Korrektheitsanalyse eingebetteter Systeme

Projekt OHB Abrixas PTC Test

Dr. Holger Schlingloff, Oliver Meyer*, Thomas Hülsing†

Entwurf vom 26. Juni 1998, bitte nicht weitergeben!

Zusammenfassung

In diesem Erfahrungsbericht wird der Einsatz eines neuartigen Verfahrens zur Qualitätssicherung von sicherheitskritischer Software in einem Raumfahrtprojekt beschrieben. Für die Energie- und Thermalkontrolle des von der OHB System GmbH, Bremen, entwickelten Satelliten "Abrixas" wurden anhand einer formalen Spezifikation Tests automatisiert generiert und durchgeführt. Die überprüften Eigenschaften umfaßten dabei den vollständigen Funktionsumfang des Steuermoduls: funktionales und reaktives Verhalten, Realzeiteigenschaften und Benutzerinteraktion, Streß- und Langzeitverhalten. Die Tests wurden vom Bremer Institut für Sichere Systeme mit der Originalhardware in einer Hardware-in-the-loop Simulationsumgebung ausgeführt und automatisch ausgewertet. Als Werkzeug kam das Testsystem VVT-RT zum Einsatz, welches vom Technologie-Zentrum Informatik (TZI) der Universität Bremen bis zur Produktreife entwickelt wurde. Durch die Universalität der Schnittstellen war hierbei nur ein geringer Anpassungsaufwand erforderlich.

1 Software-Qualitätssicherung

"Irren ist menschlich. Solange der Mensch Software schreibt, wird die Software mit Fehlern behaftet sein, unabhängig davon, in welcher Sprache bzw. auf welcher semantischen Ebene er seine Gedanken zum Ausdruck

*Universität Bremen, TZI-BISS, Pf. 330440, D-28334 HB; {hs,emm}@tzi.de
†OHB System GmbH, Universitätsallee 27-29, D-28359 HB; huelsing@ohb-system.de

bringt" [Sne 88]. Obwohl sich an dieser grundsätzlichen Aussage in den letzten zehn Jahren nicht viel geändert hat, sind doch in neuerer Zeit eine Reihe von formalen Methoden entwickelt worden, mit denen die Fehlerwahrscheinlichkeit in sicherheitskritischen Anwendungen entscheidend verringert werden kann.

Für viele technische Produkte ist die Qualität der (mit-)gelieferten Software ein zunehmender Erfolgsfaktor im internationalen Wettbewerb. In sicherheitskritischen Anwendungen, bei denen ein Versagen des Systems hohe materielle oder ideelle Schäden verursachen kann, ist die Korrektheit vielfach sogar das entscheidende Argument. Daher sind bei der Entwicklung solcher Software besonders hohe Qualitätsanforderungen einzuhalten. Prominente Beispiele wie der "Pentium Bug" oder der mißglückte Jungfernflug der Ariane5-Rakete machen deutlich, daß mangelnde Software-Qualitätssicherung erhebliche Kosten verursachen kann. Hier kann durch den Einsatz formaler Methoden nicht nur eine deutliche Erhöhung der Zuverlässigkeit, sondern gleichzeitig auch eine deutliche Senkung des erforderlichen Aufwands und der Kosten erreicht werden. Durch erhebliche Fortschritte in der Theorie sind nun seit kurzem Werkzeuge verfügbar, die es ermöglichen, solche Methoden in der industriellen Software-Erstellung einzusetzen. Das BISS im TZI unterstützt seine Projektpartner in der Anwendung und Anpassung dieser Werkzeuge.

Bevor auf die technischen Besonderheiten dieser neuen Verfahren eingegangen wird, soll hier eine kurze Abgrenzung zu konventionellen Werkzeugen der Software-Qualitätssicherung gegeben werden. Diese können in folgende Klassen eingeteilt werden:

- Versions–, Revisions– und Konfigurationsmanagement
  Der Einsatz von Werkzeugen zur Verfahrensdatenverwaltung verhindert Inkonsistenzen, die durch eine parallele Bearbeitung der selben Module von mehreren Entwicklern entstehen können. Bei größeren Software-Projekten ist der Einsatz solcher Tools unumgänglich.

- Anforderungsüberwachung und Codeinspektion
  Durch eine entsprechende Strukturierung der vom System zu erfüllenden Funktionalitäten kann mit automatisierten Werkzeugen überwacht werden, ob zu jeder Anforderung ein entsprechendes Codesegment vorhanden ist. Der Ablauf einer manuellen Überprüfung des Codes gegenüber den Anforderungen kann mit Codeinspektionswerkzeugen protokolliert werden.

- Kodierungsstandards und Softwaremetriken
  Falls in firmen– und programmiersprachenspezifischen Dokumenten sti-

listische Richtlinien festgelegt sind, an die sich die Entwickler halten sollen, kann die Einhaltung dieser Richtlinien automatisch mit den entsprechenden Präprozessoren geprüft werden. Die automatisierte Berechnung verschiedener Metriken ermöglicht eine Aussage über die Übersichtlichkeit des Programmierstils und gibt Hinweise auf mögliche Schwachstellen im System.

- Statische und dynamische Analyse
  "Intelligente" Compiler und Analysewerkzeuge können verschiedene Fehler im Code (etwa die Verwendung nichtinitialisierter Variablen und die statische Überschreitung von Indexgrenzen) bereits zur Übersetzungszeit durch Berechnung des Datenflusses feststellen. Zur Laufzeit können zusätzliche Integritätsbedingungen bei geringen Leistungseinbußen geprüft werden.

- "capture-and-replay"-Testwerkzeuge
  Solche Tools erlauben dem Benutzer, einmal aufgezeichnete Testsequenzen für veränderte Programmversionen erneut ablaufen zu lassen. In Belastungstests kann dabei das selbe Test-Script von mehreren Prozessen parallel abgearbeitet werden.

- Strukturelles Testen und Debugging
  Beim strukturellen Testen wird der Programmcode mit zusätzlichen Abfragen und Protokollmeldungen annotiert, aus denen dann nach einer Reihe von Programmabläufen die Testüberdeckung einzelner Anweisungen abgeleitet werden kann. "Intelligente" Debugger erlauben ein flexibles Setzen von Unterbrechungspunkten z.B. bei der Verletzung von Invarianzbedingungen.

- Simulation
  In einer Simulation werden Umgebung und System modelliert, um Aussagen über das Verhalten des Systems unter Last zu erhalten, noch bevor das System implementiert ist. Es existieren eine Reihe formaler Werkzeuge zur Simulation unterschiedlicher Aspekte eines Systems.

- Formale Verifikation und Modellprüfung
  In einer formalen Verifikation werden alle möglichen Abläufe in Bezug auf eine logische Anforderungsspezifikation überprüft. Aufgrund der hohen Komplexität verwendet man dazu meist ein abstraktes Modell des Systems.

Auch der extensive Einsatz dieser Werkzeuge bietet oftmals jedoch keine ausreichende Gewähr für die erforderliche Zuverlässigkeit in sicherheitskritischen

Systemen. Die genannten Werkzeuge können zwar die durchschnittliche Fehlerhäufigkeit senken, sie sind jedoch für viele Anwendungen nicht ausreichend. Hauptprobleme ergeben sich dabei etwa in folgenden Punkten.

- Obwohl die Codeinspektion dazu beitragen kann, die Vollständigkeit des Codes zu gewährleisten, ist allein durch die Zusicherung der *Existenz* eines Codesegments für eine Anforderung noch nicht dessen *Korrektheit* garantiert. Vor allem durch die parallele und verschränkte Ausführung von Modulen kommt es häufig zu unvorhergesehenen Situationen.

