Eine anwenderorientierte Sicherheitsfachsprache zur Verifikation von Steuerungsprogrammen

Thomas Mertke
Institut für Produktionsforschung
Brandenburgische Technische Universität Cottbus
Postfach 10 13 44
03013 Cottbus
Tel.: +49 (0)355 692396
Fax.: +49 (0)355 692387
E-Mail: mertke@aut.tu-cottbus.de

Peter Deussen, Monika Heiner
Institut für Informatik
Brandenburgische Technische Universität Cottbus
Postfach 10 13 44
03013 Cottbus
Tel.: +49 (0)355 693885
Fax.: +49 (0)355 693830
E-Mail: {pd, mh}@informatik.tu-cottbus.de

1 Einleitung

Bei der heutigen kundenorientierten Fertigung von Maschinen und Anlagen stellt die Erzeugung der zugehörigen Steuerungssoftware einen wesentlichen Zeit- und Kostenfaktor dar. Zudem lassen sich Unfälle und Katastrophen der jüngeren und älteren Geschichte (Beispiele: Airbus-Unglücke, Ariane-5-Erststart, Mars-Mission) oftmals auf Unzulänglichkeiten der Steuerungssoftware zurückführen. Die zunehmende Komplexität moderner Software und das steigende Sicherheitsbedürfnis der Endanwender erfordern neue Methoden und Herangehensweisen, mit denen sich die Korrektheit und Sicherheit von Steuerungsprogrammen nicht nur überprüfen, sondern auch beweisen lässt. Heute benutzte Verfahren beziehen sich hauptsächlich auf einen mehr oder weniger gründlichen Test der Software, der jedoch niemals vollständig sein kann.
Die Erstellung eines Steuerungsprogramms für eine zu automatisierende Anlage kann man als einen Entwicklungsprozess betrachten, bei dem festgelegte und gewünschte Verhaltenseigenschaften als Software implementiert werden. Die erzeugten Programme weisen jedoch meist nicht nur diese Eigenschaften auf (falls überhaupt). Es kommt häufig vor, dass die Programme zusätzliche „versteckte“ Eigenschaften enthalten (Abb. 1). Diese wurden nicht spezifiziert und vom Programmierer auch nicht bewusst implementiert, sind jedoch trotzdem vorhanden. Während die meisten dieser Eigenschaften harmlos sein können, gibt es unter Umständen aber auch solche, die

während des Betriebes zu riskanten und gefährlichen Reaktionen des Programms führen können. Somit entstehen bei der Erstellung von Software zwei kritische Bereiche, die in der Abbildung 1 schraffiert hervorgehoben wurden. Sie entstehen, wenn einerseits die Vorgaben des Auftraggebers nicht vollständig umgesetzt wurden und wenn andererseits (durch unvollständige Spezifikation) unbeabsichtigt kritische Systemeigenschaften in der Software implementiert wurden.

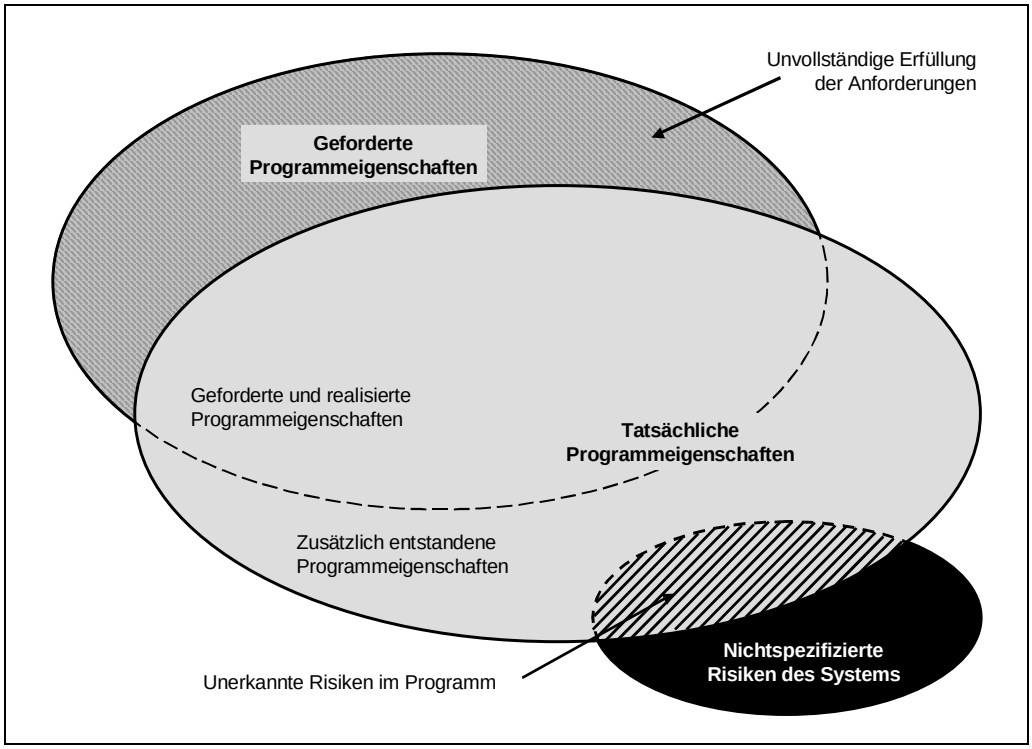

Abbildung 1: Eigenschaften von Steuerungsprogrammen

Ziel anschließender Test- und Verifikationsverfahren ist es, diese Unzulänglichkeiten zu beseitigen. Die Überprüfung der Eigenschaften wird i. Allg. durch verschiedene Tests durchgeführt, die vor allem die Inspektion der geforderten Solleigenschaften des Programms sowie die Kontrolle der Programmreaktionen auf zuvor festgelegte Fehlerszenarien beinhalten. Aufgrund der Komplexität industrieller Programme müssen Tests aber immer als unvollständig betrachtet werden. Aus diesem Grund sind Verfahren zum automatischen und vollständigen Nachweis der geforderten Eigenschaften notwendig. Ein wesentlicher Problempunkt hierbei besteht in der formalen Darstellung der geforderten Programmeigenschaften.

