Formale Grundlagen der Fehlertoleranz in verteilten Systemen

Felix C. Gärtner

Technische Universität Darmstadt
felix@informatik.tu-darmstadt.de

Die Literatur über fehlertolerante Computersysteme ist mannigfaltig: Mindestens ein halbes Dutzend wissenschaftliche Zeitschriften beschäftigen sich ausschließlich mit Fehlertoleranzmethoden und deren Validierungsmöglichkeiten, und es gibt etwa doppelt so viele jährlich stattfindende Konferenzen zu diesem Thema. Obwohl in den letzten Jahren durchaus lesenswerte Einführungen in Form von Büchern oder Zeitschriftenartikeln erschienen sind, ist das Gebiet noch weit entfernt von einer einheitlichen, konsistenten Darstellung. Für Neulinge bleibt der Einstieg in diesen Themenbereich darum schwer. Es gibt viele Argumente dafür, daß der Grund hierfür in der mangelnden formalen Ausrichtung des Gebietes liegt. Schließlich existiert noch keine gesicherte theoretische Basis vieler Fehlertoleranzverfahren. Dieser Beitrag faßt die Ansätze einer solchen Theorie aus der Literatur zusammen und beschreibt in aller Kürze die Beiträge, die in der Dissertation des Autors zu dieser Theorie geleistet wurden.

1 Einführung

Heute haben Computersysteme unseren gewöhnlichen Lebensalltags so weit durchdrungen, daß sie oft kaum noch explizit wahrgenommen werden. Da Computer (wie alle technischen Systeme) einem Alterungsprozeß unterliegen, sind Ausfälle gewissermaßen "vorprogrammiert". Die zwangsläufig auftretenden Probleme und die dadurch verursachten Zwischenfälle resultieren nicht nur in Chaos und Konfusion, sondern können Schäden in Millionenhöhe bis hin zum Verlust von Menschenleben verursachen [Ne95]. Systeme, die in kritischen Bereichen wie der Steuerung von Flugzeugen, dem Gesundheitswesen, der Finanzwelt und der Energieversorgung eingesetzt werden, müssen trotz interner Fehler weiterhin ihre Aufgabe erfüllen. Derartige Systeme nennt man *fehlertolerante Computersysteme*.

Oft wird die Auflage des US-amerikanischen Raumfahrtprogramms in den 1960er Jahren als die Geburtsstunde der umfassenden ingenieursmäßigen Beschäftigung mit fehlertoleranten Computersystemen angesehen. Heute beschränkt sich der Einsatz derartiger Systeme nicht mehr nur auf technische Randgebiete (wie die Raumfahrt): Bei Telekommunikationsanbietern, Internet-basierten Versandhäusern und in der Automobilindustrie sind fehlertolerante Computersysteme im täglichen Einsatz. Diese praktische Relevanz ist vielleicht der Grund dafür, daß es heute eine verwirrende Vielfalt von Terminologien und

Konzepten in diesem Gebiet gibt, die Anfangs der 1990er Jahre selbst ausgewiesene Experten verunsicherte. Beispielsweise stellen Arora und Gouda in einem 1993 erschienenen Artikel fest [AG93]:

> [...] *the discipline* [*of fault-tolerance*] *itself seems to be fragmented.*

Wenigstens die Terminologiedebatte wurde etwas besänftigt, als 1992 das Buch "*Dependability: Basic Concepts and Terminology*" von Laprie [La92] erschien. Es enthält einen in Gemeinschaftsarbeit mit vielen anderen Forschern festgelegten Terminologie- und Konzeptkatalog, der viele Bereiche der Fehlertoleranz abdeckt. Dennoch bleiben zentrale Konzepte der Fehlertoleranz, wie zum Beispiel das der *Redundanz*, unerwähnt.

Obwohl sich praktisch alle Forscher heute auf Laprie beziehen, bleibt der Einstieg in dieses Gebiet für Neulinge schwer. Es gibt mittlerweile durchaus lesenswerte Einführungen in Form von Büchern [Ja94, LA90, Ec90] oder Zeitschriftenartikeln [HW92, Avi76, Cr91], jedoch sind diese Texte noch weit entfernt von einer einheitlichen, konsistenten Darstellung des Gebietes. Es gibt viele Argumente dafür, daß der Grund hierfür in den mangelnden formalen Ausrichtung des Gebietes liegt. Schließlich existiert noch keine gesicherte theoretische Basis vieler Fehlertoleranzverfahren.

