Software Plattform Embedded Systems (SPES) 2020

Deliverable D 5.1 A

Version: 1.0

Projektbezeichnung	SPES 2020	
Verantwortlich	FIRST	
QS-Verantwortlich	OFFIS, Siemens	
Erstellt am	9. November 2009	
Zuletzt geändert	9. November 2009	
Freigabestatus		Vertraulich für Partner:
		Projektöffentlich
	X	Öffentlich
Bearbeitungszustand:		in Bearbeitung
		vorgelegt
	X	fertig gestellt

Weitere Produktinformationen

Erzeugung	Andreas Schramm
Mitwirkend	

Änderungsverzeichnis

Anderung			Geänderte	Beschreibung der Anderung	Autor	Zustand
Nr.	Datum	Version	Kapitel			
1	9. November 2009	1.0	Alle	Initiale Produkterstellung	Andreas Schramm	fertig

Kurzfassung

Elimination von funktionellem oder auch nur zeitlichem Nichtdeterminismus in Modellierungsverfahren für parallele Echtzeitsysteme erfordert zusätzliche Regeln und/oder Ausdrucksmittel zur semantischen Einschränkungen von Modellierungen.

Wir präsentieren eine Ausdrucksmittel, die auf der Trennung zeitlicher von struktureller Koordinationsaspekte der Systemkomponenten basieren. Sie erreichen eine höhere Flexibilität als herkömmliche ganzheitliche datenparallele Modelle, jedoch letztlich nicht immer uneingeschränkt deren Stringenz.

Inhaltsverzeichnis

1 Einordnung und Kurzbeschreibung

1.1 Motivation und Einordnung

Das vorliegende Dokument präsentiert eine Konzeption zur Elimination von zeitlichem und funktionalem Nichtdeterminismus in Modellierungsverfahren zur Spezifikation von parallelen Echtzeitsystemen.

1.2 Management Summary

Eine Einschränkung von zeitlichem Nichtdeterminismus in Modellierungsverfahren zur Spezifikation paralleler Echtzeitsysteme kann nicht ohne die gleichzeitige Betrachtung des funktionellen Nichtdeterminismus betrachtet werden; beides kann nur durch geeignete Einschränkung der Modellierungssprache erreicht werden.

Wir schlagen vor, dazu die zeitliche Koordination der Systemkomponenten von der strukturellen zu separieren und präsentieren eine entsprechende Architekturkonzeption.

Sie erreicht eine höhere Flexibilität als herkömmliche ganzheitliche datenparallele Modelle, jedoch letztlich nicht immer uneingeschränkt deren Stringenz, so daß die Abwägung zwischen Vor- und Nachteilen letztlich von der konkreten Anwendung bestimmt wird.

1.3 Überblick

Nach einer Motivation der Aufgabenstellung in Abschnitt 2 erfolgt in Abschnitt 4 zunächst eine kritische Diskussion und in Abschnitt 5 eine an deren Ergebnisse angepaßte Neufassung der ursprünglichen Aufgabenstellung. Dabei wird erarbeitet, daß eine Einschränkung von zeitlichem Nichtdeterminismus nicht ohne die gleichzeitige Betrachtung des funktionellen Nichtdeterminismus betrachtet werden kann. Beides ist weiterhin

nicht durch Erweiterungen der Ausdrucksfähigkeit der Modellierungssprache zu erreichen, sondern – im Gegenteil – nur durch ihre geeignete Einschränkung.

Nach einer Diskussion von Strukturierungskonzepten wird als Resultat in Abschnitt 9 eine eigene Konzeption für eine den Nichtdeterminismus reduzierende, wenn nicht vermeidende, Architektur gemacht, dessen Eigenschaften in einem Fazit in Abschnitt 10 diskutiert und bewertet werden.

2 Motivation

Echtzeitanforderungen an eingebettete Systeme umfassen einen weiten Bereich.

Sie reichen von weichen Anforderungen, wie der Sicherstellung einer angemessenen Reaktionszeit auf eine Benutzereingabe, bis hin zu sicherheitskritischen harten Echtzeitanforderungen, deren Missachtung eine Gefährdung für Mensch und Maschine zur Folge hat.

Vor allem die unbedingte Einhaltung letzterer ist daher mit allen Mitteln sicherzustellen.

