Generation of Container-Based Deployment Units Using an Ecosystem of Microservice-Oriented Modeling Languages

Third International Conference on Microservices (Microservices 2020)

Philip Wizenty

philip.wizenty@fh-dortmund.de

September 8-10, 2020

University of Applied Sciences and Arts Dortmund,
IDiAL Institute

Table of Content

Table of Content

Motivation

Problem statement:

- The configuration of container-based deployments for MSA is a complex and error-prone process [2]
  - There are a variety of technologies for deployment
  - The technologies have individual configuration options and formats
  - The configuration of the deployment extends over several files

Motivation

- Model-Driven Engineering (MDE) [1]:
  - Models are central artifacts in the software engineering process on an abstraction level above source code
  - Models in MDE can be used for the following purposes:
    - Static analysis
    - Code Generation
    - Documentation
    - ...
  - Expected advantages of MDE:
    - Improved reasoning possibilities for understanding the software architecture
    - Increased productivity
    - Possibility to perform analyses and simulations before implementation
    - ...

Motivation

- Model-Driven Engineering (MDE) [1]:
  - Models are central artifacts in the software engineering process on an abstraction level above source code
  - Models in MDE can be used for the following purposes:
    - Static analysis
    - Code Generation
    - Documentation
    - ...
  - Expected advantages of MDE:
    - Improved reasoning possibilities for understanding the software architecture
    - Increased productivity
    - Possibility to perform analyses and simulations before implementation
    - ...

Motivation

- Model-Driven Engineering (MDE) [1]:
  - Models are central artifacts in the software engineering process on an abstraction level above source code
  - Models in MDE can be used for the following purposes:
    - Static analysis
    - Code Generation
    - Documentation
    - ...
  - Expected advantages of MDE:
    - Improved reasoning possibilities for understanding the software architecture
    - Increased productivity
    - Possibility to perform analyses and simulations before implementation
    - ...

Motivation

- Model-Driven Engineering (MDE) [1]:
  - Models are central artifacts in the software engineering process on an abstraction level above source code
  - Models in MDE can be used for the following purposes:
    - Static analysis
    - Code Generation
    - Documentation
    - ...
  - Expected advantages of MDE:
    - Improved reasoning possibilities for understanding the software architecture
    - Increased productivity
    - Possibility to perform analyses and simulations before implementation
    - ...

Motivation

- LEMMA (Language Ecosystem for Modeling Microservice Architecture) [3]:
  - MDE approach for the model-driven development of microservice architectures
  - Modeling languages aligned to architecture viewpoints in MSA:
    - Domain Modeling Language
    - Service Modeling Language
    - Operation Modeling Language
    - Technology Modeling Language
- Expected benefits of LEMMA:
  - Increased productivity through the use of code generators
  - Improved understanding of the system architecture
  - Automation of quality attribute analysis
  - ...

Motivation

- LEMMA (Language Ecosystem for Modeling Microservice Architecture) [3]:
  - MDE approach for the model-driven development of microservice architectures
  - Modeling languages aligned to architecture viewpoints in MSA:
    - Domain Modeling Language
    - Service Modeling Language
    - Operation Modeling Language
    - Technology Modeling Language
- Expected benefits of LEMMA:
  - Increased productivity through the use of code generators
  - Improved understanding of the system architecture
  - Automation of quality attribute analysis
  - ...

Motivation

- LEMMA (Language Ecosystem for Modeling Microservice Architecture) [3]:
  - MDE approach for the model-driven development of microservice architectures
  - Modeling languages aligned to architecture viewpoints in MSA:
    - Domain Modeling Language
    - Service Modeling Language
    - Operation Modeling Language
    - Technology Modeling Language
- Expected benefits of LEMMA:
  - Increased productivity through the use of code generators
  - Improved understanding of the system architecture
  - Automation of quality attribute analysis
  - ...

