

Betriebssystemtechnik

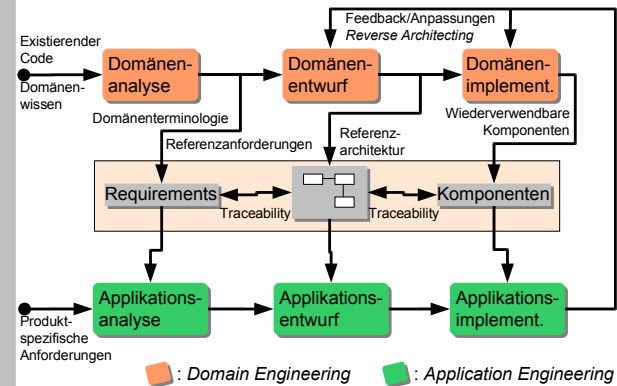
Operating System Engineering (OSE)

Domänenentwurf und Referenzarchitektur



1

Software-Produktlinienentwicklung

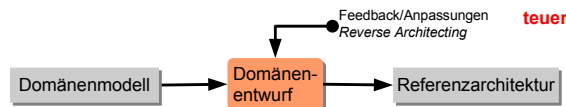


Referenzprozess für die Software-Produktlinienentwicklung [1]

© 2007 Olaf Spinczyk

2

Domänenentwurf



Analyse

- Identifikation der Architekturtreiber, Mechanismen und Sichten

Modellierung

- Architekturdokumentation mit Variationspunkten

Evaluierung

- Erkennen von Risiken in den Entwurfsentscheidungen



In Anlehnung an das Modell zur Architekturentwicklung aus [1].

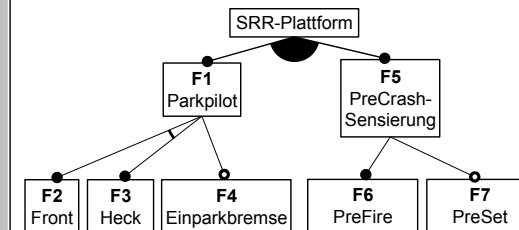
© 2007 Olaf Spinczyk

3

Beispielprojekt

Fahrerassistenz durch Short Range Radar (SRR) [1]

Merkmaldiagramm: Robert Bosch GmbH, SRR-Plattform



F3 und F5 schließen sich aus

F8 Genauigkeit der Geschwindigkeitsmessung
F9/F10 Anzahl und Position der Sensoren



© 2007 Olaf Spinczyk

4

Analyse: Architekturtreiber

- architekturrelevante Anforderungen
 - wenige im Vergleich zur Gesamtmenge der Anforderungen
 - häufig „nicht-funktionale Eigenschaften“

Architekturtreiber: **Robert Bosch GmbH, SRR-Plattform**

- Konfigurierbarkeit:** Bestimmte Ausprägungen und Kombinationen der Assistenzfunktionen Parkpilot und PreCrash-Sensierung sollen als Varianten der SRR-Plattform realisierbar sein.
- Integrierbarkeit:** Produkte der SRR-Plattform sollen kompatibel zu den KFZ-Elektronik-Infrastrukturen im Zielmarkt sein (CAN Bus, ..., begrenzter Bauraum).
- Performanz:** Parkpilot und PreCrash-Sensierung sollen gleichzeitig nutzbar sein.



Analyse: Entwurfsmechanismen

- Ermitteln geeigneter Mechanismen für die Umsetzung der Architekturtreiber
- Auswahl erprobter Lösungen für Entwurfsprobleme, z.B.:
 - Schichten:** Anordnung von Aufgaben unterschiedlichen Abstraktionsebenen
 - Pipes and Filters:** Am Datenfluss orientierte Architektur bestehend aus Operationen, die Datenströme bearbeiten
 - Model-View-Controller:** Zerlegung interaktiver Systeme in drei Hauptkomponenten: (1) Verwaltung der Daten, (2) Visualisierung, (3) Behandlung der Eingaben
 - Blackboard:** Mehrere unabhängige Komponenten arbeiten auf Basis einer zentralen Datenstruktur indirekt zusammen
- Entwurfsmechanismen/-muster werden **unter Beachtung der Architekturtreiber** gewählt und ggf. kombiniert.