Dieser Beitrag möchte in einer eher essayistischen Form die Ansätze einer solchen in der Literatur bereits existierenden Theorie aufzeigen und in aller Kürze die Beiträge skizzieren, die sich aus der Dissertation des Autors [Gä01] dazu ergeben haben.

2 Modelle fehlertoleranter verteilter Systeme

In vielen Einsatzbereichen fehlertoleranter Systeme wird verlangt, daß der zugrundeliegende Fehlertoleranzalgorithmus (wenigstens ingenieursmäßig) verifiziert worden ist, bevor das System zum Einsatz kommt. Voraussetzung für eine solche Untersuchung ist ein formales Systemmodell. Ein Systemmodell beschreibt in einer begrenzten Menge von Attributen und Reglen die Grundannahmen über die Ausführungsumgebung [Sc93]. Die Kunst dabei ist, ein Modell zu finden,

1. welches alle relevanten Systemeigenschaften aus der Realität genau genug wiederspiegelt, und
2. mit dem sich die Kernprobleme eines bestimmten Anwendungsbereiches gut und einfach formulieren lassen.

Bei der Analyse fehlertoleranter reaktiver Systeme sind zunächst qualitative Aspekte ihres Verhaltens wichtiger als quantitative, d.h. Fragen wie "Terminiert der Algorithmus?" oder "Kann der Algorithmus in Zustand x kommen wenn ein Fehler auftritt?". Diese Eigenschaften sind durch die beiden Klassen von Sicherheits- und Lebendigkeitseigenschaften (im Sinne von Lamports *safety* und *liveness* [La77]) gut beschreibbar. Bei der Formulierung dieser Eigenschaften muß aber immer eventuelles Fehlverhalten berücksichtigt wer-

den. Dabei kann man einem bekannten Ansatz der Fehlermodellierung folgen, der Fehler auf der Ebene des Algorithmus durch zusätzliche Programminstruktionen modelliert.

In der Forschung gibt es vier prominente Systemmodelle für fehlertolerante verteilte Systeme: das (partiell) synchrone Modell von Dwork, Lynch und Stockmeyer [DLS88], das Fehlerdetektormodell von Chandra und Toueg [CT96], das zeitbeschränkte asynchrone Modell von Cristian und Fetzer [CF99], sowie das quasi-synchrone Modell von Almeida, Veríssimo und Casimiro [AVC98]. Alle Modelle basieren auf dem sogenannten asynchronen (oder "zeitfreien") Modell, welches normalerweise für das Internet postuliert wird. Das asynchrone Modell zeichnet sich durch seine Einfachheit in Bezug auf Echtzeitannahmen aus: Es werden keine Zeitschranken angenommen, weder auf Nachrichtenlaufzeiten noch auf den Geschwindigkeitsunterschieden der einzelnen Rechnerknoten. Dieses Systemmodell ist jedoch erwiesenermaßen nicht gut geeignet für Fehlertoleranzsysteme, weil darin das in der Fehlertoleranz so wichtige Übereinstimmungsproblem des *consensus* nicht gelöst werden kann [FLP85]. Die vier oben angesprochenen Systemmodelle versuchen, diese Tatsache durch zusätzliche Annahmen in unterschiedlicher Weise zu umgehen:

1. Das (partiell) synchrone Modell durch die Einführung von expliziten Zeitschranken.
2. Das Fehlerdetektormodell durch die Einführung von unzuverlässigen Fehlerdetektoren.
3. Das zeitbeschränkte asynchrone Modell durch die Annahme von Hardware-Uhren, zeitbeschränkten Diensten und bedingten Lebendigkeitsannahmen.
4. Das quasi-synchrone Modell durch die Einführung von expliziten, dynamisch adaptierbaren Zeitschranken und einem perfekten Zeitfehlerdetektor.

