

Modellgetriebene Softwareentwicklung

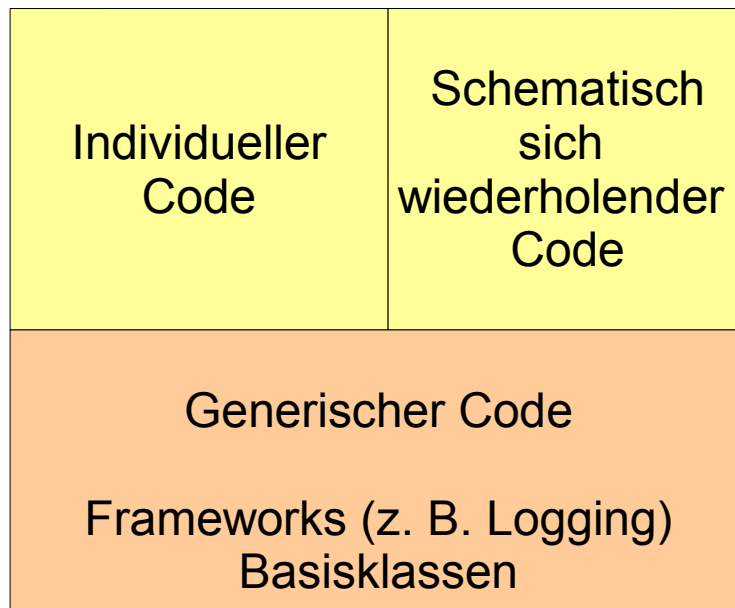
Model driven software development (MDSD)

Gliederung

- Motivation, Konzepte
- Einordnung Faktor-IPS
- Subdomänen & Partitionierung
- Variante: Modellinterpretation
- Codegenerierung & Konfigurationsmanagement
- Modelltransformationen

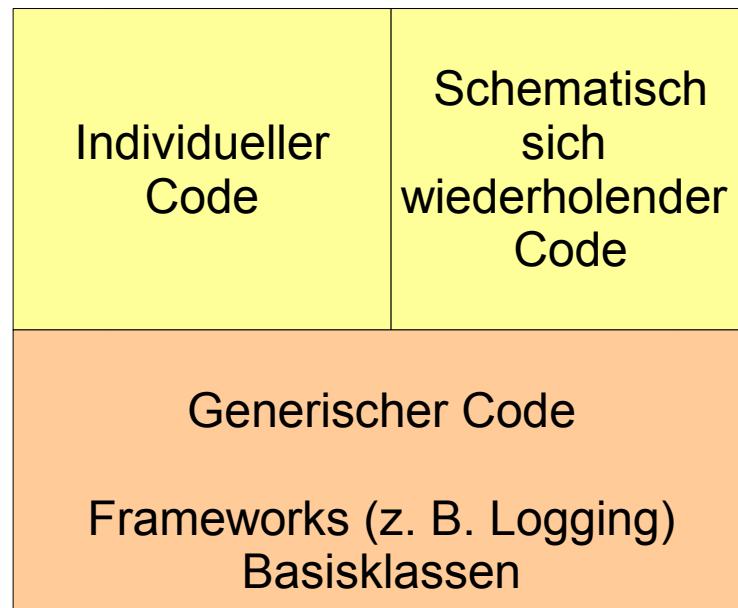
Modellgetriebene Softwareentwicklung: Motivation

Referenzanwendung



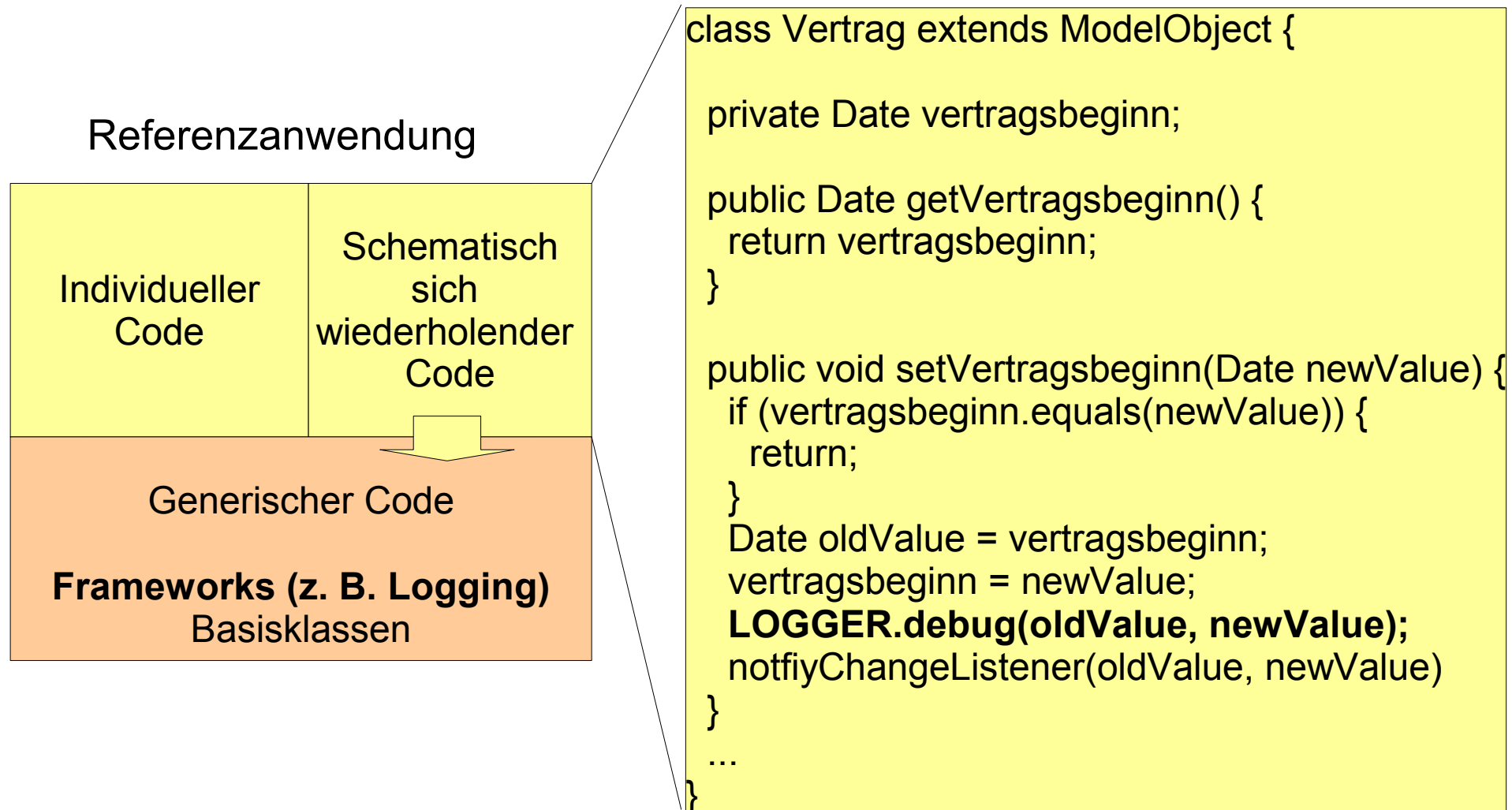
Modellgetriebene Softwareentwicklung: Motivation

Referenzanwendung



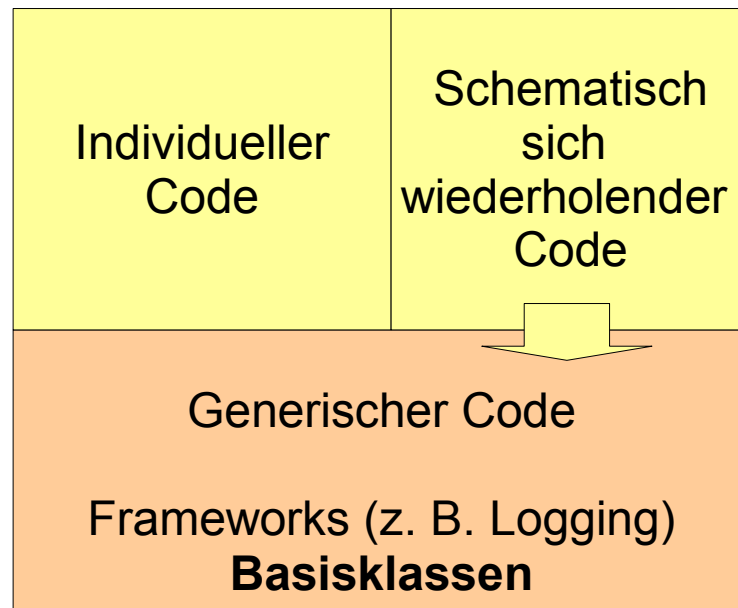
```
class Vertrag extends ModelObject {  
  
    private Date vertragsbeginn;  
  
    public Date getVertragsbeginn() {  
        return vertragsbeginn;  
    }  
  
    public void setVertragsbeginn(Date newValue) {  
        if (vertragsbeginn.equals(newValue)) {  
            return;  
        }  
        Date oldValue = vertragsbeginn;  
        vertragsbeginn = newValue;  
        LOGGER.debug(oldValue, newValue);  
        notifyChangeListener(oldValue, newValue)  
    }  
  
    ...  
}
```