- Metriken liefern zwar ein objektiviertes Maß der Komplexität eines Programms, machen aber keine Aussage über mögliche Fehler. Aufgrund theoretischer Unentscheidbarkeits- und Komplexitätsresultate ist es oftmals nicht möglich, das Verhalten eines Programmes vollständig statisch vorherzubestimmen.

- Die Festlegung von Testfällen und Testsequenzen ist zeitintensiv und unsicher. Es ist bislang nur unzureichend möglich, strukturelle Kriterien anzugeben, die ein vertrauenswürdiges Maß für die Testüberdeckung bilden. So besteht z.B. die Gefahr, daß derselbe (uninteressante) Testfall mehrfach abgeprüft wird, während andere (interessante) Fälle gar nicht getestet werden. Als Abbruchkriterium für eine Testreihe gilt daher meistens der Mangel an Zeit oder Geld.

- Wenn beim Testen ein Fehler gefunden wurde, ist mit dem modifizierten Code i.a. die gesamte Testsuite nochmals zu absolvieren, da sich durch die Fehlerkorrektur neue Fehler eingeschlichen haben können. Der so entstehende Zusatzaufwand für Regressionstests ist enorm und kann potentiell bis ins Unendliche wachsen.

- Vielfach ist es wünschenswert, ein System als "Black Box" zu betrachten. Dies kann einerseits lizenz- und firmenpolitische Gründe haben, andererseits ist es im Sinne einer Abtrennung der Belange sinnvoll, Entwicklungs- und Testaktivitäten voneinander zu trennen. Die meisten auf Codeinspektion und -analyse basierenden Werkzeuge erfordern, daß die Entwickler an der Qualitätssicherung mitwirken.

- Realzeitaspekte lassen sich mit strukturellen Methoden grundsätzlich nicht überprüfen. Falls zum Testen der Code selbst modifiziert wird, differiert das Systemverhalten vom Testverhalten. Zuverlässige Aussagen über das Zeitverhalten einzelner Anweisungen sind meist nicht zu erhalten.

- Ein Hauptproblem bei der vollständigen formalen Verifikation ist die Komplexität des Suchraums. Daher werden meist nur ausgewählte sicherheitskritische Aspekte modelliert und verifiziert. Der kombinatorische Aufwand bei Simulation und Verifikation ist meist schwer abzuschätzen, was eine Skalierbarkeit dieser Verfahren erschwert.

Die geschilderten Probleme führen dazu, daß heutige Softwaresysteme nur bis zu einem gewissen Grad beim Hersteller getestet werden; die extensivsten Tests erfolgen in einer "beta-Version" beim Kunden. Verbleibende Fehler werden dann als notwendiges Übel betrachtet, das sich teilweise über mehrere Generationen der Software erstreckt.

In vielen Anwendungsfällen ist ein solches Vorgehen inakzeptabel. Von einem Satelliten kann beispielsweise keine "beta-Version" ins All geschossen werden. Nach dem Start des Satelliten müssen alle Teilkomponenten sofort und fehlerfrei funktionieren. Die nachträgliche Analyse und Korrektur von Programmierfehlern während des Flugs ist nur schwer möglich. Selbst "kleine" Fehler können zum Versagen von Experimenten, zur Verminderung der Leistungsfähigkeit des Satelliten oder im schlimmsten Fall sogar zum Verlust der Mission führen. Daher ist die Korrektheit der Software ein äußerst wichtiges Entwurfskriterium.

In Bereichen, in denen der Einsatz der Software direkten Einfluß auf Leben oder Gesundheit von Menschen haben kann (etwa in medizinischen Apparaten oder Flugzeug- und Eisenbahnsteuerungen), ist dieser nur zu verantworten, wenn nachgewiesen werden kann, daß zur Vermeidung von Fehlern alle nach dem Stand der Technik möglichen Maßnahmen ergriffen wurden. In Bereichen, in denen die Software der Produkthaftung unterliegt, können die Kosten für den Austausch der Software — vor allem bei Massenprodukten wie KFZ-Steuersystemen — die Erstellungskosten weit übersteigen. Für Individualsoftware schließlich können Fehler nicht nur einen erheblichen Vertrauensverlust beim Kunden bedeuten, sie können unter Umständen wie im oben genannten Beispiel das gesamte Projekt zum Scheitern bringen.

Zusätzlich zu den oben genannten konventionellen Methoden der Qualitätssicherung ist es heute möglich, eine weitere Erhöhung der Zuverlässigkeit mit formalen (mathematischen) Methoden zu erreichen. Da Programme abstrakte Objekte mit einer klar definierten Semantik sind, können sie einer Analyse mit logischen Methoden unterzogen werden. In der Vergangenheit scheiterte eine solche Vorgehensweise allerdings zumeist an der inhärenten Komplexität. Hier wurden nun in den letzten Jahren in der Forschung bahnbrechende Verbesserungen erzielt, die eine Anwendung auf Programme von industriell relevanten Größenordungen erlauben. Als besonders erfolgversprechend im Bereich der eingebetteten Systeme und Steuerungs- und Kontrollsoftware

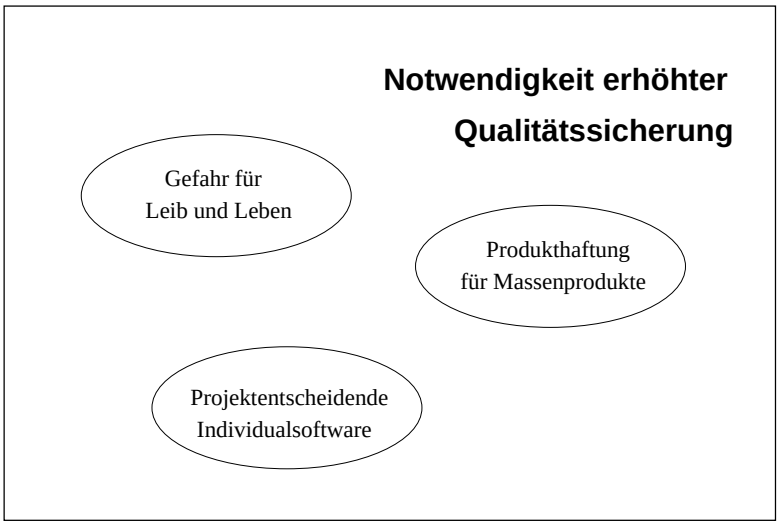

Abbildung 1: Notwendigkeit erhöhter Qualitätssicherungs-Maßnahmen

hat sich dabei *automatisiertes Testen* erwiesen.

Dabei werden aus einer logischen Spezifikation von System und Umgebungskomponenten vollautomatisch Testsequenzen generiert, mit denen das gesamte System (Hard- und Software) als "Black Box" untersucht wird. Der Testtreiber versorgt dabei das System kontrolliert mit Inputs und überwacht dessen Reaktionen (Outputs). Durch die zugrundeliegende Theorie wird gewährleistet, daß alle relevanten Testfälle irgendwann abgedeckt werden. Die vollständige Automatisierung des Testprozesses ermöglicht einen Überdeckungsgrad, der manuell mit vertretbarem Aufwand nicht zu erreichen wäre.