Vergleich der Systemmodelle. Das synchrone oder partiell synchrone Modell (Modell 1) ist relativ einfach zu verstehen, weil es das Zeitverhalten des Systems global festlegt. Es hat in den Anfängen der Fehlertoleranztheorie vielfältigen Einsatz in der Beschreibung und Analyse von Algorithmen gefunden [Ly96].

Im Gegensatz zum synchronen Modell erlaubt das Fehlerdetektormodell (Modell 2) eine feinere Abstufung verschiedener Synchronitätsannahmen sowie die Trennung von Funktion und Mechanismus der Fehlererkennung. Das Konzept der Fehlerdetektoren ist zwar relativ schwierig zu verstehen und muß für jede Fehlerannahme neu angepaßt werden, doch ist es das einzige Modell, welches zumindest auf der "Benutzungsoberfläche" jeden Bezug zu einer Vorstellung von Echtzeit vermeidet. Das Fehlerdetektormodell ist deswegen zu einem der populärsten Systemmodelle für die Analyse und Verifikation von Fehlertoleranzalgorithmen geworden.

Vergleicht man das Fehlerdetektormodell (Modell 2) und das zeitbeschränkt asynchrone Modell (Modell 3) [Fe99], so zeigt sich, daß dem zeitbeschränkten asynchronen Modell ohne die dort getroffenen Fortschrittsannahmen die Grundlage für die Lebendigkeit der in ihm ablaufenden Algorithmen fehlt. Durch die Hinzunahme von Hardware-Uhren und die notwendige Zeitbeschränkung aller Dienste ist es allerdings möglich, sogenanntes Fehlerbewußtsein herzustellen und wenigstens zu garantieren, daß das System in einem sicheren

Zustand verbleibt. Das zeitbeschränkt asynchrone Systemmodell eignet sich darum vor allem für den Entwurf von Systemen, wo es eine "sichere Alternative zu normalem Arbeiten" gibt [Ru94]. Dies ist beispielsweise der Fall in automatischen Steuerungssystemen für Züge. In diesem Anwendungsgebiet ist das zeitbeschränkt asynchrone Systemmodell auch bereits zum Einsatz gekommen [EAP99]. Die Trennung von eigentlichem Systemmodell und Fortschrittsannahmen unterstreicht die praktische Ausrichtung dieses Modells.

Das quasi-synchrone Systemmodell (Modell 4) ist neben Modell 1 das "synchronste" Modell, welches in der Arbeit untersucht wurde. Die Fehlererfassung ist durch die Annahme von oberen Schranken auf den Fehlerhäufigkeiten nicht so hoch wie im zeitbeschränkt asynchronen Modell. Dafür ist es möglich, die Effizienz einer Datenübertragung relativ genau zu überwachen. Der zentrale Mechanismus hierfür ist der perfekte Zeitfehlerdetektor, der mit modernen Echtzeitnetzwerktechnologien realisiert werden kann. Das quasi-synchrone Modell ist das einzige, in dem man eine Art "Echtzeit-Lebendigkeit" erreichen und nachweisen kann. Es eignet sich dewegen insbesondere für fehlertolerante Anwendungen, die Echtzeitanforderungen erfüllen müssen [KV93, Ve93].

Beiträge zur Fehlertoleranzmethodik. Die hier erwähnten Systemmodelle sind bei der Betrachtung von Fehlertoleranzmethoden in verteilten Systemen unerläßlich. Sie bilden quasi die Diskussionsgrundlage, d.h. eine Basis, auf die man sich verständigen muß, bevor es mit der eigentlichen Arbeit losgehen kann. Ein Vergleich ihrer Stärken und Schwächen erleichtert dem Wissenschaftler und dem Ingenieur die Wahl des jeweils "richtigen" Modells.

3 Die Fehlertoleranztheorie von Arora und Kulkarni und die Rolle von Redundanz in der Fehlertoleranz

Die Vielfalt an verschiedenen Fehlertoleranzmethoden ist auch für Experten heute kaum mehr zu überblicken. Beim Lesen von neuen Veröffentlichungen beschleicht einen jedoch oft das Gefühl, daß viele solche Verfahren einfache Neuanwendungen ein und desselben Inventars an Grundmechanismen der Fehlertoleranz sind. Die Fehlertoleranztheorie von Arora und Kulkarni [AK98] unterstützt diese Vermutung.

