A Non-Intrusive Approach to Extend Microservice Modeling Languages with Architecture Pattern Support

Third International Conference on Microservices (Microservices 2020)

Florian Rademacher [1]

florian.rademacher@fh-dortmund.de

September 8–10, 2020

[1] University of Applied Sciences and Arts Dortmund, IDiAL Institute

Table of Contents

Introduction

Extending LEMMA with Architecture Pattern Support

Future Work

Table of Contents

Introduction

Extending LEMMA with Architecture Pattern Support

Future Work

Introduction

- Introduction
  - Microservice Architecture (MSA) introduces challenges in the design, implementation, and operation of an application [10, 11, 3]
    - Design: e.g., efficient service tailoring
    - Implementation: e.g., manage technology heterogeneity
    - Operation: e.g., maintain deployment and operation infrastructure

Line of Research

Investigate architecture modeling languages [9] to support in coping with challenges.

Introduction

- Introduction
  - Microservice Architecture (MSA) introduces challenges in the design, implementation, and operation of an application [10, 11, 3]
    - Design: e.g., efficient service tailoring
    - Implementation: e.g., manage technology heterogeneity
    - Operation: e.g., maintain deployment and operation infrastructure

🎓 Line of Research

Investigate architecture modeling languages [9] to support in coping with challenges.

Introduction

- Development of LEMMA[1]

Goals	Solution Building Blocks
Mitigate complexity of MSA engineering for stakeholder groups	Provide modeling languages that are aligned to stakeholder concerns
Balance conceptual design and technology-specific implementation	Provide selective level of technology abstraction
Automate architecting and implementation tasks	Provide framework for model processor implementation

[1] Language Ecosystem for Modeling Microservice Architecture (`https://fh.do/lemma`)

Introduction

- Development of LEMMA[1]

Goals	Solution Building Blocks
🏁 Mitigate complexity of MSA engineering for stakeholder groups	💡 Provide modeling languages that are aligned to stakeholder concerns
🏁 Balance conceptual design and technology-specific implementation	💡 Provide selective level of technology abstraction
🏁 Automate architecting and implementation tasks	💡 Provide framework for model processor implementation

[1] Language Ecosystem for Modeling Microservice Architecture (`https://fh.do/lemma`)

Introduction

- Development of LEMMA[1]

Goals	Solution Building Blocks
🏁 Mitigate complexity of MSA engineering for stakeholder groups	💡 Provide modeling languages that are aligned to stakeholder concerns
🏁 Balance conceptual design and technology-specific implementation	💡 Provide selective level of technology abstraction
🏁 Automate architecting and implementation tasks	💡 Provide framework for model processor implementation

[1] Language Ecosystem for Modeling Microservice Architecture (`https://fh.do/lemma`)

Introduction

- Development of LEMMA[1]

Goals	Solution Building Blocks
🏁 Mitigate complexity of MSA engineering for stakeholder groups	💡 Provide modeling languages that are aligned to stakeholder concerns
🏁 Balance conceptual design and technology-specific implementation	💡 Provide selective level of technology abstraction
🏁 Automate architecting and implementation tasks	💡 Provide framework for model processor implementation

[1] Language Ecosystem for Modeling Microservice Architecture (`https://fh.do/lemma`)

Introduction

- Development of LEMMA[1]

Goals	Solution Building Blocks
🏁 Mitigate complexity of MSA engineering for stakeholder groups	💡 Provide modeling languages that are aligned to stakeholder concerns
🏁 Balance conceptual design and technology-specific implementation	💡 Provide selective level of technology abstraction
🏁 Automate architecting and implementation tasks	💡 Provide framework for model processor implementation

[1] Language Ecosystem for Modeling Microservice Architecture (`https://fh.do/lemma`)

Introduction

- Development of LEMMA[1]

Goals	Solution Building Blocks
🏁 Mitigate complexity of MSA engineering for stakeholder groups	💡 Provide modeling languages that are aligned to stakeholder concerns
🏁 Balance conceptual design and technology-specific implementation	💡 Provide selective level of technology abstraction
🏁 Automate architecting and implementation tasks	💡 Provide framework for model processor implementation

