Kombination von Modellierungstechniken für den Softwareentwurf

Dr. Eckhardt Holz
Institut für Informatik
Humboldt-Universität zu Berlin

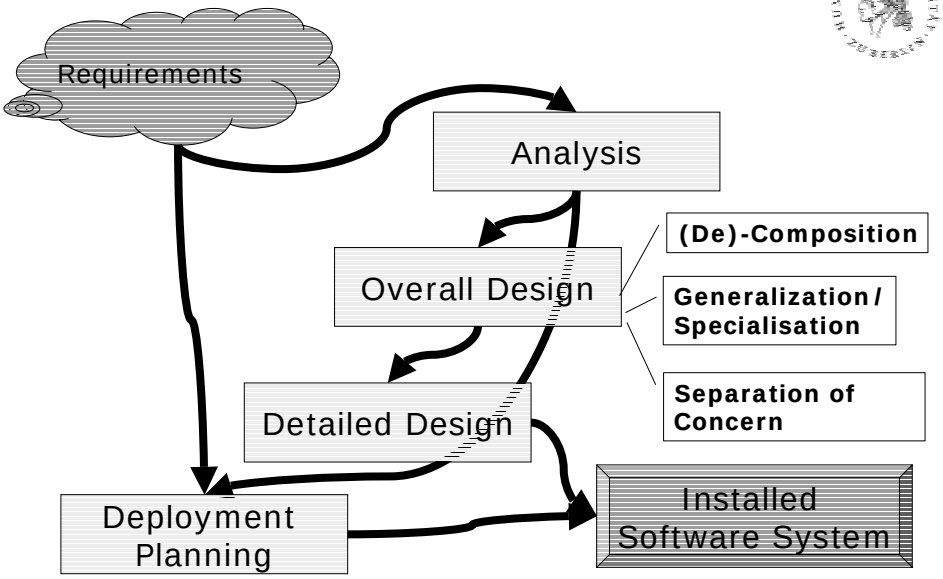
- Warum Modellieren wir beim Softwareentwurf?
- Kombination von Modellen
  - Ursachen
  - Unterschiedliche Formen
- Kombination von Modellierungstechniken
  - Ziel ist die Unterstützung der Kombination von Modellen
- Technische Unterstützung
- Ausblick

Warum Modellieren wir beim Softwareentwurf?

- Beherrschung der Komplexität
- Unterstützung des Entwurfsprozesses
- Automatisierung von Routinearbeiten
- Kommunikation und Dokumentation
- Verifikation/Validierung des Entwurfs

  ...
- *Das Hauptziel bleibt aber die Entwicklung eines Softwareproduktes!*

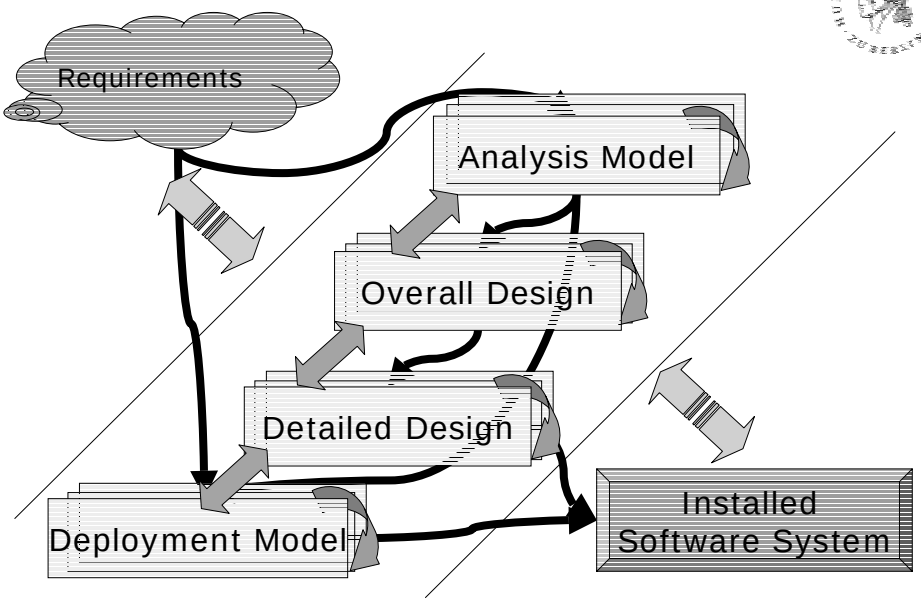
Modelle im Softwareentwurfsprozeß



Software Engineering ...

... nutzt eine *Vielzahl von Modellen*

- Konzeptionelle Abstraktion
  - Dekomposition und Refinement
  - Abstraktes Modell bildet Kontext für Verfeinerungen
- *Separation of Concern*
  - Views
  - Modelle beschreiben das System von einem begrenzten Sichtpunkt aus



Modelle und Spezifikationen ...

- ... können bereits in unterschiedlichen Modellierungstechniken vorliegen
- ... können in unterschiedlichen Modellierungstechniken erstellt werden

- Der Entwurf komplexer Softwaresysteme nutzt eine *Vielzahl von Modellierungstechniken.*

Formal Description Techniques
- SDL
  - Specification and Description Language
- MSC
  - Message Sequence Charts
- VHDL
  - Very High Speed Integrated Circuit Hardware Description Language
- ASM
  - Abstract State Machines
- LOTOS
  - Language of Temporal Ordering Specifications
- Petri Nets

Semi-formal Design Techniques
- UML
  - Unified Modeling Language
- OMT
  - Object Modeling Technique
- Booch
- ER-Diagrams
  - Entity Relationship Diagrams
- State Charts

Programming Language Style
- IDL
  - Interface Definition Language
- ODL
  - Object Definition Language
- ASN.1
  - Abstract Syntax Notation Nr. 1

Informal Techniques
- Requirements Specification Document
- Structured Text

Kombination von Modellen

- In welcher Beziehung stehen die in den unterschiedlichen Modellen gehaltenen Informationen zu einander?