Die Theorie von Arora und Kulkarni. Grundlage der Fehlertoleranztheorie von Arora und Kulkarni ist ein vollständig formales Systemmodell, in dem auch die Korrektheit eines Programms bezüglich einer Spezifikation genau definiert werden kann. Die Grundaussage der Theorie lautet:

> Ein fehlertolerantes Programm ist die Komposition aus einem fehlerintoleranten Programm und einer Menge von Fehlertoleranzkomponenten.

Es gibt nur zwei grundlegende Arten von solchen Fehlertoleranzkomponenten: Detektoren und Korrektoren (*detectors and correctors*). Detektoren sind Programmmodule, die

feststellen, ob ein Prädikat auf dem Zustand des Gesamtsystems gilt oder nicht. Korrektoren sind Programmmodule, die einen bestimmten Systemzustand herstellen. Mittels dieser beiden Komponenten kann man tatsächlich eine große Menge an fehlertoleranten Systemen entwerfen. Bekannte Entwurfsmuster der Fehlertoleranz wie Schneiders *state machine approach* [Sc90] sind gleichermaßen konstruierbar mittels Detektoren und Korrektoren. Arora und Kulkarni zeigen außerdem zwei zentrale Theoreme für fehlertolerante Systeme:

- Detektoren sind hinreichend und notwendig, um eine Sicherheitseigenschaft einzuhalten.
- Korrektoren sind hinreichend und notwendig, um letztendlich wieder eine Sicherheitseigenschaft zu erfüllen.

Die Rolle von Redundanz. Detektoren und Korrektoren sind nur über ihre Funktionalität definiert. Es bleibt also die Frage: Wie funktionieren diese Module grundsätzlich? Bei Untersuchung dieser Frage kommt man zu dem Schluß, daß nicht-erreichbare Zustände und Zustandsübergänge hierbei eine entscheidende Rolle spielen. Dies sind Zustände und Zustandsübergänge, die ein Programm im Normalfall (d.h. wenn keine Fehler auftreten) nicht benötigt. Die betrachteten Systemeigenschaften unterteilen sich wieder in Sicherheits- und Lebendigkeitseigenschaften. Man kann nun zwei weitere grundlegende Theoreme zeigen [Gä01]:

1. Um Sicherheitseigenschaften zu erfüllen, benötigt man nicht-erreichbare Zustände.
2. Um letztendlich wieder eine Sicherheitseigenschaft zu erfüllen, benötigt man nicht-erreichbare Zustände und nicht-erreichbare Zustandsübergänge.

Nicht-erreichbare Zustände bilden die Grundlage für den neu definierten Begriff der *Speicherredundanz*. Im Rahmen der Theorie ist demnach Speicherredundanz notwendig, um Sicherheit zu erreichen. Nicht-erreichbare Zustandsübergänge bilden die Grundlage für den neu definierten Begriff der *Zeitredundanz*. Um eine gewisse Form von Lebendigkeit zu erreichen, benötigt man demnach Zeitredundanz *und* Speicherredundanz.

Übertragen auf die Theorie von Arora und Kulkarni bedeutet dies:

1. Detektoren enthalten Speicherredundanz, und
2. Korrektoren enthalten Speicherredundanz und Zeitredundanz.

Diese Ergebnisse geben also Einblicke in die grundsätzliche Wirkungsweise von Fehlertoleranzkomponenten.

4 Fehlertolerantes Beobachten

Die Fehlertoleranztheorie aus dem vorhergehenden Abschnitt bietet Einsichten, die etablierte Resultate auf dem Gebiet der Fehlertoleranz erklären hilft und sich erfolgreich bei

der Entwicklung neuer Fehlertoleranzmethoden anwenden lassen. Eine solche Einsicht ist: Man kann Fehler nur erkennen, wenn ein Fehlereffekt Teil des Systemzustandes geworden sind. Fehler, die von normalen Programmaktionen nicht zu unterscheiden sind, kann man nicht verläßlich erkennen.

