(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 796 779 A1

(12) EUROPÄISCHE PATENTANMELDUNG

(43) Veröffentlichungstag:
24.09.1997 Patentblatt 1997/39

(51) Int. Cl.[6]: **B61L 21/00**

(21) Anmeldenummer: **96104501.0**

(22) Anmeldetag: **21.03.1996**

(84) Benannte Vertragsstaaten:
AT BE DE DK FR GB IT NL PT SE

(71) Anmelder: **Alcatel Austria Aktiengesellschaft**
A-1210 Wien (AT)

(72) Erfinder: **Montigel, Markus, Dr.**
1200 Wien (AT)

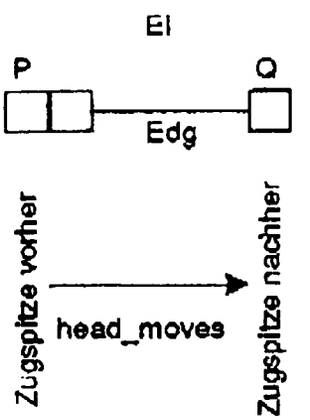
(74) Vertreter: **Pechhold, Eberhard, Dipl.-Phys. et al**
Alcatel Alsthom
Intellectual Property Department,
Postfach 30 09 29
70449 Stuttgart (DE)

(54) Verfahren zur Steuerung und Sicherung eines spurgeführten Transportsystems

(57) Es wird ein Verfahren zur Steuerung und Sicherung des Betriebes eines spurgeführten Transportsystems mit Hilfe einer Datenverarbeitungsanlage angegeben, in der ein Modell aller im Transportsystem z.B. im Gleisnetz einer Eisenbahnanlage vorkommenden Zustände und Ereignisse in einer formalen Sprache gespeichert ist.

Typbezogene, dynamische Daten sind in Form von Petri-Netzen oder aus diesen abgeleiteten Netzen, exemplarbezogene, statische Daten (Gleisnetzdaten) in Form von Doppelpunktgraphen dargestellt. Ein erster Algorithmus berechnet vor Auslösung eines kontrollierbaren Ereignisses alle möglichen Folgezustände und unterbindet die Auslösung, wenn unter diesen ein unzulässiger Zustand ist. Ein weiterer Algorithmus tritt in Funktion, wenn ein normalerweise nicht anzunehmendes unzulässiges Ereignis (z.B. Schienenbruch) eintritt, und löst kontrollierbare Ereignisse aus, wenn diese die Zahl der infolge des unzulässigen Ereignisses zu erwartenden unzulässigen Zustände vermindern. Geschwindigkeitsbegriffe und verschiedene Fahrzeugmanöver werden in das o.g. Modell einbezogen.

Fig. 3



EP 0 796 779 A1

Beschreibung

Die Erfindung betrifft ein Verfahren gemäß dem Oberbegriff des Patentanspruchs 1.

Ein solches Verfahren ist aus der Dissertationsschrift von M. Montigel "Modellierung und Gewährleistung von Abhängigkeiten in Eisenbahnsicherungsanlagen", vorgelegt 1994 an der Eidgenössischen Technischen Hochschule, Zürich (DISS.ETH Nr. 10776) bekannt.

Es soll die heute verwendeten Verfahren der Steuerung und Sicherung spurgeführter Transportsysteme, insbesondere der Eisenbahnnetze ersetzen und darüberhinaus die Planung und wirtschaftliche Nutzung solcher Transportsysteme auf eine völlig neue Art und Weise unterstützen.

Die heute verwendeten Lösungsarten zur Systemsteuerung und -Sicherung stützen sich auf Gleiselemente mit Nachbarbeziehungen, versehen mit Zuggeschwindigkeitsdaten, die zur Bildung einer Fahrstraße entweder über geographische Stromkreise miteinander verknüpft und anschließend gesichert werden, oder entsprechend einer vorbereiteten, die einzustellende Fahrstraße wiedergebenden Liste nacheinander auf ihren Belegungszustand und ihre aktuelle Einstellung hin geprüft und, ggf. nach Umstellung in eine vorgegebene Lage, gesichert werden.

Diese Lösungen sind unbefriedigend im Hinblick auf eine Behandlung des Systems mit formalen Methoden. Das Konstrukt Gleiselement, von dem es relativ viele Basistypen gibt, ist hierzu noch zu komplex.

Die o.g. Dissertation sieht deshalb die Darstellung des Gleisnetzes mit Hilfe von Doppelpunktgraphen vor, welche gegenüber einer Darstellung mit einfachen gerichteten Graphen den Vorteil hat, daß keine unnatürliche Auszeichung einer Fahrtrichtung erfolgt (was bei Gleisdreiecken und Wendeschleifen zu Problemen führt), und daß topologische Besonderheiten besser berücksichtigt werden können, beispielsweise Spitzkehren auf Weichen von vornherein nicht möglich sind, somit nicht extra ausgeschlossen werden müssen.

In der eingangs angegebenen Dissertation wird das System Eisenbahnsicherung als verteiltes diskretes Ereignissystem verstanden, das mit Hilfe formaler Sprachen als logisches Datenmodell wiedergegeben werden kann, wobei einzelne Datenbereiche wie eingangs angegeben repräsentiert sind.

Die dort vorgeschlagene Methode beruht auf nahen Abhängigkeiten und verzichtet zunächst auf die Formulierung von Sicherheitsbedingungen. Es wird ein Modell der Zustände und Ereignisse des Systems entwickelt, das die Formulierung von unzulässigen Zuständen und Ereignissen auf lokaler Basis erlaubt, d.h. es brauchen keine fernen Abhängigkeiten betrachtet zu werden. Die Anforderungen an das Sicherungssystem werden nicht direkt beschrieben, sondern das Verhalten des Systems inklusive der sich darin fortbewegenden Fahrzeuge wird als solches modelliert, wie wenn keine Sicherungsebene vorhanden wäre. Die in der Sicherheitsebene zu implementierenden Sicherheitsbedingungen ergeben sich dann als Konsequenzen der modellierten Systemeigenschaften.

Statt, wie sonst üblich, im Sicherungssystem die Sicherheitsbedingungen explizit zu implementieren, wird eine Repräsentation aller für die betriebliche Sicherheit für relevant gehaltenen Zustände und Ereignisse entwickelt, insbesondere auch solcher, die aus Sicherheitsgründen als unzulässig betrachtet werden. Die Ereignisse werden in kontrollierbare und unkontrollierbare aufgeteilt. Ein Ereignis heißt genau dann kontrollierbar, wenn in jedem beliebigen Systemzustand die Freiheit besteht, dieses am Eintreffen zu hindern, andernfalls heißt das Ereignis unkontrollierbar.

Zur Laufzeit enthält das Sicherungssystem stets ein Abbild aller Zustände des Gesamtsystems sowie die Regeln, wie aus dem aktuellen Zustand sämtliche Folgezustände berechnet werden können. Im weiteren ist ein Algorithmus (sog. Eunomischer Algorithmus) implementiert, der jedesmal, wenn ein kontrollierbares Ereignis ausgelöst werden soll, die Menge aller möglichen, durch Folgen von unkontrollierbaren Ereignissen hervorgerufenen Folgezustände berechnet und daraufhin untersucht, ob sie einen oder mehrere unzulässige Zustände enthält.

Um die große Anzahl Zustände platzsparend im System darstellen zu können, wurde eine verteilte Repräsentation mit Petri-Netzen gewählt. So können mit n binären Stellen oder Bedingungen 2^n Zustände dargestellt werden. Der oben erwähnte Algorithmus, der die Zulässigkeit eines kontrollierbaren Ereignisses prüft, berechnet dann nicht wirklich die Menge der unkontrollierbar erreichbaren Folgezustände, da diese viel zu umfangreich wäre, sondern er klärt lediglich die Herleitbarkeit jeder einzelnen Stelle bzw. jedes einzelnen Ereignisses einzeln (nicht in Kombination) ab. Es kann gezeigt werden, daß jeder erreichbare Zustand bzw. jedes erreichbare Ereignis herleitbar im obigen Sinne ist, so daß kein unzulässiger Zustand erreicht werden kann, wenn die entsprechenden Stellen nicht herleitbar sind.

