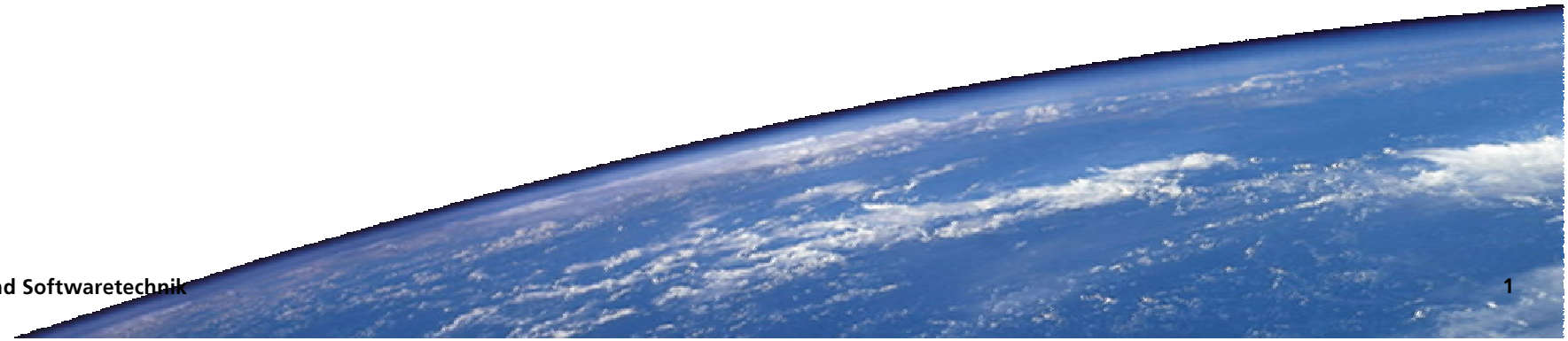
Objektorientierung in eingebetteten Echtzeitsystemen - Ein Widerspruch?

Olaf Maibaum

DLR, Simulations- und Softwaretechnik

http://www.sistec.dlr.de



Übersicht

- Begriffe
- Autonomes Lageregelungssystem des Kleinsatelliten Bird
- Anforderungen an die Softwaretechnik bei der Entwicklung eingebetteter Systeme
- Vorurteile zur Objektorientierung
- Nötige Einschränkungen von C++
- Standardbibliothek
- Speichermanagement
- Einsatz abgeleiteter Klassen
- Wiederverwendbarkeit
- Schlussbemerkungen

Begriffe

- Eingebettetes System (Embedded System)
  - Umgeben durch ein technisches System
  - Von außen nicht als Computersystem zu erkennen
- Echtzeitsystem (Real-Time System)
  - Vorhersagbares Zeitverhalten
  - Rechtzeitigkeit
  - Deterministisch
- Objektorientiert
  - UML in Architektur- und Detailed-Design Phase
  - Einsatz von C++ als höhere Programmiersprache

Autonomes Lageregelungssystem des Kleinsatelliten Bird

- **Aktorik**
  - 4 Reaktionsräder
  - 6 Magnetspulen
- **Sensorik**
  - 2 Sonnensensorsysteme
  - 1 Magnetfeldsensor
  - 1 Laser-Gyroskop
  - 2 Sternenkameras
  - 1 Onboard-Navigationssystem
- **Umfang der Lageregelungs-Software**
  - > 25000 Codezeilen
  - > 220 Klassen

Anforderungen an die Softwaretechnik bei der Entwicklung eingebetteter Systeme

- Exaktes Timing für zeitkritische Systemkomponenten
- Statisches Speichermanagement
- Hardwarenahe Programmbestandteile
- Ergänzungen um den Softwaretest zu ermöglichen

Vorurteile zur Objektorientierung

- Objektorientierte Anwendungen benötigen
  - leistungsstarke Prozessoren
  - viel Speicherplatz
  - sind schwer überschaubar
- Eingebettete Systeme lassen sich nicht mit objektorientierten Techniken entwickeln

Nötige Einschränkungen von C++

- Einschränkungen die auch in C nötig sind
- C++-spezifische Einschränkungen
  - Keine Ereignisse verwenden (`try`, `throw` und `catch`)
  - Bedachter Einsatz von `inline` bzw. Definition von Methoden in Headerdateien
  - Eingeschränkter Einsatz des Schlüsselworts `virtual`
  - „Wrapper“-Klasse für Templates verwenden
  - Zustandsänderungen nur lokal auf einer Klasse