[1] Language Ecosystem for Modeling Microservice Architecture (`https://fh.do/lemma`)

Introduction

- Development of LEMMA[1]

Goals	Solution Building Blocks
🏁 Mitigate complexity of MSA engineering for stakeholder groups	💡 Provide modeling languages that are aligned to stakeholder concerns
🏁 Balance conceptual design and technology-specific implementation	💡 Provide selective level of technology abstraction
🏁 Automate architecting and implementation tasks	💡 Provide framework for model processor implementation

[1] Language Ecosystem for Modeling Microservice Architecture (`https://fh.do/lemma`)

Introduction

- LEMMA Model Examples

Domain Model

```
1  // Model file: Banking.data
2  context Banking {
3    structure Account<entity> {
4      long id<identifier>,
5      Person owner,
6      double balance
7    }
8
9    structure Person {
10     string firstname,
11     string lastname,
12     date birthday
13   }
14 }
```

Service Model

```
1  // Model file: Banking.services
2  import datatypes from "Banking.data" as Domain
3  functional microservice org.example.BankingService {
4    interface AccountManagement {
5      ---
6      API endpoint for creating a new account
7      @required account The new account
8      ---
9      createAccount(
10       sync in account : Domain::Banking.Account,
11       sync out accountId : long
12     );
13   }
14 }
```

- By intent, no integrated modeling concepts for technologies (e.g., JDL[2]) and patterns (e.g., MicroDSL [12] or MiSAR [1]) ⇒ Keep language core concise, foster learnability

[2] https://www.jhipster.tech/jdl

Introduction

- LEMMA Model Examples

Domain Model

```
// Model file: Banking.data
context Banking {
  structure Account<entity> {
    long id<identifier>,
    Person owner,
    double balance
  }

  structure Person {
    string firstname,
    string lastname,
    date birthday
  }
}
```

Service Model

```
// Model file: Banking.services
import datatypes from "Banking.data" as Domain
functional microservice org.example.BankingService {
  interface AccountManagement {
    ---
    API endpoint for creating a new account
    @required account The new account
    ---
    createAccount(
      sync in account : Domain::Banking.Account,
      sync out accountId : long
    );
  }
}
```

- By intent, no integrated modeling concepts for technologies (e.g., JDL[2]) and patterns (e.g., MicroDSL [12] or MiSAR [1]) ⇒ Keep language core concise, foster learnability

[2] https://www.jhipster.tech/jdl

Introduction

- LEMMA Model Examples

Domain Model

```
// Model file: Banking.data
context Banking {
  structure Account<entity> {
    long id<identifier>,
    Person owner,
    double balance
  }

  structure Person {
    string firstname,
    string lastname,
    date birthday
  }
}
```

Service Model

```
// Model file: Banking.services
import datatypes from "Banking.data" as Domain
functional microservice org.example.BankingService {
  interface AccountManagement {
    ---
    API endpoint for creating a new account
    @required account The new account
    ---
    createAccount(
      sync in account : Domain::Banking.Account,
      sync out accountId : long
    );
  }
}
```

- By intent, no integrated modeling concepts for technologies (e.g., JDL[2]) and patterns (e.g., MicroDSL [12] or MiSAR [1]) ⇒ Keep language core concise, foster learnability

[2] https://www.jhipster.tech/jdl

Introduction

- LEMMA Model Examples

Domain Model

```
// Model file: Banking.data
context Banking {
  structure Account<entity> {
    long id<identifier>,
    Person owner,
    double balance
  }

  structure Person {
    string firstname,
    string lastname,
    date birthday
  }
}
```

Service Model (with technology metadata)

```
// Model file: Banking.services
import datatypes from "Banking.data" as Domain
import technology from "Spring.technology" as spring
@technology(spring)
functional microservice org.example.BankingService {
  interface AccountManagement {
    ---
    API endpoint for creating a new account
    @required account The new account
    ---
    @endpoints(spring::_protocols.rest: "/accounts";)
    @spring::_aspects.PostMapping
    createAccount(
      sync in account : Domain::Banking.Account,
      sync out accountId : long
    );
  }
}
```