Dies erfordert (teilweise je nach den näheren Umständen):

- Bereitsstellung ausreichender Rechenkapazität, z.B. durch den Einsatz moderner Multi-Core-Prozessoren, um die für eine benötigten Systemreaktion erforderlichen Berechnungen überhaupt rechtzeitig abschließen zu können,
- die Formulierung von Softwarekomponenten speziell im Hinblick auf die unterliegende Hardware-Architektur, um meist zunächst nur implizit vorhandenen Parallelität tatsächlich nutzen zu können,
- Informationen über die Laufzeit der beteiligten Prozesse und das Scheduling auf den Rechnerknoten, ohne die eine Garantie für die Einhaltung der Reaktionszeiten unmöglich wäre,
- neue Methoden zur Spezifikation, Modellierung und Analyse paralleler Echtzeitsysteme und entsprechende Werkzeuge, mit denen die parallelisierungsbedingt stark ansteigende Komplexität – besonders im Hinblick auf eine Zertifizierbarkeit – beherrschbar gemacht werden kann.

Zusätzlich umfassen die meisten eingebetteten Echtzeitsysteme sowohl diskrete als auch kontinuierliche Aspekte, so daß die Notwendigkeit geeigneter Ausdrucks- und Analysemittel für die Kombination beider Aspekte (Stichwort "*Regelungstechnik*") besteht.

In bestehenden Programmiermodellen für parallele und verteilte Systeme wird durch das Scheduling ein Nichtdeterminismus bezüglich des zeitlichen Verhaltens verursacht, der im Hinblick auf eine Garantie für die Einhaltung von Reaktionszeitanforderungen stark unvorteilhaft ist.

Aufbauend auf dem in Task ZP-Task 2.3 erarbeiteten Ansatz zur Spezifikation von Echtzeitanforderungen, wird daher im Rest dieses Papiers die Frage der Elimination von zeitlichem Nichtdeterminismus in den vorgeschlagenen Modellierungsverfahren untersucht, wobei der Versuch unternommen wurde, die Anforderungen der Anwendungsgebiete bezüglich der Zielplattformen (Hardware und Betriebssystem) weitestmöglich zu berücksichtigen.

3 Untersuchte parallele Programmiersysteme

FOCUS [BS01] [Bro05] [BKPS07] [Bro97] [HHR09] ist eine Modellierungssprache auf der Grundlage asynchroner Nachrichtenkommunikation und diskreter globaler Zeit.

Rich Components [DVM+05] [JMM08] bauen auf der weit verbreiteten und akzeptierten *Unified Modelling Language (UML)* auf. Sie erweitern ihren Begriff der *Schnittstelle (Interface)* um dynamische Apekte, die in sogenannten *Contracts* formuliert sind. Zur Verhaltensspezifikation werden hybride bzw. zeitbehaftete Zustandsautomaten eingesetzt, deren Aktionen in einer `C`-ähnlichen Sprache notiert werden können. Neben diskreten Ereignissen sind kontinuierliche Datenflüsse beschreibbar.

Mechatronic UML [GBSO04] [GB03] [BGT05] ist eine UML-Erweiterung speziell im Hinblick auf mechatronische Systeme. Sie unterstützt insbesondere Echtzeit, Verteilungsaspekte, dynamischen Komponentenaustausch, sowie diskrete und kontinuierliche Regelkreise. Sie erlaubt die Beschreibung diskreter Steuerungsteile mithilfe einer Art

von *Timed Automata* [AD94], hier als *Real-Time State Charts* bezeichnet. Kontinuierliche Regelkreise können als *Hybrid State Charts* beschrieben werden.

4 Diskussion der ursprünglichen Aufgabenstellung

Die Ausgangs-Aufgabenstellung lautet: "*Konzept für die Erweiterung bestehender paralleler Programmiermodelle um Determinismus in Echtzeitsystemen*".

Die Forderung nach Determinismus eines Programms bedeutet zunächst einmal eine zeitliche Translations-Invarianz des Verhaltens des Programms. Das heißt, ein gegebenes Programm soll sich unter gegebenen äußeren Umständen (Eingabedaten und andere) immer gleich verhalten, unabhängig von seinem Startzeitpunkt.

Zur Abgrenzung halten wir fest: Mit "*Determinismus in Echtzeitsystemen*" sind nicht gemeint (i) Garantien der Terminierung und (ii) Laufzeitgarantien. Die Erbringung solcher allgemeinen Garantien stieße auch auf theoretische Schwierigkeiten, z.B. wegen der Unlösbarkeit des allgemeinen Halteproblems [Tur36, Tur37].

Zu den "*äußeren Umständen*" gehören nicht nur die Eingabedaten des Programms (Sensordaten usw.), sondern auch solche, denen man einen Einfluß auf den Determinismusbegriff nicht oder nur notgedrungen einräumen möchte, z.B.