2 Projektbeschreibung

ABRIXAS ist ein Röntgensatellit, der von der OHB-System GmbH, Bremen, in Zusammenarbeit mit verschiedenen wissenschaftlichen Institutionen in Deutschland entwickelt wird. Die Aufgabe des Kleinsatelliten ist die wissenschaftliche Himmelsdurchmusterung im Mittelenergie-Röntgenbereich bis zu 10 keV, die von ABRIXAS erstmals vollständig durchgeführt werden soll. Die für ABRIXAS entwickelte Software ist modular aufgebaut aus Bus-Kontrolle, Lageregelung, Kamerasteuerung und Energie/Thermalkontrolle. Das System ist hochgradig redundant konzipiert: jede Hardware-Komponente des Satelliten ist mehrfach vorhanden. Auch die Software befindet sich mehrfach in verschiedenen Speicherbereichen der verwendeten Controller. Dies ist natürlich

nur ein Schutz vor Hardware-bedingten Fehlern. An die Software sind jedoch ebenfalls, wie bereits erwähnt, besonders hohe Anforderungen in bezug auf die Korrektheit zu stellen. Ein Versagen der Batterieladekontrolle etwa kann im schlimmsten Fall zum Verlust der Batterie und somit zum Scheitern der Mission führen. Daher werden bei der Entwicklung der Software extensive Qualitätssicherungsmaßnahmen eingesetzt.

Das Bremer Institut für Sichere Systeme (BISS) im Technologie-Zentrum Informatik (TZI) der Universität Bremen ist spezialisiert auf innovative Methoden der Qualitätssicherung sicherheitskritischer Software. In diesem Bereich wurde, aufbauend auf dem kommerziellen Verifikationswerkzeug FDR (**F**ailure-**D**ivergence-**R**efinement), das Testautomatisierungssystem VVT-RT (**V**alidation, **V**erification and **T**est of **R**eal **T**ime Systems) entwickelt. Im folgenden wird die Analyse der von OHB entwickelten Software zur Steuerung des Energie- und Thermalsystems (PTC, **P**ower and **T**hermal **C**ontrol) mit diesem formalen automatisierten Testsystem beschrieben.

2.1 Die ABRIXAS–PTC

Die Aufgabe der ABRIXAS–PTC ist es, die Versorgung aller aktivierter Verbraucher mit der benötigten elektrischen Energie zu gewährleisten. Die Energie wird während der Sonnenphase vom Solargenerator geliefert, in der Schattenphase versorgt die Batterie die Verbraucher. Während der Sonnenphase steuert die PTC über Stringschalter die Aufladung der Batterie, während der Schattenphase überwacht sie die Entladung bis zur maximalen Entladetiefe. Die PTC beeinflußt ferner die Stromversorgung und regelt die Temperatur verschiedener Komponenten des Satelliten (Batterie, Spiegelsystem, Kamerakopf, Sternkamera, Antennen) mittels Heizmatten. Weitere Aufgabe der PTC sind die zentrale Meßwerterfassung einer beträchtlichen Anzahl von elektrischen und thermischen Daten des Satelliten, sowie deren Übermittlung an die Bodenstation. Ein Überblick über die von der PTC gesteuerten Komponenten ist in Abb. 2 zu finden.

Die PTC ist als eine Box mit einem Motherboard realisiert, auf dem mehrere Steckkarten zur Meßdatenerfassung, Energieverteilung und Laderegelung sowie der PTC-Controller (PTCC) sitzen. Die PTCC-Karte enthält einen weltraumtauglichen 80C31 Prozessor, der über einen digitalen Bus Daten von jeder anderen Karte lesen und schreiben kann. Die einzelnen Karten erfassen über spezialisierte Meßelektronik Spannung, Strom, Druck oder Temperatur verschiedener Komponenten des Satelliten.

Die PTC Software ist in den Programmiersprachen C und Assembler codiert. Sie wird mit einem Satz Tabellen im ROM konfiguriert. Die Eingabe

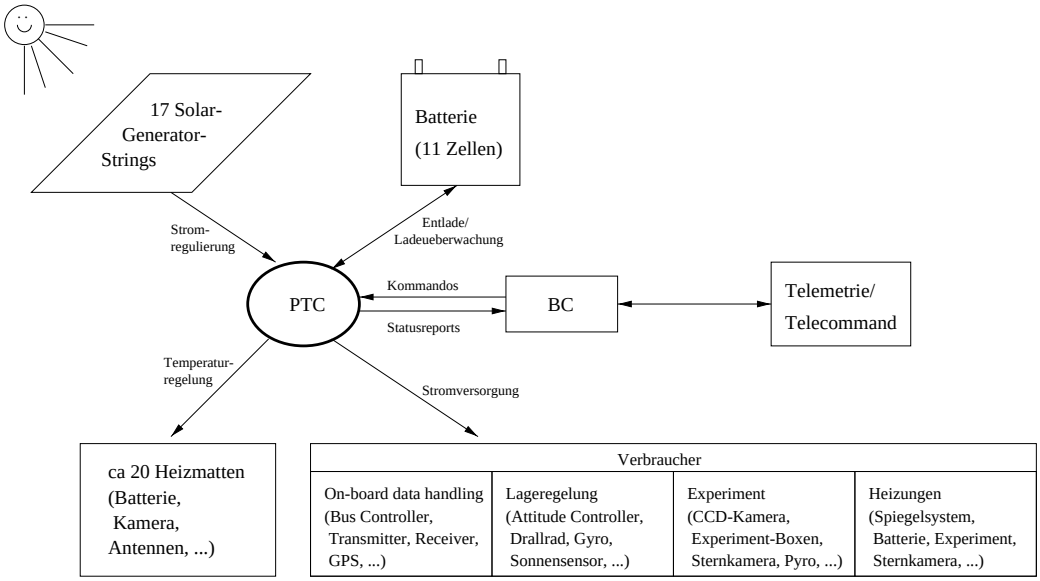

Abbildung 2: Die Umgebungskomponenten der ABRIXAS-PTC

der PTC besteht aus etwa 260 Signalen, d.h. analogen Meßwerten, die von der PTC erfaßt werden. Die PTC kontrolliert etwa 100 Schalter (über den Bus) und 70 Open-collector-Leitungen (direkte digitale Verbindungen). Von der Bodenstation kann die PTC ca. 20 verschiedene Befehlstypen mit Parametern empfangen und Nachrichten und Diagnosemeldungen übermitteln. Es existiert ferner die Möglichkeit, eine Latch-Up-Kontrolle über den Bus vorzunehmen, sowie eine zweite RS232-Schnittstelle zur anderen (redundanten) PTC. In Abb. 3 sind die Schnittstellen der PTC im Überblick aufgelistet.

Die Software der PTC ist modular aufgebaut und umfaßt etwa 36 Module. Typische Regelungsaufgaben der PTC sind es etwa, die Temperatur der einzelnen Spiegelsysteme innerhalb eines vorgegebenen Toleranzintervalls konstant auf 20°C zu halten. Eine andere Aufgabe ist es, die Batterie während der Sonnenphase kontrolliert zu laden, bis die während der Schattenphse entnommene Ladungsmenge wieder nachgeladen ist und ein gewisser Mindestabsolutdruck an der Batterie erreicht ist. Da die Hardware hochgradig fehlertolerant ausgelegt ist, muß die PTC diese Spezifikation auch bei Hardwarefehlern einhalten und ggf. korrektive Maßnahmen ergreifen.

