# Reactive Jersey Client

Michal Gajdos
michal.gajdos@oracle.com

January, 2015

# Safe Harbor Statement

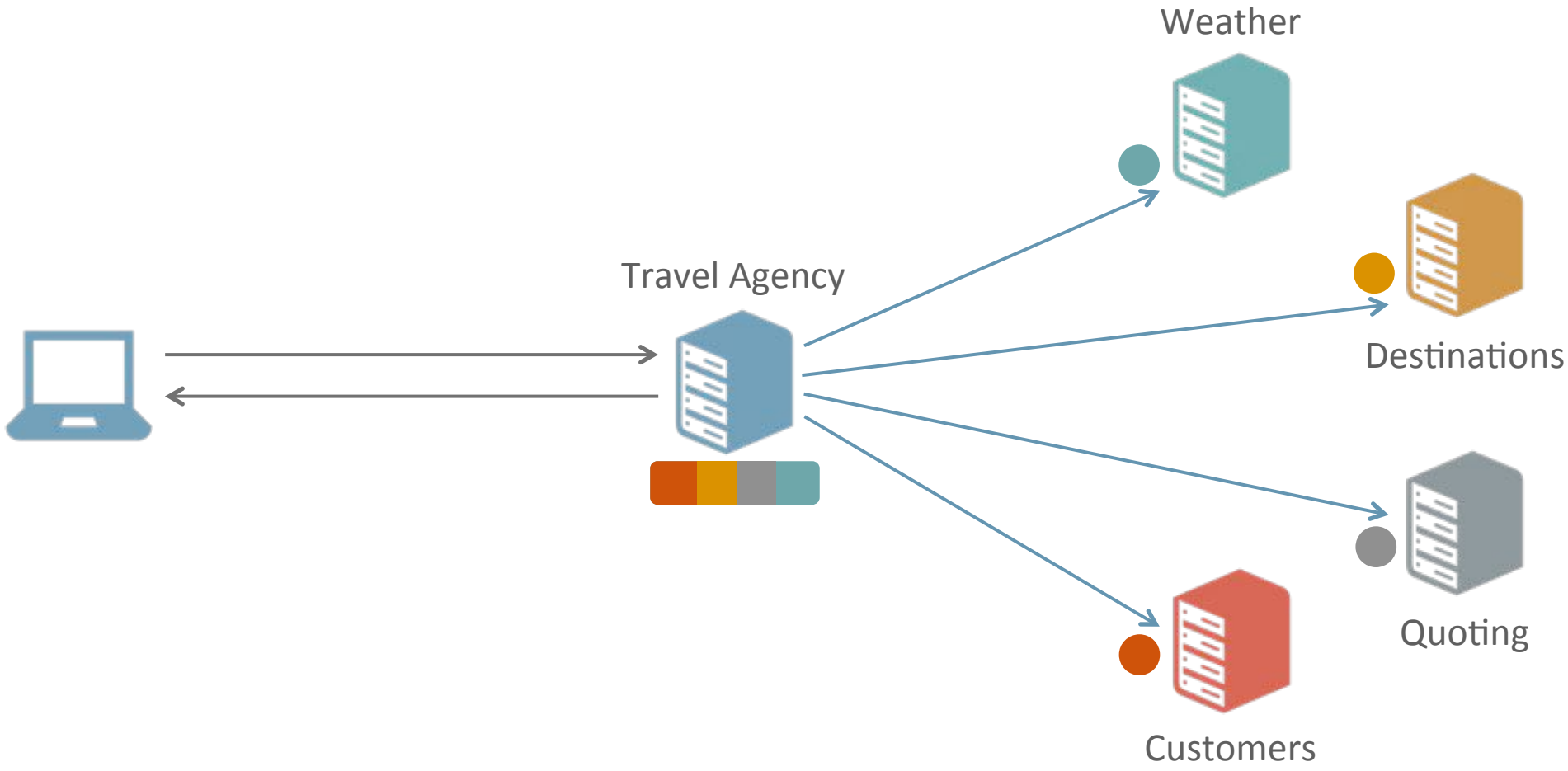
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

# The Problem

## A Travel Agency

# Orchestrating Services

## A Travel Agency Service



# The Why

# The Why

## Building an Orchestration Layer

- Client specific API
  - Different needs for various devices: screen size, payment methods, ...
- Single Entry Point
  - No need to communicate with multiple services
- Thinner client
  - No need to consume different formats of data
- Less frequent client updates
  - Doesn't matter if one service is removed in favor of another service

