Informatik aktuell

H. Unger (Hrsg.)

Echtzeit und Sicherheit

Echtzeit 2018

Springer Vieweg

Informatik aktuell

Herausgegeben
im Auftrag der Gesellschaft für Informatik (GI)

Herwig Unger (Hrsg.)

Echtzeit und Sicherheit

Echtzeit 2018

Fachtagung des gemeinsamen Fachausschusses
Echtzeitsysteme von
Gesellschaft für Informatik e.V. (GI),
VDI/VDE-Gesellschaft für Mess- und Automatisierungstechnik (GMA) und
Informationstechnischer Gesellschaft im VDE (ITG)
Boppard, 15. und 16. November 2018

GESELLSCHAFT FÜR INFORMATIK E.V.

VDI VDE VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik

ITG INFORMATIONSTECHNISCHE GESELLSCHAFT IM VDE

Springer Vieweg

CR Subject Classification (2001): C3, D.4.7

ISSN 1431-472X

ISBN 978-3-662-58095-0 ISBN 978-3-662-58096-7 (eBook)
https://doi.org/10.1007/978-3-662-58096-7

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Springer Vieweg

Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer-Verlag GmbH, DE und ist ein Teil von Springer Nature
Die Anschrift der Gesellschaft ist: Heidelberger Platz 3, 14197 Berlin, Germany

Vorwort

Kontrolle des Eigenheims aus der Ferne, Überweisungen und andere Finanztransaktionen in Echtzeit, Videogespräche über das Internet mit Partnern am anderen Ende der Welt: längst ist eine Reihe von Anwendungen mit Echtzeitcharakter oder -anforderungen an Verarbeitungs- und/oder Übertragungsgeschwindigkeit bequeme Selbstverständlichkeit für jedermann geworden. Weitaus länger und genauso intensiv nutzt die Industrie über das Internet zugängliche oder beeinflussbare Steuerungen und Kommunikationseinrichtungen. Andererseits sind mittlerweile selbst die Nachrichten in den Medien voll mit Meldungen über ungewollte Störungen Dritter, private oder staatlich gewollte oder kontrollierte Angriffe.

Der Wurm Stuxnet, der gezielt kerntechnische Anlagen befiel und empfindlich hätte stören können, war nur ein erster, massiver Fall, der in die Öffentlichkeit durchdrang und damit der Allgemeinheit die Gefährdungslage unmissverständlich vor Augen führte. Die wirklichen Gefahren, die durch Cyberattacken oder gar einen Cyberkrieg heraufbeschworen werden, konnte der gewillte Bürger von diesem Zeitpunkt an leicht selbst einschätzen, insbesondere wenn er die Abhängigkeit der Industrie und des öffentlichen Lebens vom Internet in vollem Umfange begreift. Es wird klar, dass es hierbei nicht mehr nur um finanzielle Verluste, sondern um reale Gefahren für Leib und Leben der Bevölkerung geht.

Vergessen wird in den Betrachtungen gerne, dass immanente Gefährdungen jedoch vor allem von der Komplexität der zu steuernden und der steuernden Systeme selber ausgehen. Genaue und tiefgehende Durchdringungen aller Anwendungsfälle und Tests aller möglichen Kombinationen von Eingabedaten sind i. A. aber vor allem durch viel zu kurze Zeitvorgaben bei den Entwicklungsterminen nicht zu realisieren: ökonomische Zwänge dominieren Sicherheitsbedenken. So sind selbst bei einem optimal verlaufenden Systementwurf (systematische) Fehler in den Quellcodes in keiner Weise völlig auszuschließen und heute eher die im Prototyp zu erwartende Regel. Ungewollte und unberücksichtigte Veränderungen in den Systemumgebungen, insbesondere durch Wechselwirkungen mit anderen (fremdentwickelten) Systemen, tragen während der Nutzungsdauer ein weiteres, besonderes Risiko in sich, sind aber gerade im Zeitalter mehr und mehr automatischer und sich selbst organisierender Produktion typische Situationen.

