Constraint Verifier for Jolie using LEMMA

Harsh Chandel (19JE0347), BTech(CSE), IIT(ISM) Dhanbad

Under the supervision of

Dr Saurabh Srivastava, Assistant Professor, Department of Computer Science & Engineering, IIT(ISM) Dhanbad

Acknowledgement

I would also like to thank

- **Dr Saverio Giallorenzo**, Assistant Professor, Department of Computer Science & Engineering, University of Bologna
- **Dr Florian Rademacher**, University of Applied Sciences and Arts, Dortmund

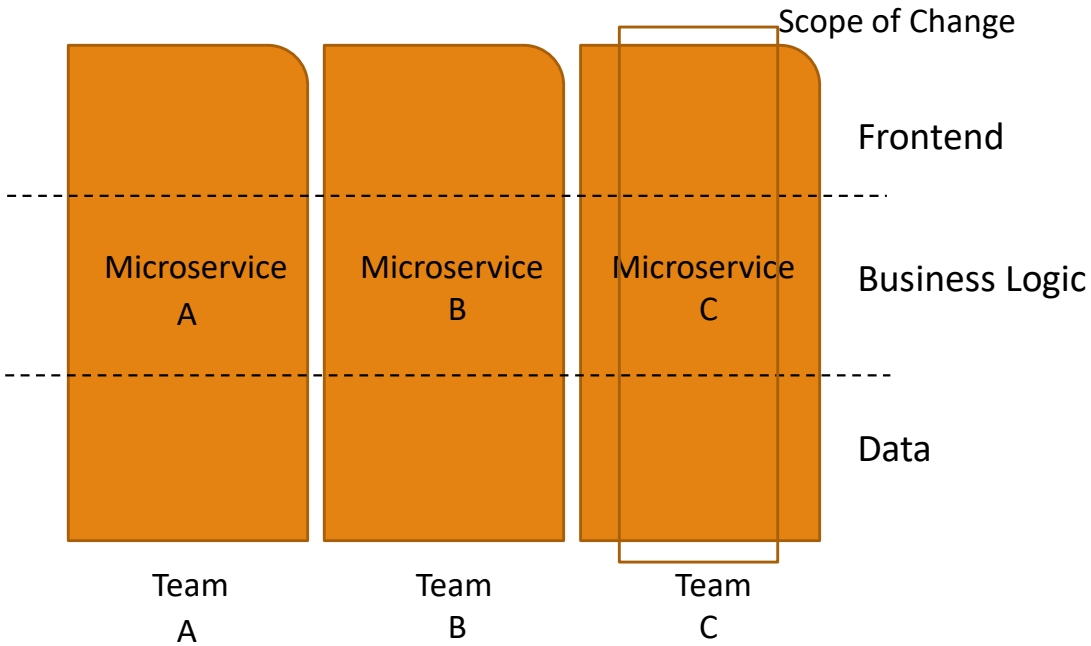
for their constant help and guidance throughout the project. Without their help, it wouldn't have been possible to do what we were able to do.

Contents

Introduction

- **Jolie** is a programming language
  - Meant specifically for building **Microservices**
  - Runs on JVM
- **LEMMA**
  - Is a Domain Description Framework
  - Can be used to model business use cases
- **Model Driven Development** and **Microservice Architecture** is the center of focus for this project

Introduction:- Microservices Architecture



Microservices Architecture for Software Applications
Making a change in a particular business domain only impacts the microservice corresponding to that domain

Aim

- This project
  - Aims at creating a Constraint Verifier for Jolie
  - Involves **implementing constraint verification methods** to ensure that **a Jolie Microservice doesn't violate domain constraints** specified in the **LEMMA specification**
- In our project, we are interested in application of **Model Driven Development** in **Microservices Architecture**

Solution Approaches

- The problem statement for the project was formulated
- Solution Approaches
  - **Using Symbolic Execution to detect constraints at runtime** is not feasible for our problem as it's an exponential time complexity approach. Prof. Saurabh and Prof. Saverio suggested using symbolic execution with some constraints but on further discussions, this approach was ruled out because symbolic execution wouldn't scale much and scalability is the prime concern for microservices architecture
  - **Using Static Analysis to detect constraints at compile time** is feasible

