Karlsruher Institut für Technologie

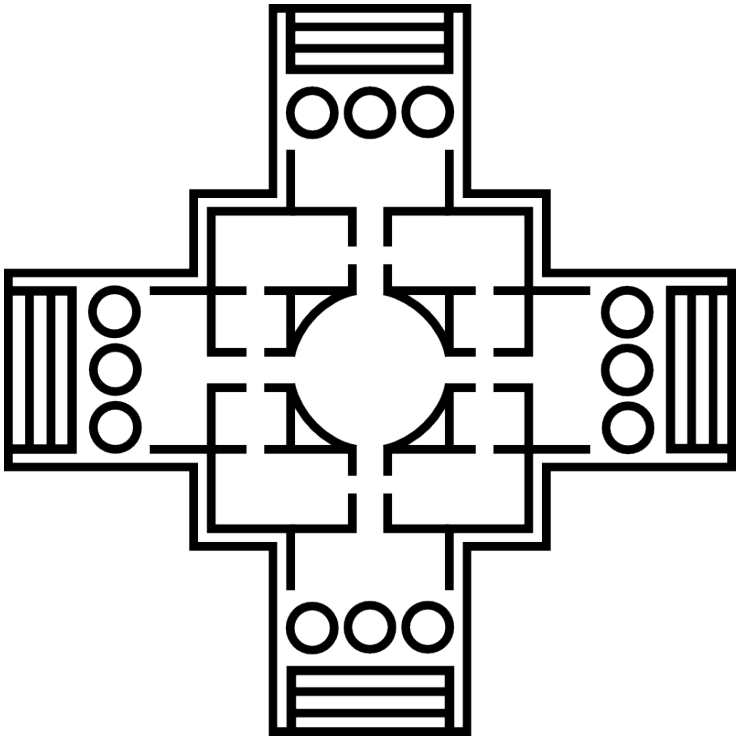
Palladio Fluent API Model Generator

Praktikum: Werkzeuge für Agile Modellierung

ARCHITECTURE-DRIVEN REQUIREMENTS ENGINEERING,
INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION, KIT-FAKULTÄT FÜR INFORMATIK

Inhalt

- Hintergrund
  - Palladio
  - Palladio Repository Model
  - Fluent Interfaces
- Das Projekt: Palladio Fluent API Model Generator
  - Ergebnis



PALLADIO

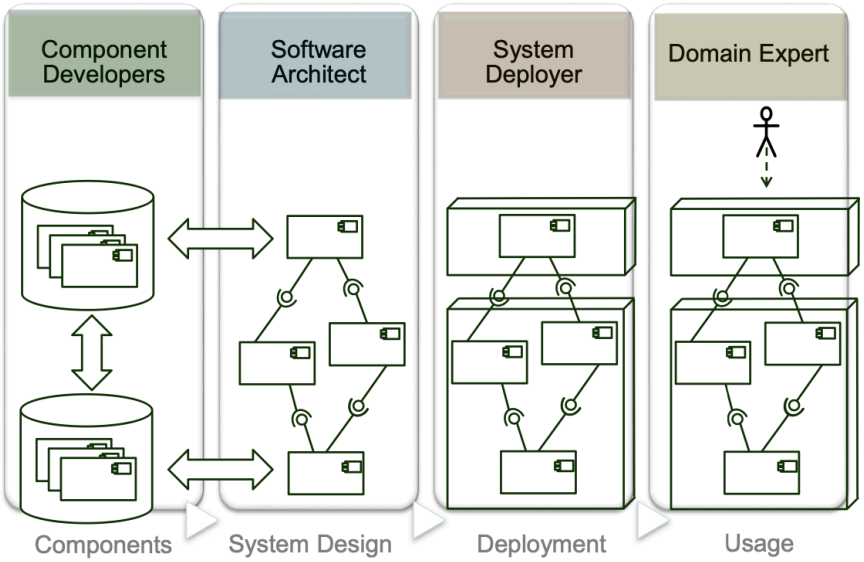
Palladio

- Tool für die Simulation von Software-Architektur
- Analyse und Vorhersage von qualitativen Eigenschaften der Software bezüglich:
  - Performance Engpässe
  - Skalierbarkeit
  - Verlässlichkeit
  - Wartbarkeit
  - Kosten
- Kern des Palladio Ansatzes: Palladio Component Model (PCM)