Motivation

- LEMMA (Language Ecosystem for Modeling Microservice Architecture) [3]:
  - MDE approach for the model-driven development of microservice architectures
  - Modeling languages aligned to architecture viewpoints in MSA:
    - Domain Modeling Language
    - Service Modeling Language
    - Operation Modeling Language
    - Technology Modeling Language
- Expected benefits of LEMMA:
  - Increased productivity through the use of code generators
  - Improved understanding of the system architecture
  - Automation of quality attribute analysis
  - ...

Table of Content

Contributions

Approach:

- Using LEMMA's Operation Modeling Language (OML) allows model-based specification of microservice and infrastructure deployment
- The model contains all information necessary for the deployment:
  - Dependencies on infrastructural services such as API Gateways or Service Discoveries
  - Configurations for connection to database systems or message brokers
  - Service-specific configuration Deployments

Code Generation Pipeline

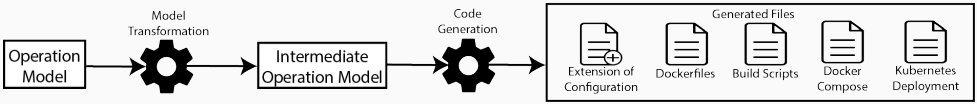


Figure 1: Workflow for the generation of deployment relevant artifacts.

- Model transformation
  - Refinement *Model-to-Model Transformation* [1]
  - Enrichment of the intermediate model with additional deployment relevant information
- Code Generation
  - *Model-to-Text Transformation* [1]
  - Generation of deployment relevant artifacts

Code Generation Pipeline

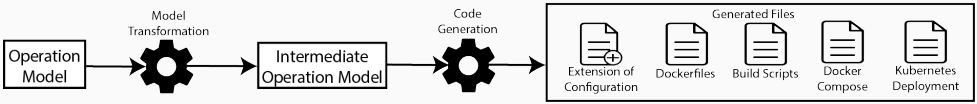


Figure 1: Workflow for the generation of deployment relevant artifacts.

- Model transformation
  - Refinement *Model-to-Model Transformation* [1]
  - Enrichment of the intermediate model with additional deployment relevant information
- Code Generation
  - *Model-to-Text Transformation* [1]
  - Generation of deployment relevant artifacts

Code Generation Pipeline

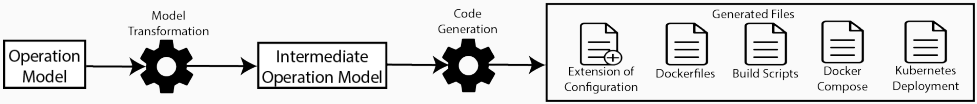


Figure 1: Workflow for the generation of deployment relevant artifacts.

- Model transformation
  - Refinement *Model-to-Model Transformation* [1]
  - Enrichment of the intermediate model with additional deployment relevant information
- Code Generation
  - *Model-to-Text Transformation* [1]
  - Generation of deployment relevant artifacts

