Cyber-Physical Systems – digital vernetzt in die physikalische Wirklichkeit

Erweiterte Kurzfassung

Manfred Broy

Institut für Informatik
Technische Universität München
D-80290 München Germany
broy@in.tum.de

Abstract: Cyber-Physical Systems im Sinne der acatech Studie (vgl. [Geisberger, Broy 12]) sind software-intensive Systeme, die unmittelbar mit der physikalischen Wirklichkeit verbunden sind, aber im Gegensatz zu klassischen eingebetteten Systemen tief und umfassend über globale Netzstrukturen, wie das Internet und damit verfügbare Daten und Dienste, vernetzt sind. Dadurch entstehen Systeme, die eng in die physikalische Wirklichkeit integriert sind, diese auch in das Internet transportieren und umgekehrt Daten und Dienste aus dem Internet nutzen. Informatiksysteme werden Teil der Wirklichkeit und spiegeln diese in ihren Konzepten unmittelbar wider. Sie bilden die technische Basis für das „Internet der Daten, Dienste und Dinge", „Big Data" und „Context Aware Cloud".

Neben die diskreten Konzepte der Informatik und qualitative Modelle auf Basis einer zweiwertigen Logik (tertium non datur) treten in Cyber-Physical Systems Unschärfe (engl. „Uncertainty") und quantitative Modelle. Dies hat drastische Auswirkungen auf die Modelle, Methodik und Vorgehensweisen der Informatik bei der Gestaltung solcher Systeme. Bisher zielten die Modelle der Informatik eher auf den Lösungsraum (die abstrakte Rechenmaschine) und weniger auf den Problemraum (die Welt der organisatorischen oder physikalischen Prozesse).

1 Konnektivität

Ein Phänomen ist die zunehmende Vernetzung unterschiedlicher Systeme. Anders als in den ersten Jahren der Informatik, als die Verbindung von Programmen zu ihrer Umgebung als so unwesentlich erachtet wurde, dass in der ansonsten richtungsweisenden Sprache Algol 60 darauf verzichtet wurde, die Ein/Ausgabe mit zu standardisieren, ist heute der interaktive Umgang mit Programmen unverzichtbar. Interaktion zwischen Programmen über Kommunikationsverbindungen ist wesentliches Merkmal heutiger Informatiksysteme.

Es erfordert spezifische Konzepte, um Phänomene der Kommunikation und Interaktion zu spezifizieren, zu modellieren und zu realisieren. Die alte Vorstellung, dass ein Programm, das in einem bestimmten Zustand gestartet wird, so lange rechnet, bis ein Endzustand erreicht ist, der das Ergebnis repräsentiert, entspricht dieser Sichtweise längst nicht mehr. Allerdings ist zu beobachten, dass selbst neuere Programmiersprachen nicht wirklich konsequent auf den Umstand ausgerichtet sind, dass Programme heute interaktiv und verteilt ablaufen und ein Großteil ihrer Funktionalität sich aus der Interaktion – mit Nutzern, mit der physikalischen Umgebung und mit anderen Systemen – ergibt. Dies erfordert konsequente Ausrichtung der Systeme auf Interoperabilität und auch eine Erweiterung theoretischer Konzepte wie des Begriffs der Berechenbarkeit (vgl. [Broy 07]).

Weitere Phänomene heutiger Systeme sind Multifunktionalität, Nebenläufigkeit, hohe Sicherheitsanforderungen und Themen der Safety und Security. Betrachtet man diese essentiellen Merkmale und Anforderungen heutiger Software, so stellt man fest, dass keiner dieser Aspekte in heutigen Programmiersprachen und methodischen Programmieransätzen in dem Umfang berücksichtigt ist, wie das erforderlich scheint.

2 Von Software zum System

Software ist immer mehr Teil umfassender Systeme (Ausführungsplattform und -umgebung). Systeme sind selbst wieder in einen operationellen Kontext (Nutzer, physikalische Umgebung, technische oder organisatorische Prozesse, weitere Software-intensive Systeme) eingebettet, mit dem sie interagieren. Dies erfordert in den Anforderungen und in der Architektur eine Berücksichtigung und systematischen Erfassung der Interaktion mit dem operationellem Kontext (Schnittstellenverhalten der Systeme und der Software) und der Annahmen über den operationellen Kontext (insbesondere über sein „Verhalten“, siehe auch Stichwort „Domänenmodellierung“, vgl. [Jackson 10]).

