TREND GUIDE

Embedded Automotive

MICRO CONSULT

No.1 in Developer Training

Embedded Automotive

Vorwort **04**

Trend Spots **06**

7-Ebenen Modell – Qualität von A bis Z **08**

Requirements Management – Sicherer Projektstart **13**

Architekturmodellierung – Königsweg UML 2.0 **18**

Algorithmenmodellierung – Schlaue Tools **24**

Implementierung – Vom Makro- zum Mikrokosmos **26**

Netzwerktechnik – Schlüssel zur Systemintegration **30**

Chiparchitektur – Starke Peripherien **36**

Qualität, Test und Integration – Stunde der Wahrheit **38**

Fazit – Vorteile auf allen Ebenen **42**

Info Pool **44**

Kurse zum Thema **46**

Impressum **52**

Ihre Partner für Embedded Automotive! **50**

Klaus-Peter Rosenthal
Produktmanager
MicroConsult

Liebe Leser,

immer schneller dreht sich das Rad der Automobilentwicklung. Im Durchschnitt ist heute bereits nach weniger als sieben Jahren ein Modellwechsel fällig. Gleichzeitig steigt die Komplexität der Systeme.

Wer die Module seiner Zulieferer erst im Automobil testet, nimmt deshalb langfristig herbe Qualitätseinbussen in Kauf. Für ein besseres Zusammenspiel von Hersteller und Dienstleister benötigen Projektteams allerdings eine leistungsfähige Gesamtarchitektur und durchgängige Toolketten. Dafür fehlt es vielfach noch an tragfähigen Standards, doch neue Technologien wie X-by-Wire und der wachsende Kostendruck treiben die Entwicklung in diesem Bereich spürbar voran.
Einsparungen lassen sich aber auch heute schon auf allen Ebenen der Produktentwicklung realisieren, wenn diese wirksam ineinander greifen – vom Requirements Management bis zum Test.

Dieser Trend Guide möchte Ihnen praxisnah vermitteln, welche Schritte sich bis zum fertigen Automobil konkret optimieren lassen.

Dr.-Ing. Gerd Teepe
Leiter der Vorentwicklung
Automobilelektronik
Motorola

Liebe Leser,

das Wachstum in der Automobilelektronik treibt heute die gesamte Halbleiterindustrie an. Dabei werden weitere Qualitätsverbesserungen dringend benötigt, um die Defektschwelle von 1ppm zu unterschreiten. Die neuen modellbasierten Entwicklungsmethoden versprechen hier signifikante Verbesserungen, die über den Fortschritt in der Kontrolle der Produktionsprozesse hinaus gehen. Dazu gehören der vermehrte Einsatz formaler Verifikationsverfahren sowie objektorientierte Softwareentwicklungen, bei denen UML eine immer stärkere Rolle einnimmt. Die Integration der Komponenten im Auto über komplexe Netzwerke ist in Zukunft nur mit den neuen Softwarekonzepten zur hardware-abstrakten Middleware zu beherrschen. Durch die Wiederverwendbarkeit von Soft- und Hardware-Komponenten werden dann auch schnellere Entwicklungsergebnisse zu geringeren Kosten erzielt.

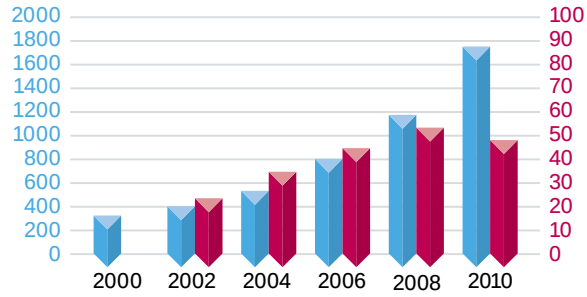
Ich bin mir sicher, dass die hervorragende Kompetenz von MicroConsult diesen expandierenden Markt maßgeblich mitgestalten wird.

Trend Spots

Für Sie kurz zusammengefasst...

- Innovationen im Automobil sind zu 90 Prozent durch Elektronik getrieben. Der Wertschöpfungsanteil der Luxusklasse beträgt in diesem Bereich bis zu 35 Prozent.
- Der Lebenszyklus eines Halbleiters ist im Durchschnitt ein Jahr, ein Fahrzeugmodell wird erst nach sieben Jahren durch ein neues ersetzt. Synchronisationsprobleme im Embedded Automotive-Bereich sind deshalb vorprogrammiert.
- 90 Prozent aller Unfälle lassen sich vermeiden, wenn ein unaufmerksamer Fahrer eine Sekunde mehr Reaktionszeit hat oder gar nicht erst abgelenkt wird.
  Quelle: National Highway Traffic Safety Administration
- Der Halbleitermarkt ist mit rund 200 Herstellern äußerst fragmentiert. Ungefähr ein Viertel davon sind mittlerweile im Automotive-Sektor mit einer unüberschaubaren Produktpalette präsent.
- Die Qualität deutscher Automobile gilt immer noch als hoch, doch unflexible Entwicklungs- und Produktionsprozesse erschweren Innovationen.
- Die Industrie kann heute adäquate Standardisierungsprozesse noch nicht meistern. Dies hemmt auch die Entwicklung durchgängiger Toolketten.
- Der weltweite Markt für X-by-Wire Subsysteme wird weltweit bis 2010 auf 22 Mrd. US-Dollar anwachsen.
  Quelle: Allied Business Intelligence
- Europäische X-by-Wire-Hersteller für Bremsen, Steuern und Beschleunigen setzten 2002 324 Mio. Euro um. 2010 sollen diese Bereiche einen Markt von 1,75 Mrd. Euro erobern.

Europamarkt für X-by-Wire Systeme:
Brake-by-Wire, Steer-by-Wire, Power-by-Wire



- Umsätze in Millionen Euro
- Umsatzwachstumsrate in %

Quelle: Frost & Sullivan 2003

7-Ebenen-Modell

Qualität von A bis Z

Aus dem Blickwinkel des Automobilherstellers erscheint das Prinzip einfach: Er gibt ein Embedded Projekt bei einem Zulieferer in Auftrag und dessen Ergebnis muss in sein späteres Gesamtsystem passen. Doch wie kann er direkt auf die geforderte Qualität des Endprodukts Einfluss nehmen?

Kein leichtes Unterfangen, denn während der langjährigen Modellentwicklung unterliegen die internen Prozesse einem ständigen Wandel und damit auch die Systemspezifikation. Sie kann zudem nur bestimmte, eng umgrenzte Bereiche des Fahrzeugs erfassen. Eine übergreifende Sicht im Sinne eine Spezifikation des gesamten Automobils und einer Komplettarchitektur der Software sind heute noch Zukunftsmusik.

Aufgrund der wachsenden Komplexität – gerade im Elektronikbereich – wird der Blick aus der Vogelperspektive jedoch langfristig unumgänglich sein. Beim mechanischen Part des Automobils ist dies Standard. Doch auch vom heutigen Standpunkt aus lassen sich bereits viele Schritte bis zum fertigen Projekt optimieren, wenn die Automobilhersteller frühzeitig mit den richtigen Methoden in den Requirements Prozess einsteigen. Alleine die Weitergabe der Anforderungen an den Zulieferer lässt beispielsweise häufig zu wünschen übrig. Kurze Telefonate bilden zwar den vermeintlich schnellsten Kommunikationsweg, aber auch gleichzeitig den schwächsten. Das Einhalten der Requirements ist in diesem Fall schwer zu überprüfen. Für beide Seiten nachvollziehbarer ist sicherlich eine schriftliche Spezifikation, doch auch sie kennt Grenzen. Häufig beschränkt sie sich auf funktionale Anforderungen, wobei die Qualität des Endprodukts einen ausgeprägt nichtfunktionalen Charakter besitzt, zum Beispiel in Bezug auf Zeit, Portabilität, Konnektivität und Sicherheit. Auch gilt es festzulegen, was das System nicht können darf. Ein Beispiel hierfür ist die meist unnötige Datumsfunktionalität, die den Programmierern beim Jahrtausendwechsel schwer zu schaffen machte. Aufgrund der aktuellen Relevanz von „Embedded Quality“ hat MicroConsult dem Thema bereits einen eigenen Trend Guide gewidmet, der kostenfrei erhältlich ist (mehr dazu auf Seite 12). Die geforderte Qualität lässt sich heute nur erzielen, wenn ein Projekt auf verschiedenen Ebenen vorangetrieben wird, die wirksam ineinander greifen.

