modus;
        start IMPLEMENTS start;
        /* Die Namen sind verschieden und somit muessen Maps angegeben werden. */
        i_abgleich IMPLEMENTS abgleich;
        i_schalten IMPLEMENTS schalten
MAPPINGEND
```

Glossar

Kurzbeschreibungen und Verweise auf zugehörige Seiten für Abkürzungen, Begriffe, Notationen und Symbole in diesem Dokument.

Begriff	Beschreibung	Seite
EOW	Elektrisch ortsbediente Weiche	2
Sbs,`sbs`	Schlüsselbedienstelle (an der Weichenspitze)	2
Skl,`skl`	Schienenkontakt links	2
Skr,`skr`	Schienenkontakt rechts	2
WLM	Weichenlage- und Ordnungsmelder	2
vBs,`vbs`	Vorgezogene Bedienstelle	2
$\rightarrow$,$\leftrightarrow$,$\neg$,$\wedge$,$\vee$	Die üblichen logischen Konnektive in mathematischen Formeln; alternative Schreibweise in ASCII-Texten: `->`, `<->`, `NOT`, `AND` und `OR`	12
$\vdash$	teilt eine Beweisverpflichtung in Prämisse und Konklusion: aus der Prämisse muß die Konklusion ableitbar sein	30
$'$	(Apostroph) bezeichnet in der ENSURES-Klausel von VSE-SL Spezifikationen den Wert eines Zustandsobjektes vor Aufruf der spezifizierten Operation	12
`abgleich`	Operation in der EOW-Steuerungssoftware, die in der Steuerung einen definierten Ausgangszustand herstellt	11
`anzeige`	Zustandsobjekt, das den aktuell auf dem Weichenlage- und Ordnungsmelder angezeigten Wert aus `anzeigestatus` (`links`, `rechts` oder `blinkend`) enthält	10
`as`	Funktionsparameter: Anzeigewert auf WLM	19
`belegung`	Parameter der Operation `schalten`; er enthält den aktuellen `belegungsstatus`, d.h. `frei` oder `besetzt`, für den Weichenabschnitt	11
`bs`	Funktionsparameter: Frei-/Besetztmeldung für Weichenabschnitt	19
`letzterlauf`	Zustandsobjekt um die zuletzt anliegende Motorlaufrichtung zu speichern	15
`ll`	Funktionsparameter: Intern gespeicherte letzte Laufrichtung	34
`modus`	Zustandsobjekt zur Kontrolle der Ablaufsteuerung; es kann den Wert `regulaer` oder `einschalt_zustand` enthalten	11
`motorstatus`	Datenstruktur für die Betriebszustände des Motors; enthält die Werte `linkslauf`, `rechtslauf` und `stillstand`	9

Literaturverzeichnis

[1] Baur et al. The Verification Support Environment VSE. In *IFA Symposium on Safety, Security and Reliability of Computers*, 1992.

[2] Bundesamt für Sicherheit in der Informationstechnik (BSI), Bonn. *VSE Handbuch*, März 1995.

[3] Ercüment Canver, Jan-Tecker Gayen, and Adam Moik. Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE — Erweiterte Version. Eine um Beweisprotokolle erweiterte Version dieses Berichtes kann über ftp bezogen werden: ftp.informatik.uni-ulm.de:/pub/papers/uib/UIB-96-01-ext.ps.Z

[4] Peter Göhner. Spezifikation und Verifikation von sicheren Softwaresystemen. *Automatisierungstechnische Praxis*, 4:24–31, 1995.

[5] IVV GmbH, Braunschweig. *MC EOW Systembeschreibung, Mikro-Computergesteuerte elektrisch-ortsbediente Weiche.* Firmeninternes Dokument.

[6] IVV GmbH, Braunschweig. *Anforderungskatalog für die Mikroprozessorgesteuerte, elektrisch ortsbediente Weiche (MC EOW-2M).* Firmeninternes Dokument.

[7] K.L. McMillan. *The SMV system.* Carnegie Mellon University, draft edition, February 1992.

[8] *Vorschrift für elektrische Bahn-Signalanlagen (DIN 57831, VDE 0831).*

[9] *Eisenbahn-Bau- und Betriebsordnung (EBO) für Bahnen des öffentlichen Verkehrs.*

[10] Eisenbahn-Bau-Betriebsordnung für Anschlußbahnen (EBOA) für Bahnen des nichtöffentlichen Verkehrs (Privatanschlußbahnen).

[11] *Bau- und Betriebsordnung für Straßenbahnen (BoStrab) für Stadt-, U- und Straßenbahnen.*