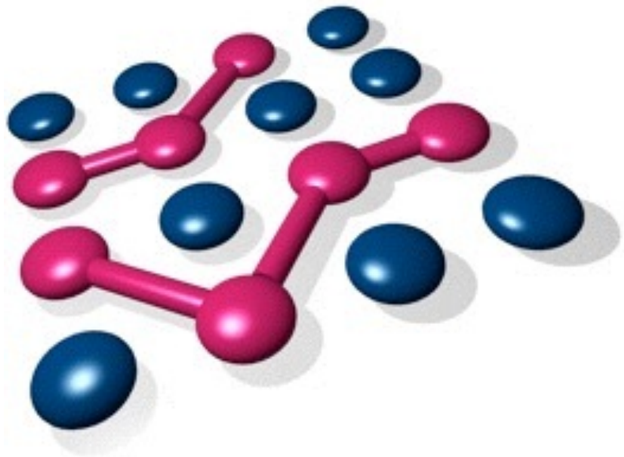
Paremus



# Asynchronous OSGi:
## Promises for the masses

**Tim Ward**

http://www.paremus.com
info@paremus.com


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Who is Tim Ward?
## @TimothyWard

- Senior Consulting Engineer, Trainer and Architect at Paremus
- 5 years at IBM developing WebSphere Application Server
  - Container Implementations for Java EE and OSGi, including Blueprint, JPA, EJB and JTA
- OSGi Specification lead for JPA, Weaving Hooks and Asynchronous Services
- PMC member of the Apache Aries project
- Previous speaker at Java One, EclipseCon, Devoxx, Jazoon, JAX London, OSGi Community Event...
- Author of Manning's Enterprise OSGi in Action
  - http://www.manning.com/cummins



Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# What we're going to cover

- Why is the World Going Asynchronous?
- Async Programming Primitives
- Making Synchronous Services Asynchronous
- Optimising Calls into Asynchronous Services
- Demo


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# New OSGi Specifications


Last week the OSGi Core Release 6 Specification received final approval

It's available now at http://www.osgi.org/Specifications/HomePage

The OSGi Board also approved a public draft of OSGi Enterprise Release 6

Available at http://www.osgi.org/Specifications/Drafts

This draft contains the specifications we're going to be talking about!


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Why is the World Going Asynchronous?


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Origins of Asynchronous Programming

Many peopler think of Async Programming as “recently popular”…

The terms “Promise” and “Future” originate from 1970s research papers

Proposed as mechanisms to handle large-scale parallel execution

Asynchronous Programming concepts have been with us for a long time!

User Interface EventHandler callbacks “must not block the UI thread!”


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Advantages of Asynchronous Programming

- Asynchronous systems offer better performance
  - This does assume that there isn't a single bottleneck!


- Asynchronous Distributed systems typically scale better
  - Less time spent waiting for high-latency calls
  - More reasonable failure characteristics
- Parallelism is easier to exploit
- Network Infrastructure is intrinsically asynchronous
  - Network Engineers have been working this way for decades


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Problems with Asynchronous Programming

Human brains aren't really designed to cope with asynchronous logic

- Thought processes tend to be "do **A**", then "do **B**" then "the result is **C**"
- Asynchronous Programming is only useful if the benefits of using it outweigh the extra cognitive overhead

Synchronous calls offer a natural "brake" in RPC

- Asynchronous "Event Storms" can overwhelm remote systems, even using a single client thread.

Deadlocks are still possible, and can be hard to diagnose


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# So what's changed?

The latest asynchronous revolution seems to be sticking!

Multi-core is pervasive now, request-level parallelism isn't always enough to max out hardware any more

JavaScript has been a major driver in the last 5-10 years, in a single threaded environment. Async is a necessity!

