



IWST22 — International Workshop on Smalltalk Technologies
Novi Sad, Serbia; August 24th-26th, 2022

# Using Moose platform for the implementation of a Software Product Line according to Model-Based Delta-Oriented Programming

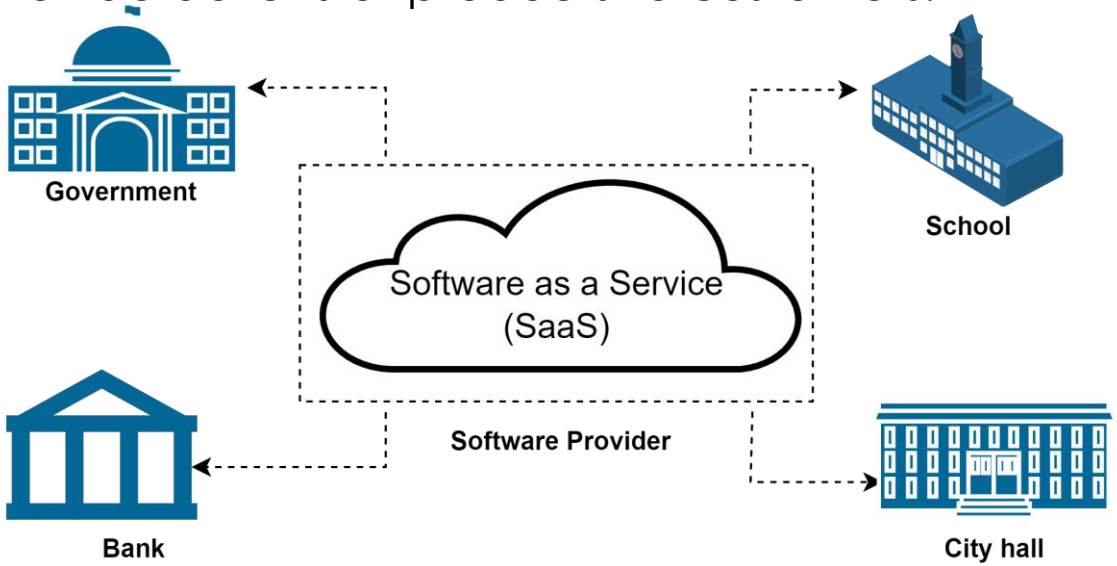
**Boubou Thiam Niang**, Giacomo Kahn, Nawel Amokrane, Yacine Ouzrout, Mustapha Derras and Jannik Laval


# Outline

- Context
- Illustrative case study
- Overview of the tool prototype
- Ongoing and future work

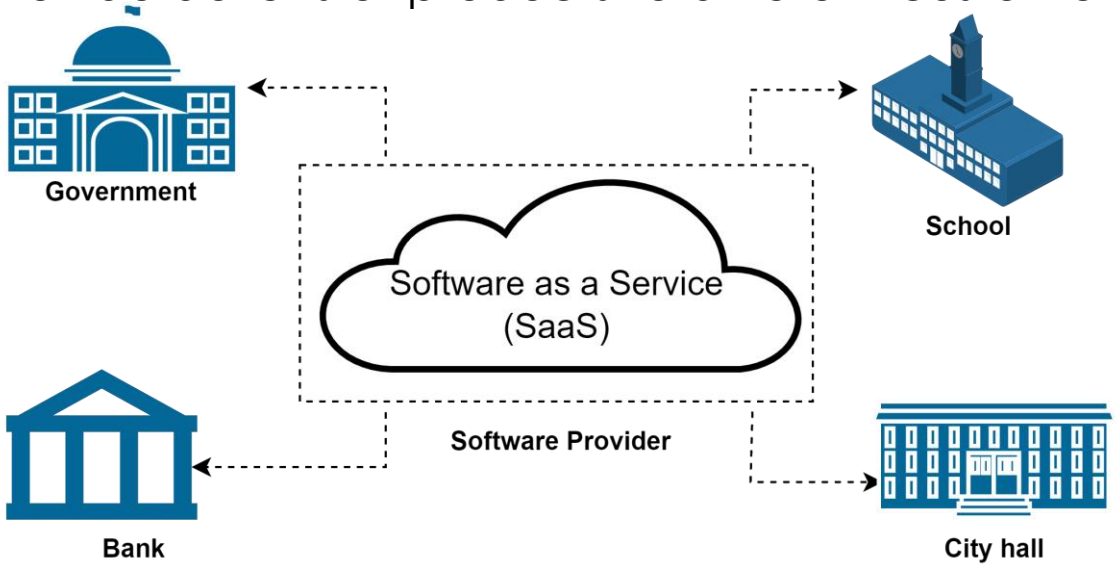
# Context – Software providers

- Software companies provide dozens of products to customers.

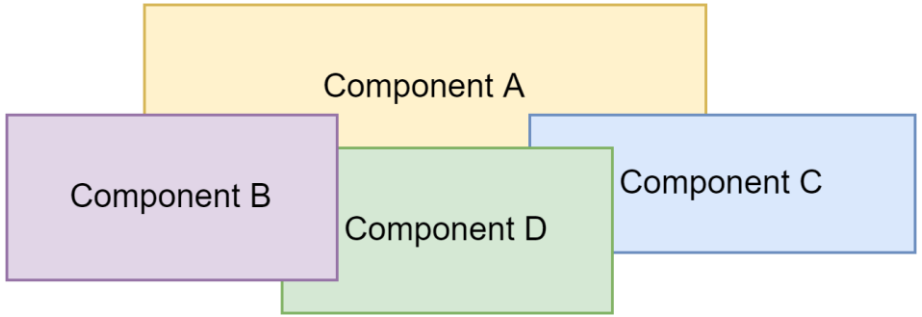


# Context – Software providers

- Software companies provide dozens of products to different customers.

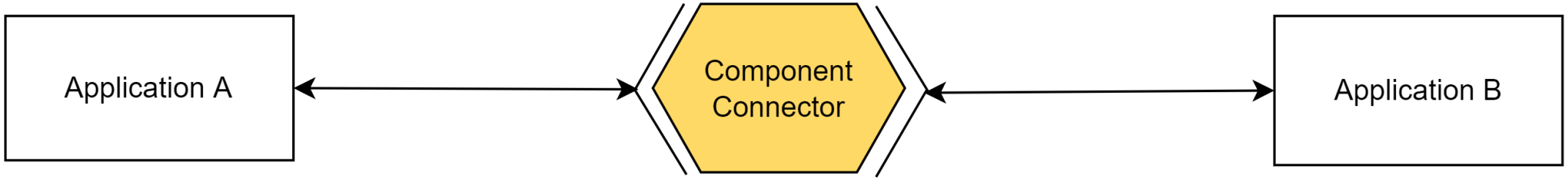



- Products can share common characteristics.



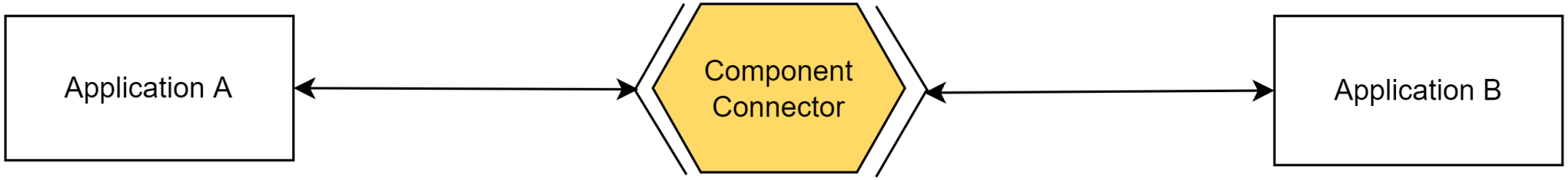
# Context – Constantly evolving systems

- Products represent business applications and technical components.



# Context – Constantly evolving systems

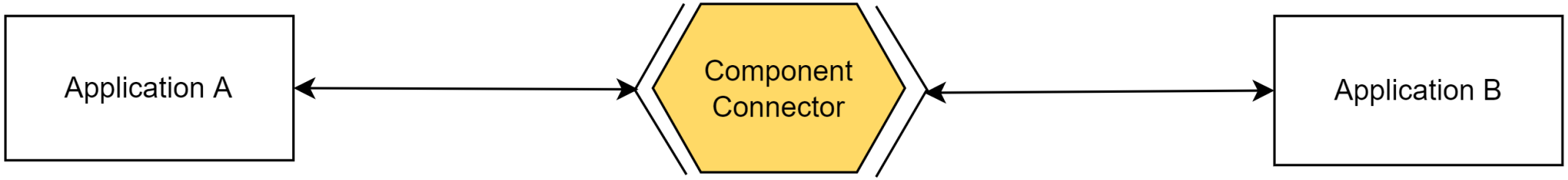
- Products represent business applications and technical components.




- Systems and components are constantly evolving.

# Context – Constantly evolving systems

- Products represent business applications and technical components.

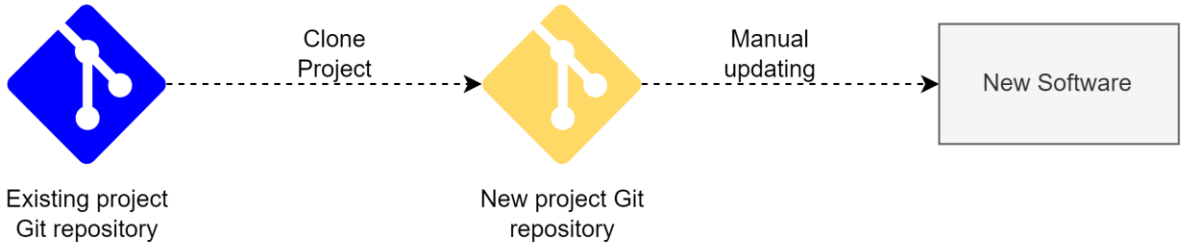



- Systems and components are constantly evolving.

- We need to increase the reusability of functionality across multiple products.