2.2 Das Power-SCOE

Zum Test der PTC wurde von OHB eine Testumgebung, das sogenannte Power-SCOE (**S**ubsystem **C**heck-**O**ut **E**quipment) aufgebaut. Das Power-

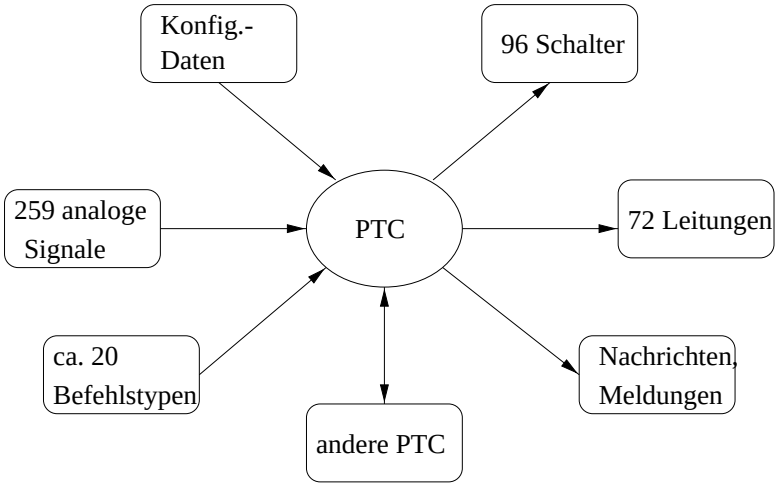

Abbildung 3: Die Schnittstellen der ABRIXAS-PTC

SCOE besteht aus folgenden Komponenten:

- Batteriesimulation (steuerbares Netzteil)
- Verbraucher-Simulatoren (regelbare Stromsenken)
- Solargenerator-Simulatoren der Firma HP
- COSMI-PC zur Ansteuerung der Simulatoren
- Extenderboard zwischen PTCC und PTC-Motherboard
- ADMEG-PC zur Meßdatenerfassung und -generierung

Ein Blockschaltbild des Power-SCOE ist in Abbildung 4 angegeben.

Zur Batteriesimulation wird ein regelbares Netzteil verwendet, da einerseits die Originalbatterie aus Kostengründen nicht zur Verfügung steht, andererseits auf diese Weise verschiedene Ladezustände der Batterie ohne großen Aufwand simuliert werden können. Bei der Batteriesimulation ist es möglich, einzelne Zellen als defekt zu markieren. Der Ladezustand der Batterie wird von der PTC anhand von zwei Druckfühlern, die außen an der Batterie angebracht sind, sowie anhand der Temperatur einzelner Batteriezellen, ermittelt. Die entsprechenden Druck- und Batteriezellen-Temperaturwerte werden im Rahmen der Batteriesimulation vom COSMI-PC generiert.

Im Satelliten sind etwa 35 Stromverbraucher, die eine variable Last erzeugen. Dazu gehören Heizmatten, Transmitter, Receiver, Bus-Controller usw.

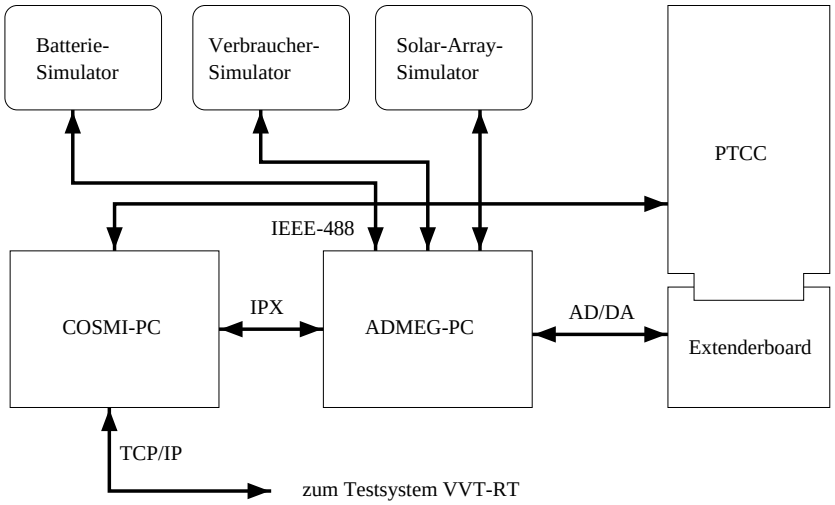

Abbildung 4: Das ABRIXAS Power-Subsystem Check Out Equipment

In der Simulation wurden die ABRIXAS-Verbraucher zunächst nur durch Software implementiert; zur Erzielung eines höheren Realitätsgrades wurde später auch eine auf Widerständen und Kondensatoren basierende Hardware-Implementierung umgesetzt.

Der Solargenerator-Simulator ist ein von der Firma Hewlett-Packard speziell für Simulationszwecke entwickeltes Netzteil, welches das Verhalten eines echten Solargenerators detailgenau nachbildet. Im vorliegenden Projekt simuliert er die elf Solar-Strings des ABRIXAS-Satelliten. Als Parameter kann dabei ein prozentualer Wert von Strom und Spannung angegeben werden, wie sie die Strings in Abhängigkeit von der Sonneneinstrahlung erzeugen. Ferner läßt sich die veränderte Entfernung von der Sonne durch Angabe eines Sommer/Winter-Flag simulieren, sowie der Alterungszustand des Solargenerators und eventuelle Defekte einzelner Strings und einzelner Stringschalter.

Der COSMI-PC führt die obengenannte Simulation der Batterie durch und ersetzt den Bus-Controller des Satelliten. Dieser ist für die Einstellung der Schalter und die Telemetrie/Telecommand Interaktion mit der Bodenstation zuständig. Die Stellung der Schalter, die analogen Meßdaten sowie die PTC-Konfigurationsdaten können vom COSMI dargestellt und von Hand manipuliert werden. Der COSMI kommuniziert mit dem ADMEG über eine IPX-Datenleitung und mit VVT-RT über ein TCP-IP 10BaseT Netz.

Die Aufgabe des ADMEG ist es, die analogen Meßwerte in Realzeit zu erfassen und sie an die PTC über das Extenderboard weiterzuleiten. Dabei ist es möglich, jeden einzelnen Wert durch VVT-RT individuell zu manipulieren. Daten, die nicht bereits durch eine Hardware- oder Software-Simulation

verfügbar sind, werden von VVT-RT auf der Grundlage einer formalen Spezifikation des Umgebungsverhaltens generiert. Auf diese Weise kann außerdem die Reaktion der PTC auf abnorme Meßwerte getestet werden, indem statt der regulären Daten manipulierte Daten bereitgestellt werden. Dadurch ist die vollständige Simulation aller Umgebungskomponenten der PTC sowohl im Normalverhalten als auch in Ausnahmesituationen möglich.

3 Das Tool VVT-RT

Das Testsystem VVT-RT wurde von der JP Software Consulting in Zusammenarbeit mit dem Bremer Institut für Sichere Systeme im Technologiezentrum Informatik der Universität Bremen entwickelt. Es basiert auf der klassischen Testtheorie von M. Hennessy [Hen 88] sowie der formalen Spezifikationssprache CSP (Communicating Sequential Processes) von C.A.R. Hoare [Hoa 85]. Die derzeitige Version von VVT-RT baut auf dem kommerziellen Verifikationswerkzeug FDR (Failure-Divergence-Refinement) von Formal Systems, Ltd., auf [FDR 94]. Eine moderne Einführung in CSP und FDR ist in der Monographie [Ros 98] zu finden.

VVT-RT bietet eine vollautomatische Durchführung folgender Arbeiten:

- Testfallgenerierung
- Testausführung und -überwachung
- Testprotokollierung und -auswertung

Für die Tests wird dabei das Pflichtenheft in eine logische Spezifikation des zu testenden Zielsystems und seiner Umgebung umgesetzt. Von VVT-RT wird aus dieser Spezifikation eine große Zahl von Testfällen generiert. Die erzeugten Testfällen werden dann von VVT-RT zur sofortigen Testdurchführung verwendet, indem das System kontrolliert mit Inputs versorgt wird, und gleichzeitig die Reaktionen (Outputs) des Systems auf ihre Zulässigkeit überprüft werden. Ein Vorteil dabei ist es, daß die interne Architektur des Zielsystems für VVT-RT vollständig verborgen ist und somit das Gesamtsystem (Hard- und Software) im Zusammenspiel getestet wird. Die zugrundeliegende Theorie garantiert dabei die Auswahl von "sinnvollen" Testfällen. Dadurch, daß der gesamte Testprozeß nach der formalen Spezifikation der geforderten Systemeigenschaften vollständig automatisch abläuft, ist ein Testüberdeckungsgrad erreichbar, der weit über das mit herkömmlichen Mitteln Erreichbare hinausgeht. Anhand der automatisch erstellten Protokolle können

Probleme im System einfach lokalisiert und beseitigt werden; der Zusatzaufwand für Regressionstest ist minimal. VVT-RT erlaubt es, Hardware-in-the-loop-Tests von Programmen auf den für sie vorgesehenen Original-Hardware Plattformen durchzuführen. Anwendungsbereiche sind neben Prozeßsteuerungssoftware und Firmware, logischen Controllern und SPS auch Kommunikationsprotokolle, fehlertolerante Rechensysteme und eingebettete Realzeitsysteme.

