Ein Wegweiser für Forschung und Lehre im Software-Engineering eingebetteter Systeme

Manfred Broy · Wolfgang Pree

Bisher wurde das Gebiet der Software für eingebettete Systeme („embedded systems") von der Informatik zu wenig beachtet. Die Softwareentwicklung für eingebettete Systeme konzentriert sich in der konventionellen Herangehensweise darauf, isolierte Kontroll- und Regelungsaufgaben zu behandeln und dabei die vorhandenen Hardwareressourcen bei Garantie der Echtzeitanforderungen optimal auszunutzen. Das lässt wenig Freiraum für Softwareabstraktionen, die in anderen Anwendungsbereichen zur besseren Beherrschbarkeit, zur Rationalisierung der Entwicklung und zur Verbesserung der Softwarequalität geführt haben. Das sich positiv verschiebende Preis-Leistungs-Verhältnis bei der Hardware und die steigenden Anforderungen an die Funktionalität der Software fordern neuartige Methoden. Wir zeigen auf, welche Forschungsthemen und Chancen sich für die Informatik aufgrund geänderter Rahmenbedingungen auftun. Die Vision ist, durch Forschung den Weg von einer hardwarezentrierten Entwicklung eingebetteter Systeme zu einer anforderungszentrierten Entwicklung zu finden. Das muss sich auch in der Ausbildung widerspiegeln, wo wir eine Kombination von Softwaretechnik und von formalen Grundlagen der Informatik mit ausgewählten Gebieten der Regelungs- und Elektrotechnik für notwendig erachten.

1 Embedded Software als zukunftsträchtiges Gebiet

Ein eingebettetes System ist nach Simon [5] ein Computersystem, das in einem anderen Produkt als einem Computer verborgen ist. Das Spektrum eingebetteter Systeme reicht von Mobiltelefonen über Antiblockiersysteme bei Fahrzeugen bis hin zu Satellitensteuerungen. Heute werden etwa 98% der programmierbaren CPUs in eingebetteten Systemen verwendet. Der Aufwand bei der Entwicklung eingebetteter Systeme verlagert sich zunehmend von der Hardware hin zur Software. Durch den bisherigen Fokus auf Hardware ist die Software bei Änderungen der Hardware aufwendig anzupassen. Dabei müssen Entwickler aufgrund steigender Funktionalitätsanforderungen eine deutlich höhere Komplexität meistern. Eine systematische, wenn möglich automatisierte Konstruktion von Software für eingebettete Systeme stellt daher eine bemerkenswerte Herausforderung für die Informatik dar.

Aus folgenden Gründen erachten wir Embedded Software-Engineering als zukunftsträchtiges, fruchtbares und relevantes Gebiet für Forschung und Lehre in der Informatik:

- Es besteht eine signifikante technologische und methodische Lücke bei der Entwicklung zwischen eingebetteter und sonstiger Software, die es zu schließen gilt. Ein Hauptgrund dafür sind die ökonomischen und organisatorischen Rahmenbedingungen, unter denen eingebettete Systeme hergestellt werden: Die Massenproduktion erfordert ein rigides Sparen bei den Hardwareressourcen. Das lässt bisher keinen Spielraum für Softwareabstraktionen, die typischerweise mehr Speicher und Rechenleistung erfordern. Diese Situation hat sich in den letzten Jahren dadurch zu ändern begonnen, dass das Preis-Leistungs-Verhältnis von Hardware ein Niveau erreicht hat, welches ein nahezu optimales Ausnutzen der Ressourcen nicht mehr zu einem ökonomischen Imperativ macht. Beispielsweise wurden bei Satellitensteuerungen bisher Prozessoren eingesetzt, die auf dem Mil-Std 1753 basieren und 64 KB Speicherbereich haben. Künftige Systeme werden voraussichtlich den ERC32-Prozessor verwenden, der als SPARC-Pro-

Manfred Broy
Technische Universität München, D-80290 München
E-Mail: Broy@informatik.tu-muenchen.de

Wolfgang Pree
Universität Salzburg, A-5020 Salzburg
E-Mail: Pree@SoftwareResearch.net

zessor für weltraumtauglich qualifiziert wurde und über beträchtlich mehr Speicher und Rechenleistung verfügt. Eine analoge Entwicklung ist in diversen Anwendungsbereichen eingebetteter Systeme zu beobachten beziehungsweise zu erwarten.

