

MODEL DRIVEN SOFTWARE DEVELOPMENT MIT DEM TOOLPAKET SWAGGER

Ein Vortrag von :

Herval Bernica Nganya Nana
Oussama Mzoughi
Aliridha Haouari
Christian Lange



Gliederung

- 1. - Einleitung**
- 2. - Model-Driven-Development**
- 3. - Swagger**
- 4. - Konzept**
- 5. - Prototyp**
- 6. - Demo**
- 7. - Zusammenfassung und Ausblick**
- 8. - References**



1. - Einleitung
2. - Model-Driven-Development
3. - Swagger
4. - Konzept
5. - Prototyp
6. - Demo
7. - Zusammenfassung und Ausblick
8. - References



Einleitung

- Die Erweiterung von Software kann viele Probleme beherbergen
 - z.B. Architekturerosionen
- Lösung: ein Verfahren, um Software aus Modellen zu generieren, Model Driven Software Development
- 1. Versuch eines Projektes mit diesem Verfahren als Gruppenarbeit



1. - Einleitung
2. - **Model-Driven-Development**
3. - **Swagger**
4. - **Hibernate**
5. - **Konzept**
6. - **Prototyp**
7. - **Demo**
8. - **Zusammenfassung und Ausblick**
9. - **References**



Model Driven Architecture (MDA)

- Bei der MDA bilden Modelle die zentralen Elemente des Softwareentwicklungsprozesses. Ziel ist es, plattformspezifische Modelle möglichst automatisiert aus plattformunabhängigen Modellen abzuleiten. Dadurch soll die Adaptierung an neue Technologien erleichtert werden.
- Mit MDA liegt ein Standardisierungsvorschlag der Objekt Management Group (OMG) vor.
- Aufhebung der Repräsentation von der Programmcodenebene auf die Modellebene , Um die Komplexität auf Modellebene zu reduzieren.



Modelle CIM-PIM-PSM

- In MDA wird hauptsächlich zwischen drei Typen von Modellen unterschieden:
- CIM (computerunabhängiges Modell) :
 - Beschreibung eines Softwaresystems auf fachlicher Ebene.
 - Fachliches Verständnis der Anwender des Systems .
 - Dienen dem besseren Diskurs zwischen Softwarearchitekten und Anwendern.