VVT-RT ist ein auf praktisch allen UNIX-Plattformen verfügbares System, welches mit dem Zielsystem über alle Standardschnittstellen (seriell, parallel, Ethernet usw.) kommunizieren kann. Falls dieses keine solche Standardschnittstelle anbietet, kann mit meist geringem Aufwand ein Interface-Treiber erstellt werden, der die Kommunikation mit der zu testenden Anwendung durchführt. Bei zeitkritischen Anforderungen ist es möglich, VVT-RT auf mehreren Rechnern verteilt ablaufen zu lassen und so eine prinzipiell beliebig große Geschwindigkeit zu erreichen. Es ist vorgesehen, eine Java-Anbindung an das Internet zu integrieren, mit der dann Tests über beliebige Entfernungen durchgefürt werden können.

Ein Überblick über den Aufbau von VVT-RT ist in Abbildung 5 [Pel 96a] angegeben. Die einzelnen Komponenten werden dabei anhand des vorliegenden Beispiels im Folgenden genauer erläutert.

Über den Einsatz von VVT-RT in anderen Anwendungsgebieten wird in [Pel 96b] und [AD 97] berichtet.

Im vorliegenden Projekt wurde ein K6-200 PC durch ein Standard Ethernet mit dem COSMI-PC des Power-SCOE verbunden. Aus Lizenzgründen wurden die FDR-abhängigen Teile der Tests auf einer Sun unter Solaris erstellt und auf den VVT-RT PC übertragen. Von diesem aus wurden vom Testsystem unter Linux die Tests durchgeführt und protokolliert.

Aus der formalen Spezifikation der Systemanforderungen wird vom VVT-RT **Testgenerator** mittels FDR ein sogenannter Transitionsgraph erstellt, der alle erlaubten Zustandsübergänge des Zielsystems repräsentiert. Jedem Übergang ist ein sogenanntes abstraktes *Event* zugeordnet. Im Gegensatz zu CSP unterscheidet VVT-RT dabei zwischen Input- und Output-events. Beobachtbare Änderungen an den Ausgängen des Zielsystems sind z.B. das Verschicken einer Nachricht auf dem seriellen Port, das Setzen eines Schaltbefehls auf dem Bus oder das Anlegen einer Spannung an einer Open-Collector-Leitung. Aus diesen Observablen werden in der *Event-mapping library* abstrakte Output-events generiert und an VVT-RT weitergeleitet. Umgekehrt dienen gewisse abstrakte Input-events der Spezifikation dazu, das Verhalten des Zielsystems zu beeinflussen. Beispiele hierzu sind die Modifikation von analogen Signalwerten, das Abschicken von Telekommandos an die PTC und die Steuerung

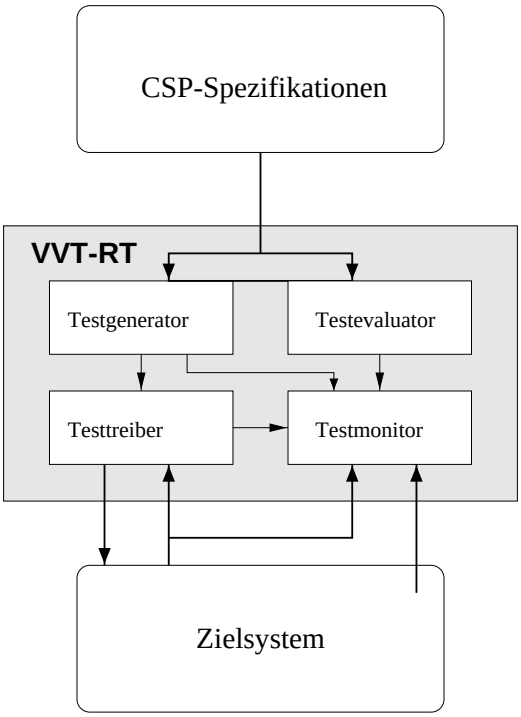

Abbildung 5: Das Testsystem VVT-RT

der Hardware-Simulatoren. Jedes solche Input-event wird von der Event-mapping-library in eine entsprechende Aktion an den Eingangskanälen des Zielsystems umgesetzt.

Bei der Durchführung von Tests wählt der **Testtreiber** von VVT-RT ein abstraktes Event entsprechend dem aktuellen Zustand und den möglichen Übergängen im Transitionsgraphen aus. Die entsprechende Aktion wird dann an das Zielsystem weitergeleitet und führt dort ggf. zu einer Reaktion. VVT-RT führt einen Zustandsübergang im Transitionsgraphen durch und überwacht, ob die beobachtbaren Reaktionen gemäß der Spezifikation erlaubt sind. Falls die Spezifikation eine erfolgte Reaktion als unzulässig erklärt oder die Reaktion nicht erwartet wurde, meldet der VVT-RT **Testmonitor** einen Fehler. Im Gegensatz zu Werkzeugen für funktionale Tests ist es dabei möglich, daß nur die Kombination mehrerer Input-events einen Output verursacht, oder daß ein einziger Input eine Kette von Outputs als Reaktion auslöst.

Der **Testevaluator** von VVT-RT sorgt dafür, daß die Testüberdeckung streng monoton zunimmt. Wenn in einem gegebenen Zustand des Transitionsgraphen mehrere Input-events möglich sind, wird dasjenige ausgewählt, welches in dieser Situation bislang am seltensten selektiert wurde. Dadurch

wird eine Testüberdeckung gewährleistet, die monoton mit der Länge der Tests zunimmt: Wenn das Testsystem nur lange genug läuft, wird jede Kombination von Ereignissen innerhalb des Transitionsgraphen irgendwann einmal getestet werden.

Bei reaktiven Systemen, die im allgemeinen nicht terminieren, ist jeder zulässige Ablauf unendlich. Im vorliegenden Fall wurden die Tests daher "on-the-fly" durchgeführt, während die PTC kontinuierlich in Betrieb war. Dabei war es möglich, verschiedene Eigenschaften zu überprüfen und gleichzeitig Belastungs- und Langzeittests während des selben Testlaufs durchzuführen.

4 Spezifikation der Anforderungen

Im vorliegenden Projekt wurden in Zusammenarbeit mit OHB vom TZI alle relevanten Eigenschaften identifiziert, welche das geforderte Leistungsspektrum der PTC abdecken. Jede dieser Eigenschaften entspricht dabei einer zu prüfenden Funktionalitätsklasse. Die Anforderungen lassen sich dabei in drei Gruppen einteilen:

- Anforderungen für die Schaltfunktionen,
- Anforderungen für die Energiekontrolle und
- Anforderungen für die Thermalkontrolle.

Als Beispiel für eine Anforderung an die Schaltfunktionalität sei hier das manuelle Schalten eines Verbrauchers von der Bodenstation aus erwähnt. Dieser Schaltbefehl wird der PTC über den Bus-Controller zugestellt. Die PTC überprüft den Befehl auf syntaktische und semantische Korrektheit und führt ihn dann aus, indem sie ein (dem gewünschten Zustand entsprechendes) Signal auf den Bus setzt. Sie überprüft dann anhand der an dem Schalter anliegenden Strom- und Spannungsdaten, ob der entsprechende Schalter tatsächlich geschaltet hat, und sendet eine Bestätigung an die Bodenstation. Eine einfache Anforderung hierbei ist in Abbildung 6 angegeben.