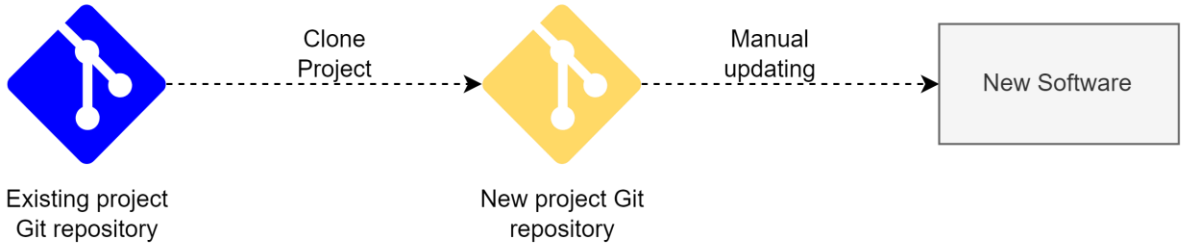
# Context – Common reuse approach

- Clone-and-Own (CaO)

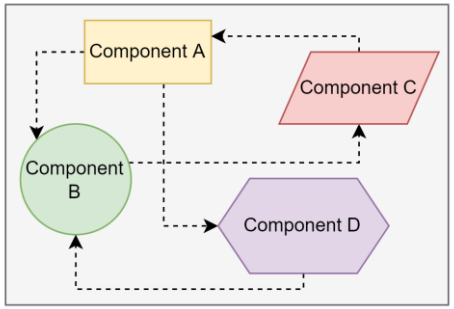


# Context – Common reuse approach

- Clone-and-Own (CaO)

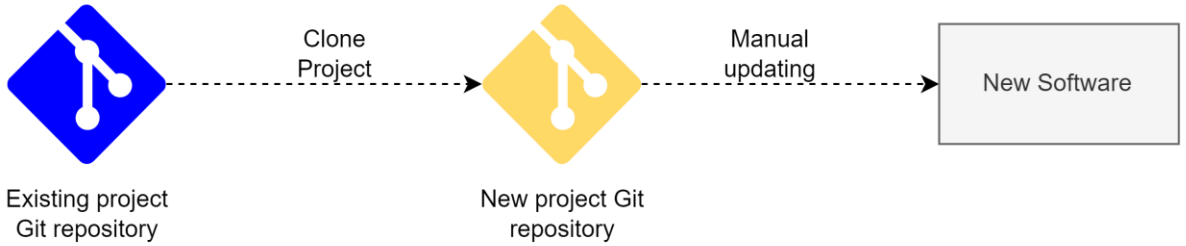



- Component-Based Software Engineering: limited possibility

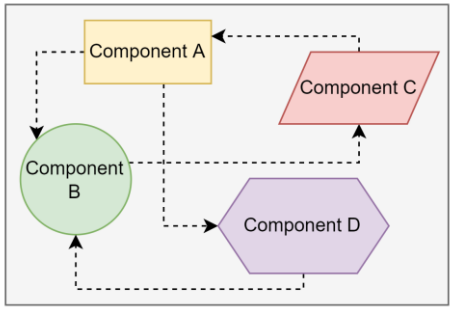


# Context – Common reuse approaches

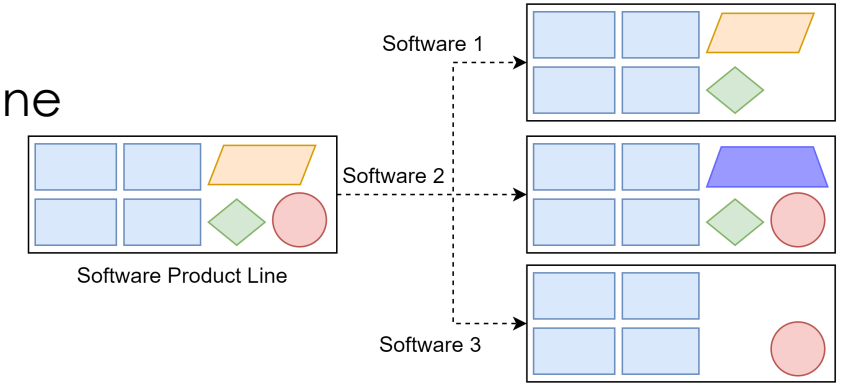
- Clone-and-Own (CaO)




- Component-Based Software Engineering: limited possibility

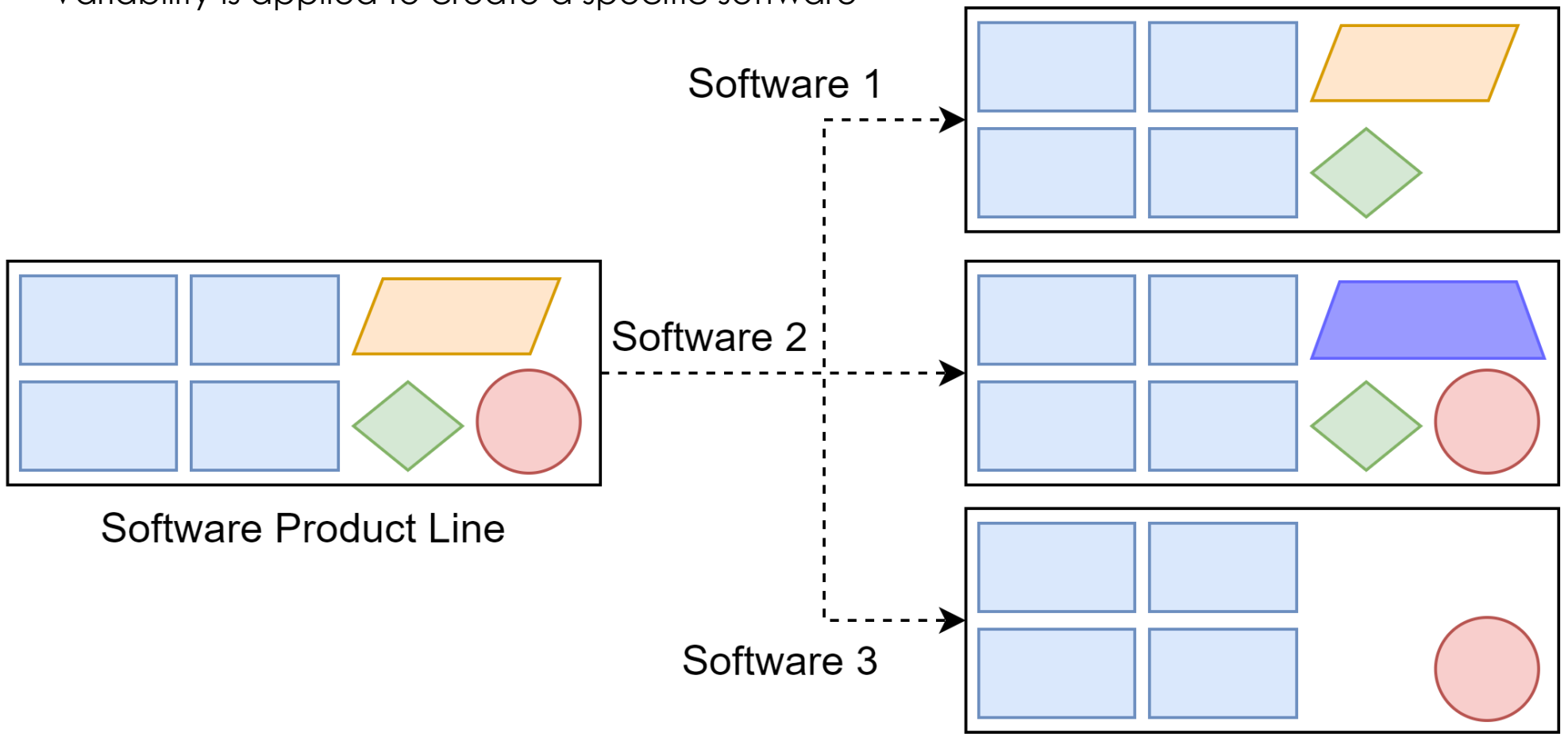



- Software Product Line

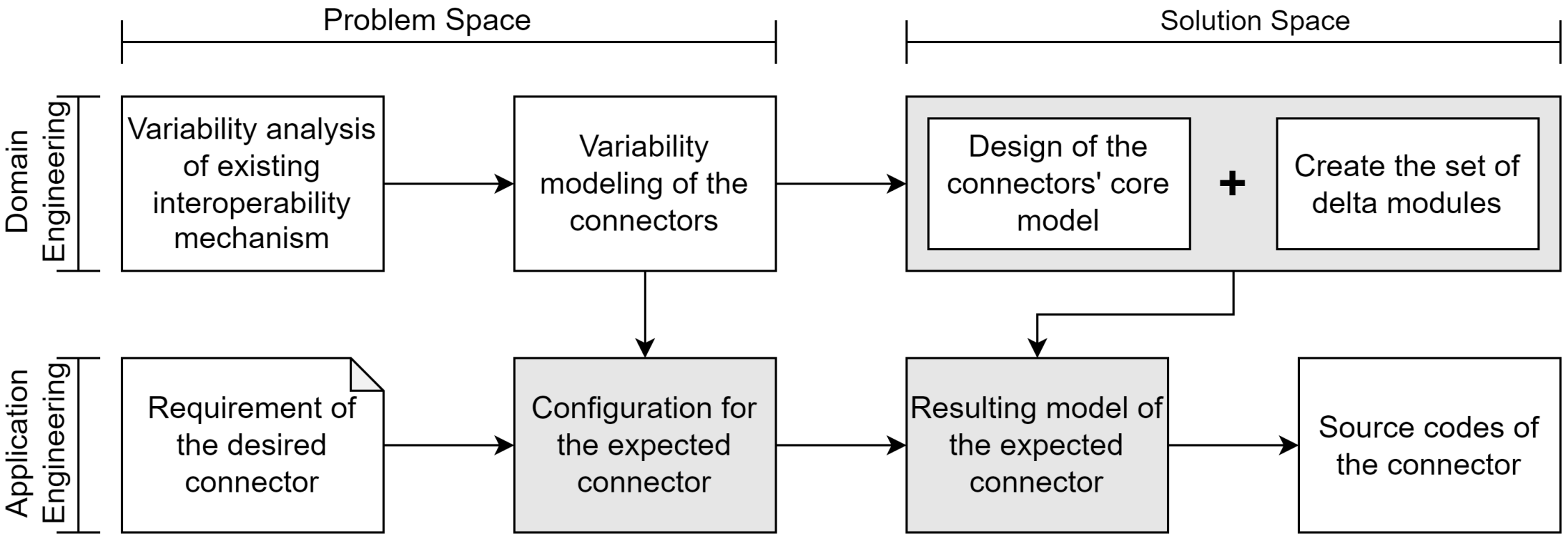


# Context – Software Product Line

- Software Product Line
  - A set of products with common
  - Variability is applied to create a specific software



# Context – SPL Engineering



# Context – SPL implementation

- Paradigms

| Paradigms | Approaches | Approaches comment |
|---|---|---|
| Feature oriented programming (FOP) [Ferreira et al., 2014] | Compositional (positive variability) | • Implement feature as distinct code units.<br>• Generate a product for a feature selection, the corresponding code units are determined and composed. |
| Aspect-Oriented Programming (AOP) [Kiczales et al., 1997] | | |
| **Delta-Oriented Programming (DOP)** [Schulze et al., 2013] | | |
| Preprocessor [Kästner 2013] | Annotative (Negative variability) | Implement features with some form of explicit or implicit annotations in the source code (Preprocessor #ifdef…endIf) |

