Semi-Automated Viewpoint-based Reconstruction and Analysis of Microservice Architecture

AK Microservices and DevOps 2023, Cologne, Germany

Philip Wizenty

philip.wizenty@fh-dortmund.de

15 Sep 2023

University of Applied Sciences and Arts Dortmund,
IDiAL Institute

Inhaltsverzeichnis

Table of Content

Motivation

- MSA promotes to increase service independence by
  - letting it realize a distinct, self-contained capability
  - decreasing its coupling to other software components w.r.t., e.g., implementation, testing, and operation
  - transferring its ownership to a dedicated team, being responsible for all aspects related to service design, implementation, and operation
  - add modifiability
- Improved maintainability by facilitating the replacement of services with improved versions

Motivation

- MSA promotes to increase service independence by
  - letting it realize a distinct, self-contained capability
  - decreasing its coupling to other software components w.r.t., e.g., implementation, testing, and operation
  - transferring its ownership to a dedicated team, being responsible for all aspects related to service design, implementation, and operation
  - add modifiability
- Improved maintainability by facilitating the replacement of services with improved versions

Motivation

- MSA promotes to increase service independence by
  - letting it realize a distinct, self-contained capability
  - decreasing its coupling to other software components w.r.t., e.g., implementation, testing, and operation
  - transferring its ownership to a dedicated team, being responsible for all aspects related to service design, implementation, and operation
  - add modifiability
- Improved maintainability by facilitating the replacement of services with improved versions

Motivation

Problem Statement

- Increased modifiability facilitates service evolution
- Increased independence enables teams to autonomously adapt different parts of the software system

$\Rightarrow$ Increased risk for the erosion of the anticipated architecture design

Motivation

Solution Proposal

- Software Architecture Reconstruction (SAR) [1] to (semi-) automatically recover a microservice architecture's design
- Model-based SAR to recover architecture information from different viewpoints
- The viewpoints addressing concerns of different type of stakeholders in the software engineering process
- Models to facilitate the engineering process of the MSA-based software system

Table of Content

Viewpoint-based MSA Modeling with LEMMA

- Model-driven Engineering (MDE) [2] is an approach to software engineering that aims to facilitate the design, implementation, and operation of a software system though the use of *models*
- A *model* [2] in sense of MDE is an artifact that:
  - Abstracts from selected characteristics of the considered software system
  - Is expressed in a dedicated modeling language
  - Is (semi-) automatically processible for specific purposes in the software engineering process

Viewpoint-based MSA Modeling with LEMMA

- Model-driven Engineering (MDE) [2] is an approach to software engineering that aims to facilitate the design, implementation, and operation of a software system though the use of *models*
- A *model* [2] in sense of MDE is an artifact that:
  - Abstracts from selected characteristics of the considered software system
  - Is expressed in a dedicated modeling language
  - Is (semi-) automatically processible for specific purposes in the software engineering process

Viewpoint-based MSA Modeling with LEMMA

- Model-based *viewpoints* [4, 3] provide means to reduce the software system's complexity by describing only a specific part of the system
- *View models* are specifically effective in making the parts and underlying concepts of complex software architectures explicit to facilitate the reasoning about them [8]

Viewpoint-based MSA Modeling with LEMMA

- Model-based *viewpoints* [4, 3] provide means to reduce the software system's complexity by describing only a specific part of the system
- *View models* are specifically effective in making the parts and underlying concepts of complex software architectures explicit to facilitate the reasoning about them [8]

Viewpoint-based MSA Modeling with LEMMA

- LEMMA[1] is an MDE-based ecosystem that focuses on the concerns of different stakeholder groups in MSA engineering
- LEMMA enables the construction of models for...
  - ... domain-driven service design (Domain Data Modeling Language)
  - ... API management (Service Modeling Language)
  - ... service operation (Operation Modeling Language)

[1] `https://fh.do/lemma`

Viewpoint-based MSA Modeling with LEMMA

- LEMMA[1] is an MDE-based ecosystem that focuses on the concerns of different stakeholder groups in MSA engineering
- LEMMA enables the construction of models for...
  - ... domain-driven service design (Domain Data Modeling Language)
  - ... API management (Service Modeling Language)
  - ... service operation (Operation Modeling Language)