Als Beispiel für eine Anforderung an die Energiekontrolle sei hier ein zentraler Teil der Ladekontrolle erwähnt. Die Batterie des Satelliten ist hochempfindlich und sollte immer kontrolliert entladen und wieder aufgeladen werden. Die Aufladung erfolgt dabei in zwei Phasen: Haupt- und Ergänzungsladung. Während der Hauptladung ist es wichtig, die Batterie relativ schnell mit einem hohen Ladestrom aufzuladen, während bei der Ergänzungsladung der normierte Druckverlauf und die Batterietemperatur einen Einfluß auf den

Zu jedem Zeitpunkt ist es möglich, Befehle zum Ein- oder Ausschalten aller Verbraucher zu senden. Die notwendigen Schaltoperationen werden dann innerhalb der Zeitkonstante `T_Swtch_Consumer` ausgeführt.

Abbildung 6: Anforderung für die Schaltfunktionalität

Ladestrom haben. Die genaue Anforderung für die Hauptladung ist in Abbildung 7 definiert.

Während der Sonnenphase fließt ein Ladestrom von `I_Charge`, solange bis die während der letzten Schattenphase entnommene Ladungsmenge nachgeladen ist und der Mindestabsolutdruck `P_minabs` erreicht ist, oder bis die Sonnenphase beendet ist.

Abbildung 7: Anforderung für die Energiekontrolle

Als drittes und letztes Beispiel sei hier eine Anforderung für die Thermalkontrolle des Kamerakopfes erwähnt. Die Aufgabe der PTC ist es, diese Temperatur mit Hilfe der Heizmatten "möglichst konstant" zu halten. Die in Abbildung 8 dargestellte Anforderung für das Normalverhalten repräsentiert den generischen Typ eines einfachen Regelkreises. Im Fall von Hardwareproblemen, etwa defekten Heizmatten oder Stromknappheit, ist diese einfache Regelung allerdings nicht ausreichend.

Die Temperatur der CCD-Kamera ist immer innerhalb fest vorgegebener Grenzen. Bei eingeschalteter Kamera muß die Heizung deaktiviert sein. Wenn die Kamera ausgeschaltet ist, gilt die folgende Regelung: Falls die Temperatur des CCD's kleiner als `H_CCD_opt-H_CCD_tol` ist, wird die entsprechende Heizung eingeschaltet. Falls die Temperatur des CCD's größer als `H_CCD_opt+H_CCD_tol` ist, wird die Heizung ausgeschaltet.

Abbildung 8: Anforderung für die Thermalkontrolle

Damit aus den Anforderungen automatisch eine (im allgemeinen sehr große) Anzahl von Testfällen mit unterschiedlichen Daten generiert werden kann, müssen diese zunächst formalisiert werden. Die Formalisierung erfolgte in der Spezifikationssprache CSP und wurde im vorliegenden Fall vom

TZI durchgeführt. Das TZI bietet seinen Projektpartnern im Rahmen der Technologietransfer-Aktivitäten allerdings auch Schulungen in CSP an, so daß diese Codierung letztlich vom Systementwickler vorgenommen werden kann.

Zunächst wurden die Schnittstellen der einzelnen Anforderungen erfaßt und systematisiert. Dann wurden Abhängigkeiten zwischen den Anforderungen lokalisiert und die Anforderungen wurden in einzelne Gruppen eingeteilt. Für jeden analogen Kanal wurden in Abstimmung mit OHB die Schwellwerte festgelegt, deren Über- oder Unterschreitung zur Auslösung eines Events führt. Die Event-mapping-library wurde erstellt und die Kommunikation mit dem Power-SCOE mit der bei OHB verwendeten Script-Sprache realisiert.

```
SPEC = ( SWITCHCONS [|{| Tau_nextTC |}|] TCTIM )  |||  TIMCHK

SWITCHCONS =  Tau_nextTC -> (
     (Com_PYRO_PWR_CONS_ON -> setTimSwt ->
      Swt_BS_ON_MAIN_ON -> Swt_PYRO_PRE_MAIN_ON -> Swt_PYRO_PWR_MAIN_ON ->
      resTimSwt -> SWITCHCONS)
 |~| (Com_PYRO_PWR_CONS_OFF  -> setTimSwt  ->
      Swt_PYRO_PWR_MAIN_OFF -> resTimSwt -> SWITCHCONS)
 |~|  ... )

TIMCHK = elaTimSwt -> errorSwitchTimer -> TIMCHK

TCTIM = Tau_nextTC -> setTimTick -> elaTimTick -> TCTIM
```

Abbildung 9: CSP-Code für die Anforderung in Abbildung 6

Danach erfolgte die CSP-Codierung der Anforderungen. Exemplarisch seien hier die zu den drei obengenannten Beispielen gehörenden CSP-Testprozeduren beschrieben. Abbildung 9 stellt einen Auszug aus dem CSP-Code für die Anforderung in Abbildung 6 dar. Die gesamte formale Spezifikation `SPEC` besteht aus drei parallelen Teilprozessen `SWITCHCONS`, `TIMCHK` und `TIMER`. Der Prozeß `SWITCHCONS` wählt nichtdeterministisch einen zu schaltenden Konsumenten aus (im Beispiel ist nur der Konsument `PYRO` zum Öffnen des Teleskopdeckels des Spiegelsystems). Er löst dann ein Event `Com_PYRO_PWR_CONS_ON` bzw. `Com_PYRO_PWR_CONS_OFF` aus, welches von der Event-mapping-library in ein entsprechendes Telekommando zum Ein- bzw. Ausschalten des Pyro umgesetzt wird. Dann aktiviert er einen Timer `TimSwt` zur Überwachung der in der Spezifikation geforderten Zeitkonstante `T_Swtch_Consumer`. Zum korrekten Einschalten des Pyro muß die PTC die drei Schalter `Swt_BS_ON_MAIN_ON`, `Swt_PYRO_PRE_MAIN_ON` und `Swt_PYRO_PWR_MAIN_ON` der Reihe nach betätigen. Der Prozeß `SWITCHCONS` wartet auf die entsprechenden Events und setzt, falls sie rechtzeitig eintref-

fen, den Timer TimSwt zurück und verhindert damit sein Ablaufen. Zum Ausschalten des Pyro genügt es, den Schalter Swt_PYRO_PWR_MAIN_OFF zu betätigen. Alle Tests laufen in einer Endlosschleife, die hier durch einen rekursiven Aufruf realisiert wird. Der Prozeß TIMCHK wird verschränkt zum Prozeß SWITCHCONS ausgeführt. Sobald der Timer TimSwt abläuft, wird von VVT-RT das Event elaTimSwt erzeugt und diesem Prozeß zugestellt, woraufhin er ein entsprechendes Fehler-Event generiert. Auf diese Weise werden Fehler im Timing erkannt, ohne daß die PTC angehalten oder neu gestartet werden muß.