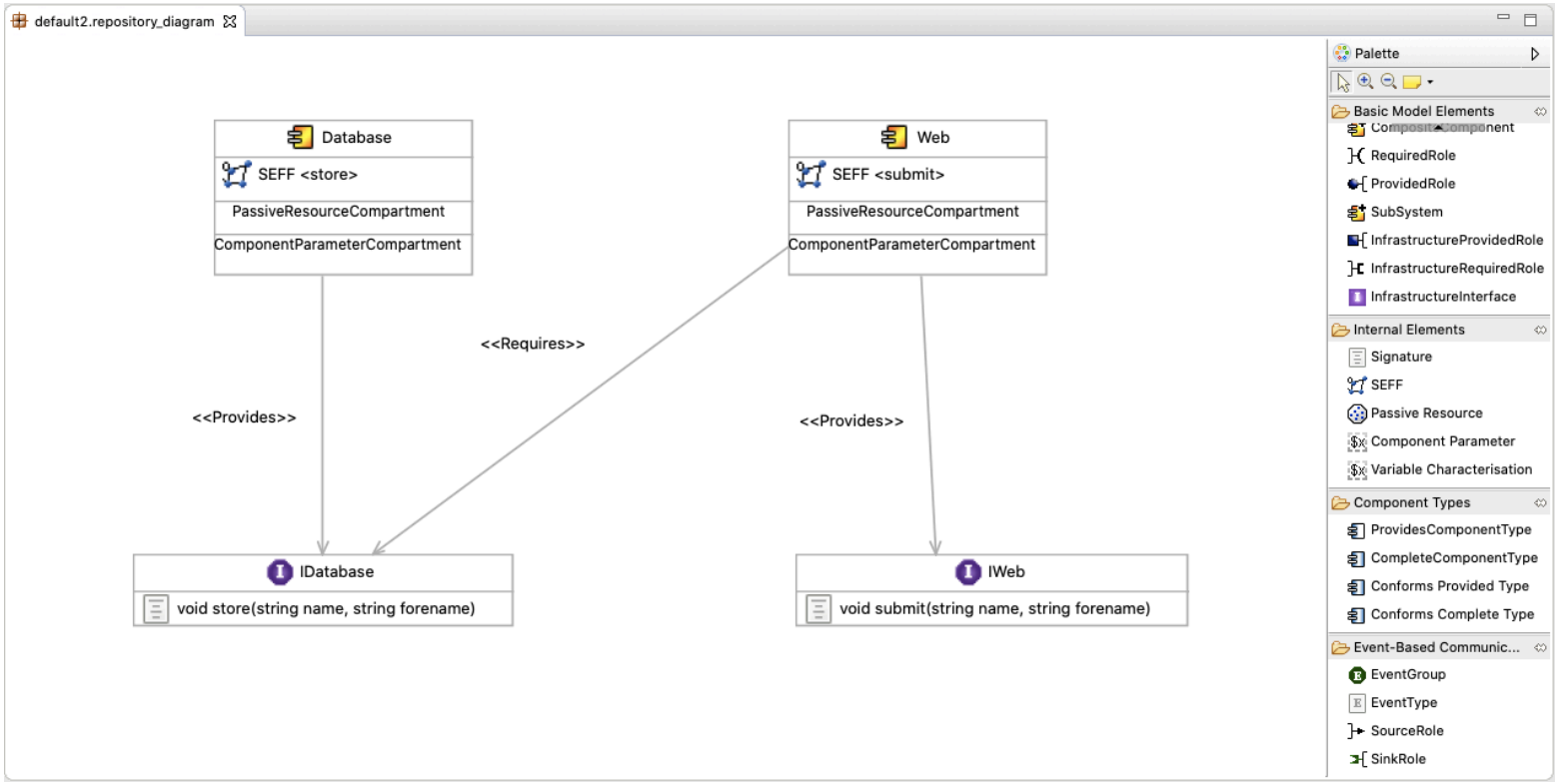
Palladio Component Model (PCM)

- ist eine Domänenspezifische Modellierungssprache (DSL)