- Klassifikation der Beziehungen zwischen Modellen
- Strategien für den Übergang zwischen bzw. die Kombination von Modellen

Bei der Entwicklung komplexer Software werden unterschiedlicher Modellierungstechniken und Modelle kombiniert …

… bei dem Übergang zwischen Entwurfsphasen
- Analyse, Design, Implementation, …
- *Process Driven Combination*

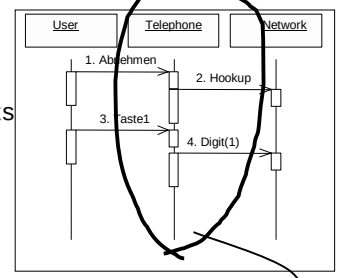
… innerhalb einer Entwurfsphase
- Einzelkomponenten, Sichten
- *Structural* oder *Abstraction Driven Combination*

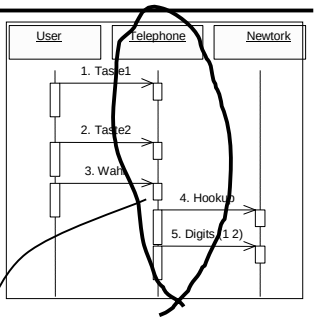
… für unterschiedliche Entwurfsaktivitäten
- Refinement, Implementation, Verifikation/Validierung
- *Process Driven Combination*

Process Driven Combination

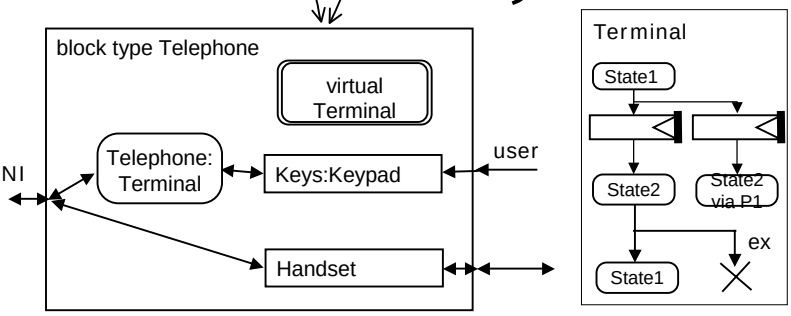
- Merkmale:
  - Modelle einer Designphase oder Aktivität dienen als Ausgangspunkt für die nächste Phase/Aktivität
  - Für die unterschiedlichen Phasen/Aktivitäten können verschiedene Modellierungstechniken eingesetzt werden
- Benötigt:
  - Abbildung zwischen den Modellen
- *Die Modelle stehen in einer zeitlichen Relation zu einander.*