# The How

## JAX-RS 2.0 and Jersey 2

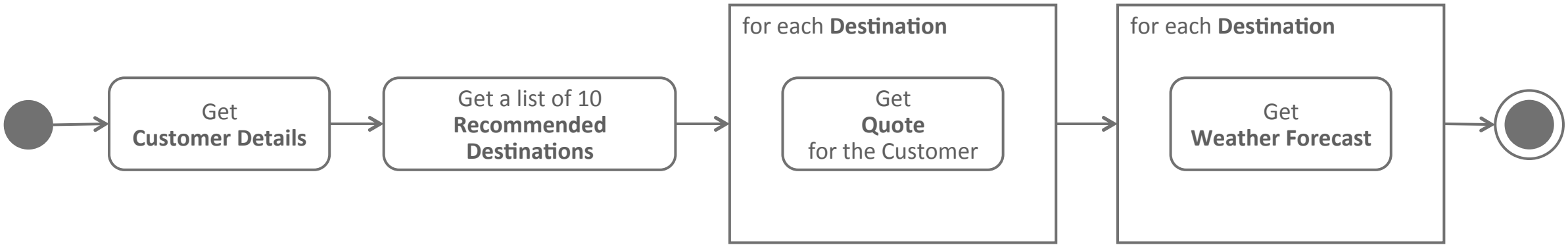
# Demo Application

## Exposed resources

- “Remote”
  - application/json, application/xml
  - delays
- “Agent”
  - application/json
  - dependent calls

# Implementing the Service

## A Naïve Approach



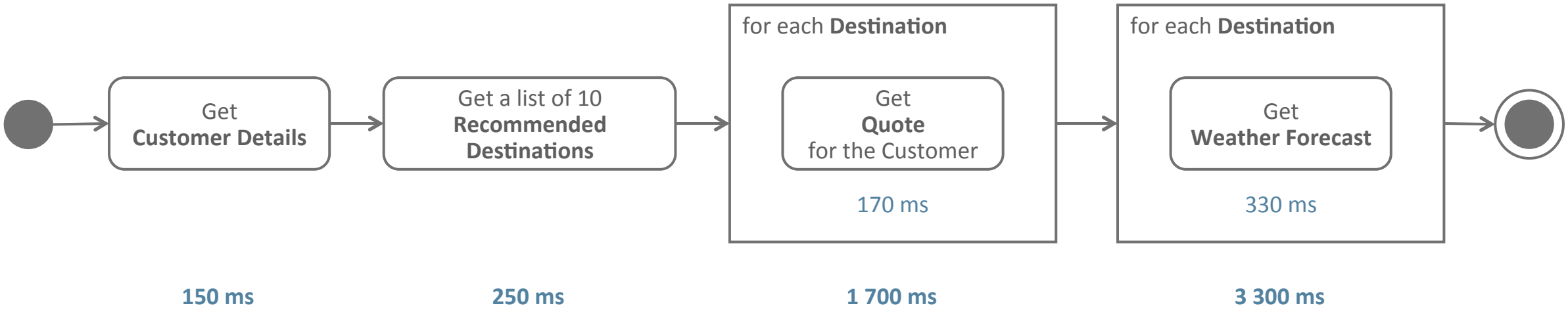
# JAX-RS 2.0 Client – Synchronous

```
Client client = ClientBuilder.newClient();

WebTarget rx = client.target("http://example.com/rx").register(JacksonFeature.class);

WebTarget forecasts = rx.path("remote/forecast/{destination}");

Forecast forecast = forecasts.resolveTemplate("destination", dest.getDestination())
                             .request("application/xml")
                             .get(Forecast.class);
```