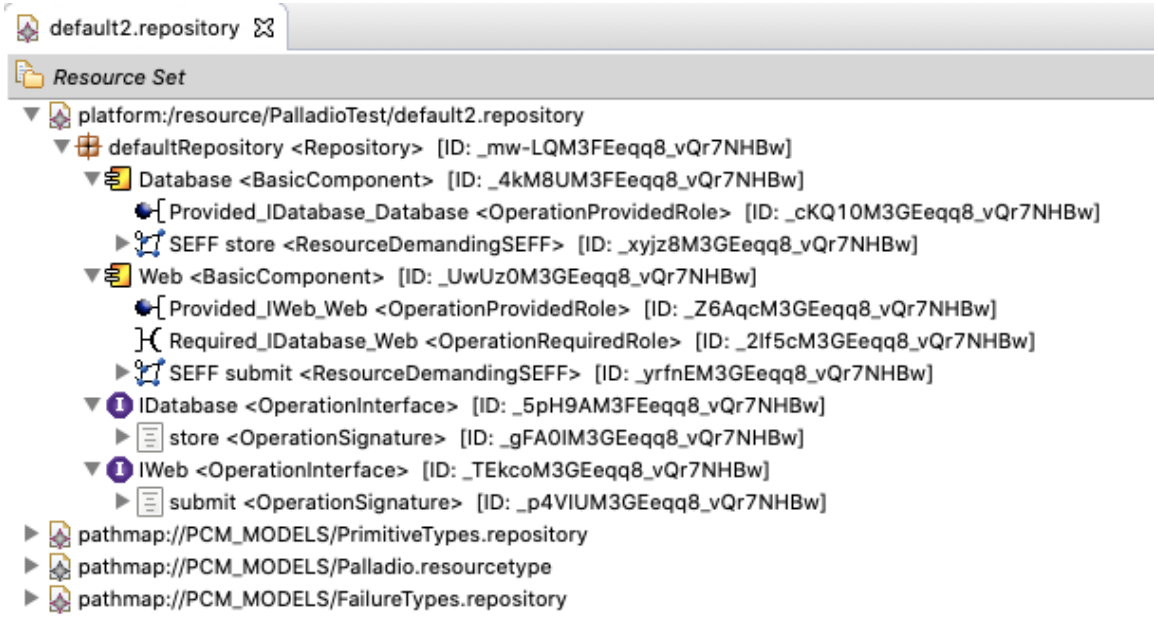
- Erstellung von PCM Instanzen
  - Repository model
  - Usage model
  - System model
  - Resource environment model



PALLADIO REPOSITORY MODEL



Erstellung einer Palladio Repository Model Instanz mit Hilfe des graphischen Editors



Palladio Repository Model

Baumansicht

Ein Design Prinzip

FLUENT INTERFACES

Fluent Interfaces

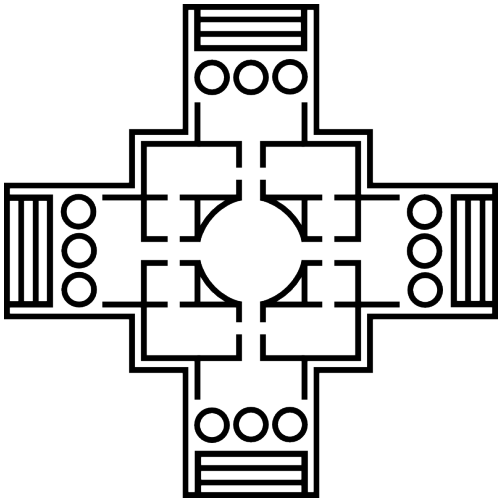
- Besonders nützlich, um Objekte zu erstellen und zu manipulieren
- Ziel: lesbarer Code
- Methodenverkettung
  - die sich wie ein natürlich-sprachlicher Satz liest
- Gibt einen Rahmen vor und bietet eine natürliche Intuition für verfügbare Features
- Beispiele: Java Stream API, JMock