Dieser Ansatz ist jedoch nicht allen Bereichen optimal realisierbar. So fehlt es immer noch an durchgängigen Toolketten und etablierten Standards. Folgende Ebenen in der Soft- und Hardware-Entwicklung werden näher beleuchtet:

Ebenen der Entwicklung

- **Requirements Management** (Gewichtung von Software- und Hardwareanteilen)
- **Architekturmodellierung** (softwarespezifisch)
- **Algorithmenmodellierung** (softwarespezifisch)
- **Implementierung** (softwarespezifisch)
- **Kommunikation** (Architektur bestimmt die Grenze zwischen Software und Hardware)
- **Design der Steuergeräte** (hardwarespezifisch)
- **Chiparchitektur** (momentan hardwarespezifisch, aber Trend zur Software zeichnet sich mit dem Hardware/Software Co-Design ab)
- **Test und Integration**

7 Ebenen der Software-Entwicklung

Requirements Management	**Zeitdiskret** Statemate UML VeriState ModelCertifier	**Zeitkontinuierlich** Matlab/Simulink Matrixx ASCET-SD	**Requirements Tracing** DOORS RequisitePro	
Architekturmodellierung	Statemate ROOM UML	**Co-Simulation** Excite		
Algorithmenmodellierung	**Zeitkontinuierlich** Matlab/Simulink ASCET-SD Saber	**Co-Simulation** Excite		
Implementierung	**Sprachen** Assembler C, C++, Java	**Codegeneratoren** UML-Basierte Werkzeuge TargetLink Rhapsody for Micro-C (Statemate)		
Design der Steuergeräte	**Co-Simulation** Saber	**Feldbusse** CAN, TTCAN, MOST Flexray, LIN, IEEE 1394		
Chiparchitektur	**Co-Simulation** Saber	**Chipentwicklung**		
Qualität; Test und Integration	**Testsuiten** Realtime Test Tessy Quast	**Testvektoren** VeriState ModelCertifier		

Die Prozesse werden in sieben Ebenen dargestellt und führen zu einem Steuergerät. Jeder von ihnen erfordert einen eigenen Blickwinkel und seine abstrakte Sicht auf das System. Der Automobilhersteller deckt oft das Requirements Management, die Algorithmenmodellierung und Teile des Tests ab. Die anderen Prozesse sind Sache des Zulieferers. *Quelle: MicroConsult 2003*

7-Ebenen-Modell

Fünf Thesen zur Embedded Qualität

01 Qualitätsziele sollten auf Basis der Firmenstrategie festgelegt werden und sind damit durch die Produkte und Märkte bestimmt, die der Automobilhersteller bedienen will.

02 Es muss eindeutig unterschieden werden zwischen
- der Qualität des Prozesses: Wie gut sind die Entwicklungsabläufe, mit denen Embedded Systeme erstellt werden?
- der Qualität des Produkts als Resultat der Prozesse: Sie wird vor allem von der Eignung für den späteren Einsatz bestimmt.

03 Eine gute interne Qualität bildet die Voraussetzung für eine gelungene externe Qualität (Axiom des Software-Engineering von Norman E. Fenton).

04 Der Schlüssel für den kosteneffizienten Projekterfolg liegt in der Gewichtung und Beschränkung von internen und externen Qualitätsmerkmalen (z.B. Funktionalität oder Zuverlässigkeit)

05 Die Spezifikation ist die Hauptquelle aller Qualitätsprobleme. Die Fehler werden meist zu spät erkannt und sind dann nur mit immensem Kosten- sowie Zeitaufwand zu beseitigen.

Mehr zu diesem Thema im bereits erschienenen Trend Guide "Embedded Quality". Interessenten können diesen kostenlos anfordern. Einfach ein E-Mail schicken an: **trendguide@microconsult.de**

Sicherer Projektstart

Den Einstieg in den Anforderungsprozess schaffen die Automobilhersteller am besten gemeinsam mit Dienstleistern durch Requirements Capturing. Im wesentlichen steht dabei die Kapselung des auszulagernden Teilprodukts im Vordergrund, denn Wechselwirkungen mit anderen Bereichen sollen weitgehend ausgeschlossen werden.

Die Methoden und Quellen für Requirements Capturing sind vielfältig:

- **Interviews:** Diskussionen mit den Projektbeteiligten, Marktexperten und Management.
- **Studien, Fragebögen und Data Mining:** Damit lassen sich gemeinsame und individuelle Funktionalitäten eruieren, beispielsweise Formate für die Dateneingabe.
- **Soft System Methodology (SSM):** Dieses Verfahren verbindet Requirements Engineering mit den Unternehmensstrategien. Statt technologischer Probleme stehen hier die organisatorische Struktur und der Kontext, in dem das Produkt entwickelt wird, im Vordergrund.
- **Cooperative Requirements Capture (CRC):** Diese Methode unterstützt die Entwicklung eines generischen Produkts. Eine CRC-Sitzung bezieht nicht nur Kunden und Entwickler ein, sondern auch andere Beteiligte. Sie dient dem Identifizieren von Problemen, der Teamauswahl sowie der Selektion von Nutzern und Anwenderumgebungen.
- **Marktanalysen:** Sie sind ein guter Ausgangspunkt für Einzelprodukte und Produktlinien. Marktanalysen beleuchten den Mitbewerb sowie heutige und zukünfti-

ge Trends. Damit bekommen Unternehmen eine Vorstellung, wie sich ihr Umfeld in den nächsten Jahren entwickeln wird.

- **Designer as Apprentice:** Diese Technik kommt dann zum Einsatz, wenn das interne Know-how in manchen Bereichen fehlt. So können Zulieferer als „Lehrlinge“ auf Zeit beim Automobilhersteller das nötige Projektwissen erwerben.

Generell unterteilen sich die aus dem Capturing resultierenden Requirements beim Automobilhersteller in drei Bereiche.

- **Projekt- und domänenspezifische Anforderungen:** Sie kommen aus den Fachbereichen, sind dem Zulieferer aber nicht unbedingt bis ins Detail bekannt.
- **Standardbasierte Requirements:** Sie bilden die Schwachstelle im Automotive-Bereich, da er sich aus seiner Historie heraus wenig Standards auferlegt hat. Andere Branchen wie die Eisenbahn- und Medizintechnik sind hier aufgrund der geforderten Zuverlässigkeit erheblich fortschrittlicher. Eine Wende zeichnet sich allerdings durch den Vormarsch von X-by-Wire ab. Die Übernahme von Standards dieser neuen Technologien aus dem Flugzeugbau bedarf aber einer differenzierten Betrachtung. Nicht zuletzt sind die Sicherheitsanforderungen an Automobil-Kontrollsysteme nicht die selben wie für Airliner und in einigen Bereichen aufgrund des komplexeren Verkehrsgeschehens sogar höher.
- **Auf Modellen basierende Requirements, wie sie beispielsweise Statemate liefert.**

Wer mit der Anforderungsspezifikation für eine weitreichende Kapselung der Teilprodukte sorgt, kann beispielsweise ausschließen, dass nach der Implementierung zu viele Geräte am selben CAN-Bus hängen und diesen partiell überlasten.

Mächtige Modelle

Die Kapselung schafft dabei eine klare Interface-Funktion, mit der sich die Busauslastung besser voraussagen lässt. Ein besonderes Augenmerk liegt auf Telegrammen und physikalischen Steckern.

Speziell die auf Modellen basierenden Requirements eröffnen der Kapselung neue Horizonte. Nur mit modellbasierten Ansätzen lässt sich nämlich aus der Gesamtarchitektur Simulationscode ableiten. Damit werden die mehr oder weniger strukturierten Informationen des Requirements Capturing in eine logisch konsistente, ausführbare Form übergeleitet. Wer nicht mit Modellen arbeitet, trägt immer das Risiko, dass seine Spezifikation Widersprüche beinhaltet. Die modellbasierte Spezifikation schafft dagegen eine durchgängige Sicht auf ein Steuergerät oder ein Teilsystem und stellt die Requirements Analyse auf den Prüfstand.

Den Anforderungen auf der Spur

Ein weiterer wesentlicher Schritt im Requirements Management ist das Tracing. Entsprechende Tools wie DOORS von Telelogic oder RequisitePro von Rational erfassen, welche Anforderungen zu verfolgen, schon bearbeitet oder erledigt sind. Künftig können diese

Werkzeuge auch Tests integrieren, die direkt aus den Requirements heraus verwaltet werden. Damit lässt sich auf Knopfdruck ablesen, welche Anforderungen bereits durch Testing abgedeckt sind.