Dieser Beitrag stellt eine Spezifikationssprache vor, mit der die Anforderungen an ein Steuerungsprogramm einerseits ausreichend formal für eine Verifikation, andererseits einfach genug für eine leichte Benutzbarkeit, erstellt werden können. Diese Sicherheitsfachsprache wurde im Zusammenhang mit einem Forschungsprojekt entwickelt, dass sich mit Methoden zum vollständigen Nachweis der Eigenschaften eines Steuerungsprogramms beschäftigt. Diese Methoden basieren auf formalen Grundlagen, sind jedoch mit anwenderfreundlichen Benutzerschnittstellen ausgerüstet, so dass sie eine hohe Benutzerakzeptanz finden können. Mit dem Tool-Set „Safety-Knight“ ist es möglich, die Funktions- und Sicherheitsanforderungen an ein Steuerungsprogramm

vollständig zu überprüfen. Die dazu notwendigen Verfahren sind teilweise bereits in der Informatik etabliert und wurden innerhalb des Projektes an die spezifischen Anforderungen der Automatisierungstechnik angepasst. Andere Teilkomponenten wurden neu geschaffen. Ausführlichere Informationen zu diesem DFG-geförderten Forschungsprojekt sind [HMMM1997, MeiMe1998; MerMe2000] zu entnehmen, Schwerpunkt dieses Papiers ist die Vorstellung einer anwenderorientierten Sicherheitsfachsprache (SFS) zur Spezifikation von Eigenschaften von Steuerungsprogrammen.
Der Beitrag ist folgendermaßen gegliedert: der Abschnitt 2 gibt einen kurzen Überblick über das Projekt, der Abschnitt 3 erläutert die notwendigen Randbedingungen einer formalen Beschreibung der Programmeigenschaften. Der Abschnitt 4 geht dann ausführlich auf die entwickelte Sicherheitsfachsprache ein, es folgen eine kurze Beschreibung der rechentechnischen Realisierung sowie eine Zusammenfassung.

2 Model-Checking von Automatisierungssystemen

Zur qualitativen Analyse von Systemen wird in der Informatik seit längerer Zeit das Model-Checking angewendet. Man versteht darunter ein Verfahren, das überprüft, ob ein betrachtetes Systems eine gegebene Spezifikation erfüllt. Bestrebungen, dieses Verfahren auch auf Steuerungssysteme der Maschinen- und Anlagentechnik zu übertragen, gibt es bereits seit längerer Zeit [MPBC1991, Kow1996, HFW1996, HFWW1998, FrLi1998, u.a.]. Der grundlegende Gedanke vieler dieser aufgezeigten Projekte - so auch des von uns hier vorgestellten Projektes - besteht aus zwei Hauptbereichen (Abb. 2).