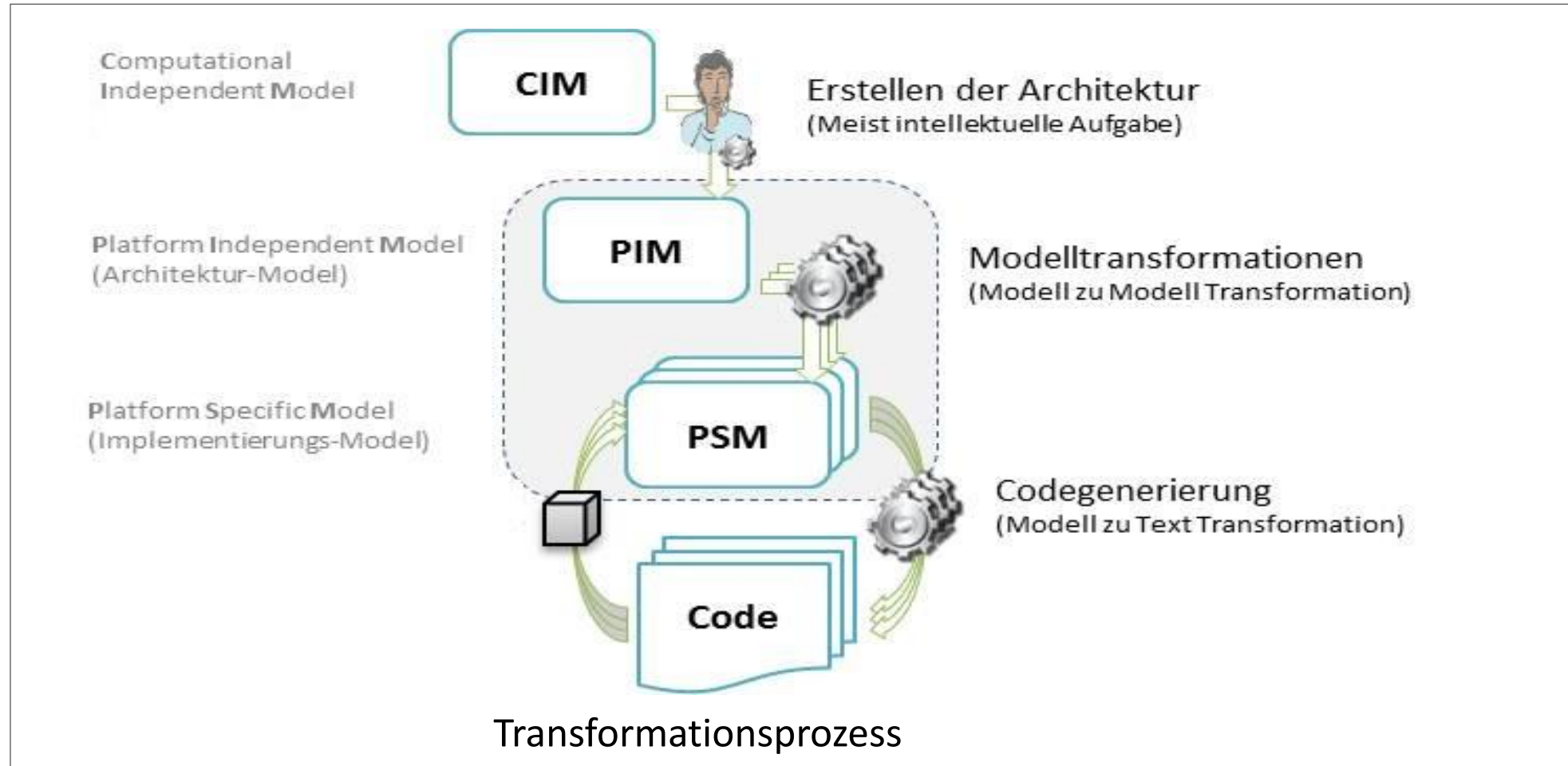


Modelle CIM-PIM-PSM

- **PIM**(plattformunabhängiges Modell):
 - Modellierung der Funktionalität einer Komponente unabhängig von der gegebenen Plattform.
 - Enthält den Teil eines Systems, der sich beschreiben lässt, ohne die endgültige Zielplattform zu kennen.
- **PSM**(plattformspezifisches Modell):
 - kennt eine spezielle Plattform und realisiert ein PIM.
 - Bereitstellung der Schnittstellen.



MDA-Transformation





1. - Einleitung
2. - Model-Driven-Development
3. - **Swagger**
4. - **Konzept**
5. - **Prototyp**
6. - **Demo**
7. - **Zusammenfassung und Ausblick**
8. - **References**



Swagger und Open Api

- **Swagger:** ein **MDSD** Framework das sich auf den Aufbau von Microservices konzentriert.
- Basiert auf der [OpenApi-Spezifikation](#).
- hilft dem Nutzer dabei, REST-APIs zu **konzipieren**, zu **dokumentieren** und zu **konsumieren**.

- **OpenApi:** ein APIBeschreibungsformat für REST-APIs.
- API-Spezifikationen können in YAML oder JSON geschrieben werden.
- Format für Maschinen und Menschen, leicht zu erlernen und lesbar.



Swagger Tools

- **Swagger Editor:**
 - browserbasierter Editor.
 - Mit OpenApi-Spezifikation schreiben und mit der Swagger-Spezifikation vergleichen.
- **Swagger UI:**
 - Mit der Spezifikationsdatei eine Benutzeroberfläche bereitstellen.
 - Anfrage und Antwort direkt von dem UI benutzen.
- **Swagger Codegen:**
 - Generiert Server-Stubs und Client-Bibliotheken aus einer OpenAPI-Spezifikation.