Modellgetriebene Softwareentwicklung: Motivation



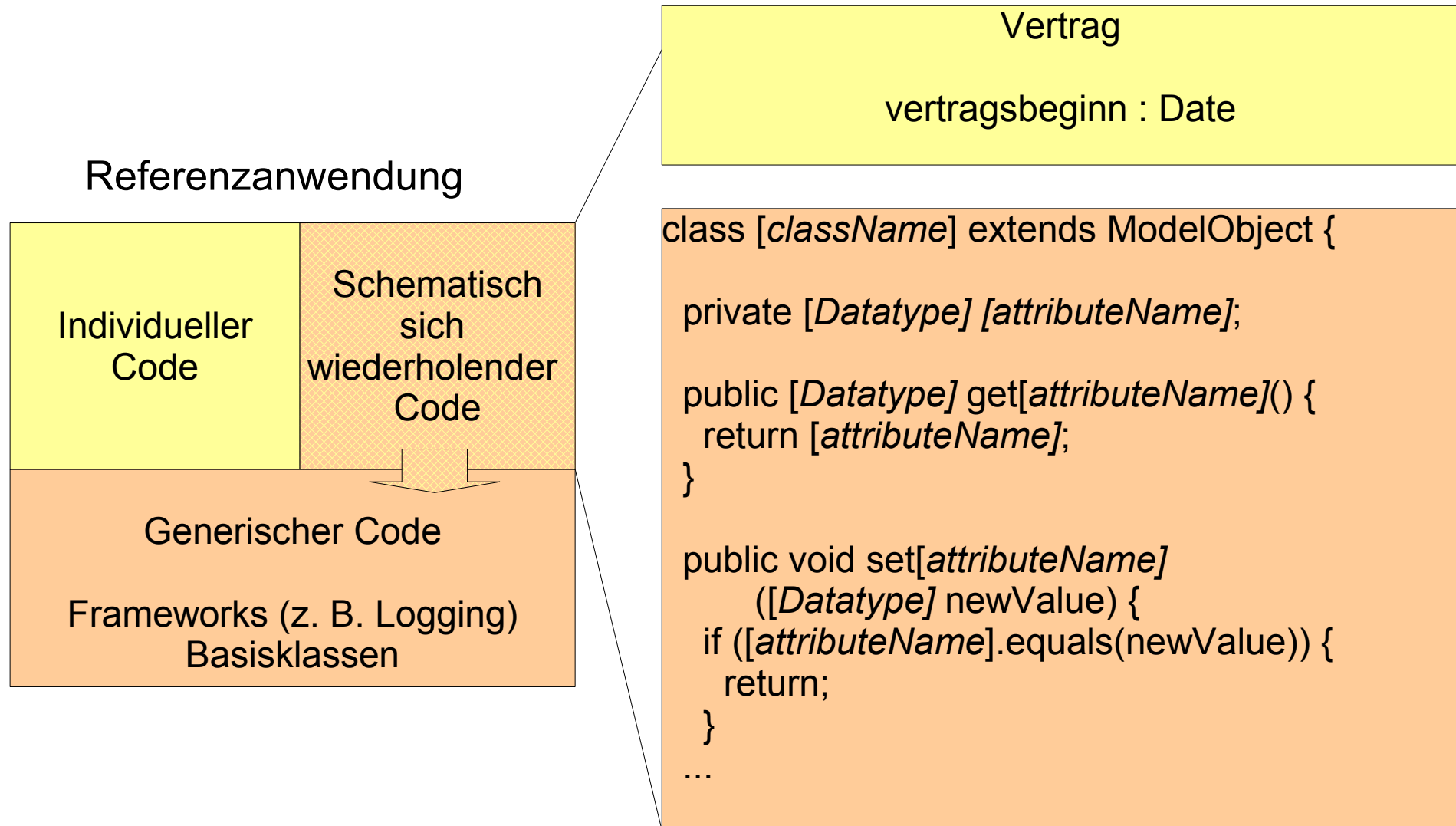
Modellgetriebene Softwareentwicklung: Motivation

Referenzanwendung

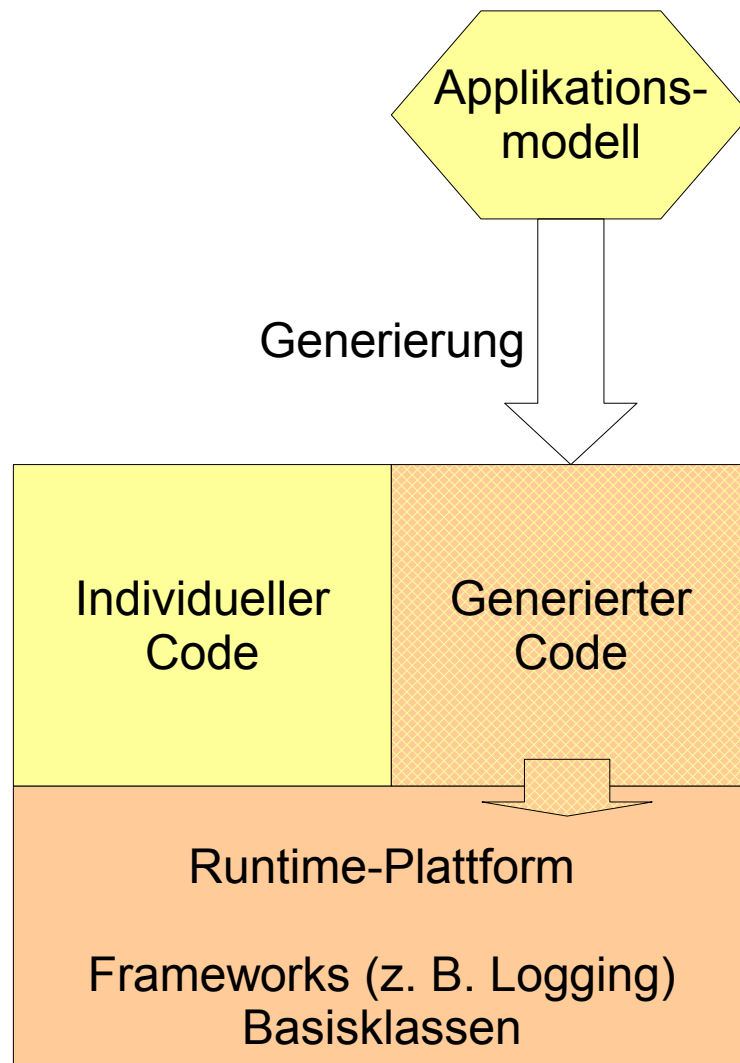


```
class Vertrag extends ModelObject {  
  
    private Date vertragsbeginn;  
  
    public Date getVertragsbeginn() {  
        return vertragsbeginn;  
    }  
  
    public void setVertragsbeginn(Date newValue) {  
        if (vertragsbeginn.equals(newValue)) {  
            return;  
        }  
        Date oldValue = vertragsbeginn;  
        vertragsbeginn = newValue;  
        LOGGER.debug(oldValue, newValue);  
        notifyChangeListener(oldValue, newValue)  
    }  
  
    ...  
}
```

