

# Java-Schnittstelle für die Instanziierung von EMF-Modellen

**Praktikum: Werkzeuge für Agile Modellierung**

**Louisa Lambrecht**

**Betreuer: Yves Richard Kirschner**

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



# Inhalt

- Motivation

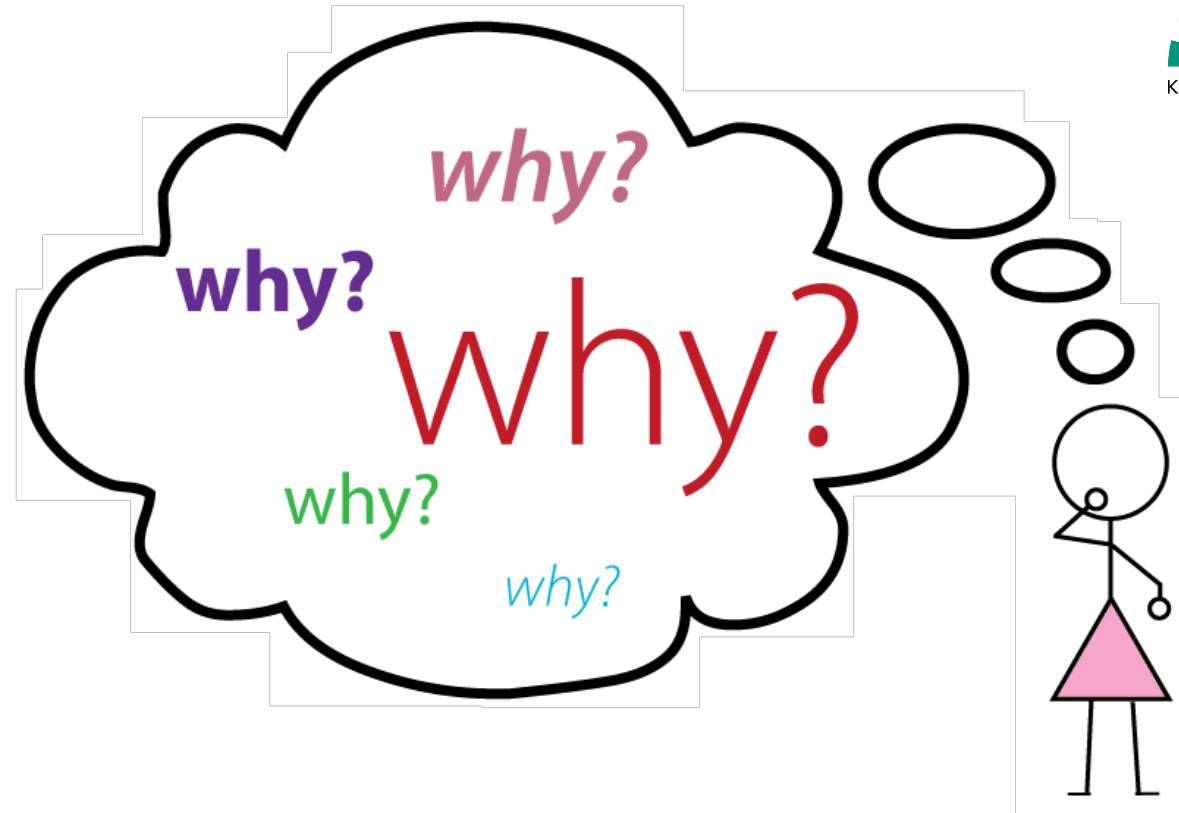
- Grundlagen

  - Palladio Repository Model

  - Fluent Interfaces

- Palladio Fluent API Model Generator