Online Shop Example

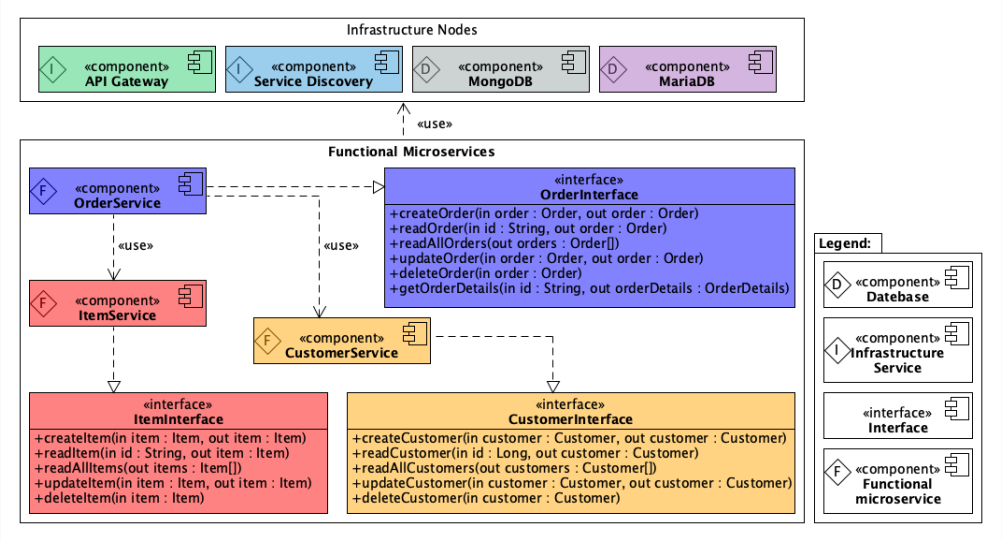


Figure 2: UML diagram of the Online Shop example.

Technology LEMMA Model - I

```
1  technology container {
2    deployment technologies {
3          Kubernetes {
4              operation environments = "openjdk:11-jdk-slim" default,
5                          "python:3";
6
7
8              service properties {
9                  // General spring boot configuration parameters
10                 string springApplicationName <mandatory>;
11                 int serverPort <mandatory>;
12         }
13     }
14 }
```

Listing 1: Excerpt of the Kubernetes deployment technology model.

Technology LEMMA Model - I

```
technology container {
  deployment technologies {
        Kubernetes {
            operation environments = "openjdk:11-jdk-slim" default,
                        "python:3";


            service properties {
                // General spring boot configuration parameters
                string springApplicationName <mandatory>;
                int serverPort <mandatory>;
        }
    }
}
```

Deployment

Listing 2: Excerpt of the Kubernetes deployment technology model.

Technology LEMMA Model - I

```
1  technology container {
2    deployment technologies {
3          Kubernetes {
4              operation environments = "openjdk:11-jdk-slim" default,
5                          "python:3";
6
7
8              service properties {
9                  // General spring boot configuration parameters
10                 string springApplicationName <mandatory>;
11                 int serverPort <mandatory>;
12         }
13     }
14 }
```

Deployment

Operation Environment

Listing 3: Excerpt of the Kubernetes deployment technology model.

Technology LEMMA Model - I

```
1  technology container {
2    deployment technologies {
3          Kubernetes {
4              operation environments = "openjdk:11-jdk-slim" default,
5                          "python:3";
6
7
8              service properties {
9                  // General spring boot configuration parameters
10                 string springApplicationName <mandatory>;
11                 int serverPort <mandatory>;
12         }
13     }
14 }
```

Deployment

Operation Environment

Service Properties

Listing 4: Excerpt of the Kubernetes deployment technology model.

Technology LEMMA Model - II

```
technology container {
      operation aspects {
        aspect Dockerfile<singleval> for containers, infrastructure {
            selector(technology = Kubernetes);
            string contents <mandatory>;
        }

        aspect KubernetesDeployment<singleval> for containers, infrastructure {
            selector(technology = Kubernetes);
            string contents <mandatory>;
        }

        aspect ComposePart<singleval> for containers, infrastructure {
            selector(technology = Kubernetes);
            string contents <mandatory>;
        }
    }
}
```

Listing 5: Excerpt of the Kubernetes deployment technology model.

Technology LEMMA Model - II

```
1  technology container {
2        operation aspects {
3          aspect Dockerfile<singleval> for containers, infrastructure {
4              selector(technology = Kubernetes);
5              string contents <mandatory>;
6          }
7
8          aspect KubernetesDeployment<singleval> for containers, infrastructure {
9              selector(technology = Kubernetes);
10             string contents <mandatory>;
11         }
12
13         aspect ComposePart<singleval> for containers, infrastructure {
14             selector(technology = Kubernetes);
15             string contents <mandatory>;
16         }
17     }
18 }
```

Dockerfile Aspect

Listing 6: Excerpt of the Kubernetes deployment technology model.