# Context – SPL implementation

- Paradigms

| Paradigms | Approaches | Approaches comment |
|---|---|---|
| Feature oriented programming (FOP) [Ferreira et al., 2014] | Compositional (positive variability) | • Implement feature as distinct code units.<br>• Generate a product for a feature selection, the corresponding code units are determined and composed. |
| Aspect-Oriented Programming (AOP) [Kiczales et al., 1997] | | |
| **Delta-Oriented Programming (DOP)** [Schulze et al., 2013] | | |
| Preprocessor [Kästner 2013] | Annotative (Negative variability) | Implement features with some form of explicit or implicit annotations in the source code (Preprocessor #ifdef…endIf) |

- We opt for DOP paradigms

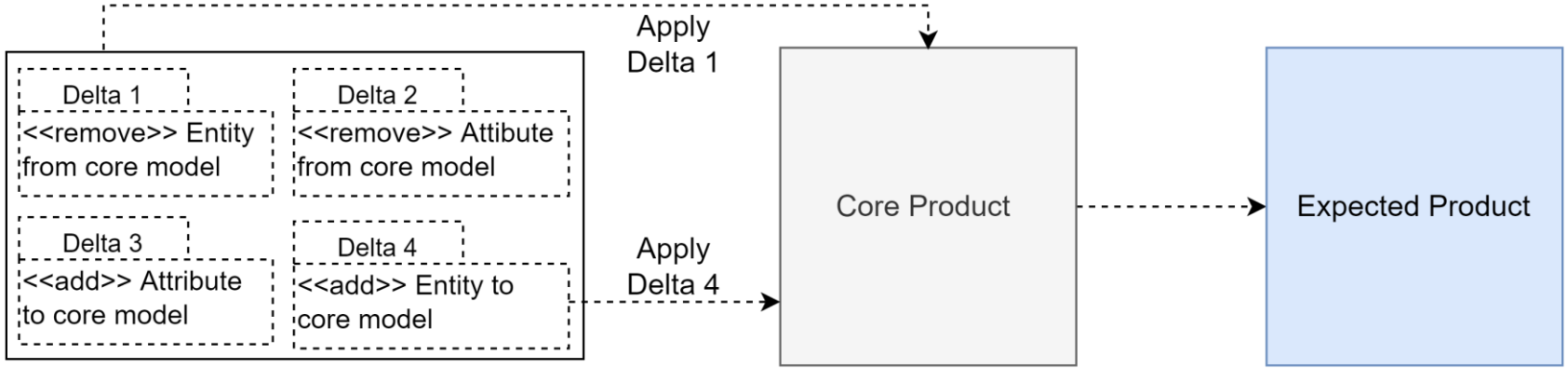
# Positioning – DOP paradigms

- Delta-Oriented Programming
  - A recent paradigm for the implementation of software product lines.

# Positioning – DOP paradigms

- Delta-Oriented Programming
  - A recent paradigm for the implementation of software product lines.

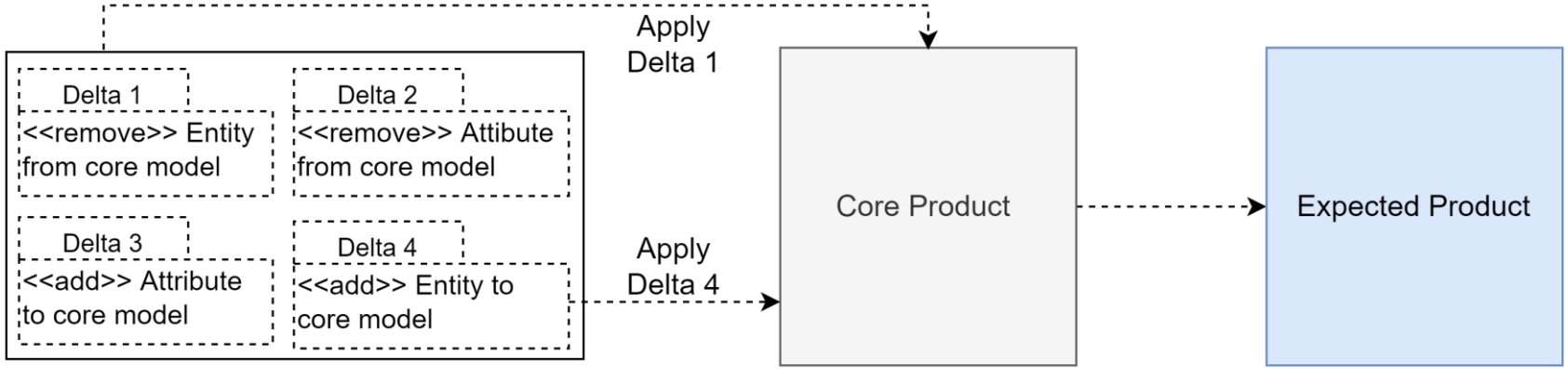
- Delta-Oriented Programming principle



# Positioning – DOP paradigms

- Delta-Oriented Programming
  - A recent paradigm for the implementation of software product lines.

- Delta-Oriented Programming principle




- Motivation for using Delta-Oriented Programming
  - Its ability to introduce variability into an existing software product
  - Interesting for ensuring the scalability of software and systems.

# Problem

- Not many examples of concrete implementations the DOP
  - Lack of tools

# Problem

- Not many examples of concrete implementations the DOP
  - Lack of tools

- There two main tools


# DeltaJava

```java
public class CoreClass {
    public String sendMessage(String header, String protocol, String payload) {
        Map<String, String> values = new HashMap<>();
        StringBuilder container = new StringBuilder();
        container.append(header).append(protocol).append(payload);
        return String.valueOf(container);
    }
}
```

Core Class

Delta operation

```
delta Delta1 {
    modifies com.bl.connector.these.deltaj.example.CoreClass {
        adds private String  cibledProtocol="AMQP";
        adds protected String changeTrameProtocol(String header, String message) {
            return header+message+cibledProtocol;
        }
    }
}
```

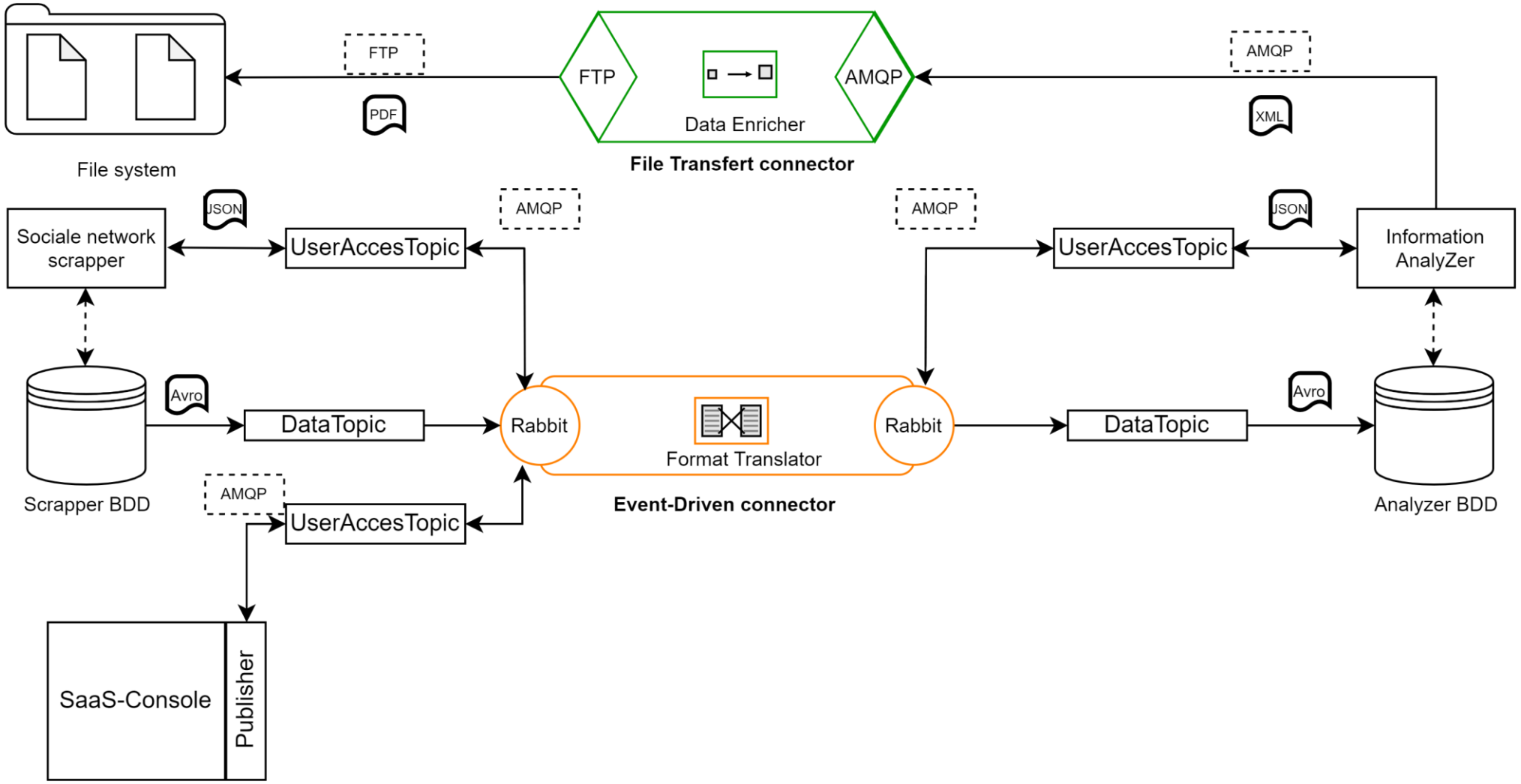
```
configuration delta "Delta1"
   dialect <http://www.emftext.org/java>
       modifies <../src/com/bl/connector/these/deltaj/example/CoreClass.java>
       {
           addMember(<class:com.bl.connector.these.deltaj.example.CoreClass>,
               "private String  cibledProtocol=\"AMQP\";");
           addMember(<class:com.bl.connector.these.deltaj.example.CoreClass>,
               "protected String changeTrameProtocol(String header, String message) {
           return header+message+cibledProtocol;
       }
   ");
   }
```