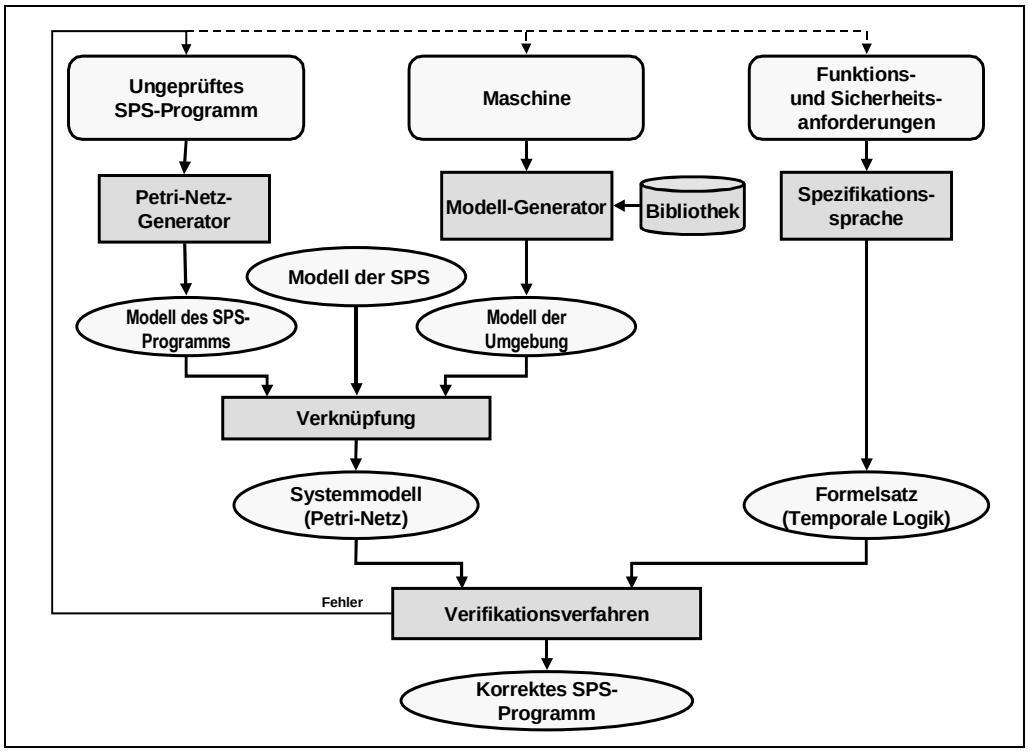

Abbildung 2 - Übersicht über das Projekt

Um die notwendigen Grundlagen für das Model-Checking zu erhalten, muss einerseits ein formales Modell des zu untersuchenden Automatisierungssystems und andererseits eine formale Beschreibung der Programmanforderungen existieren.
Ein Automatisierungssystem besteht üblicherweise aus einer Speicherprogrammierbaren Steuerung (SPS) und der zu steuernde Maschine oder Anlage. Auf der SPS wird, neben den Funktionen des SPS-Betriebssystems, ein SPS-Anwenderprogramm abgearbeitet. Die SPS erhält dazu Sensorwerte von der Maschine, verarbeitet diese und liefert ihrerseits Signale zur Ansteuerung der Maschinen-komponenten zurück. Dieses betrachtete System wird in mehreren Schritten, die automatisch bzw. interaktiv ablaufen, in ein formales Modell überführt, das auf Petri-Netzen beruht. An diesem Systemmodell sind bereits klassische Analysen der Petri-Netz-Theorie durchführbar, die auf den Nachweis von Programmeigenschaften wie etwa Verklemmungsfreiheit und allgemeinen Lebendigkeitseigenschaften abzielen.
Eine höhere Qualität wird durch die Analyse und den Nachweis von speziellen Programm-eigenschaften erzielt. Als Benutzerschnittstelle für die Anforderungsformulierung dient hier eine eigens entwickelte Spezifikationssprache - die Sicherheitsfachsprache ‚SFS'. Mit ihr ist es möglich, die Steuerungsanforderungen aus Lasten- und Pflichtenheft 'semiverbal', d. h. in nahezu natürlicher Sprache, auszudrücken. Diese Anforderungen lassen sich wegen ihrer definierten Struktur in Formeln der Temporalen Logik übersetzen. Die vollständige Erfüllung dieser Formeln wird am Systemmodell durch Verifikationsverfahren bewiesen. Sollten einzelne Formeln (bzw. die dahinter stehenden Programmanforderungen) nicht erfüllt werden, erhält man Hinweise auf Fehlerquellen im Anwenderprogramm.

3 Spezifikation der Eigenschaften eines Steuerungsprogramms

Ein Verfahren zur rechnerunterstützten Formulierung der geforderten Eigenschaften eines Steuerungsprogramms muss verschiedenen - teilweise konträren - Anforderungen gerecht werden. Einerseits muss es die notwendige formale Basis besitzen, um die Darstellung der Programmeigenschaften als temporallogische Formeln zu ermöglichen, andererseits muss es auch von einem Ingenieur verwendet werden können, der nicht über detaillierte Kenntnisse dieser formalen Grundlagen verfügt. Als dritter Punkt müssen die speziellen technischen Gegebenheiten einer speicherprogrammierbaren Steuerung und die sich daraus ergebenden Konsequenzen bei der Erstellung der Anforderungen berücksichtigt werden. Bei der Formulierung von Programmanforderungen können primär nur steuerungstechnische Elemente, wie Sensoren und Aktoren, verwendet werden. Die Informationen über die Existenz dieser Elemente können der Variablendeklaration des zu verifizierenden SPS-Programms entnommen werden. Der Variablendeklarationsteil des SPS-Programms nimmt somit eine zentrale Stellung für das Verifikationsverfahren ein. Die Bezeichnungen und Spezifika der eingesetzten Sensoren und Aktoren werden sowohl im modellierten Automatisierungssystem (nicht nur im Modell des SPS-Programms, sondern auch im Modell der Maschine), als auch in den Anforderungen verwendet.

Die Erzeugung des Modells eines Automatisierungssystems kann in vier grundlegenden Schritten zusammengefasst werden:

1. Überführung des SPS-Anwenderprogramms in ein Petri-Netz-Modell,
2. Modellierung der Maschine als Petri-Netz mit Hilfe einer geeigneten Bibliothek,
3. Modellierung der Systemfunktionen der SPS,
4. Verbindung dieser drei Teilmodelle zu einem Gesamtmodell in Form eines Petri-Netzes.

Während der Schritt 3 lediglich einmal durchzuführen ist (die Systemfunktionen einer SPS sind bekannt) und der Schritt 2 auch nur einmal für jede zu verifizierende Applikation erfolgt, muss der erste Schritt für jedes SPS-Programm so lange wiederholt werden, bis alle Fehler beseitigt wurden und alle gegebenen Anforderungen erfüllt sind. Formale Basis der Modellierung sind die speziell im Rahmen dieses Projektes entwickelten Registernetze, die als eine Variante gefärbter Netze eine einfache und kompakte Darstellung der Zustände der einzelnen Systemvariablen ermöglichen.
Die Überführung des *SPS-Anwenderprogramms* in das formale Modell wird durch den Petri-Netz-Generator durchgeführt. Das verarbeitete Format des Steuerungsprogramms ist die Anweisungsliste (AWL) entsprechend der Norm IEC 61131-3. Diese spezielle Darstellungsform wurde gewählt, da die AWL die elementarste genormte Darstellungsform ist. Alle anderen Programmiersprachen, wie Funktionsbausteinsprache, Kontaktplan, Strukturierter Text oder die Ablaufsprache lassen sich informationserhaltend in die AWL überführen. Der umgekehrte Weg ist jedoch nur mit Einschränkungen möglich. Die Überführung eines AWL-Programms in ein Registernetz geschieht

vollständig automatisch, die Größe des erzeugten Registernetzes ist von der Anzahl der Programmzeilen sowie der Anzahl der benutzten SPS-Variablen abhängig.

Die Modellierung der *Maschine* muss erfolgen, da eine Zertifizierung immer nur am Gesamtsystem „Software - SPS - Maschine“ möglich ist. Die Erstellung des Modells einer Maschine oder einer kompletten Anlage gestaltet sich i. Allg. sehr schwierig. Hierfür gibt es in der Theorie und Praxis sehr viele unterschiedliche Ansätze und Herangehensweisen, so dass an dieser Stelle eine Aufzählung der Möglichkeiten nicht sinnvoll ist. Je nach Zielstellung der Untersuchungen sind diese Modellierungsverfahren entweder funktionell oder analytisch orientiert [Kie1997, Sau1999] .

