From Monolithic to Microservice Architecture: The Case of Extensible and Domain Specific IDEs

Romain Belafia – Pierre Jeanjean – Olivier Barais – Benoit Combemale – Gurvan Le Guernic

DevOps@MODELS 2021

Pre-print: https://hal.inria.fr/hal-03342678
Integrated Development Environment (IDE): provides Language Services

- Auto-completion
- Refactoring
- Compilation
- ...

Language Server Protocol (LSP)

- Allowed the separation between language-agnostic client IDE and a language server
- Allowed the migration toward cloud architectures

Language server built as **Monolithic Application**
Opportunities for new software architecture

1. **Lack of modularity**: monolithic architectures are not able to manage modular structures

2. **Heterogeneity of language services**: language services have various needs in response time and resources [Modular and Distributed IDE, Coulon et al., SLE 2020]

   - **Auto-completion**
     - **Light**: little demanding in resource
     - **Accessed often**: needs a short response time
       - Deployed **locally**

   - **Compilation**
     - **Heavy**: very demanding in resources
     - **Occasional access**: response time less impacting
       - Deployed in a **distant server**

**Explored software architecture: Microservices**

- Divide an application in **light** and **independent** modules focused on **one functionality**
Microservices have significant benefits, e.g.:
- Better deployment flexibility
- Have independent development life-cycle

Benefits in a DevOps context, e.g.:
- Separation of concerns
- Better scalability
- Shorter deployment times
Microservicization of legacy monolithic applications is highly dependent of specific properties of the application of interest

- No “universal” guide to migrate a monolithic application toward microservices
- Generalization for applications sharing specific properties

**Cloud-based IDEs specificities**

1. **Heterogeneity of language services**: language services have various needs in resources and response time
2. **Heterogeneity of development**: services are developed by various stakeholders
3. **Manipulation of rich and complex data structures**: programs are manipulated as rich and complex data structures (e.g., AST) which needs to be exchanged between microservices
Experimental approach

➢ Study Cloud-based IDEs’ microservicization process to provide insights for future applications’ microservicization

➢ Empirical approach: based on the study of a case study and the difficulties encountered to draw lessons learned

**Contribution:** Set of lessons learned based on our experiment of a Cloud-based IDE microservicization
Case study presentation

Sirius Web

- Open-source framework
- Developed by Obeo
- Allows the conception of Graphical Modeling Workbenches

Goal: Study Sirius Web migration toward microservices
A Sirius Application relies on 2 main elements:

1. **Sirius Specification**
   - **EMF Project**: Provides the API to manipulate models
   - **Sirius Configuration**: Specifies the graphical representation of the manipulated models

2. **Sirius Runtime**
   - **Set of Sirius Components**: Maven modules specifying the functionalities, the API and the structure of a Sirius Web application
First microservice: AQL Interpreter

- **Acceleo Query Language**: DSL to perform queries over an EMF meta-model
- **AQL Interpreter**: evaluates AQL expressions

```
<conditionalStyles predicateExpression="aql:self.tension>0"> 
  <style color="yellow" /> 
</conditionalStyles>
```

*Use of AQL to specify a conditional style in an .odesign file*

AQL Interpreter decoupled as a microservice
Lesson learned 1: open-world challenge

**Modular application:** incrementally add features to an application

**Challenges for microservicization:**
- Make new modular features available to several microservices

**Modular functionalities**

**Lesson 1: Modularity** leads practitioners to adapt the organization and the granularity of microservices to make modular functionalities available to different microservices.

**Opportunities:** Best architecture for modular structures in cloud application
Lesson learned 2: serialization of rich and complex data structures

**Serialization**: conversion process of java objects to byte stream to transfer data among services

Complex data structures, such as AST, must be serialized and transferred among language services

➢ No implementation for automatic serializations

**Lesson 2: Serialization** of data to transfer can be an arduous process when rich and complex structures are manipulated. The serialization should be planned upstream in the development process, especially to audit existing solution or schedule the development of adapted tools.

**Opportunities**: Static analysis tools to extract the data to exchange
Lesson learned 3: manipulation of shared resources

Various functionalities can access to shared resources (e.g., EPackages) initialized once

Lesson 3: Shared resources raise challenges for the microservicization process. One must be aware of their impacts on the microservice granularity and the deployment organization.

Opportunities: Find the right balance between statefulness and statelessness
Lesson learned 4: rupture of pass-by-reference chains

**Pass-by-reference chain in a monolithic application**

\[
\begin{align*}
&f_a(x) \\
&x_0 \rightarrow x_1 \\
&f_b(x) \\
&x_0 \rightarrow x_1 \\
&f_c(x)
\end{align*}
\]

**Pass-by-reference chain in a microservice application**

\[
\begin{align*}
&f_a(x) \\
&x_0 \rightarrow x_0 \\
&f_b(x) \\
&x_0 \rightarrow x_0 \\
&f_c(x)
\end{align*}
\]

**Lesson 4: Pass-by-reference chains** should be considered when migrating an application toward microservices. The microservicization process can affect and even break the original behavior of the application.

**Opportunities:** Annotation framework to explicit the side effects when using pass-by-references
Conclusion

➢ Migration of a Cloud-based IDE toward microservices
➢ Case study: Sirius Web
➢ 4 lessons learned from AQL Interpreter decoupling:
  ✓ Open-world challenge
  ✓ Serialization of rich and complex data structures
  ✓ Manipulation of shared resources
  ✓ Rupture of pass-by-reference chains

Perspectives

➢ Study of other Sirius components / Cloud-based IDE microservicization
➢ Development of tools to assist migration toward microservices
  ✓ Static analysis tool to analyze and adapt the granularity of the application
  ✓ Framework to annotate the expected behavior of pass-by-references
  ✓ ...

From Monolithic to Microservice Architecture: The Case of Extensible and Domain Specific IDEs
14 DevOps@MODELS2021 - 11/10/2021