Software Engineering ist dann Teil eines Systems Engineering. Systeme sind eingebettet in umfassendere Systeme, abgegrenzt durch präzise festlegte Systemgrenzen, über die sie im Rahmen ihres Schnittstellenverhaltens mit ihrem operationellen Kontext interagieren.

3 Konsequenzen stärkerer „Physikalität“

Software Engineering war in seinen Anfängen vor mehr als 50 Jahren zunächst bestrebt physikalische Aspekte soweit möglich aus seinen Modellen, Spezifikationen und letztlich Programmiersprachen, abgesehen von einigen Spezialsprachen, zu verbannen, so dass etwa Fragen der Zeit bewusst weg abstrahiert wurden. Ziel war, Programmierparadigmen zu schaffen, die abstrakt, von der Ausführungsumgebung losgelöst Algorithmen als mehr oder weniger rein mathematische Gebilde erfassten. Die ersten Jahre der Informatik waren somit durch die Suche nach umfassenden Abstraktionen gekennzeichnet, die sich von den technischen Details einer Ausführungsumgebung lösen und möglichst abstrakte Programmierkonzepte anbieten. Nahezu alle herkömmlichen Programmiersprachen klammern bis heute das Thema Zeit konzeptionell aus und stellen Abstraktionen bereit, für die Zeit in Berechnungen keine explizite Rolle spielt. Dies reflektieren auch die Methoden und Theorien der Programmiersprachen und ihrer Semantik. Ziel ist von Zeit- und Ausführungsdetails zu abstrahieren und ein möglichst abstraktes „maschinenunabhängiges“ Konzept anzubieten, um über geeignete Abstraktion Portierbarkeit zu garantieren. Dies ändert sich mit der Entwicklung von Cyber-Physical Systems, in denen physikalischen Gesichtspunkten, unter anderem Zeit (vgl. [Broy 06]), ein dominierende Rolle zukommt.

Für die Realisierung zeitkritischer Systeme ist bisher eine ganze Reihe spezieller Modelle, Programmiersprachen und Paradigmen entwickelt worden, allerdings alle mit einer oft eigenartigen Vermischung aus Ausführungszeit und funktionalen Anforderungen in Hinblick auf die physikalische Zeit in dem operationellem Kontext, auf den das System wirkt. Zeitkonzepte werden bisher in Programme nicht direkt eingebracht, sondern eher implizit über Ausführungsplattformen und Laufzeitumgebungen. Das Zeitverhalten ist dann für Echtzeitanwendungen durch komplizierte Verfahren („Worst Case Execution Time“ WCET vgl. [Wilhelm, Grund 14]) bezogen auf die Ausführungsplattform zu ermitteln. Programmiersprachen werden um spezifische Konstrukte erweitert, die es erlauben auf Ausführungs- oder Systemzeit zuzugreifen, allerdings nur mit indirektem Bezug zu physikalischer Zeit im Sinn der Anwendungsdomäne.

Es stellt sich die Frage, ob es nicht konsequenter wäre, Modellierungskonzepte zu verwenden und Sprachen zu schaffen, die Zeit als explizites Konzept (vgl. [Broy 09]) enthalten und sich so von der Vorstellung lösen, dass das Zeitverhalten eines Programms Ergebnis der Ausführungsgeschwindigkeit der entsprechenden Hardware ist. Im Gegensatz dazu sind Programmiersprachen denkbar, die es erlauben, explizit funktionale Zeitanforderungen zu formulieren. Solche Programme können natürlich auf Hardware nur korrekt ausgeführt werden, wenn die Ausführungsgeschwindigkeit der Hardware ausreicht. Dieses Phänomen ist aber nicht neu. Gleiches gilt heute für Algorithmen allgemein. Denn wenn der Speicherplatz einer Maschine nicht ausreicht, ist das idealisierte Konzept eines Programms, das über Speicherplatzbeschränkungen ja keine Aussagen macht, auch nur korrekt ausführbar, wenn letztendlich genügend Speicherplatz zur Verfügung steht.