- Temperatur des Computers
- Verfügbarkeit von Betriebsmitteln.

Spätestens bei der zweiten genannten Einflußgröße wird klar, daß man die Forderung nach Determinismus um eine zweite Forderung mit schwächerer Voraussetzung, dafür aber ohne absolute Geschwindigkeitsaussage ergänzen muß.

Zusammenfassend wird gefordert:

1. Ein gegebenes Programm soll sich unter gegebenen äußeren Umständen (Eingabedaten und andere) immer gleich verhalten, unabhängig vom Startzeitpunkt.

2. Ein gegebenes Programm soll bei gegebenen Eingabedaten ungeachtet der absoluten Geschwindigkeit immer die gleiche Reihenfolge der (beobachtbaren) Einzelschritte ausführen.

Die Semantik so gut wie jeder praxisbezogenen Programmiersprache/ jedes Programmiermodells enthält Nichtdetermismen, und zwar aus mehrerlei Gründen:

- Der Beschreibungsaufwand der Semantik würde sich andernfalls erhöhen;

- deterministische Regelungen wären oft willkürlich, bezögen sich z.B. auf die textuelle oder alphabetische Reihenfolge, oder würden Prioritäts-Annotationen erfordern. Sie stünden evtl. Fairness-Erfordernissen entgegen.

- Der Interpreter[1] erhält durch Unterspezifikationen in der Semantik die erforderlichen Freiheiten, das Programm auf verschiedene Zielarchitekturen abzubilden und individuell zu optimieren.

Diese Nichtdetermismen bzw. Freiheiten betreffen insbesondere die Berechnungsreihenfolge, sehr häufig z.B. bei Unterausdrücken und Aktualparametern.

Bei parallelen Programmiermodellen ist es nun aber geradezu das Entwurfsziel, daß bestimmte Vorgänge in beliebiger Reihenfolge oder auch parallel ausgeführt werden dürfen. Um diese parallelen Vorgänge zu koordinieren, enthalten solche Programmiermodelle dedizierte Konstrukte wie z.B. *critical regions* oder *message passing.* Deren Definition

[1] "Interpreter" hier immer im abstrakten Sinne, z.B. auch Compiler mit Laufzeitsystem.

zielt regelmäßig darauf ab, den ursprünglichen zeitlichen Nichtdeterminismus gerade nur soviel einzuschränken wie nötig.

Die Forderung nach (zeitlichem) Determinismus läuft dieser Vorgehensweise direkt zuwider. Zur Erlangung des Determinismus eines Programmiermodells wäre jeglicher Nichtdetermismus aus der Semantik-Definition desselben zu entfernen, zumindest dann, wenn er potentiell beobachtbar ist. Festlegungen, die üblicherweise als Interna von Interpretern angesehen werden, müssten in die Semantik-Definition aufgenommen werden. Dies gälte insbesondere für das Scheduling von Parallelismus-Konstrukten.

Um die Aufgabenstellung den Buchstaben nach zu erfüllen, könnte man nun jedes (sequentielle oder nebenläufige) Programmiermodell deterministisch machen, indem man jede offengelassene Wahlmöglichkeit in der Semantik und im Interpreter durch eine semantische Festlegung ersetzt. Bei einem nebenläufigen Programmiermodell bedeutete z.B. die Forderung, ein Programm müsse sich unabhängig von der Anzahl der verfügbaren Prozessoren immer gleich verhalten, daß auch nur ein Prozessor eingesetzt werden kann. Jedwedes parallele Programmiermodell würde damit zu einem deterministischen Koroutinen-artigen Programmiermodell degradiert.

Da eine solche Vorgehensweise in vielerlei Hinsicht kontraproduktiv wäre, wird die ursprüngliche Aufgabenstellung eine freieren Interpretation unterzogen.

5 Reinterpretation der Aufgabenstellung

Ein nebenläufiges eingebettetes System fassen wir wie folgt auf:

- Vielzahl von kooperierenden zustandsbehafteten Komponenten;
- jede Komponente führt im Laufe der Zeit, ausgehend von einem Anfangszustand, eine Folge von äußerlich unteilbaren Zustandsübergängen von einem "stationären" Zustand in den nächsten aus, potentiell mit Seiteneffekten wie dem Ansteuern von Aktuatoren;

- jede Komponenten kann die stationären Zustände anderer Komponenten sowie die Außenwelt (Sensoren) beobachten.