Progress

- Learnt Jolie and ran basic calculator microservices on my host machine and communication was achieved among those microservices using http, https, SODEP(Simple Operation Data Exchange Protocol), SODEPS(Simple Operation Data Exchange Protocol Secure), SOAP(Simple Object Access Protocol) *[done in last semester]*
- The link to the Jolie code and results for the calculator microservice is available at https://docs.jolie-lang.org/v1.10.x/tutorials/getting-started/ *[done in last semester]*
- We set up environment in the Ubuntu Virtual Machine(VM), eclipse instance was created on the Virtual Machine(VM) and was used to understand various **DDD patterns** using the **breakpoints** feature of **Eclipse IDE**

Progress

- After having studied and examined the DDD patterns using the Eclipse IDE, a need was felt for a **tool** that could **generate information** about the various **patterns** that are implemented in **a jolie source file**

- Such a tool has two fold purposes:-
  - Firstly, it could be used to **generate critical information about the patterns** implemented in a jolie source file and what sort of **constraints can be supported on them** which can eventually be used in **the constraint verifier**
  - Secondly, it could be used to **display information about various patterns** in a jolie source file (**in a tabular form in an automatically generated webpage**) **without having to read source code line by line**(quite useful as commercial source files may have tens (if not hundreds) of thousands of lines of codes)

- So we started working on this tool (let us call it **Jolie_Doc_Proto**)

Tasks Completed

- Following two primary tasks were completed :-
  - Understood and learnt about **DDD patterns** and made some examples of Jolie code that respects the constraints of a set of DDD patterns
  - Understood and learnt about:-
    - **Jolie interpreter**, i.e. what objects the programmer can use to analyze Jolie code (https://github.com/jolie/jolie) *[Useful for development of Jolie_Doc_Proto]*
    - **parser lib** https://github.com/jolie/jolie/tree/master/libjolie/src/main/java/jolie/lang) *[Useful for development of Jolie_Doc_Proto]*

Jolie_Doc_Proto:-Tech Stack

- Tech stack used :-
  A. Java and **IntelliJ IDE** :- As the jolie interpreters and all other required language software are implemented in **JAVA,** I had to go through the java code for abovementioned software and add the java code for **Jolie_Doc_Proto** wherever needed. Further **lambda expressions** of **Java** and **OOP** (Object Oriented Programming) were used
  B. Gradle :- the build tool used for the Jolie_Doc_Proto tool project
  C. Jolie :- was used to write the jolie source files
  D. HTML:- was used to generate the webpage

Jolie_Doc_Proto:- directory structure of source(./src) of this tool

```
D:\jolie_project\src>tree /F
Folder PATH listing for volume DATA
Volume serial number is E49D-7C26
D:.
│   .DS_Store
│
├───app
│   │   .DS_Store
│   │
│   └───java
│       │   .DS_Store
│       │
│       └───HarshsExecutable
│           │   .DS_Store
│           │   HashsExecutable.java
│           │
│           └───HarshsVisitor
│               │   HarshsVisitor.java
│
└───test
    │   .DS_Store
    │
    ├───java
    │   │   .DS_Store
    │   │
    │   └───example
    │       │   Test.java
    │
    └───jolie
        │   MyInterface.ol
```

- The folder **src/app/java** contains the **HashsExecutable.java** and **HarshVisitor.java** that contains the code for parsing the jolie file and generating the **index.html** that is the output of this tool in the project root directory
- **Test.java** (in **src/test/java**) contains the driver code for this tool and also specifies the location of jolie file to parse
- As of now, the tool parses the file names **MyInterface.ol** in **src/jolie/test**

Jolie_Doc_Proto:- Patterns Supported

Table 1. Domain-driven-design patterns for domain-driven microservice design, in UML notation.[5]