Eine zweite Einsicht ist: Jedes Fehlertoleranzverfahren besteht aus einem Fehlererkennungs- und einem Fehlerbehebungsmodul. Fehlererkennung und Fehlerbehebung sind demnach zwei unterschiedliche Aufgaben, die unabhängig voneinander betrachtet und untersucht werden können. Verbunden mit der ersten Einsicht bedeutet Fehlererkennung, daß man wissen möchte, ob sich das System in einem gewissen unerwünschten Zustand befindet. Unter diesem Blickwinkel ist Fehlererkennung also eine Art Beobachtungsproblem: Beobachte das System und melde, wenn ein bestimmter (globaler) Zustand eingetreten ist. Formal versucht man die Gültigkeit eines Prädikates auf dem globalen Systemzustand zu entdecken. Es gibt relativ viele Arbeiten, die das Beobachtungsproblem in fehlerfreien Umgebungen untersuchen [CM91, MN91, BM93]. Eine grundsätzliche Untersuchung, die auch Fehler in Betracht zieht, fehlte bisher.

Unsicherheit bei der Beobachtung. Die Arbeiten des Autors auf dem Gebiet des fehlertoleranten Beobachtens [GK00, Gä01] basieren auf den Annahmen des asynchronen Systemmodells (vergleiche Abschnitt 2). In diesem Systemmodell ist das Problem der Beobachtung bereits intensiv untersucht worden, aber natürlich unter der Annahme, daß keine Fehler auftreten. Selbst unter dieser Annahme hat sich herausgestellt, daß das Beobachtungsproblem schwieriger ist, als es anfangs erscheint.

Es gibt viele verschiedene Arten von Prädikaten, und man kann mit Recht annehmen, daß manche einfacher zu entdecken sind als andere. Im allgemeinen Fall läßt sich die Frage nach der Gültigkeit eines Prädikates schon bei ganz einfachen Beispielen nicht mehr eindeutig global beantworten. Vereinfachend gesagt liegt das daran, daß "annähernd gleichzeitige" Ereignisse auf verschiedenen Prozessen von unterschiedlichen Beobachtern in einer unterschiedlichen Reihenfolge gesehen werden können. Antworten auf die Frage nach der Gültigkeit eines Prädikates hängen also oft vom Beobachter ab. Wenn man aber gerne unabhängig vom Beobachter entscheiden möchte, ob ein Prädikat gilt oder nicht, dann bedient man sich sogenannter *Beobachtungsmodalitäten*: Man fragt beispielsweise nicht mehr "Gilt das Prädikat?" sondern: "Könnte es einen Beobachter geben, der dieses Prädikat erkennt?" [CM91, MN91] Wenn allerdings *crash*-Fehler auftreten dürfen, stellen sich viele knifflige Fragen. Zum Beispiel: Wenn ein Prädikat φ auf einem Prozeß gilt und dieser Prozeß abstürzt, gilt φ dann weiterhin? Angesichts der Unsicherheiten bei der *crash*-Fehlererkennung machen die Beobachtungsmodalitäten für fehlerfreie Systeme keinen Sinn mehr. Die Leitfrage dieses Kapitels lautet also: Welche sinnvollen alternativen Beobachtungsmodalitäten gibt es in asynchronen verteilten Systemen, in denen *crash*-Fehler auftreten können?

Ergebnisse. Die vorgestellten Lösungen können anhand der folgenden drei Entwurfsentscheidungen charakterisiert werden:

1. Rückgriff auf Ergebnisse aus der Fehlermodellierung. Der Absturz eines Prozesses wird behandelt, als wenn dieser eine lokale Variable *up* von *true* auf *false* setzen würde. Alle anderen Variablenwerte bleiben unverändert. Diese Idee entstammt Arbeiten auf dem Gebiet der Fehlermodellierung [Cr85, Ec84, Aro92, Gä99b]. Diese Entscheidung hat zwei wesentliche Vorteile: Erstens kann man Fragen nach der Gültigkeit von Prädikaten beim Absturz eines Prozesses sehr flexibel zu beantworten, indem man explizit auf den funktionalen Zustand des Prozesses innerhalb des Prädikats zurückgreift. Zweitens kann man nun auch auf relativ einfache Art und Weise ein Analogon zum Verband der konsistenten Zustände eine Berechnung definieren, ein aus der "fehlerfreien Welt" bekanntes und nützliches Konzept [BM93, SM94].

