The Simulation Game – alles Auslegungssache

Embedded-Systeme modellbasiert spezifizieren und simulieren

Dr. Klaus Birken, itemis AG

Die Gretchenfrage des Embedded-Ingenieurs: Läuft die Software auf der Hardware, ohne diese zu sprengen? Die Software muss alle erforderlichen Funktionen umsetzen und dabei die Hardware optimal ausnutzen. Die Wirtschaftlichkeit des Produkts wird hauptsächlich von den Hardware-Kosten bestimmt; dies gilt besonders für Produkte, die in hohen Stückzahlen hergestellt und verkauft werden.
In diesem Beitrag wird eine Methode und ein zugehöriges Tool vorgestellt, mit dem diese Interaktion von Hardware und Software modelliert und simuliert werden kann. Dies kann im Entwicklungsprozess bereits dann geschehen, wenn es noch keine Prototypen gibt und die Software-Implementierung ebenfalls noch in der Zukunft liegt. Egal, welche Änderungen der Systemingenieur am Modell seiner Software oder Hardware vornimmt - das Tool zeigt deren Auswirkungen direkt und live. Somit wird eine interaktive, spielerische Art des Systementwurfs möglich.

Entwicklung von Embedded Software und Systemen

Heutige Embedded-Produkte sind reich an Features und Funktionen. Die dafür nötige System-Architektur, Software-Architektur und auch die eigentliche Implementierung werden dadurch immer komplexer und herausfordernder für Architekten und Entwickler. Bereits zu einem sehr frühen Zeitpunkt während der Entwicklung ist eine der Kernfragen, ob das geplante Hardware-Design leistungsfähig genug ist, um alle Software-Szenarien abzudecken. Wenn man im späteren Verlauf der Entwicklung feststellt, dass die verbaute Hardware nicht ausreicht, um bestimmte Produktfeatures umzusetzen, gibt es im besten Fall nur noch wenige, kostspielige Möglichkeiten zur Rettung des Projekts. Dazu zählen Streichen von Features, aufwendige Software-Optimierungen oder Aufrüsten der Hardware.

In Märkten, die von hohen Stückzahlen bestimmt werden, ist die obige Frage umso interessanter und drängender. Dort hängt der betriebswirtschaftliche Gewinn maßgeblich von den Hardwarekosten pro Stück (sog. *bill of material*) ab, was zu der Tendenz führt, das Hardware-Design bereits im Vorfeld sehr knapp auszulegen. Gleichzeitig ist es in diesen Märkten meist schwierig, die Hardware kurzfristig zu ändern. Somit potenziert sich obiges Problem.

Wie können Architekten und Systemingenieure die Sicherheit erhöhen, dass Software- und Hardware-Design zusammenpassen, bevor das System fertig implementiert ist? Ein sehr üblicher Ansatz ist es, Erfahrung aus Vorprojekten zu extrapolieren, d.h. das über Jahre aufgebaute Know-how intuitiv oder systematisch auf neue Projekte anzuwenden. Dabei kommen meist generische Tools wie Excel-Tabellen zum Einsatz, um Performance, Ressourcenauslastung und Timing für die aufwendigsten Use-cases des neuen Produkts abzuschätzen.

Dieses Vorgehen ist stark abhängig von der Fähigkeit und Erfahrung der beteiligten Ingenieure. Die Betrachtung der statischen Systemstruktur (d.h. Architektur) reicht nicht aus, um den Ressourcenverbrauch bei dynamischen Abläufen zu bewerten. Daher müssen bei einer ad-hoc-Lösung (z.B. mit Excel-Tabellen) die dynamischen Abläufe in eine Tabellenform gebracht werden; der Entwickler erfindet bei jeder Analyse eine neue Methode. In diesem Beitrag wird eine Alternative zu dieser Vorgehensweise vorgeschlagen: Durch die

Kombination aus Modellierung von Software und Hardware zu einem frühen Zeitpunkt und dem Einsatz eines Simulationstools wird der Ingenieur in die Lage versetzt, ein lauffähiges Systemmodell zu erstellen und daran die nötigen Analysen durchzuführen (sog. *design space exploration*).

