RFP-192-Messaging

Final

8 Pages

Abstract

Asynchronous communication is an important factor in today's business applications. Especially in the IoT domain but also for distributed infrastructures the communication over publish/subscribe protocols are common mechanisms. Whereas the existing OSGi Event Admin specification already describes an asynchronous event model within an OSGi framework, this RFP addresses the interaction of an OSGi environment with third-party communication protocols using a common interface.

0 Document Information

0.1 Table of Contents

0.2 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 6.1.

`Source code is shown in this typeface.`

0.3 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	*28.12.2018*	*Initial - Mark Hoffmann*
0.1	*31.01.2019*	*David Bosschaert – minor editorial changes and some comments*

1 Introduction

In the past there have already been some efforts to bring asynchronous messaging into the OSGi framework. There was the Distributed Eventing RFC-214 and the MQTT Adapter RFC-229. In addition to that there are further available specification like OSGi RSA, Promises and PushStreams, that focus on remote events, asynchronous processing and reactive event handling. Promises and PushStreams are optimal partners to deal with the asynchronous programming model, that comes with the messaging pattern.

Especially because of the growing popularity of the IoT domain, it is important to enable OSGi to be connected with the IoT world.

Besides that, protocols like AMQP, the Kafka Protocol or JMS are also heavily used in back-end infrastructures. This RFP~~C~~ tries to ~~re~~address this use-cases again, in respect to the existing supporting specifications. For a seamless use of OSGi as well in IoT infrastructures and cloud-infrastructures, it is important to provide an easy to use and also seamless integration of different communication protocols in OSGi.

With the Event Admin specification, there is already an ease to use approach, for in-framework events. Distributed events often needs additional configuration parameters like quality of service, time-to-live or event strategies that needs to be published to. So this RFP~~C~~ is seen for standalone use but also as an extension to the Event Admin to provide the possibility for a Remote Event Admin.

2 Application Domain

Messaging is a pattern to reliably transport messages over a inherent unreliable network from a produce to one or more receivers. The process is to move messages from one system to another or many. The messaging system uses channels to do that. Because it is never clear, if the network is available or the receiver system, it is the task of the messaging system to handle that.

There are also different concepts in moving messages:

1. Send and Forget – Waits to successful send
2. Send and Forward – Waits to successful receive

Using the ”Send and Forget” concept, the sender, will not know, if the message was received, whereas with “Send and Forward” this receive information is known to the sender.

There are also some differences in the messaging channel types, that are common:

1. Point-to-point
2. Publish-Subscribe
3. Guaranteed delivery
4. Invalid message
5. Dead-letter

These channel types are used to deal with certain use-cases like RPC over messaging or a quality of service. The last is heavily used in MQTT.

Another important fact is the structure of the messages. They usually consist of a header and body. The body contains the payload that is an array of bytes. The header provides additional properties for the message. There are some common properties that are used for handling RPC, sequencing/ordering of messages, time-synchronization, filtering and others. This is important because messaging decouples communication and has different demands regarding assumptions that are made to the process compared to a local call. Thus there is are additional semantics for. e.g. time-outs, retry-counts, address-spaces.

Messaging in general induces an asynchronous programming model. Therefore Promises and PushStreams are already existing specifications that are an optimal solution for data-handling as well as scaling of actors over more than one thread. These specifications allowing flexible message transformation into internal data formats. Further Promises provide the possibility to realize patterns like message construction or aggregation. [1]

2.1 Terminology + Abbreviations

2.1.1 Message

A message is a data structure that holds the payload and additional meta-information about the content.

2.1.2 Channel

Channels are queues. Theses are paths to connect programs and interchange the messages. They are concurrently accessible, so many senders can write to them and receiver from channels with different threads.

2.1.3 Sender / Publisher

A sender writes a message into a channel. Depending on the concepts, described above, the sender waits for the receive acknowledge of the receiver/s or just for a successful submit.

2.1.4 Receiver / Subscriber

A receiver reads from a channel.

2.1.5 Messaging RPC

Remote procedure calls over messaging. Sending a request and receiving a request are two separate atomic operations. Thus waiting for a response is not a blocking operation, like in ordinary RPC operations. A special message information, the correlation identifier, is used to assign a request to a response. Sometimes the reply-to address can be generated from the messaging system and is also submitted as property with the request message.

3 Problem Description

The OSGi Alliance already has a successful specification for messaging within an OSGi framework. The EventAdmin specification ~~are~~is well defined and widely used. The same is for the RSA specification that provides a good ground for synchronous calls. Also asynchronous remote services are supported in the RSA.