Die immer stärkere Präsenz von Physikalität im Kontext der Programme hat zur unvermeidlichen Konsequenz, Konzepte in der Modellierung und letztlich auch Modelle, Spezifikation und Programme zu entwickeln, die physikalische Realität direkt widerspiegeln. Das gilt nicht nur für die Modellierung von Zeit, sondern auch für weitere Gesichtspunkte wie Kausalität, Raum und Geometrie.

4 Diskretes und kontinuierliches Verhalten

Ein wesentlicher Unterschied vieler Modelle der physikalischen Realität zu den Modellen der Informatik besteht darin, dass Modelle der Informatik inhärent diskret und abstrakt sind und sich stark auf klassische Logikkonzepte stützen, wohingegen physikalische Modelle in aller Regel kontinuierliche Funktionen als Basis haben, wie sie unter anderem in der Regelungstechnik verwendet werden, um unmittelbar gängige Phänomene der physikalischen Realität zu modellieren. Heute gilt, dass kontinuierliche Funktionen auf digitalen Maschinen nicht analog umgesetzt werden, sondern über numerische Methoden diskretisiert approximiert, also doch wieder diskret ausgeführt werden. In den Modellierungsansätzen werden aber diese Funktionen völlig zurecht über kontinuierliche Modelle der Mathematik behandelt, da eine ganze Reihe typischer Methoden durch diese Modellbildung zur Verfügung stehen.

Allerdings ist bisher nicht konsequent verstanden und untersucht worden, inwieweit eine stärker logische Sicht, also nicht eine numerische approximierende Sicht, sondern eine diskrete Sicht, die Logik eines Systems, das über kontinuierliche Funktionen modelliert ist, beschreibt. Auch in kontinuierlichen Systemen sind logische Ereignisse identifizierbar (vgl. [Broy 12]). Aus dem kontinuierlichen Verhalten eines Systems lässt sich durch die Einführung entsprechender prägnanter Ereignisse ein diskretes Modell, etwa in Form einer Zustandsmaschine, gewinnen, das bei geschickter Wahl die Logik und damit die Funktionalität des Systems erfasst.

5 Unschärfe und unsicheres Wissen

Klassische Informatik und Algorithmik leben – abgesehen von der Numerik – in einer diskreten Welt. Algorithmen sind digital. Antworten auf Fragen sind eindeutig. Eine Bedingung ist wahr oder falsch. Ein Programm ist korrekt oder inkorrekt.

Aus den bisher beschriebenen Phänomenen bei der Modellierung physikalischer Wirklichkeit ergeben sich Unschärfe und unsicheres Wissen. Die Gründe sind vielfältig (vgl. [Fantl, McGrath 12]). Fehler, Unschärfe und Ungenauigkeiten in den Modellen („wann ist eine Verkehrssituation kritisch, so dass eine automatische Notbremsung ausgelöst wird“), in der Erfassung von Werten durch Sensoren („ist das Werkstück passgenau“), ihrer Interpretation („ist auf dem Videobild ein Kind zu erkennen“), in den Berechnungen mit approximierten Werten (Probleme der Fehlerfortpflanzung) sowie in der Hardware („liegt ein Übertragungs- Speicher- oder Berechnungsfehler vor“) und Software („ist die Anwendung korrekt modelliert“) führen zu Unsicherheit über die

Zuverlässigkeit von Systemen, die Gültigkeit von berechneten Werten und in Konsequenz zu unsicherem Wissen. Diese Unschärfe und Ungenauigkeiten akkumulieren sich in Systemen über die Zeit durch die Alterung von ohnehin unsicheren Daten (Berechnung des Standorts eines Quadrocopters in einem Gebäude ohne GPS-Zugriff unter fortschreibender Berechnung des Standorts aus Sensordaten und gespeicherten Daten).