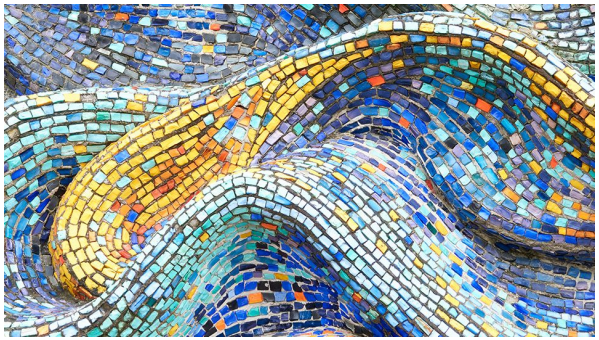
Fluent Interfaces: Beispiel

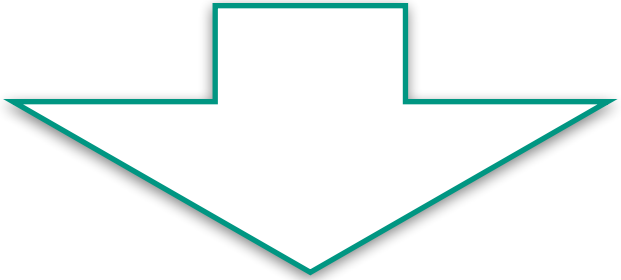
```
private void makeNormal(Customer customer) {
        Order o1 = new Order();
        customer.addOrder(o1);
        OrderLine line1 = new OrderLine(6,
Product.find("TAL"));
        o1.addLine(line1);
        OrderLine line2 = new OrderLine(5,
Product.find("HPK"));
        o1.addLine(line2);
        OrderLine line3 = new OrderLine(3,
Product.find("LGV"));
        o1.addLine(line3);
        line2.setSkippable(true);
        o1.setRush(true);
    }
```

```
private void makeFluent(Customer customer) {
        customer.newOrder()
                .with(6, "TAL")
                .with(5, "HPK").skippable()
                .with(3, "LGV")
                .priorityRush();
    }
```

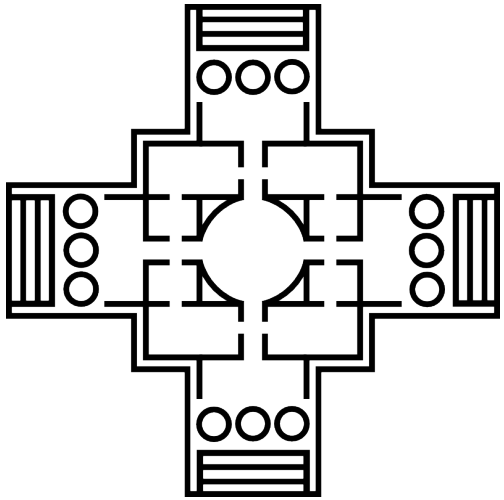
Quelle: https://martinfowler.com/bliki/FluentInterface.html (Martin Fowler)

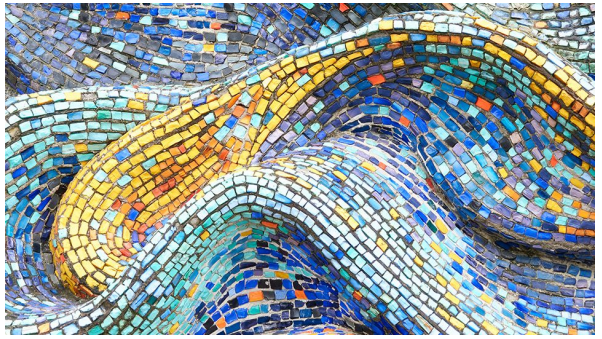


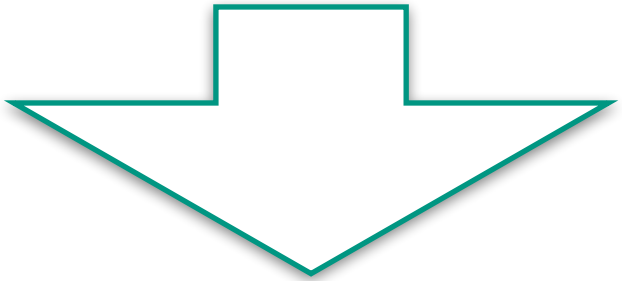




PALLADIO FLUENT API MODEL GENERATOR







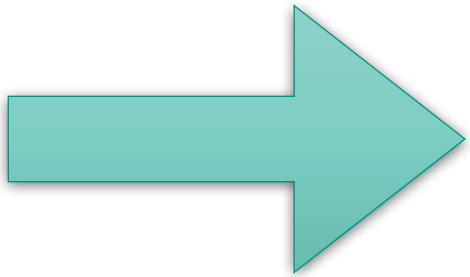
PALLADIO FLUENT API MODEL GENERATOR

Recherche

- Fluent Interfaces mit Builder Pattern
  - Erstellen einer Grammatik
- Kleiner Entwurf mit Basic Component