Standardbibliothek

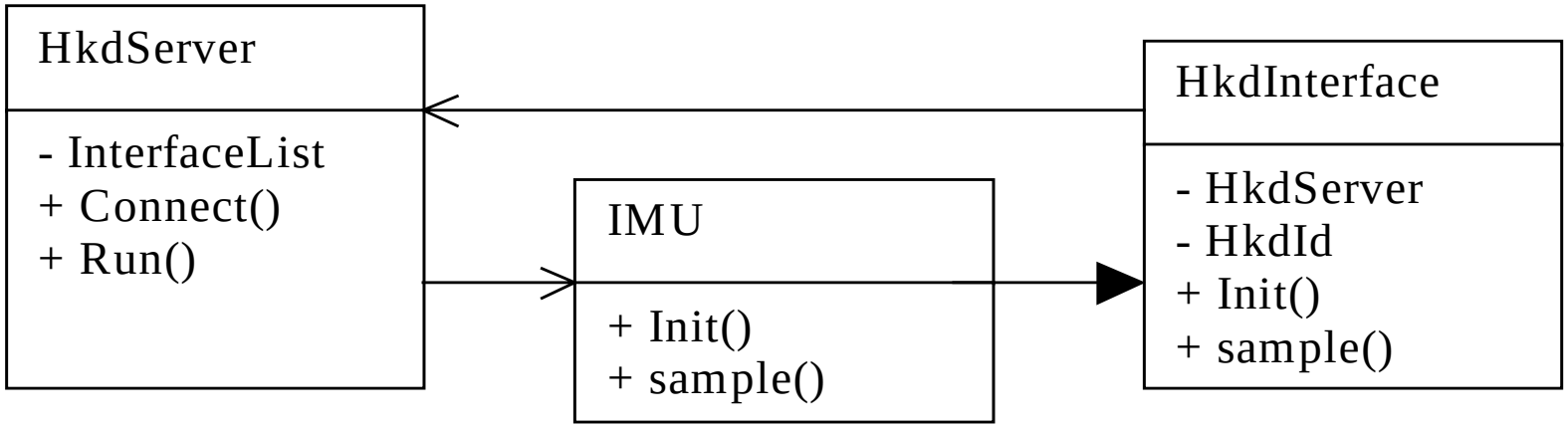
- Container-Klassen (Stacks, Queues, Vektoren, ...)
  - Nicht verwendbar, da keine statische Speicherverwaltung
- Numerische Bibliothek
  - Kann ohne Einschränkungen verwendet werden wenn Container-Klassen nicht dynamisch ihre Größe ändern und nur zum Systemstart initialisiert werden

Speichermanagement

- Statische Speicherstruktur verwenden
- Keine Verwendung der Schlüsselworte `new` und `delete`
- Für dynamische Zwecke stellt das Bird-Betriebssystem die Template-Klasse `pool` zur Verfügung

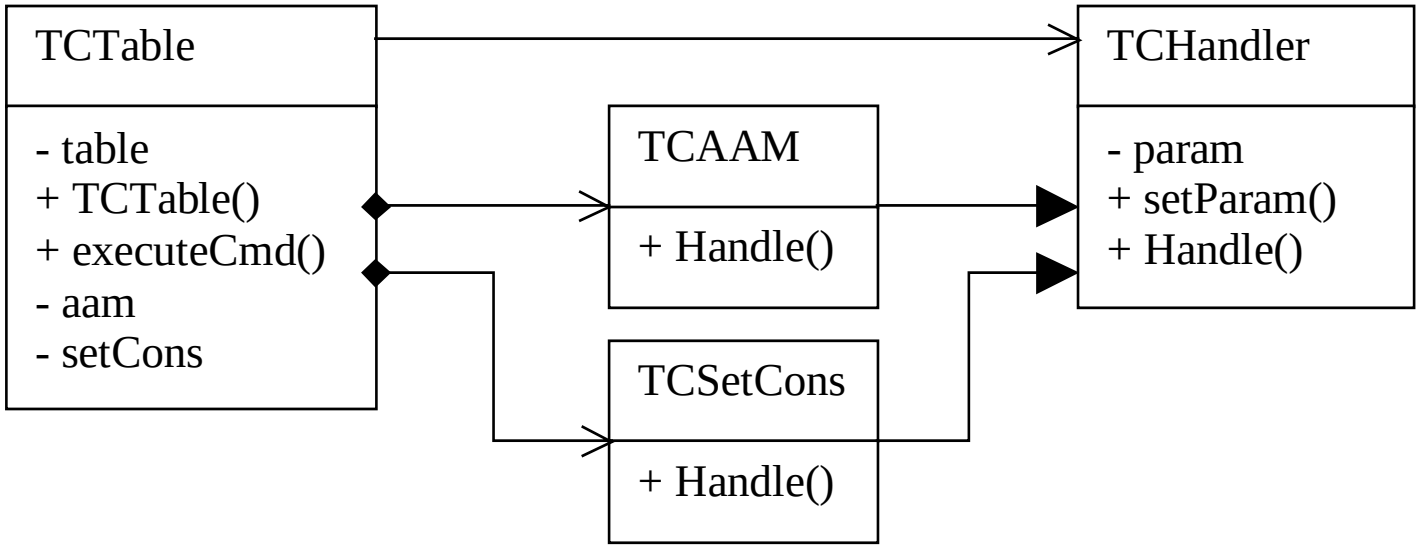
Einsatz abgeleiteter Klassen (1)