2. Verbleiben im asynchronen Modell. Statt das reine asynchrone Modell zu verlassen und beispielsweise (wie in neueren Arbeiten untersucht [GP01, GP02]) perfekte Fehlerdetektoren anzunehmen, zielte die Untersuchung auf Lösungen, die wenigstens potentiell in asynchronen Systemen implementiert werden können. Glücklicherweise waren die Fragestellungen unter den gegebenen Voraussetzungen größtenteils lösbar. Zunächst mußten für die gegebene Aufgabe entsprechend angepaßte Fehlerdetektoren entworfen werden, die eine zusätzliche "Kausalitätseigenschaft" besitzen (d.h. man kann eine Verdächtigung in Bezug setzen zu normalen Prozeßereignissen). Die resultierenden Detektoren sind nicht perfekt, sondern lediglich *unendlich oft genau*, d.h. während jeder Prozeßabsturz nach endlicher Zeit entdeckt wird, kann darf ein korrekter Prozeß nicht dauerhaft verdächtigt werden. Ein auf diesen Eigenschaften aufbauender Entdeckungsalgorithmus kann demnach nie "fehlerfrei" sein. Das ist quasi der Preis, den man für das Verbleiben im asynchronen Modell bezahlen muß.

Daß ein Fehlerdetektor in asynchronen Systemen notwendigerweise "unsicher" sein muß, ist eine Tatsache, mit der man leben kann. Zwar folgt aus ihr, daß die bereits vorgeschlagenen Modalitäten *possibly* und *definitely* [CM91, MN91] nicht mehr beobachterunabhängig sind. Jedoch kann man neue Beobachtungsmodalitäten definieren, die ihren Wahrheitswert nicht nur aus der "wahren" Gültigkeit des zugrundeliegenden Prädikates beziehen, sondern zusätzlich aus den Informationen des Fehlerdetektors. Wenn man beispielsweise entdecken will, ob $\varphi \equiv$ "Prozeß p_i ist abgestürzt" gilt, so gibt es prinzipiell die beiden Alternativen: (1) Melde es, wenn auch nur ein lokaler Fehlerdetektor p_i verdächtigt (neue Modalität *negotiably*), bzw. (2) melde es erst, wenn alle lokalen Fehlerdetektoren dies tun (neue Modalität *discernibly*). In Analogie zu optimistischen und pessimistischen Netzwerkprotokollen wurde argumentiert, daß dies durchaus praktikable Konzepte sind, die die "Unsicherheit" in die Semantik neuer Modalitäten aufnehmen können. Die eingeführten Modalitäten sind demnach trotz ihrer "Unsicherheit" noch nützlich und zeigen, daß ein "Leben mit den Nachteilen" unzuverlässiger Fehlerdetektoren möglich ist.

3. Modularität der Lösungen. Obwohl die "alten" Modalitäten *possibly* und *definitely* in asynchronen Systemen mit *crash*-Fehlern ihre Beobachterunabhängigkeit verlieren, muß man sie nicht verwerfen, da die Modalitäten *negotiably* und *discernibly* auf ihnen aufbauen. Hier kommen die verschiedenen Ideen aus den oben genannten Punkten zusammen: Der Rückgriff auf die Ergebnisse der Fehlermodellierung erlaubt es, einen Zustands-

verband für den fehlerbehafteten Fall zu definieren, in dem die Informationen der lokalen Fehlerdetektoren entweder optimistisch oder pessimistisch interpretiert werden können. Auf diesen beiden alternativen Zustandsverbänden kann man nun ganz gewöhnliche *possibly*- bzw. *definitely*-Entdeckung betreiben. Entsprechend dem existentiellen Charakter von *possibly* wählt man hier die pessimistische Alternative, für *definitely* analog die optimistische. Dies hat einerseits den Vorteil, daß die neuen Modalitäten *negotiably* und *discernibly* in enger Analogie zu den altbekannten Modalitäten verstanden werden können und eine Affinität zu bestimmten Formen von Sicherheits- und Lebendigkeitseigenschaften behalten. Andererseits kann man entsprechende Entdeckungsalgorithmen unter Rückgriff auf bereits veröffentlichte Algorithmen zur Entdeckung von *possibly* und *definitely* realisieren. Dies führt zu verständlicheren, modularen Lösungen.