# DEMO

# Implementing the Service

## A Naïve Approach



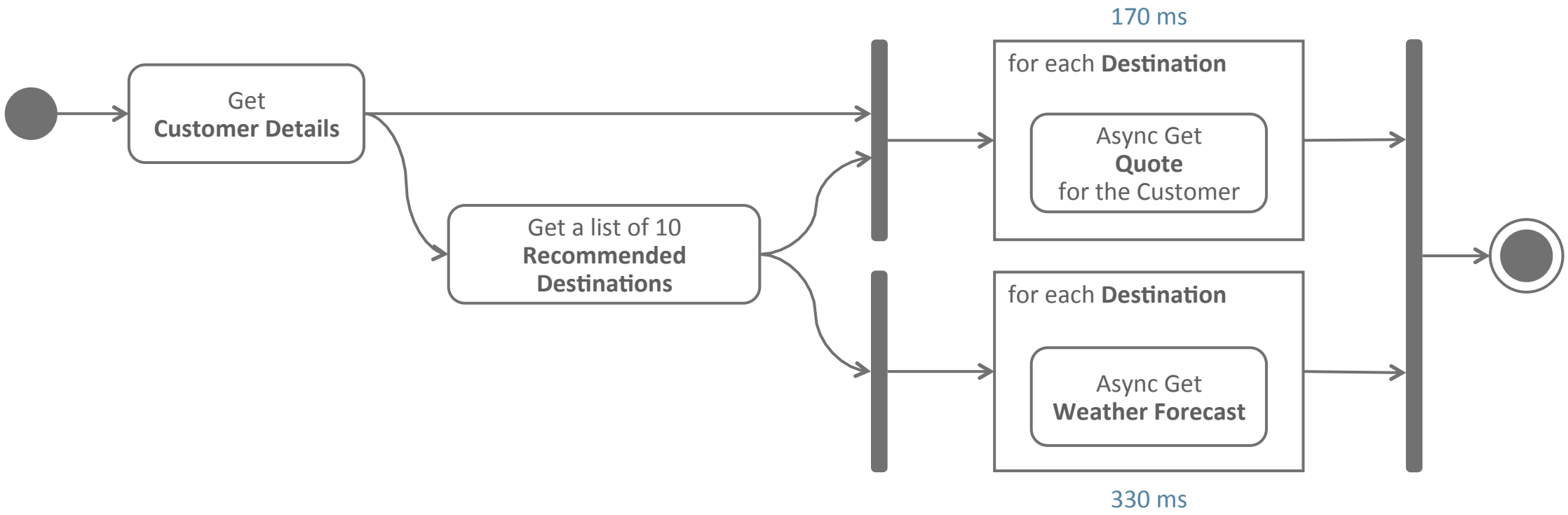

**5 400 ms**

# Client – Synchronous Approach

- Easy to read, understand and debug
  - Simple requests, Composed requests
- Slow
  - Sequential processing even for independent requests
- Wasting resources
  - Waiting threads
- Suitable for
  - Lower number of requests
  - Single request that depends on the result of previous operation

# Implementing the Service

## Optimized Approach



# JAX-RS 2.0 Client – Asynchronous

```
Future<Forecast> forecast = forecasts.resolveTemplate("destination", d.getDestination())
                                     .request()
                                     .async()
                                     .get(new InvocationCallback<Forecast>() {
                                         @Override
                                         public void completed(Forecast forecast) {
                                             // Do Something.
                                         }

                                        @Override
                                        public void failed(Throwable throwable) {
                                            // Do Something else.
                                        }
                                     });
while (!forecast.isDone()) {
    // Do Something.
}
System.out.println(forecast.get());
```

# DEMO

# Client – Asynchronous Approach

**Futures**

- Returns immediately after submitting a request
  - Future
- Harder to understand, debug
  - Especially when dealing with multiple futures
- Fast
  - Each request can run on a separate thread
  - Need to actively check for completion event (future.isDone()) or block (slow)

# Client – Asynchronous Approach

## The Callback Hell

- "Don't call us, we'll call you"
- Harder to read, understand and debug
  - Especially for composed calls (dependent)
- Need to find out when all Async requests finished
  - Relevant only for 2 or more requests (CountDownLatch)
- Fast
  - Each request can run on a separate thread
- Suitable for
  - Many independent calls

# Beyond The Callback Hell

## Reactive (Jersey) Client

# Client – Reactive Approach

- Data-Flows
  - Execution model propagates changes through the flow
- Asynchronous
  - Preferably, Speed
- Event-based
  - Notify user code or another item in flow about continuation, error, completion
- Composable
  - Compose/Transform multiple flows into the resulting one

# Reactive Java Libraries

- RxJava – `Observable`
  - Analogy to `Iterable`
  - Currently most advanced reactive API in Java
  - Contributed by Netflix – hardened & tested in production
- Java SE 8 – `CompletionStage` and `CompletableFuture`
  - Native part of JDK
  - Fits the new Java `Stream` API programming model
  - JSR166e – Support for `CompletableFuture` on Java SE 6 and Java SE 7
- Guava – `ListenableFuture` and `Futures`
  - Similar to Java SE 8