Damit ist es nur konsequent, dass das Leitthema unserer nun zum 39. Mal stattfindenden Tagung „Echtzeit“ mit der Unterthematik Systemsicherheit wieder einmal die aktuelle Entwicklung aufgreift und sich (wiederholt) der Frage stellt, welche Veränderungen hier in den letzten Jahren eingetreten sind.

Mit der nun DIN-normierten Echtzeitsprache SafePEARL hat der entsprechende Arbeitskreis unseres Fachausschusses unzweifelhaft einen einzigartigen Beitrag für sichere Systeme geleistet: eine Arbeit, die auf eine inzwischen fast fünfzigjährige Entwicklung von den ersten Anfängen an zurückblicken kann. Diesem Erfolg widmen wir in diesem Jahr einen speziellen Eröffnungsvortrag, der nach allen Einzelbeiträgen der Vorjahre das Gesamtwerk ausführlich würdigt.

Neben weiteren Beiträgen zur Vermeidung von Störungen der Arbeit datenverarbeitender Systeme und von Kommunikation durch Dritte ist Fehlervermeidung und -toleranz die zweite Seite der Medaille. Funktionale Sicherheit stellt daher das Thema einer weiteren Sitzung unseres Treffens dar, das sowohl in industriellen Normen als auch in der Lehre seine Widerspiegelung findet. Aktuelle Anwendungen im Rahmen von Industrie 4.0, wie z. B. flexible Transportsysteme und insbesondere Autonomes Fahren, bieten hier eine Vielzahl von Themen zur Diskussion. Eine Reihe weiterer Beiträge aus Hardware- und Betriebssystemsicht und insbesondere in Bezug auf Schnittstellen komplettiert die Perspektive des Fachausschusses auf die Probleme.

Offen bleibt trotz aller wissenschaftlichen und technischen Leistungen dennoch eine Reihe von Fragen. Ihr Kern lässt sich in einer Frage zusammenfassen, und zwar warum die Industrie auf alle bislang präsentierten und z. T. sogar sehr einfachen Hardwarelösungen (wie z. B. den Übergang zur Harvard-Architektur) verzichtet und stattdessen ausschließlich auf immer wieder korrumpierbare Softwarelösungen setzt.

Alle Organisatoren der Tagung hoffen, dass das Hotel „Ebertor“ in Boppard mit der unvergleichlichen Natur des mittleren Rheintals erneut ein inspirierender und für den Fachausschuss fast heimischer Ort für eine Reihe innovativer Vorträge und fruchtbringender Diskussionen sein wird. Nicht zuletzt sei wiederum Frau Düring für ihre Arbeit bei der Organisation der Tagung und der technischen Erstellung des vorliegenden Tagungsbandes gedankt.

Hagen, im August 2018 Herwig Unger

Inhaltsverzeichnis

Eröffnungsvortrag

Funktionale Sicherheit

Lehre

Applikationen von Echtzeitsystemen

Sicherheitsgerichtete Echtzeitsysteme

Von Algol 68 zu SafePEARL

Wolfgang A. Halang und Marcel Schaible

ehem. Lehrstuhl für Informationstechnik, insb. Realzeitsysteme
FernUniversität in Hagen, 58084 Hagen
{wolfgang.halang|marcel.schaible}@fernuni-hagen.de

Zusammenfassung. Anlässlich des Erscheinens der neuen Norm DIN 66253 für die Echtzeitprogrammiersprache PEARL im Jahre 2018 wird die Entwicklungslinie der Sprache während der letzten 50 Jahre nachgezeichnet. Durch Berücksichtigung der im Laufe der Zeit gemachten Erfahrungen weist diese Linie eine eindeutige Richtung auf: Weg von komplexen und hin zu einfachen domänenorientierten Konstrukten, die sich durch sichere Handhabbarkeit und Eignung zur Programmierung solcher automatisierungstechnischer Anwendungen auszeichnen, die hohen Anforderungen an ihre funktionale Sicherheit genügen müssen. Dazu wird dargelegt, wie die neue Sprachversion SafePEARL der internationalen Sicherheitsnorm IEC 61508 gerecht wird. Der Entwicklungsstand der für die Lehre gedachten Sprachversion OpenPEARL wird skizziert.

