RFC 214: Distributed Eventing

Draft

15 Pages

Abstract

10 point Arial Centered.

The OSGi specifications have described how to distribute OSGi services across VM boundaries now for a number of years. However many OSGi services are synchronous in their nature. Many of today's business applications require asynchronous distributed communication mechanisms. While the OSGi Event Admin specification describes an asynchronous eventing model inside the Java VM this does not address event distribution to other Vms. In addition, while the OSGi Asynchronous Services specification defines mechanisms for asynchronously invoking services, it does not address some concerns specific to eventing. This RFP aims to address the issue of Distributed Events in an OSGi context.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

0.2 Trademarks

0.3 Feedback

This document can be downloaded from the OSGi Alliance design repository at https://github.com/osgi/design The public can provide feedback about this document by opening a bug at https://www.osgi.org/bugzilla/.

0.4 Table of Contents

0.5 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 1.

`Source code is shown in this typeface.`

0.6 Revision History

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

Revision	Date	Comments
Initial	Aug 27 2014	Initial version of the document ahead of the Madrid F2F. Copied from the RFP. Tim Ward (Paremus)

1 Introduction

This RFC began as an RFP nearly two years ago, in an effort to provide a better asynchronous messaging and eventing solution between OSGi framework. The RFP experienced some delays because parts of the problem space related to other OSGi RFCs. The primary blocks were the lack of an "updatatable" remote service, and the lack of native support for asynchronous primitives. The Enterprise R6 release will include both RSA 1.1, the Async Service, and OSGi Promises, meaning that further progress is now possible for Distributed Eventing.

The RFC aims to overcome some of the limitations of the existing EventAdmin, particularly when applied to remote systems, and to make use of the advanced features now available within the framework.

2 Application Domain

Distributed systems may be built using a number of different *interaction patterns.* Despite vocal proponents for each approach - it is increasingly clear that no one architectural solution is optimal in every context. Rather there is a continuous spectrum of interaction behaviors. If at all possible – these should ideally be supported in a consistent / coherent manner.