1. - Einleitung
2. - Model-Driven-Development
3. - Swagger
4. - Konzept
5. - Prototyp
6. - Demo
7. - Zusammenfassung und Ausblick
8. - References



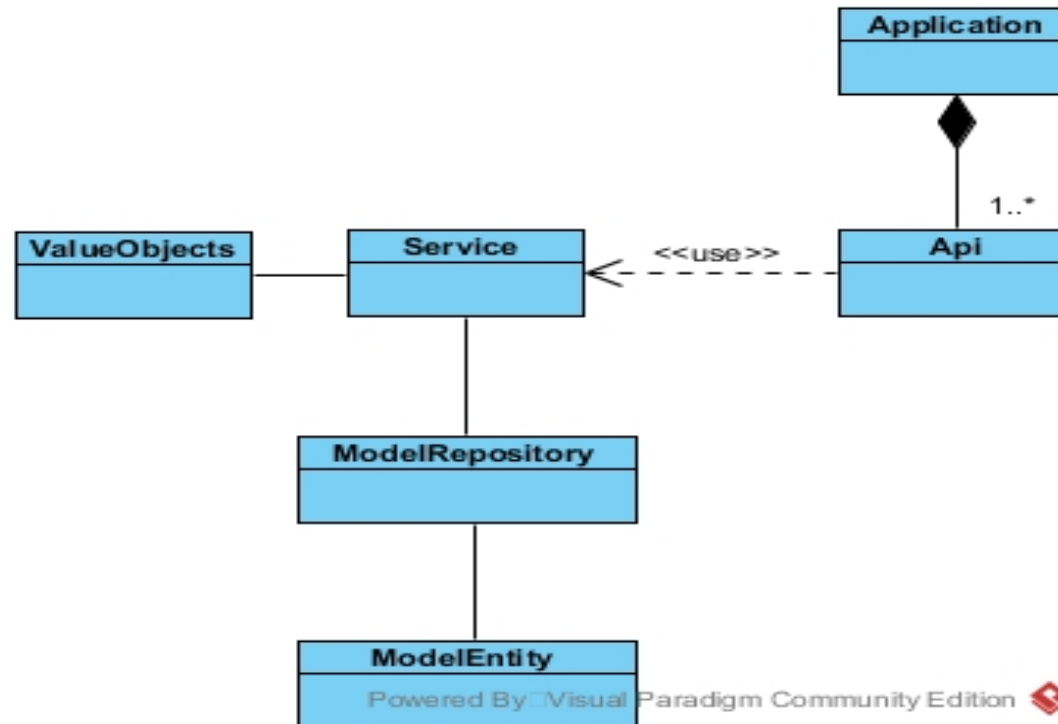
Beschreibung der Software

- Entwicklung eines Dateimanagementsystems für Studierende
- Aufgaben
 - neue Benutzer registrieren
 - Benutzer ein- und ausloggen
 - Dateien hoch- und herunterladen, löschen, umbenennen und auflisten
 - Ordner erstellen, löschen, umbenennen und auflisten