1 Automatisierungstechnik und Programmiersprachen

Die in der Automatisierungstechnik vorherrschende Programmierpraxis ist durch den Einsatz ungeeigneter Hilfsmittel gekennzeichnet. Selbst die in der Norm IEC 61131-3 speziell für die Steuerungstechnik eingeführte höhere Programmiersprache Strukturierter Text schließt weder potentiell unsichere Sprachkonstrukte aus noch sieht sie eine sichere Teilsprache vor. In der industriellen Praxis werden nicht anwendungsgerechte und nicht echtzeitfähige Sprachen wie C, C++ oder Java eingesetzt, und zwar nicht, weil es keine geeigneteren Sprachen gäbe, sondern weil es Markt und Kunden aus nicht sachgerechten Gründen so fordern. Dies zwingt dazu, die Unzulänglichkeiten dieser Sprachen in komplizierter, für Dritte kaum nachvollziehbarer und nicht portabler Weise durch Betriebssystemaufrufe, Assembler-Einschübe u.ä. wettzumachen. Zwar eliminiert bspw. die von der Firma ETAS entwickelte „Embedded Software Development Language“ (ESDL) [2] einige Fehlermöglichkeiten von C, sieht aber weder Echtzeitfähigkeit noch Orientierung an den Sicherheitsintegritätsstufen gemäß IEC 61508 vor.

Die Entwicklung der deutschen Echtzeitprogrammiersprache PEARL geht auf den Wunsch zurück, automatisierungs- und kerntechnische Anwendungen angemessen und mit hoher Produktivität formulieren zu können. Schon um das Jahr 1967 herum nahmen deshalb Ingenieure und Physiker von Firmen und Forschungsstellen die Arbeit an der Sprachspezifikation auf, die im April 1973 dann als erster Forschungsbericht des vom Bund geförderten Programms „Prozeßdatenverarbeitung“ (PDV) veröffentlicht wurde. Unmittelbar danach wurden erste

Übersetzer erstellt und begann auch die Normung von PEARL, wozu 1974 der Arbeitskreis DIN/FNI AK 5.8 gegründet wurde. Aus seinen Arbeiten gingen eine einfache und eine recht komplexe Version der Sprache hervor, die in den Teilen 1 und 2 der DIN 66253 als Basic PEARL 1981 und als Full PEARL 1982 genormt wurden. Daran schloss sich 1989 der Teil 3 Mehrrechner-PEARL an, der durch konzeptionelle Klarheit und Eleganz besticht und als Muster zur Strukturierung und Programmierung verteilter Systeme bis heute unerreicht ist.

Die in Tabelle 1 dargestellte und sicherlich nicht vollständige Zeittafel zur Geschichte von Programmiersprachen lässt auf den in der Konzeptionsphase von PEARL um 1970 herum herrschenden Zeitgeist schließen. Von den Anfängen der Entwicklung höherer Programmiersprachen 1953 an, als Backus die Idee zu Fortran äußerte, dauerte es nur 15 Jahre, bis mit Algol 68 eine Sprache definiert worden war, die sich durch überbordende Komplexität auszeichnet – woran sie wohl auch gescheitert ist. Daher nimmt es nicht Wunder, dass Full PEARL Eigenschaften wie bspw. dynamische Erzeugung und Hierarchisierung von Tasks vorsah. Derartige Konstrukte machen es praktisch unmöglich, zur Entwurfszeit das spätere Laufzeitverhalten sich dynamisch verändernder Mengen nebenläufiger Prozesse vorherzusagen und vor Inbetriebnahme zu verifizieren.

