Uhren und Synchronisation

Synchronisation bei fehlerbehafteten Uhren

Problemstellung

- Die bisherigen Algorithmen basierten alle auf der Annahme von fehlerfreien Uhren.
- Im Folgenden werden Algorithmen betrachtet, die mit einer maximalen Anzahl von m fehlerbehafteten Uhren umgehen können.
- Insgesamt soll das System aus n Uhren bestehen. Betrachtet werden im Besonderen auch byzantinische Fehler (die fehlerhafte Einheit kann beliebige Ausgaben produzieren).
- Die maximal zulässige Abweichung zweier Uhren bezeichnen wir mit ε.
- In Frage kommen dabei nur verteilte Algorithmen, um einen Single-Point-of-Failure auszuschließen.

Konvergenzalgorithmus (Leslie Lamport, 1985) [1]

- Algorithmus:
  - Jede Einheit liest die Uhr der anderen Rechner und berechnet den Mittelwert.
  - Ist die Abweichung einer Uhr größer als ε, so verwendet der Algorithmus stattdessen den Wert der eigenen Uhr.
- Aussage:
  - Der Algorithmus arbeitet erfolgreich, falls gilt: $n \geq 3m$.
- Annahmen:
  - vernachlässigbare Ausführungszeit
  - Einheiten lesen zeitgleich die Uhren ab bzw. Unterschiede sind vernachlässigbar

[1] Synchronizing Clocks in the Presence of Faults, Leslie Lamport and P.M. Melliar-Smith, SRI International, Menlo Park, California

Konvergenzalgorithmus (Leslie Lamport, 1985)

- Beweis:
  - Seien *p,q* zwei fehlerfreie Einheiten, *r* eine beliebige Einheit.
  - Sei *t(p,r)* die Uhrzeit von r, die die Einheit *p* für die Mittelwertsberechnung verwendet.

    → r fehlerfrei: $t(p,r) \approx t(q,r)$

    → r fehlerbehaftet $|t(p,r)-t(q,r)|<3\varepsilon$
  - Einheit p stellt seine Uhr auf: $1/n * \sum_r t(p,r)$
  - Einheit q stellt seine Uhr auf: $1/n * \sum_r t(q,r)$
  - Schlechtester Fall:
    - (n-m) Uhren fehlerfrei: $t(p,r) \approx t(q,r)$
    - m Uhren fehlerbehaftet $|t(p,r)-t(q,r)|<3\varepsilon$

  → Differenz beider Uhren: $\Delta(p,q)=1/n\ |\sum_r t(p,r)-\sum_r t(q,r)| \leq m/n*3\varepsilon < \varepsilon$