Der Maschinenmodellierung innerhalb von „Safety-Knight“ erfolgt zustandsorientiert. Der Ansatz beruht darauf, die Modellierung auf die für die SPS wesentlichen Zustände der Maschine und deren Übergänge zu beschränken. So ist für die SPS die exakte Position eines Hydraulikzylinders nicht relevant, durch die angebrachten Sensoren „sieht“ sie lediglich einzelne Ausschnitte. Sind an dem betrachteten Hydraulikzylinder zwei Sensoren zur Markierung der Endlagen montiert, so kann die SPS lediglich drei Zustände dieses Zylinders unterscheiden: die beiden Endlagen, bei denen jeweils ein Sensor ein Signal liefert, und einen dritten Zustand, bei dem sich der Zylinder zwischen den beiden Endlagen befindet und keiner der beiden Sensoren ein Signal an die SPS liefert. Eine quantitative Modellierung der Bewegungsgeschwindigkeiten oder Übergangsdauern erfolgt nicht. Auf diese Weise erhält man sehr kompakte und effiziente Modelle für die Teilkomponenten der Maschine, die für die funktionalen Analysen jedoch völlig ausreichen.

Ein wichtiger Punkt der Steuerungsmodellierung ist die Nachbildung der verwendeten *Steuerung*. Bei der Modellierung der typischen Arbeitsweise einer SPS muss eine automatisierungstechnische Annahme berücksichtigt werden, wonach die SPS in der Lage sein muss, alle relevanten Signaländerungen in ihrer Umgebung zu erkennen.

Zur Verdeutlichung dieser Forderung muss man sich mit der charakteristischen Arbeitsweise einer Speicherprogrammierbaren Steuerung auseinandersetzen. Eine SPS arbeitet in einem festgelegten und konstanten Rhythmus, der auch als *SPS-Zyklus* bezeichnet wird. Dabei erfolgt am Zyklusanfang ein Signalabgleich mit der Umgebung: es werden einerseits Signale aus der Umgebung eingelesen und zwischengespeichert, andererseits werden frühere Berechnungsergebnisse an die Umgebung abgegeben. Mit den eingelesenen Werten und internen Variablen wird danach das Anwender-programm abgearbeitet. Die Ergebnisse dieser Berechnung werden jedoch erst im nächsten SPS-Zyklus an die Umgebung ausgegeben. Der Neustart des SPS-Zyklus erfolgt in einem konstanten Zeitraster, der *Zyklus- oder Abtastzeit*. Aus diesem Grund entspricht die Reaktionszeit einer SPS im günstigsten Fall der Zykluszeit, im worst-case steigt diese Zeit auf das Doppelte an. Diesen SPS-typischen Umständen muss auch bei der späteren Formulierung der Anforderungen Rechnung getragen werden.

Im vorliegenden Projekt wurde der Ablauf des SPS-Zyklus gemäß Abb. 3 umgesetzt.

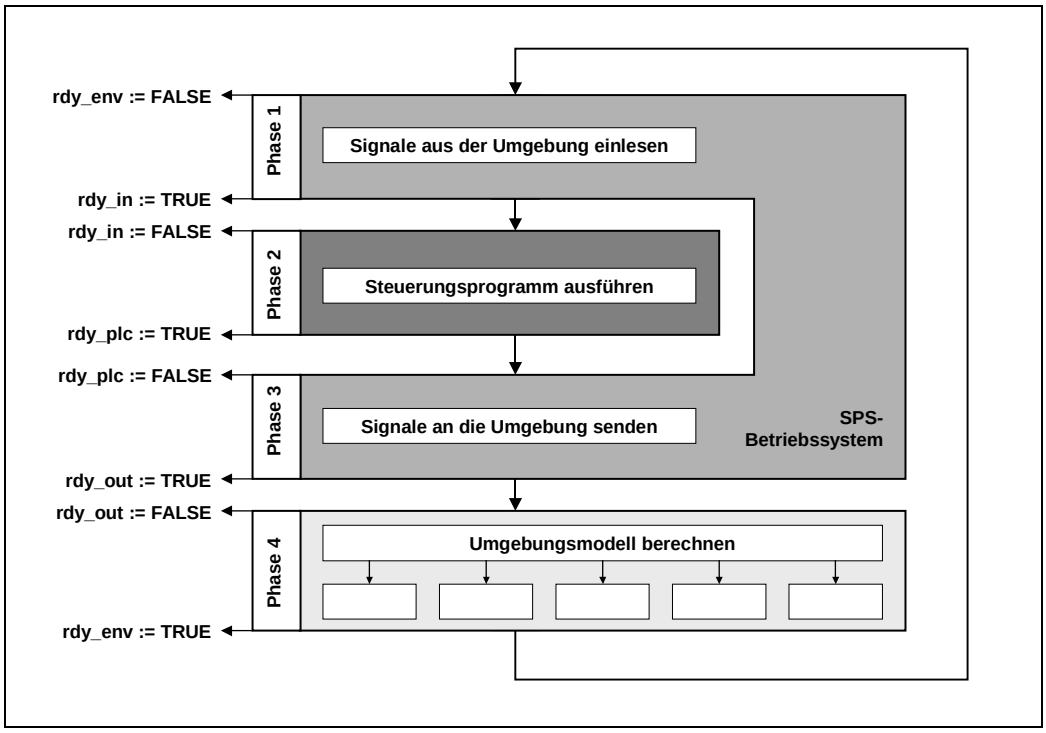

Abbildung 3 - Nachbildung des SPS-Zyklus

Der Vorgang wurde in vier Phasen unterteilt, wobei die einzelnen Phasen klar voneinander getrennt sind. Mit der Bearbeitung der folgenden Phase kann erst begonnen werden, nachdem die vorherige Phase komplett abgeschlossen wurde. Bei der Modellierung werden vier zusätzliche Variablen erzeugt (Abb. 3), die jeweils die Komplettierung einer Phase anzeigen sollen. Die Variablen ‚rdy_in', ‚rdy_plc', ‚rdy_out' und ‚rdy_env' werden dazu für einen Simulationsschritt auf ‚TRUE' gesetzt. Diese Monitoring-Variablen kommen bei der Erzeugung der temporallogischen Formeln wieder zum Einsatz.

