## Service Computation 2021

The Thirteenth International Conference on Advanced Service Computing

**April 18, 2021 to April 22, 2021 - Porto, Portugal**

IARIA

# Executable Architectures for Complex Software Systems

Sebastian Apel
Technische Hochschule Ingolstadt
Germany

**Thomas M. Prinz** (Presenter)
Course Evaluation Service,
Friedrich Schiller University Jena, Germany
Thomas.Prinz@uni-jena.de

FRIEDRICH-SCHILLER-UNIVERSITÄT JENA

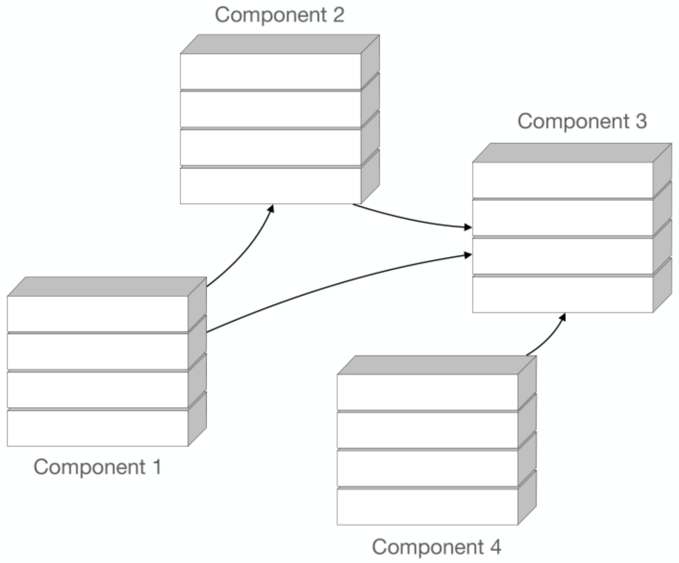
## About the presenter

Photo: Anne Günther (University Jena)

- Diploma in computer science at Friedrich Schiller University Jena (FSU), Germany (2010)
- Ph.D. in computer science at FSU (Dr. rer. nat., 2017)
- Since 2017, researcher and software architect at the Course Evaluation Service, FSU

- Research in:
    - Compiler construction
    - Business process verification and management
    - Software engineering
    - Human Computer Interaction (HCI)
    - Evaluation theory

## How to build a system today

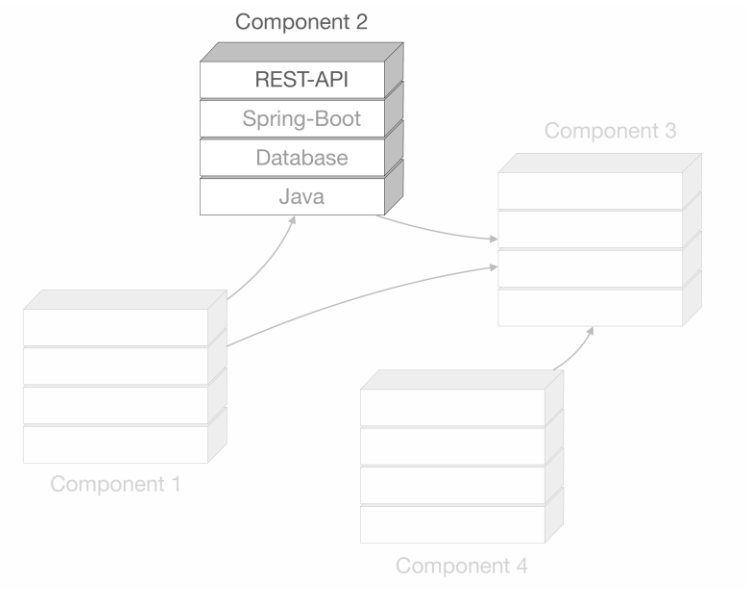


- Architectures describe abstract components
- They further describe how they interact / communicate

- Modern architecture styles like *microservices* separate software into small independent services (components)
- They interact in a network

FRIEDRICH-SCHILLER-UNIVERSITÄT JENA

# Motivation



- Each component has its own individual tool stack and runtime environment

✓ Proper separation of functionality
✓ High availability for reuse
✓ Exchangeable

Business Logic

```java
public class MyBusinessModel {
    public String doSomething() {
        return "Hello, World!";
    }
}
```

- ➕ Persistence Layer
- ➕ Service Layer
- ➕ Data Transfer Objects
- ➕ Dependency Management
- ➕ Continuous Integration
- ➕ Container Descriptor

## Motivation

**BUT:**

- There is a gap between architecture description and implementation
    - No translation from architecture to implementation
    - The implementation does not automatically result from the architecture
    - Developers necessary for different abstraction levels
- Overhead of 1:3 in implementation [Apel2019]:
    - 300 lines of organizational code (communication, mapping, etc.)
    - 100 lines of functional code

**GOAL:**

- **Benefit in time, robustness, and correctness if everyone can focus on functionality only**

[Apel2019] S. Apel, F. Hertrampf, and S. Späthe, "Towards a Metrics-Based Software Quality Rating for a Microservice Architecture - Case Study for a Measurement and Processing Infrastructure," in Innovations for Community Services - 19th International Conference, I4CS 2019, Wolfsburg, Germany, June 24-26, 2019, Proceedings, ser. Communications in Computer and Information Science, K. Lüke, G. Eichler, C. Erfurth, and G. Fahrnberger, Eds., vol. 1041. Springer, 2019, pp. 205–220.