Erste Tests mit dieser Codierung ergaben, daß die PTC nicht in der Lage war, die Spezifikation einzuhalten, da das Testsystem hier Befehle sehr schnell hintereinander absetzt. Sobald ein Schalter korrekt geschaltet wurde, verlangt das Testsystem ohne Verzögerung, den nächsten Schalter umzuschalten. Diese Situation ist für den realen Betrieb, in dem die Schaltkommandos vom Bedienpersonal am Boden abgesetzt werden, unrealistisch. Die informelle Anforderung, daß "zu jedem Zeitpunkt" Schaltbefehle akzeptiert werden, wurde um die Vorbedingung ergänzt, daß pro Sekunde nur ein Befehl gesendet wird. In der CSP-Formalisierung wurde diese Vorbedingung durch einen zusätzlichen Prozeß TCTIM realisiert, welcher über das interne Event Tau_nextTC mit dem Prozeß SWITCHCONS synchronisiert wird und diesen um die geforderte Zeitspanne verzögert. Der sich ergebende Code ist in Abbildung 9 dargestellt.

```
LADEREGELUNG = Tau_SUN_ON -> setTimLaderegelung ->
               HAUPTLADUNG(false, false)

HAUPTLADUNG(P_BATT_PRESS_1_OK, Nachgeladen) =
    (Evt_I_BATT_MAIN_IS_I_Charge -> resTimLaderegelung ->
               HAUPTLADUNG(P_BATT_PRESS_1_OK, Nachgeladen))
 [] (elaTimLaderegelung -> errorLaderegelung -> setTimLaderegelung ->
               HAUPTLADUNG(P_BATT_PRESS_1_OK, Nachgeladen))
 [] (Evt_I_BATT_MAIN_ISNOT_I_Charge -> setTimLaderegelung ->
               HAUPTLADUNG(P_BATT_PRESS_1_OK, Nachgeladen))
 [] (Evt_P_BATT_PRESS_1_OK ->
      if Nachgeladen then ERGAENZUNGSLADUNG
                     else HAUPTLADUNG(true, Nachgeladen))
 [] (Evt_Nachgeladen ->
      if P_BATT_PRESS_1_OK then ERGAENZUNGSLADUNG
                           else HAUPTLADUNG(P_BATT_PRESS_1_OK, true))
 [] (Tau_SUN_OFF -> ENTLADEUEBERWACHUNG)
```

Abbildung 10: CSP-Code für die Anforderung in Abbildung 7

Abbildung 10 zeigt die Formalisierung der Anforderung an die Hauptladeregelung in der Energiekontrolle. Die Laderegelung beginnt mit der

Hauptladung, sobald die Sonne aufgeht. Innerhalb der durch den Timer `TimLaderegelung` vorgegebenen Zeit muß die PTC den maximalen Ladestrom `I_Charge` einstellen. Das Erreichen dieses Wertes (innerhalb einer einzustellenden Genauigkeit) wird durch einen analogen Überwachungsprozeß von VVT-RT durch das Event `Evt_I_BATT_MAIN_IS_I_Charge` angezeigt. Falls der Prozeß `HAUPTLADUNG` dieses Event erhält, bevor der Timer `TimLaderegelung` abläuft, setzt er ihn zurück; falls der Timer abläuft, bevor der geforderte Ladestrom erreicht ist, wird ein Fehler angezeigt. Wenn während der Hauptladung aufgrund von Schaltvorgängen der Ladestrom den Bereich um `I_Charge` verläßt, so wird der Timer `TimLaderegelung` erneut gesetzt.

Die Spezifikation fordert, daß die Hauptladung solange läuft, bis die während der letzten Schattenphase entnommene Ladungsmenge nachgeladen ist und der Mindestabsolutdruck `P_minabs` erreicht ist, oder bis die Sonnenphase beendet ist. In der CSP-Kodierung werden die Zustandsvariablen "nachgeladen" und "Mindestabsolutdruck erreicht" durch boolesche Parameter des Prozesses `HAUPTLADUNG` realisiert. Falls das Event `Evt_P_BATT_PRESS_1_OK` eintrifft, welches vom Testsystem beim Erreichen des Mindestabsolutdrucks ausgelöst wird, und der Prozeß im Zustand `Nachgeladen` ist, wird die Überwachung der Ergänzungsladung gestartet; ansonsten ist die Hauptladung noch nicht beendet. Das Testsystem erzeugt das Event `Evt_Nachgeladen`, wenn das Integral über den Ladestrom den Wert des Integrals über den Entladestrom der letzten Schattenphase erreicht. Auf diese Weise ist es möglich, beliebig komplexe hybride Eigenschaften ohne besonderen Zusatzaufwand zu überprüfen. Der CSP-Code für die Laderegelung ist in Abbildung 10 zu finden.

Auf die Codierung für die Thermalkontrolle der CCD-Kamera soll hier nur kurz eingegangen werden. Das Temperaturverhalten der CCD-Kamera wird von einem parallel zu den Tests ablaufenden Algorithmus simuliert. Die in Abbildung 11 angegebenen CSP-Prozesse beschreiben sowohl das angenommene Verhalten der Umgebung bei einer Aktivierung und Deaktivierung der Heizer, als auch das geforderte Verhalten des CCD-Heizer-Regelungsalgorithmus. Alle mit `Evt_` beginnenden Events stellen Ereignisse zur Kommunikation mit der Simulation des Temperaturverhaltens dar. Beispielsweise wird, wann immer mindestens einer der beiden Heizer (Main/Redundant) aktiv ist, die Umgebungssimulation veranlaßt, eine Temperaturkurve auszuwählen, die einen realistischen Anstieg der CCD-Kamera-Temperatur zur Folge hat (`Evt_warmer`). Von VVT-RT wird dabei wieder die Einhaltung von Schwellwerten überwacht; falls z.B. der erste obere Schwellwert (`H_CCD_opt` + `H_CCD_tol`) überschritten wird, so generiert die Simulationskomponente das Event `Evt_warm`. Die Spezifikation verlangt im diesem

```
ED = TC2_CCD(tok,false,false)
-- 1. Parameter: Temperaturstatus, initial o.k.
-- 2. Parameter: Heizerstatus MAIN, initial aus
-- 3. Parameter: Heizerstatus RED,  initial aus

-- Thermal Control Testcase2, Inner1
TC2_CCD(temp,h_main,h_red) =
   Swt_CCD_HZR_MAIN_ON -> Evt_warmer -> TC2_CCD(temp,true,h_red)
   [] Swt_CCD_HZR_RED_ON -> Evt_warmer -> TC2_CCD(temp,h_main,true)
   [] Swt_CCD_HZR_MAIN_OFF ->
       (if (not h_red) then Evt_colder -> SKIP else SKIP);
       TC2_CCD(temp,false,h_red)
   [] Swt_CCD_HZR_RED_OFF ->
       (if (not h_main) then Evt_colder -> SKIP else SKIP);
       TC2_CCD(temp,h_main,false)
   [] Evt_too_warm -> errorTooWarm -> setTim1 ->
          TC2_CCD(temp,h_main,h_red)
   [] Evt_warm -> setTim1 -> TC2_CCD(twarm,h_main,h_red)
   [] Evt_ok -> resTim1 -> TC2_CCD(tok,h_main,h_red)
   [] Evt_cold -> setTim1 -> TC2_CCD(tcold,h_main,h_red)
   [] Evt_too_cold -> errorTooCold -> setTim1 ->
          TC2_CCD(temp,h_main,h_red)
   [] elaTim1 -> ( if ( (temp == twarm and (h_main or h_red))
            or (temp == tcold and (not (h_main or h_red))))
          then (errorTooLate -> setTim1 -> TC2_CCD(temp,h_main,h_red))
          else TC2_CCD(temp,h_main,h_red) )
   [] Swt_EXP_PWR_MAIN_ON ->
       ((Swt_CCD_HZR_MAIN_OFF -> Swt_CCD_HZR_RED_OFF -> SKIP)
         [] (Swt_CCD_HZR_RED_OFF -> Swt_CCD_HZR_MAIN_OFF -> SKIP));
        Evt_constOK -> EXPON(main)
   [] Swt_EXP_PWR_RED_ON ->
       ((Swt_CCD_HZR_MAIN_OFF -> Swt_CCD_HZR_RED_OFF -> SKIP)
         [] (Swt_CCD_HZR_RED_OFF -> Swt_CCD_HZR_MAIN_OFF -> SKIP));
             Evt_constOK
     -> EXPON(red)

EXPON(MAIN_RED) =
   Swt_CCD_HZR_MAIN_ON -> errorHeaterOnWhileExpOn -> EXPON(MAIN_RED)
   [] Swt_CCD_HZR_RED_ON -> errorHeaterOnWhileExpOn -> EXPON(MAIN_RED)
   [] Swt_EXP_PWR_MAIN_OFF ->
       (if (MAIN_RED==main) then (Evt_colder -> TC2_CCD(tok,false,false))
                            else EXPON(MAIN_RED))
   [] Swt_EXP_PWR_RED_OFF ->
        (if (MAIN_RED==red)
         then (Evt_colder -> TC2_CCD(tok,false,false))
         else EXPON(MAIN_RED))
```