Bedingungen über kontinuierlichen Funktionen sind immer kritisch, wenn sie etwa in Kontrollstrukturen abgefragt werden und nur kleine Unterschiede in den Werten für das Ergebnis der Bedingungsabfrage entscheidend sind. Durch Ungenauigkeit in der Repräsentation kontinuierlicher Funktionen und Berechnungen kann dies – wie aus der Numerik bekannt – zu verfälschten Ergebnissen führen. Anders ausgedrückt, wenn kleine Unterschiede in den Funktionswerten zu unterschiedlichen Ergebnissen bei Abfragen führen, haben wir es mit chaotischen Verhalten und damit mit Unsicherheit zu tun, inwieweit die Ergebnisse der Abfragen tatsächlich korrekt das widerspiegeln, was sie modellieren sollen, wenn mit zwangsläufig ungenauen Approximationen gearbeitet wird.

Schwerer wiegt, dass in vielen Fällen physikalische Modelle probabilistische Sichtweisen erfordern. Wahrscheinlichkeitskonzepte sind für viele Situationen der Modellierung technischer und physikalischer Phänomene unverzichtbar – gleichermaßen aleatorische Wahrscheinlichkeit als Folge eines stochastischen physikalischen Prozesses wie auch epistemische Wahrscheinlichkeit als Folge fehlenden Wissens über kausale Zusammenhänge und Hintergründe. Es wird notwendig, probabilistische Konzepte (vgl. [Neubeck 12]) in die Modellierung mit aufzunehmen, und zwar sowohl in die Domänenmodellierung, in die Spezifikation, aber auch in die Betrachtung von Programmverhalten mit probabilistischen Phänomenen ihres operationellen Kontexts. Eine besondere Herausforderung liegt in der Verwaltung von Daten unter Unschärfe, Unsicherheit und alternder Aktualität sowie der Errechnung von Prognosen unter Unschärfe und unsicherem Wissen als Voraussetzung für Reaktionen autonomer Systeme.

6 Quantitative und qualitative Modellierung

Logische und algebraische Ansätzen zur Spezifikation und diskreten Modellierung von Software führen auf logische, zweiwertige Konzepte etwa von Korrektheit. Ein Programm ist in einer qualitativen Modellierung entweder korrekt oder fehlerhaft – tertium non datur.

Software mit enger Interaktion mit der physikalischen Realität und mit menschlichen Nutzern erfordert aber „weichere“ Begriffe einer graduellen Korrektheit (vgl. [Chatterjee et al. 08]). Software ist bis zu einem bestimmten Grad korrekt, zuverlässig, sicher. Sind Maßzahlen oder Abstandsfunktionen (vgl. [Henzinger 10]) gegeben, dann kann ein Programm für eine gegebene Spezifikation „korrekter“ sein als ein anderes Programm. Probabilistische Spezifikationen fordern ein bestimmtes „korrektes“ Systemverhalten nur mit einer spezifizierten Wahrscheinlichkeit.

Dies erfordert neben den logischen, qualitativen Ansätzen quantitative Ansätze (vgl. [Broy 14]). Die Korrektheit von Software ist dann graduell (Software ist „in einem bestimmten Maß korrekt“). Praktikern ist das längst vertraut. Software ist korrekt unter bestimmten Annahmen, in einem bestimmten Grad, über einen bestimmten Zeitraum, in einem bestimmten Sinn, mit einer bestimmten Wahrscheinlichkeit.

Dies erfordert besondere Modellierungsansätze wie Probabilistik, Fuzzy Logics, Metriken oder Maßtheorie. Eine besondere Herausforderung ist die Präzisierung unscharfer Begriffe (wie Qualität, Privatheit, Nutzfreundlichkeit etc.) durch Maßzahlen und die Bewertung ihrer Aussagekraft für die Umsetzung in die Funktionalität von Software-Systemen.