Introduction

- What is LEMMA usable for?
  - Code generation [7]
  - Semi-automatic transformation of tactical Domain-driven Design models into executable microservices [2, 6]
  - Reconstruction and model-based quality analysis of microservice architectures [5]
  - Now: Support integration of architecture patterns, e.g., Event Sourcing or CQRS [8], in models without having to adapt LEMMA's modeling languages

Introduction

- What is LEMMA usable for?
  - Code generation [7]
  - Semi-automatic transformation of tactical Domain-driven Design models into executable microservices [2, 6]
  - Reconstruction and model-based quality analysis of microservice architectures [5]
  - Now: Support integration of architecture patterns, e.g., Event Sourcing or CQRS [8], in models without having to adapt LEMMA's modeling languages

Table of Contents

Introduction

Extending LEMMA with Architecture Pattern Support

Future Work

Extending LEMMA with Architecture Pattern Support

- Extending LEMMA with Architecture Pattern Support

Goals

🏁 Flexible, need-based pattern addition/usage

🏁 Check pattern compliance at design time

🏁 Interactive fixing of pattern compliance violations during model construction

Solution Steps

🔧 Construct a pattern-specific technology model

🔧 Implement a pattern-specific model validator with LEMMA's model processing framework. The framework integrates with the Language Server Protocol (LSP)[3].

[3] https://microsoft.github.io/language-server-protocol

Extending LEMMA with Architecture Pattern Support

- Extending LEMMA with Architecture Pattern Support

Goals	Solution Steps
🏁 Flexible, need-based pattern addition/usage	🔧 Construct a pattern-specific technology model
🏁 Check pattern compliance at design time	🔧 Implement a pattern-specific model validator with LEMMA's model processing framework. The framework integrates with the Language Server Protocol (LSP)[3].
🏁 Interactive fixing of pattern compliance violations during model construction	

[3] https://microsoft.github.io/language-server-protocol

Extending LEMMA with Architecture Pattern Support

- Extending LEMMA with Architecture Pattern Support

Goals	Solution Steps
🏁 Flexible, need-based pattern addition/usage	🔧 Construct a pattern-specific technology model
🏁 Check pattern compliance at design time	🔧 Implement a pattern-specific model validator with LEMMA's model processing framework. The framework integrates with the Language Server Protocol (LSP)[3].
🏁 Interactive fixing of pattern compliance violations during model construction	

[3] https://microsoft.github.io/language-server-protocol

Extending LEMMA with Architecture Pattern Support

- Extending LEMMA with Architecture Pattern Support

Goals

- 🏁 Flexible, need-based pattern addition/usage
- 🏁 Check pattern compliance at design time
- 🏁 Interactive fixing of pattern compliance violations during model construction

Solution Steps

- 🔧 Construct a pattern-specific technology model
- 🔧 Implement a pattern-specific model validator with LEMMA's model processing framework. The framework integrates with the Language Server Protocol (LSP)[3].

[3] https://microsoft.github.io/language-server-protocol

Extending LEMMA with Architecture Pattern Support

- Extending LEMMA with Architecture Pattern Support

Goals

- 🏁 Flexible, need-based pattern addition/usage
- 🏁 Check pattern compliance at design time
- 🏁 Interactive fixing of pattern compliance violations during model construction

Solution Steps

- 🔧 Construct a pattern-specific technology model
- 🔧 Implement a pattern-specific model validator with LEMMA's model processing framework. The framework integrates with the Language Server Protocol (LSP)[3].

[3] `https://microsoft.github.io/language-server-protocol`

Extending LEMMA with Architecture Pattern Support

- Example from the Event Sourcing pattern [8]
  - Event Producer concept:
    1. Event producers are microservice operations that create and send domain events
    2. The sending of domain events happens asynchronously

Event Producers in LEMMA

Event Producers need to be modeled as microservice operations with an asynchronous result parameter

Extending LEMMA with Architecture Pattern Support

- Example from the Event Sourcing pattern [8]
  - Event Producer concept:
    1. Event producers are microservice operations that create and send domain events
    2. The sending of domain events happens asynchronously

Event Producers in LEMMA

Event Producers need to be modeled as microservice operations with an asynchronous result parameter

Extending LEMMA with Architecture Pattern Support

- Example from the Event Sourcing pattern: Construct technology model

```
1 // Model file name: EventSourcing.technology
2 technology EventSourcing {
3   service aspects {
4     aspect Producer for operations;
5   }
6 }
```

Listing 1: LEMMA technology model for the Event Sourcing pattern.