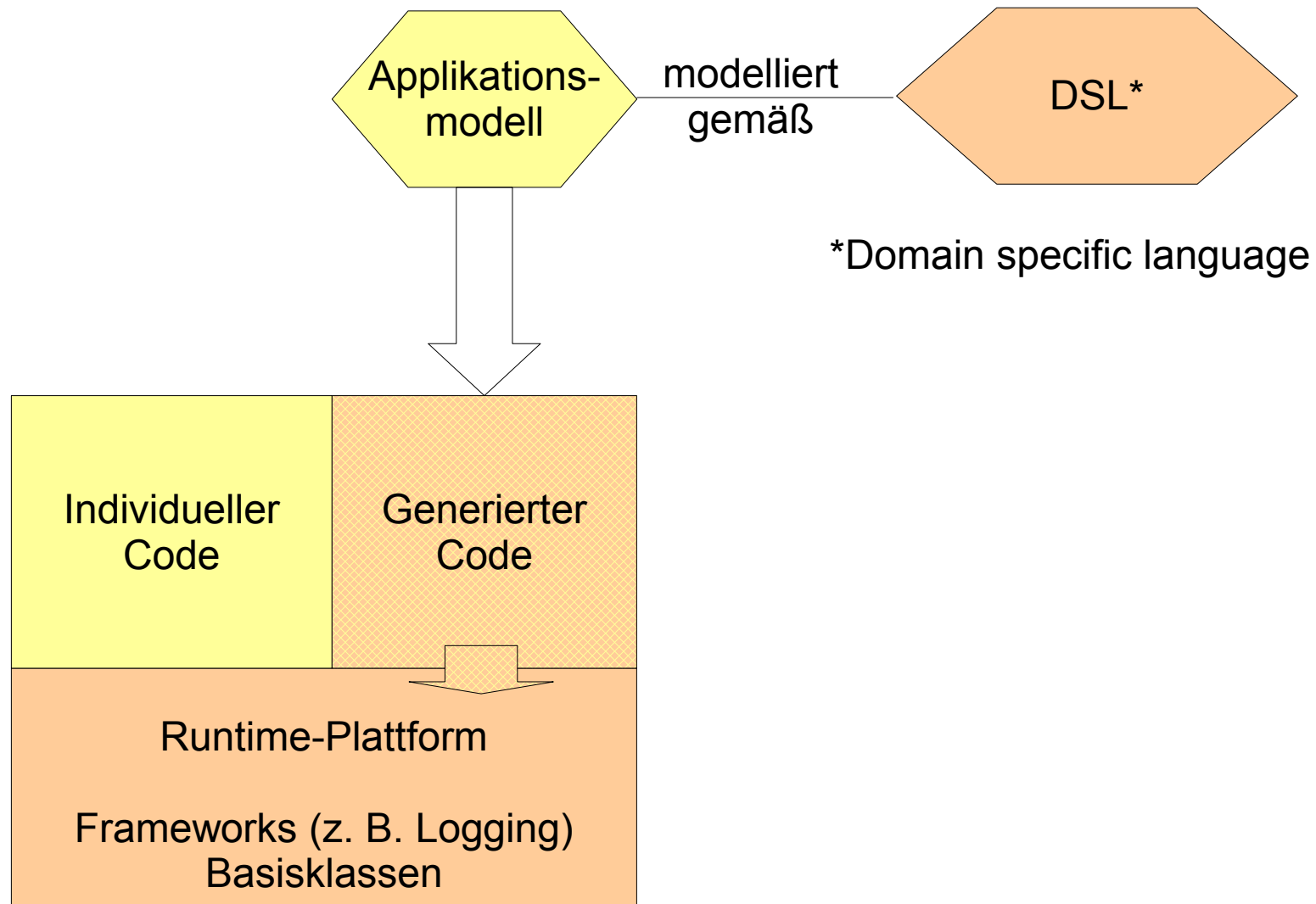
Modellgetriebene Softwareentwicklung: Motivation



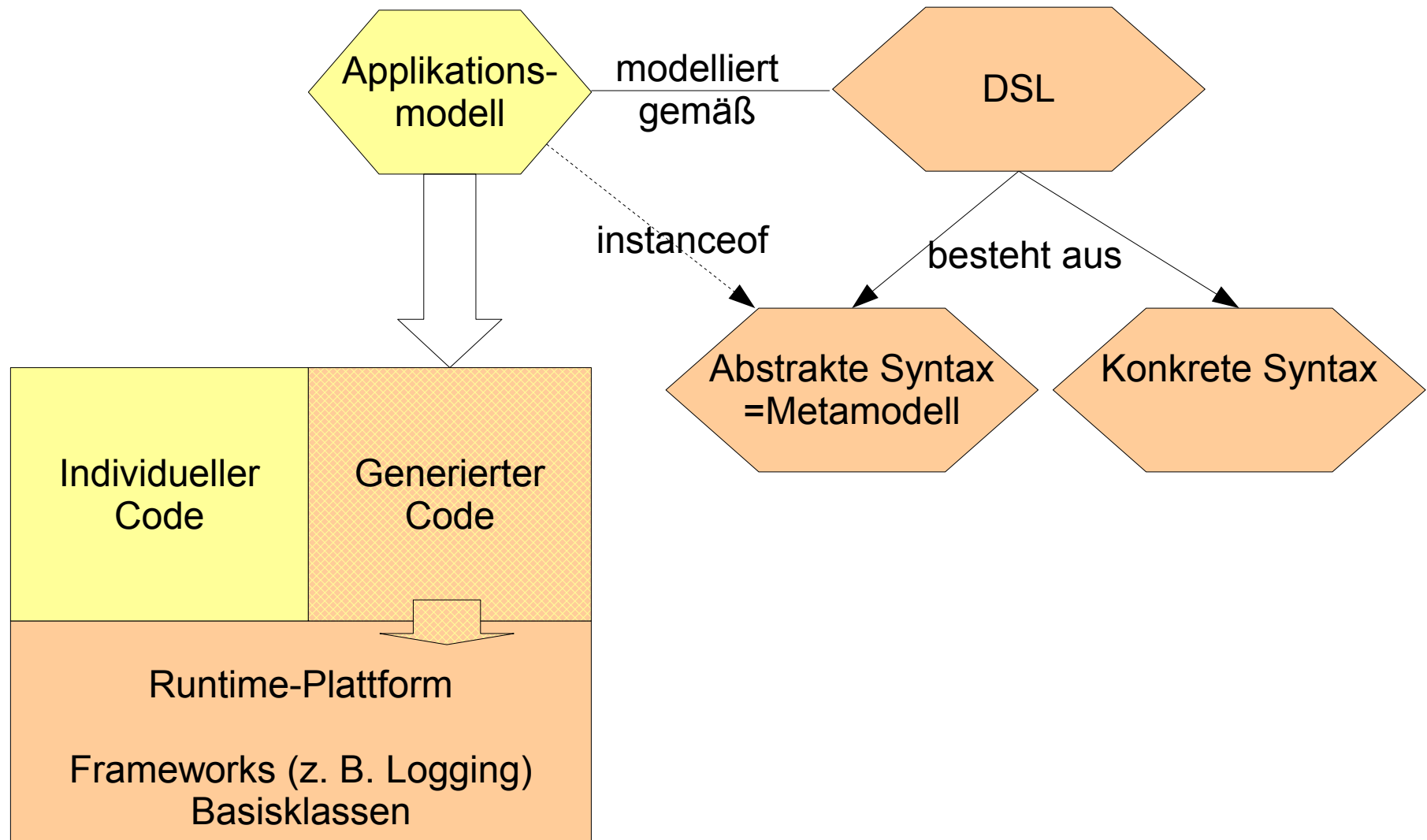
Modellgetriebene Softwareentwicklung: Motivation