Analyse: Architektursichten

- relevante Darstellungen der Architektur definieren
- Sichten ergeben sich anhand der Architekturtreiber
 - Performanz:** Darstellung mit Hilfe von Prozessen und Interprozesskommunikation (dynamischer Gesichtspunkt)
 - Korrektheit, Testbarkeit:** Benutzt-Hierarchie
 - Konfigurierbarkeit:** funktionale Hierarchie, Visualisierung der Konfigurationspunkte
 - ...



Modellierung

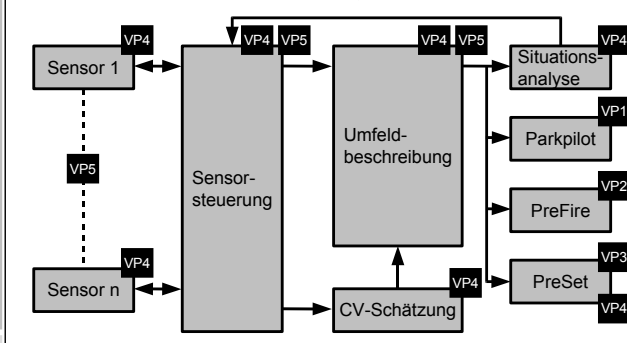
- Treffen von **Entwurfsentscheidungen**
 - unter Beachtung der Architekturtreiber und anderer Anforderungen
- Anwendung der Entwurfsmechanismen
 - schrittweise Verfeinerung
- Bearbeitung in **allen** relevanten Architektursichten
- Erstellung der Architekturdokumentation



Beispielprojekt - Architektur

- „Logische Sicht“ auf die SRR-Plattform

Referenzarchitektur: Robert Bosch GmbH, SRR-Plattform



Beispielprojekt – Treiber-Realisierung

- Konfigurierbarkeit
 - Minimierung von Abhängigkeiten durch Modularisierung
- Integrierbarkeit
 - Realisierung der Komponenten mit Standard CAN Schnittstellen
 - Minimierung der Sensoranzahl durch Nutzung für verschiedene Funktionen (Bauraumproblematik, auch Performanz)
- Performanz
 - Parallelverarbeitung durch Verteilung auf eine Plattform- und n Sensor-ECUs
 - Nutzung von Nebenläufigkeit durch *Time Sharing*-Betrieb

Beispielprojekt – Variationspunkte

- explizite Dokumentation der Variabilität

Variationspunkte: Robert Bosch GmbH, SRR-Plattform

Merkmal	Variationspunkt	Verantwortlichkeit
F1	VP1	Einbinden Komponente Parkpilot
F6	VP2	Einbinden Komponente PreFire
F7	VP3	Einbinden Komponente PreSet
F8	VP4	Konfigurierung des CV-Modus
F9, F10 (F2,F3)	VP5	Konfigurierung von Sensoranzahl und -positionen

- VP4 und VP5 sind besonders interessant
 - betreffen den inneren Aufbau der logischen Komponenten
 - wirken sich auf mehrere Komponenten aus („querschneidend“)