Extending LEMMA with Architecture Pattern Support

- Example from the Event Sourcing pattern: Construct technology model

Event Producers may be modeled as microservice operations (structural constraint)

```
1 // Model f
2 technology
3   service aspects {
4     aspect Producer for operations;
5   }
6 }
```

Listing 1: LEMMA technology model for the Event Sourcing pattern.

Extending LEMMA with Architecture Pattern Support

- Example from the Event Sourcing pattern: Apply technology model

```
1  // Model file name: Banking.services
2  import datatypes from "Banking.data" as Domain
3  import technology from "EventSourcing.technology" as EventSourcing
4
5  @technology(EventSourcing)
6  functional microservice org.example.BankingService {
7    @EventSourcing::_aspects.Producer
8    sendAccountCreatedEvent(
9      async out event : Banking.AccountCreatedEvent
10   );
11 }
```

Listing 2: LEMMA technology model for the Event Sourcing pattern.

Extending LEMMA with Architecture Pattern Support

- Example from the Event Sourcing pattern: Implement model validator

```kotlin
// Kotlin file: ServiceModelSourceValidator.kt in EventSourcingValidator.jar
@SourceModelValidator
class ServiceModelSourceValidator : AbstractXtextSourceModelValidator() {
  @Check
  private fun checkProducer(operation: Operation) {
    if (operation.hasAspect("EventSourcing.Producer") && !operation.hasAsynchronousResultParameter())
      error("The Producer aspect may only be applied to operations with a result parameter",
        ServicePackage.Literals.OPERATION__NAME)
  }
}
```

Listing 3: Constraint validation within the Event Sourcing Validator (Kotlin).

Extending LEMMA with Architecture Pattern Support

- Example from the Event Sourcing pattern: Implement model validator

```kotlin
// Kotlin file: ServiceModelSourceValidator.kt in EventSourcingValidator.jar
@SourceModelValidator
class ServiceModelSourceValidator : AbstractXtextSourceModelValidator() {
  @Check
  private fun checkProducer(operation: Operation) {
    if (operation.hasAspect("EventSourcing.Producer") && !operation.hasAsynchronousResultParameter())
      error("The Producer aspect may only be applied to operations with a result parameter",
        ServicePackage.Literals.OPERATION__NAME)
  }
}
```

Listing 3: Constraint validation within the Event Sourcing Validator (Kotlin).

Extending LEMMA with Architecture Pattern Support

- Example from the Event Sourcing pattern: Implement model validator

```
1  // Kotlin file: ServiceModelSourceValidator.kt
2  @SourceModelValidator
3  class ServiceModelSourceValidator : AbstractXte
4    @Check
5    private fun checkProducer(operation: Operation) {
6      if (operation.hasAspect("EventSourcing.Producer") && !operation.hasAsynchronousResultParameter())
7        error("The Producer aspect may only be applied to operations with a result parameter",
8          ServicePackage.Literals.OPERATION__NAME)
9    }
10 }
```

Event Producers must have at least one asynchronous result parameter (semantic constraint)

Listing 3: Constraint validation within the Event Sourcing Validator (Kotlin).

Extending LEMMA with Architecture Pattern Support

- Example from the Event Sourcing pattern: Implement model validator

```
1  // Kotlin file: ServiceModelSourceValidator.kt
2  @SourceModelValidator
3  class ServiceModelSourceValidator : AbstractXte
4    @Check
5    private fun checkProducer(operation: Operation) {
6      if (operation.hasAspect("EventSourcing.Producer") && !operation.hasAsynchronousResultParameter())
7        error("The Producer aspect may only be applied to operations with a result parameter",
8          ServicePackage.Literals.OPERATION__NAME)
9    }
10 }
```

Event Producers must have at least one asynchronous result parameter (semantic constraint)

Listing 3: Constraint validation within the Event Sourcing Validator (Kotlin).

