

Integration modellbasierter Entwurfsverfahren in Softwareverifikation und -entwicklungsprozesse

Eva Kalix, Dr. Stefan Bunzel

Continental Teves AG & Co.oHG, Guerickestr. 7, D-60488 Frankfurt a. M.
eva.kalix@conti.de, stefan.bunzel@contiteves.com

Abstract: Derzeit unternehmen viele Firmen Anstrengungen von C-Programmierung zu modellbasierter Entwicklung zu wechseln. Dieser Artikel befasst sich mit den Auswirkungen, die dieser Wechsel auf Testmöglichkeiten und Entwicklungsprozesse hat.

Der Qualitätsbegriff in der Softwareentwicklung umfasst grundsätzlich zwei Aspekte. Zum einen die korrekte Abarbeitung eines Algorithmus, zum anderen die Zuverlässigkeit und Reproduzierbarkeit des Softwareerstellungsprozesses.

Beide Aspekte werden am Beispiel des weit verbreiteten modellbasierten Entwurfswerkzeugs Simulink dargestellt.

Der erste Abschnitt behandelt die Verifikationsmöglichkeiten für eine gegebene Plattform. Der zweite Abschnitt widmet sich dem Softwareentwicklungsprozess in großen Teams.

1 Verifikation modellbasierter Entwürfe

Die modellbasierte Entwicklung (MBE) von Software erfolgt im Allgemeinen in den folgenden, in Abb. 1 dargestellten Schritten:

1. Funktionsentwurf, zunächst als Algorithmus mit Floating-Point(Gleitkomma) Arithmetik („Golden Device“)
2. Fixed-Point(Festkomma) Implementierung obiger Funktion
3. Konfiguration der Funktion für den Codegenerator

Ein großer Vorteil der modellbasierten Entwicklung gegenüber traditioneller C-Programmierung ist die Möglichkeit, in Einzelschritten vom Funktionsentwurf zum Steuergerätee-code und bei Bedarf auch wieder zurück zu gelangen.

Es wird durchgängig am gleichen Quelldokument (Modell) gearbeitet, wobei dem Modell in jedem Implementierungsschritt mehr Informationen über die Zielhardware hinzugefügt werden. Damit werden auch schrittweise mehr von den spezifischen Restriktionen der Zielplattform wirksam.

Diese Zerlegung des Entwicklungsprozesses in Einzelschritte ermöglicht eine Unterscheidung von Entwurfs- und Implementierungsfehlern. Die verschiedenen Fehlerarten können in unterschiedlichen Entwicklungsschritten überprüft werden.

Ein weiterer wesentlicher Vorteil der MBE ist, dass die beschriebenen Entwicklungsschritte separat in einem iterativen Prozess mehrfach durchlaufen werden können. Mit derartigen „kleinen“

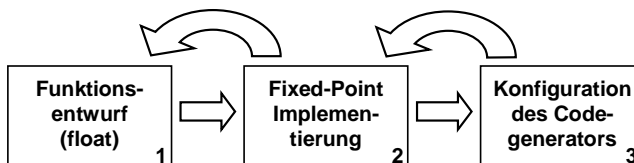


Abbildung 1: Stufen des modellbasierten Entwicklungsprozesses

Iterationsschleifen ist es möglich, Fehler bereits in frühen Phasen des Entwicklungsprozesses aufzuspüren. Gerade bei Iterationsschleifen ist ein einfacher, informationskonservierender Schritt zurück zur vorigen Stufe wichtig, damit bereits hinzugefügte Informationen nicht verloren gehen bzw. nicht jedesmal wieder neu hinzugefügt werden müssen.

1.1 Testmöglichkeiten während des Entwicklungsprozesses

Dieser Abschnitt beschreibt verschiedene Verifikations- und Validierungsszenarien, die bei modellbasierter Entwicklung mit verhältnismäßig geringem Aufwand in den täglichen Entwicklungsprozess integriert werden können.

Es werden dabei für jeden der drei genannten Entwicklungsschritte Testmöglichkeiten und Werkzeuge beschrieben. Derzeit sind in den meisten Firmen vorrangig die folgenden beiden, auf MATLAB¹ basierenden Toolketten im Einsatz:

- Simulink/Stateflow² mit Codegenerator TargetLink³ (SL/TL)
- Simulink/Stateflow mit Codegenerator Embedded Coder⁴ (SL/EC)

Wo Unterschiede bestehen, wird wenn möglich für beide Toolketten dargestellt, wie bestimmte Verifikationsziele umgesetzt werden können.⁵

1.1.1 Funktionsentwurf

Beim Testen eines Funktionsentwurfs wird validiert, ob die Funktion die spezifizierten Eigenschaften erfüllt. Die Spezifikation geht in die Simulation typischerweise in Form von definierten Eingangssignalen und zu erwartendem Ausgangsverhalten ein.

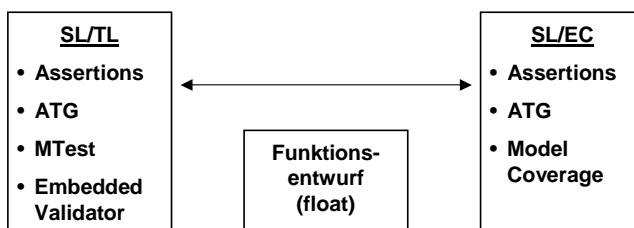


Abbildung 2: Zuordnung der vorgestellten Werkzeuge zu den beiden Toolketten SL/TL und SL/EC

¹ von The MathWorks

² originäre MathWorks-Produkte

³ von dSPACE

⁴ von The MathWorks

⁵ Aussagen über SL/TL beziehen sich auf die Versionen SL 4.x/TL 1.3p2 und/oder SL 6.x/TL2.0 (aktuelle Version), Aussagen über SL/EC beziehen sich auf die Version SL 6.x/EC 4.1 (aktuelle Version).