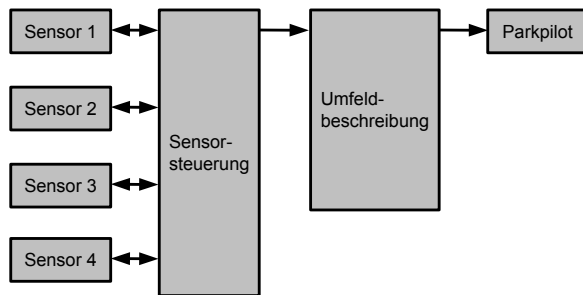
Evaluierung

- Festlegung von **Evaluierungsszenarien**
 - unter Beachtung der Architekturtreiber und anderer Anforderungen
 - kritische Anwendungsfälle, z.B. Volllastbetrieb, Fehlerfälle
 - Migrationspfad zur nächsten Produktgeneration
- Identifikation der betroffenen Komponenten pro Szenario
- Suche nach nicht umgesetzten Anforderungen
- Suche nach möglichen Risiken durch szenario- und qualitätsmerkmalspezifische Fragen
 - Welche Bedingungen oder Ereignisse beeinflussen die Systemperformanz? Wie reagiert das System darauf?

Beispielprojekt – Produkt 1

- Parkpilot Heck: F1, F3

Produktvariante 1: Robert Bosch GmbH, SRR-Plattform



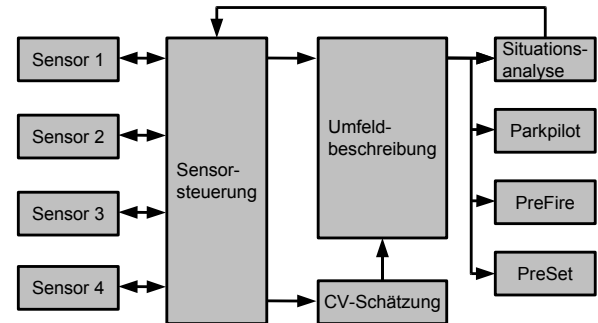
© 2007 Olaf Spinczyk

13

Beispielprojekt – Produkt 2

- Parkpilot und PreCrash-Sensierung: F1, F2, F5, F6, F7

Produktvariante 2: Robert Bosch GmbH, SRR-Plattform



© 2007 Olaf Spinczyk

14

Domänenentwurf - Zusammenfassung

wichtig ist, ...

- sich über die Architekturtreiber klar zu werden
- alternative Entwurfsmechanismen bzw. Architekturen abzuwägen (keine Dogmen!)
- die Variationspunkte explizit zu beschreiben
- durch geeignete Modularisierung Abhängigkeiten (insbesondere Zyklen) zu minimieren



© 2007 Olaf Spinczyk

15

Betriebssystemtechnik

Operating System Engineering (OSE)

Architektur von Betriebssystemfamilien



© 2007 Olaf Spinczyk

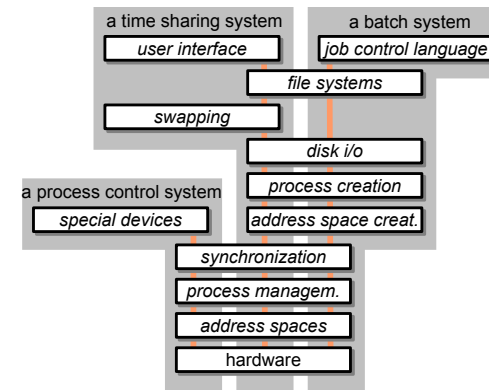
16

Architekturtreiber für BS-Familien

- Konfigurierbarkeit
 - geringer Konfigurationsbedarf **innerhalb** der Komponenten
 - große Variabilität (viele Alternativen)
 - feine Granularität (in kleinen Stufen Funktionen weglassen können)
- minimaler Ressourcenverbrauch
- keine Voraussetzungen hinsichtlich Infrastruktur
 - z.B. keine Interprozesskommunikation
- Beherrschung der Komplexität durch Abstraktion
- ? Echtzeitfähigkeit ?
- ? Fehlertoleranz ?