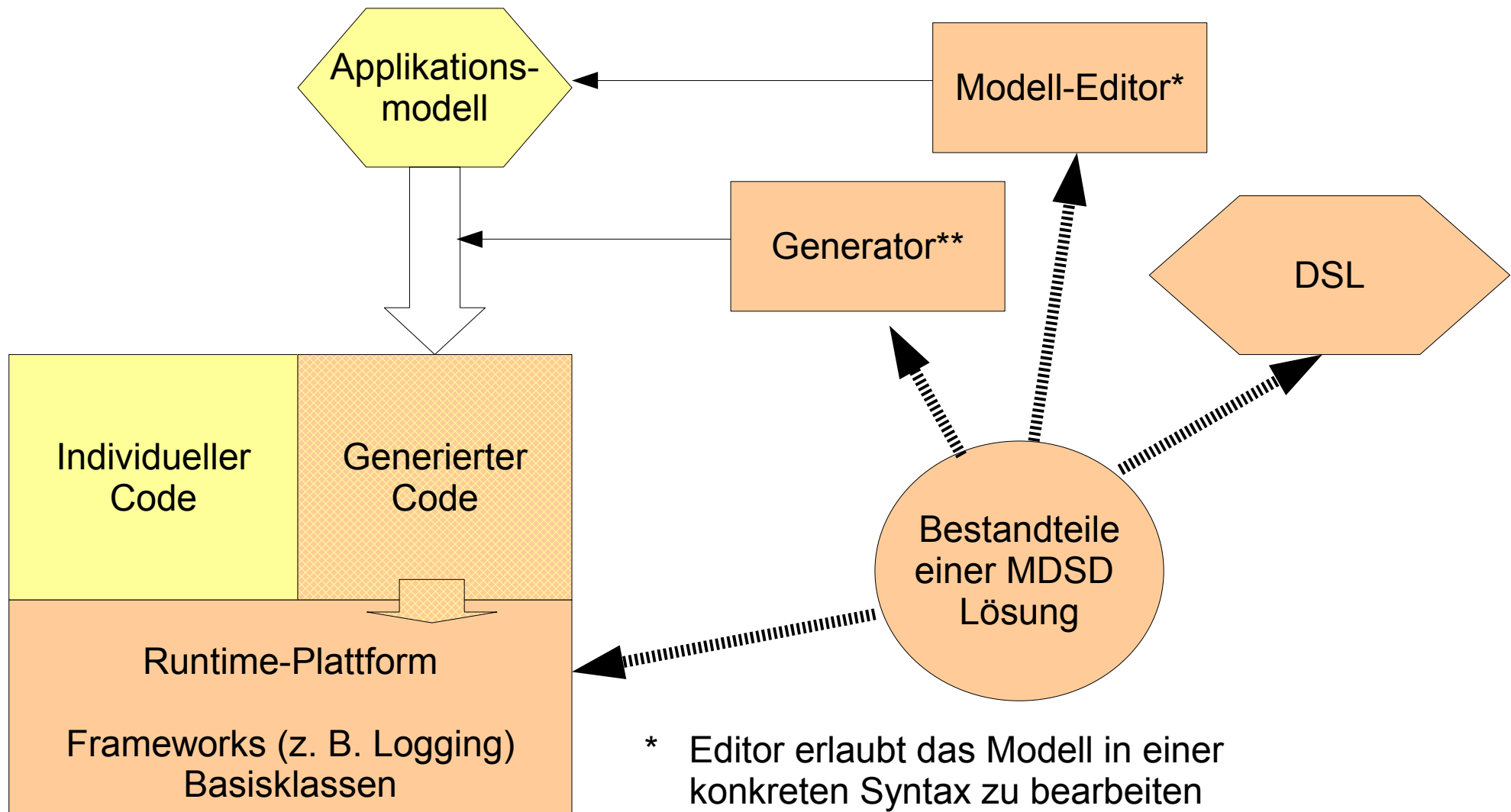
Domain specific language



Domain specific language



Bestandteile einer MDSD Lösung

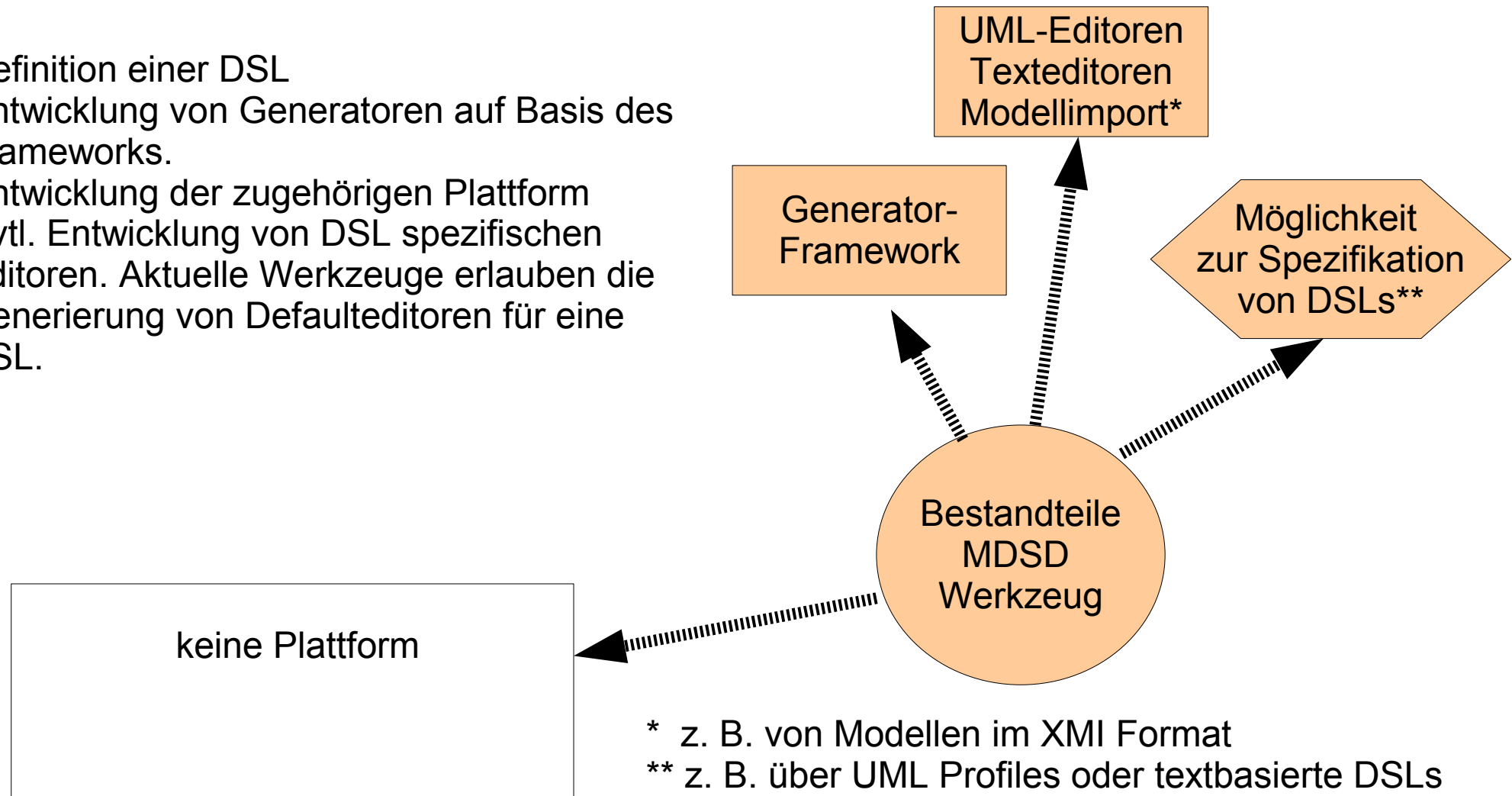


* Editor erlaubt das Modell in einer konkreten Syntax zu bearbeiten

** Generator arbeitet auf dem Metamodell

Entwicklung einer MDSD Lösung mit generellen MDSD Werkzeugen

- Definition einer DSL
- Entwicklung von Generatoren auf Basis des Frameworks.
- Entwicklung der zugehörigen Plattform
- Evtl. Entwicklung von DSL spezifischen Editoren. Aktuelle Werkzeuge erlauben die Generierung von Defaulteditoren für eine DSL.



Vorteile

- Höhere Qualität
 - Komplexität läßt sich auf Modellebene besser beherrschen, da auf einer höheren Abstraktionsebene gearbeitet wird.
 - Grundstruktur des Sourcecodes wird durch Modell & Generator bestimmt. Manuelle Teile werden in diese Struktur eingeordnet.
 - Keine Qualitätsschwankungen im generierten Sourcecode
 - Test des generierten Sourcecodes kann auf den Test des Generators reduziert werden.
- Höhere Produktivität
 - Deutliche Reduktion des Umfangs des individuell zu entwickelnden Codes.
 - Höhere Codequalität senkt Aufwand für Nachbesserungen und Regressionstest.
 - Designänderungen lassen sich (teilweise) durch Anpassung des Generators umsetzen.

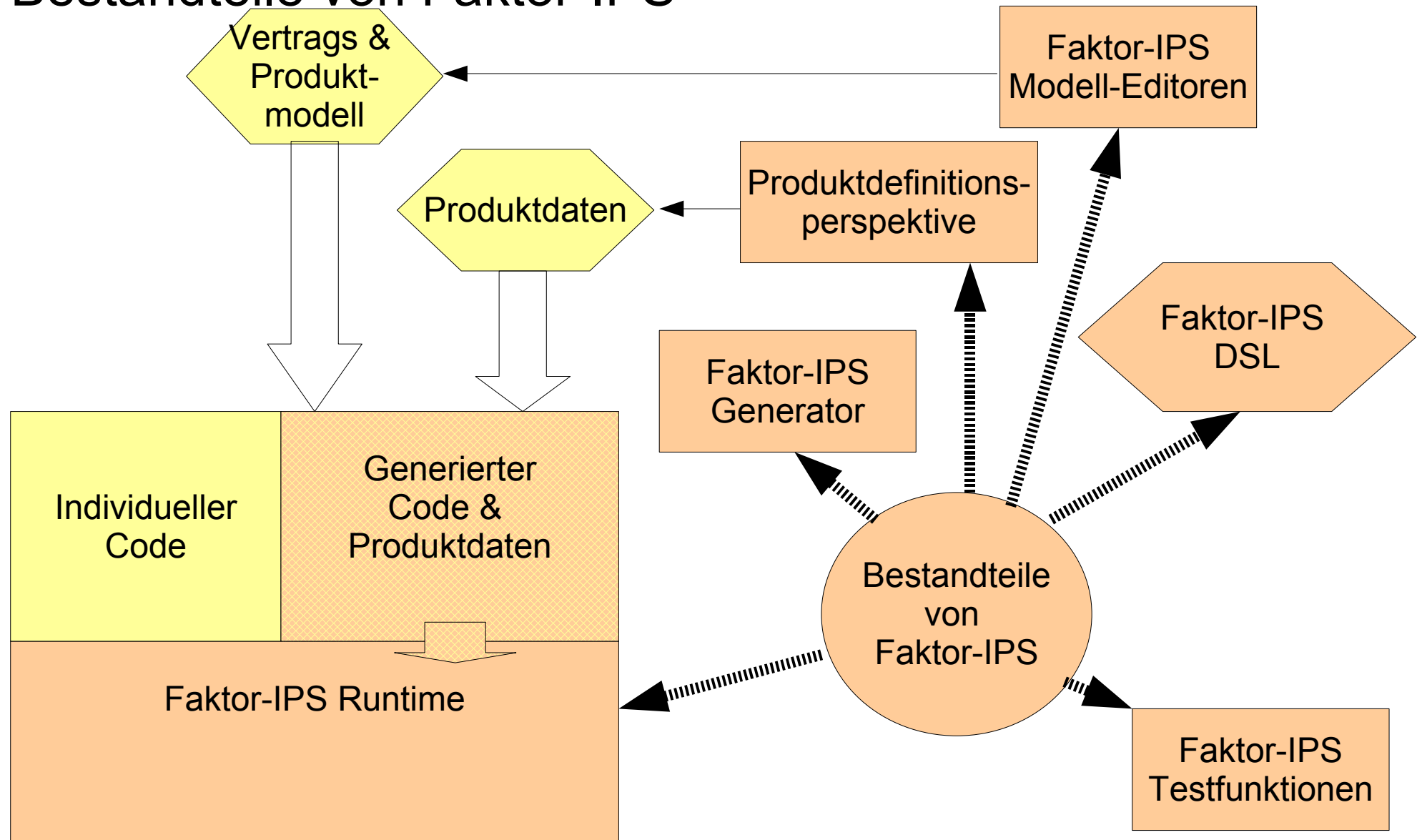
Erfolgsvoraussetzung

- Kosten für Entwicklung der MDSD Lösung darf Produktivitätsgewinn nicht übersteigen
- DSL & Generator
 - Entwicklung ist bei nicht-trivialen Modellen i.d.R. günstiger im Vergleich zur manuellen Entwicklung des Codes.
- Modelleditor
 - Vermeidung von Neuentwicklung, statt dessen Verwendung bestehender Werkzeuge (z. B. UML-Werkzeuge oder Faktor-IPS).
- Plattform
 - Wird auch ohne MDSD im Projekt entwickelt. Also kein Zusatzaufwand.
- Bau von projektbezogener MDSD Lösung darf nicht zum Selbstzweck werden.

