Evolution of Reactive Streams API for Context-aware Mobile Applications

Przemysław Dadel, AGH University

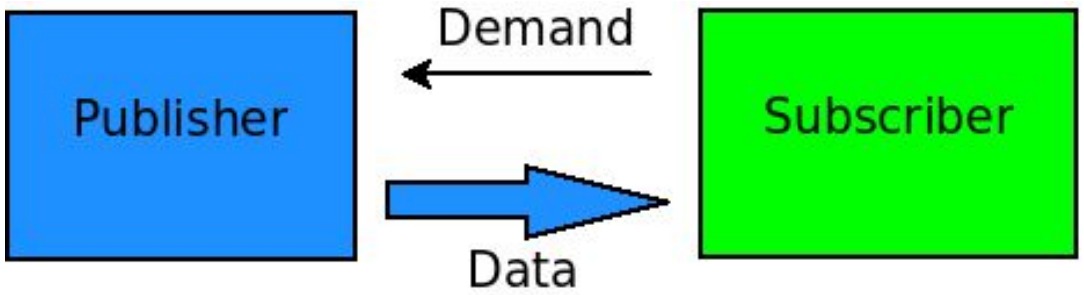
supervised by prof. Krzysztof Zieliński

What I am interested in:

- delivering services to mobile devices
- delivering services **by** mobile devices
- context-aware systems

Reactive Streams API

“The main goal of Reactive Streams is to **govern** the **exchange** of stream data across an **asynchronous boundary**—think passing elements on to another thread or thread-pool—while ensuring that **the receiving side is not forced to buffer** arbitrary amounts of data.”