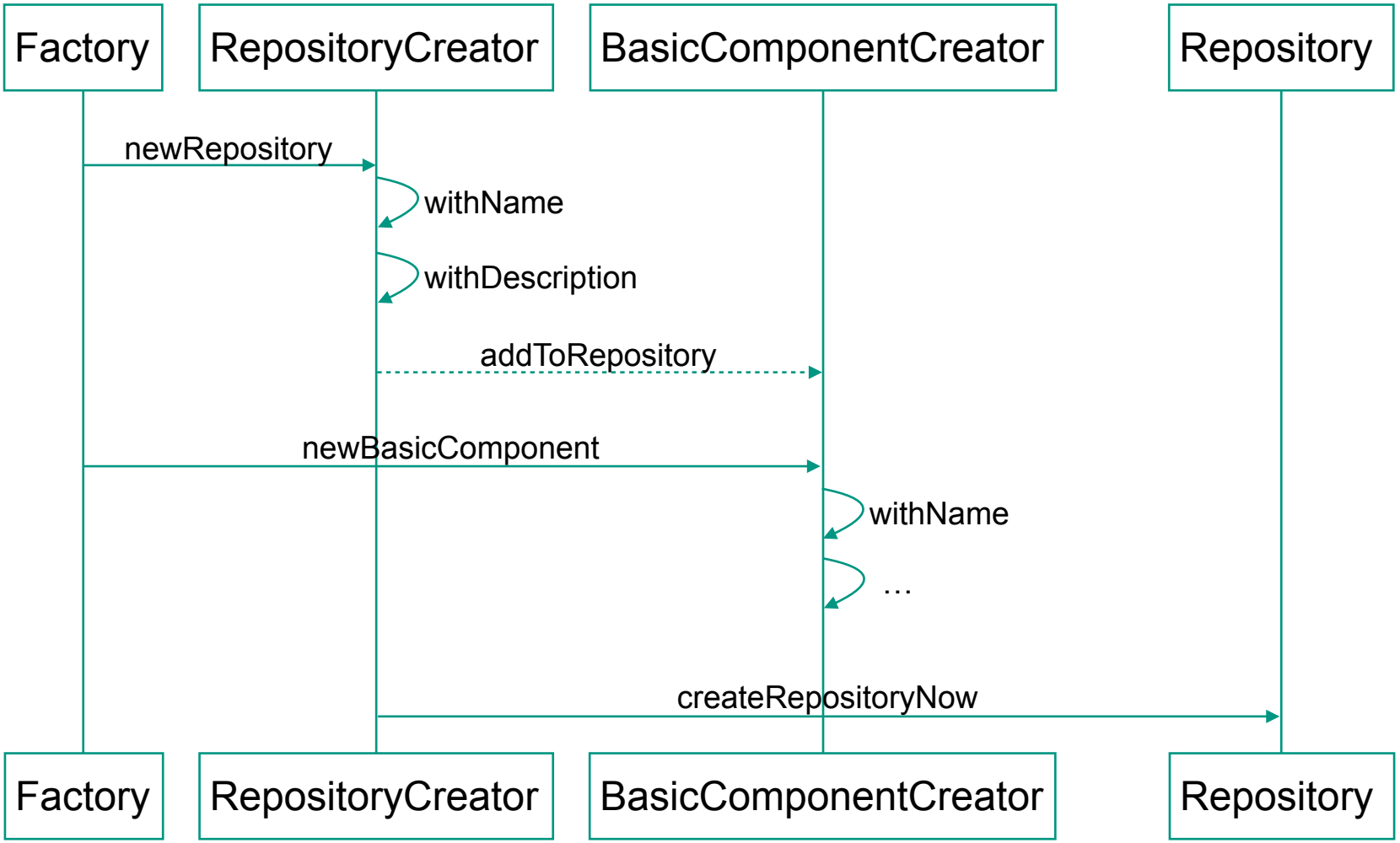
Herangehensweise

“Create a new repository model. Add to this repository a basic component with name ‘Database‘. The basic component ‘Database‘ provides an interface with name ‘IDatabase‘. This interface has a signature called ‘store‘ with the parameters ‘name‘ and ‘forename‘. …“



```
184         FluentRepositoryFactory create = new FluentRepositoryFactory();
185
186         Repository repository = create.newRepository()
187                 .addToRepository(create.newBasicComponent()
188                         .withName("Database")
189                         .provides(create.newOperationInterface()
190                                 .withName("IDatabase")
191                                 .withOperationSignature(create.newOperationSignature()