Gliederung

- Motivation, Konzepte
- Einordnung Faktor-IPS
- Subdomänen & Partitionierung
- Variante: Modellinterpretation
- Codegenerierung & Konfigurationsmanagement
- Modelltransformationen

Bestandteile von Faktor-IPS



Faktor-IPS: DSL

- Geschäftsobjekte Vertrag & Produkt
 - Teilmenge von UML Klassenmodellierung
 - Fachlicher Zusammenhang Vertrag / Produkt
- Produktbausteine
 - inkl. Abbildung Produktänderungen im Zeitablauf
- Formelsprache
- Tabellen
 - inkl. Zusammenhang zu Produktklassen / Bausteinen

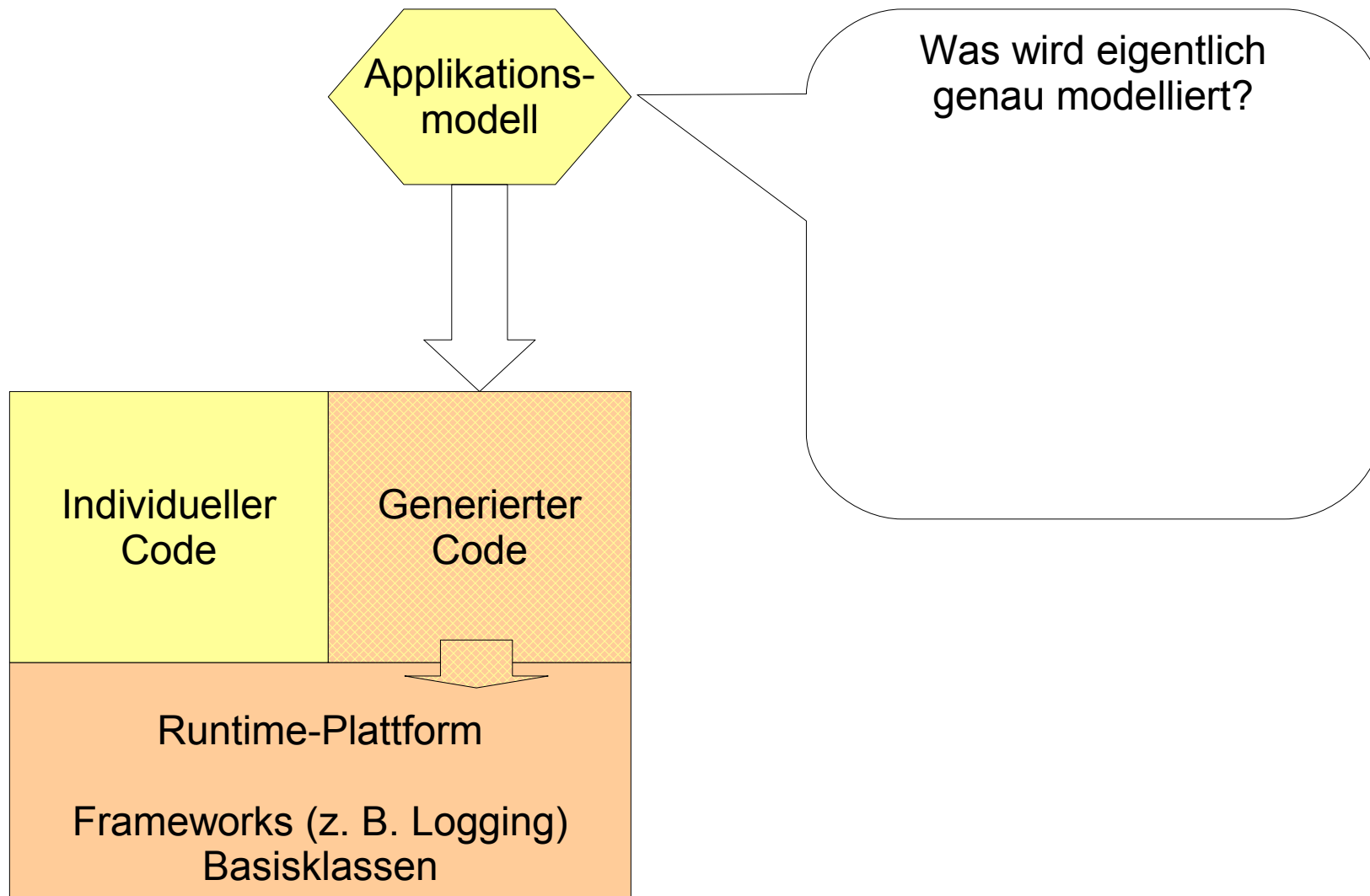
Faktor-IPS Runtime

- Effizienter Zugriff auf Produktdaten über ein Repository
- Ablauffähig auf allen Plattformen auf denen eine JVM > 1.4 zur Verfügung steht.
 - Windows
 - Unix, Linux
 - OS / 390, Z/OS
- Leicht integrierbar
 - in JEE Architekturen (EJBs)
 - in JSE Architekturen
 - über Webservice
 - über JNI
- Keine Abhängigkeit

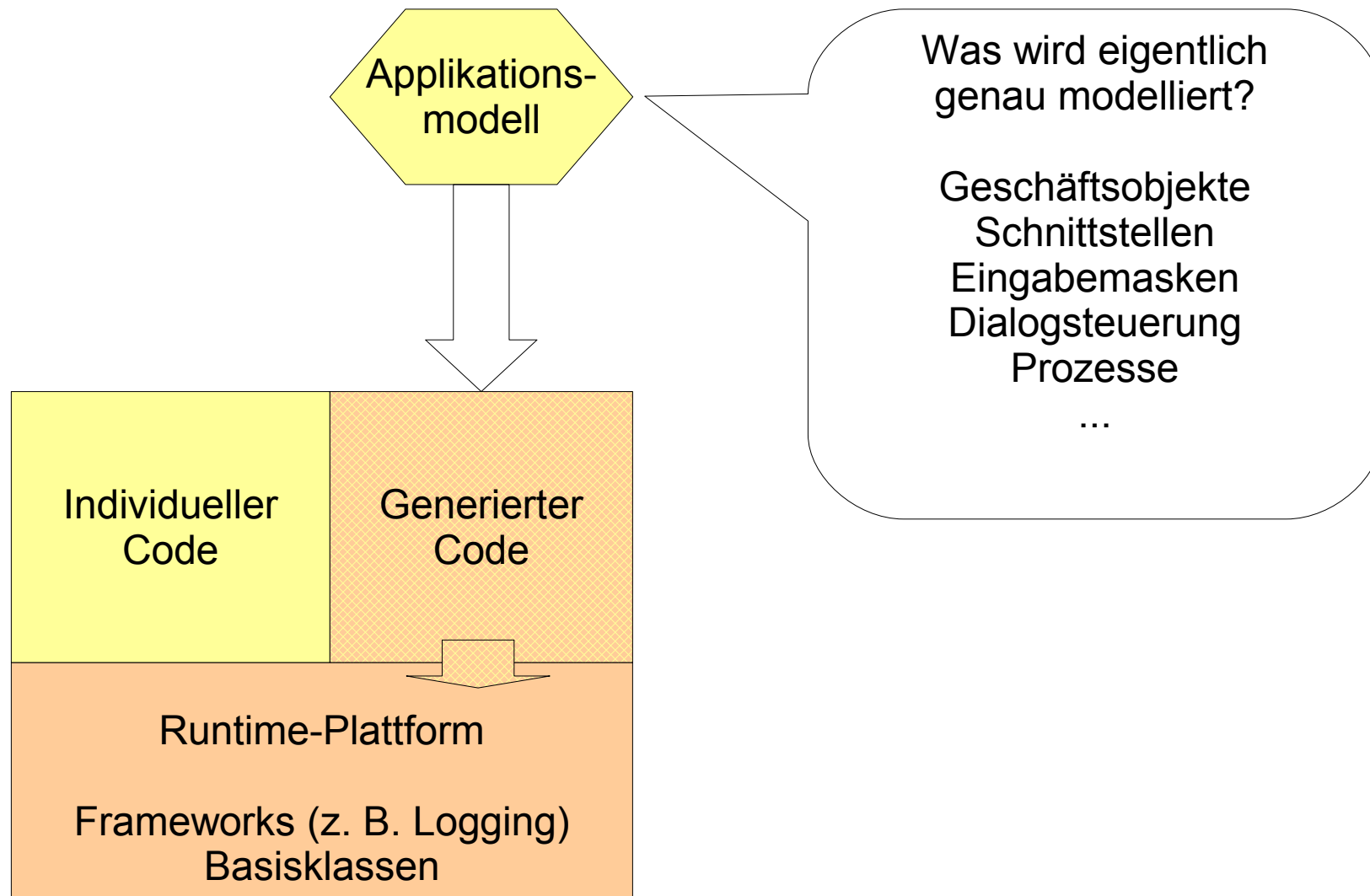
Gliederung

- Motivation, Konzepte
- Einordnung Faktor-IPS
- Subdomänen & Partitionierung
- Variante: Modellinterpretation
- Codegenerierung & Konfigurationsmanagement
- Modelltransformationen