- Ergebnis

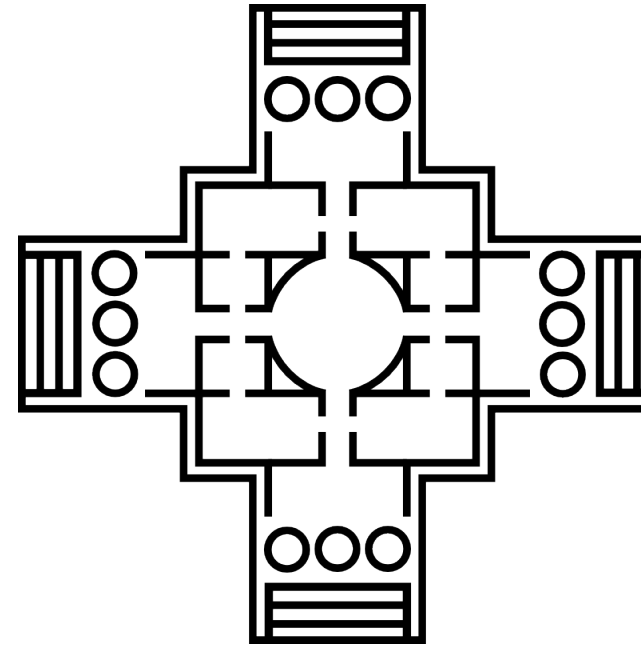


# MOTIVATION

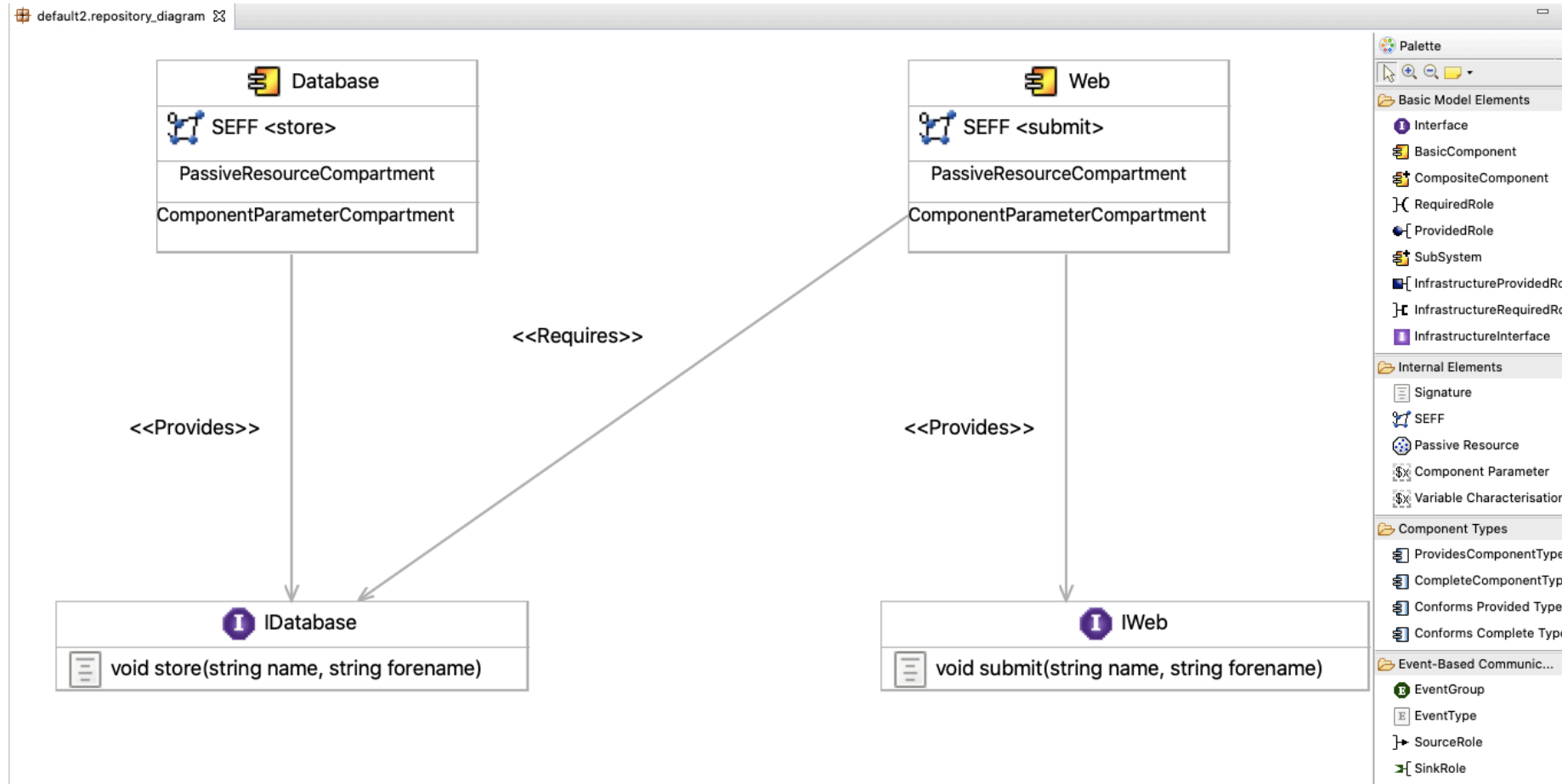
# Motivation

- Graphische Editoren sind
  - für Einsteiger geeignet
  - anschaulich
  - bei großen Modellen schnell unübersichtlich
  - ermüdend durch das „Herumklicken“
- Komplexe Objekte programmatisch zu erzeugen ist
  - unübersichtlich
  - unverständlich
  - schreibintensiv