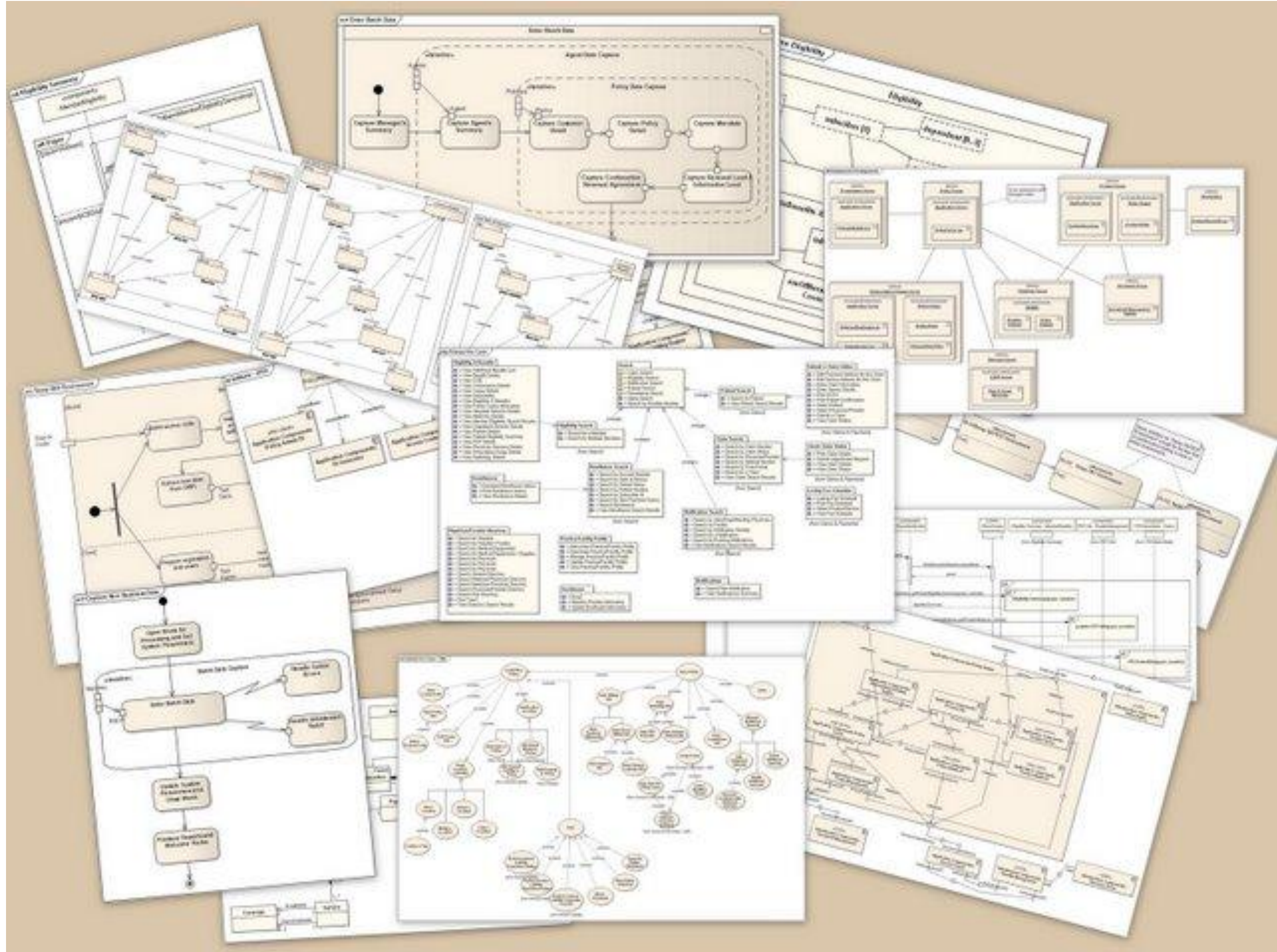
Erfolgskontrolle: Was sollten Sie aus dem Kapitel mitgenommen haben?

- Grundwissen über Zeit, Größenordnung der Uhrenabweichungen, typische Uhrenfehler und Notwendigkeit der Uhrensynchronisation
- Uhrensynchronisation:
  - Kenntnisse der verschiedenen Synchronisationstypen / -protokolle
  - Annahmen dieser Protokolle
  - Mechanismen zur Korrektur der eigenen Uhr
- Beispiel für eine Klausurfrage (Wiederholungsklausur 2006/2007):
  - Wieso kann bei der externen Synchronisation der maximal tolerierte Fehler als halb so groß, wie bei der internen Synchronisation gewählt werden?
  - **Antwort**: Bei der internen Synchronisation müssen gegensätzliche Fehler der involvierten Uhren angenommen werden (mindestens eine Uhr läuft zu schnell, mindestens eine Uhr zu langsam) – siehe auch Abbildung auf Folie 55.

Kapitel 3

Modellgetriebene Entwicklung von Echtzeitsystemen (inkl. Werkzeuge)

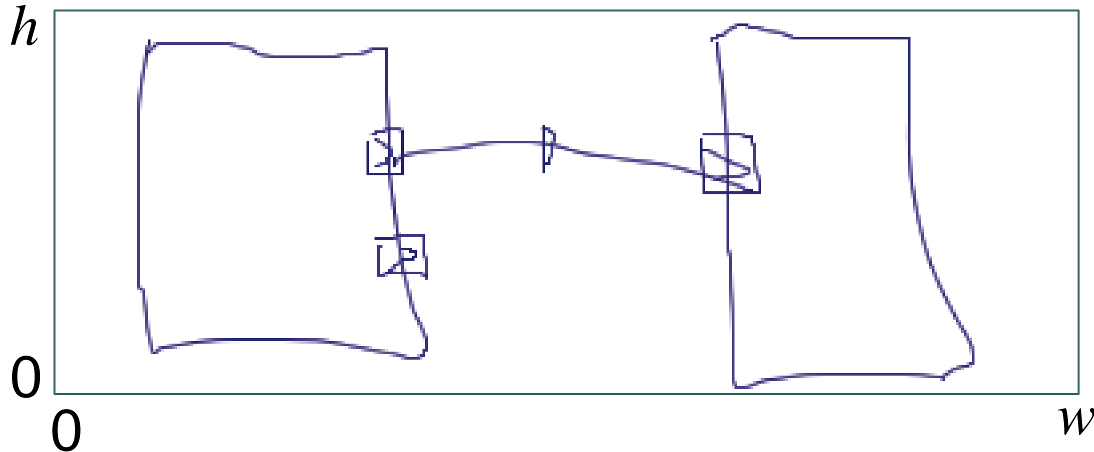
One way to build heterogeneous models: UML : Unified?