In der MBE dient die Simulationsumgebung unmittelbar als Entwicklungsplattform. Die Implementierung wird direkt in der Testumgebung vorgenommen.

Das grundsätzliche Problem beim simulationsbasierten Testen ist immer die Erstellung von Testdatensätzen, die die Spezifikation möglichst vollständig abprüfen. Zur Unterstützung dieser Aufgabe existieren eine Reihe von Tools, die entweder in der Grundausstattung der Werkzeugkette enthalten oder als Zusatzbausteine verfügbar sind. Eine Übersicht solcher Produkte findet sich in [Ran03]. Eine Zuordnung der hier vorgestellten Werkzeuge zu den beiden Toolketten zeigt Abb. 2.

Assertions

In den neueren Simulink Versionen (ab 5.x) ist das Simulink Standard Blockset um die Bibliothek „Model Verification“ erweitert worden. Diese Bibliothek enthält Blöcke, mit denen überprüft werden kann, ob ein bestimmtes Signal eine bestimmte Bedingung einhält. Eine Darstellung des Einsatzes von Zusicherungen (Assertions) findet sich in [Rau02].

Diese Verifikationsblöcke können durchgängig in allen Entwicklungsstufen im Modell verbleiben und es besteht die Möglichkeit, alle Verifikationsblöcke eines Modells zentral zu deaktivieren. Damit ist sichergestellt, dass die Verifikationsblöcke bei der Codegenerierung keinen zusätzlichen Code im Steuergerät mit sich bringen.

Die Model Verification Blöcke werden von beiden genannten Toolketten unterstützt.

Model Coverage

Ein Indikator für die Güte von Testsignalen ist der Grad der Modellabdeckung.

Für Stateflow ist eine Überprüfung des Abdeckungsgrades von Zuständen und Transitionen in allen Versionen enthalten. Als Verfahren wird MCDC (Modified condition/decision coverage) verwendet.

Kontrollstrukturen werden aber auch oft in Simulink modelliert. Um den Grad der Abdeckung aller Signalpfade zu prüfen, gibt es auch für Simulink ein Model Coverage Tool. Dies war zuvor Teil der „Simulink Performance Tools“ und ist nun in „Simulink Verification and Validation“ enthalten. In [Ald02] finden sich weitere Erläuterungen zum Einsatz der Modellabdeckungsanalyse und den dazu verwendeten Methoden.⁶

Systematische Testerstellung

Zur Unterstützung der systematischen Erstellung von Testfällen existiert das Tool MTest [LBE04]. MTest verwendet die Klassifikationsbaummethode als Testpartitionierungsverfahren. MTest wird derzeit als Teil des Produktes Automation Desk von dSPACE angeboten und ist damit nur für die Toolkette SL/TL verfügbar⁷.

Eine umfangreiche Sammlung von Literatur zu diesem Thema bietet www.systematic-testing.com.

⁶Die Stateflow Lösung zur Überprüfung des Abdeckungsgrades funktioniert auch mit TargetLink. Das Model Coverage Tool kann nur über den Umweg der Konvertierung nach Simulink verwendet werden.

⁷Prinzipiell arbeitet MTest mit beiden Toolketten. Durch die Einbettung in das Produkt Automation Desk ist die Verwendung aber nur in der Toolkette SL/TL sinnvoll.

Automatische Testvektorgenerierung

Zur Unterstützung der automatischen Erstellung von Testeingangssignalen werden für viele Simulationstools Programme zur automatischen Testvektorgenerierung (ATG) angeboten. Speziell für Simulink und Stateflowmodelle gibt es z. B. die Produkte Reactis⁸ oder TVGS⁹. Da diese Produkte direkt mit Simulink Modellen und nicht mit generiertem Code arbeiten, sind sie für beide Toolketten einsetzbar.

Formale Verifikation

Neben der Simulation finden sich gerade auf der Ebene des Funktionsdesigns auch formale Methoden der Verifikation und Validierung. Für die Toolkette SL/TL wird hierfür seit kurzem das Produkt Embedded Validator von OSC angeboten [BBS04]. Dieses Werkzeug dient dazu, bestimmte Eigenschaften eines finiten Zustandsautomaten nachzuweisen. Mit Hilfe dieser formalen Verifikation kann allgemein gezeigt werden, ob z. B. ein bestimmter Zustand erreichbar ist, oder ein bestimmtes Ereignis eine definierte Reaktion nach sich zieht.

Der Embedded Validator ist derzeit nur für die Toolkette SL/TL verfügbar, da der von TargetLink generierte C-Code die Grundlage für die Analyse bildet. Dennoch ist dieses Tool im Bereich des Funktionsentwurfs und nicht der Codeverifikation anzusiedeln, da vom Embedded Validator bestimmte Einstellungen vorgenommen werden, um analysierbaren C-Code durch TargetLink zu erzeugen. Es wird also ein „Golden Device“ und nicht der Code einer Steuergeräteapplikation untersucht. Die Brücke zum Testen der ECU Applikation kann aber dadurch geschlagen werden, dass der Embedded Validator Signale generiert, mit Hilfe derer die Ergebnisse einer formalen Verifikation nachsimuliert werden können. Diese Testsignale können für spätere Entwicklungsschritte weiter verwendet werden.

