

Abschlusspräsentation

Praktikum: Modellgetriebene Software-Entwicklung

Dirk Neumann, Patrick Mehl, Jakob Höfker | 27. Juli 2021

Inhaltsverzeichnis

1. M1: Meta-Modellierung
2. M2: Xtext - textuelle Syntax
3. M3: QVTo - Modelltransformation
4. M4: Xtend - Java Quellcode Generierung
5. M5: Sirius - grafischer Editor
6. Key Learnings

M1: Metamodell
○○○

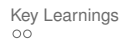
M2: Xtext
○○

M3: QVTo
○○○○

M4: Xtend
○○

M5: Sirius
○○○

Key Learnings
○○



Entwurfsentscheidungen für die Meta-Modellierung

- ① Designentscheidung: Verwendung von 4 Packages
 - Common Package für wiederverwendbare Elemente
 - Vorteil: klare Trennung der View Points
 - Nachteil: geringe Unterstützung für Subpackages von Metamodellen \Rightarrow macht spätere Entwicklung aufwendiger
- ② Designentscheidung: Entscheidung für einfachere Modellierung
 - führt zu mehr sowie komplexeren OCL Ausdrücken
 - erhöht den Bedarf nach bidirektionalen Referenzen
 - \Rightarrow beeinflusst wie man mit dem Meta-Modell weiterarbeiten kann
- ③ Designentscheidung: ENUM zur Modellierung des Type
 - Vorteil: einfaches statisches Typesystem
 - Nachteil: geringe Flexibilität \Rightarrow beschränkt den Raum der beschreibbaren Komponenten

Entwurfsentscheidungen für die Meta-Modellierung

- ④ Designentscheidung: Providing/Requiring von Interfaces als Referenz (Set) realisiert
 - jedes Interface kann nur einmal in jeder Referenz existieren
 - Nachteil: geringe Flexibilität \Rightarrow beschränkt den Raum der beschreibbaren Komponenten

Xtext - textuelle Syntax

```
EnvironmentViewType {
    environmentElements {
        Container ApplicationServer,
        Container DatabaseServer,
        Link Network {
            containers (ApplicationServer, DatabaseServer)
        }
    }
}

AllocationViewType {
    allocationContexts {
        AllocationContext {
            container ApplicationServer
            assembly Sys.WebGUIAC
        },
        AllocationContext {
            container ApplicationServer
            assembly Sys.MediaStoreAC
        },
        AllocationContext {
            container DatabaseServer
            assembly Sys.PoolingAudioDBAC
        }
    }
}
```

■ Ergebnis: mediastore.simplepalladio

Probleme beim Entwurf der textuellen Syntax

- ❶ Problem: Import von Subpackages konnte nicht aufgelöst werden
 - Lösung: Verwendung von alternativer Importschreibweise anstatt Ns URI
 - `import "platform:/resource/SimplePalladio/.../*.ecore#//..."`
- ❷ Probleme: Proxy-Ausflösung zwischen Xtext Dateien
 - erster Ansatz: für jeden ViewType eine Xtext Grammatik
 - Lösung: Grammatik als Weaving-Modell der verschiedenen Modelle in einer Datei
 - `<datei>.simplepalladio`
- ❸ Problem: zyklischen Abhängigkeiten zwischen Elementen von SystemIndependentViewType und AssemblyViewType
 - Lösung: Entkopplung des Zyklus durch Aufspaltung des SystemIndependentViewTypes und des AssemblyViewTypes in zwei Regionen
- ❹ Problem: automatische Validierung des XText \Rightarrow OCL Constraints werden automatisch ausgewertet \Rightarrow führte zu Fehlern, da OCL Constraints defekt waren

QVTo - Modelltransformation

Simple-Palladio-Beispiel
Mediastore



Palladio-Component-Model
Mediastore

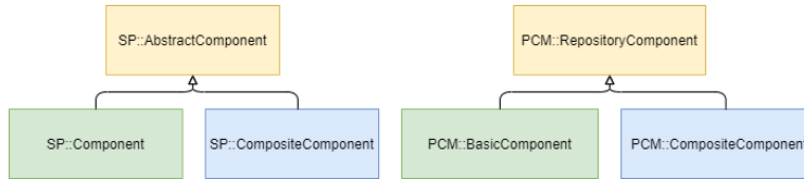
- 1 Definieren von Eingabe-Modellen, Ausgabemodellen und Transformation zwischen diesen
- 2 Probleme
 - Verschiedene Modellierungen (Enum vs. Relation vs. Objekt)
 - Vererbungshierarchien
 - Weniger-explizite Informationen in SimplePalladio