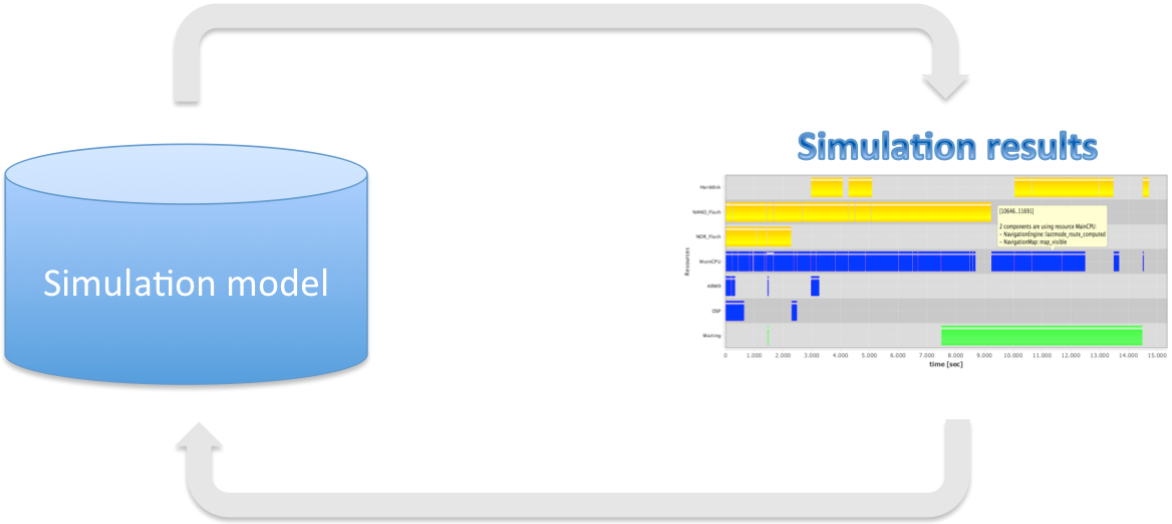

Abb. 1: Kurze Zyklen bei der Interaktion mit einem simulierbaren Modell.

Ziele der Simulation von Systemmodellen

Die hier beschriebene Methodik und das zugehörige Tool eröffnen folgende Analyse-Möglichkeiten:

- Architekturentscheidungen können so früh wie möglich bewertet werden.
- Timing-Anforderungen können untersucht werden, ohne das System vorher zu implementieren.
- Fehler im Design können frühzeitig aufgedeckt werden.
- Über die gesamte Entwicklungszeit kann überprüft werden, ob das System sich gemäß den geplanten Architekturentscheidungen verhält.
- In späteren Entwicklungsphasen können Optimierungen evaluiert werden, bevor sie tatsächlich umgesetzt werden.

Die obigen Möglichkeiten schließen dabei immer die Software-Seite ebenso wie die Hardware-Seite ein.

Abb. 1 zeigt die Interaktion mit einem ausführbaren Modell: Durch die höhere Abstraktionsebene des Modells (d.h. es wird kein Code simuliert, sondern ein abstrakter modelliertes Verhalten) kann die Simulation bei jeder Modelländerung ausgeführt werden. Damit werden die Auswirkungen von Designentscheidungen sofort sichtbar, sowohl bei Software- als auch bei Hardware-Änderungen. Bei der Arbeit mit einem realen Embedded-System dauert es meist mehrere Minuten, bis die Auswirkung einer Software-Änderung sichtbar wird; Hardware-Änderungen können überhaupt nur aufwendig in Form von Prototypen bewertet werden.

Wie werden ausführbare Modelle in der frühen Phase modelliert?

Damit ein System bereits in der Analysephase so modelliert werden kann, dass es mit Hilfe eines Simulators ausführbar ist, müssen geeignete Abstraktionen eingeführt werden. Die Hardware wird dabei auf wenige Modellkonzepte reduziert, die jeweils durch einen möglichst kleinen Parametersatz modelliert werden:

- Micro-Controller und Prozessoren werden über einen Leistungsfaktor, die Anzahl Kerne und ggfls. Angaben zu Prioritäten und Scheduling modelliert.
- Netzwerke, Kommunikationsverbindungen, Festplatten oder Flashbausteine werden unter dem Konzept der *bandbreiten-limitierten Ressourcen* zusammengefasst. Solche Ressourcen werden über ihre Bandbreite, ihre Auswirkung auf CPU-Zeiten und einen Faktor für Kontextwechsel-Kosten beschrieben.
- Schließlich werden *Pool-Ressourcen* wie z.B. Hauptspeicher oder abzählbare, beschränkte Ressourcen über ihre Größe bzw. Anzahl definiert.