Commonly used OO languages are becoming more "functional"

Java and C++ have added lambdas








Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Async Programming Primitives


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Async Basics

The Promise is the primitive of Asynchronous Programming

A “Promise” represents a delayed value that will be “resolved” in the future

“Resolving” a promise sets its value, or its failure

OSGi Promises are based on JavaScript Promise concepts, but also significantly influenced by Scala



Strictly, the OSGi (and most JavaScript) Promises are “Future” types as they have no methods for resolving themselves

In both cases a Deferred type can create and resolve a default Promise implementation


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# The OSGi Promise

The Promise Contract (based on JavaScript Promises)

- The Promise is resolved or failed with a single value, at most once
- Listener callbacks are called at most once
- Promises remember their state (including their resolution / failure value)
- Promises behave the same way regardless of whether they are already resolved or resolved in the future.

Promises can be treated as a shareable value type

- Effectively immutable (unless you created it!)
- Thread safe
- No need to unwrap it…


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# The Deferred

OSGi provides a default Promise implementation for clients to use

The Deferred is held by the provider of the Promise, and can be used to set its value or failure

```
final Deferred<Long> deferred = new Deferred<Long>();

new Thread() {
   public void run() {
      try {
         Long total = service.calculateDifficultSum();
         deferred.resolve(total);
      } catch (Exception e) {
         deferred.fail(e);
      }
   }
}.start();

Promise<Long> promise = deferred.getPromise();
```


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Promise Callbacks

Promises do support “synchronous” usage similar to Java’s Future

- isDone(), getValue(), getFailure()
- The getXxx() methods block

It’s better to use Promises with callbacks (never block, remember?)

- `then(Success)` or `then(Success, Failure)`

Success and Failure are both SAM interfaces, suitable for Lambda usage

```
then((p) -> success(p.getValue()),
  (p) -> fail(p.getFailure()));
```

Callbacks should be fast - Success can return a Promise for longer tasks


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Chaining Promises

Basic chaining uses then(…), a new Promise is created that resolves when:

- the success callback returns `null`; or
- the success callback throws an error; or
- the Promise returned by the success callback resolves

Chaining allows you to compose sequences of events

- Promises are Monads
- Complex behaviours can be achieved without `if/else` branching

More complex functional chains are natively supported

- Filtering/Mapping values, recovering failures…


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# The Promises utility class

The `Promises` type provides number of common functions

- Creating Promises that are already resolved or failed
- Very useful when mapping / recovering promises

Another useful utility is Promises.all(Promise…)

- It creates a “latch” Promise that resolves when all others are complete
- Good for final clean up, or for notifying users about aggregate tasks

Remember that chains and latches work even if a Promise is already resolved!


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# A Chaining Example

Download an XML file, preferably from a mirror URL, and parse it using JAXB

```
Promise<Library> promise = getMirror(libraryURL)
        .fallbackTo(Promises.resolved(libraryURL))
        .then((p) -> downloadFile(p.getValue()))
        .map((p) -> (Library) JAXB.unmarshal(p, Library.class)));
```

```
public Promise<URL> getMirror(URL url)
    throws Exception {
    . . .
}

public Promise<File> downloadFile(URL url)
    throws Exception {
    . . .
}
```

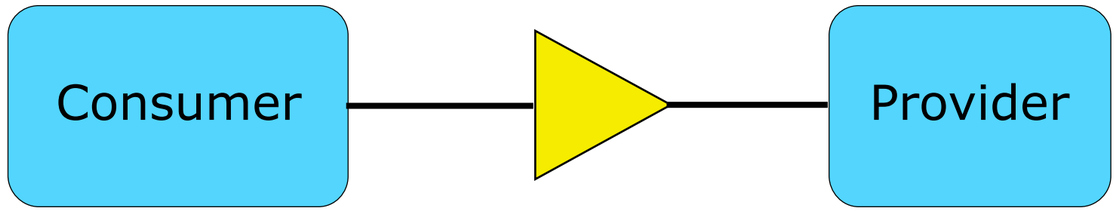

Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Making Synchronous Services Asynchronous


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# The OSGi service model

The OSGi service model is useful and widely used




Service objects are obtained from the Service Registry and invoked

- The call semantics depend upon the Service API
- Most APIs are synchronous in operation


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Question?