Grundsätzlich unterscheidet man zeitkontinuierliche und zeitdiskrete Requirements Modelle. Die Automobilindustrie setzt aufgrund der Problemstellung auf zeitkontinuierliche Systeme, um mathematische Modelle für die Steuerung von Regler, Motortakt, Soll- und Umgebungswerten zu bilden. Sie werden durch Tools spezifiziert wie Matlab/Simulink von Mathworks, Matrixx von National Instruments, ASCET-SD von ETAS oder Saber von Synopsys.

Formale Verifikation

Wer die Systemsicherheit zustandsgetriebener Applikationen signifikant erhöhen will, ist mit der formalen Verifikation gut bedient. Sie wird seit 15 Jahren entwickelt und hat nun endlich eine hohe Marktreife erlangt. Dabei ist der Begriff „Verifikation“ missverständlich, denn das System und seine Eigenschaften werden letztendlich validiert. Dies war bis dato nur mit Reviews möglich. Auch Design- und Modellierungstools wie Statemate von I-Logix bieten mittlerweile Add-Ons für die formale Verifikation.

Ausschließlich für das Prüfen von zeitdiskreten Modellen auf Basis von Statemate-Modellen und zukünftig Stateflow von Mathworks dienen Tools wie ModelChecker und ModelCertifier von OSC oder VeriState von Motorola. Diese Produkte haben ihre Wurzeln im Chipdesign. In Statemate formt man beispielsweise zuerst Module nach dem Verfahren der funktionalen Dekomposition und bildet das Ergebnis in Activities ab. Diesen können State Charts zugeordnet werden, die wiederum die eigentliche Funktion des Systems beschreiben. Auf Basis dieses Prinzips können Eigenschaften als Requirements in eine logische, konsistente Form gebracht werden.

Bildet das Modell beispielsweise eine Zentralverriegelung mit 15 bis 20 State Charts ab, lassen sich diese simulieren. Damit lässt sich feststellen, ob alle Bestandteile richtig berücksichtigt wurden. Wird ein stehendes Automobil in einen Auffahrunfall verwikkelt, könnte das für den Insassen fatale Folgen haben, wenn er die Türen nicht öffnen kann, weil er zuvor die Zentralverriegelung betätigt hat. Eine zu prüfende Modelleigenschaft könnte deshalb sein: Öffnet sich die Tür, wenn der Crash Sensor einen Unfall registriert? Diese Eigenschaft wird mathematisch nach der formalen Verifikation gegen das Modell geprüft. Das in Bezug auf die Systemrelevanz zu interpretierende Ergebnis zeigt Lücken in der Spezifikation auf. Hersteller garantieren heute, dass mit ihren Tools hundert Prozent aller Fälle berücksichtigt sind, wenn die Prüfung durchlaufen ist. Die Zuverlässigkeit dieses Ansatzes ist durch das Chipdesign nachgewiesen, denn die Markteinführung eines neuen Chips wäre heute ohne formale Verifikation undenkbar. ◆

Architekturmodellierung

Köngsweg UML 2.0

Die Architekturmodellierung mit der Unified Modeling Language (UML) dient dazu, Softwaresysteme grafisch und ohne Code zu entwickeln. Gerade in der Automobilindustrie kommen die Qualitäten der Notation voll zum Tragen, da Hersteller und Zulieferer die Architektur gleichermaßen verstehen müssen. Auch kann die UML den Makrokosmos des gesamten Fahrzeugs auf den Mikrokosmos des Steuergeräts abbilden.

Gerade im Bereich der Kommunikation resultieren die größten Integrationsprobleme aus nicht geklärten Schnittstellen. So müssen sich die Interfaces im Steuergerät auf diejenigen der Gesamt-Busstruktur beziehen, damit die Konsistenz des Modells gewährleistet ist. Version 2.0 der UML könnte hier den Weg zur formalen Verifikation ebnen. Bereits vor einigen Jahren hat die Object Management Group (OMG) wichtige Konzepte für Analyse und Design objektorientierter Anwendungen unter dem Dach der UML zusammengeführt. Mit der Mitte Juni 2003freigegebenen Version 2.0 ist jetzt so manche Kinderkrankheit überwunden.

Zwar gab der Standard in Version 1.x bereits eine einheitliche Notation vor, doch die Symbole und Diagramme führten häufig zu Missverständnissen und deckten die branchenspezifischen Anforderungen nicht ausreichend ab. Alle führenden Hersteller erarbeiten deshalb UML-Erweiterungen für Echtzeitsysteme im Automobilbereich.

Mit Version 2.0 können sie jetzt komplexe Systeme durchgängig entwerfen und das Verhalten eines Systems optimiert darstellen. Auch an der Semantik hat das Industriekonsortium OMG gearbeitet, damit die UML-Diagrammtypen eindeutiger und damit die Module ausführbar sowie automatisch weiter zu verarbeiten sind.

Bei den Eingaben zur Superstructure besteht die Neuerung vor allem in einem sauber definierten Metamodell, mit dem Anforderungen während der Analyse- und Designphase besser als bisher zu verfolgen und prüfen sind.

Die Sequenzdiagramme für Objekte und Interaktionen teilen sich nun in Subroutinen auf, die sich schachteln lassen. Komponentendiagramme können in Version 2.0 – ähnlich wie bei Statemate – neben Schnittstellen auch Abhängigkeiten zu anderen Komponenten abbilden.

Für alle Seiten profitabel

Für die Anwender ist der Umstieg auf UML 2.0 sicherlich einfacher zu bewältigen als für die Tool-Hersteller, welche die neuen Vorteile in vielen Fällen erst noch in ihre Systeme integrieren müssen. Von der neuen Version profitieren sicherlich auch die Kunden. Dem Auftraggeber kann per Simulation demonstriert werden, wie sich das System verhalten wird, wenn die Anforderungen wie vereinbart realisiert werden. Nachbesserungen sind zu diesem frühen Zeitpunkt ohne großen Aufwand möglich.

Die UML bietet also weitreichende Lösungsansätze für wesentliche Herausforderungen der Entwicklung:

- Überblick über komplexe Systeme
- Systematische Bildung von Modulen
- Schnelles Stabilisieren von Schnittstellen

Überblick und systematische Module

UML als gemeinsame Notation schafft noch keine Interoperabilität zwischen den Modellen unterschiedlicher Unternehmen. So muss definiert werden, wie die Modelle und Interfaces aussehen. Dies wird über Model Styleguides realisiert, wie es sie heute bereits für C gibt.

Eine saubere Klassifizierung von Funktionen durch Module schafft zwar bereits eine brauchbare Architektur, feiner strukturiert und übersichtlicher wird sie allerdings durch Objekte als übergreifendes Ordnungsprinzip, das sich die UML zu nutze macht. Objektorientierte Programmierung ist dennoch auch zukünftig die Antwort auf die oben genannten Herausforderungen eines Überblicks über komplexe Systeme sowie der systematischen Bildung von Modulen. Abläufe im System lassen sich mit Objekten alleine allerdings nicht darstellen.

Abläufe spezifizieren

Wenn ein Programmierer früher Code geschrieben hat, konnte er diesen nach einigen Jahren nur schwer nachvollziehen und damit auch die Abläufe im System. UML-Diagramme helfen ihm jetzt auf die Sprünge:

Zustandsfolgediagramme beschreiben dabei das individuelle Verhalten, Sequenz- und Kollaborationsdiagramme das Verhalten der Objekte untereinander. Die UML unterstützt den Entwickler zudem bereits in frühen Projektphasen, in denen die Einsatzmöglichkeiten von Tools bis dato relativ begrenzt waren. Über den inkrementellen Ansatz von Prozessmodellen

Architekturmodellierung

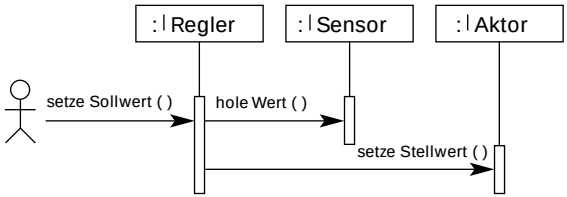
wie S.P.E.E.D oder ROPES schaffen Entwickler überschaubare Software-Teilprodukte. Früher lag der Fokus vorwiegend auf dem Code, der am Ende seine Dienste auf der Hardware verrichtete – oder auch nicht. Heute rücken Modelle für Anforderungsanalyse, Analyse, Design und Test immer mehr in den Vordergrund, da sie als weitere Software-Teilprodukte die Effizienz bei der Entwicklung maßgeblich steigern.