- Da die Entwicklung eingebetteter Systeme ihre Wurzeln in der Elektro- und Regelungstechnik hat und Software dabei früher kaum eine Rolle spielte, hat bisher die Hardware Priorität vor der Software. Entsprechend wird heute Software noch konsequent auf eine spezifische Hardwareplattform zugeschnitten. Ändert sich die Hardware, ist die Software mühsam anzupassen oder gar neu zu schreiben, um wiederum die Hardware möglichst gut auszunützen. Das führt in weiterer Folge dazu, dass Elektro- und Regelungstechniker auf der einen Seite und Softwareentwickler auf der anderen Seite weitgehend voneinander getrennt an die Entwicklung herangehen. Die Hardwarelastigkeit der Entwicklung mag ein weiterer Grund dafür sein, warum das Gebiet für Softwareingenieure bisher wenig attraktiv scheint. Der Teufelskreis schließt sich dadurch, dass Elektro- und Regelungstechniker fachbedingt wenig Expertise auf Seiten der Software einbringen, um etwa die Softwareentwicklung zu rationalisieren oder um Softwarekomponenten wiederzuverwenden.
- Zunehmende Komplexität: Aufgrund der Fortschritte bei der Hardware gemäß der Prognose von Gordon Moore [3] werden eingebettete Systeme die zusätzlichen Ressourcen nutzen und mehr an Funktionalität bereitstellen, was wiederum die Komplexität erhöht.

Ein Großteil der zusätzlichen Funktionalität wird durch Software abgedeckt. Ein weiterer Faktor, der die Komplexität erhöht, resultiert aus der Verteilung von Rechnerknoten und neuen Anforderungen, wie beispielsweise die, Software zur Laufzeit umkonfigurieren zu können. Bisher sind viele eingebettete Systeme Einzellösungen. Die Vernetzung eingebetteter Systeme erfordert eine umsichtige Konzeption, insbesondere um das Zeitverhalten in einem verteilten, dynamisch konfigurierbaren System vorhersehbar und damit beherrschbar zu machen.

Allgemein werden Softwaremethoden und -werkzeuge zur Bewältigung der Komplexität nötig, die auf die diversen Anwendungsgebiete von eingebetteten Systemen abgestimmt sind. Nur so lassen sich die Qualitätsanforderungen, insbesondere für sicherheitskritische Systeme, erfüllen.

Als unabdingbare Voraussetzung dafür, die skizzierten Hürden zu überwinden, erachten wir die entschlossene Umorientierung von der hardwarezentrierten hin zu einer anforderungszentrierten Entwicklung von eingebetteten Systemen. Das ist analog zum Einsatz höherer Programmiersprachen zu sehen, die von Details der darunter liegenden Hardware abstrahieren. Es ist heute unvorstellbar, kommerzielle Anwendungen auf eine bestimmte Prozessorarchitektur zuzuschneiden. Durch Schichtenarchitekturen werden Hardware und Software entkoppelt. Vergleichbare Abstraktionen für eingebettete Systeme fehlen. Der Weg zu einer anforderungszentrierten Entwicklung von eingebetteten Systemen wird möglich und ökonomisch, da ausreichende Rechnerressourcen zu einem günstigen Preis zur Verfügung stehen und somit ein Spielraum für Softwareabstraktionen entsteht. Das erfordert, wie im nachfolgenden Abschnitt skizziert, intensive Forschung im Bereich Software, die über die bei der Entwicklung konventioneller Systeme vorhandenen Erkenntnisse hinausgehen muss. Es gilt beispielsweise zu erforschen, wie Komposition und damit Wiederverwendung von Softwarekomponenten unter den spezifischen Rahmenbedingungen eingebetteter Systeme methodisch möglich wird.

Um bereits in den nächsten Jahren verwertbare Ergebnisse zu erzielen, sollten sowohl in der Forschung wie in der Lehre Synergien zwischen Softwaretechnik, formalen Grundlagen der Softwarekonstruktion sowie Elektro- und Regelungstechnik angestrebt werden. Wir schlagen als Oberbegriff dafür den Begriff „Software-Engineering eingebetteter Systeme“ vor.

Aus Sicht der Ausbildung sollten Software- und Hardwarethemen adäquat in einem Curriculum kombiniert werden. Eine solide Ausbildung wird zum Teil auf einer Neukonzeption von Lehrinhalten beruhen (s. Abschnitt 3).