Das Tool kann für einfachere Tests auch während der Entwicklung verwendet werden, wird aber aufgrund seiner Komplexität sicherlich hauptsächlich im Bereich der Qualitätssicherung eingesetzt.

1.1.2 Fixed-Point Implementierung

Im nächsten Entwicklungsschritt werden dem bestehenden Funktionsentwurf Informationen über die zu verwendende Fixed-Point Skalierung hinzugefügt. Der als Floating-Point Modell erstellte Funktionsentwurf sollte zu diesem Zeitpunkt bereits validiert und verifiziert sein. Es geht in diesem Schritt deshalb lediglich darum nachzuweisen, dass die Ergebnisse der Fixed-Point Implementierung denen des Floating-Point Entwurfs hinreichend ähnlich sind. Dieser Nachweis wird durch Simulation erbracht. Die Eingangssignale der für die Floating-Point Simulation erstellten Testfälle können in dieser zweiten Entwicklungsstufe wieder verwendet werden. Referenzsignal ist jeweils das Ergebnis der Floating-Point Simulation.

Oftmals führen Schwierigkeiten bei der Umsetzung in Fixed-Point dazu, dass der Funktionsentwurf nochmals abgeändert werden muss. Dies hat zur Folge, dass der Verifikations-

⁸Hersteller: Reactive Systems

⁹„T-VEC Tester for Simulink“ von T-VEC Technologies

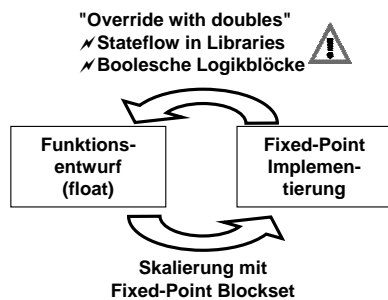


Abbildung 3: Funktionsentwurf und Fixed-Point Implementierung in der Toolkette SL/EC: Schwierigkeiten beim Zurückschalten von Entwicklungsstufe 2 auf 1

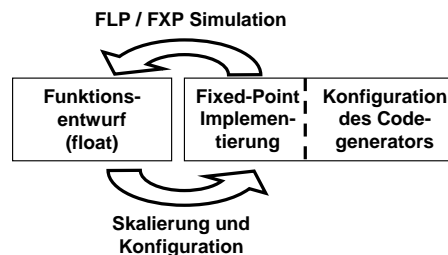


Abbildung 4: Funktionsentwurf und Fixed-Point Implementierung in der Toolkette SL/TL: Keine Trennung zwischen Entwicklungsstufe 2 und 3

und Validierungsschritt auf der Funktionsentwurfsebene nochmals durchlaufen werden muss. Wichtig ist daher auch eine einfache und schnelle Möglichkeit zum Funktionsentwurf zurückzukehren.

Beide Toolketten (SL/TL, SL/EC) bieten sowohl die Möglichkeit Fixed-Point und Floating-Point Simulation zu vergleichen, als auch von Fixed- zu Floating-Point zurückzuschalten. Dennoch gibt es zwischen den Toolketten Unterschiede hinsichtlich Benutzbarkeit und Systematik.

Fixed-/Floating-Point Simulation in SL/EC

In der Toolkette SL/EC wird das Zusatzprodukt „Simulink Fixed Point“ zur Fixed-Point Simulation verwendet. Die Simulation läuft weiterhin vollständig in Simulink ab, das Fixed-Point Verhalten wird in der Simulationsengine emuliert. Zur Visualisierung können die gewohnten Scope Blöcke u. Ä. eingesetzt werden. Der Weg von Floating-Point zum Fixed-Point Entwurf verläuft daher weitgehend reibungslos, nicht aber der Weg zurück.

Prinzipiell sieht die Toolkette SL/EC ein Zurückschalten auf Floating-Point Arithmetik und damit einen iterativen Entwicklungsprozess vor: Auch ein Modell mit Fixed-Point Informationen kann im Modus „Override with doubles“ betrieben werden. Diese Einstellung funktioniert jedoch nicht in Kombination mit manch anderem Feature. Die wichtigste Einschränkung ist hier die Inkompatibilität des Floating-Point Modus mit Stateflow Charts in Simulink Bibliotheken und bestimmten Logikblöcken (siehe Abb. 3). Ein iterativer Entwicklungsprozess ist deshalb aufgrund von Kompatibilitätsproblemen in Simulink bislang nur mit diversen Einschränkungen möglich.

Fixed-/Floating-Point Simulation in SL/TL

Die Toolkette SL/TL realisiert Fixed-Point Simulationen grundsätzlich über Codegenerierung. Der Codegenerator TL generiert C-Code, der als DLL von Simulink aufgerufen wird. Das Umschalten zwischen Fixed- und Floating-Point erfolgt, indem entweder die Simulink Implementierung (Floating-Point) oder der generierte Code (Fixed-Point) aufgerufen

werden. Der Umschaltprozess funktioniert in beide Richtungen bequem und reibungslos. Für die Fixed-Point Simulation wird jedoch immer der vollständig konfigurierte Steuergerätecode verwendet (siehe Abb. 4). In der Toolkette SL/TL ist daher eine scharfe Trennung der Entwicklungsschritte Fixed-Point Implementierung und Code Konfiguration nicht vorgesehen. Eine gewisse Trennung der beiden Entwicklungsschritte kann herbeigeführt werden, indem für den Vergleich zwischen Funktionsentwurf und Fixed-Point Implementierung C-Code der niedrigsten Optimierungsstufe erzeugt wird. Dieser Code ist wegen des hohen Ressourcenbedarfs in der Regel nicht für Steuergeräte geeignet, aber deutlich näher an der Simulink Implementierung¹⁰.