Besonders komplexe Systeme mit harten Echtzeitanforderungen lassen sich mit inkrementellen Verfahren besser in den Griff kriegen. Dafür planen die Entwickler einfach zusätzliche Schleifen oder Prototypen ein. Jedem Inkrement, also jedem Entwicklungsschritt, ist dabei ein eindeutiges Programmierziel mit einer klaren Zeitvorgabe zugeordnet.

Schnittstellen stabilisieren

Wer bei der UML-Modellierung Pakete mit Schnittstellen schnürt, kann deren Ineinanderwirken frühzeitig testen. Dies geschieht auf Basis von Simulationscode, der von Zustandsfolgemaschinen generiert wird. Die Schnittstellen lassen sich durch Szenarien, also Sequenzdiagramme in der UML, rasch stabilisieren.

Use Case: Regelung



Schnittstellen lassen sich über Sequenzdiagramme schnell stabilisieren. *Quelle: MicroConsult 2003*

Szenarien über Interfaces

Moderne Case-Tools sorgen dafür, dass neu eingetragene Methoden automatisch in die Interfaces der Szenarien übernommen werden. Klassen wie „Motor“ oder „Druck“ entstehen erst während Objektanalyse oder Design. Auf Basis von Zustandsfolge-Maschinen lässt sich daraus im nächsten Schritt bereits Code generieren. Damit kann das System in einem sehr frühen Stadium simuliert und das saubere Ineinanderwirken der Schnittstellen getestet werden. Ihre frühzeitige Stabilisierung sorgt für große Effizienz in der Teamarbeit, einfaches Testen im System und ein hohes Maß an Entkopplung.

Dies garantiert letztendlich auch verbesserte Portier-, Erweiter- und Wartbarkeit. Erfahrungen aus der Praxis zeigen, dass sich über dieses Verfahren in zehn Prozent der Projektzeit zwischen 70 und 80 Prozent der Schnittstellen festlegen lassen.

Wer im nächsten Schritt seine Architektur zur Ausführung bringen beziehungsweise simulieren möchte, kann Schnittstellen in Zustandsfolgediagramme einbauen. Diese werden später wieder entfernt oder weiterentwickelt. Die Interfaces beinhalten in diesem Stadium noch keine Funktionalität und dienen ausschließlich Simulationszwecken. Es wird dabei geprüft, ob die Komponenten auf hoher Ebene die funktionalen Anforderungen erfüllen, die in der Anforderungsanalyse festgehalten wurden. ◆

Schlaue Tools

Im Bereich der Modellierung von Algorithmen leisten Tools wie Matlab Simulink und ASCET-SD wertvolle Dienste. Matlab Simulink ist ein interaktives Werkzeug zur Modellierung, Simulation und Analyse von dynamischen Mehrdomänensystemen. Entwickler können damit ein Blockdiagramm erstellen, das Systemverhalten simulieren, Leistungsdaten bestimmen und ihre Entwürfe verfeinern.

ETAS ASCET-SD dient dem Entwurf und Realisieren der Steuergerätefunktionen. Mit dem Tool kann man Regelalgorithmen abstrakt entwerfen, testen, verbessern und anschließend aus der Spezifikation automatisch Steuergerätecode generieren.

Ein Problem können allerdings auch diese Tools noch nicht lösen: Die Schnittstelle zwischen Zulieferer und Hersteller basiert heute rein auf Requirements in Form entsprechender Dokumente, Modelle und Algorithmen. Die Architekturebene bleibt davon unberührt, obwohl sie ein immenses Potenzial zur Rationalisierung bietet.

Die Kombination der UML mit zeitkontinuierlichen Systemen ist von den Toolherstellern noch nicht offiziell gelöst. Für die Integration zeitkontinuierlicher und zeitdiskreter Modelle auf Basis von UML gibt es jedoch bereits drei Ansätze:

- Die Integration auf Ebene von Methoden und Funktionen: Dabei werden die Funktionen des Reglers aus einer Wrapper-Klasse (modelliert mit der UML) aufgerufen.
- Bei der Modelltransformation wird das eine Modell in das andere umgesetzt, beispielsweise zwischen Simulink und ASCET.
- Die Co-Simulation ist für das Requirements Management prädestiniert und bedeutet, dass die Simulation in beiden Entwicklungsumgebungen und sogar über Rechnergrenzen hinweg gleichzeitig abläuft.

Ein Tool dafür ist Excite von Extessy, das als Nischenprodukt unterschiedliche Domänen zusammenführt. Zur Zeit bietet diese Kommunikationsplattform eine Werkzeugkopplung zwischen Artisan und Matlab Simulink. Excite löst die Probleme der Toolkoordination, Simulationssynchronisation und verteilten Datenhaltung. Wie einfach die Kopplung der Ansätze durch Extessy sein kann, zeigt das Beispiel einer Scheibenwischersteuerung: Der Regler basiert auf einem Matlab-Modell, die Steuerung auf der UML mit Artisan.

Beide werden mit Excite zur Simulation gekoppelt. Der Vorteil dieser Lösung: Es können unterschiedliche CASE-Tools für die verschiedenen Aufgabenbereiche eingesetzt und trotzdem ein kompletter, durchgängiger Prozess geschaffen werden. ◆

Algorithmenmodellierung

Vom Makro- zum Mikrokosmos

Auf der Ebene der Implementierung rücken zwei Aspekte in den Vordergrund: Wie und auf welchen Standards wird umgesetzt? Momentan befindet sich die Implementierung in der Embedded Welt im Umbruch. Sie bewegt sich von einer prozeduralen zu einer objektorientierten Sichtweise, womit der Schritt zur Modellorientierung zwangsweise erfolgen muss, um eine brauchbare Architektur zu schaffen. Diese berücksichtigt Makro- und Mikrokosmos gleichermaßen: Das Objekt im Automobil harmoniert dann mit dem Objekt im Steuergerät.

Heutige Sprachen wie Assembler oder C reichen nicht mehr aus, um die geforderte Komplexität im Automobil zu schaffen, die im wesentlichen auf der Integrationsfähigkeit beruht.

Bis dato beschränkte sich die Kunst des Zulieferers darauf, die Spezifikation des Herstellers in prozeduralen Code mit einer relativ unsicheren Architektur umzusetzen. Der Makrokosmos des Gesamtfahrzeugs geriet dabei schnell aus dem Blickfeld. Der Zwang zur Integration führte im Automobilbau zur Etablierung von Standards wie OSEK, der Java-Applikationsplattform OSGI oder Automotive Corba als Zukunftsvision. So lässt sich die Branche die Standards nicht von den Tool-Herstellern diktieren, sondern schafft sie schrittweise selbst. Dies gilt vor allem für die Betriebssysteme.

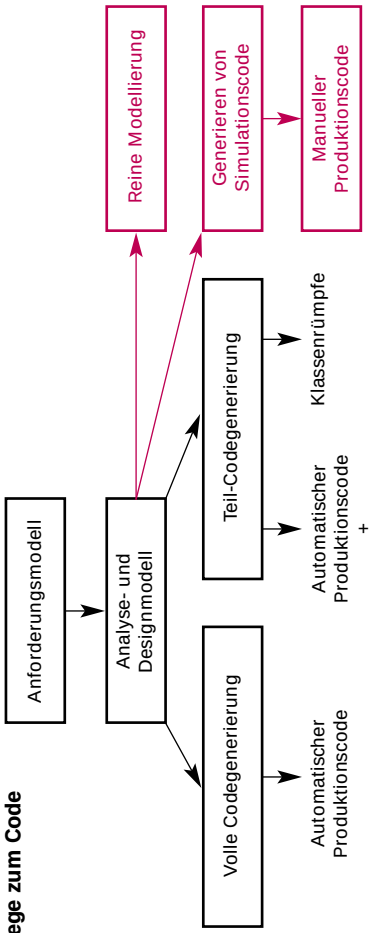
Für den Embedded Bereich kann der modellbasierte Ansatz kritisch sein, weil ein hoher Grad an Abstraktion generell mehr Speicher und Performance kostet als weitgehend optimierter, manuell erstellter Code. Bei einem 8-Bit Mikrocontroller mit wenig Möglichkeiten ist die modellbasierte Methode also sicher überdimensioniert. Allerdings werden in der heutigen Embedded Welt die Themen Speicherverbrauch und Performance in vielen Fällen überbewertet. Im mittleren und unteren Stückzahlenbereich sollten sich Entwickler die Frage stellen, ob sie aus dem Controller wirklich das letzte Bit heraus holen müssen.