Decorator

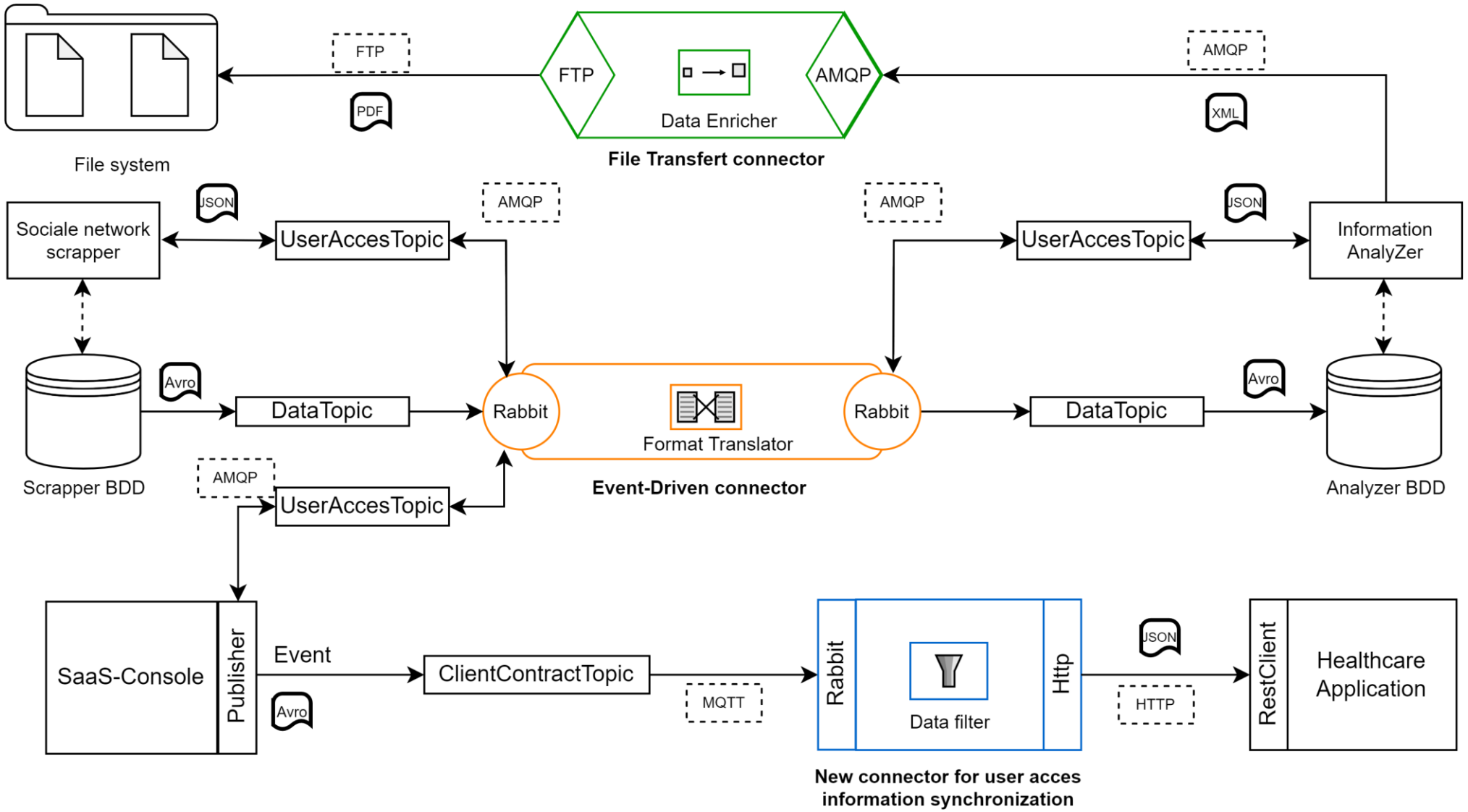
# Outline

- Context
- **Illustrative case study**
- Overview of the tool prototype
- Ongoing and future work

# Use case – Variability in connectors

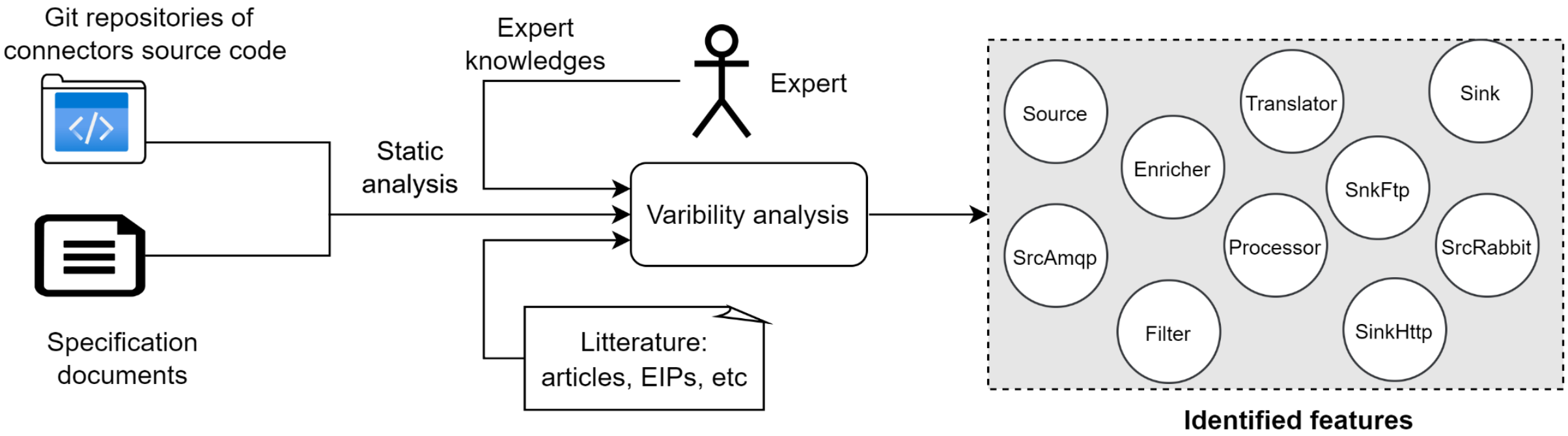


# Use case – Variability in connectors



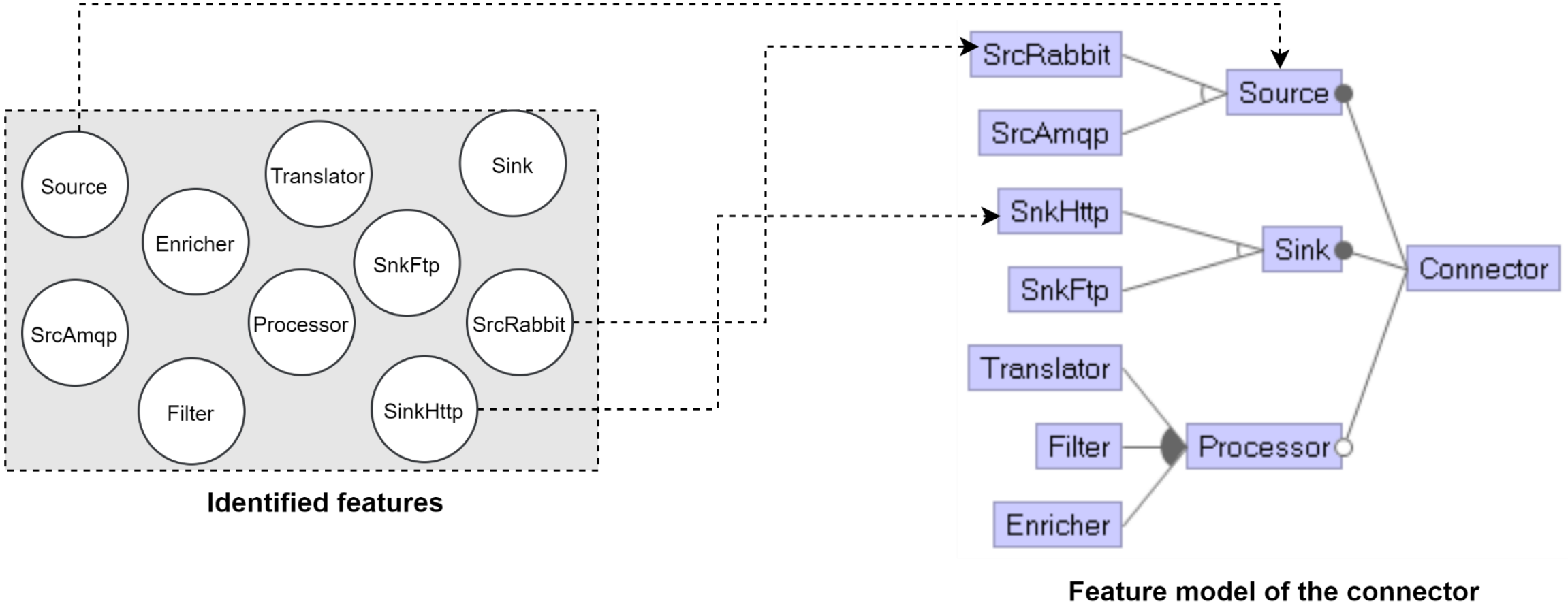
# Details of the process – Domain Engineering