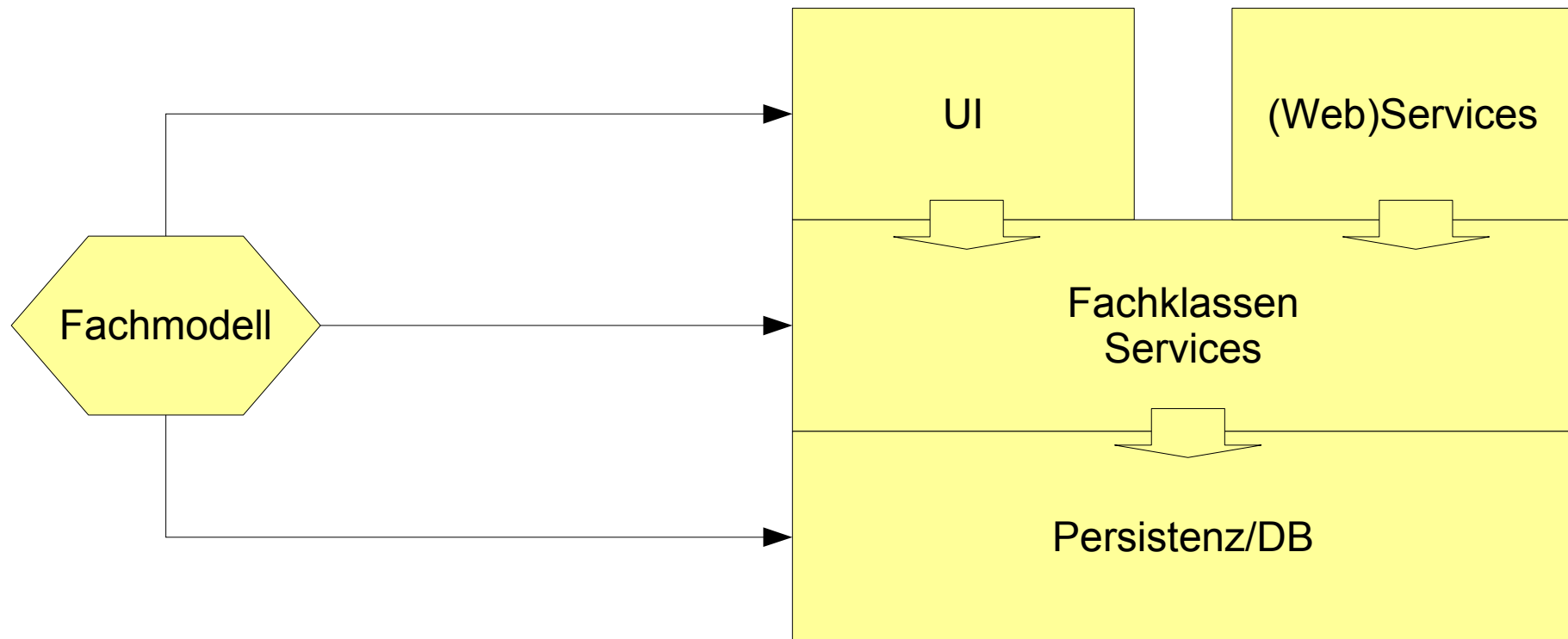
Applikationsmodell



Applikationsmodell

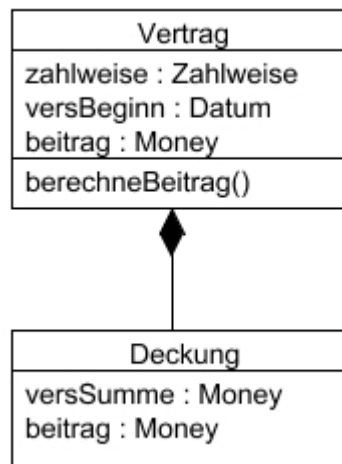


Pitfall



Aus einem fachlichen Modell werden unterschiedliche Ebenen der Softwarearchitektur generiert.

Pitfall: Beispiel



- Idee
Generierung von Dtos und eines statless Session Beans für die Beitragsberechnung
- Konsequenz
 - Das Modell ist nun ein Fachmodell und Schnittstellenmodell.
 - Änderungen am Modell sind Schnittstellenänderungen.

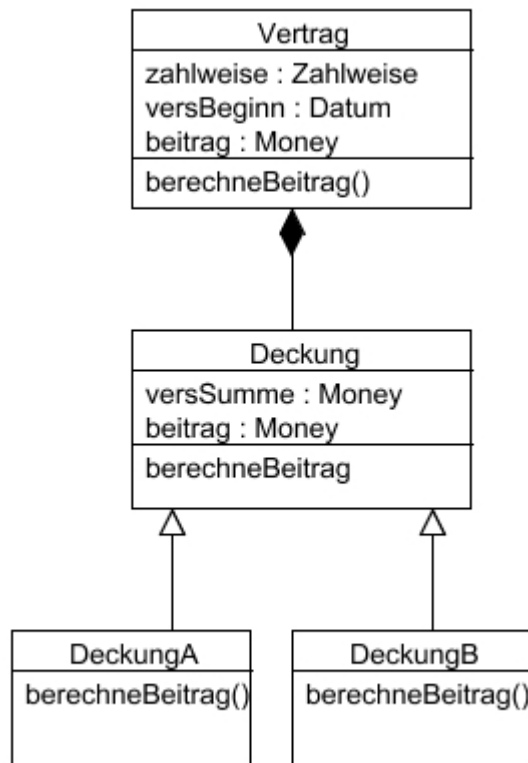
Pitfall: Beispiel

- Anforderung:
 - Ersetzen der Datentypen Zahlweise, Money und Datum durch Java-Standarddatentypen in der Schnittstelle.
- Umsetzung:
 - Erweiterung des Metamodells um „DatentypInSchnittstelle“
- Kann man machen

Pitfall: Beispiel

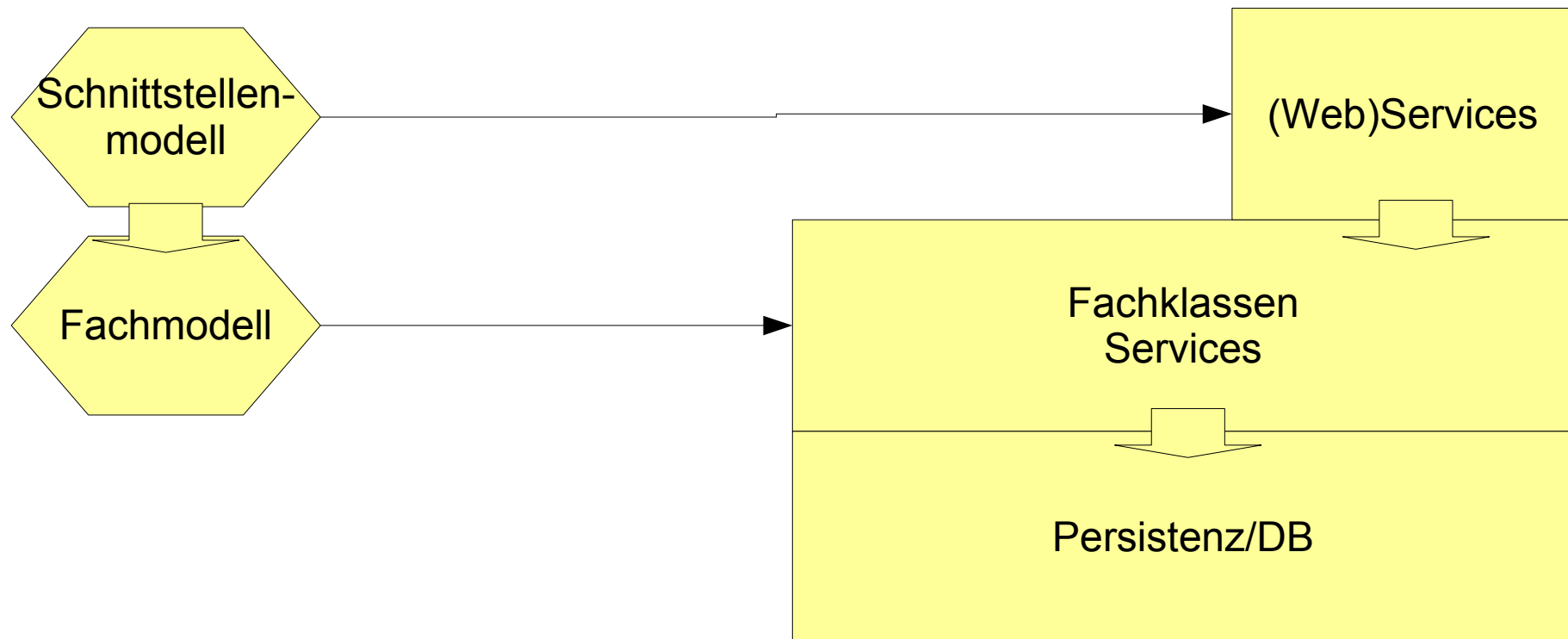
- Anforderung 2:
 - Bereitstellung einer Schnittstelle für PL/1 Programme
- Umsetzung
 - Erweiterung des Metamodells um DatentypInPI1Schnittstelle
- Jetzt wird es etwas unschöner.