Wie so häufig gilt auch hier das Pareto-Prinzip: In 20 Prozent des Systems stecken 80 Prozent seiner Performance. Wer diese 20 Prozent optimiert, die möglicherweise in Assembler oder C geschrieben sind, kann die restlichen 80 Prozent mit modellbasierter Software-Entwicklung abdecken und trotzdem ein hochoptimiertes, performantes System erzeugen. Mögliche Quellen für Ineffizienz gibt es zahlreiche. So besteht eine handgeschriebene Applikation statistisch gesehen nach zwei Jahre zu 20 Prozent aus totem Code. Wer diesen Anteil darauf verwendet, modellbasiert zu programmieren, arbeitet wesentlich effizienter.

Kritische Bereiche isolieren

Weil ein Modell komplexe Systeme grafisch und damit übersichtlich darstellt, kann der Entwickler die kritischen Bereiche bereits innerhalb der Architektur identifizieren und frühzeitig Gegenmaßnahmen einleiten. Und er kann die 20 Prozent erkennen, die 80 Prozent der Performance beeinflussen. Es gilt, diese performance- oder echtzeitkritischen Bereiche, die über viele Pakete verteilt sind, aus dem Gesamtsystem zu isolieren. Sie können später beispielsweise in Assembler und der Rest modellbasiert programmiert werden.

Wege zum Code



Vom Anforderungsmodell bis zum fertigen Code gibt es viele mögliche Wege. Der rote Bereich bedeutet, dass aus dem Modell kein Code für das Steuergerät produziert wird. Damit ist die Synchronizität von Code und Modell gefährdet. *Quelle: MicroConsult 2003*

Das Spektrum der Möglichkeiten reicht von der reinen Modellierung bis zum komplett automatischen Produktionscode. Modellierung und Simulation dienen der Funktionsdefinition, welche die Vorlage für die manuelle Implementierung bildet. Das birgt allerdings die Gefahr des Verlusts der Modell-Code-Synchronizität, das heißt der Entwickler berücksichtigt das Modell während des Codeschreibens nicht mehr. In speziellen Fällen kann das automatische Generieren von Produktionscode sinnvoll sein, wenn das Tool diese Möglichkeit bietet wie beispielsweise TargetLink von dSPACE. Der Königsweg wird zukünftig wahrscheinlich die Teil-Codegenerierung mit verlinkten, manuell codierten Libraries sein.

Vorhandener Code

Die innere Struktur von Softwaresystemen erschließt sich dem Betrachter häufig nicht sofort, weil der vorhandene Code meist schlecht strukturiert ist. Dieses Problem können Entwickler von Beginn an mit gutem Design und einer funktionalen Partitionierung umgehen.

Aus der Algorithmenmodellierung entstehen Teile, die in das Architekturmodell eingeklinkt werden müssen. Sie werden dann wie vorhandener Code betrachtet. ◆

Netzwerktechnik

Schlüssel zur Systemintegration

Die Integration neuer Komponenten im Fahrzeug gestaltet sich in zunehmendem Masse schwierig, wenn sie die schon bestehende Komplexität im Fahrzeug weiter erhöht. Notwendig ist deshalb ein industrieweiter Plattformansatz, der durch Standards unterstützt wird. Dabei müssen die Hardware- und Softwareschnittstellen so ineinander greifen, dass sich ein Ökosystem an Lieferanten etablieren kann, welche die verschiedenen Produkte zuliefern und Dienstleistungen bereit stellen.

Die Einführung offener Schnittstellenstandards wurde in den letzten Jahren mit dem Ziel betrieben, die wesentlichen Parameter gleichzeitig zu verbessern: Qualität, Flexibilität, Innovation und langfristige Verfügbarkeit der Komponenten. Dabei gebührt besondere Aufmerksamkeit den drei treibenden Netzwerkstandards im Auto: LIN, CAN und FlexRay. Jeder Standard steht dabei für eine Entwicklungsphase der automobilen Netzwerktechnik:

CAN: voll vernetzt

Dieser Standard stand am Beginn der Vollvernetzung im Fahrzeug, bei der die leistungsfähigen Module miteinander verknüpft werden. Mercedes Benz hat in den frühen 90er Jahren den CAN-Standard ausgewählt, der sich in der Folge in zwei Varianten als globaler Standard etabliert hat: 125 kbit/s versehen mit Fehlertoleranzmechanismen im Body-Bereich und 500 kbit/s im Motorsteuerungs- und Chassis-Bereich. Leider waren zur Einführung des Busprotokolls die Softwareschichten noch nicht festgelegt, was zu unterschiedlichen Implementierungen bei den verschiedenen Fahrzeugherstellern geführt hat. Das neue OSEK-COM 3.0.1 als klares Software-Interface soll nun diesen Mangel ausgleichen.

LIN: Schritt zur Mechatronik

Dieses Busprotokoll wurde zur Unterstützung des mechatronischen Konzepts entwickelt. Mit 20 kbit/s ist es wesentlich langsamer als CAN und ermöglicht als Sub-Bus die Aufteilung des Systems in mechatronische Komponenten. Mechatronik ist in diesem Zusammenhang so zu verstehen, dass die mechanischen Bauteile wie Motor, Sensor oder Ventil ihre eigene Ansteuer- oder Auswerteelektronik besitzen, die sich auch in enger räumlicher Nähe zur Mechanik befindet. LIN soll außerdem die Systemkosten auf der Mechatronik-Seite so gering wie möglich halten. Das Ergebnis:

- eine konsequente Master-Slave Architektur, bei der die Mechatronik-Module als „Slave“ agieren,
- kostengünstige Slaves, die ohne eigenen Quarz auskommen,
- eine zeitgesteuerte Signalarchitektur, welche die Signallaufzeiten schon beim Entwurf festlegt und dadurch die Systemintegration erheblich vereinfacht,
- festgelegte APIs als Software-Schnittstellen zwischen Kommunikations- und Applikations-Code,
- ein Software-Entwurfssystem mit einer festgelegten „Configuration Language“, das die Topologie des Netzwerks und deren Signalzusammensetzung ein deutig beschreibt.

Seit einigen Jahren beweist LIN im Serieneinsatz seine hervorragende Integrationsfähigkeit. Nachweisen muss LIN aber noch, dass es wirklich das mechatronische Konzept beflügelt und nicht als schwaches CAN-Äquivalent missbraucht wird. Dazu gehört auch eine einheitliche LIN-Steckerfestlegung, die bisher nicht existiert.

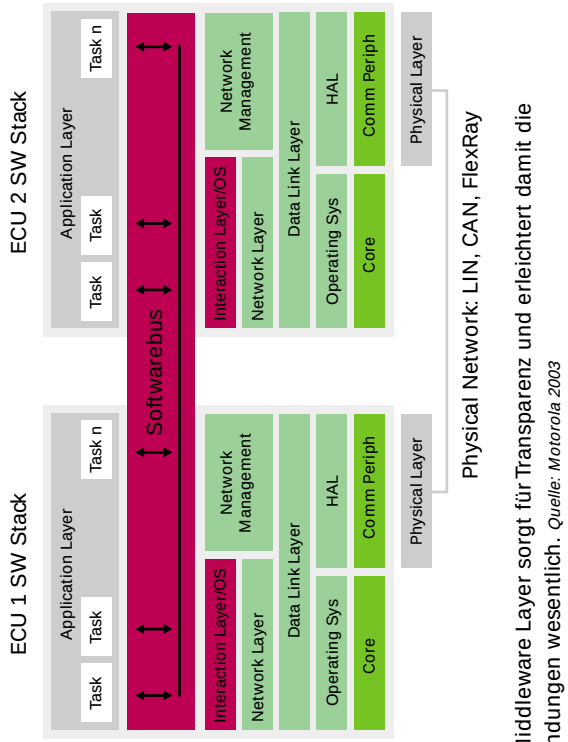
FlexRay: Zukunft für X-by-Wire

Dieser Standard ist heute noch in der Entwicklung von einem Konsortium aus Automobilherstellern, Zulieferern und Halbleiterproduzenten. Das Ziel ist ein Bussystem für fehlertolerante Kommunikation im Automobil, das für sicherheitskritische Steuer- und Regelaufgaben eingesetzt werden kann. Dabei sind die harten Echtzeitanforderungen im fehlertoleranten Umfeld ebenso wichtig wie die Systemintegrationsfähigkeit der Komponenten. Das Protokoll ist offen, das heißt es werden keine Lizenzgebühren für die Benutzung des Standards erhoben. Dies ist eine wesentliche Voraussetzung für eine schnelle Verbreitung und das Entstehen eines Ökosystems an Zulieferern, die sich diesem Standard anschließen.

