Reactive Web Services and Real–Time Applications with Eclipse Vert.x

Mattia Vandi
mattia.vandi@studio.unibo.it

C.d.L. Magistrale in Ingegneria e Scienze Informatiche
Alma Mater Studiorum – University of Bologna, Cesena

Academic Year 2017/2018

Outline

1. Reactive Manifesto

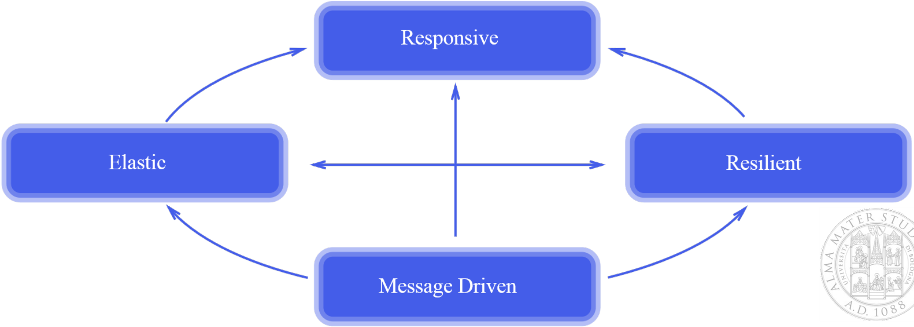
2. Vert.x

3. Demo

Reactive Manifesto [1]

Reactive Systems

"Systems built as Reactive Systems [...] are significantly more tolerant of failure and when failure does occur they meet it with elegance rather than disaster. Reactive Systems are highly responsive, giving users effective interactive feedback."



Reactive Systems

Goal

Provide **responsiveness** in the form of **elasticity** and **resiliency** exploiting **Message Driven** Architectures as mean.

Reactive Traits

- **Responsive**: focuses on providing rapid and consistent response times encouraging further interaction from the user
- **Resilient**: stays responsive in the face of *failure* by self-recovering from them
- **Elastic**: stays responsive under varying workload increasing or decreasing the amount of allocated resources
- **Message Driven**: relies on asynchronous message-passing allowing recipients to only consume resources while active

Outline

1 Reactive Manifesto

2 Vert.x

3 Demo

What is Vert.x?

Vert.x's Motto

"Eclipse **Vert.x** *[2] is a tool-kit for building* **reactive** *applications on the* **JVM***."*

Terms

- tool-kit: Vert.x is not a restrictive framework it just gives you a set of tools and lets you create your application in the way you want
- **reactive**: Vert.x is *event-driven* and *non blocking*, it can handle a lot of concurrency using a small number of kernel threads [3]
- **JVM**: Vert.x is *polyglot* and lets you choose the languages you want based on the task at hand and the skill-set of your team
  - You can use Vert.x with multiple langaugues including Java, JavaScript, Groovy, Ruby, Ceylon, Scala and Kotlin
  - Generated APIs are idiomatic for every language that Vert.x supports

“Simple but not simplistic”

Features

- Aynchronous by nature
- Slim and lightweight core (about 650 kB)
- Super simple concurrency model
- Based on Multi-Reactor Pattern

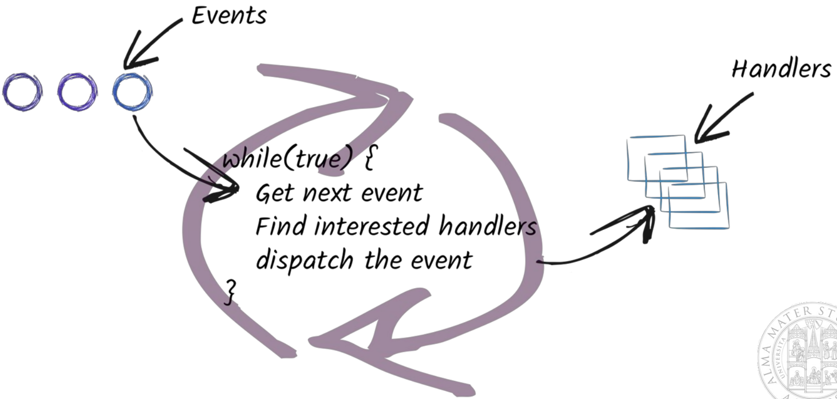
Terminology

- **Vert.x instance**: control centre of the Vert.x ecosystem, it is how you do pretty much everything
- **Verticle**: optional, non-strict, *actor-like* deployment and concurrency model
- **Distributed Event Bus**: allows different parts of your application to communicate with each other
- **Handler**: procedure that will receive events when they are available

Reactor Pattern I

Event loop

An *event loop* is attached to a single kernel thread, it enqueues *events* to a FIFO queue and delivers them to the intrested *handlers* as they arrive.