[Image from Wikipedia Commons. Author: Kishorekumar 62]

© Edward A. Lee, UC Berkeley, invited talk at EMSOFT 2011

The Truly Unified Modeling Language
TUML



Achtung Ironie!!!

A *model* in TUML is a function of the form

$$f:\ [0,w]\times[0,h]\to\{0,1\},\quad w,h\in\mathbb{N}$$

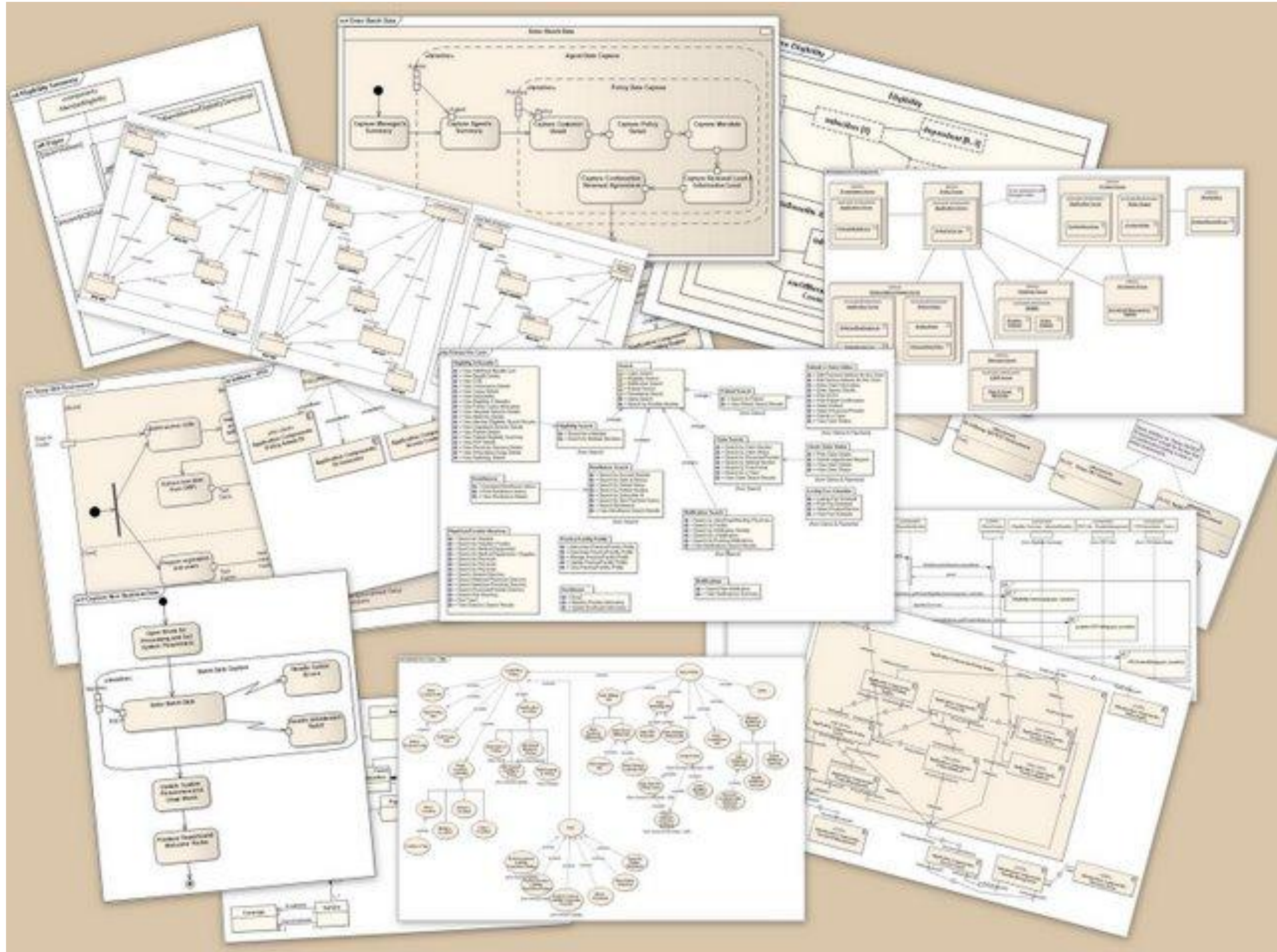
(notice how nicely formal the language is!)