- **Variability analysis**



# Details of the process – Domain Engineering

- **Variability modeling**

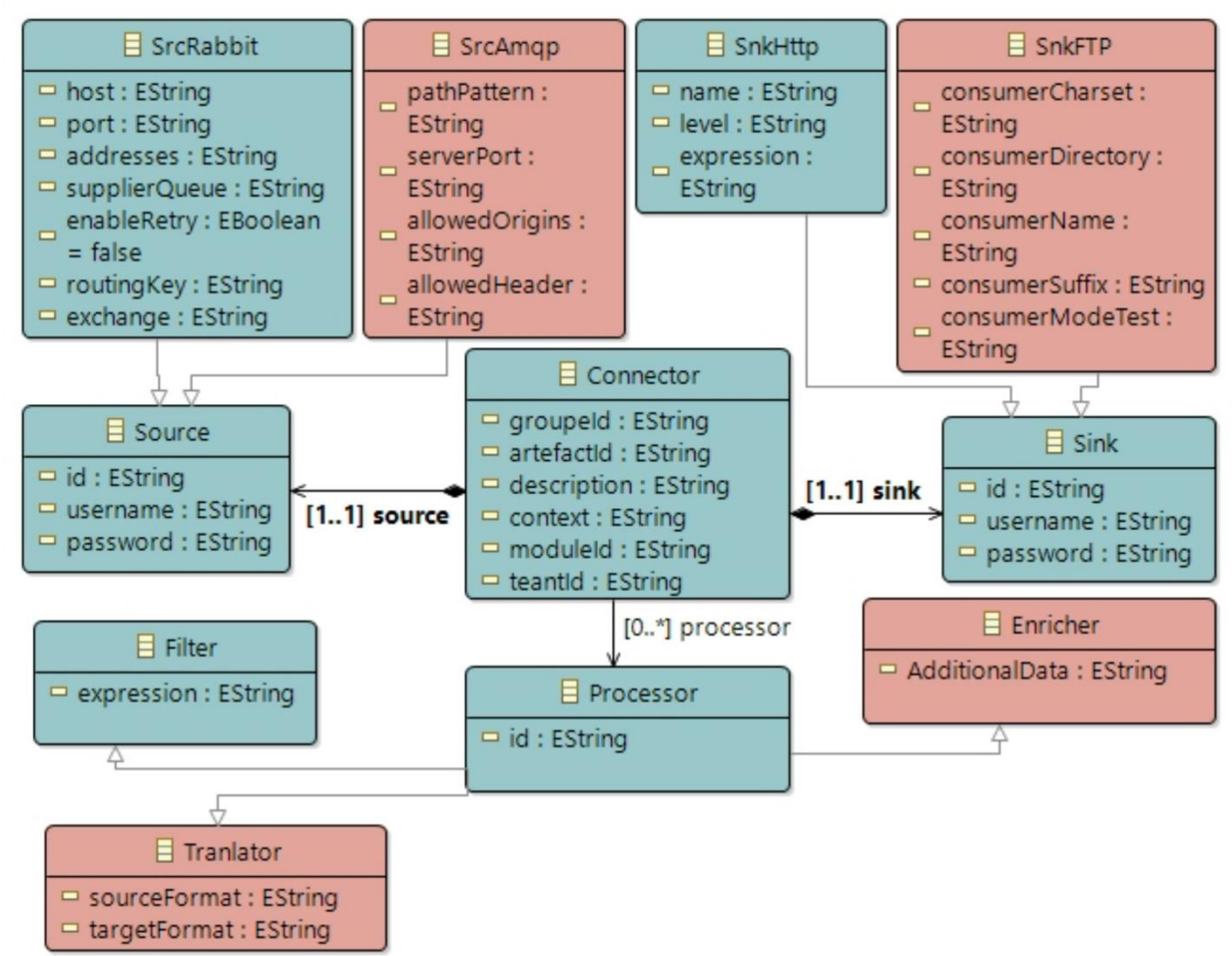



**Identified features**

**Feature model of the connector**

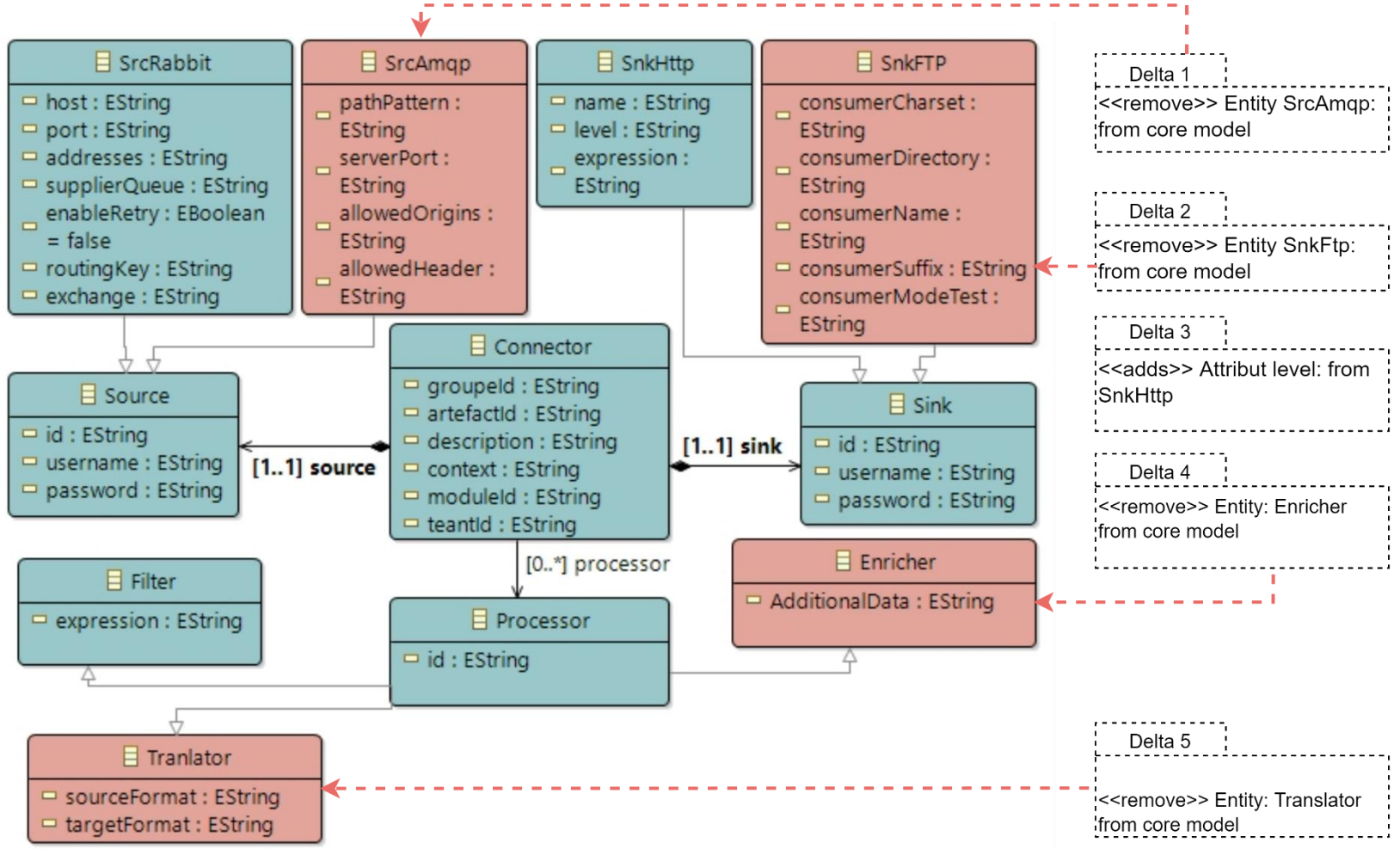
# Details of the process – Domain Engineering

- The software product line implementation using the Delta-Oriented Programming.

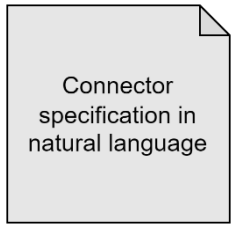


# Details of the process – Domain Engineering

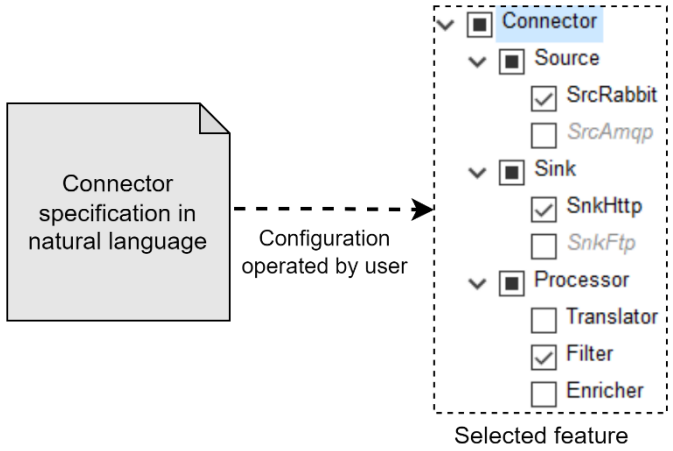
- The software product line implementation using the Delta-Oriented Programming.



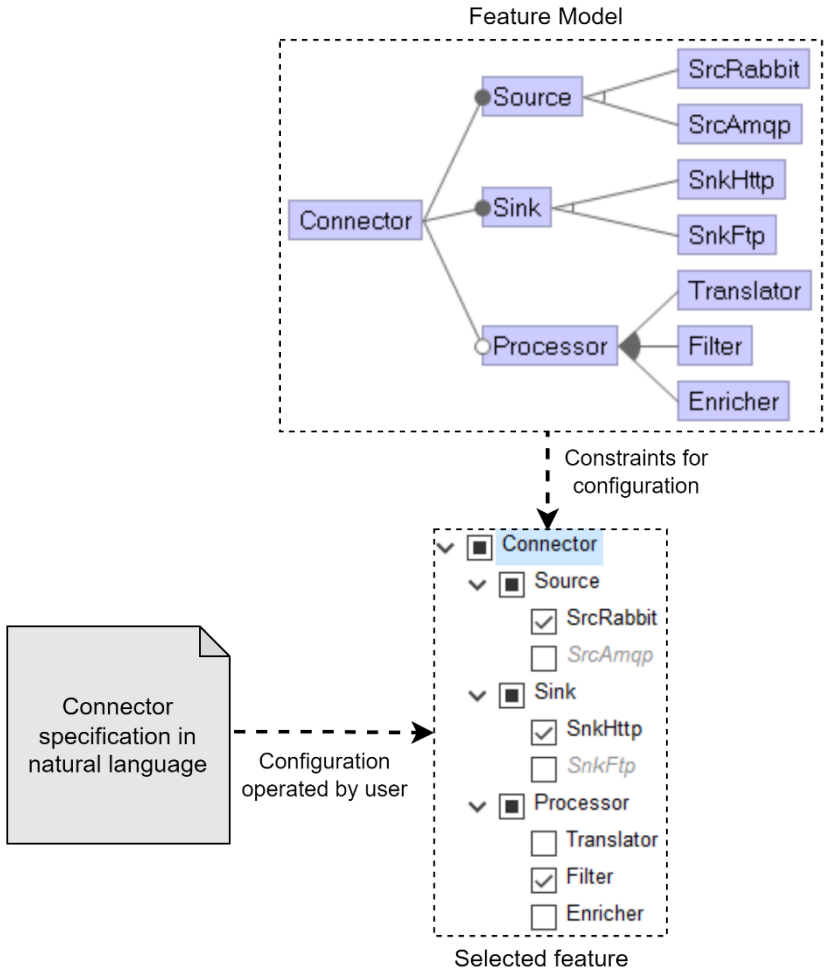
# Details of the process – Application Engineering