Ausblick. Trotz der vielen Anwendungsmöglichkeiten sind diese Ergebnisse leider noch weit von der Praxistauglichkeit entfernt. Dies liegt im wesentlichen daran, daß das zugrundeligende Problem der Entdeckung von *possibly* und *definitely* allein schon sehr komplex ist [CG98]. In der Vergangenheit hat sich jedoch gezeigt, daß es aufbauend auf grundlegenden theoretischen Untersuchungen wie dieser durch geschickte Einschränkung der Annahmen (wie z.B. der Eigenschaften auf Prädikaten) durchaus effiziente und in der Praxis einsetzbare Verfahren entstehen können. Denn trotz der potentiellen Ästetik einer schönen Theorie sollte die praktische Verwertbarkeit nie aus den Augen verloren werden.

Danksagungen

Mein besonderer Dank gebührt den beiden Gutachtern meiner Arbeit, Prof. Dr. Peter Kammerer und Prof. Dr. Friedemann Mattern, für ihre andauernde Unterstützung.

Literaturverzeichnis

[AG93] Anish Arora and Mohamed Gouda. Closure and convergence: A foundation of fault-tolerant computing. *IEEE Transactions on Software Engineering*, 19(11):1015–1027, 1993.

[AK98] Anish Arora and Sandeep S. Kulkarni. Component based design of multitolerant systems. *IEEE Transactions on Software Engineering*, 24(1):63–78, January 1998.

[Aro92] Anish Kumar Arora. *A foundation of fault-tolerant computing*. PhD thesis, The University of Texas at Austin, December 1992.

[AVC98] Carlos Almeida, Paulo Veríssimo, and António Casimiro. The quasi-synchronous approach to fault-tolerant and real-time communication and processing. Technical Report CTI RT-98-04, Instituto Superior Técnico, Lisboa, Portugal, July 1998.

[Avi76] Algirdas Avižienis. Fault-tolerant systems. *IEEE Transactions on Computers*, 25(12):1304–1312, December 1976.

[BM93] Özalp Babaoğlu and Keith Marzullo. Consistent global states of distributed systems: Fundamental concepts and mechanisms. In Mullender [Mu93], chapter 4, pages 55–96.

[CF99] Flaviu Cristian and Christof Fetzer. The Timed Asynchronous Distributed System Model. *IEEE Transactions on Parallel and Distributed Systems*, 10(6), June 1999.

[CG98] Craig M. Chase and Vijay K. Garg. Detection of global predicates: Techniques and their limitations. *Distributed Computing*, 11(4):191–201, 1998.

[CM91] Robert Cooper and Keith Marzullo. Consistent detection of global predicates. *ACM SIGPLAN Notices*, 26(12):167–174, December 1991.

[Cr85] Flaviu Cristian. A rigorous approach to fault-tolerant programming. *IEEE Transactions on Software Engineering*, 11(1):23–31, January 1985.

[Cr91] Flaviu Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, February 1991.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

[EAP99] Didier Essame, Jean Arlat, and David Powell. Padre: A protocol for asymmetric duplex redundancy. In *Proceedings of the Seventh IFIP International Working Conference on Dependable Computing for Critical Applications*, San Jose, USA, January 1999.

[Ec84] Klaus Echtle. Fehlermodellierung bei Simulation und Verifikation von Fehlertoleranz-Algorithmen für Verteilte Systeme. In F. Belli, S. Pfleger, and M. Seifert, editors, *Software-Fehlertoleranz und -Zuverlässigkeit*, number 83 in Informatik-Fachberichte, pages 73–88. Springer-Verlag, 1984.

[Ec90] Klaus Echtle. *Fehlertoleranzverfahren*. Springer-Verlag, 1990.

[Fe99] Christof Fetzer. A comparison of timed asynchronous systems and asynchronous systems with failure detectors. In *Proceedings of the Third European Research Seminar on Advances in Distributed Systems (ERSADS'99)*, pages 109–118, Madeira Island, Portugal, April 1999.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[Gä99a] Felix C. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31(1):1–26, March 1999.