Pitfall: Beispiel

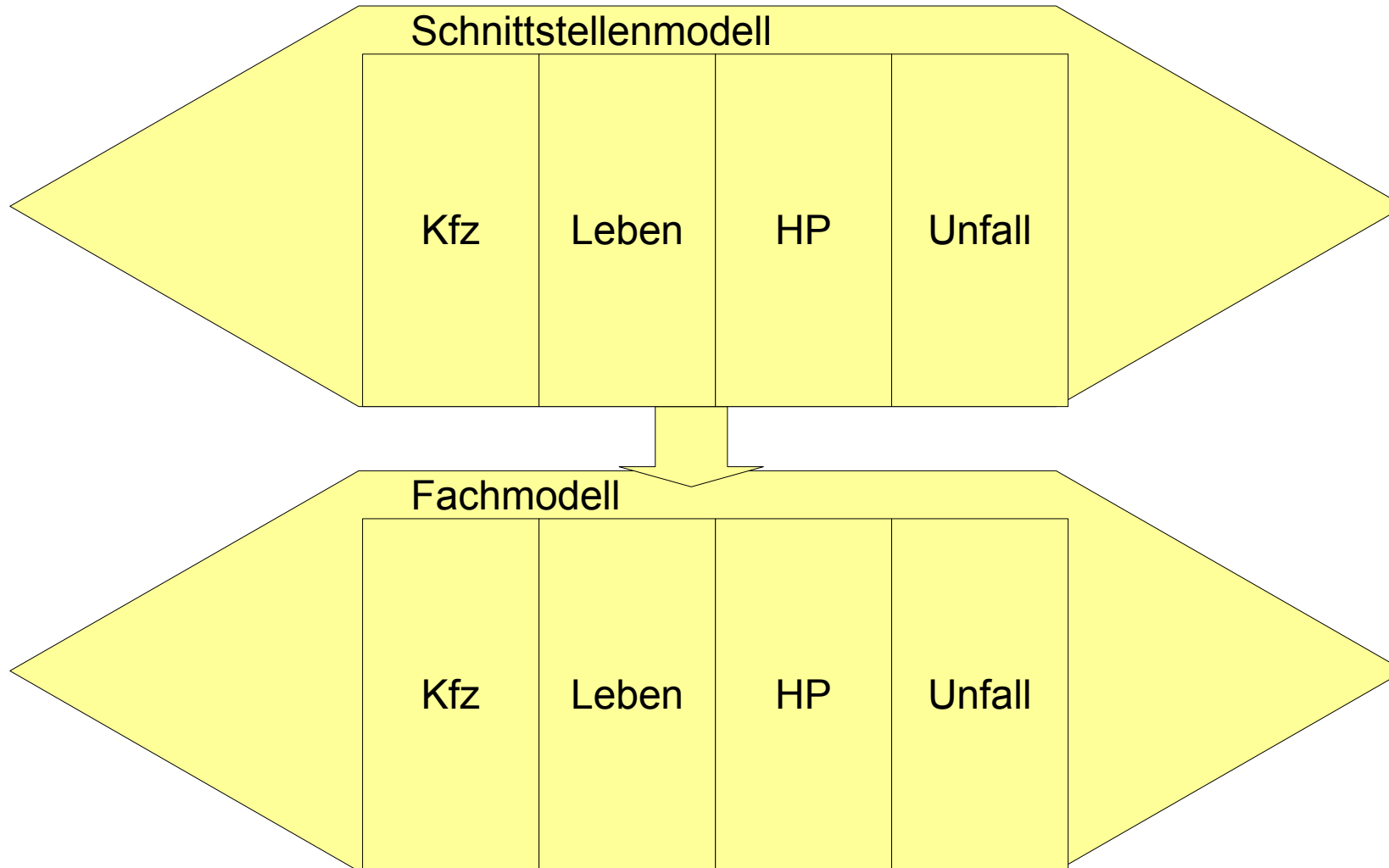


- Die Klassen DeckungA und DeckungB implementieren unterschiedliche Beitragsberechnungen, sind aber ansonsten identisch mit der Klasse Deckung.
- Die Klassen werden in der Schnittstelle nicht benötigt.

Alternative: Verwendung von Subdomänen



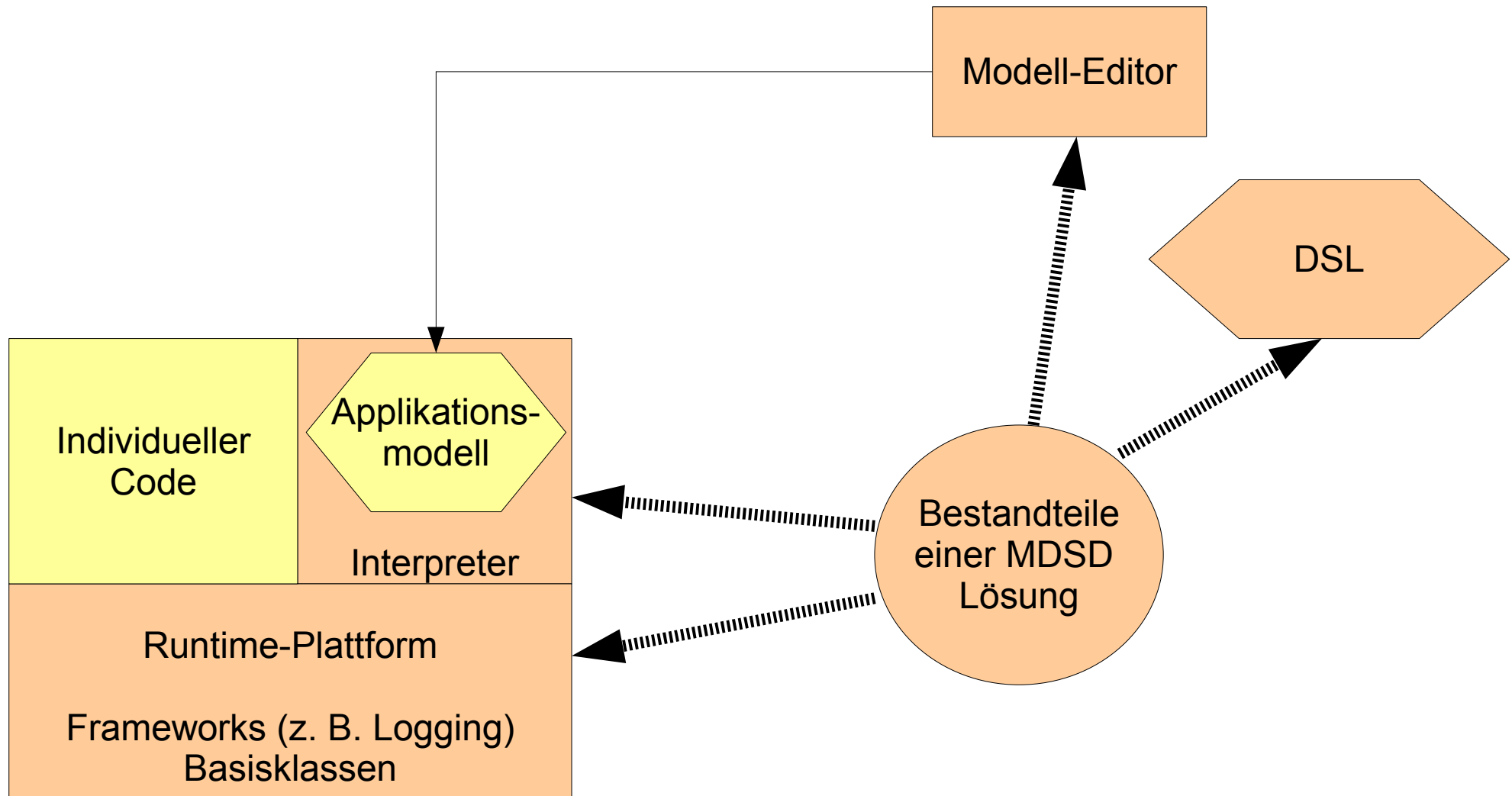
Modellpartitionierung vs. Subdomänen



Gliederung

- Motivation, Konzepte
- Einordnung Faktor-IPS
- Subdomänen & Partitionierung
- Variante: Modellinterpretation
- Codegenerierung & Konfigurationsmanagement
- Modelltransformationen