Pattern name	Example	Description
Entity	«Entity» Customer; +ex ID	Instances of domain concepts modeled as Entities are distinguishable from others by a domain-specific identity.
Value Object	«ValueObject» Person; +forename; +surname	Instances of domain concepts modeled as Value Objects are typically immutable and lack a domain-specific identity. They might act as value containers when exchanging information between Bounded Contexts (see below).
Aggregate	«AggregateRoot, Entity» Car; 1; «ValueObject» {aggregateRoot = Car} Tire	An Aggregate is a cluster of associated Entity and Value Object instances. It is treated as a whole that can be accessed only by referencing its root Entity instance. Next to Bounded Contexts, Aggregates might also denote a primary driver for domain decomposition.[6]
Repository	«Repository» Cars; +readByMarque(marque); «Entity» Car; marque	A Repository permits access to persistent domain concept instances via operations that perform instance selection based on given search criteria.
Service	«Service» FundsTransferDomainService; transfer(accountFrom, accountTo, amount); «Entity» Account; accountNo; +credit(amount); +debit(amount)	Services provide capabilities that are not the responsibility of Entities or Value Objects—e.g., control of business processes or domain concept instance transformations. A special type of domain-driven-design service is domain services. They interact with domain concept instances to realize business capabilities. For example, a funds transfer domain service might be responsible for coordinating credits and debits between banking accounts.[1] Services are parts of Bounded Contexts and are not to be confused with microservices that are derived from Bounded Contexts.
Specification	«Entity» Building; «Spec» BuildingInspection; isTechnically Approved(Building) : Boolean	Specifications may be employed to determine if a domain concept instance fulfills a certain specification. Specification classes contain a set of Boolean operations to perform specification checks.
Bounded Context	«BoundedContext» Customer	Bounded Contexts define scopes for enclosed domain concepts—i.e., boundaries for concept validity and applicability. A functional microservice[3] should be aligned to a Bounded Context because, like a microservice, a context bundles and isolates coherent domain concepts and thus • defines the scope of an isolated business capability, • needs to explicitly define access to domain concept instances via exchange relationships to other contexts, and • has exactly one responsible team assigned.[2]

Fig:- **Patterns supported in Jolie_Doc_Proto**

Jolie_Doc_Proto:-Illustration

```
///@valueObject
type ParkingSpaceBooking {
    bookingID: long
    driver: Driver
    timeSlot: TimeSlot
    priceInEuro: double
}
type priceInDollars_type {
    self?: ParkingSpaceBooking
}

///@valueObject
type Location {
    latitude: double
    longitude: double
}

type TimeSlots {
    s*: TimeSlot
}

type ChargerSpeed {
    literal: string(enum(["FAST", "NORMAL"]))
}

///@entity
type Driver {
    ///@identifier
    id: string
}

///@valueObject
type TimeSlot {
    start: string
    end: string
}
interface ParkingSpaceBooking_interface {
    RequestResponse:
        priceInDollars(priceInDollars_type)(double)
}
//<@endCtx
```

MyInterface.ol

index.html

File | D:/jolie_project/index.html

Gmail YouTube Maps

Class Name	Pattern Type	Description
ParkingSpaceBooking_interface	@endCtx	Bounded Context
Driver	@entity	Instances of domain concepts modeled as Entities are distinguishable from others by a domain-specific identity
ParkingSpaceBooking	@valueObject	Instances of domain concepts modeled as Value Objects are typically immutable and lack a domain-specific identity. They might act as value containers when exchanging information between Bounded Contexts.
id	@identifier	Identifier of Entity
TimeSlot	@valueObject	Instances of domain concepts modeled as Value Objects are typically immutable and lack a domain-specific identity. They might act as value containers when exchanging information between Bounded Contexts.
Location	@valueObject	Instances of domain concepts modeled as Value Objects are typically immutable and lack a domain-specific identity. They might act as value containers when exchanging information between Bounded Contexts.

Jolie_Doc_Proto:- Requirements to work

- It requires the programmer to **add a comment about the pattern type just before the he starts the definition of the pattern** in **MyInterface.ol** to function properly. Thus it requires adherence to this **coding standard/ programming habit** while writing the jolie source code. For example

```
///@valueObject
type ParkingSpaceBooking {
    bookingID: long
    driver: Driver
    timeSlot: TimeSlot
    priceInEuro: double
}
```

Future Works:-

- Refinements like adding some context information For example, the HTML can show what file the documentation regards and what type of element concerns a comment. Looking at the example, additional information can include indicating that `Driver` is a type and that `id` is a component within another type (and possibly show its name)
- Integrating the Jolie_Doc_Proto with the jolie toolchains
- Using the output generated by Jolie_Doc_Proto to develop the constraint verifier

References

1. Building Microservices by Sam Newman
2. Graphical and Textual Model-Driven Microservice Development
3. Aspect-oriented Modelling of Technology Heterogeneity in Microservice Architecture
4. Deriving Microservice Code from Underspecified Domain Models Using DevOps-Enabled Modelling Languages and Model Transformations
5. Challenges of domain-driven microservice design A model-driven perspective Rademacher, Sorgalla, Sachweh - 2018

THANK YOU