192                                         .withName("store")
193                                         .withParameter("forename", Primitive.STRING, ParameterModifier.NONE)
194                                         .withParameter("name", Primitive.STRING, ParameterModifier.NONE))))
195                 .createRepositoryNow();
```

Fluent Interface Grammatik



Methodenverkettung
Methodenverschachtelung

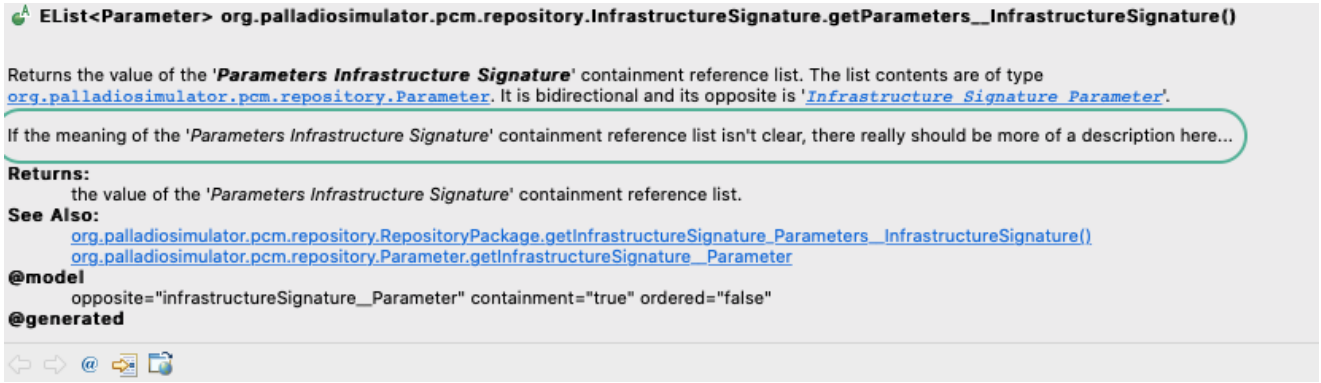
Recherche

- Fluent Interfaces mit Builder Pattern
  - Erstellen einer Grammatik
- Kleiner Entwurf mit Basic Component
- Methodenverkettung vs. Methodenverschachtelung
- PCM Backend
  - Was bietet der graphische Editor?
  - Was bietet die Baumansicht?
  - Was implementiert das Referenzprojekt?
  - Welche Factorys bietet das PCM Backend?