[1] `https://fh.do/lemma`

Viewpoint-based MSA Modeling with LEMMA

- **Domain Data Modeling Language (DDML)**
- Focuses on concerns of domain experts and microservice developer
- Provides linguistic support for Domain-driven Design (DDD)

Listing 1: Excerpt from LEMMA's Domain Data Model.

```
context customerManagementBackend {
    structure InteractionEntity<entity> {
        string id<identifier>,
        date createDate,
        string content,
        boolean sentByOperator
    }
}
```

Viewpoint-based MSA Modeling with LEMMA

- **Service Modeling Language (SML)**
- Developers can use the SML to model microservice APIs and endpoints.

Listing 2: Excerpt from LEMMA's Service Model.

```
public functional microservice
    com.lakesidemutual.customerManagementBackend.CustomerManagementBackend {
    required microservices { coreServices::com.lakesidemutual.CustomerCore }
    interface customerCoreClient {
        getCustomers(
      sync filter? : string,
           sync limit? : int,
      sync customerId? : int
    ...
  );
...
}
```

Viewpoint-based MSA Modeling with LEMMA

- **Operation Modeling Language (SML)**
- The OML defines modeling concepts for microservice operators to express microservices' deployment and use of operation infrastructure.

Listing 3: Excerpt from LEMMA's Operation Model.

```
@technology(container_base)
@technology(protocol)
 container CustomerManagementContainer
    deployment technology container_base::_deployment.Kubernetes
    deploys customerManagementServices
  ::com.lakesidemutual.customerManagementBackend.CustomerManagementBackend
  depends on nodes eureka::ServiceDiscovery {
  eurekaUri = "http://localhost:8761"
  basic endpoints { protocol::_protocols.rest: "http://localhost:8100"; }
}}
```

Table of Content

LEMMA-Enabled Approach for MSA Reconstruction

- (a) LEMMA-Enabled Microservice Architecture Reconstruction (MAR) Framework
  - Orchestrates the stages of the SAR process
  - Provides functionalities for reconstructing viewpoint-specific information
  - Manages MAR plugins
- (b) MAR Plugins
  - Derive viewpoint-specific architecture information from source code artifacts

LEMMA-Enabled Approach for MSA Reconstruction

- (a) LEMMA-Enabled Microservice Architecture Reconstruction (MAR) Framework
  - Orchestrates the stages of the SAR process
  - Provides functionalities for reconstructing viewpoint-specific information
  - Manages MAR plugins
- (b) MAR Plugins
  - Derive viewpoint-specific architecture information from source code artifacts

LEMMA-Enabled Approach for MSA Reconstruction

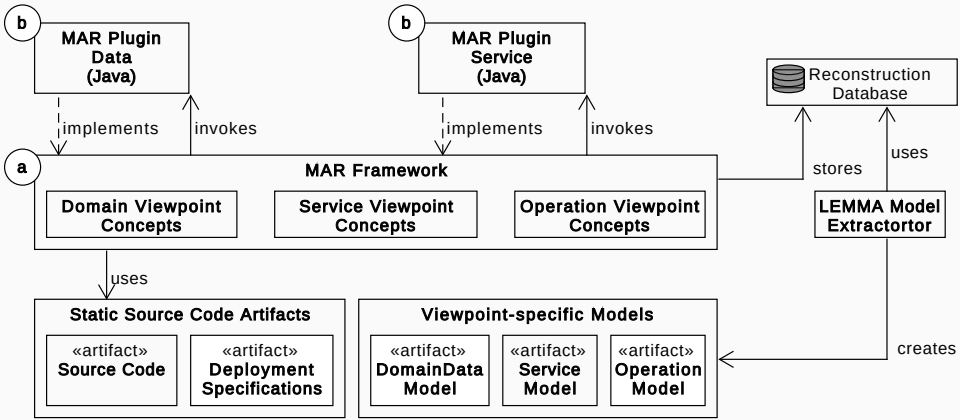


Figure 1: LEMMA-Enabled Approach for MSA Reconstruction.