Andererseits wurde seinerzeit der Schritt von der Maschinen- bzw. Betriebssystemorientierung hin zur Problemorientierung noch nicht konsequent genug vollzogen. Das eklatanteste Beispiel dafür sind die in allen bisherigen PEARL-Versionen vorgesehenen Synchronisationskonstrukte niedrigsten Niveaus, nämlich Semaphore und Bolts, die schon sehr früh als häufigste Fehlerquelle der Echtzeitprogrammierung identifiziert wurden. Erst die neueste, 2018 normierte Version von PEARL schafft hier Abhilfe (vgl. [3], Abschnitt 2 und Tabelle 2).

Gründlich gelöst haben sich die Autoren von PEARL allerdings vom damals und großenteils auch heute noch vorherrschenden Denken in der Informatik, das sich im Fehlen des Zeitbezugs von Algorithmen als zentralem Konzept der Informatik manifestiert. Mit speziellen Datentypen für Zeitpunkte und -dauern sowie expliziten Einplanungsbedingungen in Abhängigkeit von sowohl absoluten als auch relativen Zeitangaben zeigt PEARL seine besondere Stärke. So benötigt man in der für die Programmierung eingebetteter Systeme derzeit am Weitesten verbreiteten, jedoch nicht echtzeitfähigen Sprache C mehrere Dutzend überflüssig komplizierter und semantisch intransparenter Anweisungen, um außerhalb der Ausdrucksfähigkeit der Sprache mittels Betriebssystemaufrufen die meistens auch nur durch relative Zeitangaben spezifizierte periodische Aktivierung einer Task einzuplanen, wozu in PEARL eine einzige, auch für Nichtfachleute unmittelbar verständliche und selbstdokumentierende Anweisung ausreicht.

Die Sprachversionen Basic und Full PEARL wurden nie in reiner Form realisiert. Implementierungen bewegten sich immer zwischen diesen Extremen. Die im Laufe von drei Jahrzehnten insbesondere im praktischen Einsatz gemachten Erfahrungen fanden 1998 in der Norm DIN 66253-2 ihren Niederschlag, mit der beide Versionen durch PEARL90 abgelöst wurden. Damit vollzog sich eine Entwicklung, die Biedenkopf mit dem Satz „Fortschritt ist der Weg vom Primitiven über das Komplizierte zum Einfachen.“ charakterisierte [1]. Hierbei ist in

Tabelle 1. Zeittafel zur Entwicklung von Programmiersprachen

Jahr	Programmiersprache	Entwickler	Beeinflusst von
1957	Fortran	Backus	A-2
1958	Algol 58	ACM/GAMM-Mitglieder	–
1958	Fortran II	IBM	Fortran
1960	Algol 60	Backus, Naur	Algol 58
1964	PL/I	IBM	Fortran, Algol 60
1965	Fortran IV	IBM/Fortran Working Group	Fortran II
1966	Algol W	Wirth	Algol 60
1966	Fortran 66	ANSI-Mitglieder	Fortran IV
1968	Algol 68	van Wijngaarden, Koster et al.	Algol 60
1970	PEARL	AEG, BBC, Siemens, GfK Karlsruhe	Algol 60, PL/I
1971	Pascal	Wirth, Jensen	Algol 58
1972	C	Ritchie	B, BCPL, Algol 60
1977	Basic PEARL	PEARL-Verein	PEARL
1978	Fortran 77	ANSI-Mitglieder	Fortran IV
1978	Modula-2	Wirth	Pascal
1980	Ada	Ichbiah, Honeywell Bull	–
1982	Full PEARL	PEARL-Verein	Basic PEARL
1983	C++	Stroustrup	C, Simula 67, Algol 68
1983	Ada 83	DoD/ANSI-Mitglieder	Ada
1988	Oberon	Wirth	Modula-2
1989	ANSI C (C89)	ANSI-Mitglieder	C, Algol 68
1989	Mehrrechner-PEARL	Steusloff	Bsic, Full PEARL
1991	Fortran 90	ANSI-Mitglieder	Fortran 77
1995	Ada 95	ISO/ANSI-Mitglieder	Ada 83
1997	Fortran 95	ISO/Fortran Working Group	Fortran 90
1998	ISO C++ 98	ISO Working Group	C++
1998	PEARL90	GI/GMA/ITG FA Echtzeitsysteme	Full PEARL
1999	ISO C 99	ISO Working Group	ISO C 95
2003	ISO C++ 2003	ISO Working Group	ISO C++ 98
2004	Fortran 2003	ISO/Fortran Working Group	Fortran 95
2007	Ada 2005	Ada Rapporteur Group	Ada 95
2011	ISO C++ 2011	ISO Working Group	ISO C++ 2003
2012	OpenPEARL	GI/GMA/ITG FA Echtzeitsysteme	PEARL90
2017	ISO C++ 2017	ISO Working Group	ISO C++ 2011
2018	SafePEARL	GI/GMA/ITG FA Echtzeitsysteme	PEARL90 und OpenPEARL