1.1.3 Code Konfiguration

Im dritten Entwicklungsschritt Code Konfiguration werden dem Fixed-Point Modell weitere Informationen hinzugefügt, die das genaue Aussehen des generierten C-Codes betreffen: Variablenamen, Variablenklassen (z. B. global, constant, etc.), Funktionsnamen, Dateipartionierung und ähnliches.

Der Codegenerator übersetzt ein Simulink Modell mit Fixed-Point und Konfigurationsinformationen nach C. Dieser Code kann als DLL für den PC kompiliert und wiederum in Simulink simuliert werden. Mit Hilfe dieser DLL kann der Steuergerätecode auf einfache Weise auf der PC Plattform verifiziert werden, was insbesondere Modultests erheblich erleichtert.

Dieses Verifikationsszenario bietet eine gute Möglichkeit, zum einen das korrekte Funktionieren des Codegenerators zu überwachen. Zum anderen werden eigene Fehler bei der Konfiguration des Modells schnell aufgedeckt. Doch auch ein automatischer Codegenerator schützt nicht vor Programmierfehlern. Es ist zum Beispiel durchaus möglich, durch die explizite Angabe von globalen Variablen eine Funktion so zu konfigurieren, dass Werte fälschlicherweise überschrieben werden.

Code Konfiguration in SL/EC

Bei der Toolkette SL/EC existiert eine vom Codegenerator vollständig unabhängige Fixed-Point Emulation in Simulink. Es ist hier also möglich, die Ergebnisse der Simulink Fixed-Point Emulation mit denen des generierten Steuergerätecodes zu vergleichen. Beide Ergebnisse müssen bit-genau übereinstimmen.

Der Codegenerator Embedded Coder hält die Option bereit, den generierten Steuergerätecode für den PC als DLL zu übersetzen. Es wird allerdings ein einzelner Block generiert, der selbst in das Modell eingebaut und angeschlossen werden muss. Eine Beschreibung einer automatisierten Vorgehensweise ist schon in [TRB99] zu finden. Eine automatisierte Lösung von Seiten des Toolherstellers existiert jedoch nach wie vor nicht.

Auch bei dieser Option zeigen sich diverse Inkompatibilitäten zu anderen Features, die nur mit einer Reihe von selbst implementierten Behelfslösungen überwunden werden können.

¹⁰Das Beispiel Embedded Validator in Abschnitt 1.1.2 zeigt, dass es durchaus möglich ist, den Codegenerator so umzukonfigurieren, dass Code für ein Golden Device erzeugt wird. Dieses Umschalten der Konfiguration müsste jedoch mit einem gewissen Aufwand selbst implementiert werden.

Das gravierendste Problem stellt sicher die Verwendung generischer C-Datentypen, z. B. „int“, im generierten Code bestimmter Hilfsfunktionen dar. Die PC Simulation kann dadurch bei Steuergeräten, die nicht mit 32 Bit arbeiten, erheblich abweichen.

Auch bei dieser Simulation des Steuergerätescodes muss daher gesagt werden, dass die an sich guten Verifikationsmöglichkeiten wegen der Kompatibilitätsprobleme nur eingeschränkt nutzbar sind.

Code Konfiguration in SL/TL

Wie bereits in Abschnitt 1.1.2 erwähnt, benutzt TargetLink grundsätzlich den generierten Steuergerätescode für die Fixed-Point Simulation. Das Verfahren für Fixed-Point Simulation und Steuergerätescodesimulation unterscheidet sich hier also nicht voneinander.

Eine vom Codegenerator unabhängige Fixed-Point Lösung existiert nicht. Das Umschalten zwischen Simulink Simulation und Codesimulation funktioniert per Knopfdruck. Eine automatisierte Lösung, die Einbau und Anschluss der DLL im Modell übernimmt, wird mit TargetLink geliefert. Seit der Version 2.0 wird zur Verifikation auch ein Code Coverage Tool bereitgestellt¹¹.

1.2 Interplattformtests

Interplattformtests dienen der Verifikation des gleichen Verhaltens der entwickelten Software auf der Entwicklungsplattform und auf dem Zielsteuergerät. Die in den vorangegangenen Abschnitten beschriebenen Tests erfolgten alle auf einem Standard-PC, typischerweise direkt in der Entwicklungsumgebung des modellbasierten Entwicklungstools. Damit basieren sämtliche Berechnungen zur Laufzeit der Tests auf der Hardwarearchitektur eines solchen Standard-PC.

Steuergeräte für automotive Anwendungen besitzen – bedingt durch Kosten-Ressourcen-Abwägungen – in der Regel eine andere Hardwareplattform. Signifikante Unterschiede zum Standard-PC finden sich in vielen Fällen im CPU-Rechenwerk (z. B. Gleit- bzw. Festkommaarithmetik, Wortbreite), beim Speicher und dem Cache oder auch im Programmdatenfluss (z. B. Pipelining). Daher benötigen diese Plattformen speziell auf ihre Architektur abgestimmte Compiler-/Linker-Werkzeuge zur Generierung des maschinenausführbaren Binärcodes.