LEMMA-Enabled Approach for MSA Reconstruction

Listing 4: Example Java source code artifact.

```java
@Entity
@Table(name = "interactions")
public class InteractionEntity {
  @Id
  private String id;
  private Date date;
  private String content;
  private boolean sentByOperator;
  ...
}
```

Listing 5: Reconstructed LEMMA domain model.

```
context customerManagementBackend {
  structure InteractionEntity<entity> {
    string id<identifier>,
    date createDate,
    string content,
    boolean sentByOperator,
    ...
  }
}
```

LEMMA-Enabled Approach for MSA Reconstruction - One Slide

Listing 6: Example Java source code artifact.

```java
@RestController
@RequestMapping("/customers")
public class CustomerInformationHolder {
  @GetMapping(value = "/{customerId}")
  public ResponseEntity<CustomerDto> getCustomer(
    @PathVariable CustomerId customerId) {...
    return ResponseEntity.ok(customer);}}
```

Listing 7: Reconstructed LEMMA service model.

```
  public functional microservice com.lakesidemutual.CustomerManagement {
    interface CustomerInformationHolder {
      getCustomers(
        sync out customer : Customer::Customer.PaginatedCustomerResponseDto,
        sync in filter : string,  sync in integer : customerId);
...}}
```

Table of Content

Validation of LEMMA's Reconstruction Framework

- *Binary Classification* [7] for the validation of the reconstruction results
- Classification of the reconstruction architecture information:
  - *True positive* (TP): Correctly reconstructed
  - *True negative* (TN): Not reconstructed
  - *False positive* (FP) / *False negative* (FN): Wrongly reconstructed

Validation of LEMMA's Reconstruction Framework

- *Binary Classification* [7] for the validation of the reconstruction results
- Classification of the reconstruction architecture information:
  - *True positive* (TP): Correctly reconstructed
  - *True negative* (TN): Not reconstructed
  - *False positive* (FP) / *False negative* (FN): Wrongly reconstructed

Validation of LEMMA's Reconstruction Framework

1. *Recall* [7]: Probability to identify a relevant element

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

2. *Precision* [7]: The correctness of the reconstructed elements

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

3. $F_{measure}$ [7]: Accuracy of the entire reconstructed architectural design

$$F_{measure} = 2 * \frac{Recall * Precision}{Recall + Precision} \tag{3}$$

Validation of LEMMA's Reconstruction Framework

1. *Recall* [7]: Probability to identify a relevant element

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

2. *Precision* [7]: The correctness of the reconstructed elements

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

3. $F_{measure}$ [7]: Accuracy of the entire reconstructed architectural design

$$F_{measure} = 2 * \frac{Recall * Precision}{Recall + Precision} \tag{3}$$

Validation of LEMMA's Reconstruction Framework

1. *Recall* [7]: Probability to identify a relevant element

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

2. *Precision* [7]: The correctness of the reconstructed elements

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

3. $F_{measure}$ [7]: Accuracy of the entire reconstructed architectural design

$$F_{measure} = 2 * \frac{Recall * Precision}{Recall + Precision} \tag{3}$$

Validation of LEMMA's Reconstruction Framework

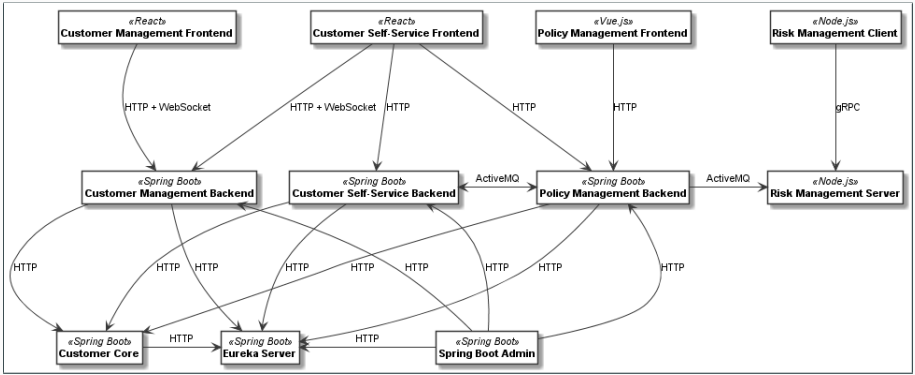


Figure 2: Intended architecture design of Lakeside Mutual[2].