Die Annahme der SPS-Reaktionsfähigkeit wird im Modell eingehalten, indem sämtliche Schaltvorgänge des Umgebungsmodells für die Dauer der Bearbeitung des Anwender- und des Systemprogramms durch spezielle Netzstrukturen blockiert werden [HeMe2000].

Die Verknüpfung der einzelnen Petri-Netze (Modell des Steuerungsprogramms, Modell der SPS-Systemfunktionen, Maschinenmodell) wird durch die Verschmelzung korrespondierender Plätze der Variablen des Steuerungsmodells und des Umgebungsmodells realisiert. An dieser Stelle wird wiederum die zentrale Bedeutung der Variablendeklaration des SPS-Programms unterstrichen. Das Systemmodell repräsentiert die Funktionsweise des gesamten Systems, aus ihm können alle erreichbaren Zustände dieses Systems abgeleitet werden. Als problematisch erweist sich hierbei der exponentiell wachsende Zustandsraum, der mit den heute üblichen Verfahren schon für Systeme mittlerer Größe als nicht mehr beherrschbar gilt. Überlegungen zur Lösung für dieses Problem zielen auf geeignete Komprimierungs- und Reduktionsverfahren (s. [Hei98] für einen Überblick über verfügbare Techniken) .

4 Eine Sicherheitsfachsprache zu Spezifikation von Programmeigenschaften

Die technologischen und technische Vorgaben für die gewünschte Funktionsweise einer Maschine werden üblicherweise durch Kooperation zwischen Auftraggeber und Automatisierungstechniker erstellt und in einem Pflichtenheft schriftlich festgehalten. Dieses beinhaltet die notwendigen Abläufe des Programms, die erforderlichen Reaktionen auf festgelegte Ausnahmesituationen, aber auch Festlegungen, welche Programmreaktionen unbedingt vermieden werden müssen.
Zur Beschreibung der Anforderungen an ein Steuerungsprogramm stehen viele Möglichkeiten zur Verfügung: Ablauf- oder Schaltdiagramme, Funktionspläne und andere Darstellungsweisen. Eine sehr elementare und auch für Nichtspezialisten verständliche Form ist die verbale Formulierung der notwendigen Programmeigenschaften. Mit ihr lassen sich bereits in einer frühen Projektphase grundlegende Anforderungen beschreiben.
Die Darstellung von Programmanforderungen mit Hilfe natürlicher Sprache ist jedoch sehr problematisch, da diese von Natur aus mehrdeutig und deshalb für eine formale Darstellung von Anforderungen nur bedingt geeignet ist. Die im Rahmen des Projektes definierte Sicherheitsfachsprache (SFS) zur Spezifikation von Programmeigenschaften orientiert sich stark an steuerungstypischen Anforderungen und Formulierstilen und bietet durch die formale Vorgabe einer Syntax die Möglichkeit der Hinterlegung einer formalen Semantik. Durch die Vorgabe fester Satzmuster und -strukturen ist sie nicht mehr mehrdeutig und eine automatische Übersetzung in Formeln der Temporalen Logik ist möglich. Der Anwender wird also in die Lage versetzt, auf einfache und verständliche Art und Weise seine Anforderungen darzustellen. Ein ähnlicher Ansatz ist [HFW1996, HFWW1998] zu entnehmen, auch hier werden zu verifizierende Eigenschaften eines Steuerungsprogramms verbal ausgedrückt. Man verwendet hier festdefinierte Satzstrukturen (Schablonen), die bestimmte SPS-relevante Anforderungen repräsentieren. Durch Vervollständigung der Schablonen mit projektspezifischen Eigenschaften und Annahmen entstehen komplette Verifikationsaufträge, die anschließend abgearbeitet werden können. Ein ähnlich aufgebautes Schablonensystem kommt auch in der SFS zum Einsatz.
Die Unterteilung der Arten von Anforderungen an ein Steuerungsprogramm erfolgt durch zwei Parameter [Mer2000] :

1. Der *Modalparameter* gibt die logische Bedeutung der Anforderung an.
2. Der *Zeitparameter* gibt die zeitliche Reichweite der Anforderung an.

Durch den Modalparameter werden drei Hauptkategorien von Anforderungen unterschieden:

- *Forderungen* beschreiben notwendige und erwünschte Reaktionen des Steuerungsprogramms auf festgelegte Signale. Sie lassen sich nochmals in kurzfristige, operative Forderungen und mittel- bzw. langfristige, strategische Forderungen unterscheiden.
- *Verbote* beschreiben, wie das Programm nicht reagieren darf, d. h. welche Reaktionen auszuschließen sind.
- *Möglichkeiten* beschreiben erlaubte Reaktionen des Steuerungsprogramms. Sie bilden das logische Verbindungsglied zwischen den Forderungen und den Verboten. Sie stellen einerseits die Umkehrung eines Verbotes dar („Alles was nicht explizit verboten ist, ist erlaubt."), andererseits sind sie abgeschwächte Forderungen („Die Reaktion ist möglich, jedoch nicht zwingend notwendig.").