Beispiel – FAMOS [2]



Familienbasierter Entwurf (1)

- die Grundlage von FAMOS

*„Rather than write programs that perform the transformation from input to output data, we design **software machine extensions** that will be useful in writing many such programs.“*

D. L. Parnas [3]



Familienbasierter Entwurf (2)

- „minimal subset of system functions“
 - allgemeine Funktionen, die von den weiter spezialisierten Varianten benötigt werden
 - implementieren Mechanismen in Form fundamentaler Grundbausteine
- „minimal system extensions“
 - Verwendung, Spezialisierung oder Anpassung der fundamentalen Systemfunktionen
 - Kapselung der strategischen und anwendungsspezifischen Entwurfsentscheidungen
 - Schrittweise bottom-up entwickelt, aber top-down getrieben
- „**decisions which restrict the family are postponed as far as possible**“ [2]



Familienbasierter Entwurf (3)

- **Pro:**
 - Konfigurierung durch Weglassen
 - feingranulare Konfigurierbarkeit
- **Contra:**
 - viel Erfahrung und Weitblick nötig
 - „minimale“ Erweiterungen erfordern viel Geduld
 - modernere Ansätze versuchen, auch **zu viel an Variabilität** zu vermeiden
 - konfigurierbare Funktionen werden nicht berücksichtigt
 - querschneidende Belange und nicht-funktionale Eigenschaften lassen sich nicht darstellen



Hierarchien im Systementwurf

- Mehrere recht verschiedene Hierarchien können beim Entwurf relevant sein:
 - **Modul-Hierarchie**
 - **funktionale Hierarchie**
 - **Benutzt-Hierarchie**
- Relevanz hängt von der Architektursicht ab
- Untersuchung anhand eines gemeinsamen Beispiels



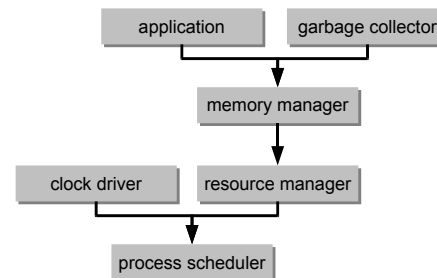
Fallstudie – BS Speicherverwaltung

- zur Vereinfachung besteht unser Beispielsystem nur aus sehr groben Systemkomponenten:
 - **garbage collector** (system thread): sucht nach reserviertem, aber unbenötigtem Speicher und gibt diesen frei
 - **memory manager**: verwaltet die Freispeicherliste, reserviert Speicher auf Anforderung, ordnet reservierten Speicher Prozessen zu
 - **resource manager**: stellt Synchronisation mit Semaphoren zur Verfügung
 - **process scheduler**: verwaltet die Ready-Liste, stoppt, verdrängt und aktiviert Prozesse auf Anforderung
 - **clock driver**: erlaubt den Entzug der CPU nach Ablauf einer Zeitscheibe



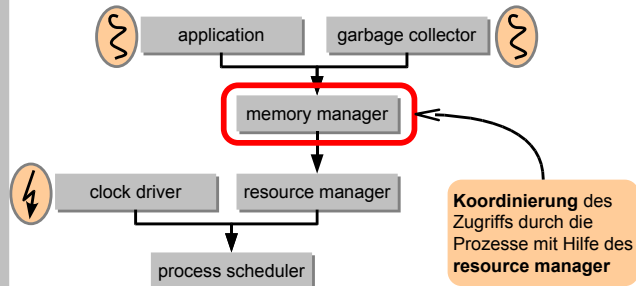
Modul-Hierarchie (1)

- Kanten sind die Aufrufbeziehungen zwischen Modulen
 - (Kapselung für Zustand und dazugehörige Operationen)



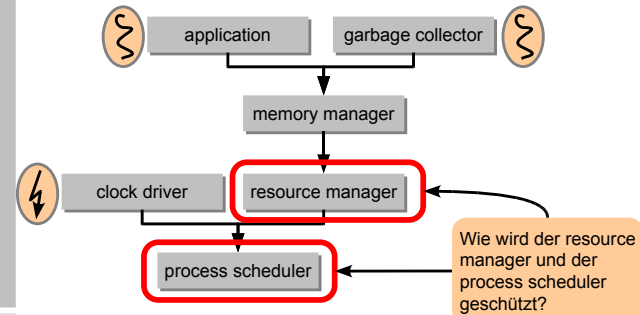
Modul-Hierarchie (1)