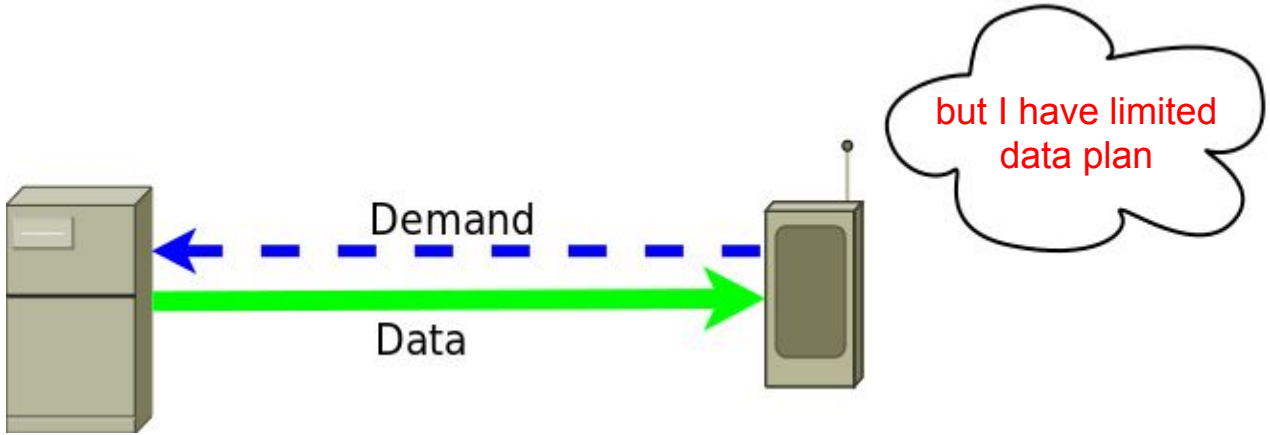


Reactive Streams API

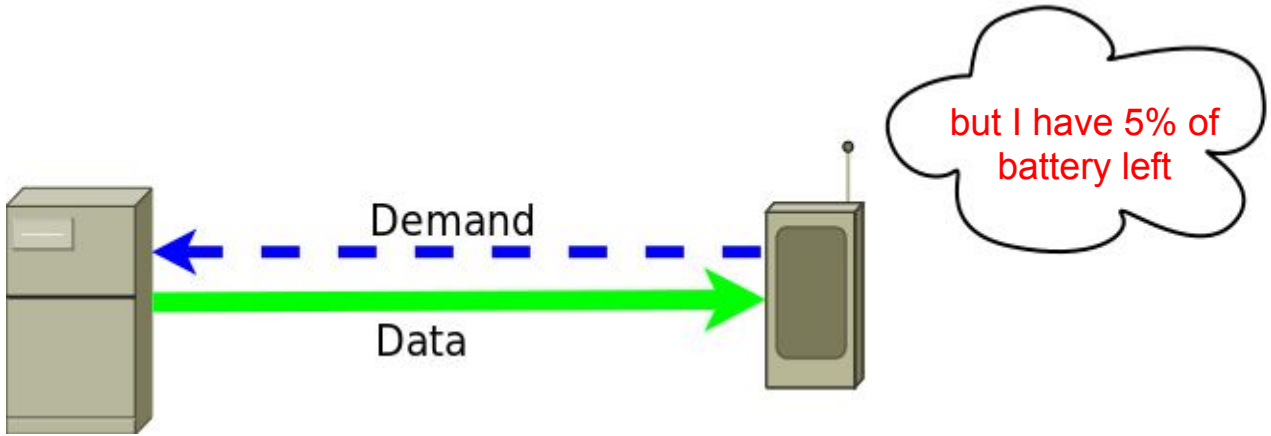
- Publisher
  - subscribe(subscription)
- Subscriber
  - onSubscribe(subscription)
  - onNext(element)
  - onComplete()
  - onError(error)
- Subscription
  - request(n)
  - cancel()

Mobile subscriber's case

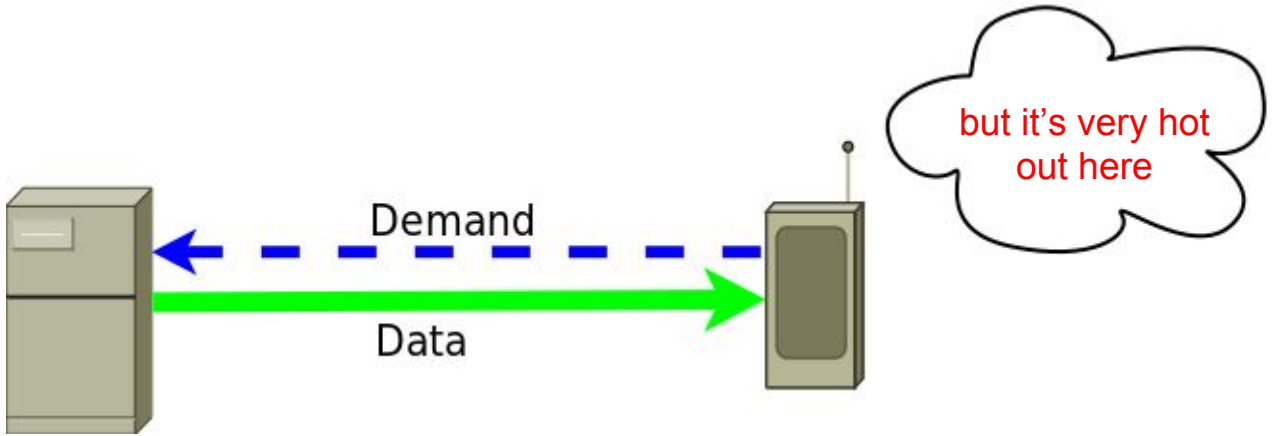
Imaginary service for delivering feed on tourists attractions/restaurants when you walk in a new city.



Mobile subscriber's case



Mobile subscriber's case



Mobile vs non-mobile

1 Non-mobile

- stable power supply*
- broadband connectivity*
- CPU and RAM dedicated to main task*
- stable outlook on the world (the server room)*

* - who knows what can happen

2 Mobile

- limited power supply
- variable network connectivity & cost
- resources allocated to front end tasks
- rich sensor set

Focus point

systems where context-awareness is an inherent complexity

Social Computing Cloud

- the experiment I run on a number of mobile devices
- I attempt to use computing power of those little machines with 8-core CPUS and 4G of RAM
- but only when they are charging and on WiFi

- in other words I use for executing computational services
- registration form a long running subscription
- device would like to get tasks that matches its capabilities (battery, network, CPU, mem)
- publisher kindly delivers tasks is when they are available

Reactive Streams are about backpressure

- We have quantitative backpressure
- We also need qualitative backpressure

Passing hints over the wire

- Publisher
  - subscribe(subscription)
- Subscriber
  - onSubscribe(subscription)
  - onNext(element)
  - onComplete()
  - onError(error)
- Subscription
  - request(n, **context**)
  - cancel()

Mobile components is far more fragile

1. Mobile device subscribe to a stream.
2. Server delivers data at its convenience.
3. There can be a lag between - context becomes misleading.
4. Mobile devices can get out-of-signal, lose connectivity, etc.