  - schneller



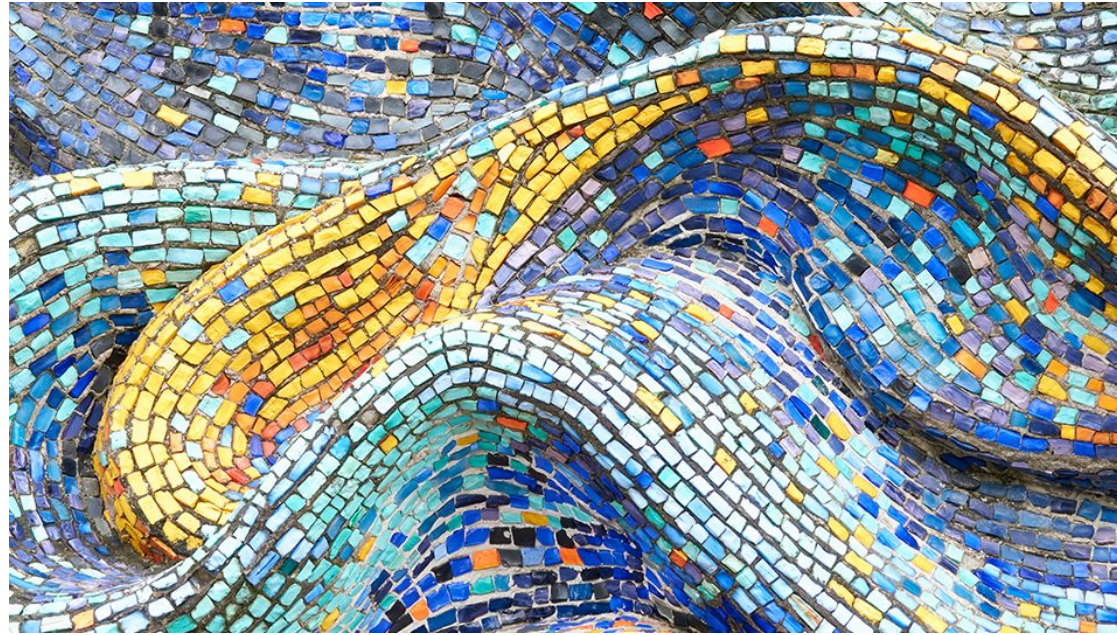


# PALLADIO REPOSITORY MODEL



# Erstellung einer Palladio Repository Model Instanz mit Hilfe des graphischen Editors

Ein Beispiel



Ein Design Prinzip

# FLUENT INTERFACES

# Fluent Interfaces

- Besonders nützlich, um Objekte zu erstellen und zu manipulieren
- Ziel: lesbarer, kompakter 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