Der Entwicklungsprozess eines Sicherungssystems gestaltet sich bei dem in der Dissertation wiedergegebenen Verfahren wie folgt:

- Schrittweises Entwickeln eines formalen Modells der Sicherungsebene des Transportsystems auf der Ebene eines Systemtyps (z.B. System für ein bestimmtes Transportsystem oder eine bestimmte Bahnverwaltung) in Form eines Predicate/Transition-Netzes. Dieses Modell beschreibt die Zustände und Zustandsübergänge der Sicherungsebene und ist unabhängig von den zu sichernden Gleisnetzen. Wenn für dasselbe oder ein ähnliches Transportsystem bereits früher ein Modell erarbeitet wurde, können die dort definierten allgemeinen Systemeigenschaften für das neue Modell übernommen werden.

- Erstellen der Spezifikationen der zu sichernden Gleisnetze (Daten der Systemexemplare)

- Durchführen eines automatisierten sogenannten Entfaltungsprozesses mittels Entfaltungstools:

  - Syntax-, Typen- und Plausibilitätsprüfung des Typ-Modells
  - Vorbereitung des Typ-Modells für den eigentlichen Entfaltungsprozess
  - Erzeugung von symbolischen Schnittstellen für die von der Sicherungsebene abhängigen Ebenen (Steuerungsebene, Hardware-Interface-Ebene etc.)
  - Herstellen eines interpretier- oder ausführbaren formalen Modells für die Systemexemplare (eigentliche Entfaltung): Für jeden allgemein beschriebenen Zustand und Zustandsübergang im abstrakten Typmodell werden alle konkreten Instanzen gesucht, die sich aus den Exemplardaten ergeben. Resultat ist ein Condition/Event-Netz, daß das entsprechende Systemexemplar beschreibt. Jeder Knoten des Netzes besitzt einen eindeutigen Namen, der sich aus dem Namen des entsprechenden abstrakten Knotens und den Namen der konkreten Objekte des Gleisnetzes zusammensetzt, auf die er sich bezieht.

- Überprüfen des formalen Exemplar-Modells mit einem Laufzeitsystem für Condition/Event-Netze. Dieses System enthält eine Implementation des o.g. Eunomischen Algorithmus, der die Zulässigkeit von kontrollierbaren Ereignissen überwacht. In der o.g. Dissertation wurden die von der Sicherungsebene abhängigen Ebenen (Außenanlage, Steuerungsebene etc.) mit interaktiven Tools simuliert. Die Sicherungsebene kann sowohl auf hohem Abstraktionsniveau (Simulation von Zügen) wie auch auf dem Detailniveau (Betrachtung einzelner Knoten des Condition/Event-Netzes) simuliert werden.

Das in der o.g. Dissertation wiedergegebene Verfahren enthält noch kein Konzept für eine Fahrstraßenüberwachung und für verschiedene andere in Eisenbahnsteuerungssystemen heute übliche Steuerungskonzepte.

Es ist deshalb Aufgabe der Erfindung, eine Funktion in das bekannte Verfahren zu integrieren, die eine Überwachung eines zunächst für zulässig erachteten Fahrweges ermöglicht.

Diese Aufgabe wird durch die im Patentanspruch 1 angegebenen Merkmale gelöst.

Durch die Beachtung des Eintrittes sogenannter "normalerweise nicht anzunehmender Ereignisse", also Ereignissen wie z.B. Schienenbrüchen oder Verlieren der Endlage bei Weichen, deren Berücksichtigung von Anfang an jedes Fahren in einem Gleisnetz unzulässig machen würde, mit Hilfe des zusätzlichen, sogenannten revertierten eunomischen Algorithmus lassen sich gefährliche Folgen aus solchen Ereignissen in vielen Fällen vermeiden oder wenigstens in ihrer Eintrittswahrscheinlichkeit herabsetzen.

Weiterbildungen des Verfahrens nach der Erfindung sind in den Unteransprüchen angegeben und betreffen weitere einzubeziehende Steuerungskonzepte.

So betreffen die Ansprüche 2 und 3 den Einbezug des Geschwindigkeitsbegriffs in das Steuerungssystem, Anspruch 3 betrifft die Behandlung von Zugoperationen und die Interpretation von Abschnittsbelegungen.

Anspruch 4 betrifft die Darstellung von Transporteinheiten mit Hilfe von Folgen von belegten Gleisabschnitten.

Gegenstand des Anspruchs 5, schließlich, sind der Einsatz neuer Transporteinheiten und die Erfassung von Zugtrennungen mittels Belegung und Freimeldungen von Gleisabschnitten.

Nachfolgend soll das Verfahren nach der Erfindung anhand von Ausführungsbeispielen und mit Hilfe von vier Figuren eingehend beschrieben werden.

Die Figuren zeigen:

Fig. 1 - den Aufbau der verwendeten Datenverarbeitungsanalge,

Fig. 2 - schematisch einen Überblick über die Erstellung der Sicherungsebene,

Fig. 3 - die Bewegung der Spitze eines Zuges in einem mit Doppelpunktgraphen dargestellten Gleisnetz,

Fig. 4 - eine schematische Darstellung der Systemreaktion bei Eintritt eines unkontrollierbaren, normalerweise nicht anzunehmenden Ereignisses.

In Fig. 1 ist der Aufbau der Datenverarbeitungsanlage mit deren Hilfe das Verfahren nach der Erfindung ausgeführt wird, dargestellt. Die Figur zeigt zwei Ebenen, die - getrennt voneinander - der Steuerung bzw. der Sicherung des die Außenanlage bildenden Gleisnetzes einschließlich der vorhandenen Weichen, Signale, Freimeldeeinrichtungen und der das Gleisnetz benutzenden Züge dienen. Die wesentlichen Funktionen der Sicherungsebene und der Steuerungsebene sind in der Figur wiedergegeben.

Fig. 2 gibt einen Überblick über die Erstellung der Daten für die Sicherungsebene. Wie aus der Figur ersichtlich, wird zunächst ein allgemeines Petri-Netz z.B. ein Predicate/Transition-Netz, das eine formale Spezifikation des Systemtyps enthält, aus den über das Netz verfügbaren Informationen erstellt und auf Konsistenz und richtige Syntax hin geprüft. Es wird dann, zusammen mit den speziellen Gleistopologiedaten, die z.B. in Form von Doppelpunktgra-

phen vorliegen können, mit Hilfe eines geeigneten Entfaltungswerkzeuges entfaltet, wodurch ein spezielles Petri-Netz, z.B. ein Condition/Event-Netz entsteht, das die Daten für die Sicherungsebene enthält und in dem als Laufzeitsystem ein Algorithmus (sog. eunomischer Algorithmus) anwendbar ist, der vor kontrollierbaren Ereignissen alle durch diese möglich werdenden Netzveränderungen auf ihre Zulässigkeit hin untersucht und bei zu erwartenden unzulässigen Ereignissen die Auslösung des kontrollierbaren Ereignisses verhindert.

Die anschließenden Ausführungsbeispiele beschreiben konkret die Erstellung der Sicherungsebene und die Anwendung des sogenannten revertierten eunomischen Algorithmus.

I. Erstellung der Sicherungsebene

1. Erstellen einer formalen Spezifikation des Systemtyps aus einer informalen.

Die Beschreibungssprache sind Predicate/Transition-Netze. Die Spezifikation enthält:

- Transitionen
- Prädikate
- Kanten
- abhängige Transitionen

Beispiel:

Bewegung einer Zugspitze über das Gleisnetz (allgemeine Spezifikation, topologieunabhängig) , gemäß Bild 3

Prädikate:

```
pred  head(_) :: critical : 0.
pred  occ(_) :: critical : 0.
pred  prof_occ(_) :: critical : 0.
```

Transitionen:

```
trans head_moves(P,Edg,Q) :: internal : is_step(P, _, Q, Edg).
```

Kanten:

```
edge  head_moves(P,_,_)   <-- head(P) : true.
edge  head_moves(_,_,Q)   --> head(Q) : true.
edge  head_moves(P,Edg,Q) --> occ(Edg)          : is_step(P, Pc, Q, Edg).
edge  head_moves(_,_,Q)   <== el_occ(El)        : el_of(Q,El).
```

Konstrukte der Spezifikationssprache

pred <predname>(<arglist>) :: <predtype> : <initial_value>

predname ist ein Prädikat, dessen Stelligkeit aus der Anzahl der Argumente in arglist hervorgeht.