Context-aware Reactive Streams API

- Publisher
  - subscribe(subscription)
- Subscriber
  - onSubscribe(subscription)
  - onNext(element)
  - onComplete()
  - **onContextExpired**(context)
  - onError(error)
- Subscription
  - request(n, **context**)
  - cancel()

Context has a defined validity period.

Could I go different way ?

1. Multiple streams - selecting the best possible stream.
2. Another channel for passing context hints when needed.

?

1. Do mobile applications live as regular os processes ?
2. What is right frequency to updated context/select stream ?
3. How often would mobile device need to wake up to make decision ?
4. Where are the resources that scales better at server side or mobile ?

Being (too) active leads to the dark side

Design assumptions:

- mobile component should not do anything unless it is expected
- mobile component will take actions on demand and it itself specifies the demand

- back-end have a huge of battery!
- back-end scales easier
- system owner has more control over back-end

I broke Standard - I should be ashamed

- “Reactive Streams API was such a beautiful thing.”
- “How would you compose, transform and do all sorts of this functional operations when you have a sort of a state?”

Context should be propagated across net

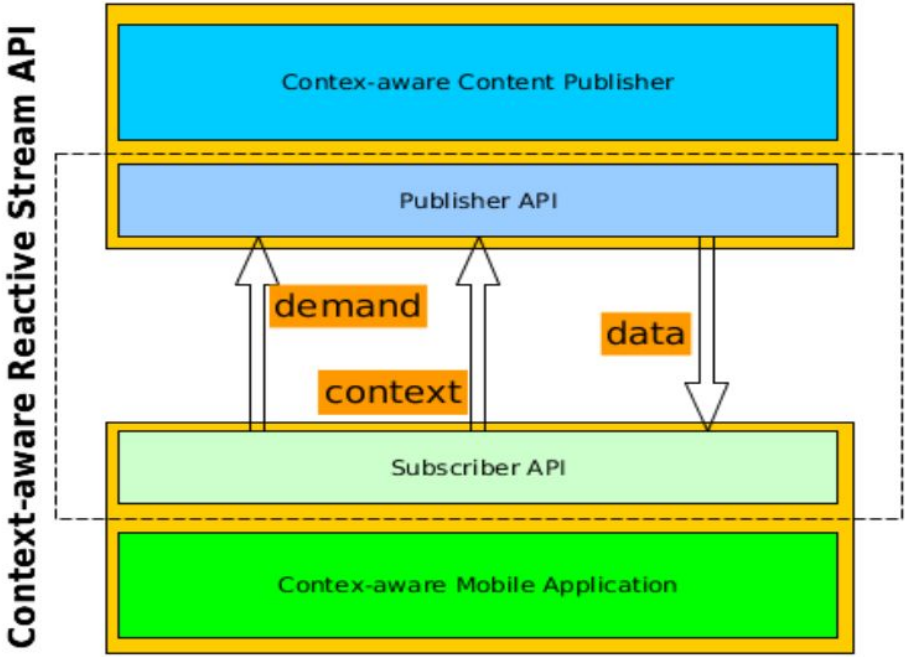


Fig. 4: Context-awareness in a software system requires for the context to be propagated across communication API.

Narrowing a subscription

```
581  val ctxAwarePublisher: CtxAwarePublisher[Task] = ...
582  val regularPublisher = ctxAwarePublisher.withContextEnrichment( (ctxAwareSubscription) -> new Subcription {
583
584    def request(n: Int) {
585      ctxAwareSubscription.request(n, computeContext());
586    }
587
588  })
```

https://en.wikipedia.org/wiki/Jade_Belt_Bridge#/media/File:Gaoliang_Bridge.JPG

Publisher's Publisher

- Publisher API does not says anything about transformation - RxJava, Akka Streams add this.
- Context should be propagated up to the point where it is understable.
- Fan-in and fan-out operation should come with sort of context merging.

Context-awareness is an inherent complexity

- with modified API I attempt to minimize **accidental** complexity
- we define context-aware communication channel
- we allow a mobile client to stay passive
- we anticipate that mobile component can be off and on frequently

- expressiveness of client implementation
- scalability responsibilities shifted to server

3 other facts I learned from *this*

1. Abstractions will leak, always, but expressing them as API forces you to comply (compiler forces you to that).
2. Mobile (Android) OS process model is not what we have on a (Linux) server: disposable activities vs processes/threads
3. API refinement should be bottom-up.

Thank you

Please type your questions here: .....