Folien zum Lehrmodul

Einführung in die modellgetriebene Software-Entwicklung

Lernziele:

- Prinzipien der modellgetriebene Software-Entwicklung verstehen
- Einsatzmöglichkeiten und -grenzen verstehen
- wesentliche Produkte / Standards einordnen können

Inhaltsverzeichnis

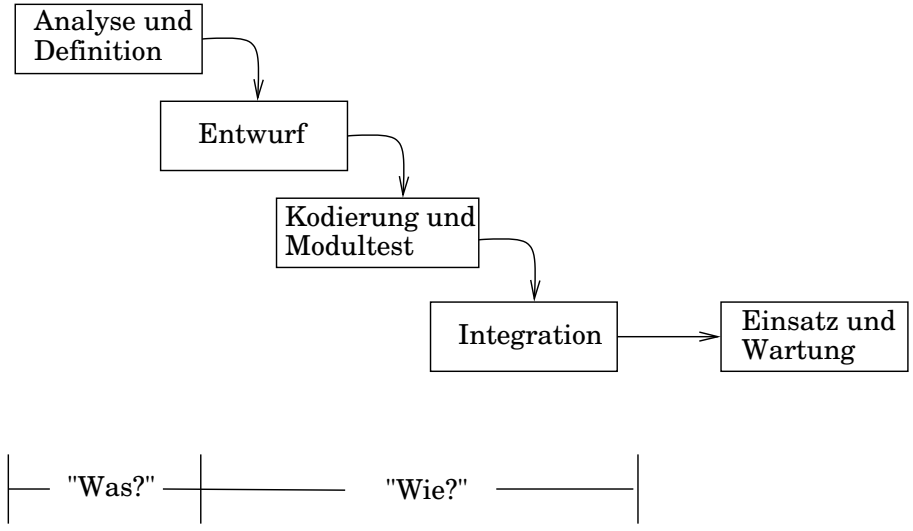
1 Motivation

Modellgetriebene Software-Entwicklung
(*model driven (software) development*, MDD)

- wird angepriesen als wichtiger zukünftiger Trend der Softwareentwicklung
- allgemeinste Definition: Einsatz von Modellen zu Beschleunigung / Effizienzverbesserung / Qualitätsverbesserung der Software-Entwicklung

2 Grundlegende Ansätze

Voraussetzung: MBSE unterstellt eine stufenweise Konkretisierung eines System (im Phasenmodell 1*, beim evolutionären Vorgehen vielfach durchlaufen):



1. Analyse-Modell
2. Entwurfsmodell, das zumindest Teile des zu entwickelnden Systems *präzise* beschreibt
3. Programmcode

Idee: Codieraufwand reduzieren, indem man

a. möglichst große Teile des Programmcodes **durch Übersetzen des Modells generiert**

   oder

b. ein generisches Programm realisiert, das ein konkretes **Modell interpretiert**

2.1 Übersetzungsansatz

Annahme: Gesamtsystem besteht aus:

1. generierten Teilen
2. von Hand realisierten Teilen
3. konstanten Teilen (für eine bestimmte Applikationsdomäne)

Übersetzungsstrategie:

- in mehreren Schritten übersetzen
- jeweils nur eine Technologieentscheidung umsetzen

erforderliche MBSE- generiert Infrastruktur (über die ohnehin notwendigen Editoren, Prüfwerkzeuge usw. für Modelle hinaus):

- diverse Übersetzer für Modelle
- ggf. Bibliotheken / Standardpakete

2.2 Interpreteransatz

- erfordert einen Interpreter,
  - der die Modelle einlesen und "verstehen" kann und
  - die generische Programmfunktionalität definiert (die sonst der "restliche Programmcode" definieren würde)

    = "Meta-Applikation"

  Beispiel:

  ein generischer Editor für Graphen, deren Knoten, Kanten und Attribute durch ein Klassendiagramm spezifiziert sind.

- Eingabedaten des Interpreters:
  1. ein Modell (ggf. mehrere)
  2. dazu passende Nutzdaten
- Interpreteransatz ist für Nutzer des Interpreters einfacher handhabbar, aber weniger flexibel als der Übersetzungsansatz

2.3 Einsatzkriterien

Beide Ansätze sind aufwendiger (!) als die direkte Implementierung eines Systems

Wann lohnt sich die modellgetriebene Software-Entwicklung?

1. wenn sich *Implementierungstechnologien häufig ändern* und dann umfangreiche Anpassungsarbeiten anfallen
2. wenn die gleiche Applikation für *mehrere Zielplattformen* realisiert werden mus ("*model once, run anywhere*");
   Kernidee des MDA-Ansatzes der OMG; vgl. CORBA

3. wenn eine *große Zahl ähnlicher Systeme* erstellt werden muß (z.B. Systemfamilie)

   gemeinsame Funktionalität = generische Programmfunktionalität des Interpreters bzw. “restlicher Programmcode”, der ggf. als Bibliothek gehandhabt wird

4. wenn beim evolutionären Vorgehen *viele Varianten* eines Systems erstellt werden

5. wenn man auf diese Weise billig einen *schnellen Prototypen* realisieren kann

6. wenn verschiedene *Personen verschiedene Kenntnisse beitragen*:
   - Kenntnis der konkreten *Applikation* (ggf. Sachbearbeiter) → Modelle
   - Kenntnis der *Applikationsdomäne* und der *Implementierungstechnologien* (Informatiker) → Gestaltung technischer Details, Implementierung gemeinsamer Funktionalität