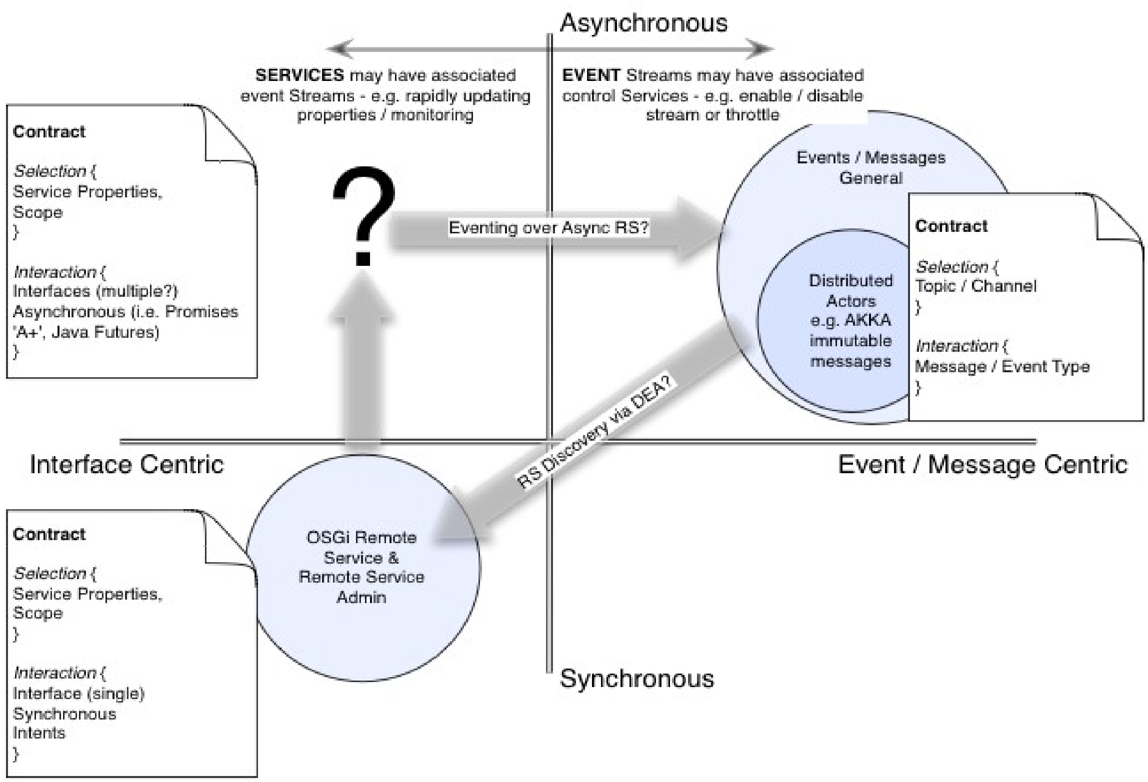

Figure 1: Types of distributed interaction

Synchronous RPC Services: The OSGi Alliance addressed the requirement for Remote Services via the *remote service* and *remote service admin specifications:* these specifications for synchronous RPC Services. In a dynamic environment (1..n) Services may be dynamically discovered, a sub-set of which (1..m where m<n) may be selected for use based on advertised Service properties. Service properties might be *Immutable* e.g. *Version, Location* or *Ownership* information – or *Mutable*: *reliability metrics*, *rating* or *cost metrics* – which may changing frequently in time.

It should be noted that:

- The RSA architecture is modular – allowing different Data Distribution and Discovery Providers to be used. This approach is extremely flexible. Some RSA implementations may choose to use a distributed P2P event substrate to provide Service discovery while other implementations use some form of static look-up service. Which ever is used - a coherent OSGi architecture is maintained. The use of an distributed Event Substrate for Service Discovery is one example of how RS/RSA and Distributed Eventing might interact.

- The current RSA specification does not address Service property updates: properties may change – and one does not necessarily wish to remove and re-add a Service because of this change. In more extreme cases, for volatile Service properties, one may wish to monitor these. Here a reference to the appropriate event streams might be advertised as Service properties. This scenario highlights a second potential relationship between RS/RSA and Distributed Eventing. (Note that it is planned to update the Remote Service Admin specification for Enterprise R6 to support Service property updates, see RFC 203.)

Note the suggested RS/RSA enhancements are out of scope of this RFP – they are mentioned to illustrate the potential relationships between RS/RSA and a Distributed Eventing specification.

Asynchronous Services: Synchronous Services will block until a response is received from the local or remote service. However, on occasions it is preferable for the client to return from the call immediately and continue – some form of notification received at a future point in time – i.e. a 'promise' or a 'future'.

While a number of remote service data distribution providers (RSA) can – in principle support - asynchronous operation, there are currently no OSGi specifications to address support for asynchronous Services (local or remote). Such a specification is desirable as asynchronous Services are increasing popular in Cloud and Distributed environments – and increasingly the JavaScript development community (e.g. node.js). Work in the planned JavaScript RFC will look at implementations of asynchronous Service Registries.

As indicated in Figure 1 – in static environments - Asynchronous Services might be used as a mechanism to implement distributed events. This makes less sense in dynamic environments as some form of discovery mechanism is required – which is usually event based for scaling. So Distributed Eventing would more likely underpin RS/RSA.

This is an important area that requires OSGi Alliance specification work, work that clear relates to both Distributed Eventing & RS / RSA, but Asynchronous Services are out of scope of this current RFP; they are the topic of RFP 132.

Distributed Events / Messages:

Asynchronous Message / Event based approaches are increasingly the underpinnings of large scale distributed environments including 'Cloud'. In these distributed systems *Publishers* endpoints are decoupled from *Subscribers* endpoints; association is achieved via the *Topic* the *Publishers* and *Subscribers* choose to subscribe to a named Topic - and /or – a specific Message type.

Implementations vary considerably – and range from 'classic' enterprise message solutions - (e.g. JMS/AMQP) with centralized message brokers – to peer-to-peer de-centralized P2P solutions – e.g. 0MQ and the Object Management Group's (OMG) Data Distribution Service - see .http://www.omg.org/technology/documents/dds_spec_catalog.html

In principle asynchronous message based system provide the potential for greater scalability. However one cannot naively claim that asynchronous messaging will always scale more effectively than synchronous services: the performance characteristics are implementation dependent. An asynchronous messaging Service implemented via a central broker may introduce significant latency, throughput bottlenecks and points of architectural fragility. Whereas a dynamic Services approach with effective Service endpoint load balancing capabilities – would avoid these issues. However, correctly implemented P2P asynchronous message based systems will out perform both - with lower latency, higher throughput and increased resilience.