Technology LEMMA Model - II

```
1  technology container {
2        operation aspects {
3          aspect Dockerfile<singleval> for containers, infrastructure {
4              selector(technology = Kubernetes);
5              string contents <mandatory>;
6          }
7
8          aspect KubernetesDeployment<singleval> for containers, infrastructure {
9              selector(technology = Kubernetes);
10             string contents <mandatory>;
11         }
12
13         aspect ComposePart<singleval> for containers, infrastructure {
14             selector(technology = Kubernetes);
15             string contents <mandatory>;
16         }
17     }
18 }
```

Dockerfile Aspect

Kubernetes Aspect

Listing 7: Excerpt of the Kubernetes deployment technology model.

Technology LEMMA Model - II

```
1  technology container {
2        operation aspects {
3          aspect Dockerfile<singleval> for containers, infrastructure {
4              selector(technology = Kubernetes);
5              string contents <mandatory>;
6          }
7
8          aspect KubernetesDeployment<singleval> for containers, infrastructure {
9              selector(technology = Kubernetes);
10             string contents <mandatory>;
11         }
12
13         aspect ComposePart<singleval> for containers, infrastructure {
14             selector(technology = Kubernetes);
15             string contents <mandatory>;
16         }
17     }
18 }
```

Dockerfile Aspect

Kubernetes Aspect

Docker Compose Aspect

Listing 8: Excerpt of the Kubernetes deployment technology model.

Technology LEMMA Model - III

```
technology container {
      Eureka {
            operation environments = "openjdk:11-jdk-slim";
            service properties {
                string hostname <mandatory>;
                int port <mandatory>;
                string imageType <mandatory>;
            }
        }

        Zuul {
            operation environments = "openjdk:11-jdk-slim";
            service properties {
                string hostname <mandatory>;
                int port <mandatory>;
                string imageType <mandatory>;
            }
        }
}
```

Listing 9: Excerpt of the infrastructure technology model.

Technology LEMMA Model - III

```
technology container {
      Eureka {
            operation environments = "openjdk:11-jdk-slim";
            service properties {
                string hostname <mandatory>;
                int port <mandatory>;
                string imageType <mandatory>;
            }
        }

        Zuul {
            operation environments = "openjdk:11-jdk-slim";
            service properties {
                string hostname <mandatory>;
                int port <mandatory>;
                string imageType <mandatory>;
            }
        }
}
```

Eureka Technology Definition

Listing 10: Excerpt of the infrastructure technology model.

Technology LEMMA Model - III

```
1  technology container {
2        Eureka {
3              operation environments = "openjdk:11-jdk-slim";
4              service properties {
5                  string hostname <mandatory>;
6                  int port <mandatory>;
7                  string imageType <mandatory>;
8              }
9          }
10
11         Zuul {
12             operation environments = "openjdk:11-jdk-slim";
13             service properties {
14                 string hostname <mandatory>;
15                 int port <mandatory>;
16                 string imageType <mandatory>;
17             }
18         }
19 }
```

Eureka Technology Definition

Zuul Technology Definition

Listing 11: Excerpt of the infrastructure technology model.

ItemService OML Model

```
1  import microservices from "item.services" as itemService
2  import technology from "kubernetes.technology" as kubernetes
3  @technology(kubernetes)
4  container ItemService deployment technology kubernetes::_deployment.Kubernetes
5      with operation environment "openjdk:11-jdk-slim"
6      deploys itemService::v01.de.fhdo.online_shop.itemservice {
7          default values {
8              springApplicationName="ItemService"
9              serverPort=8081
10             springDataMongodbHost="mongo"
11             springDataMongodbPort=27017
12             springDataMongodbDatabase="item"
13             basic endpoints {kubernetes::_protocols.rest : "http://localhost:8081/
                   itemservice";}
14         }
15 }
```

Listing 12: ItemService OML model configuration.

ItemService OML Model

```
1  import microservices from "item.services" as itemService
2  import technology from "kubernetes.technology" as kubernetes
3  @technology(kubernetes)
4  container ItemService deployment technology kubernetes::_deployment.Kubernetes
5      with operation environment "openjdk:11-jdk-slim"
6      deploys itemService::v01.de.fhdo.online_shop.itemservice {
7          default values {
8              springApplicationName="ItemService"
9              serverPort=8081
10             springDataMongodbHost="mongo"
11             springDataMongodbPort=27017
12             springDataMongodbDatabase="item"
13             basic endpoints {kubernetes::_protocols.rest : "http://localhost:8081/
                   itemservice";}
14         }
15 }
```

Container Specification

Listing 13: ItemService OML model configuration.