- **Ererbte Schnittstellen**
  - **Ermitteln von Housekeeping Daten**
  - **Überwachung des Zeitverhaltens**



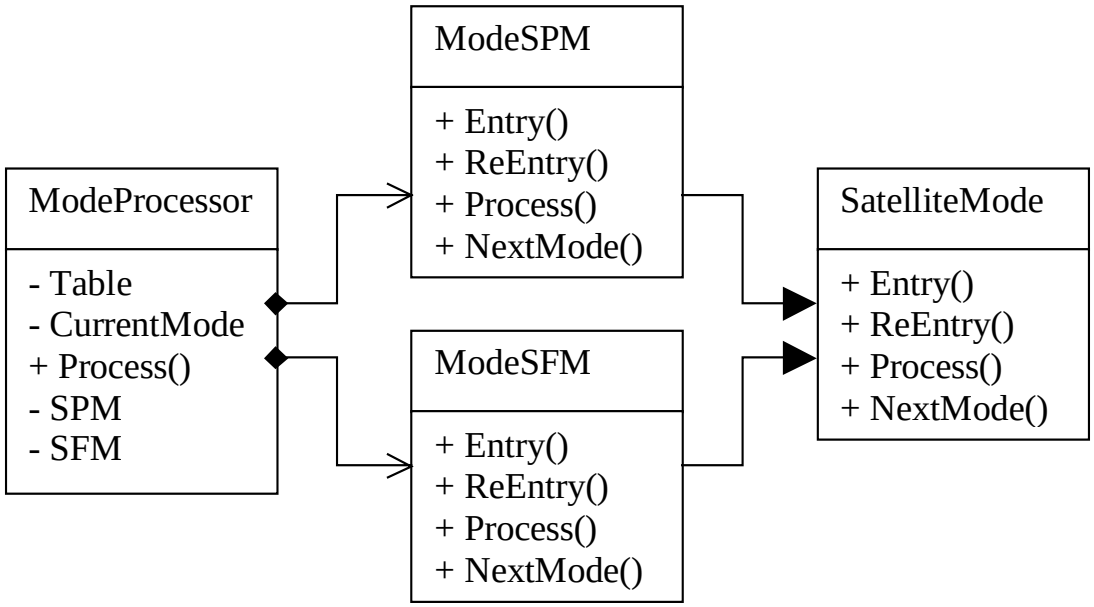
Einsatz abgeleiteter Klassen (2)

- **Sprungtabellen**
  - **Handlermodule von Telekommandos**



Einsatz abgeleiteter Klassen (3)

- **Betriebsmodi**
  - **Austausch der Lageregelungsalgorithmen je nach Betriebsmode**
  - **Zustandsmaschine für Sternenkamera**



Wiederverwendbarkeit

- Innerhalb des Bird Systems
  - Handhabung eines redundanten Bussystems
  - Handhabung von Sensorwerten aus A/D-Wandler
  - Basis-Struktur von Kommandos durch Vorgaben des Kommunikationsprotokolls
- Für spätere Projekte
  - Komplettsystem der Reaktionsrädern
  - Einzelne Sensoren oder Aktoren

Schlussbemerkungen

- Nur geringfügig zusätzlicher Bedarf an Rechenzeit und Speicherplatz
- Sauberes Design durch den Einsatz von objektorientierten Methoden
- Klare Trennung zwischen Systembestandteilen
- Identifizieren gemeinsamer Bestandteile in Systemkomponenten
- Saubere Schnittstellen
- Wiederverwendung von Systembestandteilen