Die obigen Angaben lassen sich für ein konkretes Modell entweder definieren, aus Vorprojekten extrapolieren oder über einfache Benchmarks ermitteln.

```
public functional component MediaServer {
  sim {
    on trigger load {
      ◇ application_loaded {
        read MediaImage : 22 MB via NAND_Flash.compressed
        use 500
      }
      ◇ init {
        read MusicMetadata : 30 MB via Harddisk.standard
        use 1500
      }
      ◇ available {
        precondition AudioManagement::available
        use 100
      }
      << ... >>
    }
  }
}
```

Abb. 2: Modellierung des Startup-Verhaltens einer Software-Komponente.

Die Softwarestruktur wird als Hierarchie von kommunizierenden Komponenten modelliert. Jede Komponenteninstanz kann ein oder mehrere Abläufe definieren, die von außen getriggert werden. Abb. 2 zeigt als Beispiel die Modellierung des Startup-Verhaltens einer MediaServer-Komponente, z.B. aus einem Entertainment-System. Dabei wird für jeden Ablauf eine Folge von Schritten definiert; bei jedem Schritt wird der Ressourcenverbrauch angegeben.

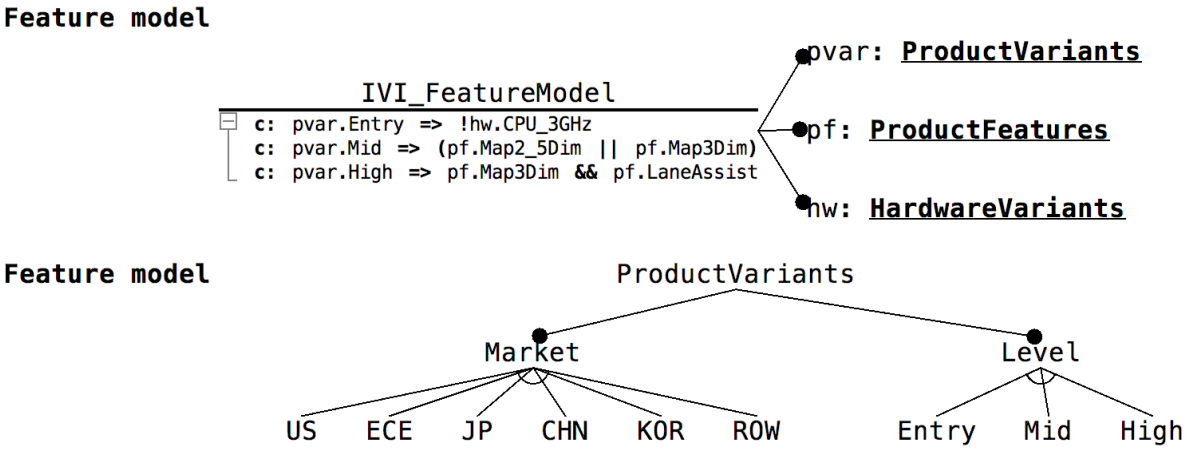

Abb. 3: Beschreibung von Produktlinien über Featuremodelle.

```
map_visible {
precondition Graphics::available
use 300
read MapDatabasePart :
```

Feature condition	Result
fm.pf.Map2Dim	15
fm.pf.Map2_5Dim	40
fm.pf.Map3Dim	80
default:	<no default>

```
MB via Harddisk.standard
}
```

Abb. 4: Modellierung eines Modellparameters in Abhängigkeit von Features einer Produktkonfiguration.

Modellierung von Produktlinien

Die hier beschriebene Methodik unterstützt auch die Beschreibung von Produktlinien. D.h. die ausführbaren Modelle beschreiben nicht nur ein einzelnes Produkt, sondern eine komplette Produktfamilie oder einen Satz von Produktvarianten. Durch Featuremodelle (siehe Abb. 3) kann die Produktlinie strukturiert werden und z.B. gegenseitige Ausschlüsse von Features definiert werden.