ItemService OML Model

```
1  import microservices from "item.services" as itemService
2  import technology from "kubernetes.technology" as kubernetes
3  @technology(kubernetes)
4  container ItemService deployment technology kubernetes::_deployment.Kubernetes
5      with operation environment "openjdk:11-jdk-slim"
6      deploys itemService::v01.de.fhdo.online_shop.itemservice {
7          default values {
8              springApplicationName="ItemService"
9              serverPort=8081
10             springDataMongodbHost="mongo"
11             springDataMongodbPort=27017
12             springDataMongodbDatabase="item"
13             basic endpoints {kubernetes::_protocols.rest : "http://localhost:8081/
                   itemservice";}
14         }
15 }
```

Container Specification

Service Configuration

Listing 14: ItemService OML model configuration.

Zuul OML Model

```
1  import technology from "../Technology/infrastructure.technology" as infrastructureTech
2  import nodes from "../CustomerService/customer.operation" as customerService
3  import nodes from "../ItemService/item.operation" as itemService
4  import nodes from "../OrderService/order.operation" as orderService
5  import nodes from "eureka.operation" as eureka
6
7  @technology(infrastructureTech)
8  Zuul is infrastructureTech::_infrastructure.Zuul
9      with operation environment "openjdk:11-jdk-slim"
10     depends on nodes eureka::Eureka
11     used by nodes customerService::CustomerService
12       itemService::ItemService
13       orderService::OrderService {
14     default values {
15         hostname="Zuul"
16         port=8080
17         imageType="SpringComponent"
18     }
19     endpoints {infrastructureTech::_protocols.html: "http://localhost:8080";}
20 }
```

Listing 15: Zuul OML model configuration.

Zuul OML Model

```
1  import technology from "../Technology/infrastructure.technology" as infrastructureTech
2  import nodes from "../CustomerService/customer.operation" as customerService
3  import nodes from "../ItemService/item.operation" as itemService
4  import nodes from "../OrderService/order.operation" as orderService
5  import nodes from "eureka.operation" as eureka
6
7  @technology(infrastructureTech)
8  Zuul is infrastructureTech::_infrastructure.Zuul
9      with operation environment "openjdk:11-jdk-slim"
10     depends on nodes eureka::Eureka
11     used by nodes customerService::CustomerService
12       itemService::ItemService
13       orderService::OrderService {
14     default values {
15         hostname="Zuul"
16         port=8080
17         imageType="SpringComponent"
18     }
19     endpoints {infrastructureTech::_protocols.html: "http://localhost:8080";}
20 }
```

Import

Listing 16: Zuul OML model configuration.

Zuul OML Model

```
1  import technology from "../Technology/infrastructure.technology" as infrastructureTech
2  import nodes from "../CustomerService/customer.operation" as customerService
3  import nodes from "../ItemService/item.operation" as itemService
4  import nodes from "../OrderService/order.operation" as orderService
5  import nodes from "eureka.operation" as eureka
6
7  @technology(infrastructureTech)
8  Zuul is infrastructureTech::_infrastructure.Zuul
9      with operation environment "openjdk:11-jdk-slim"
10     depends on nodes eureka::Eureka
11     used by nodes customerService::CustomerService
12       itemService::ItemService
13       orderService::OrderService {
14     default values {
15         hostname="Zuul"
16         port=8080
17         imageType="SpringComponent"
18     }
19     endpoints {infrastructureTech::_protocols.html: "http://localhost:8080";}
20 }
```

Import

Node Specification

Listing 17: Zuul OML model configuration.