Werte für predtype:

harmless: Wenn zur Systemlaufzeit versucht wird, eine bereits erfüllte Bedingung nochmals zu erfüllen, erzeugt dies lediglich ein Warning

critical: Wenn zur Systemlaufzeit versucht wird, eine bereits erfüllte Bedingung nochmals zu erfüllen, ist dies sicherheitskritisch.

initial_value: Initialisierungswert beim Systemstart

trans <transname>(<arglist>) :: <transtype> : transcondition.

transname ist eine Transition vom Typ transtype. transcondition ist ein logischer Ausdruck, für den beim Entfalten (siehe II-4.) alle Lösungen von Variablenbelegung gesucht werden, für die der Ausdruck wahr wird. Für jede solche Variablenbelegung gibt es dann ein Transitionsexemplar dieses Typs im speziell für eine Gleistopologie hergestellten Petri-Netz. Die logischen Ausdrücke werden mit der PROLOG-Syntax formuliert.

Werte für transtype:

internal: interne Transition, modelliert nicht direkt ein Ereignis der Außenanlage oder der Benutzerschnittstelle

outside: modelliert direkt ein in der Außenanlage vorkommendes Ereignis

command: modelliert direkt ein an der Benutzerschnittstelle angebotenes Ereignis

critcommand: modelliert direkt ein in der Benutzerschnittstelle angebotenes *zählpflichtiges* Ereignis

intolerable: modelliert ein unzulässiges Ereignis

unexpected: modelliert ein unzulässiges, normalerweise nicht angenommenes Ereignis

edge <trans>(<targlist>) <edgtype> <pred>(<parglist>) : <condition>.

Es wird eine Kante definiert, die vom Prädikat pred zur Transition trans führt oder umgekehrt. Im ersten Fall ist pred eine Vorbedingung, im zweiten eine Nach-

bedingung von `trans`. Bei der Entfaltung werden für jedes Transitionsexemplar von `trans` Lösungen für `condition` gesucht. Jede solche Lösung erzeugt dann ein Exemplar der Kante dieses Typs im speziell für eine Gleistopologie hergestellten Petri-Netz.
Werte für `edgtype`:

`<--:`	`pred` ist Vorbedingung von `trans`
`<==:`	`pred` ist Vorbedingung von `trans`, ändert seinen Zustand beim Eintreffen des Ereignisses aber nicht (Zustandsabfrage).
`-->:`	`pred` ist Nachbedingung von `trans`
`**>:`	Löschkante: `pred` wird beim Eintreffen von `trans` gelöscht
`++>:`	Notfallabhängigkeit zwischen `pred` und `trans`.

```
depend <transname1>(<arglist>) :::
            <transname1>(<arglist>) :: <othernet> :
            <condition>.
```

Paar von abhängigen Ereignissen, wobei die Zulässigkeit des Eintreffens von `transname1` davon abhängt, ob `transname2` im Petri-Netz `othernet` ebenfalls zulässig ist.

2. Syntax- und Konsistenzcheck

Es wird überprüft:

- Korrekte Syntax
- Entspricht <transtype> und <predtype> einem vordefinierten Typ?
- Sind die Typendefinitionen der Variablen konsistent?
- Sind <trans> und <pred> in allen Kantendefinitionen korrekt definiert?

3. Erstellung der Daten der Gleistopologie

Zur Darstellung der Gleistopologie werden Doppelpunktgraphen verwendet

4. Entfaltung

Für jeden Transitionstyp werden alle passende Variablenbelegungen aus der Objektmenge der Gleistopologie, so daß die Bedingung <transcondition> wahr wird. Jede solche Variablenbelegung ergibt ein Transitionsexemplar. Für jedes Transitionsexemplar werden alle Kanten gesucht, so daß die Kantenbedingungen mit der entsprechenden Variablenbelegung wahr sind.

5. Laufzeitsystem

Das Laufzeitsystem führt die speziellen, zur entsprechenden Gleistopologie gehörenden Petri-Netze aus. Es enthält eine Implementation des EUNOMISCHEN Algorithmus.