→ Vollständige Implementierung

Herausforderungen

- Dokumentation des Palladio Projekts



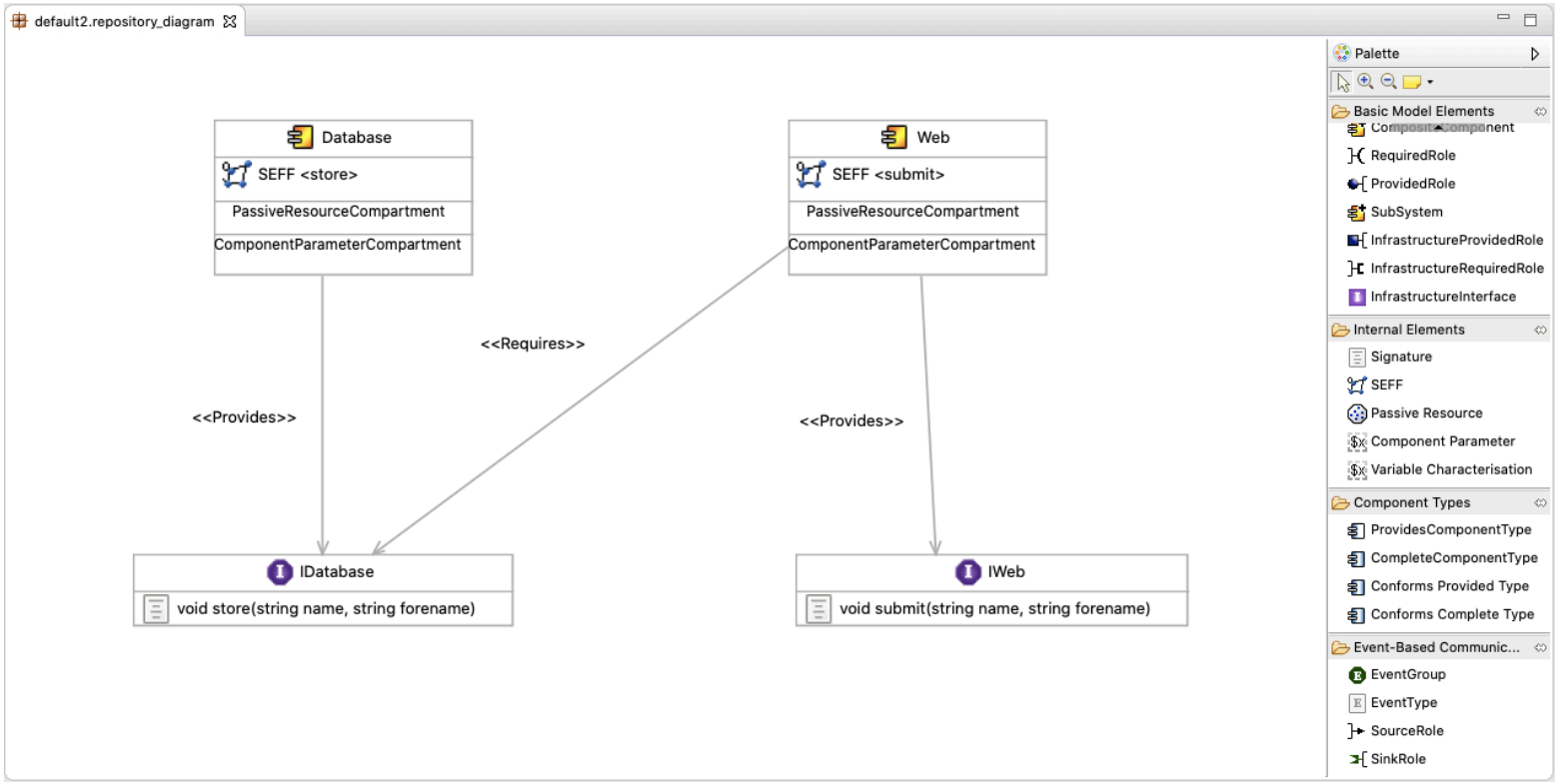
  - Welche Modell-Elemente gibt es und in welcher Verbindung stehen sie zu anderen Modell-Elementen?

- Design Entscheidungen
  - Welche Modell-Elemente sollten über die Factory erstellt werden?
  - Wie stark sollte die Reihenfolge der Methodenaufrufe festgelegt werden?
  - Sollte man Methoden, die Werte setzen und nicht hinzufügen, überhaupt mehrfach aufrufen können?

ERGEBNIS

Aufbau der FluentRepositoryFactory

- Die Factory erstellt
  - das Repository
- und folgende Elemente, die dem Repository hinzugefügt werden
  - Components
  - Interfaces
  - Data types
  - Failure types
- Interne Elemente, die ebenfalls über die Factory erstellt werden
  - SEFFs
  - Variable usages
  - Signatures
- Die Factory ermöglicht den Zugriff auf alle bereits erstellten Entitäten