Verschiedene Modellierungen (Enum vs. Objekt)

```
mapping SIMPLESystemIndependent::Type::primitiveDataTypeMapping(in rS:Set(PCMRRepository::DataType)) : PCMRRepository::PrimitiveDataType
when{Set{SIMPLESystemIndependent::Type::VOID, SIMPLESystemIndependent::Type::BOOLEAN, SIMPLESystemIndependent::Type::CHAR,
SIMPLESystemIndependent::Type::DOUBLE, SIMPLESystemIndependent::Type::FLOAT, SIMPLESystemIndependent::Type::INT, SIMPLESystemIndependent::Type::LONG,
SIMPLESystemIndependent::Type::STRING}->includes(self)}
{
  switch {
    case(self = SIMPLESystemIndependent::Type::BOOLEAN) {result.type := PCMRRepository::PrimitiveTypeEnum::BOOL}
    case(self = SIMPLESystemIndependent::Type::CHAR) {result.type := PCMRRepository::PrimitiveTypeEnum::CHAR}
    case(self = SIMPLESystemIndependent::Type::DOUBLE or self = SIMPLESystemIndependent::Type::FLOAT) {result.type := PCMRRepository::PrimitiveTypeEnum::DOUBLE}
    case(self = SIMPLESystemIndependent::Type::INT) {result.type := PCMRRepository::PrimitiveTypeEnum::INT}
    case(self = SIMPLESystemIndependent::Type::LONG) {result.type := PCMRRepository::PrimitiveTypeEnum::LONG}
    case(self = SIMPLESystemIndependent::Type::STRING) {result.type := PCMRRepository::PrimitiveTypeEnum::STRING}
  }
}

mapping SIMPLESystemIndependent::Type::collectionDataTypeMapping(in rS:Set(PCMRRepository::DataType)) : PCMRRepository::CollectionDataType
when{Set{SIMPLESystemIndependent::Type::LIST, SIMPLESystemIndependent::Type::MAP}->includes(self)}
{
  switch {
    case(self = SIMPLESystemIndependent::Type::LIST) {
      result.entityName := 'LIST';
      result.innerType_CollectionDataType := SIMPLESystemIndependent::Type::STRING.resolveone(PCMRRepository::DataType);
    }
    case(self = SIMPLESystemIndependent::Type::MAP) {
      result.entityName := 'MAP';
      result.innerType_CollectionDataType := rS -> select(s|s.ocIsKindOf(PCMRRepository::CompositeDataType))
      ->oclAsType(PCMRRepository::CompositeDataType) -> select (s|s.entityName = 'KeyValuePair')->asSequence()->first();
    }
  }
}
```

Vererbungshierarchien



```

abstract mapping SIMPLESystemIndependent::AbstractComponent::abstractComponentMapping() : PCMRepository::RepositoryComponent
disjuncts
SIMPLESystemIndependent::Component::basicComponentMapping,
SIMPLESystemIndependent::CompositeComponent::compositeComponentMapping {}
    
```

```

mapping SIMPLESystemIndependent::Component::basicComponentMapping() : PCMRepository::BasicComponent{
    
```

```

mapping SIMPLESystemIndependent::CompositeComponent::compositeComponentMapping() : PCMRepository::CompositeComponent{
    
```

Weniger-explicite Informationen

1 Lösung → Intermediate Objekte einführen und diese mappen

```

intermediate class TRole {
    communicator:SIMPLESystemIndependent::InterfaceCommunicator;
    interface:SIMPLESystemIndependent::Interface;
};

mapping SIMPLESystemIndependent::Component::basicComponentMapping() : PCMRepository::BasicComponent{
    result.entityName := self.name;
    result.providedRoles_InterfaceProvidingEntity := self.providedInterfaces->
        collect(i| object TRole {
            interface := i;
            communicator := self;
        }).map provRoleMapping();
    result.requiredRoles_InterfaceRequiringEntity := self.requiredInterfaces->
        collect(i| object TRole {
            interface := i;
            communicator := self;
        }).map reqRoleMapping();
    result.serviceEffectSpecifications__BasicComponent := self.services->collect(srv|srv->map serviceMapping(result));
}

mapping TRole::provRoleMapping() : PCMRepository::OperationProvidedRole{
    init{
        var x := self.communicator.oclAsType(SIMPLECommon::NamedElement);
    }
    result.entityName := x.name.concat('_').concat(self.interface.name).concat('_provider');
    result.providedInterface__OperationProvidedRole := self.interface.resolveone(PCMRepository::OperationInterface);
}