Requirements MSC



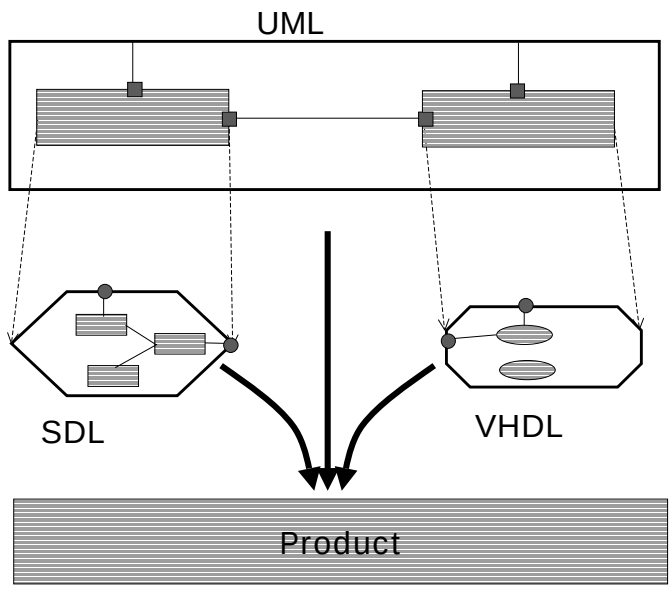


Design SDL



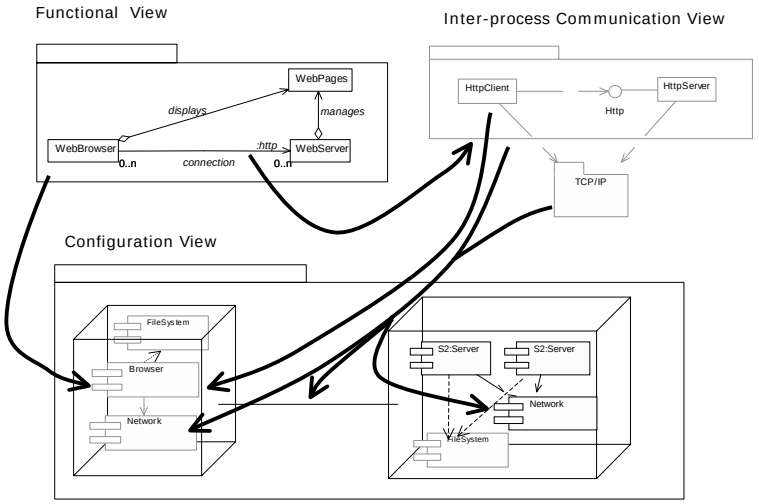
Structural Combination

- Merkmale:
  - Dekomposition in (disjunkte) Teilmodelle
  - Teilmodelle gehören zur gleichen Entwurfsphase
  - Anwendung unterschiedlicher domänen- oder aufgabenspezifischer Modellierungstechniken
- Benötigt:
  - Beschreibung der Gesamtstruktur (Kontext)
  - Schnittstellenspezifikationen
  - Offene Teilmodelle
- *Das Gesamtmodell ist eine Komposition der Teilmodelle.*



Abstraction Driven Composition

- Merkmale:
  - Modelle (Sichten) beschreiben das System unter Nutzung unterschiedlicher Abstraktionen
  - Modelle gehören zur gleichen Entwurfsphase
  - Anwendung unterschiedlicher aufgabenspezifischer Modellierungstechniken
- benötigt:
  - Gesamtsicht oder Relationen zwischen Sichten
  - Identifikation von Referenzpunkten
- *Das Gesamtmodell entsteht durch Überlagerung der Einzelmodelle (Sichten).*



Unterstützung der Modellkombination auf der Sprachebene

Klassische Vorgehensweisen

Sprachintegration

- *Single-Language*-Prinzip
- vielfach komplexe Sprache mit semantischen Problemen
  - UML (Booch + OMT + OOSE)
  - ESPRIT-Project SPECS (SDL+LOTOS+Estelle)

- Umfaßt alle Elemente der Einzelsprachen
  - kleinstes gemeinsames Vielfache
  - oft auch Unifikation ähnlicher Konzepte

Übersetzung

- Angepaßter und begrenzter Konzeptraum
- oft keine explizite Grammatik für semiformale Techniken
- Starke Abhängigkeit von Grammatikänderungen
  - ESPRIT-Project INSYDE (OMT-OMT*-SDL/VHDL)
  - Example: SDL-ASM

- Konzentration auf gemeinsame/ähnliche Anteile
  - Größter gemeinsamer Teiler
  - Spezifische Konzepte der Einzelsprachen von diesem Kern abgeleitet

- Beide Verfahren versuchen eine möglichst weitgehende Kombination
  - Spezifika der Gründe der Modellkombination werden kaum berücksichtigt
- Was passiert, wenn mehr als zwei Sprachen betrachtet werden?

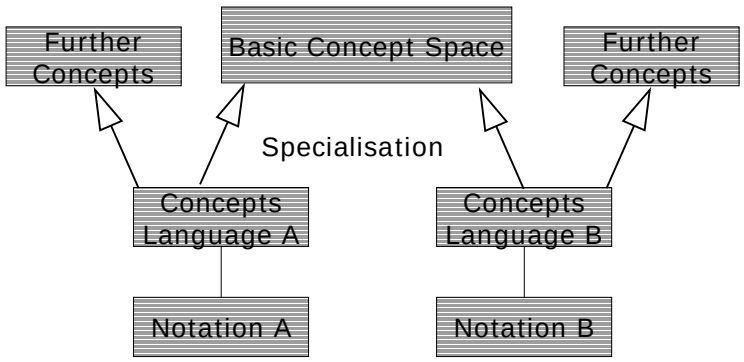
Kooperation der Einzelsprachen

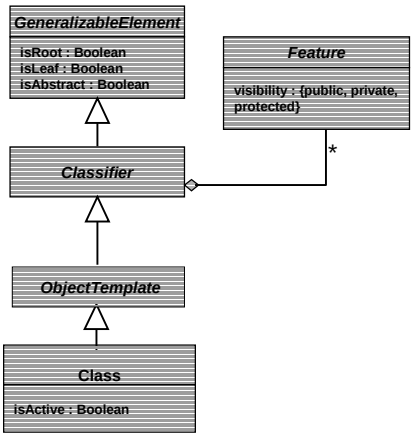
- Zielgerichtete Kombination
- basiert auf Konzeptwelt (Meta-Model, abstrakte Grammatik)
- keine Abhängigkeit von konkreten Notationen
  - SDL-UML (Z.109)
  - OMG *Model Driven Architecture* (MDA)

Konzeptraumbasierte Kombination

- Merkmale:
  - Modelle basieren auf Konzepten
  - Notationen reflektieren die gesamte oder einen Teilbereich der Konzeptwelt
  - Konzeptweltdefintion ist definiert durch
    - Abstrakte Grammatik
    - Meta-Model
- Kombinationsformen
  - Gemeinsame Konzepte als Basis
  - Gemeinsame Konzepte als Spezialisierung
  - Beziehungen zu einer gemeinsamen Konzeptwelt

Gemeinsame Konzeptbasis





Object-template	::	*Agent-type-definition* | *Data-type-definition*
Agent-type-definition	::	*Agent-type-name* *Agent-kind* [*Agent-type-identifier*] *Agent-formal-parameter** *Data-type-definition-**set*** ***(...)*** *Procedure-definition-**set*** *Agent-definition-**set*** *Gate-definition-**set*** *Channel-definition-**set*** [*State-machine-definition*]