II. Beispiel einer formalen Spezifikation einer Sicherungsebene

```
/* ------------------------------------------------------------------------ */
/*  ELEMENTS                                                                */
/* ------------------------------------------------------------------------ */

pred  el_free(_):: harmless: 0.

pred  el_occ(_):: harmless : 0.

trans el_gets_occ(El)::  outside : is_el(El).
edge  el_gets_occ(El) <-- el_free(El) : true.
edge  el_gets_occ(El) --> el_occ(El)  : true.

trans el_gets_free(El):: outside : is_el(El).
edge  el_gets_free(El) <-- el_occ(El)  : true.
edge  el_gets_free(El) --> el_free(El) : true.


/* ------------------------------------------------------------------------ */
/*  SWITCHES                                                                */
/* ------------------------------------------------------------------------ */

pred  edg_open(_,_):: harmless : 0.
pred  edg_excl_open(_,_):: harmless : 0.
pred  edg_next_excl(_,_):: harmless : 0.
pred  not_lost_excl(_):: harmless : 0.

trans edg_to_excl(P,Edg2):: outside : is_switch(P,_,Edg2).

trans edg_from_excl(P,Edg2):: command : is_switch(P,_,Edg2).

trans edg_new_excl(P,Edg2):: command : is_switch(P,_,Edg2).

trans edg_lost_excl(P,Edg2):: outside : is_switch(P,_,Edg2).

pred  edg_lost_tmp(_,_):: harmless : 0.
trans edg_lost_excl_2(P,Edg2):: unexpected : is_switch(P,_,Edg2).

trans reset_lost(P,Edg2):: critcommand : is_switch(P,_,Edg2).

context switches :: is_switch(P,Edg1,Edg2) :::
[
             edge  edg_to_excl(P,Edg2) <-- edg_open(P,Edg1)       : true,
             edge  edg_to_excl(P,Edg2) <-- edg_next_excl(P,Edg2)  : true,
             edge  edg_to_excl(P,Edg2) --> edg_excl_open(P,Edg2)  : true,
```

```
                edge  edg_from_excl(P,Edg2) <== not_lost_excl(P)          : true,
                edge  edg_from_excl(P,Edg2) <-- edg_excl_open(P,Edg2)     : true,
                edge  edg_from_excl(P,Edg2) --> edg_open(P,Edg1)          : true,
                edge  edg_from_excl(P,Edg2) --> edg_next_excl(P,Edg1)     : true,
                edge  head_moves(Pc,Edg1,_) <== edg_open(P,Edg1): joined(P,Pc),

                edge  edg_lost_excl(P,Edg2) <-- edg_excl_open(P,Edg2)     : true,
                edge  edg_lost_excl(P,Edg2) --> edg_lost_tmp(P,Edg2)      : true,
                edge  edg_lost_excl_2(P,Edg2) <-- edg_lost_tmp(P,Edg2)  : true,
                edge  edg_lost_excl_2(P,Edg2) --> edg_open(P,Edg1)        : true,
                edge  edg_lost_excl_2(P,Edg2) --> edg_next_excl(P,Edg1)   : true,
                edge  edg_lost_excl_2(P,Edg2) **> not_lost_excl(P)        : true,


                edge  reset_lost(P,Edg1)    <== edg_next_excl(P,Edg1)          : true,
                edge  reset_lost(P,Edg1)    --> not_lost_excl(P)               : true,
                edge  edg_new_excl(P,Edg2) <== not_lost_excl(P)                : true,
                edge  edg_new_excl(P,Edg2) <-- edg_next_excl(P,Edg1)           : true,
                edge  edg_new_excl(P,Edg2) --> edg_next_excl(P,Edg2)           : true
        ].


        /* ------------------------------------------------------------------     */
        /*  SWITCH GUARDS                                                         */
        /* ------------------------------------------------------------------     */

        trans edg_crash(P, Edg1) :: intolerable : is_switch(P,Edg1,_).

        context switch_guard :: is_switch(P,Edg1,Edg2) :::
        [             edge  edg_crash(P, Edg1) <== occ(Edg1)     : true,
                      edge  edg_crash(P, Edg1) <== edg_open(P,Edg2) : true
        ].


        /* ------------------------------------------------------------------     */
        /*  TRAIN MOVEMENT                                                        */
        /* ------------------------------------------------------------------     */

        pred  head(_) :: critical : 0.
        pred  occ(_) :: critical : 0.
        pred  prof_occ(_) :: critical : 0.
        pred  tail(_) :: critical : 0.

        trans head_moves(P,Edg,Q) :: internal : is_step(P, _, Q, Edg).
        edge  head_moves(P,_,_)   <-- head(P) : true.
        edge  head_moves(_,_,Q)   --> head(Q) : true.
        edge  head_moves(P,Edg,Q) --> occ(Edg)                 : is_step(P, Pc, Q, Edg).
        edge  head_moves(_,Edg,_) --> prof_occ(Prof)           : prof_set(Edg,Prof).
        edge  head_moves(_,_,Q)   <== el_occ(El) : el_of(Q,El).

        trans tail_moves(Pc,Edg,Qc) :: internal : is_step(Qc, _, Pc, Edg).
        edge  tail_moves(Pc,_,_)   <-- tail(Pc) : true.
        edge  tail_moves(_,_,Qc)   --> tail(Qc) : true.
        edge  tail_moves(_,Edg,_)  <-- occ(Edg)   : true.
        edge  tail_moves(_,Edg,_)  <-- prof_occ(Prof): prof_set(Edg,Prof).
        edge  tail_moves(_,_,Qc)   <== el_occ(NextEl): el_of(Qc,NextEl).
        edge  tail_moves(Pc,_,_)   <== el_free(El)   : el_of(Pc,El).
```

```
trans tail_moves_back(Qc,Edg,Pc) :: unexpected : is_step(Qc, _, Pc, Edg).
edge  tail_moves_back(Qc,_,_)          <-- tail(Qc) : true.
edge  tail_moves_back(_,_,Pc)          --> head(Pc) : true.
edge  tail_moves_back(_,Edg,_)         --> occ(Edg)   : true.
edge  tail_moves_back(_,Edg,_)         --> prof_occ(Prof): prof_set(Edg,Prof).
edge  tail_moves_back(_,_,Pc)          <== el_occ(El): el_of(Pc,El).


/* ------------------------------------------------------------------   */
/*  CREATE A NEW TRAIN                                                  */
/* ------------------------------------------------------------------   */

pred  el_not_unexp(_):: harmless        : 0.

trans el_unexp_occ(El):: unexpected : is_el(El).

edge  el_gets_occ(El) --> el_not_unexp(El) : true.

edge  el_unexp_occ(El) <-- el_not_unexp(El) : true.
edge  el_unexp_occ(El) <== new_valid(P,#0) : (el_of(P,El), joined(P,Pc),
is_block(Pc)).

edge  el_unexp_occ(El) --> prof_occ(El) : true.
edge  el_unexp_occ(El) --> occ(Edg) : edg_of_el( El, Edg).
edge  el_unexp_occ(El) --> head(P) : el_of(P,El).
edge  head_moves(_,_,P) **> el_not_unexp(El) : el_of(P,El).
context createtrain :: ( el_of(P,El), joined(P,Pc), el_of(Pc,NEl) )
[ edge  el_unexp_occ(El) <== el_free(NEl): true
].

trans make_head(P):: command: (joined(P, _), el_of(P,_)).

edge make_head(P) <-- tail(P) : true.
edge make_head(P) --> head(P) : true.

trans make_tail(P):: critcommand: (joined(P, _), el_of(P,_)).

edge make_tail(P) <-- head(P) : true.
edge make_tail(P) --> tail(P) : true.

trans clear_head(P):: critcommand : end(P,_,_,_,_,_).
edge  clear_head(P) <-- head(P) : true.

trans clear_occ(Edg):: critcommand : edg(Edg,_,_,_).
edge  clear_occ(Edg) <-- occ(Edg) : true.

edge  tail_moves(Pc,_,Qc)   **> head_did_move_1(P,Q) : (joined(P,Pc),joined(Q,Qc)).


/* ------------------------------------------------------------------   */
/*  SET NEW SPEED VALID                                                 */
/* ------------------------------------------------------------------   */

pred  new_valid(_,_):: harmless : 0.
pred  unlocked(_):: harmless : 0.

trans set_new_valid(P,#LowSpeed,#Speed) :: command :
              ( exists_new_speed_valid(P,#LowSpeed),
                exists_new_speed_valid(P,#Speed),
                LowSpeed < Speed ).
```

```
depend set_new_valid(P,#LowSpeed,#Speed) :::
            set_new_valid(OtherP,#LowSpeed,#Speed) :: OtherNet :
            ( joined(P,Q), blo( Q, OtherP, _, OtherNet) ).

edge  set_new_valid(P,#LowSpeed,_)      <-- new_valid(P,#LowSpeed) : true.
edge  set_new_valid(P,_,_)              <== not_signal_reset(S) :
            ( is_main_sig(S,P); ( is_pre_sig(S,P),
              not is_main_sig(_,P) ) ).
edg  set_new_valid(P,#0,_)              <== new_valid(Pc,#0) :
            ( joined(P,Pc), ( is_block(P); is_block(Pc)) ).
edge  set_new_valid(P,_,#Speed)         --> new_valid(P,#Speed) : true.
edge  set_new_valid(P,#0,_) --> unlocked(P) : is_main_sig(_,P).
edge  set_new_valid(P,#0,_) --> head(P) : is_block(P).
edge  set_new_valid(P,#0,_) --> tail(Q) : (is_block(P), joined(P,Q)).


/* ------------------------------------------------------------------- */
/*  SET NEW SPEED DEST                                                 */
/* ------------------------------------------------------------------- */

pred  new_dest(_,_):: harmless : 0.

trans set_new_dest(P,#LowSpeed,#Speed) :: command :
            ( exists_new_speed_dest(P,#LowSpeed),
              exists_new_speed_dest(P,#Speed),
              LowSpeed < Speed ).

depend set_new_dest(P,#LowSpeed,#Speed) :::
            set_new_dest(OtherP,#LowSpeed,#Speed) :: OtherNet :
            ( joined(P,Q), blo( Q, OtherP, _, OtherNet) ).

edge  set_new_dest(P,#LowSpeed,_)       <-- new_dest(P,#LowSpeed) : true.
edge  set_new_dest(P,_,_)               <== not_signal_reset(S) :
                                        (is_main_sig(S,P);
                                        (is_pre_sig(S,P),not is_main_sig(_,P))).
edge  set_new_dest(P,_,#Speed)          -> new_dest(P,#Speed) : true.

edge  set_new_valid(P,#0,_) --> new_dest(P,#0) : exists_new_speed_dest(P,#0).


/* ------------------------------------------------------------------- */
/*  RESET NEW SPEED VALID AND DEST                                     */
/* ------------------------------------------------------------------- */

trans reset_new_speed(P) :: command :
            (exists_new_speed_valid(P,#0);
            (exists_new_speed_dest(P,#0), not exists_new_speed_valid(P,#0) )).

depend reset_new_speed(P) ::: reset_new_speed(OtherPJoined) :: OtherNet :
                              blo( P, _, OtherPJoined, OtherNet ).

edge  reset_new_speed(P)    <== not_train_did_enter(P): is_block(P).
edge  reset_new_speed(P)    --> new_valid(P,#0) : exists_new_speed_valid(P,#0).
edge  reset_new_speed(P)    **> new_valid(P,#SomeSpeed) :
            ( exists_new_speed_valid(P,#SomeSpeed), SomeSpeed>0 ).
edge  reset_new_speed(P)    --> new_dest(P,#0) : ( exists_new_speed_dest(P,#0),
                                    not exists_new_speed_valid(P,#_)).