As a service client, how do I make an asynchronous call to a service?

- I can't / don't want to rewrite the service
- I'd prefer not to start threads or manage a thread pool
- I'd also like my code to stay as clean as possible

Ad-hoc solutions are messy, but have been the only choice up to now


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# The Async Service - Requirements

- Allow arbitrary OSGi services to be called asynchronously
  - The call must be transparent to the backing service, no opt-in required
- Must support methods that have a return value, as well as void
- Must not rely on generated source, client-local interfaces or reflection
  - Source gets out of sync, and is hard to maintain
  - Reflection is just ugly!
- Must allow services that are already asynchronous to “hook-in” to the model
  - This does require opt-in


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Unit testing Mocks - Requirements

- Allow arbitrary Objects to be called, remembering the calls that are made
  - The mocking must be transparent, no opt-in required
- Mocks must allow methods to return a specific value
- Must not rely on generated source, client-local interfaces or reflection
  - The test code is using the real interfaces/objects

Notice any similarities?


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# How to use the Async Service

To make asynchronous calls on a service you need an Async Mediator

```
MyService mediated = async.mediate(myService);
```

Services can be mediated from a `ServiceReference` or the service object

- `ServiceReference` is better because the mediator can track availability

Clients then use the mediated object to make a normal method call

- This records the call, just like a mock would in a test

To begin the asynchronous task you pass the return value to async.call(T)

```
Promise<Long> p = async.call(mediated.pianosInNYC());
```


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# How to use the Async Service (2)

Void methods can be slightly trickier

- There's a no-args version of `call()` which returns Promise<?>
- Promises for `void` methods resolve with `null` when they complete

If you don't need a Promise, then consider fire-and-forget

- The `execute()` method doesn't create a Promise
- Fire-and-forget can be *much* more efficient*

* More about this later!

The Async Service is Thread Safe, but mediators may not be

- Never share the Async object between Bundles, always get your own
- Failure to do the above can lead to class space errors and security flaws!


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Async Service Example

Perform two long running calls in parallel, then aggregate the result

```
public void setAsync(Async async) {
   this.async = async;
}

public void setMyService(ServiceReference<MyService> myService) {
   this.myService = myService;
}

public Long getCombinedTotal() throws Exception {
   MyService mediated = async.mediate(myService);

   Promise<Long> bars = async.call(mediated.barsInNYC());

   return async.call(mediated.pianosInNYC())
         .flatMap((pVal) -> bars.map((bVal) -> pVal + bVal))
         .getValue();
}
```


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Optimising Calls into Asynchronous Services


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# What if we can do better?

Sometimes there's more to a service than meets the eye…

Some service implementations are asynchronous under the covers

- Blocking a thread from the Async Service to call these services is a waste!

We need to allow these services to provide a Promise to the Async Service

- Enter the AsyncDelegate interface!

Services that implement AsyncDelegate are given the opportunity to provide their own Promise, bypassing the Async Service

`null` can be returned if the AsyncDelegate can't optimise a particular call


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

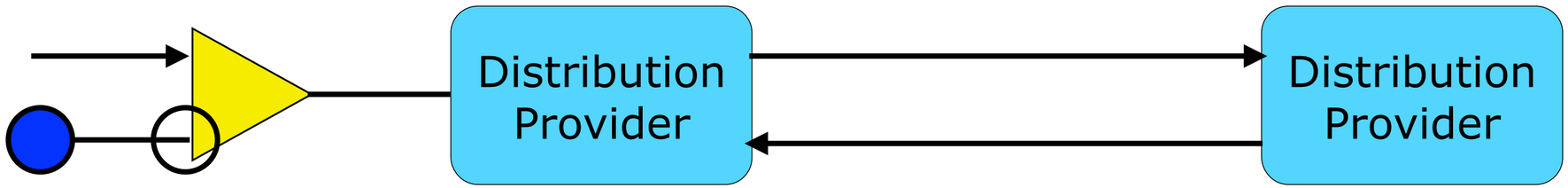
# Isn’t this a bit niche?