Allen drei Kategorien gemeinsam ist die Bildung eines modalen, implikativen Zusammenhangs zwischen einem bereits bestehenden Ereignis und einem anderen Ereignis, dessen zeitlicher Horizont durch den Zeitparameter definiert wird.

Der Zeitparameter spezifiziert als ein weiteres Unterscheidungsmerkmal innerhalb der Modalkategorien die Reichweite der jeweiligen Aussage. Mit ihm wird ausgedrückt, über welchen Zeitraum die entsprechende Anforderung gültig sein soll. Typische automatisierungstechnische Forderungen an ein Steuerungsprogramm beziehen sich auf zwei Zeitebenen: tritt auf eine Bedingung die zugehörige Reaktion innerhalb einer vorgegebenen (möglichst kurzen) Zeitspanne ein bzw. tritt auf eine Bedingung die zugehörige Reaktion überhaupt ein?

Ausgehend von dieser Erkenntnis wurden zwei Hauptgruppen mit insgesamt fünf Varianten des Zeitparameters differenziert. Diese fünf Varianten sind im verbalen Satzmuster an charakteristischen Schlüsselwörtern zu erkennen. Die beiden ersten Varianten beziehen sich auf die bereits diskutierte Arbeitsweise einer SPS. Die Reichweite der Aussage ist in diesen Fällen an den SPS-Zyklus gekoppelt. Man unterscheidet hierbei Anforderungen, die im gleichen SPS-Zyklus („*gleichzeitig*") oder im unmittelbar nachfolgenden SPS-Zyklus („*unmittelbar danach*", „*sofort*") erfüllt werden müssen. Die drei folgenden Varianten sind dagegen unabhängig vom SPS-Zyklus und in ihrer Reichweite nicht starr festgelegt („*irgendwann danach*", „*solange*", „*niemals*", „*irgendwann davor*").

Mit diesen Modal- und Zeitparametern wurden insgesamt 18 automatisierungstechnisch relevante Satzmuster definiert (Abb. 4). Zur Unterstützung spezieller Gewohnheiten des Nutzers der SFS lassen sich durch Umstellung der Satzglieder weitere, inhaltlich redundante Satzmuster erzeugen.

Zeit-parameter / Modal-parameter	**Zustand** "gleichzeitig"	**Direkt** "sofort"	**Selbstbegrenzt** "solange"	**Fremdbegrenzt** "bis irgendwann"	**Unbegrenzt** "irgendwann"
Einfache Forderung "Wenn A, dann muss B."	✓	✓	✓	✓	✓
Erweiterte Forderung "Nur wenn A, dann muss B."	✓	✓	✓	✓	✓
Erweiterte Möglichkeit "Nur wenn A, dann darf B."	✓		✓	✓	✓
Einfaches Verbot "Wenn A, dann darf nicht B."	✓		✓	✓	✓

Abbildung 4 - Matrix relevanter Satzmuster der SFS

Nach der Auswahl des Satzmusters muss der Anwender die bestehenden Freiräume innerhalb der Sätze mit den applikationsspezifischen Wortphrasen ausfüllen. Diese werden durch Interpretation der SPS-Variablen und deren relevante Werten gebildet. In der (Abb. 5) ist zu erkennen, dass die SPS-Variable ‚x_1' einen bestimmten Taster repräsentiert und dass ihr ‚TRUE'-Zustand die Bedeutung „gedrückt" und der ‚FALSE'-Zustand die Bedeutung „nicht gedrückt" hat. Diese Zuordnung ist für alle verwendeten Variablen einmal zu erstellen. Anschließend werden nur noch diese erzeugten Wortphrasen verwendet, ohne dass sich der Anwender ständig über die steuerungstechnischen Gegebenheiten im Klaren sein muss.

Beispiele für die Benutzung der SFS sind:
Einfache Forderung - direkt:

„*Wenn der Not-Aus-Taster betätigt wurde, dann muss der Motor sofort abgeschaltet werden.*"

Einfaches Verbot - Zustand:

„*Wenn das Schutzgitter offen ist, dann darf die Presse nicht aktiviert werden.*"

Erweiterte Möglichkeit - Zustand:

„Nur *wenn das Schutzgitter geschlossen ist, dann darf die Presse aktiviert werden.*"