7 Konsequenzen für Software, Systems und Requirements Engineering

Um Software Engineering, eingebettet in Systems Engineering, auf die Cyber-Physical Systems der Zukunft auszurichten, ist eine konsequente Erweiterung der Modelle notwendig. Das bedeutet keineswegs, dass wir uns von diskreten Modellen abwenden, denn die diskrete Sicht auf die Realität ist wesentlicher Teil der Stärke der Informatik und bringt völlig neue Erkenntnisse und Möglichkeiten, wie es die letzten Jahre eindrucksvoll gezeigt haben. Jedoch muss eine viel engere Verzahnung stattfinden zwischen den klassischen Modellen der Informatik, den diskreten Systemen, Schnittstellenkonzepten, den Zustandsmaschinen, den diskreten Abläufen, den Prozessbegriffen und den Modellen, die durch die Mathematik für die Physik geschaffen worden sind, wie kontinuierliche Funktionen und Probabilistik. Von besonderer Bedeutung sind letztlich Modelle von Zeit und Raum. Eine Informatik, die diese Konzepte harmonisch integriert und in einen Engineering-Ansatz umsetzt, ist das geeignete Vehikel, um die Cyber-Physischen Systeme der Zukunft zu erschließen.

Die Einführung von Zeit, Raum und Wahrscheinlichkeit von Unsicherheit und Unschärfe in die Modellierung Software-intensiver Systeme hat weitreichende Konsequenzen für den gesamten Begriffsapparat der Informatik und des Software Engineering sowie insbesondere des Requirements Engineering. Darüberhinaus sind das Grundlagen für zentrale Themen der Informatik wie „Human Centric Engineering“ (vgl. [Broy, Schmidt 14]), Big Data, verteilte, mobile, adaptive und autonome Systeme.

Danksagung

Ich danke meinen Kollegen Herrn Bernhard Schätz, Erhard Plödereder, Alexander Pretschner für inspirierende Diskussionen und kritische Anmerkungen.

Literaturverzeichnis

[Broy 06] M. Broy: The ‚Grand Challenge' in Informatics: Engineering Software-Intensive Systems. IEEE Computer, Oktober 2006, 72 – 80

[Broy 07] M. Broy: Interaction and Realizability, In: Jan van Leeuwen, Giuseppe F. Italiona, Wiebe van der Hoek, Christoph Meinel, Harald Sack, Frantisek Plasil (eds.): SOFSEM 2007: Theory and Practice of Computer Science, Lecture Notes in Computer Science 4362, Springer 2007, S. 29–50

[Broy 09] M. Broy: Relating Time and Causality in Interactive Distributed Systems. In: M. Broy, W. Sitou, T. Hoare (eds): Engineering Methods and Tools for Software Safety and Security, NATO Science for Peace and Security Systems, D: Information and Communication Security, Vol. 22, IOS Press 2009, 75-130

[Broy 12] M. Broy: System Behaviour Models with Discrete and Dense Time. S. Chakraborty, J. Eberspächer (eds.): Advances in Real-Time Systems, Springer Berlin Heidelberg, 2012, 3-25

[Broy 14] M. Broy: Formalizing and Relating System and Software Properties: Their Dependency and Independency. Unpulished Manuscript

[Broy, Schmidt 14] M. Broy, Albrecht Schmidt: Challenges in Engineering Cyber-Physical Systems. IEEE Computer Society, Ausgabe Februar 2014, S. 70-72

[Chatterjee et al. 14] K. Chatterjee, L. Doyen, Th. A. Henzinger. Quantitative languages. Proceedings of the 17th International Conference on Computer Science Logic (CSL), Lecture Notes in Computer Science 5213, Springer, 2008, 385-400

[Fantl, McGrath 08] J. Fantl, M. McGrath: Knowledge in an Uncertain World. Oxford University Press, 2012

[Geisberger, Broy 12] E. Geisberger, M. Broy: agendaCPS – Integrierte Forschungsagenda Cyber-Physical Systems (acatech STUDIE), Heidelberg u.a.: Springer Verlag 2012

[Henzinger 10] Th. A. Henzinger: From Boolean to Quantitative Notions of Correctness. POPL '10 Proceedings of the 37th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 2010, 157-158

[Jackson 10] M. Jackson: The Operational Principle and Problem Frames. In: C. B. Jones, A. W. Roscoe, K. R. Wood (eds.): Reflections on the Work of C. A. R. Hoare, Springer Verlag, London, 2010

[Neubeck 12] P. R. Neubeck: A Probabilitistic Theory of Interactive Systems. Technische Universität Münschen, Fakultät für Informatik, Dissertation 2012

[Wilhelm, Grund 14] R. Wilhelm, D. Grund: Computation Takes Time, but how much? Commun. ACM 57(2), 2014, 94-103