Due to the increased level of end-point de-coupling and potentially the use of parallel asynchronous pipelines, interaction contracts within message / event based systems are more challenging. Unlike a Service centric approach - failure of Subscribers is not obvious to Publishers (or visa-versa).

Dependent upon the *Capabilities* of the Distributed Eventing provider – events may / may not be durable – and in-order delivery of message / events may / may not be possible.

- Broker based messaging solutions (i.e. JMS Brokers) typically rely on ACID transactions between publishers and the message broker, then the message broker and subscribers. Such solutions are typically used for chaining interactions between coarse grained services running on large compute servers – i.e. Mainframes / large Unix Systems in traditional Enterprise environments. However centralized brokers / ACID transactions represent bottlenecks and points of fragility: failing to efficiently scale in large distributed highly parallel asynchronous environments.

- Increasingly highly distributed / parallelized environments typical of 'Cloud' are using P2P messaging solutions with compensational / recovery / eventual consistency / based approaches to recovery. In such environments the components with a distributed system need to be idempotent as messages / events / may be re-injected in response to some form of timeout or failure notifications. In such environments aggregation points are still required to coordination at input (fan-out) and output (fan-in) boundaries of the parallel flows.

From a developer perspective '*Actors*' are an increasingly popular asynchronous programming style. While popularised by the the Scala / Akka community – Java Actor frameworks also exist – i.e. the kilim actor framework (http://www.malhar.net/sriram/kilim/) and the NetFlix RXJava project https://github.com/Netflix/RxJava/wiki. In these environments local asynchronous events (locally using a message / mailbox pattern) may be distributed between remote 'Actors' via a plug-able messaging layer; e.g. for Akka 0MQ or via Camel / Camel plug-ins. An OSGi Distributed Eventing specification would provide a natural remoting substrate for '*Actor*' / OSGi based applications.

2.1 Point-to-point/Queue semantics with current Event Admin Service

Some projects use the OSGi Service Registry Hooks to add point-to-point and/or queue messaging semantics to existing Event Admin Service implementations. This approach is working well for these projects and does not actually require a change to the Event Admin Service specification as it uses the hooks to only show the listeners that should receive the message to the Event Admin Service. While not distributed across remote frameworks such a design could also be relevant in a distributed context.

2.2 Existing approaches to distribute the Event Admin Service

A number of projects have successfully implemented a distribution-enabled Event Admin Service employing the existing OSGi API of the Event Admin Service to send events to remote clients. A master thesis was also written on the topic in 2009 by Marc Schaaf [4].

While this approach is very useful in certain situations, it has limitations which make the current Event Admin Service not generally applicable as a service for distributing events.

2.3 Terminology + Abbreviations

Event: a notification that a circumstance or situation has occurred or an incident happened. Events are represented as data that can be stored and forwarded using any mechanism and/or technology and often include information about the time of occurrence, what caused the event and what entity created the event.

Message: a piece of data conveyed via an asynchronous remote communication mechanism such as a message queue product or other middleware. A message can contain an event, but can also have other information or instructions as its payload.

Common definitions for messaging systems include:

Queue: A messaging channel which delivers each message only to one receiver even if multiple receivers are registered, the message will only be delivered to one of them. If a receiver disconnects than all following messages are distributed between the remaining recipients. (It should be configurable that if no recipient is registered when a message is about to be delivered if the message is kept until a receiver is registered or if the message will be lost)

Topic: A publish and subscribe channel that delivers a message to all currently subscribed receivers. Therefore one message is broadcasted to multiple recipients. If no subscription of a receiver is available when a message is about to be delivered, the message is discarded. If a messaging client is disconnected for a period of time, it will miss all messages transferred during this period.

3 Problem Description

The OSGi Alliance has an elegant approach to services & remote-services model via which local services, perhaps expressed by DS & Blueprint, may be simply made visible to clients in remote OSGi frameworks.

However, unlike Remote Services, the OSGi Alliance has no coherent approach to the support of distributed messaging / events. Given the increased mindshare in development communities – this driven by Cloud Computing and use of Actor type patterns, this is an important omission and is the focus of this RFP.

This is doubly surprising given that a number of existing OSGi specifications would benefit from such specifications:

- RSA might leverage a Distributed Eventing implementations for Service Discovery Events – i.e. to Announce/Publish local Service endpoints and Subscribe/Discover Remote Services endpoint events.

- A version of local Event Admin might leverage Distributed Eventing to distribute Events to remote frameworks.