Durch Interplattformtests lassen sich somit nicht nur Fehler aufzeigen, die aus Unterschieden in der Hardware resultieren, sondern auch Fehler deren Ursache in den Compiler-/Linker-Werkzeugen bzw. in den Codegenerierungswerkzeugen liegt.

¹¹Das Code Coverage Tool ist hier integrierter Bestandteil von TargetLink. Für den Embedded Coder wird ein Code Coverage Tool als Bestandteil des Produkts Test RealTime von der Firma Rational angeboten.

1.2.1 Compiler-/Linker

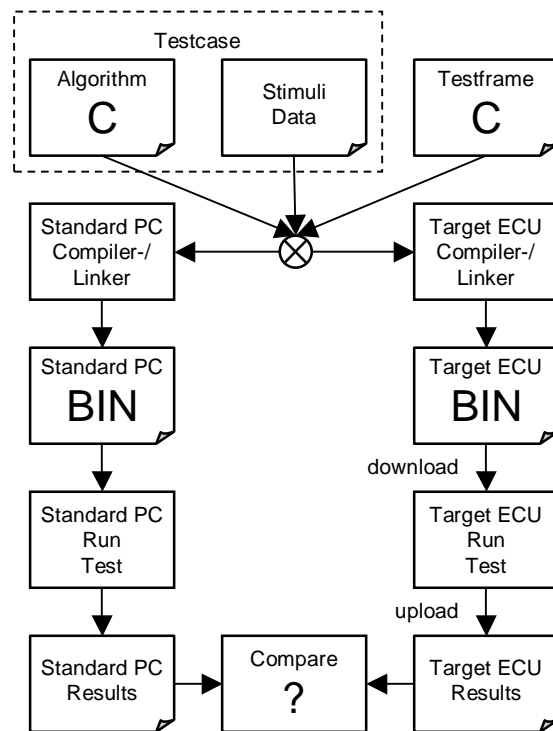


Abbildung 5: Einfacher Interplattformtest zur Compiler-/Linker-Validierung

1.2.2 Automatische Codegenerierung

Interplattformtests in Verbindung mit modellbasiertem Entwurf werfen neue Fragen auf. Die Funktionalität eines aus einem modellbasierten Entwurf automatisch generierten Codes an sich kann mit den unter Abschnitt 1.1 beschriebenen Tests bereits sehr gut und effizient geprüft werden. Für derart getestete Modelle wären im Idealfall einer fehlerfreien Kombination von Compiler-/Linker-Werkzeugen und Zielplattform keine weiteren Interplattformtests mehr erforderlich¹³.

Autocodengeneratoren unterscheiden sich jedoch bezüglich der generierten Programmkonstrukte vielfältig von den bei manueller Programmierung typischerweise gewählten Konstrukten. Ein automatischer Codegenerator verwendet zum Teil hocheffiziente, aber für einen Menschen schwer verständliche Konstrukte. Außerdem können durch den Automatismus sehr leicht tief verschachtelte oder kompliziert verkettete Konstrukte entstehen, die für einen Menschen kaum nachvollziehbar sind. Der Mensch hingegen wählt bei der Pro-

¹²zunächst nur im manuell programmierten C-Code

¹³Integrationstests im Gesamtsystem gelten in diesem Kontext nicht als Interplattformtests.

Bei Continental Teves werden mittels einfacher Interplattformtests die steuengerätespezifischen Compiler-/Linker-Werkzeuge einer Validierungsprüfung unterzogen. Dazu hat der TÜV Rheinland die bei Teves verwendeten Programmkonstrukte¹² analysiert und zu einer für die Validierung maßgeblichen Sammlung von Testfällen zusammengestellt. Die Struktur dieses einfachen Interplattformtests zeigt die Abb. 5. Wesentliches Merkmal ist, dass im Test beide Plattformen exakt den gleichen C-Code verwenden.

grammierung die Konstrukte entsprechend seiner Erfahrungen und Kenntnisse unter dem Einfluss seiner natürlichen Bequemlichkeit und mit Blick auf eine subjektiv empfundene Übersichtlichkeit.

Insofern können Interplattformtests, die die automatische Codegenerierung mit einbeziehen, zur Detektion neuer Fehler dienen. Die Abb. 6 zeigt die Struktur eines derartig erweiterten Interplattformtests.

Theoretisch steigt zwar bei dieser erweiterten Kette die Wahrscheinlichkeit einer internen Kompensation von Fehlern, so dass sie im Test nicht mehr nach außen sichtbar sind. Praktisch ist dieser Fall wegen der geringen Fehlerwahrscheinlichkeit in den einzelnen Komponenten aber vernachlässigbar. Die Detektionswahrscheinlichkeit bei Vorliegen eines oder mehrerer Fehler ist so bedeutend größer, dass der praktische Nutzen zur Aufspürung von Fehlern in der Kette überwiegt.

Trotz der Vorteile dieses erweiterten Interplattformtests muss an dieser Stelle noch mal klar herausgestellt werden: Der Aufwand und Zeitbedarf für solch einen Test ist selbst bei hohem Automatisierungsgrad (z. B. durch skriptgesteuerten Ablauf) und einfach verfügbarer Zielhardwareplattform (z. B. Evaluation Boards für PC-Anschluss) deutlich größer als bei den reinen Softwaretests nach Abschnitt 1.1. Daher sollten diese Interplattformtests soweit möglich reduziert werden.

