An Event Mesh for Event Driven IoT Applications

Roberto Berjón*, Montserrat Mateos, M. Encarnación Beato, Ana Fermoso García

Universidad Pontificia de Salamanca, Salamanca (Spain)

Received 21 April 2022 | Accepted 1 July 2022 | Early Access 19 September 2022

Abstract

In IoT contexts, software solutions are required to have components located in different environments: mobile, edge, fog or cloud. To design this type of application, event driven architecture (EDA) is used to develop distributed, scalable, decoupled, desynchronized and real-time components. The interconnection between the different components is done through event brokers that allow communication based on messages (events). Although the design of the components is independent of the environment in which they are deployed, this environment can determine the infrastructure to be used, for example the event brokers, so it is common to have to make modifications to the applications to adapt them to these environments, which complicates their design and maintenance. It is therefore necessary to have an event mesh that allows the connection between event brokers to simplify the development of applications. This paper presents the SCIFI-II system, an event mesh that allows the distribution of events between event brokers. Its use will allow the design of components decoupling them from the event brokers, which will facilitate their deployment in any environment.

Keywords

Cloud Computing, CloudEvents, Edge and Fog Environments, Event Driven Architecture, Event Mesh, Internet of Things.

DOI: 10.9781/ijimai.2022.09.003

I. Introduction

CURRENTLY, all high performance IoT applications, regardless of their computing paradigm: Cloud, Edge, Fog [1], Edge mesh [2] and Cooperative-based systems [3], are developed from an event-driven architecture based on microservices. An IoT application based on microservices is structured through a collection of loosely coupled distributed components that facilitate the scalability and performance of the system. Moreover, the use of event driven architectures allows the real-time processing not only of a data stream coming from different data sources external to the application [4], but also of the data flow exchanged between the different microservices of the application.

There are two key elements to consider in these systems: how to represent the data to be processed and the communication channels through which to transport this data.

The data to be processed is represented in the form of an event. Through it, a series of other elements can be included that can be of great importance during its processing: its source, correlation, transactionality, etc. For this reason, the Cloud Native Computing Foundation (CNCF) promoted the CloudEvents specification [6] for describing event data regardless of the format (json, avro, protocol buffers, xaml) and protocol used for its transport (MQTT, AMQP, Kafka, HTTP, ...), thus guaranteeing its portability and interoperability. A CloudEvent contains two parts: data and metadata.

CloudEvent event data is the data represented through the event. It can be text or binary information. If it is text, the value is included in the "data" attribute of the cloud event. Conversely, if the data is binary, its Base64 encoded value is included in the "data_base64" attribute.

* Corresponding author.

E-mail address: rberjonga@upsa.es

CloudEvent event metadata provides contextual information. It is a set of mandatory and optional attributes in the form of a key value. Table I describes the attributes included in the specification. In addition, if necessary, applications can add new attributes.

TABLE I. CloudEvent Metadata Attributes

Attribute	Description	Category
source	Represents the identifier of the publisher app that broadcast the event. It is expressed as a URI.	Required
id	Event identifier. The sender of the event must ensure that two events from the same source must necessarily have different values for this attribute.	Required
type	A publisher broadcasts different types of events. This attribute (which is a string) indicates the type of event.	Required
subject	Identifies the context in which the source emits the event. Usually, a consumer subscribes to events broadcast by a given source and subject.	Optional
datacontenttype	Represents the content type of data value. It is a string in RFC 2046 format.	Optional
dataschema	It is a URI that identifies the schema in which the data is structured.	Optional
time	Represents the datetime at which the publisher broadcasted the event. RFC 3339 encoded string.	Optional

When integrating distributed systems, it is necessary to use event brokers through which data flows. Therefore, the components sending and receiving data are coupled with respect to the event broker used. As discussed above, one of the premises to be ensured when designing

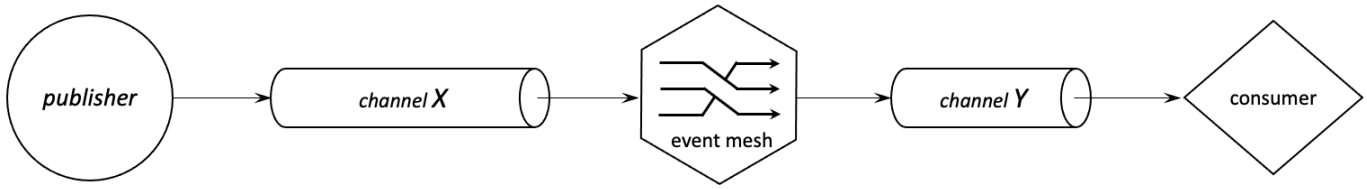

Fig. 1. Event mesh concept.