Zuul OML Model

```
1  import technology from "../Technology/infrastructure.technology" as infrastructureTech
2  import nodes from "../CustomerService/customer.operation" as customerService
3  import nodes from "../ItemService/item.operation" as itemService
4  import nodes from "../OrderService/order.operation" as orderService
5  import nodes from "eureka.operation" as eureka
6
7  @technology(infrastructureTech)
8  Zuul is infrastructureTech::_infrastructure.Zuul
9      with operation environment "openjdk:11-jdk-slim"
10     depends on nodes eureka::Eureka
11     used by nodes customerService::CustomerService
12       itemService::ItemService
13       orderService::OrderService {
14     default values {
15         hostname="Zuul"
16         port=8080
17         imageType="SpringComponent"
18     }
19     endpoints {infrastructureTech::_protocols.html: "http://localhost:8080";}
20 }
```

Import

Node Specification

Dependencies

Listing 18: Zuul OML model configuration.

Zuul OML Model

```
1  import technology from "../Technology/infrastructure.technology" as infrastructureTech
2  import nodes from "../CustomerService/customer.operation" as customerService
3  import nodes from "../ItemService/item.operation" as itemService
4  import nodes from "../OrderService/order.operation" as orderService
5  import nodes from "eureka.operation" as eureka
6
7  @technology(infrastructureTech)
8  Zuul is infrastructureTech::_infrastructure.Zuul
9      with operation environment "openjdk:11-jdk-slim"
10     depends on nodes eureka::Eureka
11     used by nodes customerService::CustomerService
12       itemService::ItemService
13       orderService::OrderService {
14     default values {
15         hostname="Zuul"
16         port=8080
17         imageType="SpringComponent"
18     }
19     endpoints {infrastructureTech::_protocols.html: "http://localhost:8080";}
20 }
```

Import

Node Specification

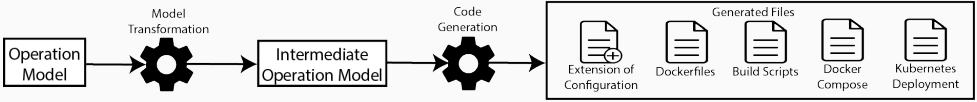
Dependencies

Node Configuration

Listing 19: Zuul OML model configuration.

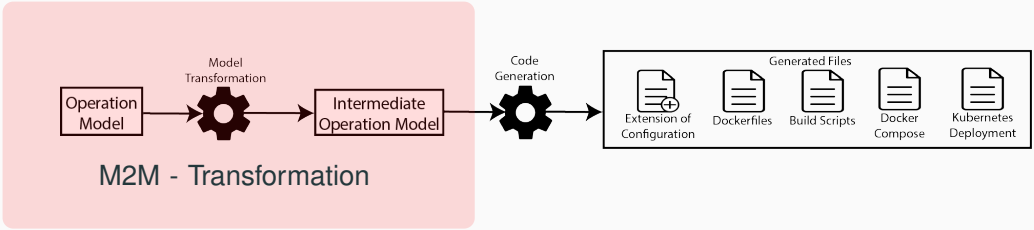
Model-to-Model Transformation

Figure 3: Refinement *Model-to-Model Transformation.*



Model-to-Model Transformation

Figure 3: Refinement *Model-to-Model Transformation.*



M2M Transformation - Artefacts

```
@technology(infrastructureTech)
Zuul is infrastructureTech::_infrastructure.Zuul
    with operation environment "openjdk:11-jdk-slim"
    depends on nodes eureka::Eureka
    used by nodes customerService::CustomerService
        itemService::ItemService
        orderService::OrderService {
    default values {
        hostname="Zuul"
        port=8080
        imageType="SpringComponent"
    }
    endpoints {infrastructureTech::_protocols.html:
        "http://localhost:8080";}
}
```

Figure 4: Excerpt of the Zuul OML Model.

M2M Transformation - Artefacts

```
@technology(infrastructureTech)
Zuul is infrastructureTech::_infrastructure.Zuul
    with operation environment "openjdk:11-jdk-slim"
    depends on nodes eureka::Eureka
    used by nodes customerService::CustomerService
        itemService::ItemService
        orderService::OrderService {
    default values {
        hostname="Zuul"
        port=8080
        imageType="SpringComponent"
    }
    endpoints {infrastructureTech::_protocols.html:
        "http://localhost:8080";}
}
```

Figure 4: Excerpt of the Zuul OML Model.

