



Preprocessing Plantestic

Jorge Quintero, Tobias Schmidt, Paula Wikidal, Fabian Wildgrube
Modellbasierte Softwareentwicklung, SoSe 2020

Recap: Plantestic



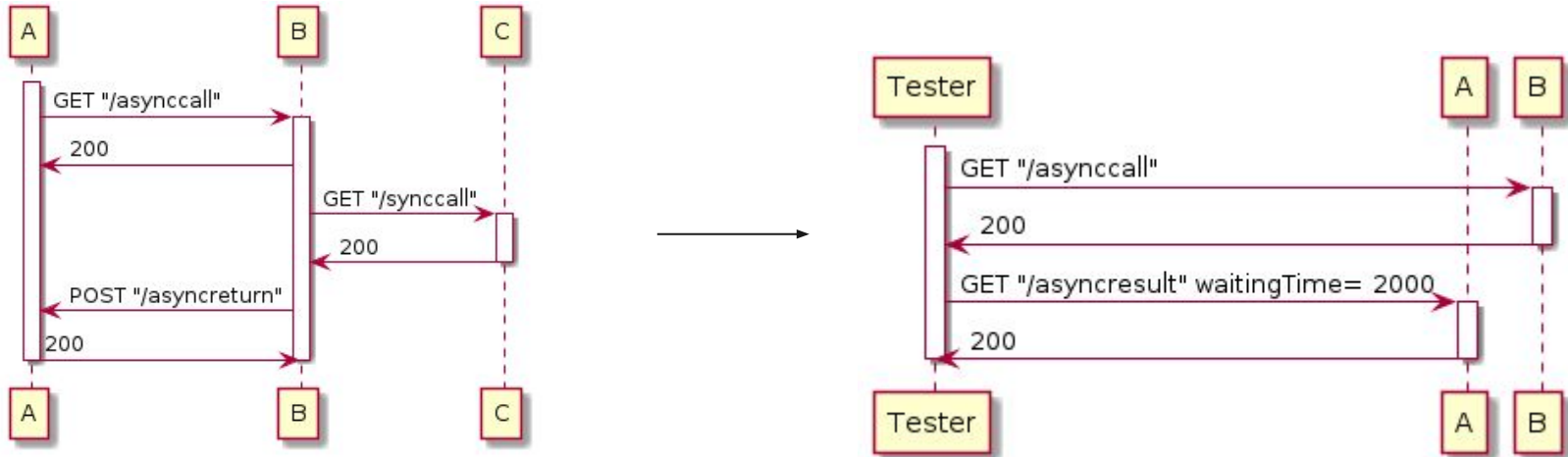
Features:

- Testfallspezifikation mittels PlantUML
- Modelltransformation mittels XText und QVTo
- Testfallgenerierung mittels Acceleo
- Java Testfälle mit REST Assured
- Condition-Evaluierung mittels JavaScript

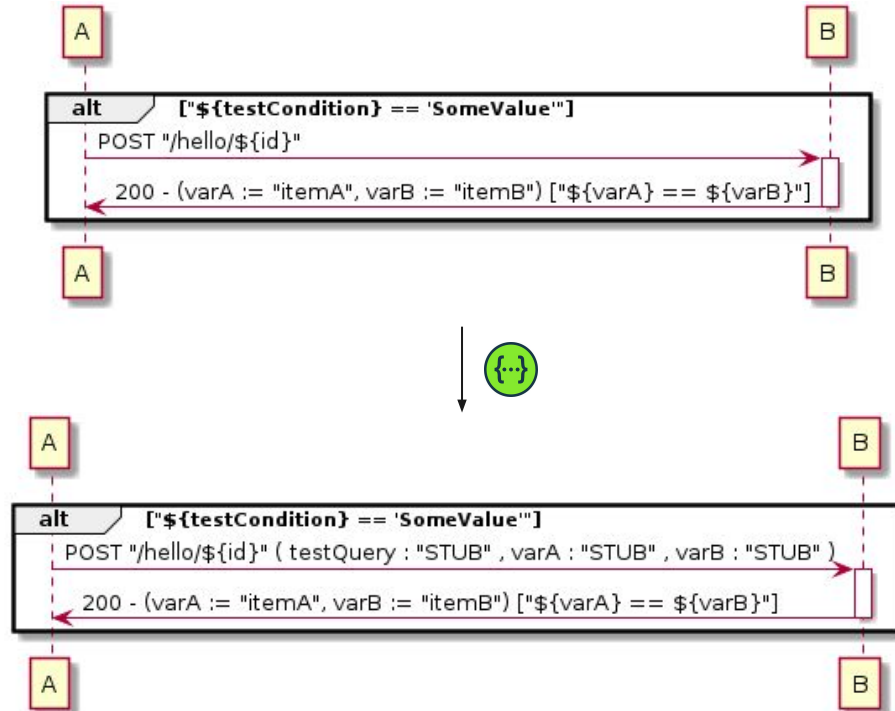
Shortcomings:

- Asynchrone Anfragen
- Swagger-Definitionen
- Keine numerische Datentypen

Requirement 1: Asynchrone Requests



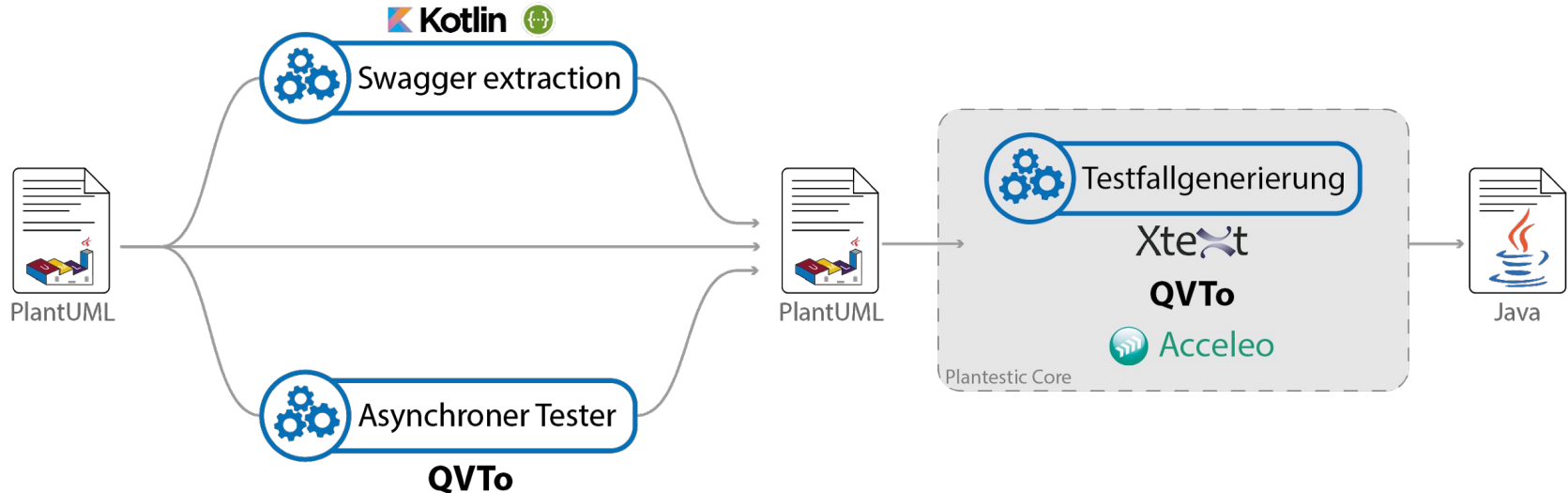
Requirement 2: Interface Specification



Lösung - eine neue Pipeline

Problem: beide Requirements sind ohne Domänenwissen nur schwer umsetzbar.

⇒ Unterstützen des Nutzers mittels Preprocessing



Implementation: Asynchroner Tester

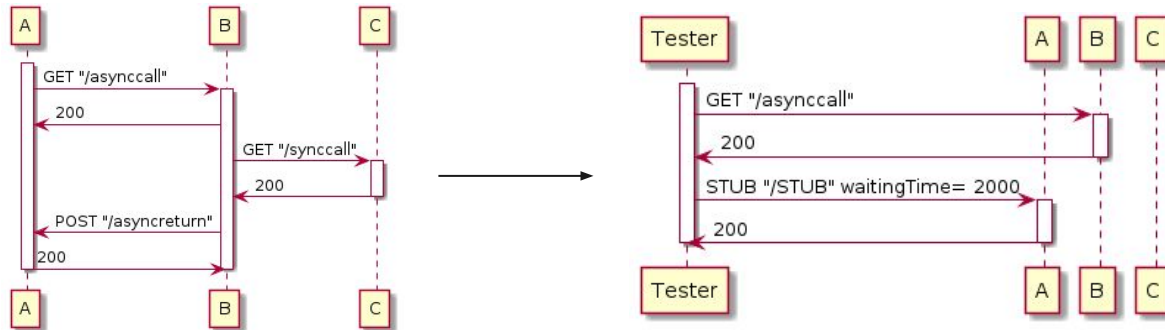
Ablauf:

1. PlantUML-Metamodell zum Parsen der Sequenzdiagramme in EMF-Modelle
2. Model-To-Model-Transformation mit QVTo
 - a. Ausgehende Nachrichten aus Participant und eingehende Nachrichten nach Participant rausziehen.
 - b. Extrahierte Nachrichten vom Tester aus versenden.
3. Serializer für Rücktransformation von EMF-Modelle zu PlantUML-Code
4. Formatter für menschenlesbare Ausgabe

xtext

QVTo

xtext



Implementation: Swagger Extraction

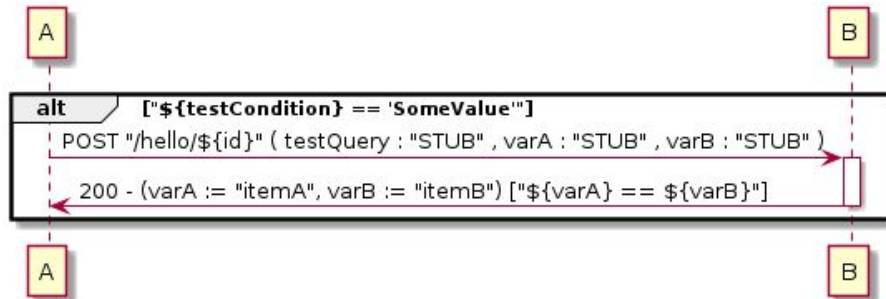
Ablauf:

1. PlantUML-Metamodell zum Parsen der Sequenzdiagramme in EMF-Modelle
2. Swagger-Metamodell zum Parsen der Schnittstellendefinitionen
3. Model-To-Model-Transformation in Kotlin
 - a. Hinzufügen benötigter Parameter zu allen Requests
 - b. Pattern-Matching zum Finden der entsprechenden Swagger-Definitionen
4. Serializer für Rücktransformation von EMF-Modell zu PlantUML-Code
5. Formatter für menschenlesbare Ausgabe

xtext

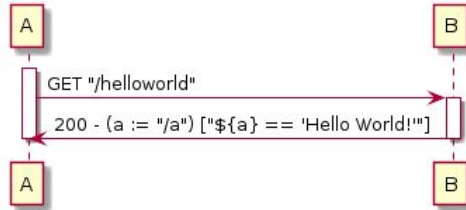
Kotlin

xtext

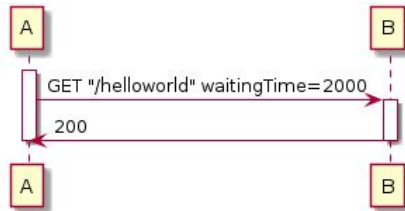


Zusätzliche Features

- **Request/Response Validation:** syntaktische Überprüfung von Nachrichten mithilfe von Swagger
- **Postcondition:** semantische Überprüfung von Responses



- **Waiting Time:** Timeout vor dem Senden von Nachrichten



- **Numeric Types:** Senden und Empfangen von Integern und Floats

Usability in CLI == 🤔 -> GUI 😍

Preprocessing Schritte erfordern jeweils eigenen CLI-Aufruf mit neuen / geänderten PlantUML-Dateien

```
./gradlew run --args="-s"
```



The screenshot displays the Plantastic Editor interface, which is a GUI for working with PlantUML diagrams. The main window is divided into several sections:

- Left Panel:** Contains three sequence diagrams. The top diagram is a simple sequence diagram with participants A, B, and C. The middle diagram is a more complex sequence diagram with a loop and a condition. The bottom diagram is a sequence diagram with a loop and a condition.
- Center Panel:** Displays the PlantUML code for the selected diagram. The code is as follows:

```
sequenceDiagram
    participant A
    participant B
    participant C
    A->>B: POST /hello/{id} ( testQuery: "STUB", varA: "STUB", varB: "STUB" )
    activate B
    B->>A: 200 - (varA := "itemA", varB := "itemB") [{"varA" == $(varB)}]
    deactivate B
    end
```
- Right Panel:** Contains a sidebar with three sections:
 - Pipeline:** Includes a button for "Autocomplete via Swagger" and a button for "Trigger Autocomplete".
 - Preprocess:** Includes a button for "Trigger Preprocessing".
 - Generate Tests:** Includes a button for "Generate Tests".

Und jetzt... Demo 

Fragen?