Im Modell können dann Variationspunkte eingeführt werden, um bestimmte Aspekte des Modells abhängig von ausgewählten Features einer bestimmten Produktkonfiguration für die Simulation zu steuern. Abb. 4 zeigt dies am Beispiel eines Modellparameters.

Wie werden die Ergebnisse von Simulation und Analyse dargestellt?

Eine wichtige Eigenschaft für die erfolgreiche Nutzung der Methodik ist, dass das verwendete Tool die Ergebnisse der Simulation unmittelbar wieder im Modell darstellt. Dies kann auf unterschiedlichste Arten passieren.

Abb. 5 zeigt, wie Timing-Ergebnisse aus der Simulation als Teil der modellierten Abläufe eingeblendet werden. Erkennt das Tool Probleme im dynamischen Ablauf (z.B. große Verzögerung entlang des kritischen Pfades), werden diese ebenfalls dargestellt. Ändert der Nutzer den modellierten Ablauf, läuft im Hintergrund die Simulation neu, so dass die Timing-Werte immer den aktuellen Stand des Modells wiedergeben.

```
public functional component NavigationEngine {
  sim {
    on trigger load {
      0 ms (ready: 0ms)
      application_loaded {
        use 80
        read NavEngineImage1 : 6 MB via NAND_Flash.compressed
      }
      1439 ms
      basic_operable ...
      9246 ms
      fully_operable ...
      10046 ms
      lastmode_route_computed ...
      13491 ms
      first_announcement ...
      13691 ms
      << ... >>
    }
    on trigger calculateLongRoute {...}
  }
}
```

Abb. 5: Darstellung des Timing-Verhaltens eines abstrakten Ablaufs. Die Ergebnisse werden direkt im Modell eingeblendet.

Abb. 6 zeigt ein Beispiel für ein Diagramm, mit dem der Ingenieur sich eine Übersicht zum Ablauf eines Szenarios verschaffen kann. Hier ist die Auslastung aller Ressourcen über die Zeit dargestellt.

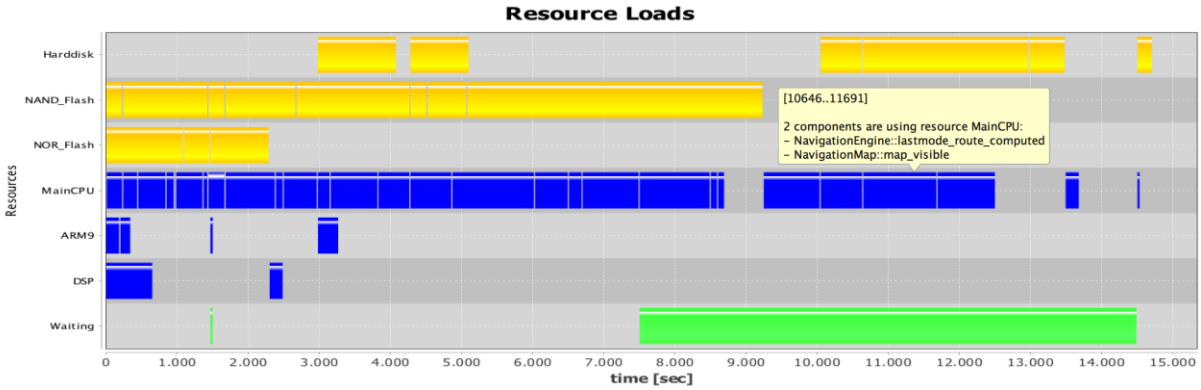

Abb. 6: Darstellung der Auslastung der Ressourcen über die Zeit.

Sofern die Timing- und Performance-Anforderungen des Systems nicht nur textuell (z.B. in DOORS), sondern semi-formal beschrieben wurden, können diese gegen die Simulationsergebnisse automatisch validiert werden. Somit wird es möglich, für jede Anforderungen sofort anzuzeigen, ob der aktuelle Stand des ausführbaren Modells die Anforderung erfüllen würde (siehe Abb. 7). Falls eine Anforderung nicht erfüllt wird, werden Informationen angezeigt, welche die weitere Analyse ermöglichen.