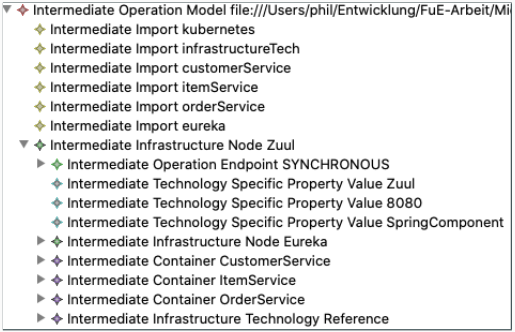
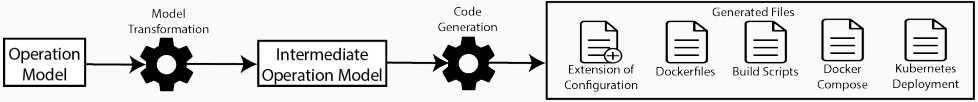


Figure 5: Excerpt of the Zuul intermediate OML Model.

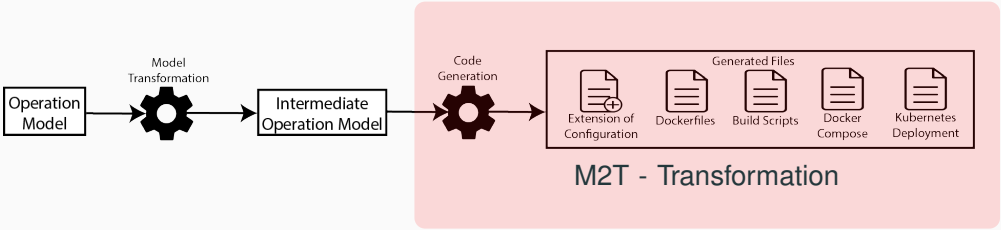
Code Generation

Figure 6: Code Generation *Model-to-Text Transformation*.



Code Generation

Figure 6: Code Generation *Model-to-Text Transformation*.



M2T Transformation - Artefacts

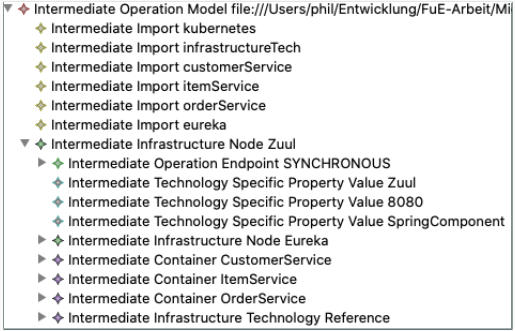


Figure 7: Excerpt of the Zuul intermediate OML Model.

M2T Transformation - Artefacts

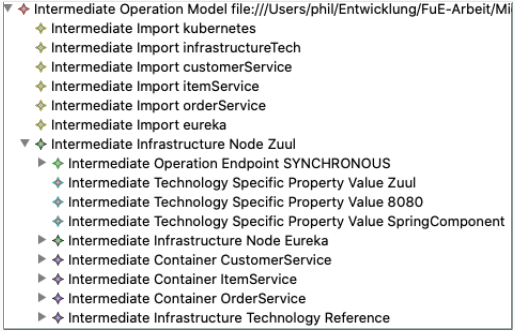
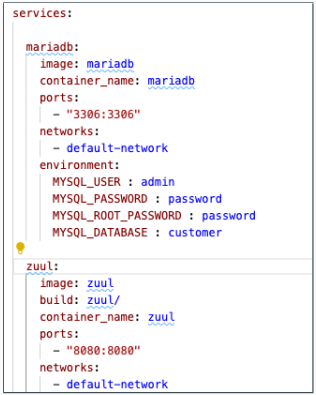