## Idea

1. Meta programming language
2. Compilation
3. Automation
4. Integrated development environment (IDE)

## Meta Programming Language

- Allows to implement in different programming languages (85% of software engineers use multiple languages during development [Zhang2019])
- Can be an extension of an existing programming language (like Java)
- Has its own compiler and runtime environment that separates the software

[Zhang2019] H. Zhang, S. Li, Z. Jia, C. Zhong, and C. Zhang, “Microservice architecture in reality: An industrial inquiry,” in IEEE International Conference on Software Architecture, ICSA 2019, Hamburg, Germany, March 25-29, 2019. IEEE, 2019, pp. 51–60.

```
class Pair {
  public int a, b;
  Pair(int a, int b) {
    this.a = a; this.b = b;
  }
}
class Computation {
  @Java
  public int handlePairs(int[][] pairs) {
    Pair[] pairList = new Pair[pairs.length];
    for (int i = 0; i < pairs.length; i++) {
      int a = pairs[i][0], b = pairs[i][1];
      pairList[i] = new Pair(a, b);
    }
    return this.computeSums(pairList);
  }
  @R
  public int[] computeSums(Pair[] pairs) {
    sapply(pairs, function(pair) {
      pair$a + pair$b
    })
  }
}
```

FRIEDRICH-SCHILLER-UNIVERSITÄT JENA

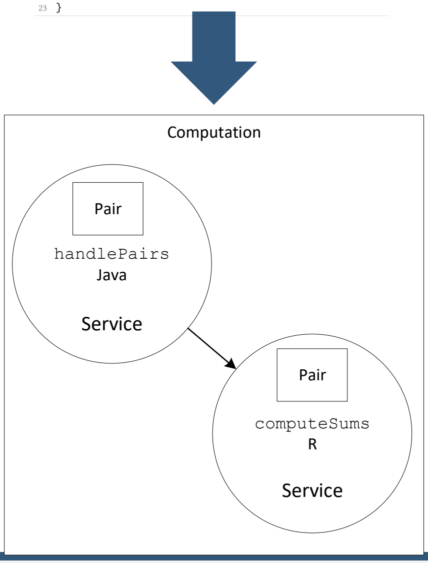
## Meta Programming Language

- ✓ Communication interfaces are easy to identify and to verify
- ✓ Data models are implemented once
- ✓ No mapping of input and output parameters

- Should allow data-orientation with streams
- Should allow to define processes

```
class Pair {
  public int a, b;
  Pair(int a, int b) {
    this.a = a; this.b = b;
  }
}
class Computation {
  @Java
  public int handlePairs(int[][] pairs) {
    Pair[] pairList = new Pair[pairs.length];
    for (int i = 0; i < pairs.length; i++) {
      int a = pairs[i][0], b = pairs[i][1];
      pairList[i] = new Pair(a, b);
    }
    return this.computeSums(pairList);
  }
  @R
  public int[] computeSums(Pair[] pairs) {
    sapply(pairs, function(pair) {
      pair$a + pair$b
    })
  }
}
```

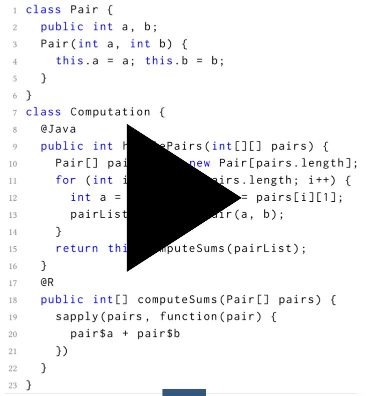
```
class Pair {
  public int a, b;
  Pair(int a, int b) {
    this.a = a; this.b = b;
  }
}
class Computation {
  @Java
  public int h[illegible]ePairs(int[][] pairs) {
    Pair[] pai[illegible]new Pair[pairs.length];
    for (int i[illegible]irs.length; i++) {
      int a = [illegible]= pairs[i][1];
      pairList[illegible]ir(a, b);
    }
    return thi[illegible]puteSums(pairList);
  }
  @R
  public int[] computeSums(Pair[] pairs) {
    sapply(pairs, function(pair) {
      pair$a + pair$b
    })
  }
}
```