Welche Teile eines Zustands (Faktoren bzw. Projektionen des Zustandsraumes) von welchen anderen Komponenten tatsächlich beobachtbar sind, kann durch programmiersprachliche Sichtbarkeitsregeln, "Beobachtungs-Topologien" u.ä. geregelt sein. Dies ist nicht Gegenstand dieses Deliverables.

Die hier gewählte Sichtweise auf eingebettete Systeme ist angemessen, da diese Systeme dazu dienen, das Verhalten von Maschinen im Laufe der Zeit zu steuern. Das Gesamtverhalten eines solchermaßen aufgefassten Systems ist die "Summe" der Einzelverhalten aller Komponenten. Ein formaler Kompositionskalkül (hier nicht ausgearbeitet) beträfe

- die gemeinsame zeitliche Koordination und
- die gegenseitigen Zustandsbeobachtungen.

Statt eines reinen Ergebnis-Nichtdeteminismus im Kontext seiteneffektfreier (funktionaler) Programmierung untersuchen wir Aspekte des Verhaltensdeterminismus seiteneffekt-behafteter Programmierung hinsichtlich Zeit und Reihenfolge der beoabachtbaren Aktivitäten.

Dabei suchen wir Konzepte für die *graduelle* Kontrolle von (Nicht-) Determinismus von Nebenläufigkeit. Prinzipiell können solche Konzepte auf verschiedenen Ebenen eingebracht werden, z.B. durch:

- Konstrukte in graphischen Entwicklungsumgebungen;
- Entwurf programmiersprachlicher Design-Patterns;
- Ersatz von Prozessen/Threads durch andere Konzepte;
- globale Systemzeit, Annotationen individueller Aktionen.

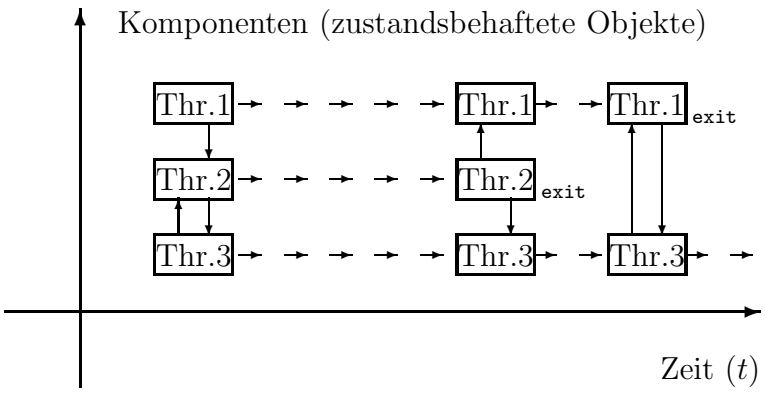

Abbildung 1: Horizontale und vertikale Struktur

6 Horizontale vs. vertikale Strukturierung

Die oben vorgestellte Auffassung von nebenläufigen eingebetteten Systemen (Vielzahl von zustandsbehafteten Komponenten, Abfolgen von zustandsverändernden Einzelschritten) in ein Programmiermodell zu gießen, gibt es mehrere Wege. Diese lassen sich in zwei grundlegende Strukturierungstechniken unterteilen (vgl. Abb. 1):

- *vertikale Strukturierung:*
  Das zentrale Konstrukt erfasst das Verhalten des Gesamtsystems in einem Einzelschritt (parallele Operation auf Container-Objekt).

- *horizontale Strukturierung:*
  Das zentrale Konstrukt erfasst das Verhalten der einzelnen Komponente über der Zeit (Thread, Prozess).

In diesem Bild wird die Zeitachse als horizontal verlaufend angesehen, die vertikale Achse repräsentiert die (ggf. "räumlich" strukturierte) Vielzahl der Komponenten.

Diese gegensätzlichen Strukturierungstechniken unterscheiden sich in vielerlei Hinsicht, z.B.

- Explizitheit der zeitlichen Koordination
- Determinismus
- Fähigkeit zur Beschreibung von dynamischen Komponentenstrukturen
- Anschaulichkeit und Beherrschbarkeit
- Eignung für graphische Darstellungen und Entwicklungssysteme
- Fähigkeit zu Strukturaussagen, formale Verifizierbarkeit.