Meta-Model Abstract Grammar

- UML
  - jede Class-Definition ist ein Object-Template
- SDL
  - jede Agent-Type-Definition ist ein Object-Template
  - jede Data-Type-Definition ist ein Object-Template

Die Basierung auf einer gemeinsamen Konzeptwelt

... erfordert **Änderungen an der Sprachdefinition**

- Meta-Model
- Abstrakte Grammatik

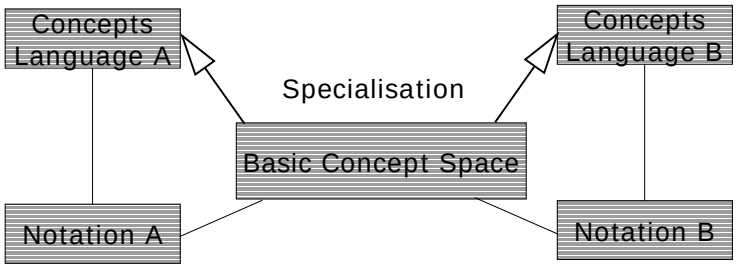
... kann auf der Notationsebene verborgen bleiben

... eignet sich für die *Process Driven Combination*

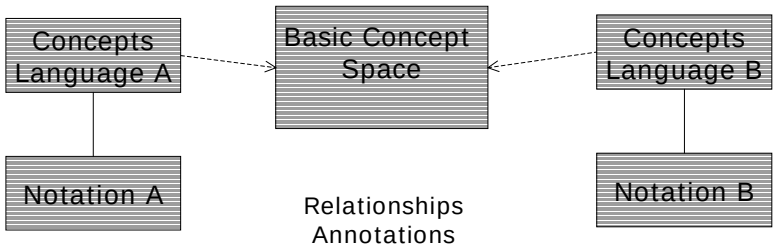
– UML-Konzepte in SDL (*Type references*)

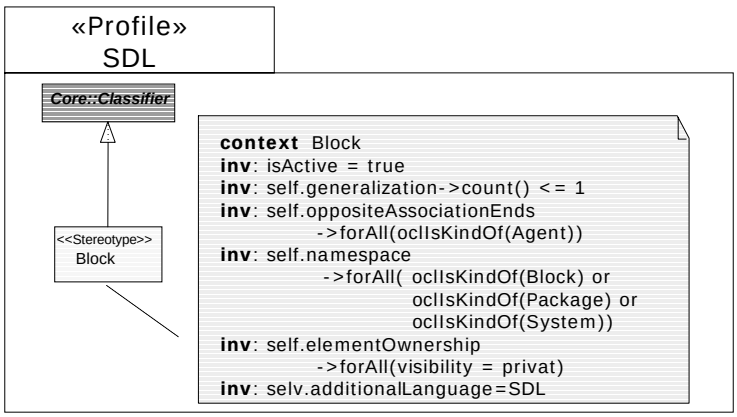
- von SDL nach UML

Ableitung der gemeinsamen Konzepte

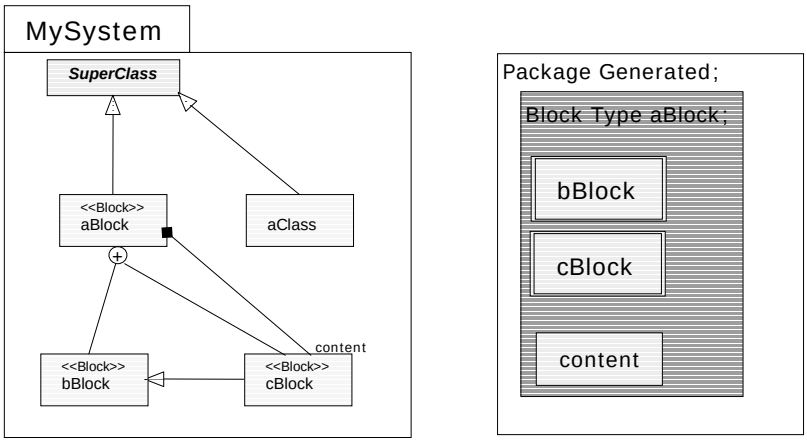


Relation zu einer gemeinsamen Konzeptwelt





im Modell!



- UML
  - Die markierten Klassen haben Block-Eigenschaften
- SDL
  - Nur die im UML-Modell markierten Klassen werden reflektiert.

Die Nutzung von Relationen und die Ableitung einer gemeinsamen Konzeptwelt

- ... erfordern Modifikationen der Sprachdefinition und/oder **Erweiterungen auf der Modellebene**
- ... benötigen Erweiterungselemente
  - spezielle Attribute in der abstrakten Grammatik
  - Stereotypes im Meta-Modell
- ... sind auf der Notationsebene sichtbar
  - selektive Auswahl (Attributwerte, Markierungen)
  - Standardnotation oder spezielle Symbole
- ... eigen sich für die *Structural Combination* und die *Abstraction Driven Combination*

- UML-SDL-Profil
  - UML und SDL

Anforderungen an Sprachentwicklung

Konzeptbasierte Sprachdefinition

- Konzeptwelt ist primär
  - strukturelle Spezifikation durch abstrakte Grammatik oder Meta-Modell
  - Ansatz für Semantikdefinition
- Notation ist abgeleitet
  - Zuordnung von Notationselementen zu Konzepten
  - Ableitung einer konkreten Grammatik aus der abstrakten Grammatik