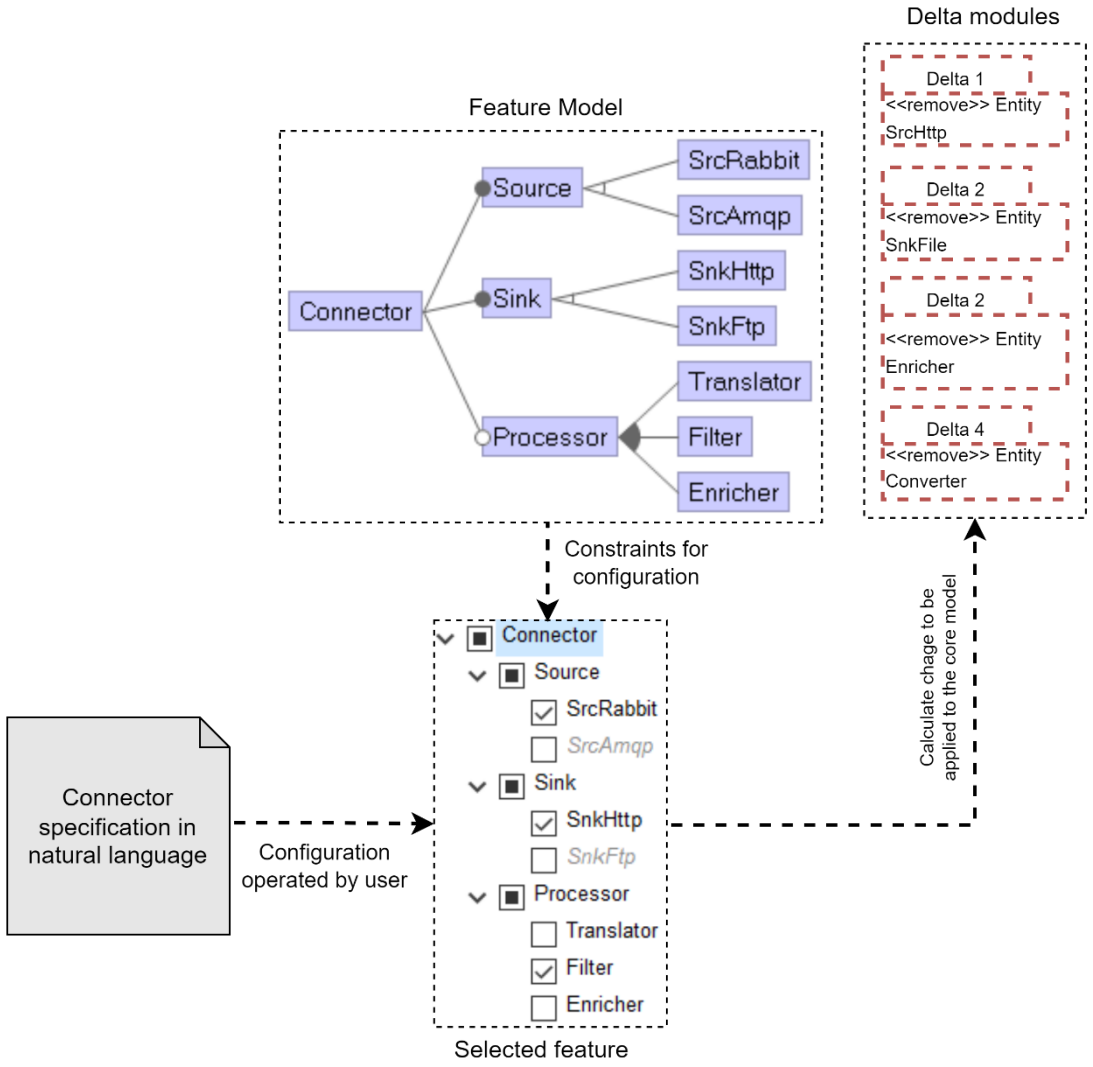
# Details of the process – Application Engineering



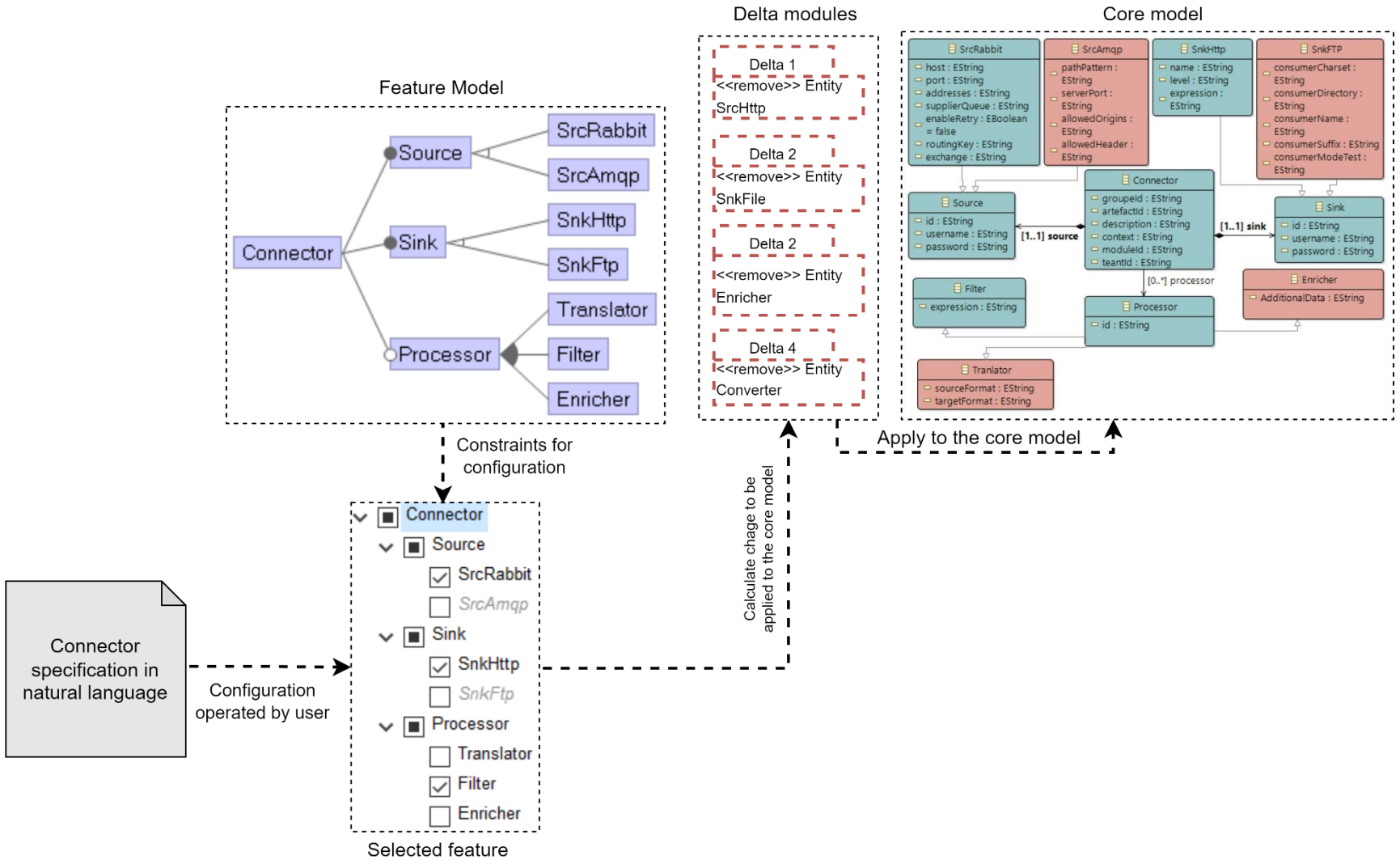
# Details of the process – Application Engineering



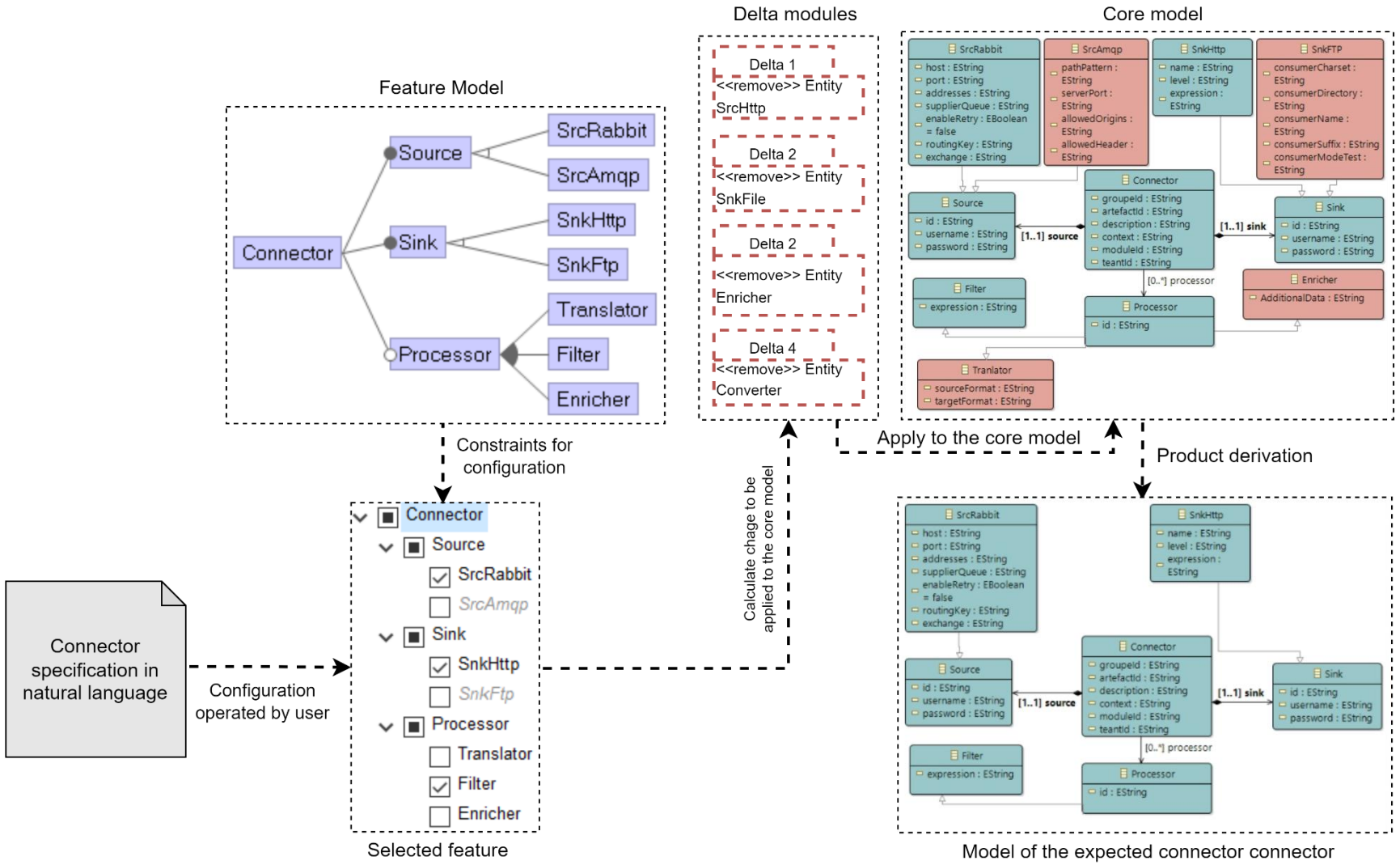
# Details of the process – Application Engineering



