Zurich University of Applied Sciences

School of Engineering
InIT Institute of Applied Information Technology

Selective Java Code Transformation into AWS Lambda Functions

Serhii Dorodko and Josef Spillner
Zurich University of Applied Sciences,
School of Engineering, Service Prototyping Lab
(blog.zhaw.ch/splab), Switzerland
{dord,josef.spillner}@zhaw.ch

Serhii Dorodko 20.12.2018

Motivation:

FaaS is a completely new and promising paradigm which requires specific knowledge for developers.

- Provider tied development model
- Data exchange model
- Runtime restrictions
- Packaging and deployment model

Legacy code to be transformed into a new model with less resources consumption.

Podilizer:

The initial research idea was to provide fully-automated approach to transform existing java-code into Lambda functions.

- CLI tool
- Input: Java project
- Output: Appropriate set of Lambda functions deployed

The approach shows satisfying results only with small projects due to complex dependency management and java-specific features

Moving to selective transformation:

After the experience gained in Podilizer we decided to use annotation mechanism.

- Inspired by Spring framework which is a good example of annotation usability
- Gives more control in FaaSification process in a simple and understandable way
- Is a part of a language

Research Questions:

RQ 1 : Is it economically viable to run a Java ap-plication entirely over FaaS?

RQ 2 : Is it technically feasible to automate this pro-cess?And if so, which percentage of code coverage can be expected, which performance can be achieved, and which code is easier, hard or impossible to convert?

RQ 3 : Is there a friction-free integration with estab-lished Java development notations and processes?

Challenges:

Disassembling java code into functions caused non trivial task to solve. In most of the cases each method has a state that needs to be handled, providing functionality correctness.

Lambda programming model allows to consume and send objects, so we used it to:

- Exchange current state of an object while requesting function
- Return state together with the return result to update the state

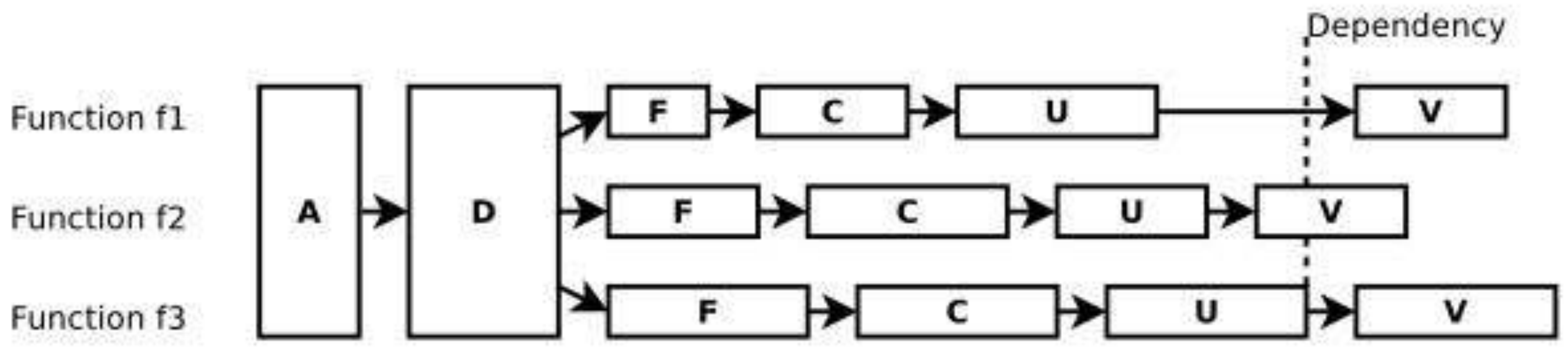
Class.handleRequest(input, output, context)

Zurich University of Applied Sciences

School of Engineering

InIT Institute of Applied Information Technology

FaaSification Pipeline:



A - static code parsing and analysis

D - decomposition into functional units

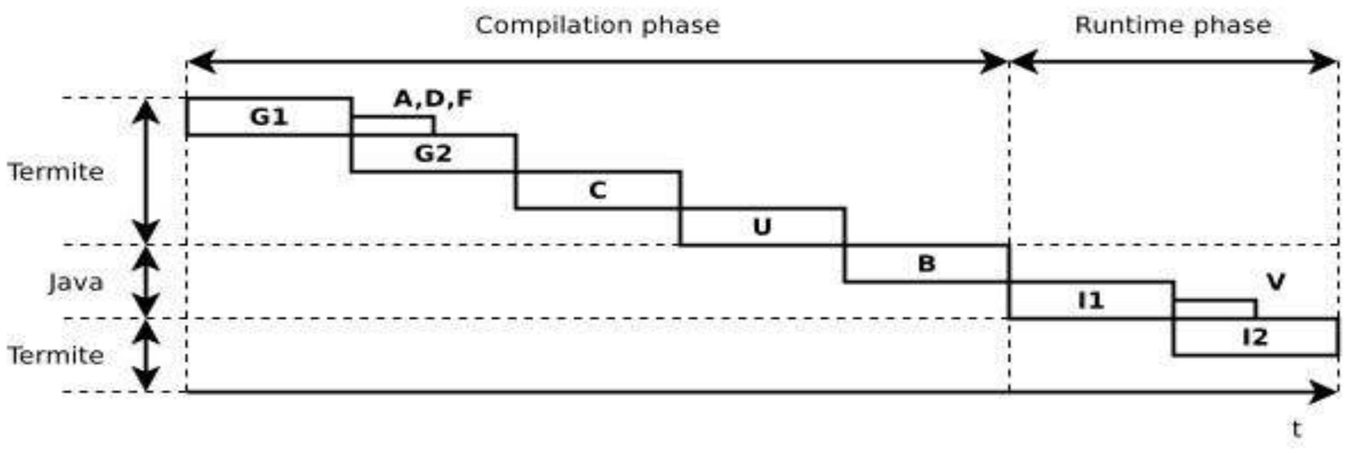
F -source-to-source translation of the functional units into FaaS units

C - compilation and dependency assembling of these units

U - upload, deployment and configuration

V - verification

Termite Design:



G1 - Generation of functions

G2 - additional sources

C - compilation

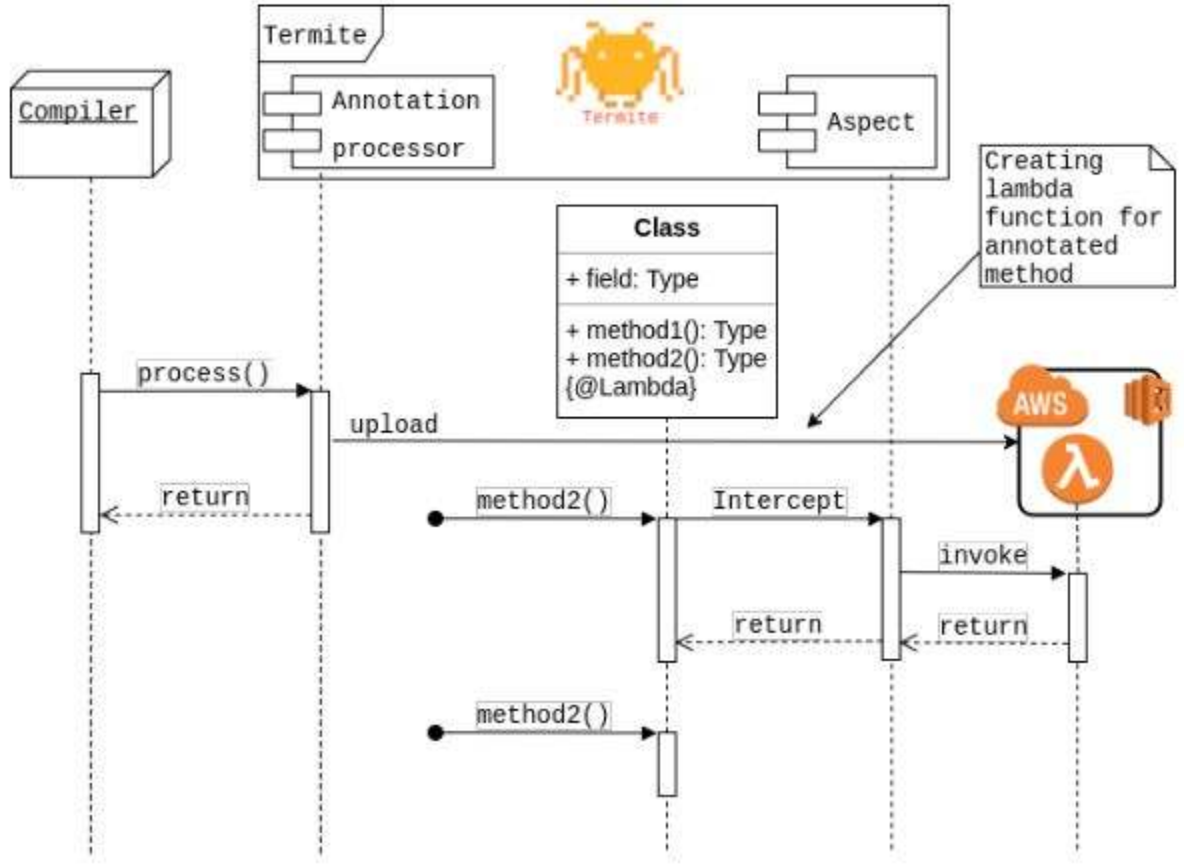
U - Upload

B -build

I1 - invocation of the annotated Java application

I2 - invocation of hosted function invocation

Implementation:

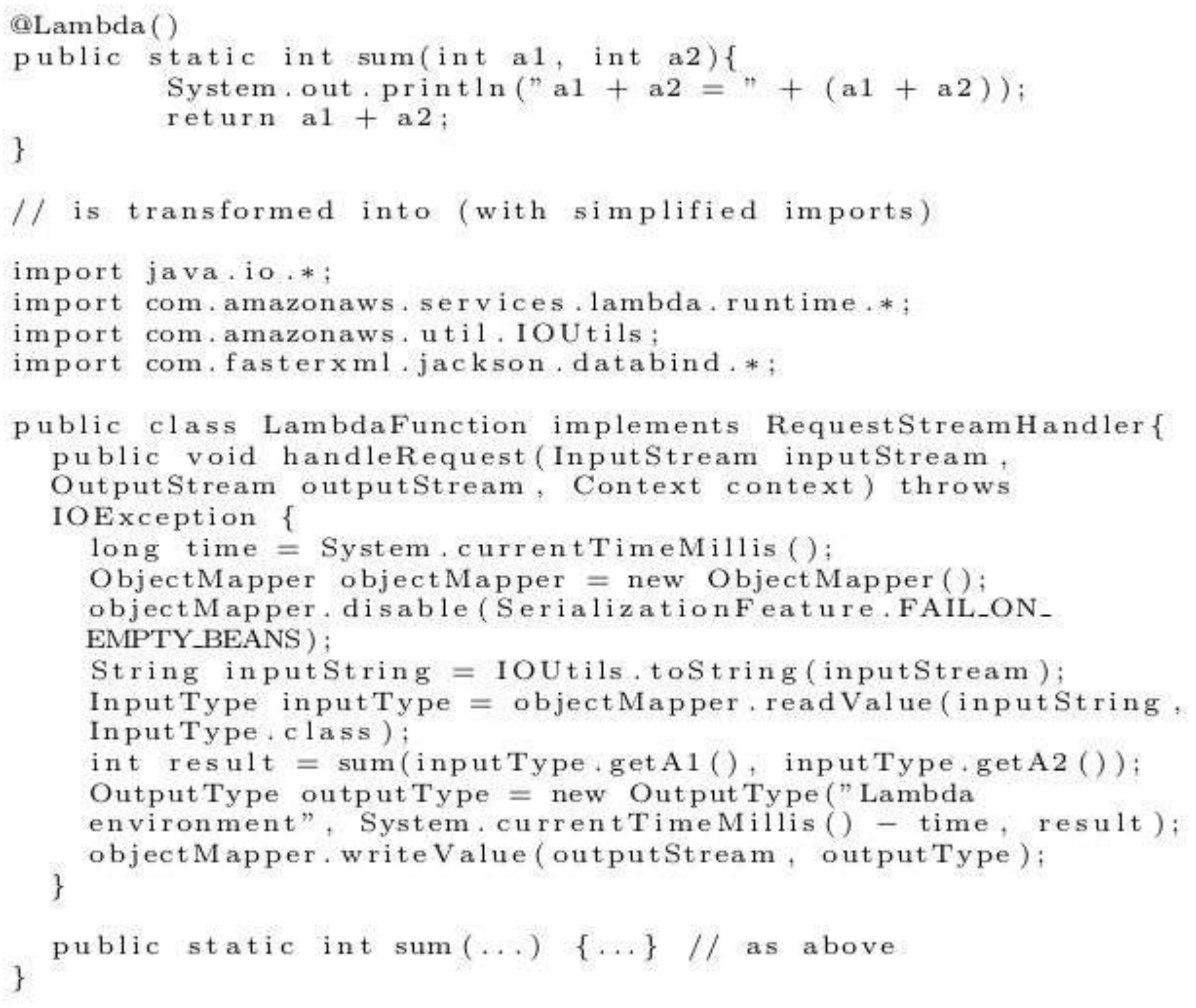


This schema shows the interaction of Termite with code during the compilation and runtime phase.

method1() - annotated

method2() - not annotated

Code transformation:



```
@Lambda()
public static int sum(int a1, int a2){
        System.out.println("a1 + a2 = " + (a1 + a2));
        return a1 + a2;
}

// is transformed into (with simplified imports)

import java.io.*;
import com.amazonaws.services.lambda.runtime.*;
import com.amazonaws.util.IOUtils;
import com.fasterxml.jackson.databind.*;

public class LambdaFunction implements RequestStreamHandler{
  public void handleRequest(InputStream inputStream,
  OutputStream outputStream, Context context) throws
  IOException {
    long time = System.currentTimeMillis();
    ObjectMapper objectMapper = new ObjectMapper();
    objectMapper.disable(SerializationFeature.FAIL_ON_
    EMPTY_BEANS);
    String inputString = IOUtils.toString(inputStream);
    InputType inputType = objectMapper.readValue(inputString,
    InputType.class);
    int result = sum(inputType.getA1(), inputType.getA2());
    OutputType outputType = new OutputType("Lambda
    environment", System.currentTimeMillis() - time, result);
    objectMapper.writeValue(outputStream, outputType);
  }

  public static int sum(...) {...} // as above
}
```

Zurich University of Applied Sciences

School of Engineering

InIT Institute of Applied Information Technology

Experiment setup:

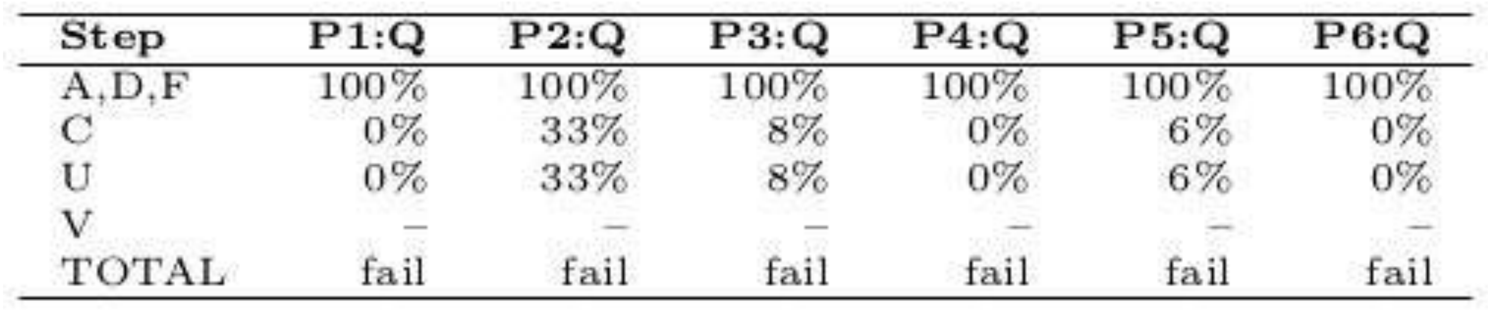
The reference input project set consists of six software applications which represent the large variety of Java software engineering, ranging from 28 to 771 significant lines of code (SLOC).

The software projects are:

- graphical window with buttons (P1)
- mathematical functions (P2)
- calculation of shipping containers and boxes (P3)
- public transport information (P4)
- image processing (P5)
- domain-specific language parsing and evaluation (P6)

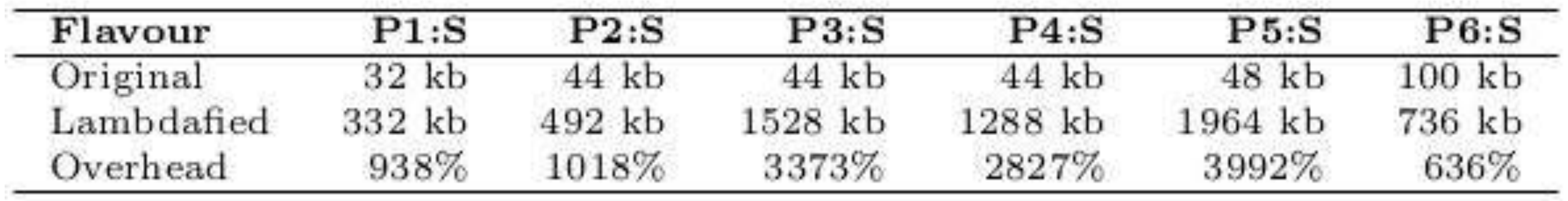
An artificial project consisting of 100 numbered Java hello world methods is used as additional comparison point

Results:



Step	P1:Q	P2:Q	P3:Q	P4:Q	P5:Q	P6:Q
A,D,F	100%	100%	100%	100%	100%	100%
C	0%	33%	8%	0%	6%	0%
U	0%	33%	8%	0%	6%	0%
V	–	–	–	–	–	–
TOTAL	fail	fail	fail	fail	fail	fail

Lambdafication pipeline characteristics (quality) for P1–P6 with Termite



Flavour	P1:S	P2:S	P3:S	P4:S	P5:S	P6:S
Original	32 kb	44 kb	44 kb	44 kb	48 kb	100 kb
Lambdafied	332 kb	492 kb	1528 kb	1288 kb	1964 kb	736 kb
Overhead	938%	1018%	3373%	2827%	3992%	636%

Application source code size comparison before/after using Termite

Conclusion:

Our findings in automated Java code to Lambda units transformation look promising for future cloud application engineering

Collected in the experiments data shows, that FaaSification process is not trivial and brings significant challenges for automated migration of a legacy code into the Functions. However, we believe that the tool designs and implementations are helpful in accelerating cloud deployments despite needing more fundamental research on the OOP to FaaS mapping and tool engineering and testing.