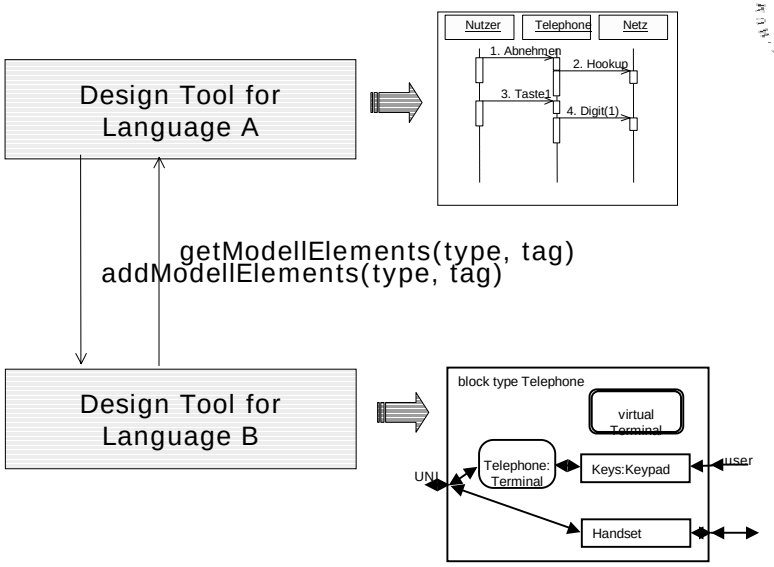
Anbindung an andere Konzeptwelten

- Meta-Model-Stereotypen/Attributierung
  - Modulare Sprache:
    - Core
    - Advanced Concepts
  - Klar definierte Erweiterungspunkte und –mechanismen zur Erweiterung/Spezialisierung der Sprache
    - Profile und Teilsprachen
    - Auswahl/Visualisierung durch Notation auf Modellebene

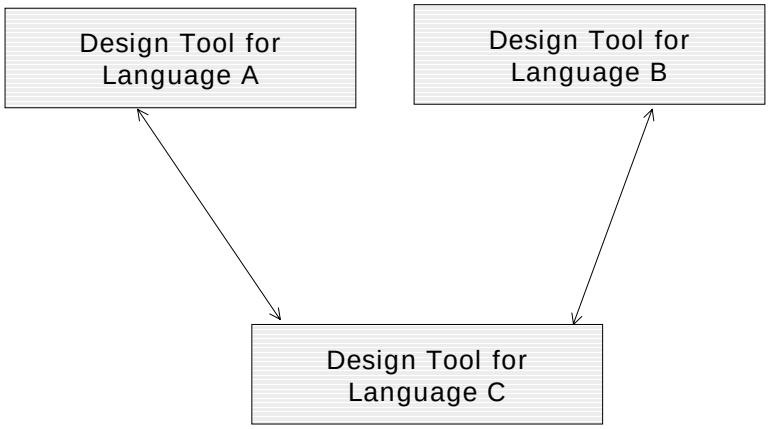
Konzeptbasierte Werkzeugschnittstelle

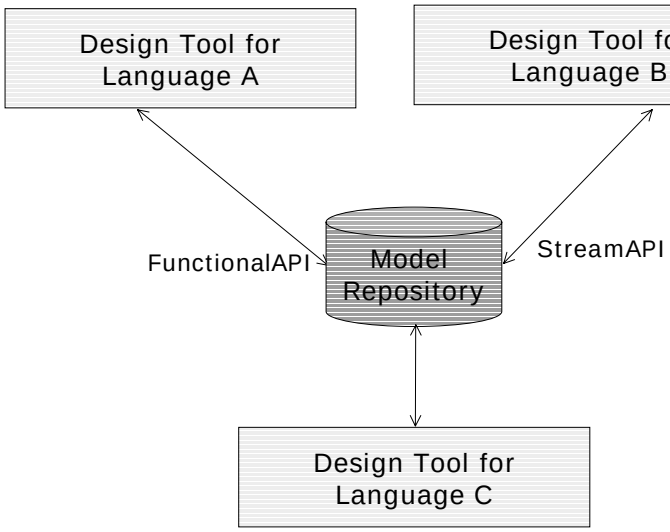
- Funktionale APIs
  - Tools basieren auf Konzeptwelt – Interfaces reflektieren Konzepte
    - UML-Facility
  - Navigation, Modifikation, Extraktion, Erzeugung von Modellen
- Explizites and flexibles Austauschformat (Stream API)
  - XML/XMI
  - mindestens auf Konzeptbasis, zusätzlich auch Notation
- Modell-Repositories
  - Speicherung von gesamten Modellen oder von auf gemeinsamen Kern basierenden Modellelementen
  - Nutzung im gesamten Lebenszyklus (Entwurf, Modellierung, Simulation, Ausführung, ...)

Konzeptraumbasierte Kombination



Konzeptraumbasierte Kombination





Unterstützende Werkzeuge

- Meta Object Facility (MOF)
  - ***model driven distributed object framework***
  - Dient der
    - Spezifikation
    - Konstruktion
    - Verwaltung
    - Austausch
    - Integration
    - Integrating

  von Metadaten und Meta-Modellen in Softwaresystemen
- Von der OMG standardisiert
  - Konzeptraum und Repository-Funktionalität
  - Programmatische und Streaming-Schnittstelle
  - Textuelle/Grafische Darstellung (MODL bzw. UML)

Sprachdefinition ist ein Meta-Modell und somit auch ein Modell!