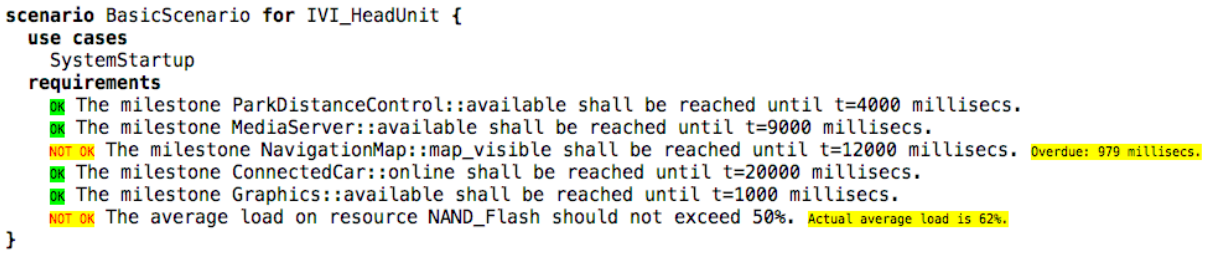

Abb. 7: Automatischer Abgleich der Simulationsergebnisse mit semi-formal definierten Anforderungen.

Hintergründe zur Tool-Plattform

Die obigen Abbildungen (außer Abb. 1) sind direkte Darstellungen aus dem innovativen Modellierungstool ISE (Integrated Specification Environment), welches die itemis AG seit Januar 2016 im Rahmen des IETS3-Forschungsprojekts entwickelt. Signifikante Teile dieses Tools sind unter einer Open-Source-Lizenz verfügbar. Aus den Abbildungen ist erkennbar, dass im ISE-Tool viele verschiedene Notationen im gleichen Modell kombiniert werden können, um die Fachkonzepte optimal auszudrücken. Dazu gehören textuelle und mathematische Notation, Tabellen, Baumstrukturen (z.B. Entscheidungsbäume, Feature-Diagramme) und grafische Elemente (z.B. Komponentenhierarchien).

Dafür ist eine Toolplattform notwendig, die es erlaubt, verschiedenste Modellierungsaspekte und Notationen flexibel zu kombinieren. Für ISE wird die Language Workbench MPS verwendet. MPS (Meta Programming System) ist eine Tool-Entwicklungs-Plattform, die unter Open-Source-Lizenz von der Firma JetBrains entwickelt wird [1]. Die Besonderheit von MPS ist es, anstatt dem (limitierenden) Einsatz von Parsern projizierende Editoren zu verwenden. Dadurch können die zugrundeliegenden Objektmodelle auf unterschiedliche Notationen projiziert werden.

Vorteile für die Embedded-Praxis

Die hier beschriebene Methodik wurde bereits in 2009/2010 entwickelt und im Bereich der Automotive-Infotainment-Systeme erfolgreich eingesetzt [2]. Die nun erfolgte neue Umsetzung auf Basis der MPS-Plattform schafft toolseitig mächtige Möglichkeiten, die den Architekten bzw. Entwickler eines Embedded-Systems befähigen, bereits zu einem sehr frühen Zeitpunkt ein tiefgreifendes Verständnis für die dynamischen Abläufe des Systems und damit für das Zusammenspiel von Hard- und Software aufzubauen.

Referenzen

[1] MPS (Meta Programming System) Homepage: https://www.jetbrains.com/mps.

[2] K. Birken, D. Hünig, T. Rustemeyer, R. Wittmann: *Resource analysis of Automotive/Infotainment systems based on domain-specific models – a real-world example.* In: Leveraging Applications of Formal Methods, Verification, and Validation, Volume 6416 of Lecture Notes in Computer Science pp 424-433, Springer, Heidelberg, 2010.

Autor

Dr. Klaus Birken studierte Informatik in Erlangen und promovierte zum Thema Parallelisierung an der Univ. Stuttgart. In zahlreichen Embedded-Projekten brachte er sein Wissen über Architekturen und Entwicklungsmethodik ein. Dr. Birken ist Vortragender auf internationalen Konferenzen und Autor (z.B. Buch "Basiswissen Softwarearchitektur", 3. Auflage). Nach fast einem Jahrzehnt als Infotainment-Architekt bei Harman arbeitet Klaus Birken seit 2012 als Principal Engineer bei der itemis AG.

Kontakt

Internet: www.itemis.de
Email: klaus.birken@itemis.de
Xing: www.xing.com/profile/Klaus_Birken