[2] https://github.com/Microservice-API-Patterns/LakesideMutual

Validation of LEMMA's Reconstruction Framework

Table 1: Results for the reconstruction of the architecture design of Lakeside Mutual[3].

Java | Domain Data Model | Service Model

Element	Expected	TP	FP	FN	Recall	Precision	$F_{measure}$
Microservices	5	4	0	1	80%	100%	88%
Interfaces	16	14	0	2	87%	100%	93%
Operations	61	50	3	8	86%	94%	90%
Data Structures	161	117	29	14	89%	80%	84%

[3] https://github.com/SeelabFhdo/microservices2022

Table of Content

Motivation

Problem Statement

- Security Smells can negatively influence the software system's security [6]
- Manual resolving security smells is costly, error-prone, and complex

⚠ Security Smells decrease the software system's overall quality and development efficiency [5]

Motivation

Solution Proposal

- Model-driven Engineering (MDE) [2] to detect Security Smells in MSA
- Refactor security smells via Model-to-Model Transformation

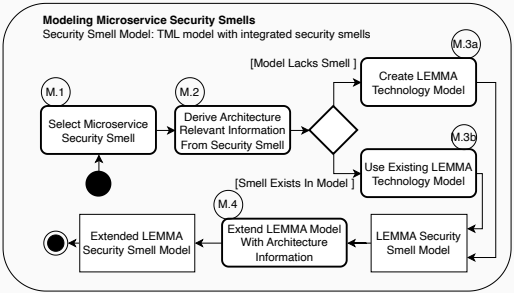
Reduce complexity, errors, and costs of refactorings in MSA

Model-driven Security Smell Resolution

- Modeling
- Detecting
- Resolving

Modeling Microservice Security Aspects

Figure 3: Activities of modeling security aspects.



Listing 11: LEMMA security aspect technology model.

```
technology SecurityAspects {
  service aspects{
    aspect usesApiGateway for microservices;
    aspect Authorization for microservices {
      string protocolName;
      }
    aspect Secured for interfaces, operations {
      string role;
    }
  }
  operation aspects {
    aspect ApiGateway for infrastructure;
  }
}
```

Detecting Microservice Security Smells

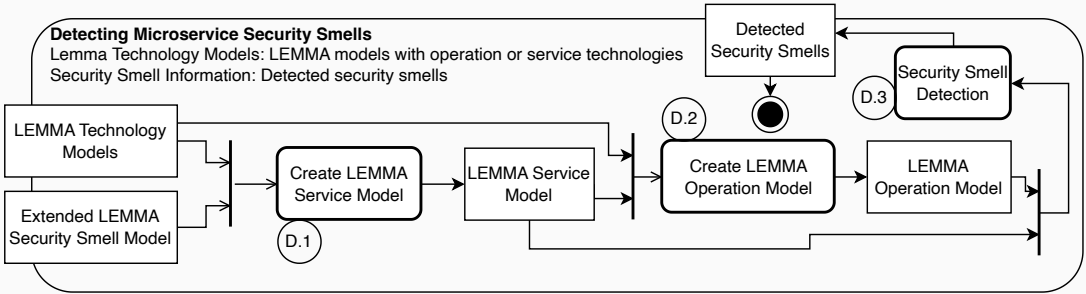


Figure 4: Activities of detecting security smells.