- ConfigAdmin – local ConfigAdmin services might be update by remote Configuration Events.

Hybrid Interactions & Contracts

As previously suggested synchronous Services and asynchronous events represent two ends of a continuous spectrum of interaction behaviors.

Examples include:

- Service announcements and discovery (i.e. Service Events) are already used by RSA – though the implementation is viewed as RSA specific and effectively isolated from the rest of the OSGi framework – unless one treats an entity as a Service. (e.g. example in Cloud EcoSystems – RFC-183).
- We may wish to associate an event channel with an advertised Service – for communicating rapidly changing properties.
- Alternatively we may wish to have a synchronous Service interaction with users of a Topic – perhaps to change the rate/throttle message flow or change recovery behavior.

Defining a generic approach to Contracts or Service Level Agreements is still mostly an area of research – especially for non-functional properties (NFP) – (see SLAng, CQML etc).

However the start of a coherent strategy for OSGi is suggested by P Kriens & BJ Hargrave who have argued that – from a Service perspective – interactions should be expressed in-terms of *contracts*; each participant having its own role with respect to the interaction contract: i.e.

- what the participant is expected to provide
- what the participant can expect from the other participants.

P. Kriens has said: "A contract is just the agreed set of interactions between modules. With Service based interactions one tends to think in terms of interfaces... e.g. the CreditCheck service provides method calculateRating(). This is a simplistic contract between one provider (i.e. the credit rating provider) and the consumers; and it doesn't appear to support asynchronous interaction because invocation always originates from a consumer.

Instead consider a contract based on a group of interfaces; this is somewhat more powerful as each participant can provide some interfaces and consume other interfaces. For example a stock exchange: the exchange itself provides the OrderEntry, and other participants provide ExecutionListeners or MarketDataProviders. Hence, the 'contract' is not with a single Java interface but with a coherent collection of interfaces: in other words a package.

The 'contract' concept - expressed as a data transfer object (DTO) - may span JVM boundaries and provide a consistent approach for; synchronous (simple interface – DTO defines rich set of service properties), asynchronous (DTO defines multiple interfaces) and event based (DTO defines event / message format) based interactions."

While not defining a 'Contract' – DTO's provide generic foundations upon which 'Contract' descriptions may be created.

Distributed Eventing should naturally comply with this philosophy, as for Distributed Eventing the interaction '*contract'* is simply a combination of:

- The structure of the message / event
- The Topic
- The SLA provided by the Distributed Eventing implementation

Data transfer object specification (RFC 185) provides a natural natural representation for the structural payload of a distributed message/event. Meanwhile R5 *Capabilities* provide the natural mechanism via which a user may selected the appropriate Distributed Eventing implementation with respect to required SLA: (in-order delivery, durable or transient etc).

Issues with current Event Admin

It should be noted that the following issues exist with the current Event Admin specification. There is no concept of 'contract' and the messages are untyped, so each participant has to continually work out what kind of message it has received, validate it, handle errors and missing info, work out what it should send in response.

- Current Event Admin only specifies how to send and receive events
- What to do after receiving an Event is unspecified...
- Current Event Admin events are maps, where the values can be anything - Java's instanceof operator to find out the type. Does this / should this / be modernized to be DTO centric?

This is fine if we don't want to go to the trouble of defining a contract for a particular interaction, but the risk is that modules become *more* tightly coupled because of hidden assumptions about the form of events they exchange. Also Event Admin is missing features such as the ability to send a point-to-point reply to a specific message, perhaps to a specific endpoint or subset of endpoints (perhaps via correlation IDs).

For these reasons it may not be possible to repurpose the existing Event Admin since it is already designed for a certain set of local use-cases, and there may be backwards compatibility concerns. Hence a completely new distributed eventing design may be required that might optionally replace or complement the local Event Admin service.

4 Requirements

DE010 – The solution MUST allow the sending of asynchronous messages to remote recipients.

DE012 – The solution MUST support a one-to-many, pub-sub/topic messaging semantic.

DE015 – The solution MUST support a one-to-one, queue messaging semantic.

DE020 – The solution MUST be independent of messaging technology used. This may be message broker based, peer-to-peer using a centralized approach or otherwise.

DE030 – The solution MUST allow implementations to advertise their supported Qualities of Service.

DE040 – The solution MUST provide a mechanism to select an Event Service provider based on its provided QoS.