```

Code-Generierung mit Xtend

```
def dispatch void generate(Component c, IFileSystemAccess fsa) {
    fsa.generateFile(c.name + '/' + c.name+"Impl.java", '''
package «c.name»;

«FOR i: c.providedInterfaces»import repository.«i.name»;«ENDFOR»
«FOR i: c.requiredInterfaces.filter[i] !c.providedInterfaces.contains(i)]»import repository.«i.name»;«ENDFOR»
«IF !c.requiredInterfaces.empty»import repository.Helper;«ENDIF»

public class «c.name»Impl implements «FOR i: c.providedInterfaces SEPARATOR ',' «i.name» «ENDFOR» {
    «FOR i: c.requiredInterfaces» private «i.name» «i.name.toFirstLower();»«ENDFOR»
    public «c.name»Impl() {}
    «FOR i: c.requiredInterfaces»
        public void set«i.name»(«i.name» «i.name.toFirstLower();») {
            Helper.assertNull(this, «i.name.toFirstLower();»);
            this.«i.name.toFirstLower()» = «i.name.toFirstLower();»
        }
    «ENDFOR»

    «FOR i: c.providedInterfaces»
        «FOR s: i.signatures»
            //Implementing «s.name» from interface «i.name»
            @Override
            public «s.generate();» {
                «FOR ireq: c.requiredInterfaces»
                    Helper.assertNotNull(this, «ireq.name.toFirstLower();»);
                «ENDFOR»
                //TODO: implement
            }
        «ENDFOR»
    «ENDFOR»
}
'''
}
```

Problem mit Xtend und OCL

- ❶ Xtend-Validator kann OCL Constraints nicht einlesen und validieren
- ❷ Vorrübergehende Lösung = *NullValidator*

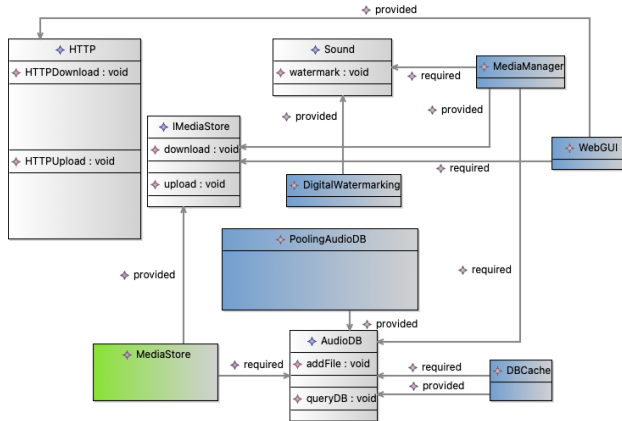
```
component = repository.RepositoryGeneratorSupport {}

component = org.eclipse.xtext.mwe.Reader {
    validate = NullValidator {}
    path = modelPath
    register = repository.RepositoryGeneratorSetup {}
    loadResource = {
        slot = "model"
    }
}

component = org.eclipse.xtext.generator.GeneratorComponent {
```

- Vorteil: Model kann eingelesen und Java generiert werden
- Nachteil: auch nicht-valide Modelle werden akzeptiert und transpiliert

Sirius - grafischer Editor



- graues Element: Interface
- grünes Element: Composite Component
- blaues Element: Component
- required Edge: z.B. Component required Interface
- provided Edge: z.B. Component provided Interface

M1: Metamodell
○○○

M2: Xtext
○○

M3: QVTo
○○○○

M4: Xtend
○○

M5: Sirius
●○○

Key Learnings
○○

Entwurfsentscheidungen beim grafischen Editor

- ① Verwendung des grafischen Editors, um die Verwendung des Meta-Modells so intuitiv wie möglich zu gestalten
 - Verwendung von üblichen Farben, Symbolen, Icons ermöglicht die Modellierung von
- ② Parameter werden als Liste innerhalb von Signaturblöcken dargestellt

Probleme beim grafischen Editor

- 1 Kontextwechsel muss man wissen
 - Beispiel: Signaturen gehören bei Sirius zu Interfaces → wenn Interface gelöscht wird, dann ist Signatur weg, in Meta-Modell ist es weiterhin da
- 2 Parameterlisten mit Klammern schwer umsetzbar, da sie nicht editierbar wäre
- 3 eigenständige Datentypen als Parameter

Key Learnings (1)

- ① OCL wurde am Anfang wenig getestet und hat dann zu Problemen im weiteren Verlauf der Meilensteine geführt
- ② Unterscheidung zwischen einfacher Modellierung und OCL hat unterschiedliche Vor- und Nachteile gegenüber von Modellierung über komplexeren Klassenstrukturen
 - kleinere Klassenstrukturen, jedoch OCL muss getestet werden
 - geringere Konsistenz-Haltung durch mehr Klassen und Attribute und weniger OCL
- ③ Unterschiedliche Form von Modellierungskonzepten
 - Beispiel: Modelltransformation (siehe Probleme Modelltransformation)
- ④ Informationsbeschaffung durch Literatur und Foren im Internet ist nur schwierig möglich → wenig vorhanden und wenn alte outdated Forumseinträge

Key Learnings (2)

- ① Interpreter in Eclipse: OCL-Ausdruck eingeben und zeigt die Werte an
- ② Vorsichtig sein mit bidirektionalen Referenzen:
 - nützlich bei OCL und Sirius → leichter um zwischen Kontexten zu wechseln (z.B. beim Löschen kann man in beide Richtungen gehen)
 - Nachteilig bei Xtext → jedoch Lösung durch Sandwicking