[Gä99b] Felix C. Gärtner. Transformational Approaches to the Specification and Verification of Fault-Tolerant Systems: Formal Background and Classification. *Journal of Universal Computer Science (J.UCS)*, 5(10):668–692, October 1999. Special Issue on Dependability Evaluation and Assessment.

[Gä01] Felix C. Gärtner. *Formale Grundlagen der Fehlertoleranz in verteilten Systemen*. PhD thesis, Fachbereich Informatik, TU Darmstadt, May 2001.

[GK00] Felix C. Gärtner and Sven Kloppenburg. Consistent Detection of Global Predicates Under a Weak Fault Assumption. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS2000)*, pages 94–103, Nürnberg, Germany, October 2000. IEEE Computer Society Press.

[GP01] Felix C. Gärtner and Stefan Pleisch. (Im)Possibilities of predicate detection in crash-affected systems. In *Proceedings of the 5th Workshop on Self-Stabilizing Systems (WSS 2001)*, number 2194 in Lecture Notes in Computer Science, pages 98–113, Lisbon, Portugal, October 2001. Springer-Verlag.

[GP02] Felix C. Gärtner and Stefan Pleisch. Failure detection sequencers: Necessary and sufficient information about failures to solve predicate detection. In *Proceedings of the 16th International Symposium on DIStributed Computing (DISC 2002)*.

[HW92] Walter L. Heimerdinger and Chuck B. Weinstock. A conceptual framework for system fault tolerance. Technical Report CMU/SEI-92-TR-33, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, October 1992.

[Ja94] Pankaj Jalote. *Fault tolerance in distributed systems*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1994.

[KV93] Hermann Kopetz and Paulo Veríssimo. Real Time and Dependability Concepts. In Mullender [Mu93], chapter 16, pages 411–446.

[LA90] Peter A. Lee and Thomas Anderson. *Fault Tolerance: Principles and Practice*. Dependable computing and fault-tolerant systems. Springer-Verlag, Berlin ; New York, second edition, 1990.

[La77] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, March 1977.

[La92] Jean-Claude Laprie, editor. *Dependability: Basic concepts and Terminology*, volume 5 of *Dependable Computing and Fault-Tolerant Systems*. Springer-Verlag, 1992.

[Ly96] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[MN91] Keith Marzullo and Gil Neiger. Detection of global state predicates. In *Proceedings of the 5th International Workshop on Distributed Algorithms (WDAG91)*, pages 254–272, 1991.

[Mu93] Sape Mullender, editor. *Distributed Systems*. Addison-Wesley, second edition, 1993.

[Ne95] Peter G. Neumann. *Computer Related Risks*. ACM Press, 1995.

[Ru94] John Rushby. Critical System Properties: Survey and Taxonomy. *Reliability Engineering and System Safety*, 43(2):189–219, 1994.

[Sc90] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

[Sc93] Fred B. Schneider. What good are models and what models are good? In Sape Mullender, editor, *Distributed Systems*, chapter 2, pages 17–26. Addison-Wesley, second edition, 1993.

[SM94] Reinhard Schwarz and Friedemann Mattern. Detecting causal relationships in distributed computations: in search of the holy grail. *Distributed Computing*, 7:149–174, 1994.

[Ve93] Paulo Veríssimo. Real-time communication. In Mullender [Mu93], chapter 17, pages 447–490.

Felix Gärtner, Jahrgang 1970, absolvierte das Abitur am humanistischen Gymnasium Philippinum in Marburg/Lahn. Zwischen 1990 und 1998 studierte er Informatik mit Nebenfach Germanistik an der Technischen Universität Darmstadt, der Philipps-Universität Marburg und der *University of Dublin, Trinity College*. Unterstützt durch ein Promotionsstipendium im Rahmen des DFG Graduiertenkollegs "Intelligente Systeme für die Informations- und Automatisierungstechnik" an der TU Darmstadt promovierte er im Mai 2001 mit einer Arbeit über die Grundlagen fehlertoleranter Systeme, deren Ergebnisse dieser Beitrag zusammenfaßt.