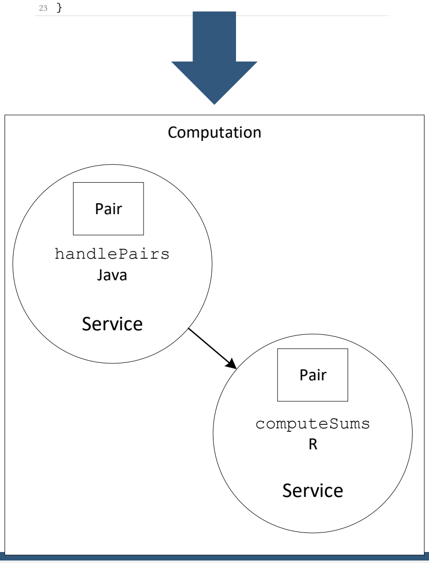
# Compilation

1. Interpretation
   - Fast error detection
   - Debugging
   - Bottlenecks identification

2. Compilation
   - Static analyses
   - Increase performance
   - Optimization



```
class Pair {
  public int a, b;
  Pair(int a, int b) {
    this.a = a; this.b = b;
  }
}
class Computation {
  @Java
  public int h[illegible]ePairs(int[][] pairs) {
    Pair[] pai[illegible]new Pair[pairs.length];
    for (int i[illegible]irs.length; i++) {
      int a = [illegible]= pairs[i][1];
      pairList[illegible]ir(a, b);
    }
    return thi[illegible]mputeSums(pairList);
  }
  @R
  public int[] computeSums(Pair[] pairs) {
    sapply(pairs, function(pair) {
      pair$a + pair$b
    })
  }
}
```

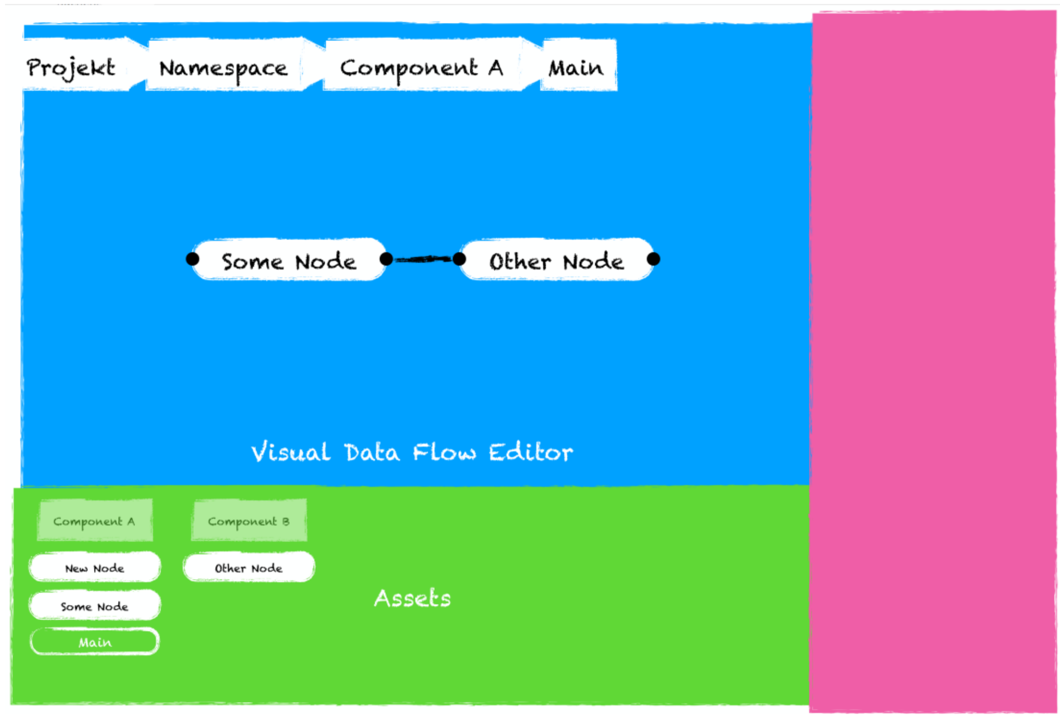


## Compilation / Automation

- Data models must be generated in all target languages that use them
- Surrounds code with persistence, communication, etc.
- Abstract functionality must be compiled into those languages best fitting the functionality's realization
- Compilation into different tool stacks
- Choosing appropriated tool stacks
- Generation of deployable artifacts

## Integrated development environment (IDE)

- IDE for the meta programming language
- IDE shall support all phases of software development
  - Planning
  - Analysis
  - Design
  - Implementation
  - Maintenance
- IDE knows complete system
  - Allows to support design / implementation
  - Avoids errors
- Shall reduce technical details



## Short discussion

- Not a complete new idea
    - Architecture description languages
    - ArchJava, Archface, etc.
    - **BUT:**
        - ✓ Usage of (new) concepts (microservices, libraries, business processes, continuous integration, etc.)
        - ✓ Service-orientation

✓ The meta language will not cover all use cases by default

- Seems to be centralized, independent service implementation may increase generalization and minimize coupling

✓ Allows agile software development and fast prototyping since the architecture can be extended successively

✓ Focus on what to do, not how to do it

FRIEDRICH-SCHILLER-UNIVERSITÄT JENA

Sebastian Apel
Technische Hochschule Ingolstadt
Germany
Email: sebastian.apel@thi.de

**Thomas M. Prinz**
Course Evaluation Service
Friedrich Schiller University Jena
Germany
Email: thomas.prinz@uni-jena.de

Photo: Anne Günther (University Jena)

## Thank you for your attention!

FRIEDRICH-SCHILLER-UNIVERSITÄT JENA