Tools already exist.

With the mere addition of a *TUML profile*, every existing UML notation is a special case!

© Edward A. Lee, UC Berkeley, invited talk at EMSOFT 2011

Examples of TUML Models



[Image from Wikipedia Commons. Author: Kishorekumar 62]

© Edward A. Lee, UC Berkeley, invited talk at EMSOFT 2011

My Claim

Modeling languages that are not executable, or where the execution semantics is vague or undefined are not much better than TUML.

We can do better.

© Edward A. Lee, UC Berkeley, invited talk at EMSOFT 2011

Useful executable modeling languages impose *constraints* on the designer.

The constraints may come with benefits.

We have to stop thinking of constraints as a universal negative!!!

Freedom from choice!!!

© Edward A. Lee, UC Berkeley, invited talk at EMSOFT 2011

Inhalt

- Fokus: Konzepte und Werkzeuge zur Modellierung **und** Generierung von Code für Echtzeit- und eingebettete Systeme

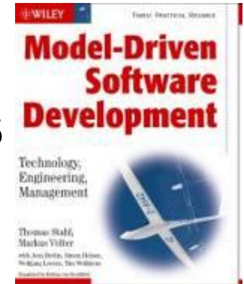
- Motivation
- Grundsätzlicher Aufbau, „Modelle“ und „Models of Computation“
  - Werkzeug Ptolemy
- Zeitgesteuerte Systeme
  - Werkzeug Giotto
- Synchrone Sprachen
  - Synchroner Datenfluss: EasyLab
  - Reaktive Systeme: Werkzeuge Esterel Studio

Fokus dieses Kapitels

- Voraussetzungen an Werkzeuge für ganzheitlichen Ansatz:
  - Explizite Modellierung des zeitlichen Verhaltens (z.B. Fristen)
  - Modellierung von parallelen Abläufen
  - Modellierung von Hardware und Software
  - Eindeutige Semantik der Modelle
  - Berücksichtigung von nicht-funktionalen Aspekten (z.B. Zeit*, Zuverlässigkeit, Sicherheit)
- Ansatz zur Realisierung:
  - Schaffung von domänenspezifischen Werkzeugen (Matlab/Simulink, Labview, SCADE werden überwiegend von spezifischen Entwicklergruppen benutzt)
  - Einfache Erweiterbarkeit der Codegeneratoren oder Verwendung von virtuellen Maschinen / Middleware-Ansätzen

* Zeit wird zumeist als nicht-funktionale Eigenschaft betrachtet, in Echtzeitsystemen ist Zeit jedoch als funktionale Eigenschaft anzusehen (siehe z.B. Edward Lee: Time is a Resource, and Other Stories, May 2008) http://chess.eecs.berkeley.edu/pubs/426/Lee_TimeIsNotAResource.pdf