Erstellung einer Palladio Repository Model Instanz mit Hilfe des graphischen Editors

```java
public static void exampleFluentAPI() {
    // Factory
    FluentRepositoryFactory create = new FluentRepositoryFactory();

    Repository repository = create.newRepository()
            // Database
            .addToRepository(create.newOperationInterface().withName("IDatabase")
                    .withOperationSignature(create.newOperationSignature()
                            .withName("store")
                            .withParameter("forename", Primitive.STRING, ParameterModifier.NONE)
                            .withParameter("name", Primitive.STRING, ParameterModifier.NONE)))
            .addToRepository(create.newBasicComponent().withName("Database")
                    .withServiceEffectSpecification(create.newSeff().onSignature(create.fetchOfSignature("store")))
                    .provides(create.fetchOfOperationInterface("IDatabase")))
            // Web
            .addToRepository(create.newOperationInterface().withName("IWeb")
                    .withOperationSignature(create.newOperationSignature()
                    .withName("submit").withParameter("forename", Primitive.STRING, ParameterModifier.NONE)
                    .withParameter("name", Primitive.STRING, ParameterModifier.NONE)))
            .addToRepository(create.newBasicComponent().withName("Web")
                    .withServiceEffectSpecification(create.newSeff().onSignature(create.fetchOfSignature("submit")))
                    .provides(create.fetchOfOperationInterface("IWeb"))
                    .requires(create.fetchOfOperationInterface("IDatabase")))
            .createRepositoryNow();

    saveRepository(repository, "./", "fluentAPIExample.repository", false);
}
```

Ergebnis

Beispiel: Web und Database Komponenten mit ihren Schnittstellen

Ergebnis: Palladio Fluent API Model Generator

- Fluent API
  - zur Erstellung von Palladio Repository Objekten
  - mit nur einer Schnittstelle/Factory
  - mit einem klaren Rahmen, welche Elemente, wo hinzugefügt und/ oder verbunden werden können
  - => auch für Einsteiger geeignet
- Vollständige Dokumentation (JavaDoc)
- Umfassende Projektbeschreibung und Einführung mit Beispielen (Readme)

```java
public static void readmeExampleBackend() {
    // Factory
    RepositoryFactory repoFact = RepositoryFactory.eINSTANCE;
    // Repository
    Repository repository = repoFact.createRepository();

    // Database component
    BasicComponent databaseComponent = repoFact.createBasicComponent();
    databaseComponent.setEntityName("Database");

    // IDatabase interface
    OperationInterface databaseInterface = repoFact.createOperationInterface();
    databaseInterface.setEntityName("IDatabase");

    // Signature store
    OperationSignature store = repoFact.createOperationSignature();
    store.setEntityName("store");
    // with parameters forename, name
    Parameter forename = repoFact.createParameter();
    forename.setParameterName("forename");
    forename.setDataType__Parameter(null); // referencing the imported data types poses another problem
    Parameter name = repoFact.createParameter();
    name.setParameterName("forename");
    name.setDataType__Parameter(null);

    // Providing connection from Database component to IDatabase interface
    OperationProvidedRole dbProvIDb = repoFact.createOperationProvidedRole();
    dbProvIDb.setProvidedInterface__OperationProvidedRole(databaseInterface);
    dbProvIDb.setProvidingEntity_ProvidedRole(databaseComponent);

    // Seff for Database component on service store
    ResourceDemandingSEFF storeSeff = SeffFactory.eINSTANCE.createResourceDemandingSEFF();
    storeSeff.setDescribedService__SEFF(store);
    databaseComponent.getServiceEffectSpecifications__BasicComponent().add(storeSeff);

    // Adding component + interfaces to the repository
    repository.getComponents__Repository().add(databaseComponent);
    repository.getInterfaces__Repository().add(databaseInterface);

    // Web component
    BasicComponent webComponent = repoFact.createBasicComponent();
    databaseComponent.setEntityName("Web");

    OperationInterface webInterface = repoFact.createOperationInterface();
    databaseInterface.setEntityName("IWeb");

    OperationSignature submit = repoFact.createOperationSignature();
    submit.setEntityName("submit");
    // with parameters forename, name
    Parameter forename2 = repoFact.createParameter();
    forename2.setParameterName("forename");
    forename2.setDataType__Parameter(null);
    Parameter name2 = repoFact.createParameter();
    name2.setParameterName("forename");
    name2.setDataType__Parameter(null);

    OperationProvidedRole webProvIweb = repoFact.createOperationProvidedRole();
```

FRAGEN?