Implementing AsyncDelegate may seem like a micro-optimisation

- Short lived objects are cheap
- How many services does this really apply to?

Remote Services allows most OSGi services to be transparently remoted

- The details of the distribution mechanism are hidden
- If the distribution provider supports asynchronous communications then it can optimise calls that it knows to be asynchronous





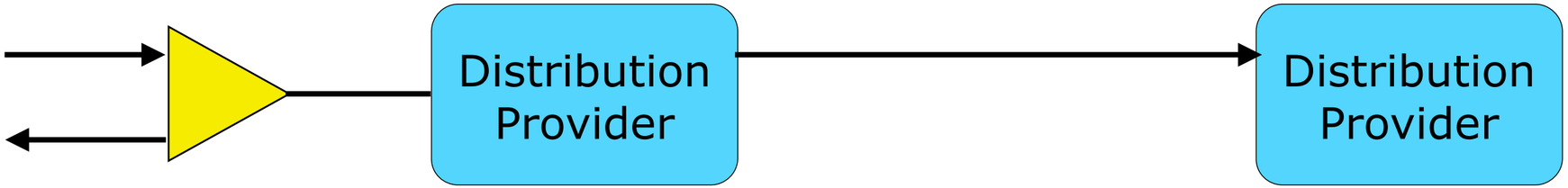
Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Further optimising our calls

Earlier we talked about fire-and-forget calls…

Fire and forget ignores return values, and offers no callback on completion

- This is often less useful, but can be exactly what you need
- It can also be very heavily optimised by remote services!




Removing the return flow decreases work at both the local and remote ends

- It also significantly reduces the load on the network


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Implementing your own AsyncDelegate

The AsyncDelegate interface defines two methods:

- Promise<?> async(Method, Object[]) throws Exception;
- boolean execute(Method, Object[]) throws Exception;

It's only worth implementing if:

- You can optimise the asynchronous execution of one or more methods
- Your service is likely to be called asynchronously

Return `null` or `false` if the call can't be optimised

Exceptions should only be thrown if the call is impossible (e.g. too few args)

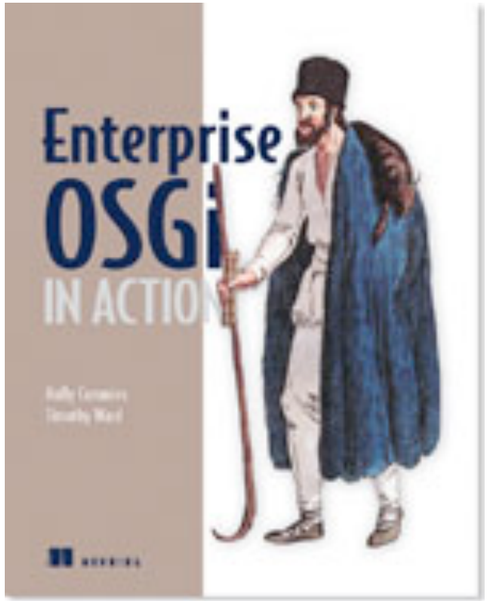

Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Demonstration: using the Async Service


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.

# Thanks!

- For more about OSGi...
  - Specifications at http://www.osgi.org
  - Enterprise OSGi in Action
  - http://www.manning.com/cummins
- For Service Fabric examples and docs
  - http://docs.paremus.com/
- For more about the demo (including source code)
  - http://bit.ly/paremus-async




**44% off** OSGi books using code **osgi14cf** at **manning.com**

Questions?

**http://www.paremus.com**
**info@paremus.com**


Copyright © 2005 - 2014 Paremus Ltd.
May not be reproduced by any means without express permission. All rights reserved.