Assembler-Sprachen und Fortran das Primitive, in Algol 68 und Full PEARL das Komplizierte und in PEARL90 und dann SafePEARL das Einfache zu sehen.

Die Entwicklungsgeschichte von PEARL ist auch ein gutes Beispiel dafür, dass es keinesfalls leicht ist, zu einfachen Entwürfen zu gelangen – ganz im Gegenteil, denn, wie Biedenkopf [1] ebenfalls aufzeigt, „sind einfache Problemlösungen die schwierigsten: Sie verlangen hohe Innovationsleistungen und vollständige geistige Durchdringung der Sachverhalte." Beim derzeitigen Stand der Technik ist Einfachheit eine Grundvoraussetzung, um den Lizenzierungsinstitutionen die formelle Abnahme rechnergestützter Systeme für sicherheitsgerichtete Automatisierungsaufgaben zu ermöglichen.

2 SafePEARL

Leitmotiv der Neuformulierung der Norm DIN 66253 [3, 9] war das Bestreben, größtmögliche Korrektheit, Konsistenz und Genauigkeit der Darstellung bei geringstmöglichem Umfang zu erreichen. Aus diesem Grunde wurden die vier den Sicherheitsintegritätsstufen nach IEC 61508 zugeordneten Teilsprachen nicht einzeln beschrieben, sondern in Kapitel 5 der Norm durch eine Tabelle definiert, welche die auf den Sicherheitsstufen SIL 1 bis SIL 4 jeweils erlaubten Konstrukte angibt. Die Kapitel 6 bis 12 legen dann die Grundsprache zur Formulierung von für Einprozessorplattformen formulierten Automatisierungsanwendungen fest, während die Kapitel 13 und 14 die Sprachkonstrukte der weltweit bisher einzigen Norm (DIN 66253 Teil 3 „Mehrrechner-PEARL") einführen, die die Programmierung verteilter Systeme ermöglichen.

Die gemäß Abb. 1 ineinander geschachtelten Sprachteilmengen für jede der Sicherheitsanforderungsklassen SIL1 bis SIL4 nach DIN EN 61508 (VDE 0803) werden in Tabelle 2 dergestalt gebildet, dass die Verwendung weniger sicherer Sprachkonstrukte ausgehend von Anwendungen ohne Sicherheitsrelevanz („SIL0") mit steigendem Sicherheitsintegritätsniveau zunehmend restriktiver gehandhabt wird. Mit diesem Ansatz wird vermieden, für jede Sicherheitsstufe eine neue