# Details of the process – Application Engineering

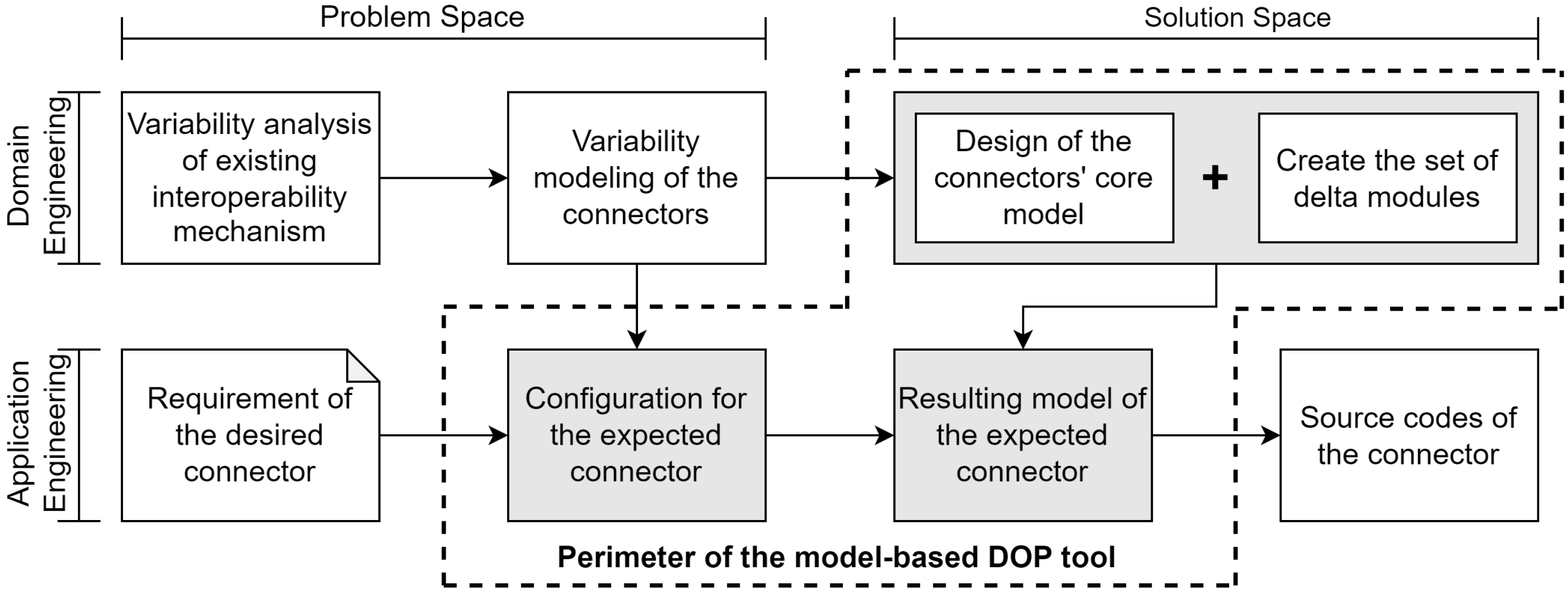


# Details of the process – Application Engineering



# Tool Perimeter

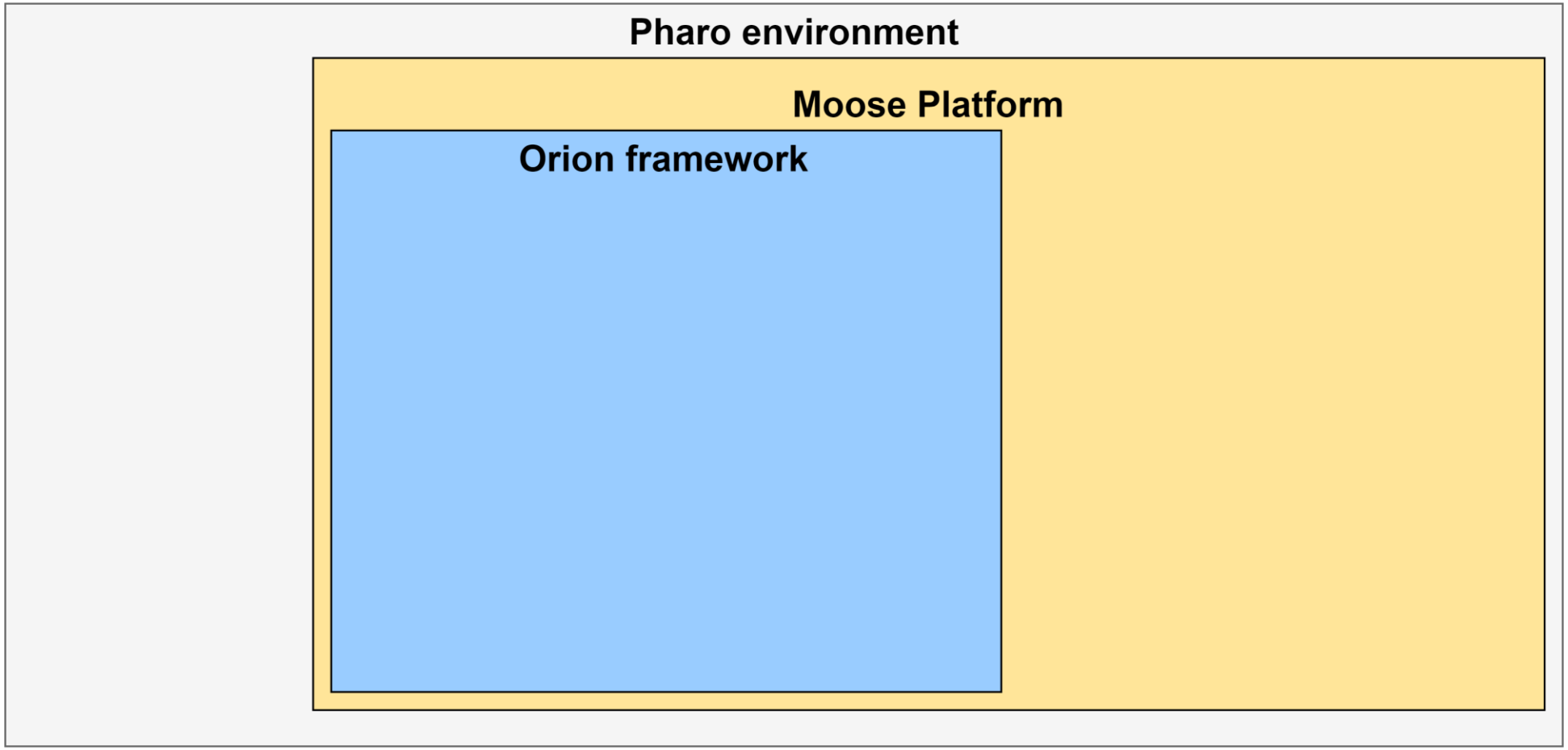
- Software Product Line steps covered by the process.



# Outline

- Context
- Illustrative case study
- **Overview of the tool prototype**
- Ongoing and future work