edge  reset_new_speed(P)    **> new_dest(P,#SomeSpeed) :
                            (exists_new_speed_dest(P,#SomeSpeed),
```

```
                              ( SomeSpeed>0;
                              ( SomeSpeed = 0,
                              exists_new_speed_valid(P,#_))) ).

edge  reset_new_speed(P)    --> tail(Pc)  : (not is_block(P), joined(P,Pc)).
edge  reset_new_speed(P)    **> head(P)   : is_block(P).
edge  reset_new_speed(P)    **> tail(Pc)  : joined(P,Pc).
edge  reset_new_speed(P)    --> not_signal_reset(S) :( is_main_sig(S,P);
                              ( is_pre_sig(S,P), not is_main_sig(_,P) ) ).

edge  reset_new_speed(P)    **> train_did_exit(P): (joined(P,Pc),is_block(Pc)).

edge  head_moves(P,_,_)     <== unlocked(P) : is_main_sig(_,P).


trans lock(P) :: internal :
                              is_main_sig(_,P).

edge  lock(P) <-- unlocked(P) : true.
edge  lock(P) <== new_valid(P,#0) : true.
pred  not_signal_reset(S):: harmless : 1.

trans reset_signal(S) :: command :
              ( is_main_sig(S,_); ( is_pre_sig(S,P), not is_main_sig(_,P) ) ).

context signal :: ( is_main_sig(S,P); (is_pre_sig(S,P), not is_main_sig(_,P)) ) :::
[ edge  reset_signal(S)     <-- not_signal_reset(S) : true,
  edge  reset_signal(S)     ++> head(P) : true,
  edge  reset_signal(S)     **> new_valid(P,#0): exists_new_speed_valid(P,#0),
  edge  reset_signal(S)     **> new_dest(P,#0) : ( exists_new_speed_dest(P,#0),
                              not exists_new_speed_valid(P,#_))
].


/* ------------------------------------------------------------------------    */
/*  RESET BLOCK                                                               */
/* ------------------------------------------------------------------------    */

trans exit_system(P) :: command : is_block(P).
edge  exit_system(P)   <-- tail(P)          : true.
edge  exit_system(P)   <-- head(Pc)         : joined(P,Pc).

depend exit_system(P) :::clear_enter_system(OtherP) :: OtherNet :
                              blo( P, OtherP, _, OtherNet ).
pred  train_did_enter(_):: harmless : 0.
pred  not_train_did_enter(_):: harmless : 1.
pred  train_did_exit(_):: critical : 0.

edge  head_moves(P,_,Q) --> train_did_exit(Q) : (joined(Q,Qc),is_block(Qc)).

trans enter_system(P) :: internal                       : is_block(P).
edge  enter_system(P)   <-- not_train_did_enter(P)      : true.
edge  enter_system(P)   <== tail(Pc)      : joined(P,Pc).
edge  enter_system(P)   <== el_occ(El)    : ( joined(P,Pc), el_of(Pc,El)).
edge  enter_system(P)   <== new_valid(Pc,#0)            : joined(P,Pc).
edge  enter_system(P)   --> train_did_enter(P)          : true.

trans clear_enter_system(P) :: command : is_block(P).
edge  clear_enter_system(P)   <== el_free(El): ( joined(P,Pc), el_of(Pc,El)).
edge  clear_enter_system(P)   <-- train_did_enter(P)                        : true.
```

```
edge  clear_enter_system(P)    --> not_train_did_enter(P)                    : true.

/* ------------------------------------------------------------------------   */
/*  PROPAGATE AND CONTROL SPEED VALID                                         */
/* ------------------------------------------------------------------------   */

pred  head_did_move_1(_,_):: harmless : 0.
pred  speed_valid(_,_):: harmless : 0.

edge  head_moves(P,_,Q)                  --> head_did_move_1(P,Q)                 :
true.

trans next_speed_valid(P,Q,#Speed) :: internal :
              ( is_step(P,_,Q,_),
                ( ( exists_new_speed_valid(P,#Speed), Speed > 0 ) ;
                  ( not exists_new_speed_valid(P,#_),exists_speed_valid(P,#Speed))
)
              ).

edge  next_speed_valid(P,Q,_)            <-- head_did_move_1(P,Q): true.
edge  next_speed_valid(_,Q,#Speed)       --> speed_valid(Q,#Speed): true.
edge  next_speed_valid(P,_,#OldSpeed)    <-- speed_valid(P,#OldSpeed):
            not exists_new_speed_valid(P,#_).
edge  next_speed_valid(P,_,#Speed)       <== new_valid(P,#Speed):
            exists_new_speed_valid(P,#Speed).
edge  next_speed_valid(P,_,#_)           **> speed_valid(P,#OldSpeed) :
                                         ( not is_block(P),
                                         exists_speed_valid(P,#OldSpeed) ).
edge  reset_new_speed(P)                 **> speed_valid(P,#OldSpeed) :
                                         ( joined(P,Q), is_block(Q),
                                           exists_speed_valid(P,#OldSpeed) ).

trans too_fast(Q,Edg,#Speed,#MaxSpeed) :: intolerable :
              ( is_step(P,_,Q,Edg),
                exists_speed_valid(P,#Speed),
                max_speed_valid(Edg,#MaxSpeed),
                MaxSpeed<Speed).

edge  too_fast(Q,_,#Speed,_) <== speed_valid(Q,#Speed): true.
edge  too_fast(Q,Edg,_,_)    <== head_did_move_1(P,Q)   : is_step(P,_,Q,Edg).
trans too_fast_exit(Pc,#Speed,#NewSpeed) :: intolerable :
              ( is_block(P), joined(P,Pc),
                exists_speed_valid(Pc,#Speed),
                exists_new_speed_valid(Pc,#NewSpeed),
                NewSpeed<Speed).

edge  too_fast_exit(Pc,#Speed,_)         <== speed_valid(Pc,#Speed) : true.
edge  too_fast_exit(Pc,_,#NewSpeed)      <== new_valid(Pc,#NewSpeed) : true.
edge  too_fast_exit(Pc,_,#_)             <== head(Pc) : (joined(Pc,P), is_block(P))

/* ------------------------------------------------------------------------   */
/*  PROPAGATE AND CONTROL SPEED DEST                                          */
/* ------------------------------------------------------------------------   */
```

```
pred  head_did_move_2(_,_):: harmless : 0.
pred  speed_dest(_,_,_):: harmless : 0.

edge  head_moves(P,_,Q) --> head_did_move_2(P,Q): true.

trans next_speed_dest(P,Q,#Dest,#Previous,#This) :: internal :
              exists_speed_break(P, Q, #Previous, #This, #Dest).
edge  next_speed_dest(P,Q,_,_,_)<-- head_did_move_2(P,Q): true
edge  next_speed_dest(P,_,#Dest,#OldBreak,_) <-- speed_dest(P,#Dest,#OldBreak)    :
              (not exists_new_speed_valid(P,#_), not exists_new_speed_dest(P,#_)).

edge  next_speed_dest(P,_,#_,#OldBreak,_)  **> speed_dest(P,#O_Dest,#O_Break)     :
              (once((exists_new_speed_valid(P,#_); exists_new_speed_dest(P,#_))),
               not is_block(P), exists_speed_break_s(_, P, #O_Break, #O_Dest),
               O_Break =< OldBreak)
edge  next_speed_dest(P,_,#Dest,_,_)     <== new_dest(P,#Dest):
                           exists_new_speed_dest(P,#Dest).
edge  next_speed_dest(P,_,#Dest,_,_)     <== new_valid(P,#Dest):
                           (exists_new_speed_valid(P,#Dest),
                           not exists_new_speed_dest(P,#_)).

edge  next_speed_dest(P,_,_,#OldBreak,_) <== speed_valid(P,#OldBreak):
                           (not exists_new_speed_valid(P,#_),
                           exists_new_speed_dest(P,#0)).
edge  next_speed_dest(P,_,_,#OldBreak,_)<== new_valid(P,#OldBreak):
                           exists_new_speed_valid(P,#0).

edge  next_speed_dest(_,Q,#Dest,_,#NewBreak) --> speed_dest(Q,#Dest,#NewBreak)    :
                                    true.

edge  reset_new_speed(P)    **> speed_dest(P,#O_Dest,#O_Break) :
                           ( joined(P,Q), is_block(Q),
                             exists_speed_break_s(_, P, #O_Break, #O_Dest) ).

trans too_fast_dest(Q,#SpeedBreak,#SpeedValid,#Dest) :: intolerable :
              ( exists_new_speed_valid(Q,#SpeedValid),
                exists_speed_break_s(_, Q, #SpeedBreak, #Dest),
                SpeedBreak>SpeedValid).

edge  too_fast_dest(Q,#SpeedBreak,#_,#Dest) <== speed_dest(Q,#Dest,#SpeedBreak) :
true.
edge  too_fast_dest(Q,#_,#SpeedValid,_) <== new_valid(Q,#SpeedValid): true.
edge  too_fast_dest(Pc,#_,#_,#_)         <== head(Pc) : (joined(Pc,P),
is_block(P)).

trans too_fast_dest2(Q,#SpeedDest,#SpeedValid) :: intolerable :
              ( exists_new_speed_dest(Q,#SpeedDest),
                ( ( not exists_new_speed_valid(Q,#_),  exists_speed_valid(Q,
                           #SpeedValid));
                  ( exists_new_speed_valid(Q,#SpeedValid)), SpeedValid > 0 ),
                           SpeedDest>SpeedValid).

edge  too_fast_dest2(Q,#SpeedDest,#_)  <== new_dest(Q,#SpeedDest) :
                           exists_new_speed_dest(Q,#SpeedDest).

edge  too_fast_dest2(Q,#_,#SpeedValid) <== speed_valid(Q,#SpeedValid):
                           not exists_new_speed_valid(Q,#_).
edge  too_fast_dest2(Q,#_,#SpeedValid) <== new_valid(Q,#SpeedValid):
```

```
                              exists_new_speed_valid(Q,#SpeedValid).

trans too_fast_exit_dest(Pc,#Speed,#NewSpeed,#Dummy) :: intolerable :
              ( is_block(P), joined(P,Pc),
                exists_speed_break_s(_, Pc, #Dummy, #Speed),
                exists_new_speed_dest(Pc,#NewSpeed),
                NewSpeed<Speed).
edge  too_fast_exit_dest(Pc,#Speed,_,#Dummy)<== speed_dest(Pc,#Speed,#Dummy)
                                        :true.
edge  too_fast_exit_dest(Pc,_,#NewSpeed,_)<== new_dest(Pc,#NewSpeed) : true.


/* ------------------------------------------------------------------    */
/*  PREVENT TAILS TO SEE GREEN SIGNALS                                   */
/* ------------------------------------------------------------------    */

pred  propag_green(_) :: harmless : 0.

trans propag_green_back(P,Q) :: internal : (exists_tail_sees_green(P),
                                        is_step(Q, _, P, _), not
is_main_sig(_,Q)).
edge  propag_green_back(P,Q) <== unlocked(P) : is_main_sig(_,P).
edge  propag_green_back(P,Q) <== propag_green(P) : not is_main_sig(_,P).
edge  propag_green_back(P,Q) <== edg_open(Qc,Edg): (joined(Q,Qc),
                          is_step(Q, _, P, Edg), is_switch(Qc,Edg,_)).
edge  propag_green_back(P,Q)             --> propag_green(Q) : true.
edge  propag_green_back(P,Q)             **> propag_green(Q) : true.

trans tail_sees_green(Q) :: intolerable : exists_tail_sees_green(Q).
edge  tail_sees_green(Q)   <== tail(Q) : true.
edge  tail_sees_green(Q)   <== unlocked(Q) : is_main_sig(_,Q).
edge  tail_sees_green(Q)   <== propag_green(Q) : not is_main_sig(_,Q).
```