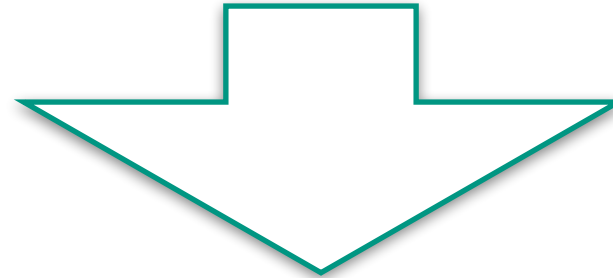
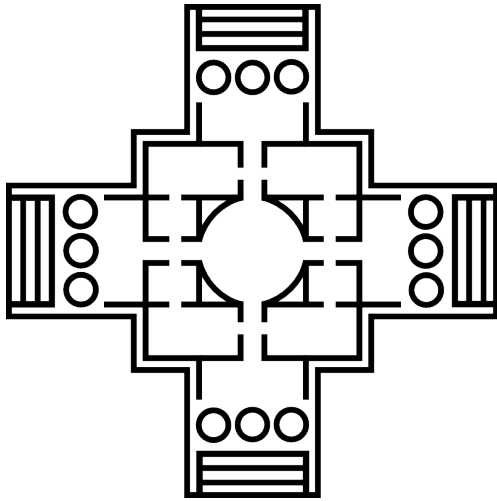


# Fluent Interfaces: Beispiel

```
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();  
}
```

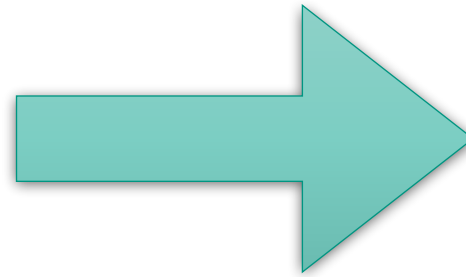
Quelle: <https://martinfowler.com/bliki/FluentInterface.html> (Martin Fowler)



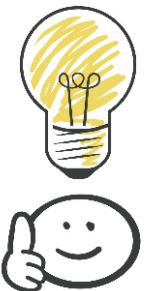
# PALLADIO FLUENT API MODEL GENERATOR

# Herangehensweise

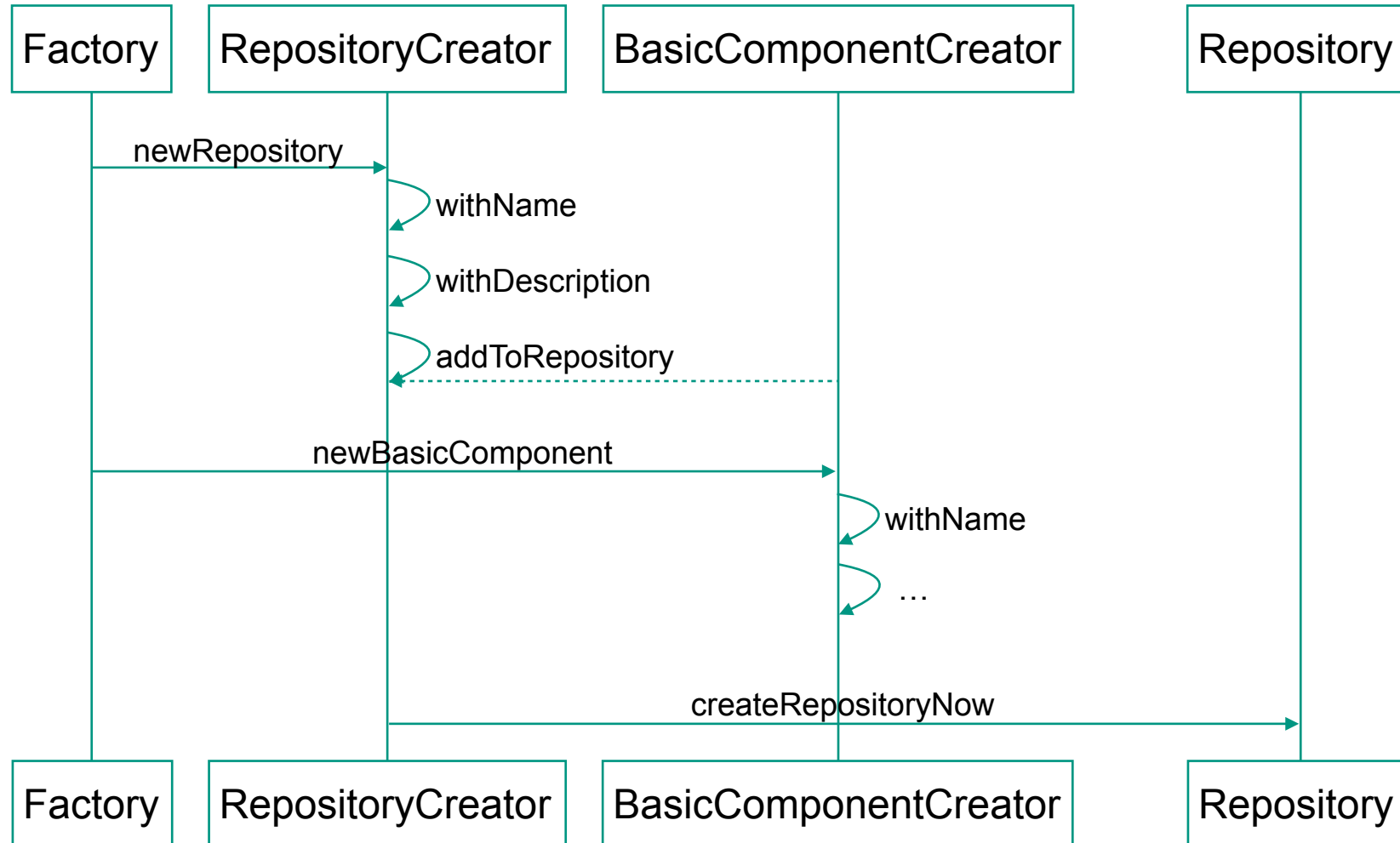
“Create a new repository model. Add to this repository a basic component with name ‘Database’.”



```
184  
185   FluentRepositoryFactory create = new FluentRepositoryFactory();  
186  
187   Repository repository = create.newRepository()  
188       .addToRepository(create.newBasicComponent()  
189           .withName("Database"))  
190       .createRepositoryNow();  
191
```