Abbildung 11: CSP-Code für die Anforderung in Abbildung 8

Fall eine Aktivierung eines der beiden CCD-Heizer innerhalb einer vorgegebenen Zeit. Bei Überschreitung des zweiten, höherliegenden Schwellwerts (`Evt_too_warm`) liegt in jedem Fall ein Fehlverhalten des Regelalgorithmus vor.

Die Regelung soll nur dann aktiv sein, wenn die Kamera abgeschaltet ist. Wird sie eingeschaltet (`Swt_EXP_PWR_xxx_ON`), reicht die Verlustwärme der Kamera als Heizung. Der Prozeß `EXPON` überwacht dann, daß die Regelung tatsächlich deaktiviert ist, d.h. daß die Heizer nicht mehr angeschaltet werden. Die Simulation wird in diesem Fall durch das Event `Evt_constOK` angewiesen, die Temperatur als konstant vorzugeben.

5 Resultate beim Automatisierten Testen

Die CSP-Testprozeduren wurden vom Testtreiber mit der oben beschriebenen Konfiguration automatisch ausgeführt. Die Testresultate konnten sowohl interaktiv mit den Erwartungsresultaten verglichen als auch automatisch ausgewertet werden. Durch die vollständige Automatisierung des Testprozesses konnten die Tests mit einer Kosteneffizienz und einer Testüberdeckung durchgeführt werden, die bei manuellem Testen nicht möglich gewesen wäre. Der Hauptaufwand bei der Vorgehensweise lag einerseits in der Anpassung des auf TCP/IP aufgesetzten Kommunikationsprotokolls an die speziellen von OHB verwendeten Formate, andererseits in der formal exakten Beschreibung der gewünschten Systemanforderungen.

Als Besonderheit ergab sich hierbei, daß das genaue Sollverhalten der PTC in vielen Fällen noch nicht im Detail spezifiziert war. Viele der aufgedeckten Probleme bezogen sich daher auch auf Unvollständigkeiten der Spezifikation und, daraus resultierend, auf nicht vorhersagbares Kontrollverhalten der PTC in unvorhergesehenen Situationen. Erst durch einen Rückkopplungsprozess mit den Systemdesignern wurden hier die Anforderungen festgelegt und führten dann teilweise zu einer Änderung der Implementierung.

Kleinere Fehler betrafen falsche Tabelleneinträge und Versionskonflikte bei den Konfigurierungsdaten. Als Beispiel sei hier ein Fehleintrag in der Schaltertabelle über die Anzahl der zu aktivierenden Schalter zum Öffnen der Schutzklappe vor der CCD-Experimentenkamera erwähnt. Dieser führte dazu, daß bei dem Kommando zum Öffnen des Teleskopdeckels nur einer von zwei notwendigen Schaltern geschlossen wurde und sich somit nach dem Aussetzen der Deckel nicht direkt geöffnet hätte, sondern zusätzlich ein manueller Eingriff von der Bodenstation notwendig geworden wäre.

Andere aufgedeckte Spezifikationsverletzungen betrafen direkt die interne Codierung der PTC. Beispielsweise führte ein Vorzeichenfehler in der PTC beim Vergleich von Lade- und Entlademenge dazu, daß die während der letzten Schattenphase entnommene Lademenge sofort zu Beginn der Sonnenphase als nachgeladen betrachtet wurde. Dies bedeutet, daß entsprechend der oben wiedergegebenen Spezifikation für die Hauptladung nur der Mindestabsolutdruck herangezogen wurde und somit bei einem Ausfall der Druckmesser keine redundante Möglichkeit existiert hätte, um die notwendige Ladedauer der Batterie zu bemessen.

Durch das systematische Testen wurden also hauptsächlich Probleme aufgedeckt, die Ausnahmesituationen betrafen und sich im "Normalfall" nie ergeben hätten. Infolgedessen wären sie durch die üblichen Debugging- und Testmethoden wahrscheinlich nicht erkannt worden. Für sicherheitskritische Anwendungen, in denen das System auch in unvorhergesehenen Situationen noch zuverlässig funktionieren soll, ist die hier angewendete Vorgehensweise als sehr erfolgreich zu bewerten.

Durch die enge Zusammenarbeit mit den Entwicklern der PTC-Software ließen sich die meisten erkannten Probleme sofort beseitigen. Unvollständigkeiten in der Spezifikation, besonders, so weit sie das von der PTC erwartete Ausnahmeverhalten bei Hardwaredefekten betrafen, wurden von OHB in Ergänzungsdokumenten zum Pflichtenheft nachgereicht und vom TZI in die CSP-Spezifikation mit eingearbeitet. Durch die Automatisierung konnten dann Regressionstests mit der veränderten Software durchgeführt werden, wobei praktisch kein zusätzlicher Aufwand entstand.

Insgesamt läßt sich konstatieren, daß der Einsatz des Testwerkzeugs VVT-RT sowohl für OHB als auch vom TZI aus erfolgreich verlief. Als Haupterfahrung wurde von OHB die Überzeugung geäußert, daß sich die neue Technologie im vorliegenden Projekt bewährt hat. Es wurde betont, daß eine frühzeitige Anwendung formaler Methoden bereits während der Design- und Analysephase, auf jeden Fall aber parallel mit der Codierung, wahrscheinlich einen noch größeren Gewinn erbringen würde. Daher ist eine Fortsetzung der Kooperation geplant.

Literatur

[AD 97] P. Amthor, S. Dick: Test eines Bordcomputers für ein dezentrales Zugsteuerungssystem unter Verwendung des Werkzeuges VVT-RT. In: *7. Kolloquium Software-Entwicklung - Methoden, Werkzeuge, Erfahrungen: Mächtigkeit der Software und ihre Beherrschung* (Sep. 1997)

[FDR 94] Formal Systems (Europe), Ltd: Failures-Divergence-Refinement FDR 1.4, User Manual and Tutorial. (1994)

[Hen 88] M.C. Hennessy: Algebraic Theory of Processes. MIT Press (1988).

[Hoa 85] C.A.R. Hoare: Communicating Sequential Processes. Prentice Hall International (1985).

[Pel 96a] J. Peleska: *Formal Methods and the Development of Dependable Systems.* Bericht 9612, Christian-Albrechts-Universität Kiel, Institut für Informatik und Praktische Mathematik (1996)

[Pel 96b] J. Peleska: Test Automation for Safety-Critical Systems: Industrial Application and Future Developments. In: M.-C. Gaudel, J. Woodcock (eds.): *FME'96: Industrial Benefit and Advances in Formal Methods.* Springer LNCS 1051 (1996), pp. 39–59.

[Ros 98] A.W. Roscoe: The Theory and Practice of Concurrency. Prentice Hall International (1998).

[Sne 88] H.M. Sneed: Software-Testen. Informatik-Spektrum 11:303-311 (1988).