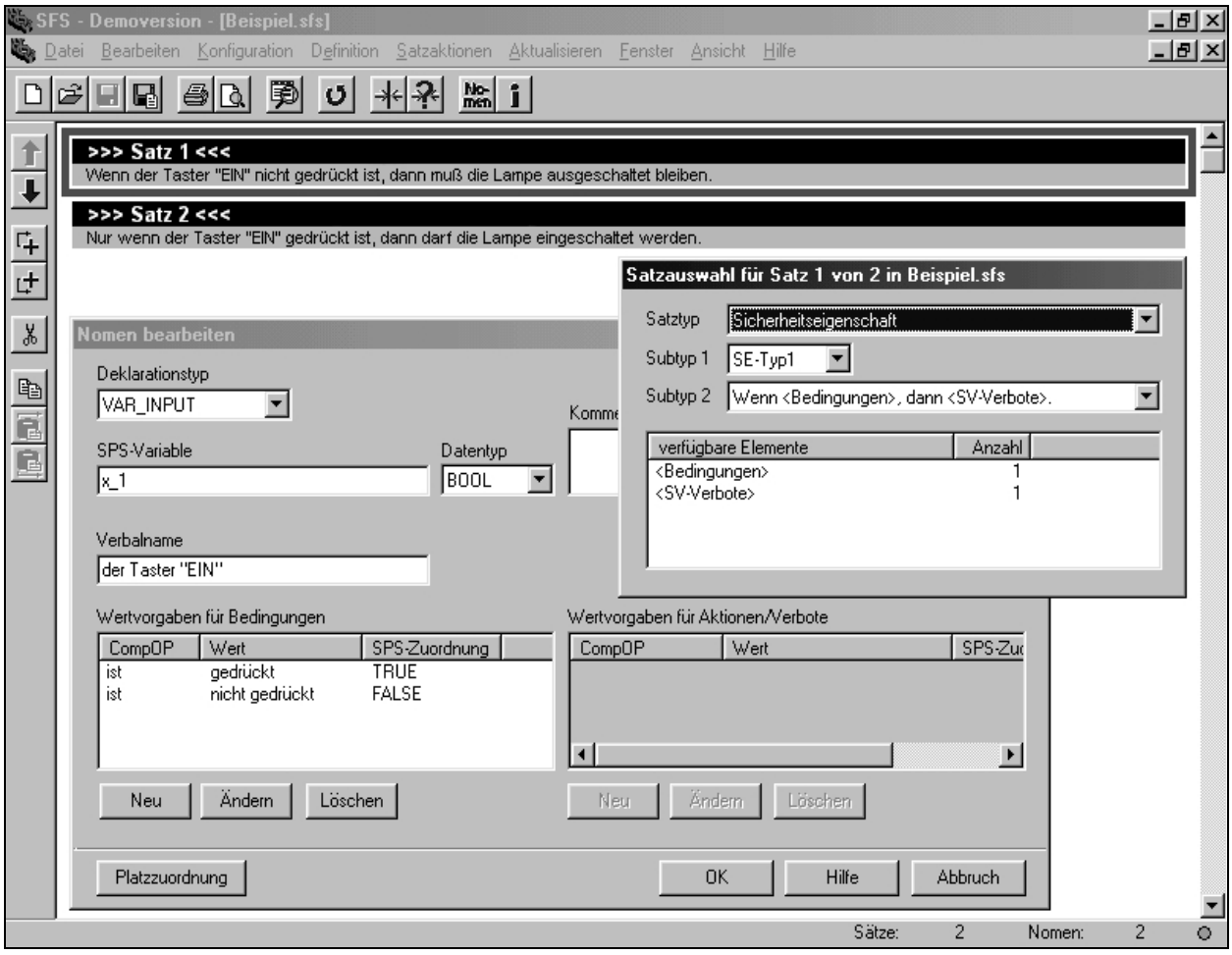

Abbildung 5 – Editor zur Erstellung der Programmanforderungen

Um Zustandssequenzen des Gesamtmodells verifizieren zu können, benötigt man eine Logik, die es gestattet, die genannten relativen Zeitbegriffe („*sofort*“, „*nie*“, „*solange*“) zu verwenden; die Wahl fiel auf CTL (computation tree logic), eine Temporale Logik, für die effiziente Model-Checking-Algorithmen existieren.

Als Besonderheit bei der Erstellung der CTL-Formeln muss die zyklische Arbeitsweise einer SPS beachtet werden. Änderungen in den Eingangssignalen können, wie bereits dargestellt, frühestens nach der Bearbeitung eines kompletten SPS-Zyklus zu einer entsprechenden Änderung in den Ausgangssignalen führen. Der CTL-„at next“-Operator wird diesem Umstand jedoch nicht gerecht, vielmehr kommen nun die im Abschnitt 3 eingeführten Monitoring-Variablen zum Einsatz.

`AG((rdy_in & `*B*`) -> A[!rdy_plc U (rdy_plc & `*F*`)])`

B - Bedingung, Eintreten eines Signals oder Ereignisses
F - Folgerung, geforderte Reaktion

Die dargestellte CTL-Formel identifiziert zunächst jeden Systemzustand, bei dem nach Beendigung des Lesens der Eingangsvariablen ein vorgegebenes Signal bzw. eine Signalkombination erfüllt

wird. Für alle nachfolgenden Systemzustände wird nun überprüft, ob im Moment der Beendigung der Bearbeitung des SPS-Programms (identifizierbar durch die steigende Flanke von ‚rdy_plc') die festgelegte Reaktion erfüllt wird. Auf diese Weise wird eine Sequenz aufeinanderfolgender Systemzustände von der Länge eines SPS-Zyklusses überwacht.
Ergebnis der Übersetzung der in der SFS formulierten Anforderungen ist eine Menge temporallogischer Formeln. Die atomaren Präpositionen dieser Formeln korrespondieren mit Variablen des Systemmodells, also wiederum mit den Einträgen im Variablendeklarationsteil des SPS-Programms.
Das anschließende Model-Checking-Verfahren liefert als Ergebnis der Überprüfung entweder eine Bestätigung für die Einhaltung der Spezifikation in allen modellierten Zuständen des Systems oder mögliche Gegenbeispiele, in denen eine oder mehrere Anforderungen nicht erfüllt werden. Die Aussagekraft des Verifikationsergebnisses hängt jedoch direkt von der Qualität der gegebenen Voraussetzungen (Programm, Umgebungsmodell, Anforderungen) ab. Als Ergebnis einer erfolgreichen Verifikation kann man somit von einem korrekten SPS-Programm bezüglich der gegebenen Voraussetzungen sprechen.