III. Revertierter EUNOMISCHER Algorithmus

Algorithmus, der beim Eintreffen eines nicht anzunehmenden unkontrollierbaren Ereignisses gestartet wird und in der Menge der kontrollierbaren Ereignisse nach einer Notfallabhängigkeit sucht, die das Eintreffen von unzulässigen Folgeereignissen verhindert. Der Algorithmus ist in Objective-C formuliert.

```
+ searchForEmergency
{

      static List *expandB= nil;
      CENode *nodeq;
      CEPlace *eb;
      List *list;
      id net;
      int q;
```

/ Es wurde vorher mit dem EUNOMISCHEN Algorithmus alle unzulässigen Ereignisse in der Liste* `intolerableList` *festgehalten.*

*Initialisierung: */*

```
net= [CENet netWithIndex:[CENet netIndexOf:
     ELEM(intolerableList,0)]];
if (!expandB) { expandB= [[List alloc] initCount:100];}
list= (List*)[net places];
for (q= 0; q<COUNT(list); q++)
{ eb=ELEM(list,q);
  eb->reachedF=FALSE; }
list= (List*)[net trans];


for (q= 0; q<COUNT(list); q++)
{ eb=ELEM(list,q);
  eb->reachedF=FALSE; }
[expandB empty];
for (q=COUNT(intolerableList)-1; q>=0; q--)
{ nodeq= ELEM(intolerableList,q);
  [expandB addObject:nodeq];
  nodeq->reachedF= TRUE;
}
```

/ Expandiere die Knoten des Petri-Netz-Graphen der Menge* `intolerableList` *rückwärts */*

```
while ( COUNT(expandB) > 0 )
{
  eb= [expandB removeObjectAt:0];
```

*/** `eb` *ist eine Bedingung: wenn sie in Notfallabhängigkeit zu einem Ereignis steht: laß das Ereignis eintreffen, sonst expandiere weiter rückwärts. */*

```
  if ([eb isKindOf:[CEPlace class]]
     &&[eb performEmergency])
  {  }
  else
  { for ( q=eb->mark[E_FIX_PRE+1]-1;
     q>=eb->mark[E_DYN_PRE]; q-- )
    { nodeq= ELEM(eb,q);
      if ( nodeq->reachedF == FALSE )
      { nodeq->reachedF= TRUE;
       if (nodeq->derivable == TRUE )
       { [expandB addObject:nodeq]; }
      }
    }
  }
}
return self;
```

Beispiel:

In dem in Bild 4 dargestellten Beispiel verliert die Weiche ihre definierte Endlage (unkontrollierbares, aber norma-

lerweise nicht anzunehmendes Ereignis). Dadurch droht ein Zustand, in dem die Zugspitze des herannahenden Zugs sich auf einer Weiche befindet, die keine definierte Endlage besitzt (edg_crash). Um dies zu verhindern, wird der obige Algorithmus gestartet. Er untersucht die Vorgängermenge des Ereignisses "edg_crash" und findet die Notfallabhängigkeit "reset_signal--head". Anschließend wird das Ereignis "reset_signal ausgelöst, das das Hauptsignal auf Halt wirft.

Patentansprüche

1. Verfahren zur Steuerung und Sicherung des Betriebes eines beliebig gestalteten spurgeführten Transportsystems mit Hilfe einer Datenverarbeitungsanlage, in der ein Modell aller im Transportsystem vorkommender Zustände und Ereignisse in einer formalen Sprache gespeichert ist, derart, daß sich aufgrund eintreffender Ereignisse ändernde (dynamische) Daten, sofern sie gleisnetzunabhängig einem Gleiselementtyp zugeordnet werden können, in Form von Petri-Netzen hoher Abstraktionsstufe, sofern sie im Gleisnetz konkret vorhandenen Gleiselementen zugeordnet werden können, in Form von aus diesen Petri-Netzen abgeleiteten Netzen, und für jeden Systemzustand geltende (statische) Daten, sofern sie gleisnetzunabhängig einem Gleiselementtyp zugeordnet werden können, in Form von Prädikaten oder Invarianten, sofern sie im Gleisnetz konkret vorhandenen Gleiselementen zugeordnet werden können, in Form von Topographieelementdaten repräsentiert sind, und in der ein Algorithmus zur Verfügung steht, der vor Auslösung eines kontrollierbaren Ereignisses aus dem aktuellen Zustand des Systems alle möglichen Folgezustände berechnet und die Auslösung verhindert, wenn unter den berechneten Folgezuständen ein oder mehrere unzulässige Zustände gefunden werden, **dadurch gekennzeichnet,** daß ein unkontrollierbares Ereignis, mit dessen Eintreffen nicht von vornherein gerechnet werden kann, als normalerweise nicht anzunehmendes Ereignis definiert wird, und daß während der Prüfung der Zulässigkeit eines kontrollierbaren Ereignisses nicht angenommen wird, daß ein normalerweise nicht anzunehmendes Ereignis eintrifft, daß ein zusätzlicher Algorithmus gestartet wird, wenn nach Feststellung der Zulässigkeit des kontrollierbaren Ereignisses ein normalerweise nicht anzunehmendes Ereignis eintritt, daß mittels des zusätzlichen Algorithmus, ausgehend von dem durch das Eintreten dieses Ereignisses erzeugten aktuellen Systemzustand, die Menge sämtlicher durch Folgen von unkontrollierbaren Ereignissen erreichbarer Zustände dahingehend untersucht wird, ob sich ein als unzulässig bezeichneter Zustand darunter befindet oder die Vorbedingungen für das Eintreffen eines als unzulässig bezeichneten Ereignisses erfüllt sind, und daß, wenn dies zutrifft, die Menge aller oder für diesen Fall besonders bezeichneter kontrollierbarer Ereignisse daraufhin untersucht wird, ob durch ihr Auslösen der Systemzustand so verändert werden kann, daß die Menge der zuvor berechneten, erreichbaren unzulässigen Zustände oder als unzulässig bezeichneten Ereignisse verkleinert wird, und daß, wenn dies der Fall ist, die entsprechenden kontrollierbaren Ereignisse ausgelöst werden.