Extending LEMMA with Architecture Pattern Support

- Example from the Event Sourcing pattern: Implement model validator
  - LEMMA Live Validation: Connect with IDE via LSP and display markers for source model validation errors

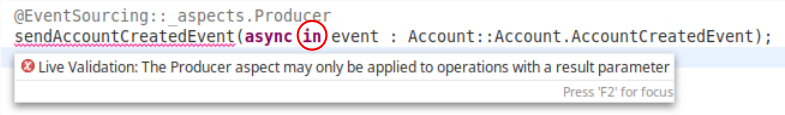

Figure 1: Pattern constraint violation displayed at design time in Eclipse

Table of Contents

Introduction

Extending LEMMA with Architecture Pattern Support

Future Work

Future Work

- Future Work
  - Lower complexity of pattern integration process
    - Employ Object Constraint Language (OCL) [4] to specify semantic pattern constraints
    - Generate LEMMA model validators from OCL models
  - Capture additional patterns for use with LEMMA, e.g., Saga and API Composition [8]

Future Work

- Future Work
  - Lower complexity of pattern integration process
    - Employ Object Constraint Language (OCL) [4] to specify semantic pattern constraints
    - Generate LEMMA model validators from OCL models
  - Capture additional patterns for use with LEMMA, e.g., Saga and API Composition [8]

Literature

[1] Nuha Alshuqayran, Nour Ali, and Roger Evans. “Towards Micro Service Architecture Recovery: An Empirical Study.” In: 2018 IEEE International Conference on Software Architecture (ICSA). IEEE, 2018, pp. 47–56.

[2] Eric Evans. Domain-Driven Design. Addison-Wesley, 2004.

[3] Welder Pinheiro Luz, Gustavo Pinto, and Rodrigo Bonifácio. “Building a Collaborative
Culture: A Grounded Theory of Well Succeeded DevOps Adoption in Practice.” In: Proc. of the 12th ACM/IEEE Int. Symp. on Empirical Softw. Eng. and Measurement. ESEM ’18. Oulu, Finland: ACM, 2018, 6:1–6:10.

Literature

[4] OMG. Object Constraint Language Version 2.4. Standard formal/2014-02-03. Object Management Group, 2014.

[5] Florian Rademacher, Sabine Sachweh, and Albert Zündorf. “A Modeling Method for Systematic Architecture Reconstruction of Microservice-Based Software Systems.” In: Enterprise, Business-Process and Information Systems Modeling. Ed. by Selmin Nurcan et al. Springer, May 2020, pp. 311–326.

[6] Florian Rademacher, Sabine Sachweh, and Albert Zündorf. “Deriving Microservice Code from Underspecified Domain Models Using DevOps-Enabled Modeling Languages and Model Transformations.” In: Proc. of the 46th Euromicro Conf. on Softw. Eng. and Advanced Applications (SEAA). IEEE, 2020, pp. 229–236.

Literature

[7] Florian Rademacher et al. “Graphical and Textual Model-Driven Microservice Development.” In: Microservices: Science and Engineering. Ed. by Antonio Bucchiarone et al. Springer, 2020, pp. 147–179.

[8] Chris Richardson. Microservices Patterns. First. Manning Publications, 2019.

[9] Davide Di Ruscio et al. “Developing next generation ADLs through MDE techniques.” In: 2010 ACM/IEEE 32nd International Conference on Software Engineering. Vol. 1. IEEE, 2010, pp. 85–94.

[10] Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan Van Den Heuvel. “The pains and gains of microservices: A Systematic grey literature review.” In: Journal of Systems and Software 146 (2018). Elsevier, pp. 215–232.

Literature

[11] Davide Taibi and Valentina Lenarduzzi. “On the Definition of Microservice Bad Smells.” In: IEEE Software 35.3 (May 2018). IEEE, pp. 56–62.

[12] Branko Terzić et al. “Development and evaluation of MicroBuilder: a Model-Driven tool for the specification of REST Microservice Software Architectures.” In: Enterprise Information Systems 12.8-9 (2018). Taylor & Francis, pp. 1034–1057. eprint: https://doi.org/10.1080/17517575.2018.1460766.