RFP-192-Messaging

Proposed Final Draft

9 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*
0.2	*13.02.2019*	*Mark Hoffmann – changed wording for RPC and added some proposed use-cases*
0.3	*19.02.2019*	*Mark Hoffmann – comments from the F2F Berlin discussion*
0.4	*13.03.2019*	*Michael Roeschter – reviewed, edited for clarity with regard to design goals. The F2F in Berlin clarified the design goal to provide a simple interface to messaging systems, but not providing access to the full feature set of Enterprise class messaging solution (like MQ Series or TIBCO EMS) or massively scalable solutions (like Kafka).*
0.5	*21.05.2019*	*Mark Hoffmann – review and fixed some typos*
0.6	*02.06.2019*	*Mark Hoffmann – Accepted changes from F2F discussion, rewrote Use-case section*
0.7	*28.06.2019*	*Christian Schneider – Some more rewording of the Use-case section*

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.

Because of the growing popularity of the IoT domain, it is important to enable OSGi to be connected with the common services of the IoT world. This RFP is meant to provide an easy to use solution in OSGi, to connect to and work with messaging systems. It is not meant to provide access to the full feature set and service guarantees of Enterprise class messaging solution like MQSeries or TIBCO EMS or massively scalable solutions like Kafka. Service guarantees and configuration details will be designed as configuration hints. The implementation will be optional and depend on the binding.

Protocols like AMQP, the Kafka Protocol or JMS are heavily used in back-end infrastructures. This RFP tries to address the use-case for connecting to those protocols with a subset of their functionality. 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 configured at connection time or set at message publication time. This RFP 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 an 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 or the receiver system is available, it is the task of the messaging system to handle that.

There are also different concepts in moving messages:

1. Send and Forget – Guarantees successful send
2. Send and Forward – Guarantees eventual successful receive

The following communication patterns exist:

1. Point-to-point
2. Publish-Subscribe
3. Guaranteed delivery
4. Temporary/transient channel
5. Dead-letter
6. Monitoring, error and administration channels

The use cases above cover Reply-To style communication and handling of common error conditions.

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 Reply-To, 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 named resources to connect programs and interchange the messages. .

2.1.3 Sender / Publisher

A sender writes a message into a channel.

2.1.4 Receiver / Subscriber

One or more receivers read messages from a channel.

2.1.5 Reply-To Messaging

Sending a request and receiving a response are two separate atomic operations. Thus waiting for a response is not a blocking operation in the underlying implementation., 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 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.

Today, to interact with such systems the implementer has to deal with messaging protocol specifics and operational conditions, that are 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 piece is a standardized way to send and receive data that supports the messaging patterns. Consuming and producing data using common protocols like AMQP, MQTT or JMS using OSGi services, would integrate an OSGi application into more systems.

Also other specifications could benefit from this RFP. It should be possible to layer RSA remote calls over messaging. It should also be possible to provide a remote Event Admin service.

3.1 Intents

Messaging systems vary widely in their capabilities and are configurable with regard to guarantees of delivery. We do not want to expose this complexity the user of this solution. The RSA specification uses intent for that purpose.

4 Use Cases

4.1 Message sending

A publisher wants to be able to send single messages as well as provide messages through a PushStream. It must be possible to provide additional message properties for either case.

4.2 Receiving messages

As a recipient I want to be able to poll the next message in a channel. I want to do that in a blocking or non-blocking way.

It is also expected to support filter criteria for receiving messages, if supported by the underlying messaging system. So it could be possible to obtain a message with a specific identifier or correlation-ID.When using a messaging system that supports journalling (like Kafka) then I want to be able to be able to start consuming from any point in a retained history. Alternatively to providing an absolute position to start from, I would like to be able to start from earliest or latest. To keep track of the position of the recipient in the journal received messages should provide suitable position information like an offset.

4.3 Reply-To messaging

I want to use efficient Reply-to communication, as long as the implementation supports it. In that case I don't want to take care about setting or/and accessing Reply-To headers, correlation-ID or generating temporary channels. This should be handled by the implementation.

The implementation should respect the use cases to be a caller or a call receiver.

As a developer I want to be able to use a programming style, that is similar to synchronous programming.

4.4 Guarantee of delivery

If an implementation supports guarantees of delivery, it should be possible to take profit of this feature. So e.g. many implementation support an auto-acknowledgment for the send-and-forget principle. Otherwise it maybe necessary to acknowledge or reject messages explicitly in certain situations.

There are also other possible implementation specific aspects like: Storage guarantees for messages, journalling, persistence, exactly once delivery, transactions, out of sequence acknowledgment, at most once delivery, duplicate delivery. It is expected that various implementations handle guarantee of delivery in a different way.

4.5 Event Admin

As a user of the system I would like 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

4.6 Intents

As a user of the system I want to be able to specify which features (similar to intents in RSA) I need. I would then only want to bind to a messaging service that supports all required intents.

Implementation specific behavior, like the before mentioned "guarantee of delivery" can also be signaled.

5 Requirements

5.1.1 General

- MSG010 – The solution MUST be technology, vendor and messaging protocol independent. MSG030 – The solution MUST be configurable (address-space, timeouts, quality of service guarantees)
- MSG050 – The solution MUST announce their capabilities/intents to service consumers
- MSG060 – The solution MUST provide information about registered channels, client connection states, if available

- MSG070 – The solution MUST support the asynchronous programming model
- MSG080 – The solution MUST support a client API
- MSG090 – The solution MUST respect requested intents
- MSG095 – The solution MUST announce its supported intents
- MSG100 – The solution MUST fail when encountering unknown or unsupported intents.

5.1.2 Channels

- MSG100 – It MUST be possible to asynchronously send messages to a channel.
- MSG120 – The solution MUST support systems that support point-to-point channel type
- MSG130 – The solution MUST support systems that support the publish-subscribe channel type
- MSG140 – The solution MUST support quality of service
- MSG150 – The solution MUST support send-and-forget and send-and-forward semantics
- MSG160 – The solution SHOULD support Reply-To calls, if possible. For that the solution MUST act as caller (publish and subscribe) as well as Reply-To receiver (subscribe on publish)
- MSG170 – The solution SHOULD support filter semantics like exchange / routing-key and wildcards for channels
- MSG180 – The solution MAY support a do-autocreate as well as do-not-autocreate

Messages

- MSG200 – Messages bodies MUST support sending of byte-data
- MSG205 – The implementation MAY place limits on the size of the messages that can be send. The existence of a message size limitation for an implementation SHOULD be signaled.
- MSG210 – It SHOULD be possible to support additional message properties like sequencing and correlation. The implementation SHOULD provide access to properties when available.
- 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
- MSG250 – The solution MAY have a journalling support
- MSG260 – It MUST be possible to identify the channel the message was received on

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