Reactor Pattern II

Benefits

- Optimal usage of CPU
- Can handle more requests with same hardware
- Can handle large number of clients at the same time

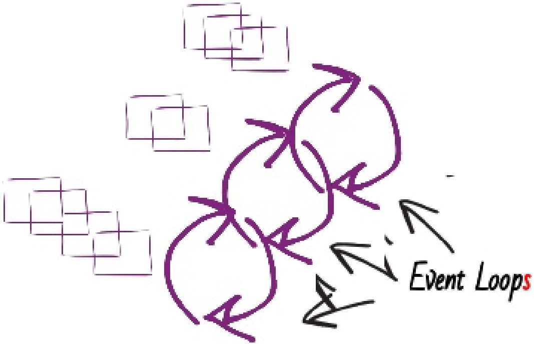
Drawbacks

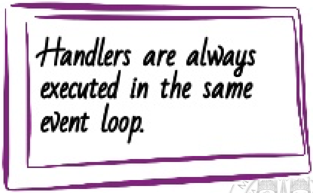
- Difficult to understand and structure code
- Complex debugging

Multi-Reactor Pattern

Definition

Maintains several event loops allowing a single process to scale across the CPU cores of the machine it is running on.




Handlers are always executed in the same event loop.

“In the beginning there was Vert.x”

- Usually one per application
- Entry point for Vert.x APIs
- Container for Verticles

Creating a Vert.x instance

```
Vertx vertx = Vertx.vertx();
```

Creating a clustered Vert.x instance

```
Vertx.clusteredVertx(new VertxOptions(), ar -> {
    if (ar.succeeded()) {
        Vertx vertx = ar.result();
        ...
    } else {
        System.err.println(ar.cause().getMessage());
    }
});
```

Verticles

- Chunks of code that get deployed and run by Vert.x
- Can be written in any supported language
- Each `Verticle` can have its own `ClassLoader`
- Each `Verticle` communicates with other Verticles via Event Bus

Writing Verticles

```java
public class MyVerticle extends AbstractVerticle {
    public void start() {
    vertx.createHttpServer()
        .requestHandler(request -> request
              .response()
              .end("Hello from Vert.x"))
        .listen(8080);
    }
}
```

The Event Bus

- **Nervous system** of Vert.x
- One for every Vert.x instance
- Supports various async message-passing patterns
  - Publish/Subscribe
  - Point to Point
  - Request-Response
- Best-effort delivery
- Uses JSON as data-interchange format
  - You can send arbitrary objects by defining a custom codec

Getting the Event Bus

```
EventBus eventBus = vertx.eventBus();
```

“Don’t call us, we’ll call you”

In Vert.x events are handled providing handlers to the Vert.x APIs.
Some examples of events are:

- Some data has been read from disk
- An exception has occurred
- An HTTP server has received a request

A generic event handler

```
@FunctionalInterface
public interface Handler<E> {
    /**
     * Something has happened, so handle it.
     *
     * @param event the event to handle
     */
    void handle(E event);
}
```

Ecosystem

Modules system

- Core
- Web (Web, Web Client, Web API Contract)
- Data Access (MongoDB client, JDBC client, etc.)
- Reactive (Vert.x Rx, Reactive streams, etc.)
- Microservices (Service Discovery, Circuit Breaker, Config)
- IoT (MQTT)
- Authentication and Authorisation (JWT auth, OAuth 2, etc.)
- Event Bus Bridge (TCP Eventbus Bridge, Camel Bridge)
- Devops (Metrics, Docker, etc.)
- Clustering (Hazelcast, Apache Zookeeper, etc.)
- And much more...

Ecosystem

Modules system

- **Core**
- Web (**Web**, Web Client, Web API Contract)
- Data Access (**MongoDB client**, JDBC client, etc.)
- Reactive (**Vert.x Rx**, Reactive streams, etc.)
- Microservices (Service Discovery, Circuit Breaker, Config)
- IoT (MQTT)
- Authentication and Authorisation (JWT auth, OAuth 2, etc.)
- Event Bus Bridge (TCP Eventbus Bridge, Camel Bridge)
- Devops (Metrics, Docker, etc.)
- Clustering (**Hazelcast**, Apache Zookeeper, etc.)
- And much more...

Outline

1 Reactive Manifesto

2 Vert.x

3 Demo

Demo

"Talk is cheap. Show me the code."
— Linus Torvalds

Links

- Sources: `https://bitbucket.org/mvandi/asw1718-seminar`
- Real—Time Chat: `https://calm-cliffs-80988.herokuapp.com`
- To—Do List: `https://afternoon-inlet-70848.herokuapp.com`

References

Jonas Bonér, Dave Farley, Roland Kuhn, and Martin Thompson.
The reactive manifesto.
https://www.reactivemanifesto.org, 2014.

Tim Fox.
Eclipse vert.x.
https://vertx.io, 2011.

TechEmpower.
Web framework benchmarks.
https://www.techempower.com/benchmarks/#section=data-r8&hw=i7&test=plaintext, 2013.

Reactive Web Services and Real–Time Applications with Eclipse Vert.x

Mattia Vandi
mattia.vandi@studio.unibo.it

C.d.L. Magistrale in Ingegneria e Scienze Informatiche
Alma Mater Studiorum – University of Bologna, Cesena

Academic Year 2017/2018