2 Forschungsagenda für Software-Engineering eingebetteter Systeme

Um die Forschungsthemen besser einschätzen zu können, skizzieren wir vorab eine heute übliche Klassifizierung von eingebetteten Systemen. Viele eingebettete Systeme sind sog. Echtzeitsysteme: Dabei kommt es nicht nur darauf an, dass die berechneten Ergebnisse logisch korrekt sind, sondern pünktlich zu einem bestimmten Zeitpunkt (Deadline) zur Verfügung stehen. Echtzeitsysteme, die noch akzeptabel funktionieren, obwohl eine Dead-

line geringfügig versäumt wurde, sind „weiche“ Echtzeitsysteme. Meist wird bei Nichteinhalten der Deadlines bei weichen Echtzeitsystemen die Qualität der Ergebnisse und der erbrachten Dienste schlechter, aber die Funktionalität wird noch bereitgestellt. Ein Beispiel ist die Dekompression von Videodaten, die über ein Netzwerk geladen werden. Je langsamer die Dekompression erfolgt, desto schlechter ist die Qualität des angezeigten Videos. Dies führt auf den Begriff des „Quality of Service“.

Bei sog. harten Echtzeitsystemen müssen hingegen die Deadlines unter allen Umständen eingehalten werden, da sonst eine Katastrophe eintreten könnte. Beispiele für harte Echtzeitsysteme sind Flugzeugsteuerungen, Motormanagement, Bremsassistent oder der Auslöser eines Airbags.

Der Bau von harten Echtzeitsystemen unterscheidet sich fundamental von weichen Echtzeitsystemen: Ein hartes Echtzeitsystem muss unter allen möglichen Ausnahme- und Fehlerbedingungen die Deadlines einhalten. Die Umgebung, in der ein hartes Echtzeitsystem eingesetzt wird, gibt vor, welches Zeitverhalten verlangt wird und welcher Grad an Parallelität beziehungsweise Verteilung auf Seiten der Software zu beherrschen ist. Beispielsweise definieren die Charakteristika eines Verbrennungsmotors die Anforderungen an die zugehörige Steuerungssoftware, also welche parallel ablaufenden chemischen und mechanischen Prozesse im Motor beobachtet werden müssen, in welchen Zeitabständen, z. B. vorgegeben durch die Motordrehzahl, die Messungen durch Sensoren und die Steuerungsimpulse durch Aktuatoren erfolgen müssen. Die Steuerungssoftware muss innerhalb der Zeitvorgaben die Berechnungen durchführen, um die Stellwerte an die Aktuatoren zu liefern, sodass das erwünschte Verhalten des Motors erzielbar ist.

Da aufgrund der Sicherheitsaspekte eine systematische Softwarekonstruktion für harte Echtzeitsysteme erstrebenswert ist, sind Forschungsergebnisse für diese Kategorie eingebetteter Systeme besonders nützlich. Einige der nachfolgend skizzierten Forschungsthemen sind daher speziell für harte Echtzeitsysteme relevant:

- Automatische Übereinstimmung der Zeitachse der Umgebung (reale Welt) mit der Zeitachse der Rechnerplattform: In der realen Welt ist Zeit ein Kontinuum, während digitale Rechner mit diskreter Zeit arbeiten. Da Rechnerplattformen für eingebettete Systeme zunehmend verteilt werden, ist es mühsam und fehleranfällig, die Übereinstimmung der beiden Zeitachsen quasi händisch zu bewerkstelligen. Das ist in der heute praktizierten Entwicklung der Fall. Stattdessen ist es erwünschenswert, eine automatische Übereinstimmung der genannten Zeitachsen zu gewährleisten. Eine profunde formale Betrachtung des Problems kann die theoretischen Grundlagen liefern, z. B. um herauszufinden, was notwendig ist, um das nichtdeterministische (Zeit-)Verhalten heutiger Systeme zu bereinigen. Evtentuell sind neue Sprachen notwendig, die sich ausschließlich darauf konzentrieren, das gewünschte Zeitverhalten von eingebetteten Systemen auf verteilten Plattformen durch automatische Erzeugung von Code sicherzustellen. Beispiele für Ansätze in diese Richtung sind Esterel [6] und Giotto [2].
- Verbesserung der Wiederverwendbarkeit: Das erfordert unter anderem eine adäquate Modularisierung, bei der Funktionalität und die plattformspezifische Reaktivität (Scheduling Code) getrennt werden. Bekannte Ansätze, um wiederverwendbare Software zu schaffen, wie objektorientierte Frameworks, müssen an die spezifischen Rahmenbedingungen von eingebetteten Systemen angepasst werden [4]. Beispielsweise sind Garbage Collection, dynamische Bindung und rekursive Objektstrukturen bei harten Echtzeitsystemen problematisch. Sprachen wie Giotto, die sich auf das Zeitverhalten konzentrieren, lassen wiederum keine Methodenaufrufe über die in ihnen definierten logischen Konstrukte hinaus zu. Um eine Komposition von Teilsystemen zu ermöglichen, müssen Komponentenstandards diese Rahmenbedingungen berücksichtigen und zusätzlich um das Zeitverhalten erweitert werden.
- Verbesserung der Methoden und Werkzeuge für die Spezifikation und Simulation dynamischer Modelle sowie die daraus resultierende Codegenerierung: Die zur Zeit von Regelungs- und Elektrotechnikern benutzten visuell-interaktiven Werkzeuge verfügen nicht über die in gängigen Programmiersprachen angebotenen Konstrukte zur Strukturierung. Weiterhin werden oft in einer Ad-hoc-Art verschiedene Ausführungsparadigmen, wie Datenfluss und imperative Konstrukte oder diskrete und kontinuierliche Modellierungselemente, kombiniert. Eine grundlegende Neukonzeption solcher Werkzeuge ist unabdingbar.
- Formale Zertifizierung sicherheitskritischer Echtzeitsysteme: Zurzeit beschränkt sich Qualitätssicherung auf die Einhaltung von Entwicklungsprozessen und auf umfangreiches Testen. Ein Forschungsziel ist es, die formale Spezifikation von Modellen und die formale Verifikation so weit praktikabel zu machen, dass zumindest kritische Softwarekomponenten verifiziert werden können. Da das Zeitverhalten eingebetteter Software ein typisches Merkmal ist, kann evtl. ein Fokus aus formaler Sicht auf diesen Aspekt die erwünschten Fortschritte bringen.

Zusammenfassend ist anzumerken, dass die Automatisierung der Softwareentwicklung für eingebettete Systeme eine Anpassung und Erweiterung der aus der Softwaretechnik bisher bekannten Konzepte erfordert und dass das außerdem Hand in Hand mit einer Anpassung und Integration formaler Methoden erfolgen muss. Als Ergebnisse sind beispielsweise neue Systemmodelle, Programmiersprachen und Komponentenstandards für eingebettete Software sowie Methoden und Werkzeuge zu erwarten, die in anderen Bereichen der Softwareentwicklung nicht bekannt sind. Einen guten Überblick zu aktuellen Forschungsarbeiten gibt der Konferenzband zum ersten Workshop „Embedded Software“ [1].

3 Software-Engineering eingebetteter Systeme in der Lehre

Die Ausbildung in Embedded Software-Engineering soll analog zur Forschung eine ausgeglichene Kombination aus Grundlagen der Informatik, Software-Engineering, formalen Methoden sowie Regelungs- und Elektrotechnik darstellen, wobei die Fortschritte auf dem Weg hin zu einer softwarefokussierten Entwicklung zu berücksichtigen sind. Das wird bisher nicht angeboten. Anzustreben ist ein Masterprogramm in Software-Engineering eingebetteter Systeme, das auf einem BSc in Software-Engineering eingebetteter Systeme aufbaut. Folgende stichwortartig aufgelisteten Inhalte erachten wir als relevant:

- Grundlagen des Entwurfs und der Entwicklung von Software: Ausführungsmodelle, Sprachen zur Modellierung von Echtzeitsystemen (Esterel, Lustre, Giotto); Architektur von Embedded Systems; Entwicklungswerkzeuge, UML-RT.
- Echtzeitbetriebssysteme: Charakteristika, Task-Management; Scheduling (EDF, RMA, dynamisches versus statisches Scheduling); Device-Treiber; Environment Triggering; Worst-Case Execution Time; Fault Tolerance; Echtzeitkommunikation (worst case latency, flow control); Fallstudien (VxWorks, OSEK, HelyOS).
- Konzepte der Signal- und Systemtheorie: formale Repräsentation von Systemen (Systeme als Separatoren, die Inputsignale in Outputsignale umwandeln; Blockdiagramme; Signalflussdiagramme); ereignisgesteuerte Systeme; nichtlineare Systeme; Stabilität; synchrone und asynchrone Kommunikation; nichtlineare Phänomene: Chaos, Bifurkation, digitale Signalverarbeitung.
- Regelungssysteme: Prozessautomatisierung; Konzepte von Steuerungen; Modellierung dynamischer Systeme; Closed Loop Control; Feedback; PID Control; Feed-Forward-Kompensation; Cascade Control; Systeme mit mehreren Variablen (Transfer Matrix; State-Space-Repräsentation); Multi-Loop Control; Open Loop Process Control. Signal- und modellbasierte Fehlererkennung und -isolierung; Überwachung in Steuerungssystemen; adaptive Steuerungssysteme; Werkzeuge zur Modellierung von Steuerungssystemen (z. B. Matlab/Simulink).
- Softwarearchitekturen für Embedded Software Systems: Modularisierung; Komponentenstandards; Erweiterungen von Komponentenstandards um Zeitverhalten; Time-triggered-Architekturen (TTP, FlexRay) versus Event-triggered-Architekturen.
- Formale Methoden: Berechnungs- und Systemmodelle; formale Spezifikation und Verifikation; formale Sprachen und attributierte Grammatiken; Automatentheorie; Theorie verteilter interaktiver Systeme.
- Hardwareprototyping: Field-Programmable Gate Arrays (FPGAs): Transistoren und Gates; Combinational Circuits; Latches und Register; Bussysteme; Speicher; formale Beschreibung von synchronen Schaltkreisen; Schnittstellen zwischen asynchronen Einheiten.
- Verteilte Systeme: Konzepte und Charakteristika; Interprozesskommunikation; Algorithmen zur Koordination von Aufgaben; Transaktionen; Nebenläufigkeit; Distributed Shared Memory; Sicherheit; verteilte Dateisysteme; Protokolle; Fallstudien verteilter eingebetteter Systeme.

Diese Inhalte werfen nicht nur Fragen für die Lehre, sondern auch für die Forschung auf. Gerade dies verspricht einen fruchtbaren Austausch zwischen Forschung und Lehre.

4 Software-Engineering eingebetteter Systeme: Neuland für die Informatik

Unser Papier skizziert, welche Chancen und Herausforderungen sich bei eingebetteten Systemen für die Informatik ergeben. Anders formuliert können eingebettete Systeme als ein weitgehend weißer Fleck auf der Landkarte der Informatik und Software gesehen werden, wo es heute noch möglich ist, bahnbrechende Beiträge zu liefern. Es ist vorstellbar, dass Softwareabstraktionen und Sprachen für eingebettete Systeme entstehen, die sich in den kommenden Jahrzehnten als ähnlich wirkungsvoll erweisen wie beispielsweise die Objektorientierung.

Zudem ist der Bau von eingebetteten Systemen eine spannende Herausforderung. Beispielsweise haben eine Reihe von Forschungsgruppen versucht, Steuerungssoftware für autonom fliegende Helikop-

ter zu bauen, wobei nur wenige Projekte erfolgreich abgeschlossen werden konnten.

Der Bau von eingebetteten Systemen erfordert eine systematische, ingenieurmäßige Herangehensweise, die nicht auf „Versuch und Irrtum“ basieren kann. In Zukunft wird der Erfolg der Forschungsarbeiten auf diesem Gebiet daran messbar sein, wie viel im Vergleich zum heutigen Stand der Technik automatisiert wird. Das Potenzial für Rationalisierungen sowie Qualitätsverbesserungen und damit für erfolgreiche Forschung schätzen wir für hoch ein.

Literatur

1. Henzinger, T.A., Kirsch, C.M. (eds.): EMSOFT 01: Embedded Software. Lecture Notes in Computer Science 2211. Berlin Heidelberg New York: Springer 2001; im Web unter http://link.springer-ny.com/link/service/series/0558/tocs/t2211.htm; s. auch http://www.emsoft.org
2. Henzinger, T.A., Horowitz B., Kirsch, C.M.: Giotto: a time-triggered language for embedded programming. In: Henzinger u. Kirsch 2001 [1]
3. Moore, G.: Interviews in Scientific American (1997); im Web unter http://www.sciam.com/interview_directory.cfm
4. Pree, W., Pasetti, A.: Embedded software market transformation through reusable frameworks. In: Henzinger u. Kirsch 2001 [1]
5. Simon, D.: An embedded software primer. Addison-Wesley 1999
6. Web-Referenz: http://www.esterel-technologies.com