7 Konzepte für die vertikale Koordination

Vertikale Koordination betont den *räumlichen Homogenitätscharakter* der parallelen Operationen und kommt vorzugsweise im Bereich des Datenparallelismus zum Einsatz. Beispiele:

- *SIMD:* (Single instruction multiple data)
  Extremform der Homogenität, nämlich auf Maschinenbefehlsebene;
- *SPMD:* (Single process multiple data)
  Homogenität auf Skript-Ebene, Autonomie der Kontrollflüsse;
- *Skeletons:*
  Formalisierung von Parallelismus und Kommunikation durch einen kleinen Satz von Funktionen zweiter Ordnung.

In diesen Modellierungsweisen sind Determinismus bzw. das Ausmaß eines verbleibenden Nichtdeterminismus sehr gut kontrollierbar und direkt an den Eigenschaften des gewählten Koordinationsprinzips ablesbar. Insbesondere Skeletons und verwandte Ansätze (z.B. MapReduce [AAR+01]) bieten ein hohes Abstraktionsniveau von parallelen Operationen auf Container-Datentypen.

Für jede Komponente müssen ihr Zustandsraum und die Zustandsübergänge als einzeln aufrufbare Operation kodiert werden. Bei vielen *homogenen* Datenelementen entsteht das natürliche Bild eines einzigen Kontrollflusses aus inhärent parallelen Operationen auf Container-Objekten.

Anders bei *heterogenen* Datenelementen und Rechenabläufen: Hier wirkt die vertikale Strukturierung künstlich, ist wesentlich unanschaulicher (und wahrscheinlich fehleranfälliger) als die Programmierung in Form von durchlaufenden Prozessen oder Threads.

8 Konzepte für die horizontale Koordination

Diese Technik, die Modellierung von Parallelismus durch Prozesse oder Threads, betont die *Heterogenität und Autonomie* der einzelnen Komponenten. Die nebenläufige Ausführung von Prozessen bzw. Threads mit ihren autonomen Kontrollflüssen ist *per se* hochgradig nichtdeterministisch [Lee06]. Homogenitäts- und Koordinationsaspekte, auch der des Determinismus, müssen "nachträglich" eingebaut werden. Das Problem hierbei ist, daß eine erwünschte Eigenschaft des Gesamtsystems als Konjunktion "von innen heraus" formulierter Einzeleigenschaften beschrieben werden muß. Nachvollziehbarkeit und Beweisbarkeit globaler Eigenschaften sind – aufgrund der semantischen Autonomie der Komponenten – wesentlich schwieriger.

9 Eigenvorschlag

Es folgt ein erster Vorschlag zur Verringerung des Nichtdeterminismus von (potentiell inhomogenen) Prozessen/Threads, aus mehreren Teilen bestehend:

1. Es gibt eine unendliche Folge von globalen *Barriers* an aufeinanderfolgenden diskreten Zeitpunkten t_1, t_2, ..., im einfachsten Fall in äquidistanten Zeitschritten. Kein Prozess/Thread rechnet weiter, bevor nicht alle Prozesse/Threads dieselbe Barrier-Instanz erreicht haben.

Message Passing für sich alleine ist schon anfällig für deadlocks; diese Gefahr würde durch eine Interaktion mit globalen Barriers noch verstärkt. Deshalb:

2. An Stelle von Message Passing tritt die gegenseitige *Beobachtung* von Zuständen zu den Zeitpunkten der Barriers (ist immer deadlock-frei). Die bisherigen Botschaften und ggf. deren Zeitstempel und Identitäten der Empfänger werden in beobachtbaren Variablen hinterlegt. Anstatt von "Kommunikations-Topologien" würde man dann besser von "Beobachtungs-Topologien" sprechen.

Das zur Verwendung innerhalb von Prozessen/Threads vorgeschlagene Konstrukt hat informell folgende Semantik:

> "Lass mich schlafen bis zum nächsten Uhrenschlag und besorge mir dann die dann gültigen Inhalte der von mir beobachteten Variablen der anderen Prozesse/Threads. Meine eigenen Variablen haben bereits die beim nächsten Uhrenschlag gültigen Werte."

Durch ein solches Schema werden die nebenläufigen Prozesse/Threads in ein starres Lock-Step-Schema gebracht; ein bestimmter Grad von Determinismus ist erst einmal erreicht. – Ein etwas allgemeineres Konstrukt könnte folgende informelle Semantik haben:

> "Lass mich schlafen bis zum n-ten nächsten Uhrenschlag; falls aber eine der von mir beobachteten Variablen der anderen Prozesse/Threads überschrieben wird, weck mich früher und sage mir welche; besorge mir die dann vorliegenden Werte aller beobachteten Variablen. Meine eigenen Variablen haben bereits die beim Aufweck-Zeitpunkt gültigen Werte."