2. Verfahren nach Anspruch 1, dadurch gekennzeichnet, daß den den im Gleisnetz konkret vorhandenen Gleiselementen zugeordneten Topographieelementdaten Geschwindigkeitsdaten in Form von Datentupeln zugeordnet werden, die neben gültigen Maximalgeschwindigkeiten auch Daten enthalten, aus denen eine bei maximalem Bremseinsatz erreichbare Verzögerung entnehmbar ist, daß die an einem Gleiselement gültigen Geschwindigkeiten, sofern sie nicht neu vorgegeben werden, rekursiv oder iterativ aus Daten berechnet werden, die dem in der jeweiligen Fahrtrichtung vorher zu befahrenden Gleiselement zugeordnet sind, und daß die berechneten Geschwindigkeitsdaten für die Festlegung der im System geltenden Maximalgeschwindigkeiten verwendet werden.

3. Verfahren nach Anspruch 2, dadurch gekennzeichnet, daß die den Gleiselementen zugeordneten Topographieelementdaten die Form von Doppelpunktgraphen besitzen und die die Geschwindigkeitsdaten enthaltenden Datentupel den Punkten oder Kanten dieser Doppelpunktgraphen zugeordnet sind.

4. Verfahren nach einem der vorstehenden Ansprüche, dadurch gekennzeichnet, daß Gleisabschnitte des Gleisnetzes mit Gleisfreimeldeeinrichtungen ausgestattet sind, daß eine zusammenhängende Folge belegter Abschnitte als sich im Gleisnetz fortbewegende Transporteinheit interpretiert und aus der Reihenfolge der neuen Belegungen oder Freimeldungen die Fahrtrichtung geschlossen wird und daß eine einzelne fehlerhafte Freimeldung die angenommene Ausdehnung der Transporteinheit nicht verändert.

5. Verfahren nach Anspruch 4, dadurch gekennzeichnet, daß eine einzelne Belegung eines Abschnittes bei gültiger Freimeldung aller benachbarten Abschnitte als neue Transporteinheit mit zunächst zwei Spitzen und keinem Schluß interpretiert wird, daß die Belegung eines in Fahrtrichtung gesehen hinter dem Schluß einer vorhandenen Transporteinheit liegenden Abschnittes als Änderung der Fahrtrichtung der Transporteinheit oder als sich nach rückwärts in Bewegung setzender, abgetrennter Teil der Transporteinheit interpretiert wird und daß in beiden Fällen der ursprüngliche Schluß der Transporteinheit im Systemmodell in eine Spitze umgewandelt wird.

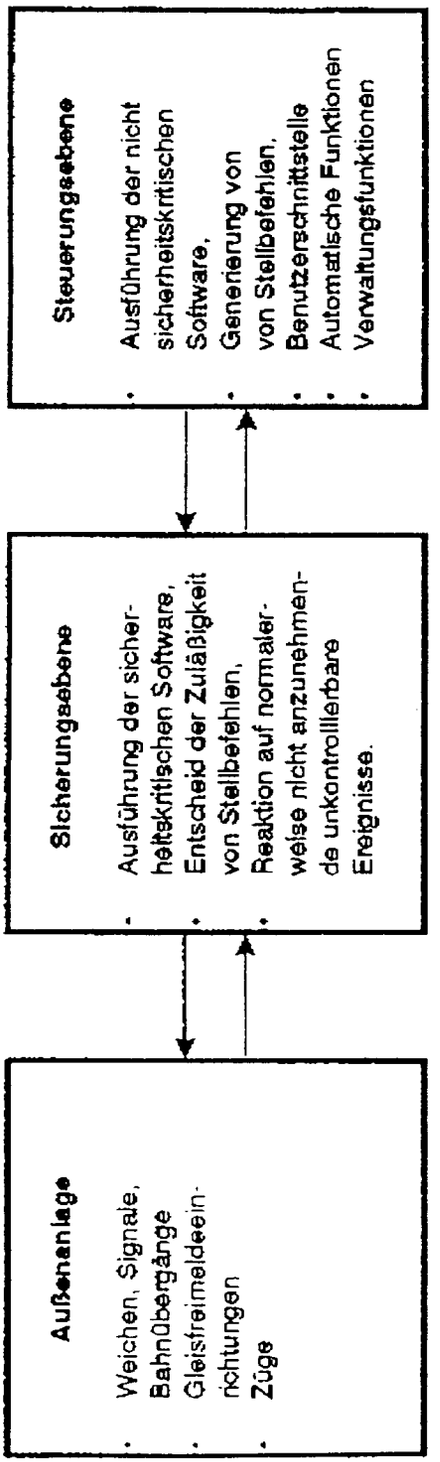

Steuerungsebene
Ausführung der nicht sicherheitskritischen Software,
Generierung von von Stellbefehlen,
Benutzerschnittstelle
Automatische Funktionen
Verwaltungsfunktionen
Sicherungsebene
Ausführung der sicherheitskritischen Software,
Entscheid der Zulässigkeit von Stellbefehlen,
Reaktion auf normalerweise nicht anzunehmende unkontrollierbare Ereignisse.
Außenanlage
Weichen, Signale, Bahnübergänge
Gleisfreimeldeeinrichtungen
Züge