   ∼ "aspektorientierte" Arbeitsteilung

2.4 Modelltransformationen und MDA

Idee: Modell nicht in einem Schritt in Code zu übersetzen, sondern *in mehreren Schritten Zwischenstufen erzeugen*

Modell-Sequenz gemäß OMG MDA-Ansatz[1]:

1. **Computation Independent Model** (CIM) - sehr abstrakt, z.B. Use-Case-Diagramme, Glossar (übersetzbar???)
2. **Platform Independent Model** (PIM) - abstrahiert von gängigen Plattformen
3. **Platform Specific Model** (PSM) - plattformabhängiges Modell; Plattform = Programmiersprache, Protokolle, weitere Basistechnologien

[1] s. MDA Guide Version 1.0.1, `http://www.omg.org/docs/omg/03-06-01.pdf`

Thema unsaubere Begriffe und deren Folgen:

- MDA = Modellgetriebene Architektur – können sich Architekturen überhaupt bewegen???

  wenn überhaupt, ist ein Interpreter modellgetrieben oder modell*basiert* oder modell*gesteuert*

- MDA Guide, 2.1.1 Background: " **Model Driven Architecture** (TM) or **MDA** ... is not, like the OMA and CORBA, a framework for implementing distributed systems. It is *an approach to using models in software development.* "

- MDA Guide, 2.2.3 Model-Driven: “MDA is an approach to system development, which increases the power of models in that work. It is model-driven because it provides *a means for using models* to direct the course of understanding, design, construction, deployment, operation, maintenance and modification”

- MDA Guide, 2.2.4 Architecture: “The architecture of a system *is a specification of the parts* and connectors of the system and the rules for the interactions of the parts using the connectors.

- was nun?? *...is a specification of the parts ...* oder *... is an approach to using ...*

- Note: Mangelhaft.
  → notorische Konfusion, worin der Unterschied zwischen MDA und MBSE besteht (kein wesentlicher)

Abgesehen von der völlig mißratenen Bezeichnung MDA ist der MDA Guide sehr lesenswert!
("Model Driven [Software] Development" oder "MDD" kommt auf den 62 Seiten *nicht* vor)

2.4.1 MDA Guide revision 2.0

nur noch 15 Seiten!

`http://www.omg.org/mda/`

`http://www.omg.org/cgi-bin/doc?ormsc/14-06-01`

`http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf`

3 MBSE-Infrastrukturen

3.1 Entwicklung einer MBSE-Infrastruktur

1. Basis: Kenntnis der *Applikationsdomäne* (oder *Systemfamilie*) und der Gemeinsamkeiten der Einzelprodukte
2. Implementierung vorhandener Einzelprodukte analysieren und folgende Codeteile unterscheiden:
   a. in allen Einzelprodukten *identisch* oder fast identisch auftretende Codeteile
   b. aus Modellen *ableitbare* Codeteile ("schematische" Codeteile, Pattern-Instanzen)
   c. *individueller* Code

4. Ableitung einer Standardarchitektur, Identifizierung von Standardmodulen (Bibiotheken), Konfigurationsparametern

5. Technologieauswahl der MBSE-Infrastruktur:
   - Modellierungssprache (Metamodell, konkrete Syntax)
   - Modellübersetzer(technologie)
   - ggf. Auswahl von Zielplattformen

6. Portierung eines Einzelprodukts in die MBSE-Infrastruktur, Evaluierung

3.2 Modelltransformatoren und -Übersetzer

Modelltransformator: gibt wieder Modell aus (verfeinertes Modell)

Modellübersetzer: gibt Programm-Quelltext aus

arbeiten alle auf Repräsentationen der Modelle (z.B. als XML-Datei)
→ unterstellen / benutzen konzeptuell ein Dokumentschema (= Metamodell)

Einzelwerkzeuge für einen speziellen Modelltyp:

- haben i.d.R. Dokumentschema "hart verdrahtet"
- Bsp: viele Übersetzer

Meta-Werkzeuge, die für mehrere ähnliche Modelltypen ähnliche Funktionen anbieten:

- Grundidee der modellgetriebenen Software-Entwicklung auf die Werkzeuge anwenden!
- Interpreter- oder Übersetzeransatz wählen .....
- brauchen Repräsentationen der Metamodelle
  $\rightarrow$ unterstellen / benutzen konzeptuell ein Dokumentschema der Metamodelle (= Meta-Metamodell)

3.3 Beispiele für MBSE-Infrastrukturen

Eclipse Modeling Framework (EMF)

- ist vor allem ein *Übersetzer*, der Modelle (und zwar Entwurfs-Klassendiagramme!!) in Java-Code übersetzt; der Java-Code deckt folgende Funktionen ab:
  - Instanzen dieses Modells erstellen, abfragen, manipulieren, serialisieren, validieren und auf Änderungen überwachen

    = große Teile eines Editors für Instanzen dieses Modells, insb. die Datenhaltungsschicht
  - JUnit-Code u.a.

- hat nur rudimentäre Möglichkeiten zur Erstellung von Modellen (Baumeditor)

  Modellerstellung typischerweise durch andere Werkzeuge (die Bezeichnung Modeling Framework ist insofern irreführend)

- zu übersetzendes Modell muß als Menge von Ecore-Objekten "implementiert" vorliegen

  Ecore basiert auf dem EMOF-Standard (Essential Meta-Object Facility).

- Dokumentation und Tutorials s.: The Eclipse Modeling Framework (EMF); `http://www.eclipse.org/modeling/emf/`