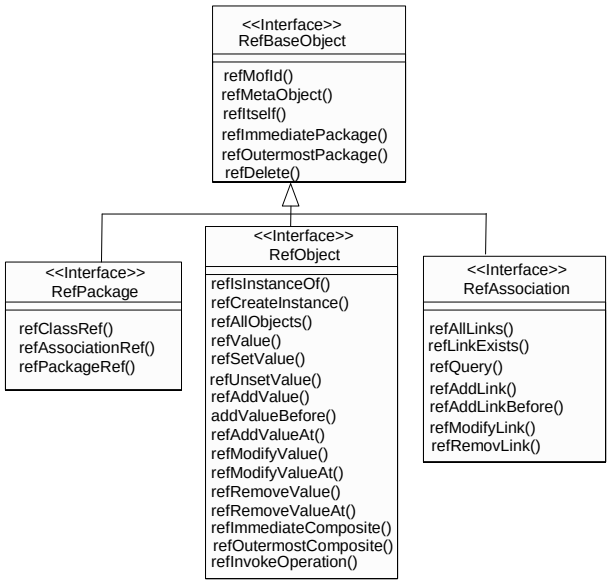
- Kann manipuliert werden wie jedes Modell
  - Erweitern um neue Konzepte
  - Spezialisieren vorhandener Konzepte
  - Einschränkungen
  - Strukturelle Änderungen
- Kann ausgetauscht werden wie jedes Modell
  - Manipulation und Austausch auf der Basis von MOF

MOF-IDL

- Zwei Module
- Reflective
  - Generische Interfaces
    - Browser-Artige Funktionalität zur Inspektion von Modellen
    - Basis für angepaßte Interfaces

```
interface RefBaseObject;
interface RefObject;
interface RefAssociation;
interface RefPackage;
```

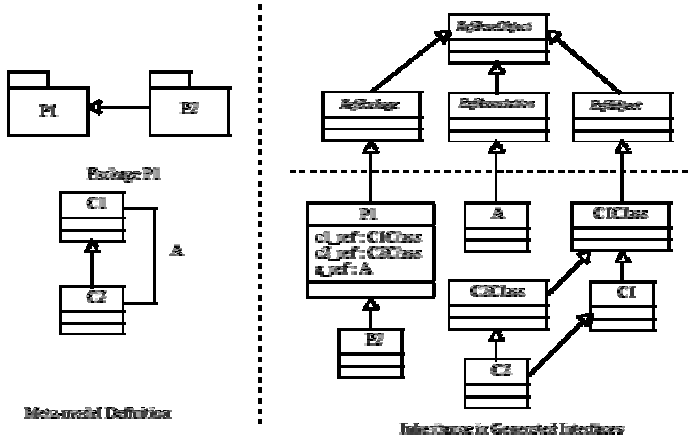
- Model
  - Angepaßte Interfaces
    - Ableitungen von generischen Interfaces
    - Instanzieren, Traversieren und Modifizieren von MOF-Modellelementen
    - basieren auf Begriffswelt des MOF-Metamodells



```
•interface ClassClass : ClassifierClass {

•readonly attribute ClassUList all_of_type_class;
•readonly attribute ClassUList all_of_class_class;
•const string CLASS_CONTAINMENT_RULES =
•"org.omg.mof:constraint.model.class.class_containment_rules";
•const string ABSTRACT_CLASSES_CANNOT_BE_SINGLETON =
•"org.omg.mof:constraint.model.class.abstract_classes_cannot_be_singleton";

•Class create_class (
•/* from ModelElement */ in ::Model::NameType name,
•/* from ModelElement */ in ::Model::AnnotationType annotation,
•/* from GeneralizableElement */ in boolean is_root,
•/* from GeneralizableElement */ in boolean is_leaf,
•/* from GeneralizableElement */ in boolean is_abstract,
•/* from GeneralizableElement */ in ::Model::VisibilityKind visibility,
•/* from Class */ in boolean is_singleton)
•raises (Reflective::MofError);

•}; // end of interface ClassClass
```

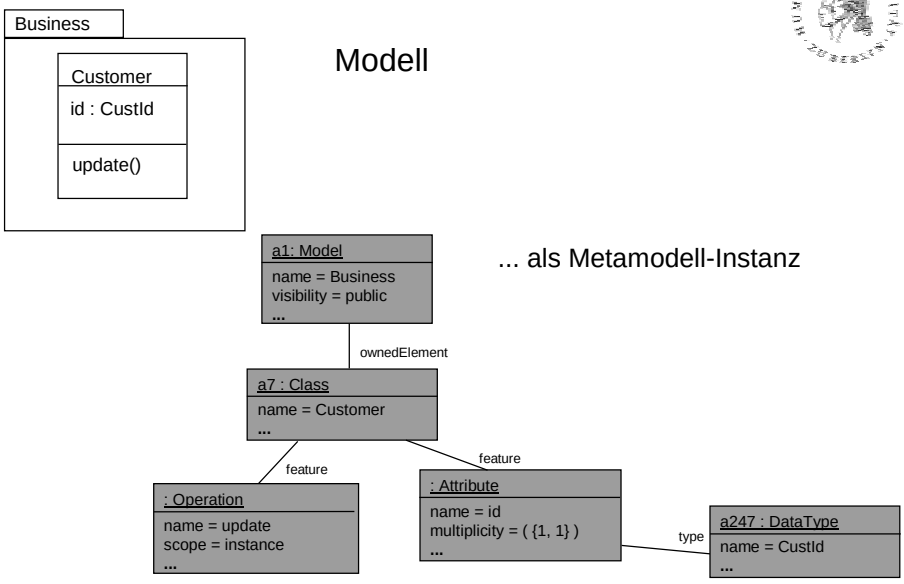
```
•interface Class : ClassClass, Classifier {

•boolean is_singleton ()
•raises (Reflective::MofError);
•void set_is_singleton (in boolean new_value)
•raises (Reflective::MofError);

•}; // end of interface Class
```