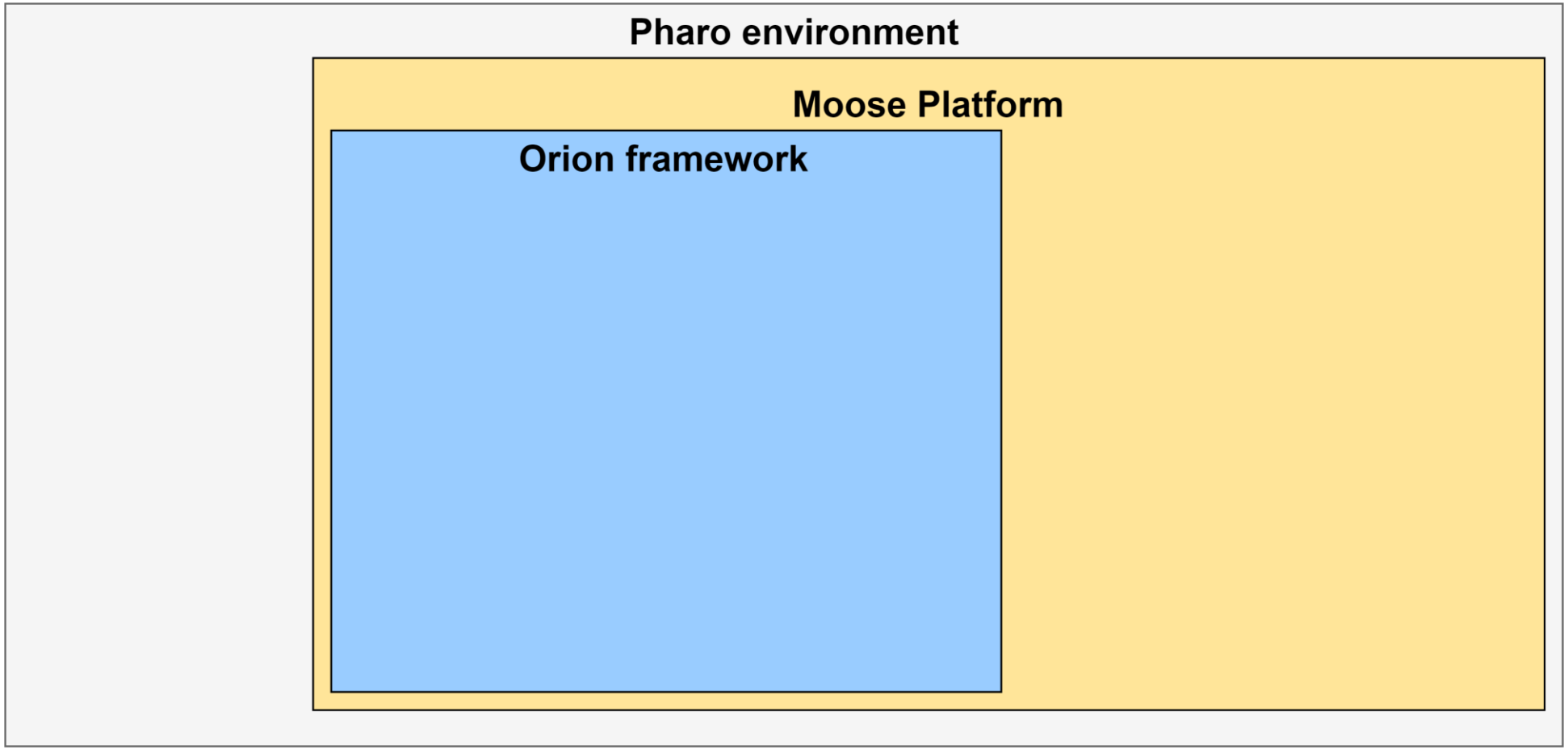
# The tool prototype



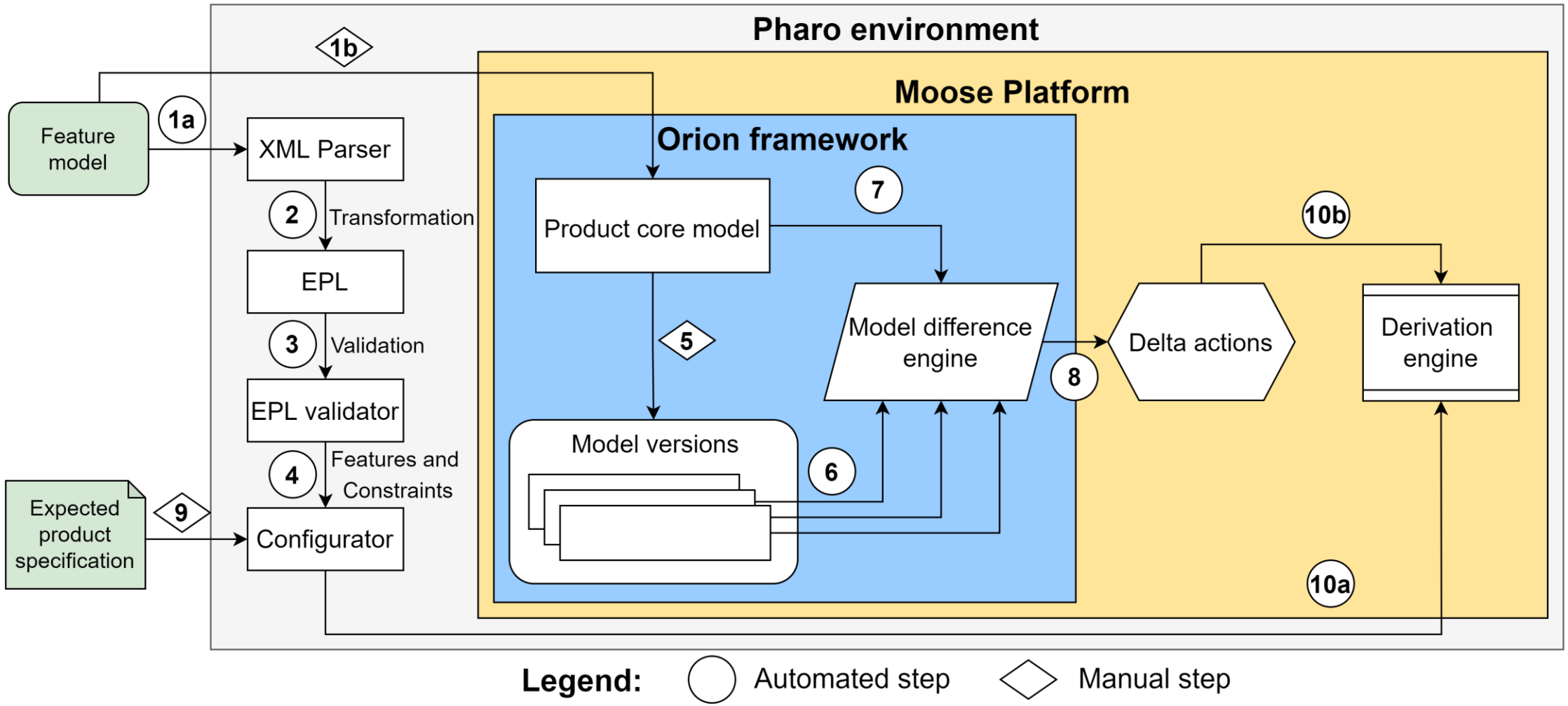
# The tool prototype

Feature model

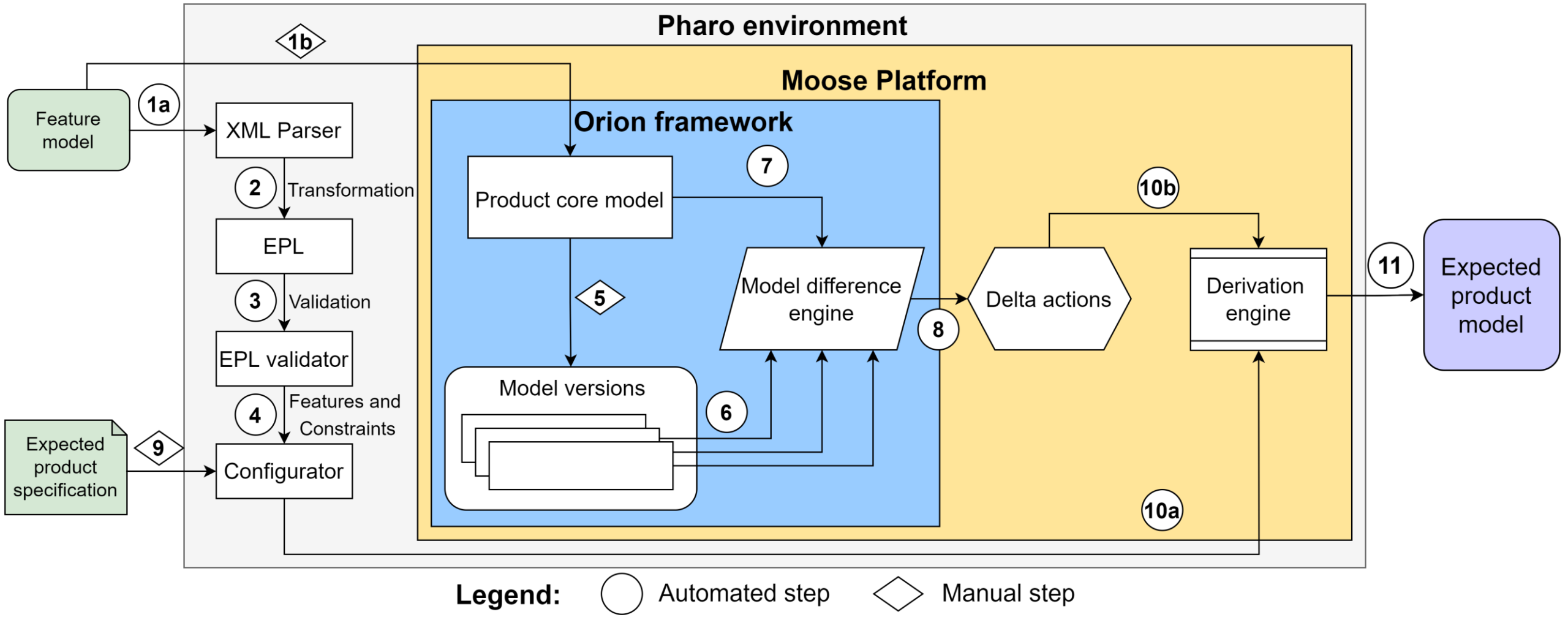
Expected product specification



# The tool prototype



# The tool prototype



# Outline

- Context
- Illustrative case study
- Overview of the tool prototype
- **Ongoing and future work**

# Progression

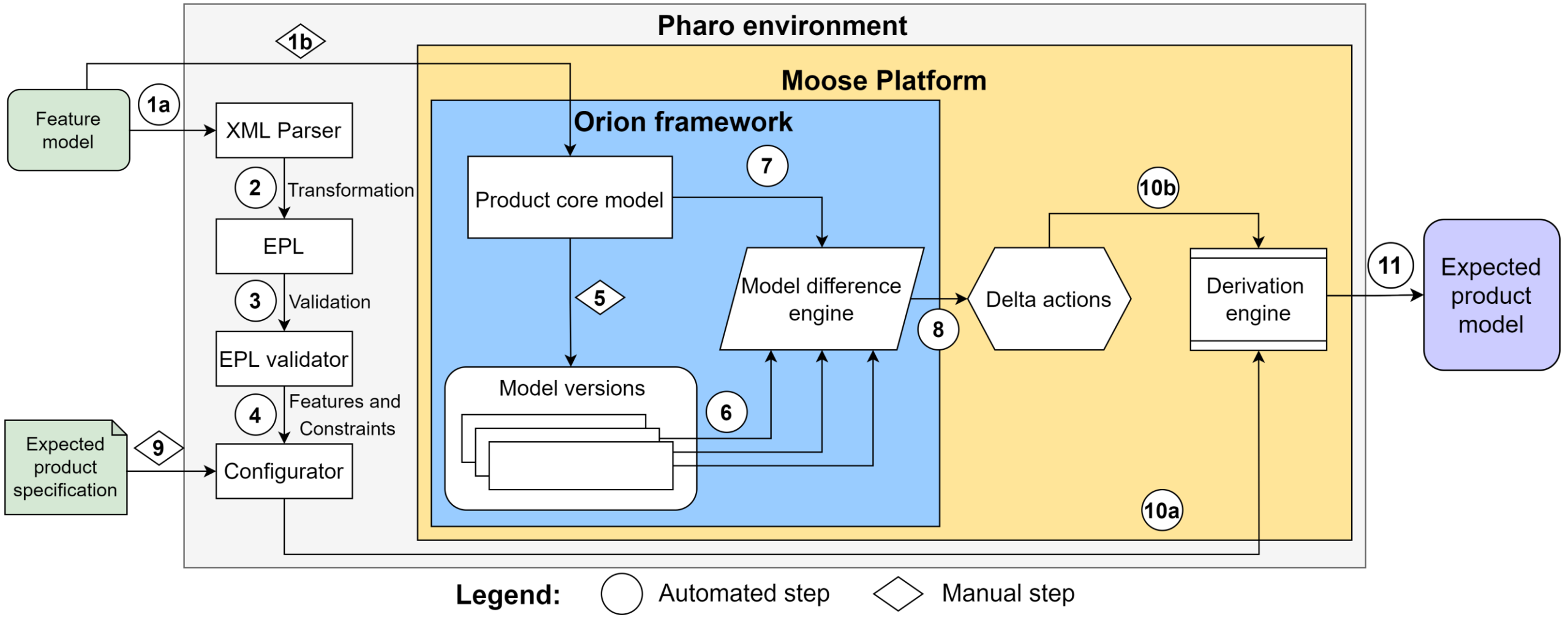
- Ongoing work
  - Parser
    → Metamodel of feature model
    → Population of the metamodel and generating the Expression Product Line (EPL)
  - Generation of the EPL

# Progression

- Ongoing work
  - Parser
    → Metamodel of feature model
    → Population of the metamodel and generating the Expression Product Line (EPL)
  - Generation of the EPL

- Future work
  - Validation of the Expression product Line
  - Configurator interface
  - The model difference calculation engine
    → Plan to exploit Orion and the FamixDiff Project
  - Derivation engine

# Thank you. Question?




**Boubou Thiam Niang**, Giacomo Kahn, Nawel Amokrane, Yacine Ouzrout, Mustapha Derras and Jannik Laval