Literatur



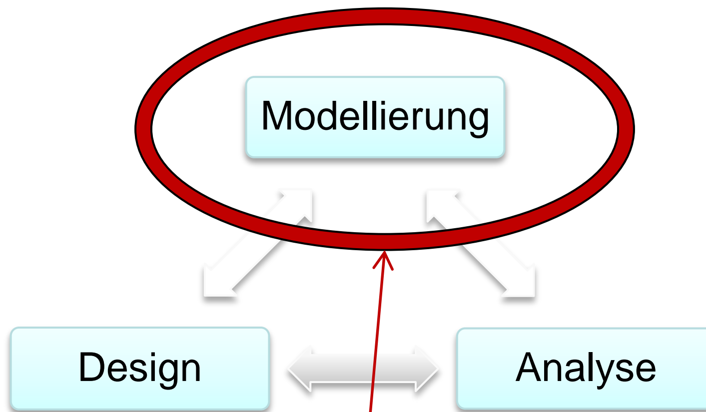
- Sastry et al: Scanning the issue – special issue on modeling and design of embedded software, Proceedings of the IEEE, vol.91, no.1, pp. 3-10, Jan 2003
- Thomas Stahl, Markus Völter: Model-Driven Software Development, Wiley, 2006
- Ptolemy: Software und Dokumentation http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm
- Benveniste et al.: The Synchronous Languages, 12 Years Later, Proceedings of the IEEE, vol.91, no.1, pp. 64-83, Jan 2003
- Diverse Texte zu Esterel, Lustre, Safe State Machines: http://www.esterel-technologies.com/technology/scientific-papers/
- David Harrel, Statecharts: A Visual Formalism For Complex Systems, 1987
- Henzinger et al.: Giotto: A time-triggered language für embedded programming, Proceedings of the IEEE, vol.91, no.1, pp. 84-99, Jan 2003

Hinweis: Veröffentlichungen von IEEE, Springer, ACM können Sie kostenfrei herunterladen, wenn Sie den Proxy der TUM Informatik benutzen (proxy.in.tum.de)

Begriff: Modell

- Brockhaus:
  Ein **Abbild** der Natur unter der Hervorhebung für **wesentlich** erachteter **Eigenschaften** und Außerachtlassen als nebensächlich angesehener Aspekte. Ein M. in diesem Sinne ist ein Mittel zur Beschreibung der erfahrenen Realität, zur Bildung von Begriffen der Wirklichkeit und Grundlage von Voraussagen über künftiges Verhalten des erfassten Erfahrungsbereichs. Es ist um so realistischer oder wirklichkeitsnäher, je konsistenter es den von ihm umfassten Erfahrungs-bereich zu deuten gestattet und je genauer seine Vorhersagen zutreffen; es ist um so **mächtiger**, je **größer** der von ihm beschriebene **Erfahrungsbereich** ist.
- Wikipedia:
  Von einem **Modell** spricht man oftmals als Gegenstand wissenschaftlicher Methodik und meint damit, dass eine zu untersuchende Realität durch bestimmte Erklärungsgrößen im Rahmen einer wissenschaftlich handhabbaren Theorie abgebildet wird.

Modellierung, Design und Analyse



Schwerpunkt dieses Kapitels:
-Wie sehen geeignete Modelle aus?
- Wie kann insbesondere die Implementierung abstrahiert werden?

- Modellierung
  - Modelle sind die Repräsentation des zu realisierenden Systems (evtl. Inklusive Umgebung) beschränkt auf die essentiellen Bestandteile
  - **Was** soll das System machen?
- Design:
  - Implementierung
  - Berücksichtigung der Hardware/Software-Aspekte
  - ***Wie** soll das System arbeiten?*
- Analyse:
  - Verstehen des Systems
  - ***Wieso** arbeitet das System so wie es arbeitet?*