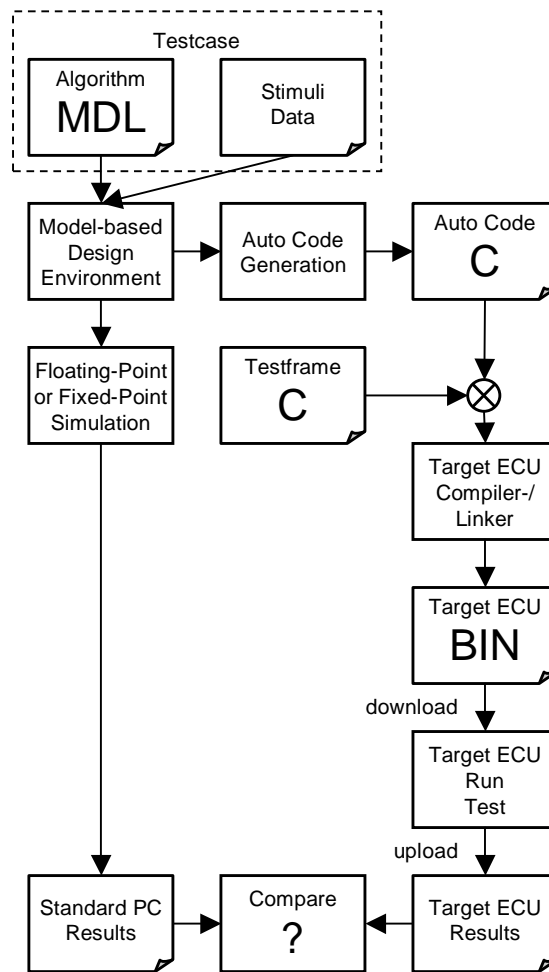


Abbildung 6: Interplattformtest inklusive automatischer Codegenerierung

Aus den Erfahrungen bei Continental Teves resultierte schließlich dieses Konzept: Die Funktionalität von Modellen wird nur mit Softwaretests nach Abschnitt 1.1 geprüft. Interplattformtests nach Abb. 6 dienen nur zur grundsätzlichen Validierung der gesamten Ket-

te Automatische Codegenerierung, Compiler-/Linker-Werkzeuge und Zielhardwareplattform.

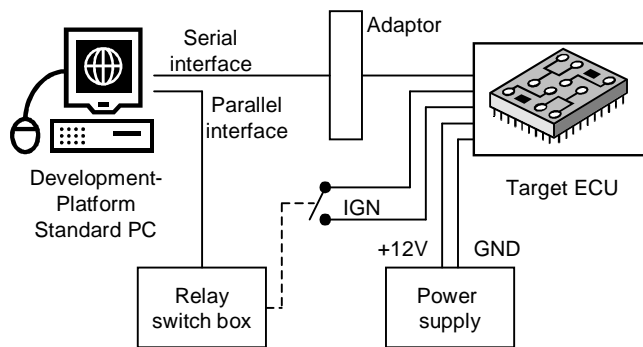


Abbildung 7: AVS Hardwarestruktur

Dazu wurde als Kooperation des TÜV Rheinland, Ford und Continental Teves die Autocoder Validation Suite (AVS) aufgebaut. Die AVS besteht aus einem Standard-PC, einem separaten Interface und der Spannungsversorgung für das Zielsteuergerät und dem Zielsteuergerät selbst. Diese Struktur ist in Abb. 7 skizziert.

Die AVS kann eine ganze Sammlung von Testfällen automatisch prüfen. Die einzelnen Testfälle bestehen aus Testmodell, Stimulidaten und ggf. auch erlaubter Abweichung zwischen Referenzergebnis (berechnet auf dem PC)- und Testergebnis (berechnet auf der Zielhardware). Die Testfallsammlung wurde von den Kooperationspartnern gemeinsam aufgebaut. Bei den Testmodellen handelt es sich nicht um einfache Einzelblockmodelle sondern um typische Blockkombinationen, die die Kooperationspartner aus einer Analyse der eigenen modellbasierten Entwürfe gewonnen haben.

Auf diese Weise entsteht allerdings keine vollständige Testfallsammlung. Dieser heuristische Ansatz versucht eine praktisch relevante Untermenge aus der Vielfalt der theoretisch möglichen Testfälle zu bilden. Mittels weiterer Analysen der Testfallsammlung in Verbindung mit einem Betriebsbewährtheitsnachweis für die Kette automatische Codegenerierung, Compiler-/Linker-Werkzeug, und Zielplattform soll die AVS in Zukunft zu einer Einsatzreifenzertifizierung dieser Kette dienen. Mittlerweile haben mehrere Automobilhersteller ihr Interesse an der Mitarbeit in der bisherigen Kooperation bekundet.

2 Entwicklungsprozesse und Werkzeuge

Dieser Abschnitt befasst sich mit dem Einsatz modellbasierter Entwicklungswerkzeuge in einem Entwicklungsprozess für Seriensoftware. Charakteristisch für einen solchen Serienentwicklungsprozess ist, dass hier die MBE nur ein Teil einer Kette von Werkzeugen ist und dass eine große Anzahl von Entwicklern/-innen gleichzeitig an einem Softwareprodukt arbeitet.

Beiden hier behandelten Toolketten ist gemeinsam, dass das Simulationsprogramm Simulink als Entwicklungsfondend verwendet wird. Die meisten Prozessintegrationsprobleme lassen sich an diesem gemeinsamen Frontend festmachen und stehen nicht in Zusammenhang mit dem jeweiligen Codegenerator.