# Observable

- Observable (push)
  - retrieve data – `onNext(T)`
  - discover error – `onError(Exception)`
  - complete – `onCompleted()`
- Iterable (pull)
  - retrieve data – `T next()`
  - discover error – throws `Exception`
  - complete – `!hasNext()`

# An Observable<Response> Example

```
Observable<Response> response = … ;

List<String> visited = new ArrayList<>(10);

// Read a list of destinations from JAX-RS response
response.map(resp -> resp.readEntity(new GenericType<List<Destination>>() {}))
        // If an exception is thrown, continue with an empty list
        .onErrorReturn(throwable -> Collections.emptyList())
        // Emit list of destinations as a new Observable
        .flatMap(Observable::from)
        // Take the first 10 destinations
        .take(10)
        // Obtain a string representation of a destination
        .map(Destination::getDestination)
        // Observe the destination events on a separate thread
        .observeOn(Schedulers.io())
        // Subscribe to callbacks – OnNext, OnError, OnComplete
        .subscribe(visited::add, async::resume, () -> async.resume(visited));
```

# Reactive Jersey Client

## Extension of JAX-RS Client

- Remember `#request()` and `#request().async()` ?
  - request() returns **Invocation.Builder**; **SyncInvoker** – sync HTTP methods
  - request().async() returns **AsyncInvoker** – async HTTP methods
- **`#rx()`** and **`#rx(ExecutorService)`**
  - Return an extension of RxInvoker

# SyncInvoker and AsyncInvoker

```
public interface SyncInvoker {

    Response get();

    <T> T get(Class<T> responseType);

    <T> T get(GenericType<T> responseType);

    // ...
}

public interface AsyncInvoker {

    Future<Response> get();

    <T> Future<T> get(Class<T> responseType);

    <T> Future<T> get(GenericType<T> responseType);

    // ...
}
```

# RxInvoker and an extension Example

```
public interface RxInvoker<T> {

    T get();

    <R> T get(Class<R> responseType);

    <R> T get(GenericType<R> responseType);

    // ...
}

public interface RxObservableInvoker extends RxInvoker<Observable> {

    Observable<Response> get();

    <T> Observable<T> get(Class<T> responseType);

    <T> Observable<T> get(GenericType<T> responseType);

    // ...
}
```

# Reactive Jersey Client – contd

## Extension of JAX-RS Client

- Affected JAX-RS interfaces
  - RxInvocationBuilder<RX extends RxInvoker> extends Invocation.Builder
  - RxWebTarget<RX extends RxInvoker> extends WebTarget
  - RxClient<RX extends RxInvoker> extends Client
- Rx class
  - RxObservable
  - RxCompletionStage
  - RxListenableFuture
  - RxCompletableFuture (JSR 166e)

# Reactive Client – Creation

```
Client client = ClientBuilder.newClient();
WebTarget target = client.target("...");


// Rx
RxClient<RxObservableInvoker> rxClient = Rx.newClient(RxObservableInvoker.class);
RxClient<RxObservableInvoker> rxClient = Rx.client(client, RxObservableInvoker.class);

RxWebTarget<RxObservableInvoker> rxTarget =
    Rx.target(target, RxObservableInvoker.class);


// RxObservable
RxClient<RxObservableInvoker> rxClient = RxObservable.newClient();
RxClient<RxObservableInvoker> rxClient = RxObservable.client(client);

RxWebTarget<RxObservableInvoker> rxTarget = RxObservable.target(target);
```

# DEMO

# Q&A

# Resources

## JAX-RS and Jersey

- JAX-RS Client API
  - https://jax-rs-spec.java.net/nonav/2.0/apidocs/overview-summary.html
  - https://jersey.java.net/documentation/latest/client.html
- Jersey Rx Client
  - https://github.com/jersey/jersey/tree/master/incubator/rx/rx-client
  - https://github.com/jersey/jersey/tree/master/incubator/rx/rx-client-guava
  - https://github.com/jersey/jersey/tree/master/incubator/rx/rx-client-java8
  - https://github.com/jersey/jersey/tree/master/incubator/rx/rx-client-jsr166e
  - https://github.com/jersey/jersey/tree/master/incubator/rx/rx-client-rxjava

# Resources

## Example and Libraries

- 3rd party libraries
  - https://code.google.com/p/guava-libraries/
  - https://github.com/ReactiveX/RxJava
  - http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html
  - http://gee.cs.oswego.edu/dl/concurrency-interest/index.html
- Example (JDK7)
  - https://github.com/jersey/jersey/tree/master/examples/rx-client-webapp