Konzept

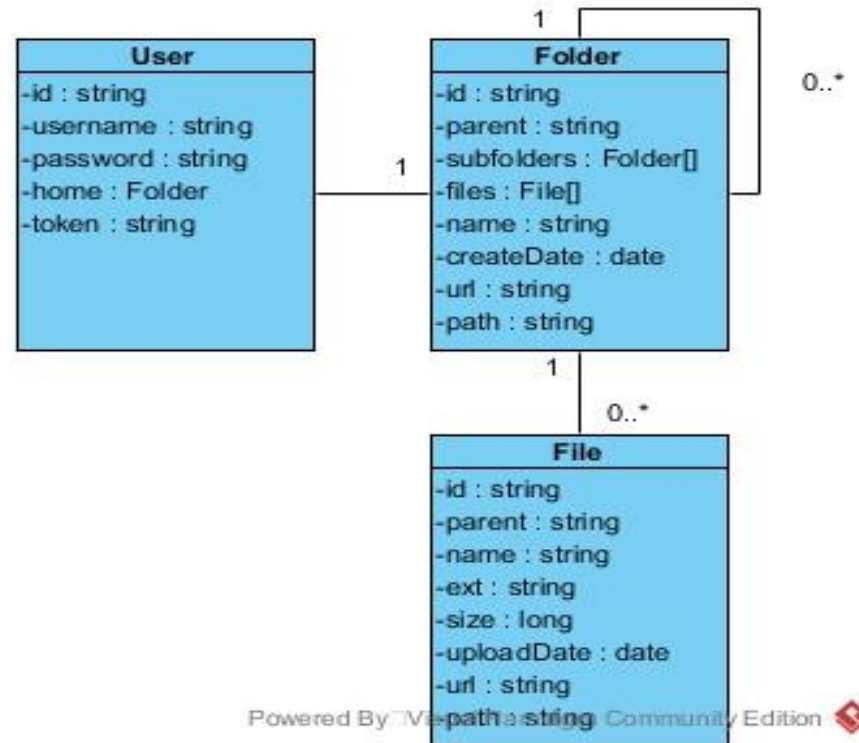
Development Metamodel



Development Metamodel vom Softwareprojekt



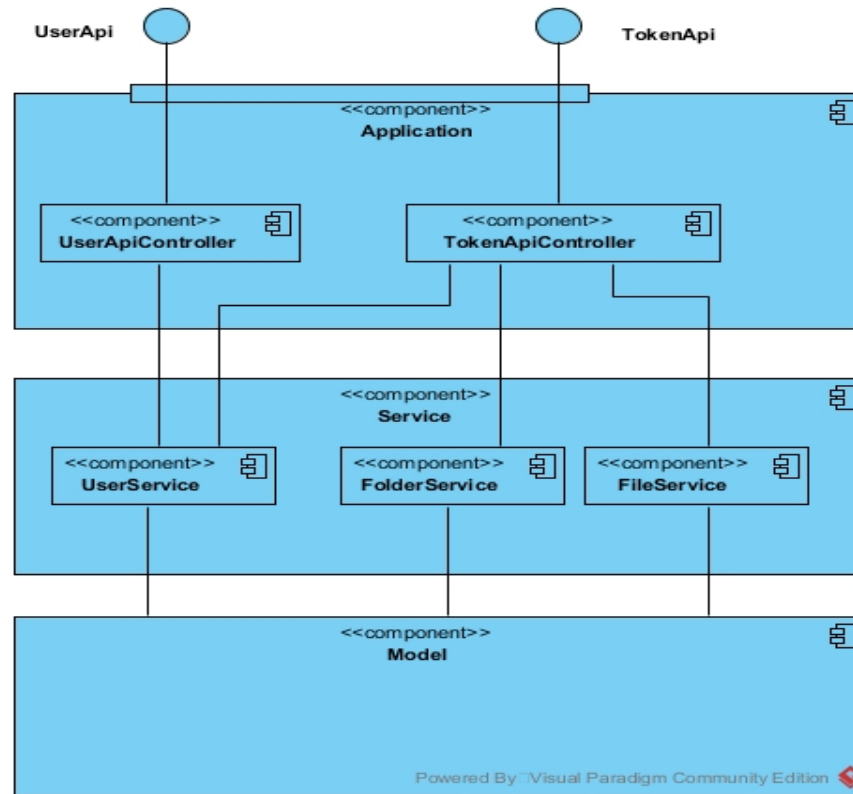
Klassendiagramm



Klassendiagramm vom Softwareprojekt



Schnittstellenbeschreibung

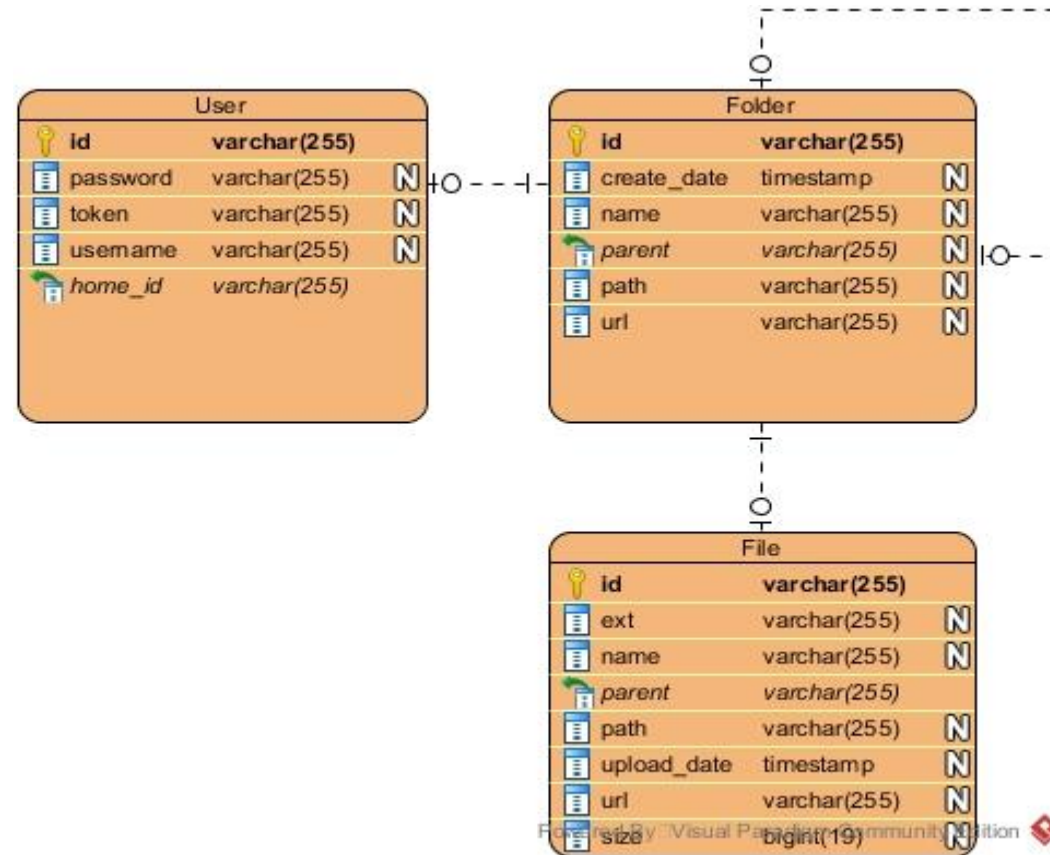


Komponentendiagramm vom Softwareprojekt



Konzept

Persistenz



ERM vom Softwareprojekt



1. - Einleitung
2. - Model-Driven-Development
3. - Swagger
4. - Konzept
5. - Prototyp
6. - Demo
7. - Zusammenfassung und Ausblick
8. - References



Prototyp

Einführung

- Entwicklung eines Prototypen mit
 - Swagger
 - AWS EC2
 - AWS S3
 - Hibernate
- Top-Down: von Modellen zum fertigen Code



Prototyp

Funktionalitäten

- Ziel des Rest-Services: Datenverwaltung in einer Cloud
- Schnittstellen:
 - Benutzer erstellen, sich ein- und ausloggen können
 - CRUD-Operationen zur Erfüllung der Aufgaben
- Insgesamt: 11 Schnittstellen



Prototyp

Schnittstellenerzeugung

- Insgesamt 6 Schritte zur Entwicklung des Prototypen
- Die 3 ersten sind modellgetrieben
 - Automatisch von Swagger erzeugt
- Die 3 letzten sind selbst programmiert
 - Selbstgeschriebene Codezeilen



Prototyp

Schnittstellenerzeugung

Erster Schritt:

- Beschreibung der Modelle, Schnittstellen und Meta-daten des Projekts mittels Swagger Editor
- Beschreibung entweder in JSON- oder YAML-Datei speichern
 - In diesem Projekt JSON



Prototyp

Schnittstellenerzeugung

Zweiter Schritt:

- Erzeugung der graphischen Dokumentation mittels Swagger UI
 - Input: JSON-Datei
 - Output: graphische Dokumentation
 - Enthält: URL-Beschreibung, Modelle, etc...