DE042 – The solution SHOULD define a list of well-known QoS. Implementations MUST NOT be required to support all of these well known QoS.

DE045 – An implementation MUST be allowed to provide additional proprietary Qualities of Service.

DE047 – The solution MUST enable the message sender to specify the actual QoS used for sending a certain message.

DE048 – The solution MUST provide a facility for failure detection and/or reporting in cases where the requested Quality of Service cannot be satisfied.

DE050 – Events / Messages MUST be language agnostic – enabling a remote non-Java party to participate; e.g. C/C++ OSGi based agents.

DE055 – The solution MAY define a standard message encoding, for example using XML, JSON and/or other technology if appropriate.

DE060 – The solution MUST provide the means for point-to-point based communications for example to allow replies to specific messages – an event targeted to a specific node.

DE080 – The solution MUST provide the means to obtain information on the sender of an event e.g. bundleID, Framework UUID, SubSystem name. This information MAY be incomplete if the message didn't originate in an OSGi framework.

DE085 – The solution SHOULD provide the means to discover available Topics and Queues..

DE087 – The solution MUST allow certain Topics and Queues to not be advertised in the discovery mechanism.

DE088 – The solution SHOULD allow certain messages to be hidden from potential receivers.

DE090 – The solution SHOULD NOT prevent an implementation from providing a basic distribution solution for the existing Event Admin. While this will not provide all features of a Distributed Eventing solution, it is shown to be useful in certain contexts.

5 Technical Solution

The existing eventing mechanism in OSGi is the Event Admin service. Event Admin is used to deliver events, either synchronously or asynchronously, to listeners registered in the local framework.

The simplest conceivable solution for distributed eventing involves using the existing Event Admin listener interface, and registering it as a Remote Service. This Remote service will then receive any and all events sent via Event Admin.

To support the point-to-point behaviours required a client could target particular service instances based on their service properties. All remote services are registered with information about the framework they originate from, and their service id within that framework. This allows any remote service to be uniquely identified and selected.

Asynchronous invocation of Event Handlers can be achieved by making use of the new OSGi Async Service. This allows Event Handler services to be called asynchronously, either in a fire-and-forget fashion or receiving a notification on completion.

One possible enhancement to this model would be to add support for the async service and/or Promise pattern to the Event Admin service. The client could potentially provide configuration describing whether redelivery should be attempted, or providing actions to run on failure.

6 Data Transfer Objects

RFC 185 defines Data Transfer Objects as a generic means for management solutions to interact with runtime entities in an OSGi Framework. DTOs provides a common, easily serializable representation of the technology.

For all new functionality added to the OSGi Framework the question should be asked: would this feature benefit from a DTO? The expectation is that in most cases it would.

The DTOs for the design in this RFC should be described here and if there are no DTOs being defined an explanation should be given explaining why this is not applicable in this case.

This section is optional and could also be provided in a separate RFC.

7 Javadoc

Please include Javadoc of any new APIs here, once the design has matured. Instructions on how to export Javadoc for inclusion in the RFC can be found here: https://www.osgi.org/members/RFC/Javadoc

8 Considered Alternatives

For posterity, record the design alternatives that were considered but rejected along with the reason for rejection. This is especially important for external/earlier solutions that were deemed not applicable.

9 Security Considerations

Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.

10 Document Support

10.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3]. The Power of Events'. D. C. Luckham. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[4]. Extending OSGi by Means of Asynchronous Messaging - Master Thesis, Marc Schaaf, September 2009, University of Applied Sciences and Arts Hannover. http://schaaf.es/docs/master_thesis_marc_schaaf_Extending_OSGi_by_Means_of_Asynchronous_Messaging.pdf

10.2 Author's Address

Name	Tim Ward
Company	Paremus
Address	
Voice	
e-mail	tim.ward@paremus.com

Name	Richard Nicholson
Company	Paremus Ltd
Address	107-111 Fleet Street London
Voice	
e-mail	richard.nicholson@paremus.com

Name	Marc Schaaf
Company	
Address	
Voice	
e-mail	marc@marc-schaaf.de

Name	David Bosschaert
Company	Adobe
Address	
Voice	
e-mail	bosschae@adobe.com

Name	Carsten Ziegeler
Company	Adobe
Address	
Voice	
e-mail	cziegele@adobe.com

Name	Graham Charters
Company	IBM
Address	
Voice	
e-mail	charters@uk.ibm.com

10.3 Acronyms and Abbreviations

10.4 End of Document