- Mittels MOF können Schnittstellen zu Metamodellen generieren
  - Erzeugung von Repositories für Modelle entsprechend ihrem Metamodell
  - Streaming-Schnittstelle
- MOF-to-IDL-Mapping
  - Ablauf
    - MOF-konformes Metamodell erzeugen
    - Metamodell in MOF-Repository ablegen
    - IDL-Mapping
  - Struktur folgt der MOF-IDL-Struktur
    - Bsp.:UMLFacility



- Streaming-Schnittstelle
  - XMI – XML Model Exchange Interface
    - DTD für MOF-Metamodelle
    - MOF-Konzeptwelt als XML auf der Basis der MOF-DTD
  - Mittels MOF können Metamodelle als XML ausgegeben werden
  - Mittels MOF kann für ein Metamodell eine DTD erzeugt werden
    - Struktur folgt der MOF-DTD
    - Modelle können in XML ausgetauscht werden
    - Bsp.:UML-XMI

Modell



... als Metamodell-Instanz

```
<!-- Document Prologue, etc.  -->
<Model XMI.id="a1"> <name>Business</name><visibility XMI. value="public"/>
   <isRoot XMI.value = "false"/> <isLeaf XMI.value="false"/>
    <isAbstract XMI.value="false"/>
    <ownedElement>
          <Class XMI. id="a7"><name>Customer</name><visibility XMI. value="public"/>
           <isRoot XMI.value = "false"/><isLeaf XMI.value="false"/>
           <isAbstract XMI.value="false"/>
           <isActive XMI.value="false"/>
                    <feature>
                    <Attribute><name>id</name><visibility XMI.value="public"/>
                              <ownerScope XMI.value="instance">

          <multiplicity><field>1</field><field>1</field></multiplicity>
                              <changeable XMI.value="true"/>
                              <targetScope XMI.value="instance"/>
                              <type>< XMI.reference href="a247"/></type>
                    </Attribute>
                    <Operation><name>update</name><visibility XMI.value="public"/>
                              <ownerScope XMI.value="instance"/><isQuery
    XMI.value="false"/>
                              <specification/><isPolymorphic XMI.value="false"/>
                              <concurrency XMI.value="guarded"/>
                    </Operation>
          </feature>
         </Class>
    </ownedElement>
</Model>
```


Zusammenfassung und Ausblick

- Die Nutzung unterschiedlicher Modelle und Modellierungstechniken im Softwareentwicklungsprozeß ist inherent.
- Die Beziehungen zwischen den Einzelmodellen resultieren aus
  - dem Entwicklungsprozeß,
  - den Strukturellen Anforderungen und
  - dem aktuellen Abstraktionsniveau
- Sprachintegration und Übersetzungstrategien besitzen nur begrenzte Anwendbarkeit
  - Semantische Probleme
  - Ausdruckskraft
  - Sprachumfang

Ein konzeptraumbasierter Ansatz verspricht ...

- die selektive Kombination von Modellen
- eine erhöhte Stabilität gegenüber Notationsänderungen
- die Unterstützung der unterschiedlichen Formen der Modellkombination
- eine effizientere Arbeitsweise

... und stellt höhere Anforderungen an die Werkzeuge.

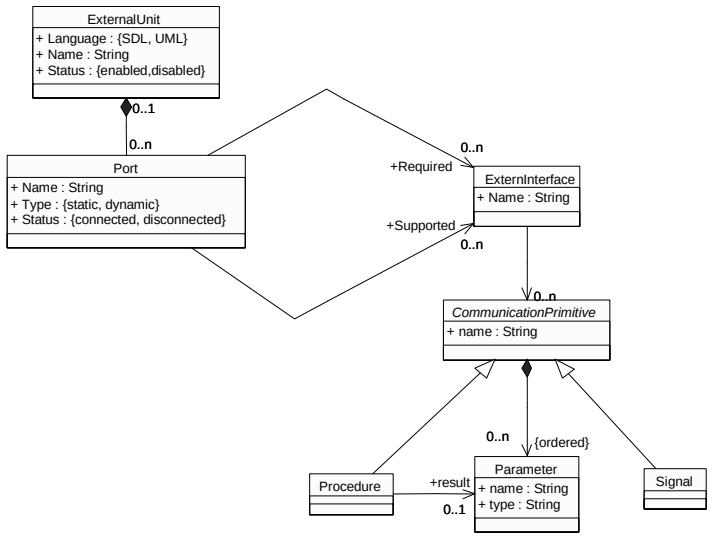
- Zukünftige Sprachentwicklungen sollten Erweiterungsmechanismen berücksichtigen.