Fig 1

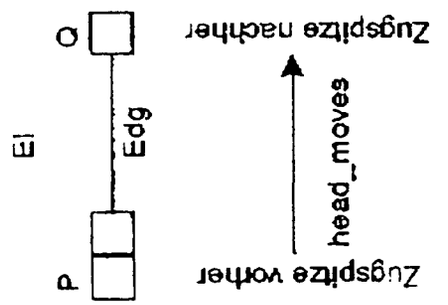

Zugspitze vorher
head_moves
Zugspitze nachher
P
Edg
O
EI

Fig 3

Fig 2

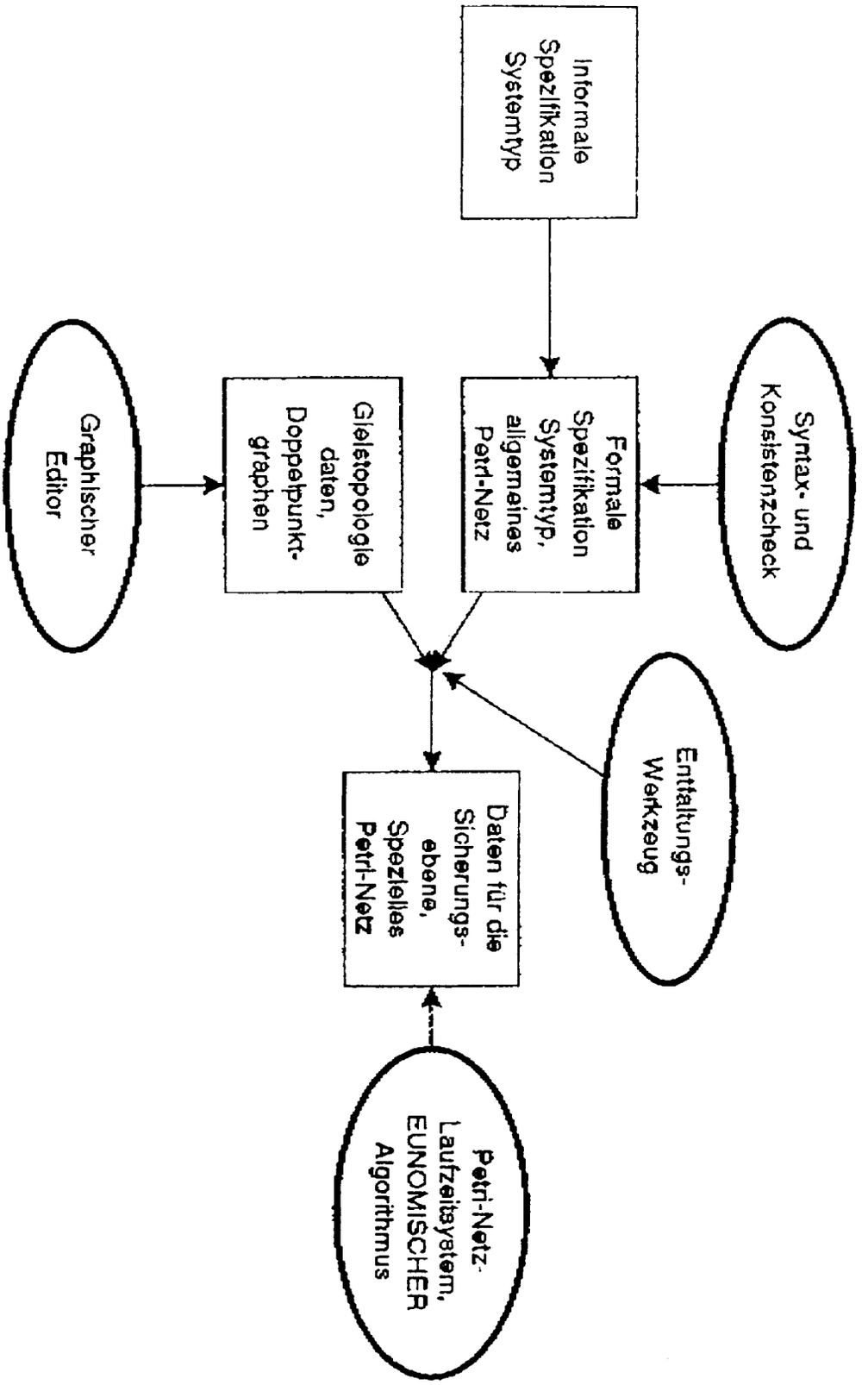
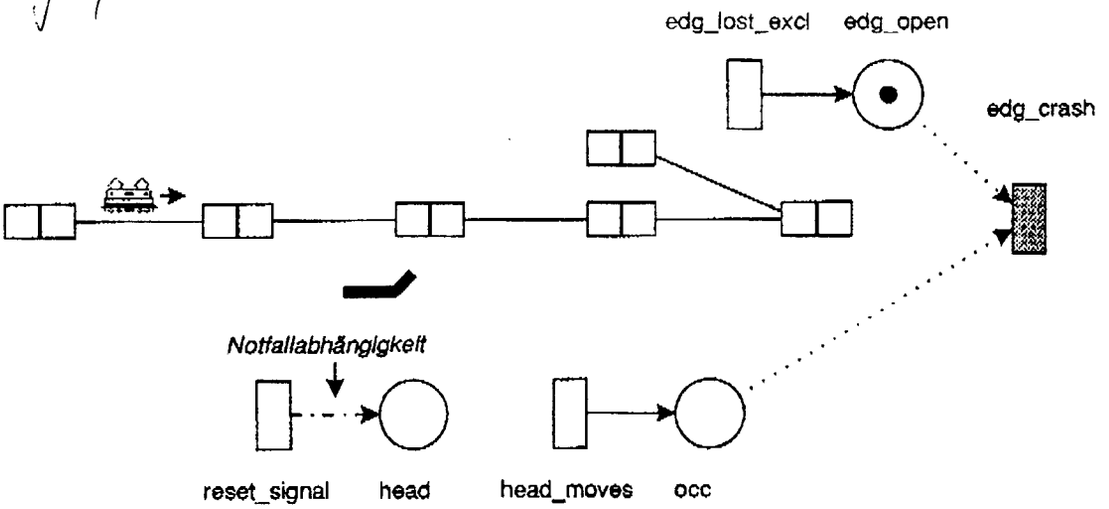
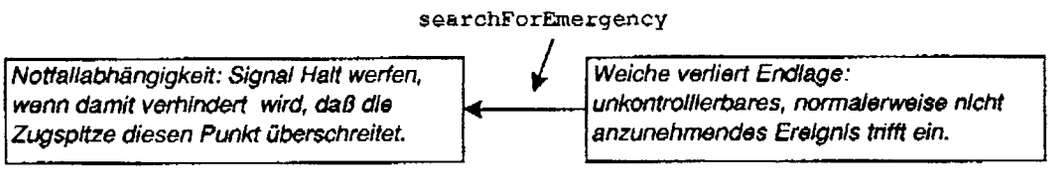

Informale Spezifikation Systemtyp
Formale Spezifikation Systemtyp, allgemeines Petri-Netz
Syntax- und Konsistenzcheck
Graphischer Editor
Gleistopologie daten, Doppelpunkt-graphen
Entfaltungs-Werkzeug
Daten für die Sicherungs-ebene, Spezielles Petri-Netz
Petri-Netz-Laufzeitsystem, EUNOMISCHER Algorithmus

Fig 4





Symbole:

- Punkt des Doppelpunktgraphen, der die Gleistopologie repräsentiert
- Kante des Doppelpunktgraphen, der die Gleistopologie repräsentiert
- Herannahender Zug
- Fahrtstellung zeigendes Hauptsignal
- Bedingung des Petri-Netzes
- Ereignis des Petri-Netzes
- Unzulässiges Ereignis des Petri-Netzes
- Kante markiert Notfallabhängigkeit
- Zustandsabfragekante

Europäisches Patentamt

EUROPÄISCHER RECHERCHENBERICHT

Nummer der Anmeldung
EP 96 10 4501

EINSCHLÄGIGE DOKUMENTE

Kategorie	Kennzeichnung des Dokuments mit Angabe, soweit erforderlich, der maßgeblichen Teile	Betrifft Anspruch	KLASSIFIKATION DER ANMELDUNG (Int.Cl.6)
A	SIGNAL + DRAHT, Bd. 85, Nr. 5, 1.Mai 1993, HAMBURG DE, Seiten 166-169, XP000413530 DANNENBERG H: "ANWENDUNG VON SIMULATIONSMODELLEN FUR FLANUNGS- UND STEUERUNGSPROBLEME IM EISENBAHNBETRIEB" * das ganze Dokument * -----	1	B61L21/00

RECHERCHIERTE SACHGEBIETE (Int.Cl.6)

B61L

Der vorliegende Recherchenbericht wurde für alle Patentansprüche erstellt

Rechercheort	Abschlußdatum der Recherche	Prüfer
DEN HAAG	28.August 1996	Wanzeele, R

KATEGORIE DER GENANNTEN DOKUMENTE

X : von besonderer Bedeutung allein betrachtet
Y : von besonderer Bedeutung in Verbindung mit einer anderen Veröffentlichung derselben Kategorie
A : technologischer Hintergrund
O : nichtschriftliche Offenbarung
P : Zwischenliteratur

T : der Erfindung zugrunde liegende Theorien oder Grundsätze
E : älteres Patentdokument, das jedoch erst am oder nach dem Anmeldedatum veröffentlicht worden ist
D : in der Anmeldung angeführtes Dokument
L : aus andern Gründen angeführtes Dokument

& : Mitglied der gleichen Patentfamilie, übereinstimmendes Dokument

EPO FORM 1503 03.82 (P04C03)