# Fluent Interface Grammatik



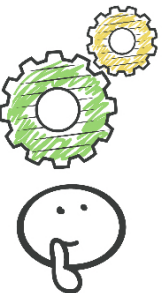
Methodenverkettung:

`a().b().c()`



Argument:

`a(x)`



# Herausforderungen

## ■ Verständnis des Meta-Models

```
EList<Parameter> org.palladiosimulator.pcm.repository.InfrastructureSignature.getParameters__InfrastructureSignature()
```

Returns the value of the '**Parameters Infrastructure Signature**' containment reference list. The list contents are of type [org.palladiosimulator.pcm.repository.Parameter](#). It is bidirectional and its opposite is '[Infrastructure Signature Parameter](#)'.

If the meaning of the '*Parameters Infrastructure Signature*' containment reference list isn't clear, there really should be more of a description here...

**Returns:**  
the value of the '*Parameters Infrastructure Signature*' containment reference list.

**See Also:**  
[org.palladiosimulator.pcm.repository.RepositoryPackage.getInfrastructureSignature\\_Parameters\\_\\_InfrastructureSignature\(\)](#)  
[org.palladiosimulator.pcm.repository.Parameter.getInfrastructureSignature\\_\\_Parameter](#)

**@model**  
opposite="infrastructureSignature\_\_Parameter" containment="true" ordered="false"

**@generated**

## ■ Design Entscheidungen

- Erstellen der Modell-Elemente: Factory vs. Methodenverkettung
- Reihenfolge der Methodenverkettung: strikt vs. frei
- Methoden dürfen gesetzte Werte überschreiben

# ERGEBNIS



# Aufbau der FluentRepositoryFactory

- Die Factory erstellt
  - das Repository
  - Hauptelemente: Components, Interfaces, Data Types, Failure Types
  - komplexe interne Elemente: SEFFs, Signatures, Variabel Usages
- Die Factory ermöglicht den Zugriff auf alle bereits erstellten Entitäten

# Palladio Fluent API Model Generator

## ■ Fluent API

- zur Erstellung von Palladio Repository Objekten
- mit nur einer Schnittstelle/Factory
- mit einem klaren Rahmen der möglichen Eigenschaften durch die entwickelte Grammatik
  - auch für Einsteiger geeignet