- Kanten sind die Aufrufbeziehungen zwischen Modulen
 - (Kapselung für Zustand und dazugehörige Operationen)

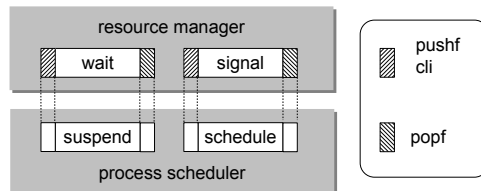


Modul-Hierarchie (1)

- Kanten sind die Aufrufbeziehungen zwischen Modulen
 - (Kapselung für Zustand und dazugehörige Operationen)



Modul-Hierarchie (2)



- Obwohl die Unterbrechungssynchronisationsfunktion sehr wichtig ist, wird sie nicht dokumentiert
- „logische Funktionen“ fehlen in der Modul-Hierarchie, wenn sie nicht als „technische Funktionen“ realisiert sind

Funktionale Hierarchie (1)

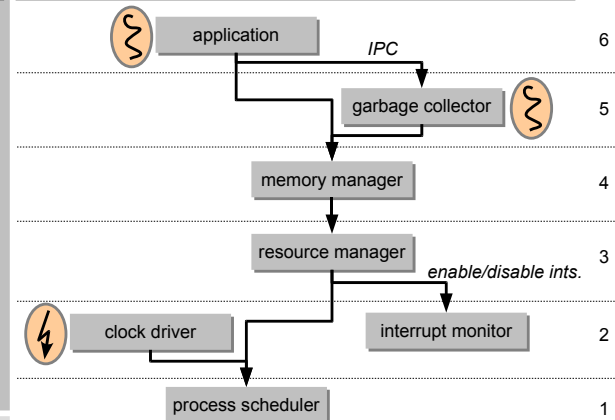
- beschreibt funktionale Abhängigkeiten zwischen den Systemkomponenten
 - logische Beziehung statt Aufrufbeziehung
 - Strukturen sind unter Umständen nur im Entwurf sichtbar
 - „It is the system design which is hierarchical, not its implementation“ [2]
- der Entwurf abstrahiert von der Funktionsimplementierung
 - „Funktionen“ können z.B. Prozesse, Module, Prozeduren oder Makros sein
 - die technische Realisierung der Funktionen könnte sogar konfigurierbar sein
- alle (logischen) Funktionen werden in der funktionalen Hierarchie erfasst

Funktionale Hierarchie (2)

- Unterteilt das System in Ebenen
 - es werden keine Aussagen über die Interaktion zwischen den Ebenen gemacht
 - Ebenen und Module sind prinzipiell unabhängige Konzepte
- die Ebene 0 repräsentiert die Funktionen der Hardware
- jede weitere Ebene 1 – n fügt eine neue virtuelle Hardware hinzu
- jede Ebene besteht aus statisch bekannten Funktionen
- alle Funktionen der Ebenen 0 – i sind in den darüber liegenden Ebenen j, mit $i < j \leq n$, **bekannt**



Funktionale Hierarchie (3)

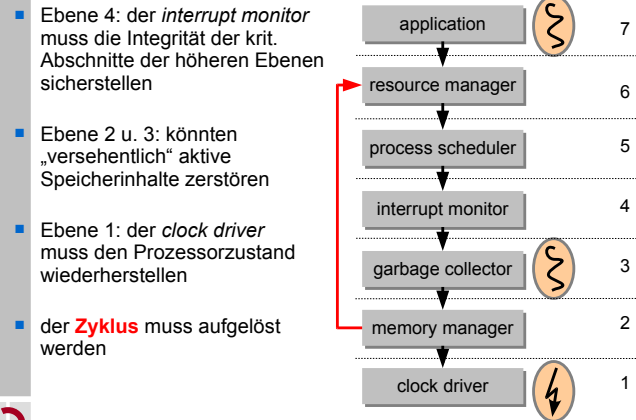


Benutzt-Hierarchie (1)