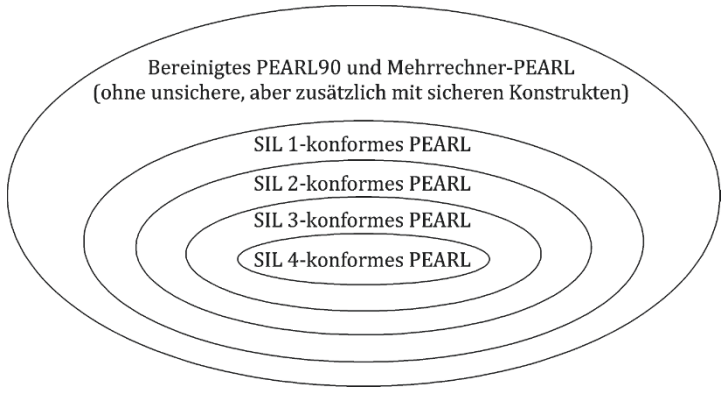


Abb. 1. Teilmengen der Echtzeitprogrammiersprache PEARL

Tabelle 2. Definition sicherheitsgerichteter Sprachteilmengen (+ zugelassen, – nicht erlaubt)

Anweisung/Klausel	**SIL0**	**SIL1**	**SIL2**	**SIL3**	**SIL4**
GOTO, EXIT	+	–	–	–	–
(bedingte) Ausdrücke und Zuweisungen	+	+	+	–	–
bedingte Anweisungen und Anweisungsauswahl	+	+	+	–	–
Angabe physikalischer Einheiten	+	+	+	+	+
Ursache-Wirkungstabellen	+	+	+	+	+
sequentielle Ablaufpläne	+	+	+	+	–
Synchronisierung mit SEMA- und BOLT-Variablen	+	–	–	–	–
Synchronisierung mit LOCK und TIMEOUT-Klausel	+	+	+	–	–
Verwendung interner Signale	+	+	+	–	–
Verwendung von Unterbrechungssignalen	+	+	+	–	–
Verwendung von Tasks	+	+	+	–	–
mit Prioritäten	+	+	–	–	–
mit Fristenangaben und Zeitüberwachung	+	+	+	–	–
(Funktions-) Prozeduraufrufe	+	+	+	+	–
Wiederholungen	+	+	–	–	–
mit MAXLOOP-Klausel	+	+	+	–	–
Verwendung von Zeigern und Referenzen	+	–	–	–	–
PUT/GET, WRITE/READ, CONVERT	+	+	+	–	–
TAKE/SEND	+	+	+	+	+
verteilte Systeme	+	+	+	+	+
dynamische Rekonfiguration	+	+	+	+	–
Botschaftenaustausch	+	+	+	–	–

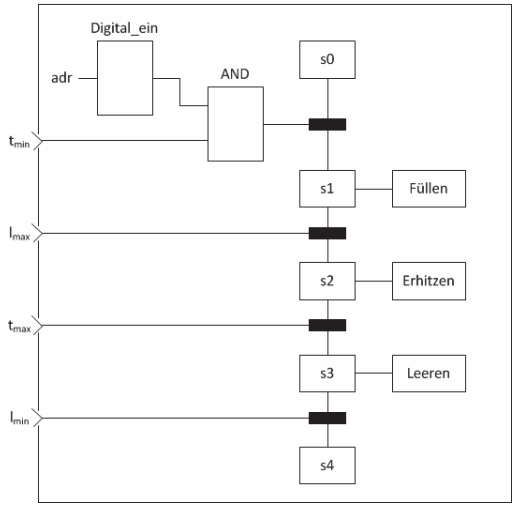
Sprache lernen zu müssen, und Übersetzern wird zu prüfen ermöglicht, ob Programme vorgegebene Sicherheitsauflagen erfüllen. Das Prinzip, Teilmengen einer Sprache für kritische Anwendungen zu definieren, gestattet es, Programme nach bestimmten Sicherheitsanforderungen zu entwickeln und Code für sicherheitskritische und -unkritische Systemteile nahtlos miteinander zu verbinden. Je sicherheitskritischer ein System ist, desto restriktivere Methoden sind einzusetzen.