## ■ Vollständige Dokumentation (JavaDoc)

## ■ Umfassende Projektbeschreibung und Einführung mit Beispielen (Readme)



```

80 public static void readmeExampleBackend() {
81     // Factory
82     RepositoryFactory repoFact = RepositoryFactory.eINSTANCE;
83     // Repository
84     Repository repository = repoFact.createRepository();
85
86     // Database component
87     BasicComponent databaseComponent = repoFact.createBasicComponent();
88     databaseComponent.setEntityName("Database");
89
90     // IDatabase interface
91     OperationInterface databaseInterface = repoFact.createOperationInterface();
92     databaseInterface.setEntityName("IDatabase");
93
94     // Signature store
95     OperationSignature store = repoFact.createOperationSignature();
96     store.setEntityName("store");
97     // with parameters forename, name
98     Parameter forename = repoFact.createParameter();
99     forename.setParameterName("forename");
100    forename.setDataType__Parameter(null); // referencing the imported data types poses another problem
101    Parameter name = repoFact.createParameter();
102    name.setParameterName("forename");
103    name.setDataType__Parameter(null);
104
105    // Providing connection from Database component to IDatabase interface
106    OperationProvidedRole dbProvIDb = repoFact.createOperationProvidedRole();
107    dbProvIDb.setProvidedInterface__OperationProvidedRole(databaseInterface);
108    dbProvIDb.setProvidingEntity__ProvidedRole(databaseComponent);
109
110    // Seff for Database component on service store
111    ResourceDemandingSEFF storeSeff = SeffFactory.eINSTANCE.createResourceDemandingSEFF();
112    storeSeff.setDescribedService__SEFF(store);
113    databaseComponent.getServiceEffectSpecifications__BasicComponent().add(storeSeff);
114
115    // Adding component + interfaces to the repository
116    repository.getComponents__Repository().add(databaseComponent);
117    repository.getInterfaces__Repository().add(databaseInterface);
118
119    // Web component
120    BasicComponent webComponent = repoFact.createBasicComponent();
121    databaseComponent.setEntityName("Web");
122
123    OperationInterface webInterface = repoFact.createOperationInterface();
124    databaseInterface.setEntityName("IWeb");
125
126    OperationSignature submit = repoFact.createOperationSignature();
127    submit.setEntityName("submit");
128    // with parameters forename, name
129    Parameter forename2 = repoFact.createParameter();
130    forename2.setParameterName("forename");
131    forename2.setDataType__Parameter(null);
132    Parameter name2 = repoFact.createParameter();
133    name2.setParameterName("forename");
134    name2.setDataType__Parameter(null);
135
136    OperationProvidedRole webProvIweb = repoFact.createOperationProvidedRole();
137    webProvIweb.setProvidedInterface__OperationProvidedRole(webInterface);

```

```

155 public static void readmeExampleFluentAPI() {
156     // Factory
157     FluentRepositoryFactory create = new FluentRepositoryFactory();
158
159     Repository repository = create.newRepository()
160         // Database
161         .addToRepository(create.newOperationInterface().withName("IDatabase")
162             .withOperationSignature(create.newOperationSignature()
163                 .withName("store")
164                 .withParameter("forename", Primitive.STRING, ParameterModifier.NONE)
165                 .withParameter("name", Primitive.STRING, ParameterModifier.NONE)))
166         .addToRepository(create.newBasicComponent().withName("Database")
167             .withServiceEffectSpecification(create.newSeff().onSignature(create.fetchOfSignature("store")))
168             .provides(create.fetchOfOperationInterface("IDatabase")))
169         // Web
170         .addToRepository(create.newOperationInterface().withName("IWeb")
171             .withOperationSignature(create.newOperationSignature()
172                 .withName("submit").withParameter("forename", Primitive.STRING, ParameterModifier.NONE)
173                 .withParameter("name", Primitive.STRING, ParameterModifier.NONE)))
174         .addToRepository(create.newBasicComponent().withName("Web")
175             .withServiceEffectSpecification(create.newSeff().onSignature(create.fetchOfSignature("submit")))
176             .provides(create.fetchOfOperationInterface("IWeb"))
177             .requires(create.fetchOfOperationInterface("IDatabase")))
178         .createRepositoryNow();
179
180     saveRepository(repository, "./", "fluentAPIExample.repository", false);
181 }

```

# Palladio Fluent API Model Generator

Beispiel: Web und Database Komponenten mit ihren Schnittstellen



# FRAGEN?