- beschreibt die Abhängigkeiten zwischen den (logischen) Funktionen hinsichtlich **Korrektheit**
- „A benutzt B“ bedeutet hier, dass die „Korrektheit von A von der Verfügbarkeit einer korrekten Implementierung von B abhängt“
 - dazu ist kein Aufruf nötig, z.B. bei Frameworks oder Interrupts
- da Korrektheit von der Implementierung **und der Spezifikation** abhängt, tut dies auch die benutzt-Relation
 - trotz eines Aufruf benutzt A B nicht, wenn z.B. A trotz inkorrektem B immer seine Spezifikation erfüllt

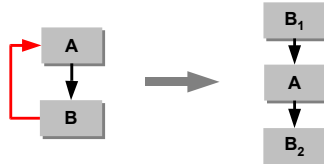


Benutzt-Hierarchie (2)



Sandwiching

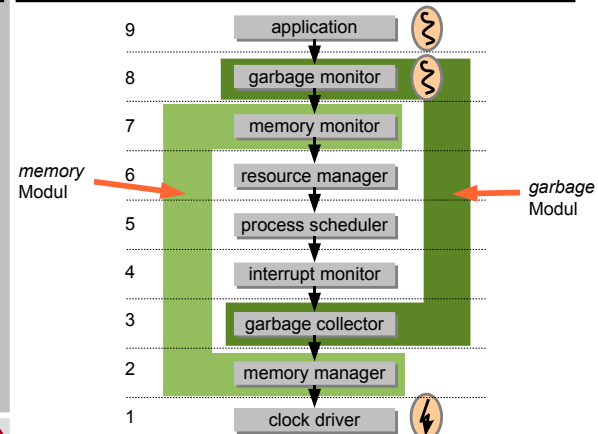
- Zyklische Benutzt-Relationen können durch Aufteilen einer der beiden Funktionen aufgelöst werden



- *memory manager* wird *memory manager* & **memory monitor**
- *garbage collector* wird *garbage collector* & **garbage monitor**



Benutzt-Hierarchie mit Sandwiching



Zusammenfassung

- für eingebettete Betriebssystemfamilien bieten sich aufgrund der nötigen Konfigurierbarkeit geschichtete Architekturen an
- je nach Architektursicht sollten man eine Benutzt- oder funktionale Hierarchie erarbeiten und daraus die Modul-Hierarchie ableiten
 - Sandwiching hilft, Zyklen aufzulösen
- querschneidende Belange und nicht-funktionale Eigenschaften nicht direkt erfasst
- es gibt einen *trade off* zwischen Entwicklungsaufwand und Variabilität der Systemplattform



Ausblick

- Vertiefung der Modularisierungsproblematik in Bezug auf „querschneidende Belange“
 - AspectC++ in Vorlesung und Übung
- Durchführung eines (Sub-)Domänenentwurfs in der Übung
- Untersuchung verschiedener Techniken zur Umsetzung von Variabilität in der Implementierung der Komponenten
 - „Domänenimplementierung“



Literatur

- [1] G. Böckle, P. Knauber, K. Pohl, K. Schmid (Hrsg.).
Software-Produktlinien. dpunkt.verlag, 2004.
ISBN 3-89864-257-7.
- [2] A. N. Habermann, L. Flon, and L. Coopridge.
Modularization and Hierarchy in a Family of Operating Systems. Communications of the ACM,
19(5):266-272, 1976.
- [3] D. L. Parnas. *Designing Software for Ease of Extension and Contraction*. IEEE Transactions on Software Engineering, SE-5(2):128-138, 1979.