Notwendig ist FlexRay für die neuen X-by-Wire-Systeme, die folgende wesentlichen Eigenschaften aufweisen müssen:

- Erweiterung der Busgeschwindigkeit auf 10 mbit/s, die dem Automobilnetzwerk mehr Bandbreite als bisher zur Verfügung stellt,
- Uhrensynchronisation der verteilten Module am Bus, die den teilnehmenden Knoten eine genaue synchronisierte Zeitbasis bietet,
- zeitgesteuerte Eigenschaften der TDMA-basierten Signalübertragung, welche die Latenzzeiten der Signalübertragungen vorhersehbar gestaltet und damit das System für harte Echtzeitbedingungen tauglich macht. Das arbitrierungsfreie Übertragungsverfahren unterstützt dabei den Determinismus der Signalübermittlung und gestattet den Einsatz in harten Echtzeitanwendungen, bei denen Kontrollschleifen über den Bus geschlossen werden können,
- Fehlertoleranz-Eigenschaften, die den Einsatz in sicherheitskritischen Systemen ermöglichen,
- Fehlereindämmung durch den Physical Layer, welcher der Busmonopolisierung durch einen fehlerhaften Knoten entgegenwirkt, wie beispielsweise dem Babbling-Idiot-Phänomen,
- Composability: Durch die Signalstruktur erleichtert das Protokoll die Erweiterbarkeit oder das Zusammenführen von Funktionen, ohne die Busknoten einer erneuten Programmierung zuführen zu müssen.

Das Konsortium besteht heute aus etwa 40 Mitgliedern. Mittlerweile haben sich alle namhaften Automobilhersteller für FlexRay ausgesprochen und planen den zukünftigen Einsatz in Serienfahrzeugen.



Ein Softwarebus als Middleware Layer sorgt für Transparenz und erleichtert damit die Integration von Anwendungen wesentlich. *Quelle: Motorola 2003*

Softwarebus: die nächste Generation

Um die Integration der Komponenten in die Busstrukturen transparent zu gestalten, arbeiten alle Entwickler an hardware-abstrahierenden Methoden. Dabei dient ein Middleware Layer als Zwischenebene, um die reinen Netzwerkfunktionen von den Applikationen zu trennen. Man kann sich den Middleware Layer auch als Softwarebus vorstellen. Er sorgt dafür, dass die applikativen Tasks über eine festgelegte Schnittstelle miteinander kommunizieren (siehe Abbildung). Das kann auch über verschiedene MCUs hinweg funktionieren. Wichtig ist, dass die zeitlichen Zusammenhänge besonders berücksichtigt und die maximalen Latenzzeiten für die Kommunikation zwischen den Tasks nicht überschritten werden.

Aus Sicht der Applikationen wird der Netzwerkraum transparent und die Integration einfacher, da der Softwarebus die Adressierung übernimmt und die Anwendungen somit leichter verschoben werden können. Ähnliche Konzepte sind im Desktop-Bereich schon länger etabliert. Hier kommt Middleware standardmäßig zum Einsatz. Die Konzeptionierung und Implementierung für die automobilen Echtzeitanforderungen müssen noch entwickelt werden. Der Softwarebus lässt sich statisch oder auch dynamisch konfigurieren. Eine optimale Tool-Unterstützung für die Konfigurationen wird ein Muß sein, um die steigende Komplexität zu beherrschen. Tools wie der Network Architect von Volcano Automotive ermöglichen heute eine ganzheitliche Netzwerkkonfiguration. ◆

Starke Peripherien

Im Bereich der Chiparchitektur sind die Halbleiterhersteller auf dem Weg, die MCU-Kerne zu standardisieren. Damit sollen sich langfristig extrem hohe Stückzahlen realisieren lassen, um die Kosten nachhaltig zu senken. Die Herausforderung besteht darin, dass die individuelle Chipfunktionalität nicht mehr durch Software, sondern durch Hardware in der Peripherie realisiert wird.

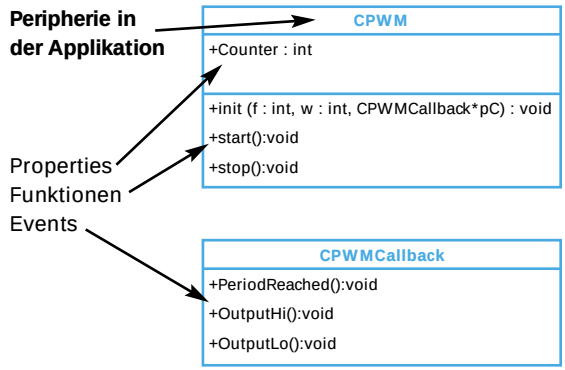
Dadurch entstehen komplexe Peripherien, die trotzdem einfach konfiguierbar sein müssen. Ein Controller muss deshalb äußerst komplexe Low-Level-Treiber bereit stellen, welche die Hersteller mit einer umfassenden Bibliothek liefern. Die bessere Alternative sind allerdings Code-Generatoren für Treiber.

Einen interessanten Ansatz für neue Chiparchitekturen bietet beispielsweise Processor Expert von Metrowerks. Die Software schafft ein abstraktes Modell für die Peripherie, bestehend aus Properties, Funktionen und Events. Properties spiegeln den Registersatz wider, Funktionen bieten auf Ebene eines einfach zu programmierenden C-APIs einen applikationsspezifischen Zugang zur Peripherie. Events bilden die Interrupts ab. Der Clou: Entwickler sind nicht mehr mit der Komplexität des Chips konfrontiert, sondern können sich auf die Applikation konzentrieren.

Eine weitere Innovation bilden die Ansätze zum Hardware/Software Co-Design. Damit ist eine Gesamtsicht auf das System entweder auf Ebene der Anforderungsspezifikation oder auf Ebene der Algorithmenbildung möglich. Diese besteht aus elektronischen, mechanischen, hardware- und softwaretechnischen Komponenten.

Beispiel hierfür ist Saber von Synopsis. Das Tool bringt alle Modelle auf der Mixed-Signal-Ebene zusammen, das heißt Chip, elektronische Ansteuerung und Mechanik werden in einem übergreifenden Mixed-Signal-Modell beschrieben. Letztendlich muss sich dieser durchgängige Ansatz aber auch in der organisatorischen Struktur des Automobilherstellers widerspiegeln, der heute noch die Hardware-, Software und mechanischen Abteilungen streng trennt.

Peripherien für Chips



Mit einem abstrakten Modell der Peripherie können sich Entwickler auf die Anwendung konzentrieren und müssen sich nicht mehr mit der Komplexität des Chips auseinandersetzen. *Quelle:MicroConsult 2003*

Stunde der Wahrheit

Wie viel Prozent eines Systems sollen heute getestet werden? 100 Prozent sind sicherlich nicht erreichbar, weil das System für den Test einen riesigen Raum aufspannt. So müssten alle Pfade in einem Programm mit allen möglichen Datenkombinationen durchlaufen werden. Und: Eine hohe Testabdeckung ist schlichtweg eine Frage der Methodik.