Listing 12: LEMMA service model CustomerCore.

```
import datatypes from "customerCore.data" as domain
import technology from "spring.technology" as spring

@technology(spring)
@spring::_aspects.ApplicationName("CustomerCore")
@spring::_aspects.Port(8080)
public functional microservice
  com.lakeside.CustomerCore {
  @endpoints(java::_protocols.rest: "/cities";)
  interface cityStaticDataHolder {
    @endpoints({spring::_protocols.rest: "/{id}";})
    @spring::_aspects.GetMapping
    getCitiesForPostalCode(
      sync in postalCode : string,
      sync out cities :
        domain::customerCore.CitiesResponseDto);}
    ...
}
```

Listing 13: LEMMA operation model CustomerCore.

```
import microservices from "customerCore.services"
  as customerCore
import technology from "deploymentBase.technology"
  as deploymentBase
@technology(deploymentBase)
container CustomerCoreContainer
  deployment technology
    deploymentBase::_deployment.Docker
  deploys customerCore::com.lakeside.CustomerCore
  depends on nodes
    infrastructure::ServiceDiscovery,
    infrastructure::H2Database {
    default values {
      basic endpoints { protocolTechnology::
        _protocols.rest: "http://localhost:8110"; }
    }
    ...
}
```

Resolving Microservice Security Smells

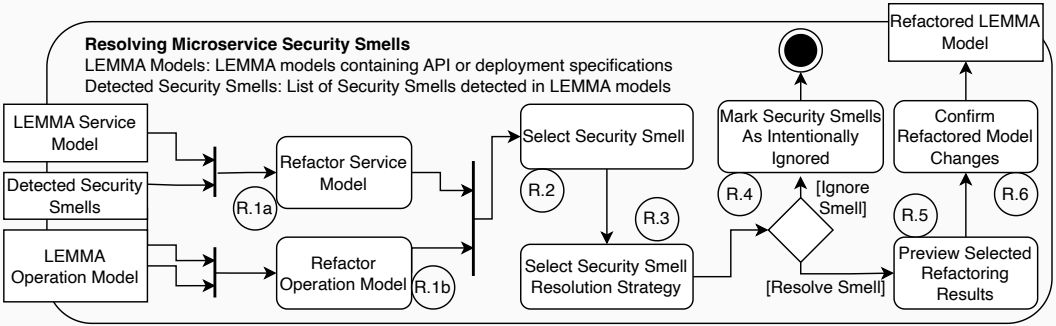


Figure 5: Activities of resolving security smells.

Resolving Microservice Security Smells

Listing 14: Refactored CustomerCore operation model.

```
...
@technology(deploymentBase)
@technology(protocolTechnology)
container CustomerCoreContainer ...
  depends on nodes
    infrastructure::APIGateway,
    infrastructure::ServiceDiscovery,
    infrastructure::H2Database
...}
```

Listing 15: Generated API Gateway operation model.

```
import ...
@technology(Zuul)
APIGateway is Zuul::_infrastructure.Zuul
  depends on nodes ServiceDiscovery
  used by services
    coreService::com.lakeside.CustomerCore,
  used by nodes container::CustomerCoreContainer {
  default values {
    hostname = "APIGateway"
    apiUri = "localhost:8080"
  }
}
 ...
```

Table of Content

Proof of Concept Implementation

Figure 6: Detecting security smells in Eclipse.