In Automatisierungsprogrammen haben viele Variablen physikalische Bedeutungen. Bei Programmierung in herkömmlichen Sprachen sind ihre numerischen Werte mit den entsprechenden physikalischen Dimensionen allein in der Vorstellung der Programmierer verbunden. In verschiedenen Programmkomponenten nicht zueinander passende Einheiten stellen eine bedeutende Fehlerquelle dar, die schon zu Raketenabstürzen und dem Verlust von Satelliten geführt hat. Deshalb werden in SafePEARL Annotationen bereitgestellt, aus denen Übersetzer zwar keinen ausführbaren Code generieren, mit denen sie jedoch den korrekten Gebrauch physikalischer Dimensionen überprüfen können.

Das Problem nicht terminierender Wiederholungen wird mittels der Klausel MAXLOOP gelöst. Wenn die Anzahl von Schliefenwiederholungen die darin festgelegte Grenze überschreitet, wird die Schleifenausführung beendet und die nach dem Schlüsselwort EXCEEDING spezifizierte Anweisungskette ausgeführt, bevor der Kontrollfluss auf die erste Anweisung nach der Schleife übergeht.

Um sicheren Zugriff auf Betriebsmittel und die zeitliche Überwachung von Synchronisierungsoperationen zu gewährleisten sowie ihre Freigabe nach Verwendung zu erzwingen, wird die LOCK-Anweisung eingeführt. Ein dieses Konstrukt ausführender Prozess wartet solange, bis alle aufgelisteten Objekte in der angegebenen Weise belegt werden können. Durch die Bereitstellung der optionalen Wartezeitklausel kann die Wartezeit begrenzt werden. Lässt sich die Verriegelung nicht in der angegebenen Zeitspanne ausführen, so erfolgt eine vordefinierte Fehlerbehandlung. Nach erfolgreicher Verriegelung wird der kritische Anweisungsblock bearbeitet. Bei Beenden des Konstrukts werden die entsprechenden Freigaben durchgeführt. Vorzeitige Freigaben können mittels der UNLOCK-Anweisung veranlasst werden. Die optionale Ausführungszeitklausel begrenzt die Zeit, während derer ein Prozess in einer kritischen Region verweilen darf. Ihre Überschreitung wird als Systemausnahme behandelt.

Steuerungen von Prozessen werden häufig als Abfolgen von Schritten festgelegt, die nacheinander ausgeführt werden sollen. Um solche Ablaufsteuerungen zu beschreiben, definiert die Norm IEC 61131-3 die spezielle, unter Programmiersprachen einzigartige Ablaufplansprache zur Partitionierung von Programmorganisationseinheiten, d. h. Programmen und Funktionsblöcken, in Schritte und deren Verbindung entlang gerichteter Kanten mittels Transitionen. Mit jedem Schritt ist eine Menge von Aktionen und mit jeder Transition eine Übergangsbedingung assoziiert. Deshalb wurden in SafePEARL zusätzliche Sprachkonstrukte zur Formulierung sequentieller Ablaufsteuerungen definiert, die aus Sicherheitsgründen jedoch die Möglichkeit paralleler Abläufe und von Ablaufzyklen explizit ausschließen. Abb. 2 zeigt den graphischen und textuellen Ablaufplan einer Steuerung, gemäß derer ein Behälter auf Anforderung durch ein Tastsignal hin befüllt, sein Inhalt erhitzt und dann entleert werden soll.




```
SEQUENCE
STEP ENDSTEP;
TRANSITION Digital_ein(adr)
    AND Temperatur LE tmin;
STEP Call Fuellen ENDSTEP;
TRANSITION Fuellstand GE lmax;
STEP Call Erhitzen ENDSTEP;
TRANSITION Temperatur GE tmax;
STEP Call Leeren ENDSTEP;
TRANSITION Fuellstand LE lmin;
STEP ENDSTEP;
ENDSEQ;
```

Abb. 2. Ablaufplan einer Behältersteuerung