Figure 7: Excerpt of the Zuul intermediate OML Model.



```
services:

  mariadb:
    image: mariadb
    container_name: mariadb
    ports:
      - "3306:3306"
    networks:
      - default-network
    environment:
      MYSQL_USER : admin
      MYSQL_PASSWORD : password
      MYSQL_ROOT_PASSWORD : password
      MYSQL_DATABASE : customer

  zuul:
    image: zuul
    build: zuul/
    container_name: zuul
    ports:
      - "8080:8080"
    networks:
      - default-network
```

Figure 8: Excerpt of zuuls docker compose deployment file.

M2T Transformation - Artefacts

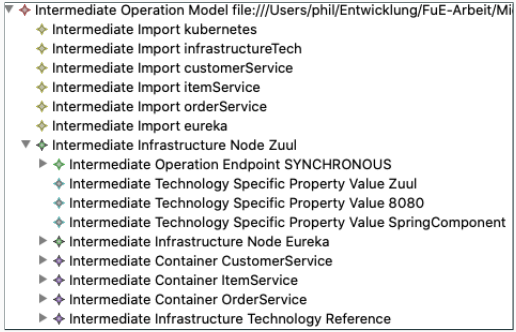


Figure 9: Excerpt of the Zuul intermediate OML Model.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: zuul
  name: zuul
spec:
  replicas: 1
  selector:
    matchLabels:
      app: zuul
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: zuul
    spec:
      containers:
      - image: zuul
        imagePullPolicy: "Never"
        name: zuul
        resources: {}
status: {}
```

Figure 10: Excerpt of Zuuls kubernetes deployment file.

M2T Transformation - Artefacts

```
eureka.client.fetchRegistry=true
eureka.client.registerWithEureka=true
eureka.client.serviceUrl.defaultZone=${EUREKA_URI:http://eureka:8761/eureka}
eureka.instance.preferIpAddress=true
ribbon.ReadTimeout=60000
server.port=8080
spring.application.name=ZuulService
zuul.routes.customerservice.path=/customerservice/**
zuul.routes.customerservice.serviceId=customerservice
zuul.routes.customerservice.strip-prefix=true
zuul.routes.eureka.path=/eureka/**
zuul.routes.eureka.serviceId=eureka
zuul.routes.eureka.strip-prefix=true
```

Figure 11: Excerpt of zuuls `application.properties` file.

Table of Content

Efficiency

- Efficiency of the Code Generation Pipeline:
  - In shown example, 1 OML LoC on average results in 5.2 LoC source/configuration code
  - The efficiency of the code generation pipeline is based on several factors:
    - Many configuration files in YAML format
    - Several files are required for the deployment of a service
    - There are dependencies between the configuration files

Efficiency

- Efficiency of the Code Generation Pipeline:
  - In shown example, 1 OML LoC on average results in 5.2 LoC source/configuration code
  - The efficiency of the code generation pipeline is based on several factors:
    - Many configuration files in YAML format
    - Several files are required for the deployment of a service
    - There are dependencies between the configuration files

Conclusion

- Conclusion:
  - Kubernetes or Docker Compose configuration are generated automatically
  - Advanced configurations can be added via operation aspects
  - The generation of deployment artifacts reduces the complexity of container-based technologies

Future Work

- Future Work:
  - Extension of the functional range of the code generator
  - Integration of technology-specific tests for checking the correctness of the models
  - Specification of CI/CD pipelines in the deployment workflow

Literatur

[1] Benoit Combemale et al. *Engineering modeling languages*. Taylor & Francis, CRC Press, 2017.

[2] Hui Kang, Michael Le, and Shu Tao. “Container and Microservice Driven Design for Cloud Infrastructure DevOps.” In: *2016 IEEE Int. Conf. on Cloud Engineering (IC2E)*. IEEE, 2016.

[3] Florian Rademacher et al. “Graphical and Textual Model-Driven Microservice Development.” In: *Microservices: Science and Engineering*. Springer, Dec. 2019, pp. 147–179.