In the domains of IoT there are standardized protocols to connect remote devices and submit data over a broker based messaging system from remote clients. But also in cloud-based infrastructures, messaging systems are often used for de-coupling of services or functions.

To interact with such systems the implementer has to deal with messaging inherent conditions and use-cases, that are currently not covered, by existing specifications. With OSGi Promises and the PushStream specification there are already major parts available to deal with an asynchronous programming model. This is a requirement when using messaging.

The missing pie~~a~~ce is a harmonized way to send, receive data that support the corresponding messaging pattern concept. Consuming and producing data using common protocol~~'~~s like AMQP or MQTT using OSGi services, would integrate an OSGi application into more systems.

Also other specifications could ~~take profit~~benefit from this RFP. So it could be possible to easily get an RSA implementation for message when using this common way. It could also be possible to ~~get~~ provide a remote Event Admin.

4 Use Cases

4.1 Message sending

A publisher wants to send single messages as well as consume messages as a stream, that are provided from another PushStream. It is necessary to provide the possibility to submit additional context information for this channel or sending process.

It should be possible to register a PushStream consumer as a service, that then is used by the corresponding messaging whiteboard. This is useful especially for send-and-forget messages.

4.2 Receiving messages

As a recipient I want to receive messages from a~~s~~ PushStream. I also want to get informed about connection errors of the channel. As receiver I want to have the possibility to get context/meta-information from the message.

It should be possible to directly inject a PushStream as service for a messaging configuration for a pre-defined channel.

RPC over messaging

I want to use RPC calls over messaging, as far as the implementation supports this intent. As I don't necessarily want to take care about a correlation and eventually generation of a return-location for the response.

The implementation should take care about the use-case when an endpoint acts as RPC caller (publish and subscribe) as well as when the current endpoint acts a RPC call receiver (publish on subscribe). Both are expected to be in an asynchronous way.

4.3 Acknowledge Filtering

The send-and-forget principle, in many implementations, uses an auto-acknowledge, which acknowledges a message after it was sent. This increases the throughput of a system, but is considered as unsafe. Because of that, it may be necessary, that messages needs to be acknowledged or rejected manually. It is expected that there are various implementations to handle this in its protocols and clients.

4.4 Event Admin

It should be possible to connect an OSGi Event Admin with a channel, to send or receive Event Admin Events. For that a topic-to-channel mapping could be defined

5 Requirements

5.1.1 General

- MSG010 – The solution MUST be technology and protocol independent.
- MSG020 – The solution MUST support peer-to-peer as well as broker based messaging systems
- MSG030 – The solution MUST be configurable (address-space, timeouts)
- MSG050 – The solution MUST announce their capabilities/intents to service consumers
- MSG060 – The solution MUST provide information about registered channels, driver connection states
- MSG070 – The solution MUST support the asynchronous programming model

5.1.2 Channels

- MSG100 – It MUST be possible to send asynchronous messages to remote recipients.
- MSG120 – The solution MUST support the point-to-point channel type
- MSG130 – The solution MUST support the publish-subscribe channel type
- MSG140 – The solution MUST support guaranteed delivery / quality of service
- MSG150 – The solution MUST support send-and-forget and send-and-forward semantics
- MSG160 – The solution SHOULD support RPC over messaging, if possible. For that the solution MUST act as caller (publish and subscribe) as well as RPC receiver (subscribe on publish)
- MSG170 – The solution SHOULD support filter semantics like exchange / routing-key and wildcards for channels
- MSG180 – The solution MAY define a last-will message handling, when the channel was disconnected from participants
- MSG190 – The solution MAY provide ~~a~~ reconnection handling, when a channel was disconnected from participants

Messages

- MSG200 – Messages bodies MUST be sent in a programming language and protocol agnostic way. So messages MUST NOT be restricted to a certain content-format
- MSG210 – It MAY be possible to support additional message sequencing and correlation
- MSG220 – The solution MAY define a content encoding

- MSG230 – The solution MAY support message time-to-live information
- MSG240 – The solution MAY support manual acknowledge/reject support for messages

5.1.3 Event Admin

- MSG300 – The solution SHOULD provide a bridge from a channel to Event Admin topics, to send or receive events

6 Document Support

6.1 References

[1]. Enterprise Integration Pattern: Designing, Building, and Deploying Messaging Solutions. Gregor Hohpe, Bobby Woolf. ISBN 0-133-06510-7.

6.2 Author's Address

Name	Mark Hoffmann
Company	Data In Motion Consulting GmbH
Address	Kahlaische Str. 4, 07743 Jena, Germany
Voice	+49 175 7012201
e-mail	m.hoffmann@data-in-motion.biz

6.3 End of Document