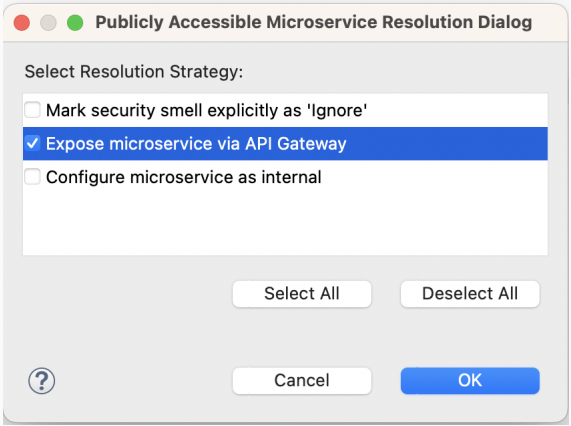
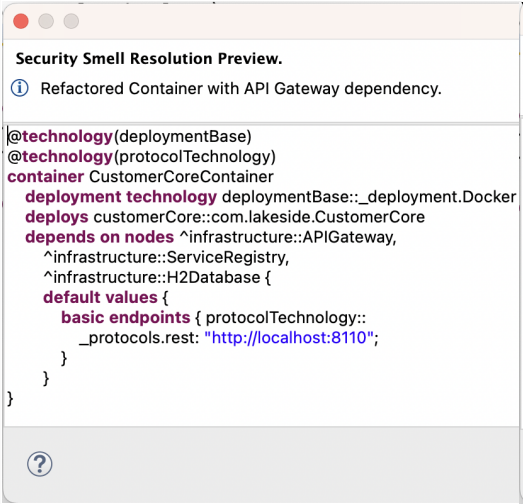
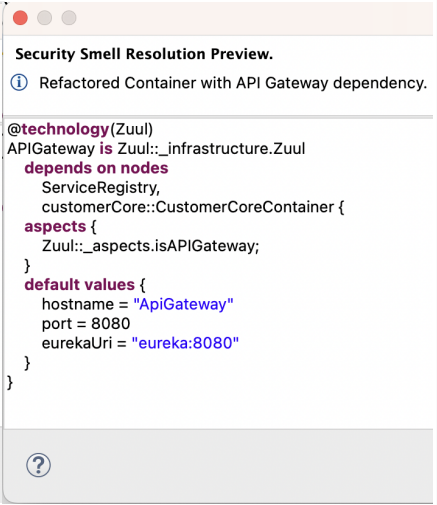


Figure 7: Select refactoring strategy in Eclipse.

Proof of Concept Implementation



(a) Preview of the extension with an API Gateway of the infrastructure operation model.



(b) Preview of the adaption of the Customer Core operation model with API Gateway dependency.

Table of Content

Conclusion

Summary:

- Model-driven resolution of security smells
  - Insufficient access control
  - Publicly accessible microservices
- Extensible approach due to LEMMAs expressiveness

Future Work:

- Software Architecture Reconstruction to construct models
- Code generator to refactor source code
- Extend analyze functionalities

Conclusion

Philip Wizenty, M.Sc.

- Ph.D. Student at IDiAL Institute
- Scientific Profile: ORCID, ResearchGate
- Contact: E-Mail, LinkedIn, XING, GitHub

Towards Resolving Security Smells in Microservices, Model-Driven

- Model microservice architecture
- Analyze models to identify security smells
- Rector model for resolving security smells

Fachhochschule Dortmund
University of Applied Sciences and Arts

IDiAL Institut für die Digitalisierung von Arbeits- und Lebenswelten

Questions

Questions?

Literatur

[1] Len Bass, Paul Clements, and Rick Kazman. **Software architecture in practice.** Addison-Wesley Professional, 2003.

[2] Benoit Combemale et al. **Engineering modeling languages: Turning domain knowledge into tools.** CRC Press, 2016.

[3] Robert France and Bernhard Rumpe. **"Model-driven development of complex software: A research roadmap."** In: Future of Software Engineering (FOSE'07). IEEE. 2007, pp. 37–54.

Literatur

[4] ISO/IEC/IEEE.
Systems and software engineering — Architecture description. Standard ISO/IEC/IEEE 42010:2011(E). 2011.

[5] Francisco Ponce et al. **“Should Microservice Security Smells Stay or be Refactored? Towards a Trade-off Analysis.”** In: Software Architecture. Ed. by Ilias Gerostathopoulos et al. Springer International Publishing, 2022, pp. 131–139.

[6] Francisco Ponce et al. **“Smells and refactorings for microservices security: A multivocal literature review.”** In: Journal of Systems and Software 192 (2022), p. 111393.

Literatur

[7] Robert Stahlbock. **Advances in Data Science and Information Engineering.** Ed. by Gary M. WeissMahmoud Abou-NasrCheng-Ying YangHamid R. ArabniaLeonidas Deligiannidis. Springer, Cham, 2021.

[8] Jon Whittle, John Hutchinson, and Mark Rouncefield. **"The State of Practice in Model-Driven Engineering."** In: IEEE Software 31.3 (May 2014). IEEE, pp. 79–85.