Die Schwierigkeiten, die bei der Integration in den Softwareentwicklungsprozess auftreten, sind bereits in [KaC04] beschrieben. Zentrale Aspekte sind in den beiden nächsten Abschnitten kurz skizziert.

2.1 Konfigurationsmanagement

Ein wesentliches Problem besteht darin, dass bei einem Entwicklungsprozess mit mehreren Personen zwingend eine Aufteilung eines Modells in mehrere Dateien notwendig ist. Im industriellen Einsatz werden diese Dateien zur Prozessabsicherung mittels eines Konfigurationsmanagement(KM) Tools verwaltet. Dadurch lässt sich eine detaillierte, aufgaben- und verantwortlichkeitsorientierte Zuordnung von Zugriffsrechten auf einzelne Dateien realisieren.

Die ursprünglich als Simulationswerkzeug und nicht als integrierte Entwicklungsumgebung entworfene Applikation Simulink zeigt auch in der aktuellen Version¹⁴ noch erhebliche Schwächen bei der Behandlung solcher auf mehrere Dateien verteilter Modelle¹⁵:

- File caching Mechanismen stellen die Reproduzierbarkeit von Softwareständen in Frage.
- Es gibt Operationen, die erzwingen, dass eine Person auf allen Dateien eines Projektes gleichzeitig Schreibrechte benötigt.

Firmen, die ihre Softwareentwicklung in großem Umfang auf Simulink umstellen wollen, müssen diese Probleme überwinden. Dort wo Simulink als Entwicklungstool in Arbeitsgruppen im Einsatz ist, haben die betroffenen Firmen deshalb oftmals mit hohem Aufwand selbst Lösungen für die Partitionierung und Aggregation von Modellen entwickelt.

Das direkte Zusammenspiel mit externen KM Tools funktioniert hingegen problemlos. Modellquellendokumente können praktisch alle derartigen Tools verwalten. Für die Interaktion aus Simulink heraus besitzt MATLAB eine Schnittstelle, die für die gängigsten KM Tools bereits vorkonfiguriert ist. Über diese Schnittstelle sind somit auch KM Informationen eines Modells (z. B. Revisionsnummer, Autor, etc.) direkt in der Entwicklungsumgebung verfügbar.

2.2 Toolüberwachung

Die Toolüberwachung ist als ein Element zur Absicherung des Softwareentwicklungsprozesses anzusehen, das vor allem auf die Reproduzierbarkeit abzielt. Die im Entwicklungsprozess verwendeten Werkzeuge sollen bei gleichen Modellquellen an jedem Arbeitsplatz

¹⁴Einige der in [KaC04] beschriebenen Probleme mit Simulink Bibliotheken wurden durch Einführung des Features „Model Referencing“ insofern entschärft, als der Einsatz von Bibliotheken nunmehr oftmals durch Modellreferenzierung ersetzt werden kann. Auch Model Referencing bietet jedoch keine vollständige Lösung der Multi-User Problematik.

¹⁵Auch die in 1.1.2 erwähnten Inkompatibilitäten stehen in Zusammenhang mit Modellpartitionierung.

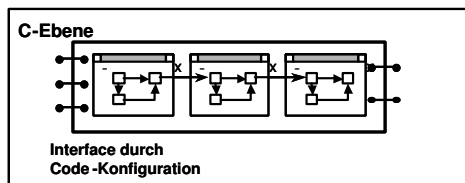


Abbildung 8: Modulintegration in Simulink

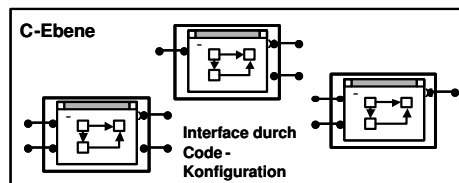


Abbildung 9: Systemintegration in C

stets die gleiche Software generieren. Da hier neben der eigentlichen Toolinstallation in der korrekten Version auch die Konfiguration der Werkzeuge großen Einfluss hat, ist eine Toolüberwachung erforderlich.

In diesem Zusammenhang weist Simulink bislang einen strukturellen Nachteil auf, der besonders bei Projekten, die aus vielen Dateien bestehen, zum Tragen kommt. In Simulink gibt es keinen eigenen Projektarbeitsbereich mit einer eigenen Projektbeschreibungsdatei. Der Dateizugriff wird ausschließlich über einen Suchpfad geregelt. Dieser Suchpfad lässt sich – beabsichtigt oder unbeabsichtigt – auf vielfältige Weise einfach manipulieren. Daher lässt sich nur schwer garantieren, dass Modelle an unterschiedlichen Arbeitsplätzen immer den gleichen Steuergerätee-Code erzeugen. Dieses Problem kann zwar mittels selbst erstellter, spezieller Konfigurationsskripte vermieden werden. Hier gilt es aber den Aufwand dazu abzuwägen. Meistens wird das Problem anders gelöst. Es wird ein separater Rechner bereit gestellt, der nur zur Codegenerierung dient. Auf diesem Rechner muss dann allerdings auch die gesamte Freigabe-, Verifikations- und Validierungsprozedur durchlaufen werden.