Deshalb müssen die qualitäts- bzw. sicherheitsrelevanten Bereiche des Systems möglichst frühzeitig fixiert werden. Für Kosteneffizienz gibt es mehrere Ansätze, die sich über das V-Modell einfach erschließen:

- Die Anforderungsspezifikation muss die Nichtfunktionen eines Systems klar definieren. Das reduziert den zu testenden Raum erheblich.
- Die Qualitätsmerkmale sind klar zu spezifizieren und priorisieren. Damit weiß jeder Projektbeteiligte, wie und was er zu entwickeln hat.
- Die Güte der Spezifikation hat damit direkten Einfluss auf die effektive Testabdeckung des Systems. So schließt sich der Kreis zum Requirements Capturing und der modellbasierten Requirements Analyse.
- Auf Ebene des Modultests sollten die Entwickler je nach Sicherheitsrelevanz mit einbezogen werden. Sie erhalten damit eine neue Sichtweise auf das System. Der Programmierer erstellt dann nicht mehr nur eine Funktion oder ein Modul, sondern entwickelt parallel Testtreiber, mit denen die Funktion nachgewiesen wird. Um Betriebsblindheit zu vermeiden, sind Reviews der vom Entwickler gefundenen Testfälle und durchgeführten Tests allerdings unerlässlich.
- Für Komponenten- und Integrationstests bietet sich der Einsatz von Werkzeugen für die Automation an. Standards wie OSEK oder Netzwerkprotokolle tragen wesentlich zur Reduktion des Testaufwands bei.
- Die formale Verifikation validiert nicht nur das System, sondern ermöglicht auch das Generieren von Vektoren aus der modellbasierten Requirements Analyse für das modellbasierte Testen. Damit lassen sich an den Systemklemmen Testvektoren einspeisen, welche die Funktion auf Basis eines hundert Prozent sicheren Modells testen. Dieses muss nicht dafür Sorge tragen, wie die Umsetzung von Seiten der Hardware und des Betriebssystems erfolgen muss. Dies ist letztendlich Aufgabe des Zulieferers.

Test aus Modell ableiten

Damit schließt sich der Kreis von der Requirements Analyse bis zum Test über alle sieben Stufen des hier beschriebenen Modells hinweg. Damit Code und Modell nicht auseinander driften, muss man aus dem Modell nicht unbedingt Code ableiten, aber den Test. Dann haben Software-Entwickler und Tester ein gemeinsames Interesse an der Konsistenz des Modells.

Leider ist die formale Methode heute noch auf zeitdiskrete Systeme begrenzt und es ist zu hoffen, dass die Grenze zur zeitkontinuierlichen Welt zukünftig fallen wird. Erste Ansätze dazu bietet das Verwalten von Testfällen innerhalb von Tools für Requirements Engineering oder Tracing, wie DOORS von Telelogic oder RequisitePro von Rational. Derzeit beginnen in Deutschland Automobilhersteller und -zulieferer damit, Strukturprozesse nach unterschiedlichen

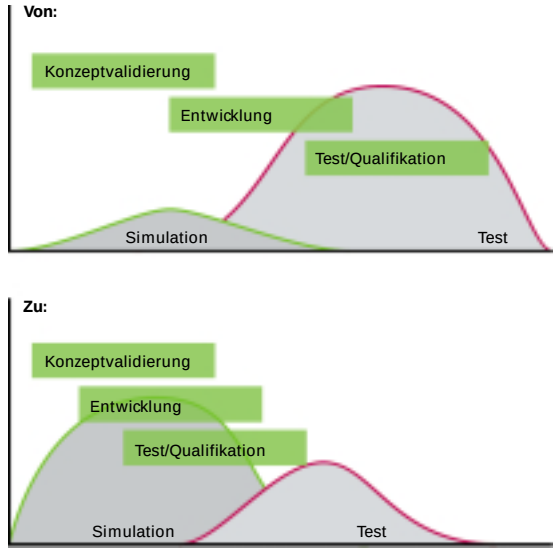
Prozessstandards auszurichten. Hier geht der Trend eindeutig in Richtung Zertifizierung. Ein führender Standard ist sicherlich das Capability Maturity Model (CMM und CMMI). Es definiert fünf Stufen der Reife einer Organisation bezüglich der Fähigkeit, Software-Entwicklungsprojekte durchzuführen. Seinen Ursprung hat es in einem Auftrag des Department of Defense, das die mangelhafte Qualität von Softwareprojekten in den Griff bekommen wollte.

Mit steigendem Reifegrad im CMM wird die Erwartung verbunden, dass die Vorhersagbarkeit von Terminen, Kosten und Qualitätszielen zunimmt. Für jede der fünf Stufen definiert das CMM, wie sich ein Projekt auf dieser Stufe darstellt und welche Maßnahmen durchgeführt werden.

Eine neuere, ebenso wichtige Norm für das Bewerten von Prozessen ist das von Europa favorisierte SPICE (ISO 15504). Dieser Standard berücksichtigt die Software-Entwicklung gültiger Normenwerke wie ISO 9000 und ISO 12207. Das Prozessmodell von SPICE ist feiner gegliedert als das in USA weit verbreitete CMM. Ersteres bietet 40 Prozesse, letzteres 18 Schlüsselprozessbereiche. SPICE wartet zudem mit Inhalten auf, die CMM nicht beinhaltet, beispielsweise für Betrieb und Wartung. SPICE geht in Einzelprozessen teilweise über CMM hinaus, wie im Projektmanagement.

Andererseits bietet CMM Inhalte, die in SPICE nicht enthalten sind wie die Koordination mehrerer Teams. Detaillierungstiefe und Ausführlichkeit sind bei CMM insgesamt größer. Einige Unternehmen kombinieren bereits beide Ansätze für eine größtmögliche Abdeckung.

Simulieren statt Testen



Der Trend geht heute eindeutig in Richtung Simulation und der Testanteil nimmt damit rapide ab.

Quelle: Volcano Automotive Group

Fazit

Vorteile auf allen Ebenen

Alle sieben Ebenen der Projektentwicklung bieten heute große Potenziale für Kosteneinsparungen:

- **Requirements Management:** Es muss ständig wechselnde Anforderungen in den Entwicklungsprozess einbringen und deren Verfolgbarkeit gewährleisten. So wird ersichtlich, wie sich Änderungen der Requirements auf andere Projektbereiche auswirken. Die Freiheit von Konflikten mit den anderen Teilsystemen wird durch eine Kapselung gewährleistet. Klar abgegrenzte und beschriebene Nichtfunktionen verringern den Entwicklungs- und Testaufwand. Modellbasierte Requirements Analyse sorgt für Konsistenz und Widerspruchsfreiheit. Durch die formale Verifikation kann validiert und eine 100%-ige Sicherheit des Modells gegen seine spezifizierten Eigenschaften garantiert werden.
- **Architekturmodellierung:** UML bietet die Chance, den Makrokosmos des Automobils auf den Mikrokosmos eines Steuergeräts abzubilden. Architekturstandards verringern die Integrationsprobleme maßgeblich. Zukünftig sollte eine Gesamtarchitektur aller Steuergeräte im Fahrzeug möglich sein.
- **Algorithmenmodellierung:** Durch die Co-Simulation kann das Verhalten von Mechanik und Software in verschiedenen Modellen gleichzeitig simuliert werden.
- **Implementierung:** Der richtige Einsatz von Codegeneratoren verringert nicht nur den Aufwand für die Implementierung, sondern schafft gleichzeitig Standards und sorgt für die Produktion von Code mit immer gleicher Qualität.
- **Design der Steuergeräte:** Für eine Standardisierung der Plattformen spricht die mögliche Erhöhung der Stückzahlen, was zur Kostenreduktion der Hardware führen wird.
- **Chiparchitektur:** Durch die Standardisierung der Steuergeräte-Plattformen können Halbleiterproduzenten wesentlich besser auf die Forderungen der Automobilhersteller und Zulieferer reagieren.
- **Qualität, Test und Integration:** Durch das Einbeziehen der Entwickler in den Modultest kann die Testabdeckung der Systeme mit geringem Kostenaufwand wesentlich erhöht werden. Testsuiten ermöglichen effiziente Regressionstests. Durch CMM oder SPICE getriebene Prozessstandards minimieren Kommunikationsprobleme zwischen Zulieferer und Hersteller. Beide profitieren von steigender Qualität. Mit das größte Einsparungspotenzial bietet der Kurzschluss von Requirements Analyse und Test, beispielsweise durch formale Verifikation und die daraus resultierenden, automatisch generierten Testvektoren.

Buch Tipps

Douglass, Bruce Powel: Doing Hard Time. Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns. Addison Wesley 1999, 749 Seiten, ISBN 0-20149-837-5
Das Werk bietet unter anderem eine aufschlussreiche Einführung in den Rapid Object-Oriented Process for Embedded Systems (ROPES).

Hay, David: Requirements Analysis Architecture. Prentice Hall 2002, 460 Seiten, ISBN 0-13028-228-6
Software-Entwickler Hay betrachtet die Requirements Analyse aus unterschiedlichsten Blickwinkeln. Dazu zählen neben Daten, Prozessen und Netzwerken auch Mitarbeiter und organisatorische Strukturen.