Variante: Interpretation des Modells zur Laufzeit



Gliederung

- Motivation, Konzepte
- Einordnung Faktor-IPS
- Subdomänen & Partitionierung
- Variante: Modellinterpretation
- Codegenerierung & Konfigurationsmanagement
- Modelltransformationen

Anforderungen an generierten Code

- Verständlich
 - Entwickler muss ihn verstehen und im Fehlerfall leicht debuggen können.
- Sollte gutes Design fördern, nicht verhindern
 - A framework should provide guidance with respect to good practice. It should make the right things easy to use. It should not impose too many requirements and not constrain the developer in ways not be appropriate.
- Leicht testbar
- Performant

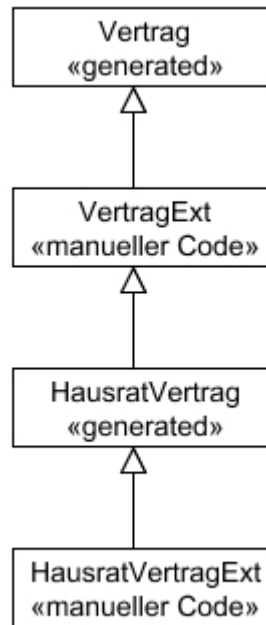
Techniken zur Codegenerierung

- Techniken
 - Verwendung von Templates (OpenArchitectureWare, EMF)
 - Programmatisch (Faktor-IPS)
- Gründe gegen Templates (damals)
 - keine vernünftigen Editoren verfügbar
 - kein Debugger verfügbar
 - Redundanz in den Templates erschwert Änderungen

Umgang mit manuellen Ergänzungen

- Ideal
 - Trennung von 100% generiertem und manuell-erstelltem Code in unterschiedliche Files ohne Beeinträchtigung des Designs und ohne Beeinträchtigung der Werkzeugunterstützung
 - Wieso ist das ideal?
 - *Weil der 100% generierte Code nicht versioniert werden muss, sondern im Buildprozess aus dem Modell generiert werden kann.*
 - Keine Inkonsistenzen möglich
 - *Mögliche Basis für Verwaltung unterschiedlicher Applikationsvarianten.*
 - Customizing einer Kundenvariante kann in den manuellen Klassen erfolgen.
- Bei MDSD für fachliche Modelle nicht leicht zu erreichen. Wieso?
 - Struktur wird modelliert und generiert.
 - Implementierung fachlicher Methoden wird dagegen programmiert.
 - Struktur und Methodenimplementierung gehören eigentlich beide ins gleiche Classfile.

Verwendung von Subclasses



- Vorteil
 - Trennung von generiertem und manuellem Code gewährleistet
- Nachteil
 - Konfigurationsmanagement bestimmt das Softwaredesign
 - zuviele unnötige Subclasses
 - Codeduplizierung, falls Implementierung einer generierte Methode angepasst werden muss.

Beispiel Codeduplizierung

Generierte Methode in Vertrag

```
public void setZahlweise(int neueZw) {  
    int altezw = this.zahlweise;  
    this.zahlweise = neueZw;  
    if (alteZw != neueZw) {  
        notifyChangeListeners(...);  
    }  
}
```

Gewünschte Anpassung:

Beim Setzen der Zahlweise soll der Beitrag neu berechnet werden und zwar bevor der ChangeEvent ausgelöst wird.

Beispiel Codeduplizierung

Einzig Lösung (ohne Anpassung Generator, DSL):
Duplizierung des Codes in der Subclass

```
public void setZahlweise(int neueZw) {  
    int altezw = this.zahlweise;  
    this.zahlweise = neueZw;  
    berechneBeitrag();  
    if (alteZw != neueZw) {  
        notifyChangeListeners(...);  
    }  
}
```

Ergänzung des generierten Codes und Mergen bei erneuter Generierung

- Varianten
 - Marker für User Code Sections
 - JMerge / Annotations im Javadoc
- Vorteil
 - Erlaubt optimales Design
 - keine Inkonsistenzen möglich, wenn man Generieren & Mergen in den Buildprozess aufnimmt.
- Nachteil
 - In der Praxis Einschränkungen beim Mergen
 - *reduzierter Entwicklungskomfort, z. B. Mergen funktioniert nicht, wenn das File fehlerhaft ist*
 - *doch wieder Einschränkungen beim Design, da bestimmte Aspekte nicht gemerged werden können, z. B. erweiterte implements Anweisung bei JMerge*

Noch mal das Beispiel

Lösung mit JMerge/Faktor-IPS

```
/**
 * @generated NOT
 */
public void setZahlweise(int neueZw) {
    int altezw = this.zahlweise;
    this.zahlweise = neueZw;
    berechneBeitrag();
    if (alteZw != neueZw) {
        notifyChangeListeners(...);
    }
}
```

Verwaltung des manuellen Codes in separaten Files und Einfügen bei Generierung

- Vorteil
 - Erlaubt optimales Design
 - Keine Inkonsistenzen
- Nachteil
 - Keine Werkzeugunterstützung für die Bearbeitung des Sourcecodes

Was muss versioniert werden?

- Sourcecode & andere Ressourcen (XML-Dateien etc.) der Anwendung
 - wie immer
- Bestandteile der MDSD Lösung
 - DSL,
 - Editoren
 - Generatoren
 - Plattform
- Editoren, DSL & Generatoren sind Teil der Entwicklungsumgebung!!!

Konsequenzen

- Auf allen Entwicklerrechnern und allen Buildrechnern muss die gleiche Version von Editoren, DSL & Codegeneratoren installiert sein.
- Alle Versionen der Entwicklungsumgebung, mit denen eine produktive Versionen des Anwendungssystem entwickelt wurde, müssen verfügbar sein.
 - Dazu muss natürlich klar sein, welche Version überhaupt benötigt wird.
- Die Weiterentwicklung der DSL kann eine Migration der bestehenden Modelle erforderlich machen.
- Management der Entwicklungsumgebung erforderlich
 - Je stärker die Entwicklung der DSL und der Generatoren noch im Fluss ist, desto mehr Koordination ist erforderlich.

Faktor IPS Lösung

- Jedes Projekt spezifiziert die Mindestversionsnummer von Faktor IPS, die benötigt wird.
 - Ist FIPS in einer kleineren Version installiert, wird kein Sourcecode generiert.
- Jede Faktor IPS Installation kennt die bisherigen Versionen und weiß, zu einer Version x.y.z ob für die Nachfolgeversion x.y.z+1 eine Migration erforderlich ist.
 - Ist eine Migration erforderlich, wird kein Sourcecode generiert (bis die Migration durchgeführt wurde).
- Jede Migration setzt zum Schluss die neue Mindestversionsnummer

Gliederung

- Motivation, Konzepte
- Einordnung Faktor-IPS
- Subdomänen & Partitionierung
- Variante: Modellinterpretation
- Codegenerierung & Konfigurationsmanagement
- Modelltransformationen

Definition Modelltransformation

- Erzeugung eines neuen Modells aus einem oder mehreren Eingabemodellen.

Einsatzbereiche

- Transformationen zwischen Werkzeugen mit unterschiedlichen Metamodellen aber gleicher bzw. sehr ähnlicher Semantik
 - Beispiel: Faktor IPS – msg.PM
 - Also: gleiches Abstraktionsniveau, aber unterschiedliche Werkzeuge
- Entwicklung von projektspezifischen Lösungen auf Basis von generellen Lösungen
 - Beispiel: Verfügbar ist eine Lösung für die Modellierung von ORM. Abstraktion der verschiedenen Persistenztechnologien.
 - Die Lösung muss eine große Bandbreite von Konzepte unterstützen, z. B. verschiedene Concurrency-Konzepte, Primary Key Konzepte etc.
 - Projektspezifisch kann man das einschränken. Z. B. Optimistisches Locking, alle Tabellen bekommen eine technische ID Spalte als PK.
 - Also: Vereinfachung der DSL, höheres Abstraktionsniveau
- MDA

Änderungen am generierten Modell

- Gründe
 - Abstraktionsniveau des Zielmodells ist niedriger.
 - Fehlende/Leicht unterschiedliche Konzepte lassen keine 1-1 Transformationen zwischen zwei Werkzeugen zu.
 - *Beispiel: Faktor IPS - msgs.PM*
- Folgen
 - Mergen der manuellen Änderungen erforderlich

Aktuell verfügbar Lösungen

- QVT (Query Views Transformations)
 - Standard der OMG.
 - Z. B. verfügbar in Together Architect von Borland.
- ATL
 - Open-Source-Projekt innerhalb Eclipse GMT.
- Xtend
 - Bestandteil von openArchitectureWare
 - Ebenfalls Open-Source-Projekt innerhalb von Eclipse GMT.