Eine programmiersprachliche (syntaktische) Ausgestaltung des vorgeschlagenen Konstrukts wäre zu diesem Zeitpunkt noch verfrüht.

Variationen: Für den bis hier suggerierten Zeitbegriff der regelmäßigen Uhrenschläge kommen z.B. folgende Erweiterungen in Betracht:

- *Uhrenschläge zu vorher nicht festgelegten (jedoch nach wie vor diskreten) Zeitpunkten.* Jeder Prozess/Thread kann sich zu beliebigen Zeitpunkten in der Zukunft aufwecken lassen. Die Gesamtmenge aller dieser Zeitpunkte bildet ein potentiell irreguläres diskretes Zeitraster.

  Zusammen mit der Möglichkeit des (vorzeitigen) Aufweckens bei Änderungen fremder Variablen ist *Discrete-Event-Simulation* hiermit bereits ausdrückbar.

- *Hierarchisch strukturierte Uhrenschläge für zeitlich multi-skalige Anwendungen.* Die Hierarchiestufen könnten dann an die logische Schachtelungsstruktur der Prozess-Skripte geknüpft werden; hierdurch würde die Nachvollziehbarkeit des Gesamtverhaltens erhöht.

10 Fazit

Es bleibt festzuhalten, daß Homogenität und Determinismus paralleler Prozesse/Threads auch nach ("freiwilliger") Benutzung der vorgeschlagenen oder ähnlicher Konstrukte nicht die Stringenz ganzheitlicher datenparalleler Modelle erreichen; auf der anderen Seite sind sie flexibler. Inwieweit Vorteile oder Nachteile überwiegen, kann nur im Zusammenhang mit der jeweiligen Anwendung bewertet werden.

Literatur

[AAR+01] P. An, A.Jula, S. Rus, S. Saunders, T. Smith, G. Tanase, N. Amato, and L. Rauchwerger. An adaptive, generic parallel

C++ library. In *Wksph. on Lang. and Comp. for Par. Comp. (LCPC), Cumberland Falls, KY*, pages 193–208, 2001.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[BGT05] Sven Burmester, Holger Giese, and Matthias Tichy. *Model-Driven Development of Reconfigurable Mechatronic Systems with Mechatronic UML*, volume 3599 of *LNCS*. Springer, 2005.

[BKPS07] M. Broy, I.H. Kruger, A. Pretschner, and C. Salzmann. Engineering automotive software. *Proceedings of the IEEE*, 95(2):356–373, Feb 2007.

[Bro97] M. Broy. Refinement of time. In M. Bertran and Th. Rus, editors, *Transformation-Based Reactive System Development, Proc. ARTS'97*, volume 1231 of *LNCS*, pages 44–63. Springer, 1997.

[Bro05] Manfred Broy. *Service-Oriented Systems Engineering: Specification and Design of Services and Layered Architectures — The Janus Approach*, volume 195, pages 47–81. Springer, 2005.

[BS01] M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001.

[DVM+05] Werner Damm, Angelika Votintseva, Alexander Metzner, Bernhard Josko, Thomas Peikenkamp, and Eckard Böde. Boosting re-use of embedded automotive applications through rich components. In *Proc. Foundations of Interface Technologies (FIT)*. Springer, Aug 2005.

[GB03] Holger Giese and Sven Burmester. Real-time statechart semantics. Technical report, Univ. Paderborn, 2003.

[GBSO04] Holger Giese, Sven Burmester, Wilhelm Schäfer, and Oliver Oberschelp. Modular design and verification of component-based mechatronic systems with online-reconfiguration. *SIGSOFT Softw. Eng. Notes*, 29(6):179–188, 2004.

[HHR09] Alexander Harhurin, Judith Hartmann, , and Daniel Ratiu. Motivation and formal foundations of a comprehensive modeling theory for embedded systems. Technical Report TUM-I0924, Technische Universität München, 2009.

[JMM08] Bernhard Josko, Qin Ma, and Alexander Metzner. *Designing Embedded Systems using Heterogeneous Rich Components*. 2008.

[Lee06] Edward A. Lee. The problem with threads. Technical Report UCB/EECS-2006-1, Univ. of California at Berkeley, Electrical Engineering and Computer Sciences, Jan 2006.

[Tur36] A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc., Ser. 2*, 42:230–265, 1936.

[Tur37] A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem (correction). *Proc. London Math. Soc., Ser. 2*, 43:544–546, 1937.