2.3 Vorgehensweisen

Prinzipiell gibt es zwei Grundtypen von Einsatzszenarien für Simulink in großen Teams:

- Multi-User Entwicklung in Simulink mit Modulintegration in Simulink (Abb. 8)
- Single-User Entwicklung in Simulink mit Systemintegration in C (Abb. 9)

Diese beiden Varianten werden in den folgenden Unterabschnitten kurz erörtert. Die Betrachtung erfolgt unter der Prämisse, dass ein Projekt mit einem großen Bearbeitungsteam auch einen Anteil bereits bestehender C-Quellen besitzt.¹⁶

2.3.1 Modulintegration in Simulink

Wesentlicher Vorteil der Modulintegration in Simulink ist die gemeinsame Simulierbarkeit aller in Simulink entwickelten Systemteile.

¹⁶Ein rein modellbasiertes Projekt stellt in der industriellen Praxis eher die Ausnahme dar und ist hier nur als Vereinfachung anzusehen.

- Damit werden auch modulübergreifende Tests in Simulink ermöglicht,
- es findet eine Konsistenzprüfung innerhalb des Modells (z. B. Fixed-Point Datentypen) statt und
- es muss lediglich eine Schnittstelle zu bestehendem C-Code konfiguriert werden.

Der entscheidende Nachteil ist im mangelhaften Projektmanagement unter Simulink zu sehen.

2.3.2 Systemintegration in C

Eine Alternative bietet die zweite Vorgehensweise, bei der die Integration in C stattfindet.

Es werden Module entwickelt, die klein genug sind, um von einer Person bearbeitet zu werden. Eine Integration dieser Module findet erst auf C-Ebene, d. h. durch Linken des aus allen Modulen generierten und kompilierten Codes statt.

Dieser Gesamtcode kann als DLL übersetzt wiederum in Simulink ausgeführt werden.

Die Vorteile dieses Vorgehens liegen darin, dass

- das gesamte Systemdesign in C stattfindet,
- die Multi-User Problematik in Simulink umgangen wird und
- in einem erprobten Softwareentwicklungsprozess nur geringe Anpassungen notwendig sind.

Nachteile sind

- der höhere Aufwand für die Konfiguration mehrerer Schnittstellen und
- die eingeschränkten Simulationsmöglichkeiten.

2.3.3 Aktuelles Vorgehen

Aufgrund der Vielzahl von Problemen, die bei der Entwicklung mit großen, partitionierten Modellen auftreten, setzt Continental Teves derzeit für die Serienentwicklung das zweite Vorgehensmodell (Integration in C) ein.

Langfristig ist es jedoch Ziel, zu einer Integration in Simulink und damit zu einer Gesamtsystemsimulation zu gelangen.

Wir verfolgen und prüfen deshalb den aktuellen Stand der Multi-User-Fähigkeit von MBE Tools in Pilotprojekten.

3 Beurteilung und Ausblick

Die modellbasierte Entwicklung bietet eine Vielzahl von Möglichkeiten schon während der Entwicklung Softwaretests durchzuführen. Es wird damit eine große Testtiefe bei geringem Testaufwand erreicht. Zudem helfen diese Tests, Fehler bereits frühzeitig in den entsprechenden Stufen des Entwicklungsprozesses zu finden.

Solche reinen Softwaretests sind einfacher, schneller und letztlich billiger als Hardwaretests und stellen einen wesentlichen Vorteil der MBE dar.

In Bezug auf die Handhabbarkeit und Zuverlässigkeit des Softwareentwicklungsprozesses bleibt die MBE aber derzeit noch deutlich unter dem Niveau, das mit traditioneller Programmierung erreicht wird.

Sofern ein breiter Einsatz der MBE angestrebt wird, sehen wir deshalb gerade an diesem Punkt die Herausforderungen für die Entwicklung zukünftiger Toolversionen.

Literatur

- [Ald02] Aldrich, W. (2002): Using model coverage analysis to improve the controls development process. In: American Institute of Aeronautics and Astronautics: AIAA Modeling and Simulation Technologies Conference and Exhibit, Aug. 5-8, 2002, Monterey, CA, USA.
- [BBS04] Bienmüller, T.; Brockmeyer, U.; Sandmann, G. (2004): Automatic Validation of Simulink/Stateflow Models. In: Proceedings of the International Automotive Conference 2004, 15-16 Juni 2004, Stuttgart, Germany.
- [KaC04] Kalix, E.; Creutzburg, U. (2004): Processes and Tool Requirements in Professional Software Development, In: Proceedings of the 5th Eurosim Congress on Modelling and Simulation (EuroSim'04), 6-10 September 2004, ESIEE-Paris FRANCE.
- [LBE04] Lamberg, K.; Beine, M.; Eschmann, M.; Otterbach, R.; Conrad, M.; Fey, I. (2004): Model-Based Testing Of Embedded Automotive Software Using MTest. In: SAE 2004, March 8-11, 2004, Detroit, Michigan, USA, Technical Paper 2004-01-1593.
- [Ran03] Ranville, S. (2003): MATLAB "add-on" tools for state of the art embedded software development. In: AIAA/IEEE Digital Avionics Systems Conference – Proceedings, 10.A.2/1-10.A.2/13, 12-16 Oct. 2003, Indianapolis, IN, USA.
- [Rau02] Rau, A. (2002): Verwendung von Zusicherungen in einem modellbasierten Entwicklungsprozess. In: IT+TI Informationstechnik und Technische Informatik, 03/2002: 137-144.
- [TRB99] Toeppe, S.; Ranville, S.; Bostic, D.; Wang, Y. (1999): Practical validation of model based code generation for automotive applications. In: Gateway to the New Millennium. 18th Digital Avionics Systems Conference. Proceedings, 10.A.3/14 pp., 24-29 Oct. 1999, St Louis, MO, USA.