Krüger, Sven; Gessner, Wolfgang: Advanced MicroSystems for Automotive Applications 2003. Springer 2003, 560 Seiten, ISBN 3-54000-597-8
Die wichtigsten Ergebnisse der internationalen AMAA 2003 von VDI/VDE-IT fasst dieser Konferenzband zusammen.

Zurawka, Thomas; Schäuffele, Jörg. Automotive-Software-Engineering. Grundlagen, Prozesse, Methoden und Werkzeuge. Vieweg 2003, 280 Seiten, ISBN 3-52801-040-1
Die Autoren geben einen Überblick über die Schlüsselprozesse in der Automobil-Elektronik-Entwicklung und gehen auf die Details des Software-Engineering-Prozesses ein. Der Trend zur modellbasierten Softwareentwicklung unter Einsatz von grafischen Werkzeugen wird mit Praxisbeispielen an den automobilspezifischen Anforderungen reflektiert.

Web Tipps

Requirements Management
Doors: www.telelogic.com/products/doors
RequisitePro: www.rational.com
Matrixx: www.ni.com/products/matrixx
Statemate: www.ilogix.com/products
Veristate: www.motorola.com/eda/products/veristate/veristate.html
ModelChecker/ModelCertifier: www.osc-es.de/products/en

Architekturmodellierung
Object Management Group: www.omg.org
Dt. UML Special Interest Group: www.umlsig.de
The Precise UML Group: www.cs.york.ac.uk/puml

Algorithmenmodellierung
Matlab/Simulink: www.mathworks.de/products/industry/auto
ASCET-SD: www.etas.info/html/products/ec/ascetsd/de_products_ec_ascetsd_index.php

Implementierung
OSEK/VDX: www.osek-vdx.org
OSGI: www.osgi.org
ROPES: www.ilogix.com/quick_links/white_papers
S.P.E.E.D: www.bms.de/speed

Kommunikation
Flexray: www.flexray-group.com; www.flexray.com
LIN: www.lin-subbus.org
CAN: www.can-cia.de
MOST: www.mostcooperation.com

Kurse zum Thema

Objektorientierte Analyse mit der UML

Teilnehmer: Projektleiter in der Software-Entwicklung; Anwendungsprogrammierer; Software-Entwickler im Embedded Bereich

Vorkenntnisse: Programmiererfahrung (z.B. CHILL, Fortran, C, C++)

Kursziel: Kompetenter Einsatz von Analyse- und Entwurfsverfahren sowie der Darstellungsform der UML

Kursinhalt: Grundbegriffe der objektorientierten Programmierung; Einführung in die objektorientierten Basiskonzepte und Darstellung mit Hilfe der UML-Klassennotation; dynamisches Verhalten objektorientierter Software; Umsetzung in verschiedenen Programmiersprachen; Neuerung mit UML 2.0

Dauer: 4 Tage

Preis: 1.480 Euro + Ust.

Die aktuellen Kurstermine und das komplette Trainingsprogramm: www.microconsult.de

MicroConsult Leistungen

Objektorientierte Analyse für Embedded Systeme

Teilnehmer: Software-Entwickler von Embedded Systemen

Vorkenntnisse: Projekterfahrung mit Embedded Systemen und Grundkenntnisse UML (siehe Kurs Objektorientierte Analyse mit der UML)

Kursziel: Richtiges Einschätzen des Einsatzes der UML für Embedded Systeme; systematisches Entwickeln von Softwaresystemen – von der Requirements Analyse bis zum Design

Kursinhalt: UML und Embedded Systeme; Vorgehensmodell S.P.E.E.D; Requirements Analyse; Analyse; Realtime Analyse; Design; Beispielprojekt als praktische Übung

Dauer: 5 Tage

Preis: 1.990 Euro + Ust.

Die aktuellen Kurstermine und das komplette Trainingsprogramm: www.microconsult.de

Kurse zum Thema

Toolkopplung und Co-Simulation in der System- und Softwareentwicklung

Teilnehmer: System- und Softwareentwickler; Anwendungsentwickler; Projektleiter; Systemintegratoren

Vorkenntnisse: Grundkenntnisse der UMLsiehe Kurs Objektorientierte Analyse mit der UML) oder Beschreibung reaktiver Systeme; Projekterfahrung in der Entwicklung technischer Systeme

Kursziel: Kombination der Modelle für individuelle Aufgabenstellungen; Kopplung von Werkzeugen der UML und der Regelungstechnik (Blockschaltbilder, Signalflussgrafen) praktisch umsetzen

Kursinhalt: Grundbegriffe der regelungstechnischen Systemmodellierung; Anwendung der UML zur Modellierung von technischen Systemen; Kombination von Beschreibungssprachen; Kopplung von Simulationswerkzeugen

Dauer: 3 Tage

Preis: 1.150 Euro + Ust.

Die aktuellen Kurstermine und das komplette Trainingsprogramm: www.microconsult.de

MicroConsult Leistungen

Software Projektmanagement

Teilnehmer: Projektleiter; Mitglieder von Projektteams

Vorkenntnisse: Projekterfahrung

Kursziel: Methoden und Vorgehensweisen in Entwicklungsprojekten kennen; Wissen zur Optimierung des eigenen Projekts anwenden

Kursinhalt: Definition und Methoden des Projektmanagements; Projektorganisation; bewährte Vorgehensmodelle im Produktentwicklungsprozess; Projekte realistisch planen; Techniken und Prozesse des Projektcontrollings; Simulation mit SimulTrain

Dauer: 5 Tage

Preis: 1.990 Euro + Ust.

Die aktuellen Kurstermine und das komplette Trainingsprogramm: www.microconsult.de

Service

Partnerverzeichnis

Fokus	Firma	Logo	Info	Kontakt
UML-Tools	ARTiSAN Software Tools GmbH Eupener Str. 135 – 137 D-50933 Köln		www.artisansw.com	Christiane Kapteina Tel.: +49 (0)221 / 485 22-60 christiane.kapteina@artisansw.com
Entwicklung, Beratung und Werkzeuge	Berner & Mattner Systemtechnik GmbH, Otto-Hahn-Str. 34 D-85521 Ottobrunn		www.bms.de	Georg Zimmermann Tel.: +49 (0)89 / 60 80 90-160 georg.zimmermann@bms.de
Software Engineering and Design	HERMES SoftLab Litijska 51 SLO-1000 Ljubljana		www.hermes-softlab.com	Martin Weiss Tel. +43 1 994 96 50 21 martin.weiss@hermes-softlab.com
Development Tools	Hitex Development Tools GmbH Greschbachstr. 12 D-76229 Karlsruhe		www.hitex.de	Christiane Simon Tel.: +49 (0)721 / 96 28-146 sales@hitex.de
Entwicklungs-werkzeuge & -umgebung, Emulatoren, Debugger	iSYSTEM GmbH Carl-Zeiss-Str. 1 D-85247 Schwabhausen		www.isystem.com	Helmut Kindermann +49 (0)8138 / 69 71-50 sales@isystem.com
Halbleiter-produkte für Embedded Control	Motorola GmbH Schatzbogen 7 D-81829 München		www.motorola.com /semiconductors	www.motorola.com /semiconductors >>Technical Support >>Technical Helpline
Hard- und Soft-ware-Entwicklungs-tools, Dienst-leistungen	Metrowerks Schatzbogen 7 D-81829 München		www.metrowerks.com	David Brook Tel. +49 (0)89 / 921 03 555 info_europe@metrowerks.com
On Top Solutions for System on Silicon Debugging	pls Programmierbare Logik & Systeme GmbH Technologiepark D-02991 Lauta		www.pls-mc.com	Heiko Riessland Tel.: +49 (0)35722 / 384-0 info@pls-mc.com

Impressum

Herausgeber:
MicroConsult GmbH
Rosenheimer Str. 143 b, 81671 München
Tel. 089 / 45 06 17-0, Fax 089 / 45 06 17-18
Internet: www.microconsult.de

Objektleitung:
Sabine Häring, MicroConsult

Autoren:
Klaus-Peter Rosenthal, MicroConsult
Dr.-Ing. Gerd Teepe, Motorola

Redaktion:
Eva Schulz, actimedia

Art Director:
Florian Gmach, entity38

Druck:
Druckerei Makowski

Trend Guide Medienpartner:

www.elektronikpraxis.de

MicroConsult GmbH • Schule für MicroElektronik & Informationstechnologie
Rosenheimer Straße 143b • D-81671 München • Tel.: (089) 45 06 17-71
Fax: (089) 45 06 17-17 • www.microconsult.de