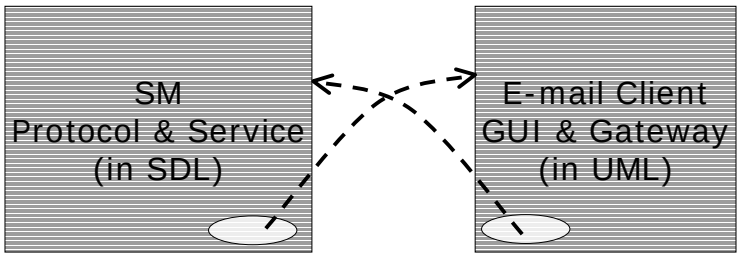
Example

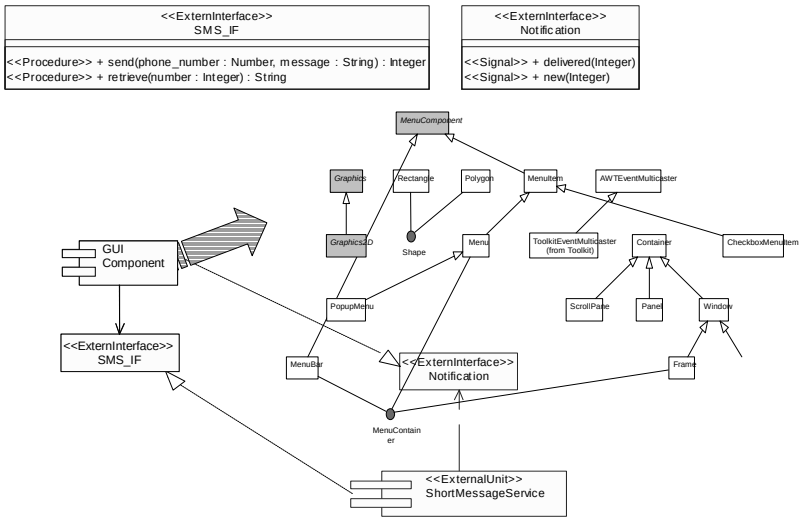
- Task: Development of an SMS-Email-Gateway

- Protocol stack and Short Message Service
  - Specified in SDL
- Gateway
  - Graphical user interface
  - Send/receive/forward functionality
  - To be developed with UML

- Structural combination of two submodels
- Connection to submodel
  - SDL: signals, remote procedures
  - UML: events, methods of objects
  - Both languages support the concept `Interface`

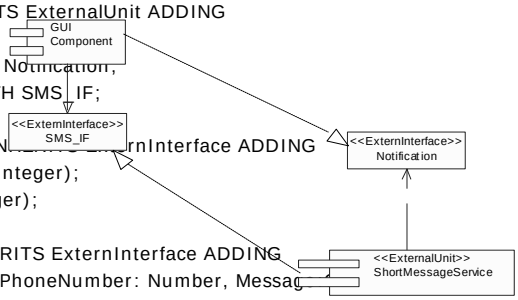


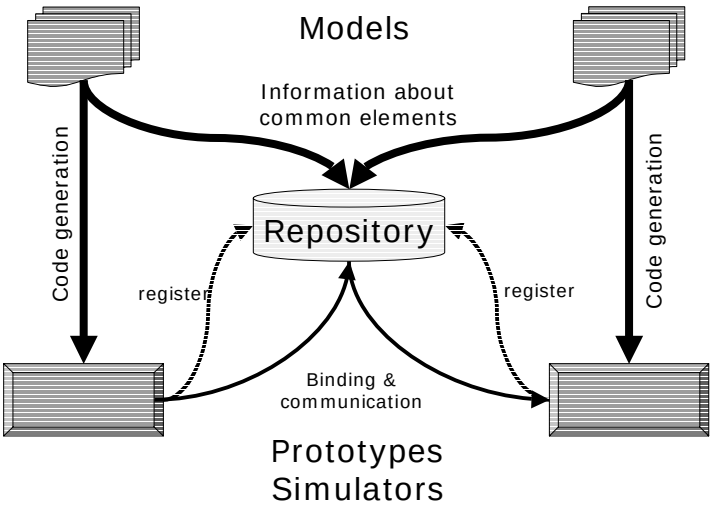




```
SYSTEM ShortMessageService
    BLOCK TYPE GUI INHERITS ExternalUnit ADDING
        GATE Environment
            IN WITH Notification;
            OUT WITH SMS_IF;
    END BLOCKTYPE;
    INTERFACE Notification INHERITS ExternInterface ADDING
        SIGNAL delivered(Integer);
        SIGNAL new (Integer);
    END INTERFACE;
    INTERFACE SMS_IF INHERITS ExternInterface ADDING
        PROCEDURE Send(PhoneNumber: Number, Message:Charstring):Integer;
        PROCEDURE Retrieve(Num:integer): Charstring;
    INTERFACE;

    BLOCK TYPE SMS_Protocol INHERITS SMS ADDING
        GATE Environment
            OUT WITH Notification;
            IN WITH SMS_IF;
        /* further implementation details ... */
END SYSTEM;
```


```
SYSTEM ShortMessageService
    BLOCK TYPE GUI INHERITS ExternalUnit ADDING
        GATE Environment
            IN WITH Notification;
            OUT WITH SMS_IF;
    END BLOCKTYPE;
    INTERFACE Notification IN                rnInterface ADDING
        SIGNAL delivered(Integer);
        SIGNAL new (Integer);
    END INTERFACE;
    INTERFACE SMS_IF INHERITS ExternInterface ADDING
        PROCEDURE Send(PhoneNumber: Number, Messag
        PROCEDURE Retrieve(Num:integer): Charstring;
    INTERFACE;

    BLOCK TYPE SMS_Protocol INHERITS SMS ADDING
        GATE Environment
            OUT WITH Notification;
            IN WITH SMS_IF;
        /* further implementation details ... */
END SYSTEM;
```





Excution scenarios