5 Zusammenfassung

Das hier vorgestellte Konzept zur Verifikation von SPS-Programmen führte zur Entwicklung des Werkzeugkastens „Safety-Knight". Dieser besteht aus mehreren Modulen, von denen zum Zeitpunkt der Drucklegung dieses Artikels wesentliche Teile bereits realisiert waren, andere Teilprogramme werden in den folgenden Monaten vervollständigt.
Bei der Implementierung der Methoden in "Safety-Knight" wurde besonderer Wert auf die leichte Anwendbarkeit und Nutzerfreundlichkeit des Systems gelegt. Die verwendeten formalen Methoden, wie Petri-Netze und Temporale Logik bilden das technische Gerüst, mit dem sich der Anwender jedoch nicht zwangsläufig auseinandersetzen muss. Das SPS-Anwenderprogramm wird durch einen automatischen Compiliervorgang in ein Petri-Netz überführt. Für die Erstellung des Umgebungsmodells und die Formulierung der Programmanforderungen stehen leicht bedienbare Editoren zur Verfügung. Der Umgang mit diesen Werkzeugen wird durch Bibliotheken mit vorgefertigten Maschinenmodellen bzw. normativen Programmanforderungen unterstützt. Als Verifikationskomponenten werden externe Model-Checking-Werkzeuge [VHHP1995, McM1992] Verwendung finden.
Die vorgestellte Methode der Zertifizierung bestehender Anwenderprogramme wird mit dem Werkzeug 'Safety-Knight' durchgängig verwirklicht. Von der Eingabe eines Programms bis zur Interpretation von Analyseergebnissen unterstützt diese Software einen weitgehend automatischen Verifikationsprozess. Besonderes Gewicht bei der Entwicklung wird auf die Erweiterbarkeit, Plattformunabhängigkeit und Konfigurierbarkeit des Programms gelegt. Das Werkzeug richtet sich vor allem an Zertifizierungseinrichtungen und Softwareentwickler. Die Einsatzmöglichkeiten

reichen dabei vom Sicherheitsnachweis der Steuerungsprogramme in sicherheitskritischen Anwendungen bis hin zur projektbegleitenden Qualitätskontrolle bei der Erstellung von SPS-Software.

Literatur

[Eve1991] Everking, H.: Verifikation digitaler System, B.G. Teubner, Stuttgart, 1991.

[FrLi1998] Frey, G., Litz, L., Entwurf und formale Verifikation von Steuerungen mit interpretierten Petri-Netzen, VDI Berichte Nr. 1397, S. 291 - 298, 1998

[Hei1998] Heiner, M.: Petri Net Based System Analysis without State Explosion, Proc. High Performance Computing ´98, 394-403, Boston, 1998, 1998.

[HeMe1997] Heiner, M., Menzel, T.: Petri-Netz-Semantik für die SPS-Anwenderprogrammier-sprache Anweisungsliste, BTU-Bericht Reihe Informatik I-20/1997, Cottbus, 1997.

[HeMe1998] Heiner, M., Menzel, T.: A Petri Net Semantics for the PLC Language Instruction List, IEEE 4th Workshop on Discrete Event Systems (WODES 98), 161-166, Cagliari / Italy, 1998.

[HeMe1999] Heiner, M., Menzel, T.: Modellierung und Analyse von SPS-Anwenderprogrammen mit Petri-Netzen, EKA '99, 6. Fachtagung, 247-265, Braunschweig, 1999.

[HeMe2000] Heiner, M., Menzel, T.: Time-related modelling of PLC systems with time-less Petri nets, Proc. WODES 2000, 275-282, 2000.

[HFW1996] Hölzlein, M., Filkorn, Th., Warkentin, P., Formale Verifikation von SPS-Programmen, VDI-Berichte Nr. 1282, S. 25 -34, 1996

[HFWW1998] Hölzlein, M., Filkorn, Th., Warkentin, P., Weiß. M.: Eine Verifikationskomponente für HiGraph, VDI-Berichte 1397, S. 281 - 290, 1998

[HMMM1997] Heiner, M., Meier, M., Mertke, T., Menzel, T.: Petri-Netz-basierte Methoden zur sicherheitstechnischen Zertifizierung von SPS-Anwenderprogrammen, BTU-Bericht Reihe Informatik I-19/1997, Cottbus, 1997.

[Kie1997] Kiencke, U.: Ereignisdiskrete Systeme, Modellierung und Steuerung verteilter Systeme, Oldenbourg-Verlag München, Wien, 1997.

[Kow1996] Kowalewski, S.: Verifikation von Steuerungen mit Hilfe von Bedingung/Ereignis-Systemen, VDI-Berichte Nr. 1282, S. 57 - 66,1996

[Krö1992] Kröger, F.: Temporal logics of Programs; Springer-Verlag, 1992.

[MaPn1992] Manna, Z., Pnuelli, A.: The temporal logic of reactive and concurrent systems - Specification, Springer-Verlag, New York, 1992.

[MaPn1995] Manna, Z., Pnuelli, A.: Temporal verification of reactive systems – Safety, Springer-Verlag, New York, 1995.

[McM1992] McMillan, K. L.: The SMV System, Techn. Report, Carnegie-Mellon Univ. 1992.

[MeiMe1998] Meier, H., Mertke, T.: Verifikation zertifizierungspflichtiger Steuerungssoftware, Zeitschrift für wirtschaftlichen Fabrikbetrieb 6/98, S. 247 – 250, 1998.

[MerMe2000] Mertke, T., Menzel, T.: Methods and tools to the verification of safety-related control software, PLCopening, 10-11, 2000.

[Mer2000] Mertke, T.: SFS - Formale Spezifikation von Anwenderprogrammen in der Automatisierungstechnik, erscheint als BTU-Bericht Reihe Informatik, 2000.

[MPBC1991] Moon, I., Powers, G. J., Burch, J. R., Clarce, E. M.: An automatic verification method using Temporal logic for sequential chamical process control systems, AIChE Journal, vol. 38, p.67, 1991

[Pet1981] Peterson, J. L.: Petri Net Theory and the Modeling of Systems, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Sau1999] Sauerbier, T.: Theorie und Praxis von Simulationssystemen, Vieweg-Verlag, Wiesbaden, 1999.

[VHHP1995] Varpaaniemi, K., Halme, J., Hiekkanen, K., Pyssyslao, T.: PROD Reference Manual, Helsinki Univ. of Technology, Digital System Laboratory, Series B: Techn. Report no 13, Espoo, 1995.