Modellbasierte Entwicklung

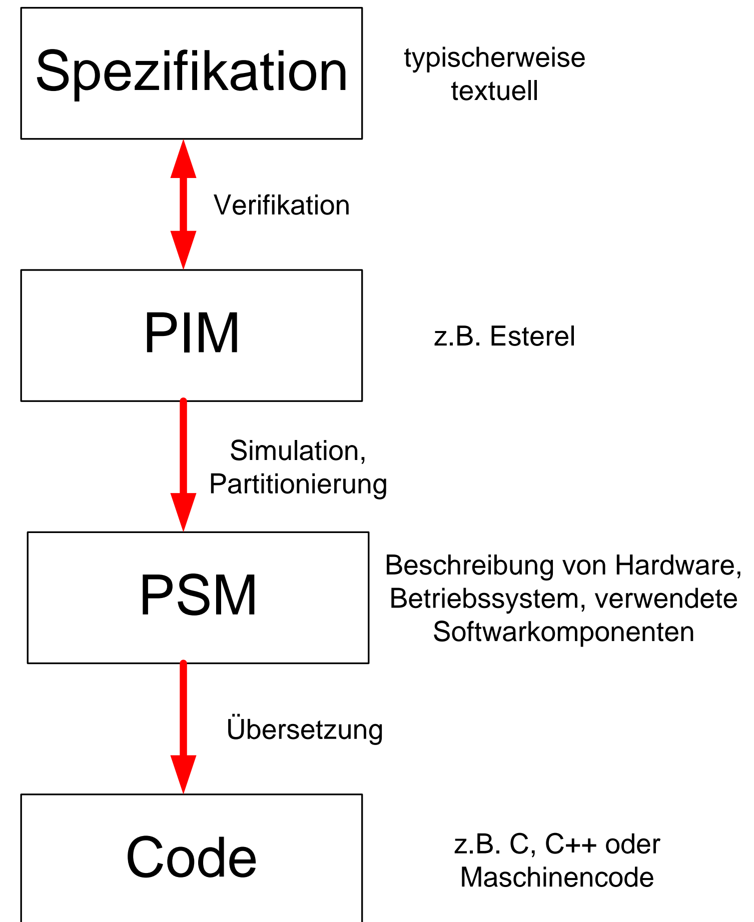
- Wir beobachten im Bereich eingebettete Systeme seit längerem einen Übergang von der klassischen Programmentwicklung zur einer Vorgehensweise, bei der Modelle (für physikalische Prozesse, für die Hardware eines Systems, für Verhalten eines Kommunikationsnetzes, usw.) eine zentrale Rolle spielen
- Modelle werden typischerweise durch verknüpfte grafische Notationselemente dargestellt (bzw. definiert)
- **Funktions-Modelle** sind dabei (meist Rechnerausführbare) Beschreibungen der zu realisierenden Algorithmen
- In unterschiedlichen Phasen der Systementwicklung kann ein Modell unterschiedliche Rollen annehmen – und etwa zum Funktionsdesign, zur Simulation, zur Codegenerierung verwendet werden
- Im günstigsten Fall kann ein System grafisch notiert („zusammengeklickt“) werden und unmittelbar Code erzeugt werden, der dann auf einem Zielsystem zur Ausführung gebracht wird
- Diese Entwicklung steht im Einklang mit der generellen Geschichte der Informatik, die immer mächtigere (Programmier-) Werkzeuge auf immer abstrakterem Niveau geschaffen hat

Vorteile Modellbasierter Entwicklung

- Für die modellbasierte Entwicklung sprechen diverse Gründe:
  - Modelle sind häufig einfacher zu verstehen als der Programmcode (graphische Darstellung, Erhöhung des Abstraktionslevels)
  - Vorwissen ist zum Verständnis der Modelle häufig nicht notwendig:
    - Experten unterschiedlicher Disziplinen können sich verständigen
  - Systeme können vorab simuliert werden. Hierdurch können Designentscheidungen vorab evaluiert werden und späte Systemänderungen minimiert werden.
  - Es existieren Werkzeuge um Code automatisch aus Modellen zu generieren:
    - Programmierung wird stark erleichtert
    - Ziel: umfassende Codegenerierung (Entwicklung konzentriert sich ausschließlich auf Modelle)
  - Mittels formaler Methoden kann
    - die Umsetzung der Modelle in Code getestet werden
    - das Modell auf gewisse Eigenschaften hin überprüft werden

Beispiel OMG: Model-Driven Architecture (MDA)

- Die Entwicklung des Systems erfolgt in diversen Schritten:
  - textuelle Spezifikation
  - PIM: platform independent model
  - PSM: platform specific model
  - Code: Maschinencode bzw. Quellcode
- Aus der Spezifikation erstellt der Entwickler das plattformunabhängige Modell
- Hoffung: weitgehende Automatisierung der Transformationen PIM → PSM → Code (Entwickler muss nur noch notwendige Informationen in Bezug auf die Plattform geben)
- http://www.omg.org/mda



MDA im Kontext von Echtzeitsystemen

- In Echtzeitsystemen / eingebetteten Systemen ist bei einem umfassenden Ansatz ein Hardwaremodell (z.B. Rechner im verteilten System, Topologie) schon in frühen Phasen (PIM) notwendig
- Das plattformspezifische Modell (PSM) erweitert das Hardware- & Softwaremodell um Implementierungskonzepte, z.B.
  - Implementierung als Funktion/Thread/Prozess
  - Prozesssynchronisation