Prototyp

Schnittstellenerzeugung

Dritter Schritt:

- Erzeugung des fertigen Codes mittels Swagger Codegen
 - In diesem Projekt: Java Code (Spring)



Prototyp

Schnittstellenerzeugung

Vierter Schritt:

- Maven-Abhängigkeiten hinzufügen
 - AWS S3
 - MySQL
 - Spring-Data

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>aws-java-sdk-s3</artifactId>
  <version>1.11.269</version>
</dependency>

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.6</version>
</dependency>

<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-jpa</artifactId>
  <version>2.0.2.RELEASE</version>
</dependency>
```



Prototyp

Schnittstellenerzeugung

Fünfter Schritt:

- Konfiguration der Datenbank in dem Projekt hinterlegen.

```
spring.datasource.url=jdbc:mysql://localhost:3307/btd
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driverClassName=com.mysql.jdbc.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

- Annotation der Entitäten

```
/**
 * Folder
 */
@javax.annotation.Generated(value = "class io.swagger.codegen.languages.SpringCodegen",
@Entity
public class Folder implements Serializable {

    private static final long serialVersionUID = -454018730743763379L;

    @Id
    private String id = null;

    private String parent = null;

    @OneToMany(fetch = FetchType.LAZY, mappedBy = "parent", cascade = CascadeType.ALL)
    private List<Folder> subFolders = new ArrayList<Folder>();
```



Prototyp

Schnittstellenerzeugung

Sechster Schritt:

- Erweiterung der von Swagger leer erzeugten Funktionen



Prototyp

Erweiterung der Schnittstellen

- Feld Encryption als Boolean in dem Request Body zur Verschlüsselung der Daten beim Ein- und Ausloggen
 - True: mit Verschlüsselung
 - False: ohne Verschlüsselung



1. - Einleitung
2. - Model-Driven-Development
3. - Swagger
4. - Konzept
5. - Prototyp
6. - Demo
- 7. - Zusammenfassung und Ausblick**
- 8. - References**



Demo

- Dokumentation der Schnittstellen
- Funktionalität der Schnittstellen
 - Benutzer erstellen und weitere Funktionalitäten
- Datenbankeinträge
- AWS S3
- Code



1. - Einleitung
2. - Model-Driven-Development
3. - Swagger
4. - Konzept
5. - Prototyp
6. - Demo
7. - Zusammenfassung und Ausblick
8. - **References**



Zusammenfassung

- Drei Transformationsphasen:
 - CIM : Development Metamodel erstellt, um Anforderungen an unsere Funktionalitäten abzubilden.
 - PIM : Klassendiagramm, Komponentendiagramm
 - PSM : Api-Spezifikationsdatei für Rest-Apis erstellen
 - Spring-Code generieren.
- Weiterentwicklung :
 - Neue Spezifikationen in die Api-Spezifikationsdatei hinzufügen.



Zusammenfassung

- getroffenen Probleme:
 - Überschreibung des vorher generierten Codes sowie des selbstgeschriebenen Codes.
 - Lösung : Gap-Ansatz benutzen zur Weiterentwicklung.



1. - Einleitung
2. - Model-Driven-Development
3. - Swagger
4. - Konzept
5. - Prototyp
6. - Demo
7. - Zusammenfassung und Ausblick
- 8. - References**



References

- Offizielle spezifikation von swagger, openapi specification 3.0.0, zuletzt aufgerufen am 19.01.2018. page <https://swagger.io/specification/>.
- Anneke Kleppe, Jos Warmer, W. B. Mda explained: The model driven architecture : Practice and promise. Addison Wesley, page 192.
- Hibernate official website, Z. a. a. Page <http://www.omg.org/mda/>.
- MDA THE ARCHITECTURE OF CHOICE FOR A CHANGING WORLD, Page <http://www.omg.org/mda/>.
- model driven software development, I. O. . M. pages <http://www.itwissen.info/MDSD-model-driven-software-development-Modellgetriebene-SoftwareEntwicklung.html>.
- Timo Greifenberg, Katrin Hildobler, C. K. M. L. P. M. S.N. K. M. A. N. P. D. P. D. R. A. R. B. R. M. S. and Wortmann, A. (2015a). A Comparison of Mechanisms for Integrating Handwritten and Generated Code for ObjectOriented Programming Languages.

vielen dank für ihre Aufmerksamkeit