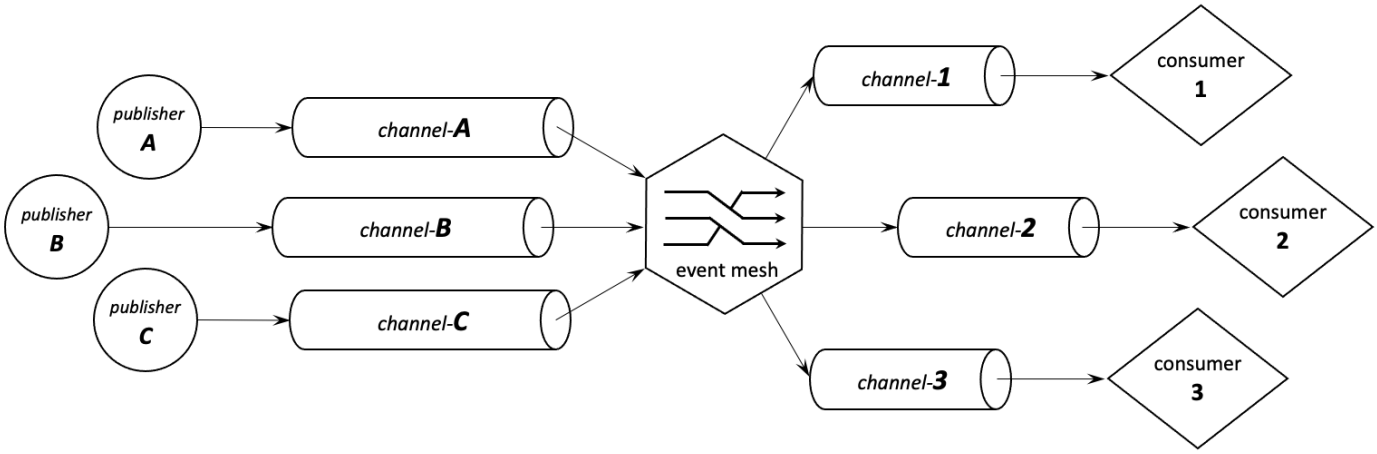

Fig. 2. Actual event mesh.

a distributed system based on microservices is the loosely coupling between its components. To ensure this, a middleware should be added to make the microservices independent of the communication channel through which data is sent and received. The simplest schematic of this type of middleware is shown in Fig. 1. It allows the publisher microservice to send data on the channel-X and the consumer microservice to receive data on another channel-Y. This intermediate layer connecting the event brokers is called event mesh.

An event mesh facilitates the simultaneous connection of different types of event brokers. It therefore acts as an event router. Depending on a set of rules, any event coming from any incoming channel can be forwarded through any of its outgoing channels. This allows great flexibility when integrating applications or extending the functionalities (adding new microservices) of an application. As can be seen in Fig. 2, consumer-1 is designed to receive events from channel-1. Through the event-mesh, channel-1 could receive events coming from channel-A, channel-B and channel-C issued respectively by publisher-A, publisher-B and publisher-C. All this without the need to modify the code of these components.

This is what SCIFI-II focuses on: the design and implementation of an event mesh that is able to connect different types of event brokers. Its mission will be to route incoming events to any of its outgoing channels. The events will be described through the CloudEvent specification. The event mesh will be configured through a dynamic set of rules. These rules will determine the route(s) to be followed by the event (to which outgoing channel to forward the event) based on the event characteristics (mainly described by its metadata).

II. Event Mesh Related Work

The following is a study of the event meshes currently available on the market and their main characteristics. Through this study, the disadvantages of these systems will be analyzed, and SCIFI-II will be presented as an alternative.

Knative Eventing [5]. It is an event mesh solution based on CloudEvents specifications [6]. Events can be redirected to any consumer if it has the ability to recognize and receive events using HTTP. Triggers are used to perform the event subscription, triggers are written using yml files, in which we indicate what values the CloudEvents event attributes should have, as a template.

The solution has two main disadvantages, the first one is that registration can only be done from addressable consumers applications over HTTP and the second is that the way applications show their interest in events is limited to the events meeting some characteristics related to their context properties, but not to the data they might contain.

Solace PubSub+ [7]. this event mesh solution is also an event broker that allows defining queues and hierarchical topics as destinations. Thus, a consumer or producer application can receive and/or send events to queues and hierarchical topics. Its main feature is the ability to register other event brokers to act as consumers or producers of these destinations, for example, an event that is directed to a Solace PubSub+ queue can be received in a MQTT topic.

The main limitation is that it is not an event centric solution, so a consumer is not self-sufficient in determining the type of events it wishes to receive. It is only a solution that connects event brokers to each other using previously configured destinations.

Argo Events [8]. It is a CloudEvents compliant solution. It incorporates a wide variety of “Triggers” (consumers) and “EventSources” (publishers). For the forwarding of events, we define “Sensors” that oversee forwarding the events (triggering of the “triggers”) when certain “dependencies” are met, which are related through “Conditions”. The characteristics of the events are determined by the dependencies from their data and/or their context.

The main limitations of this solution are its complexity in specifying the redirection rules (defining dependencies, creating conditions to relate the dependencies, including conditions within sensors) and the fact that its event sources do not include communication with mobile devices, despite including a greater number of event sources than other solutions analyzed.

TABLE II. Comparative Overview of Event Mesh Platforms

Platform	Event centric	CloudEvent compliant	Redirection based on	Rules format	Channels
Knative	Yes	Yes	Consumer rules	yaml	Based on http
Solace PubSub+	No	No	Static routes	-	Many except specific to mobile devices
Argo Events	Yes	Yes	Consumer rules	yaml	Many except specific to mobile devices
Serverless.com event Gateway	No	No	Topics	-	Based on servless http
Azure Event Grid	Yes	Yes	Azure triggers	code	Platform provided
Amazon EventBridge	Yes	No	Rules based on source and type events	Json	Platform provided
Oracle Events Service	Yes	Yes	Consumer rules	Json	Platform provided

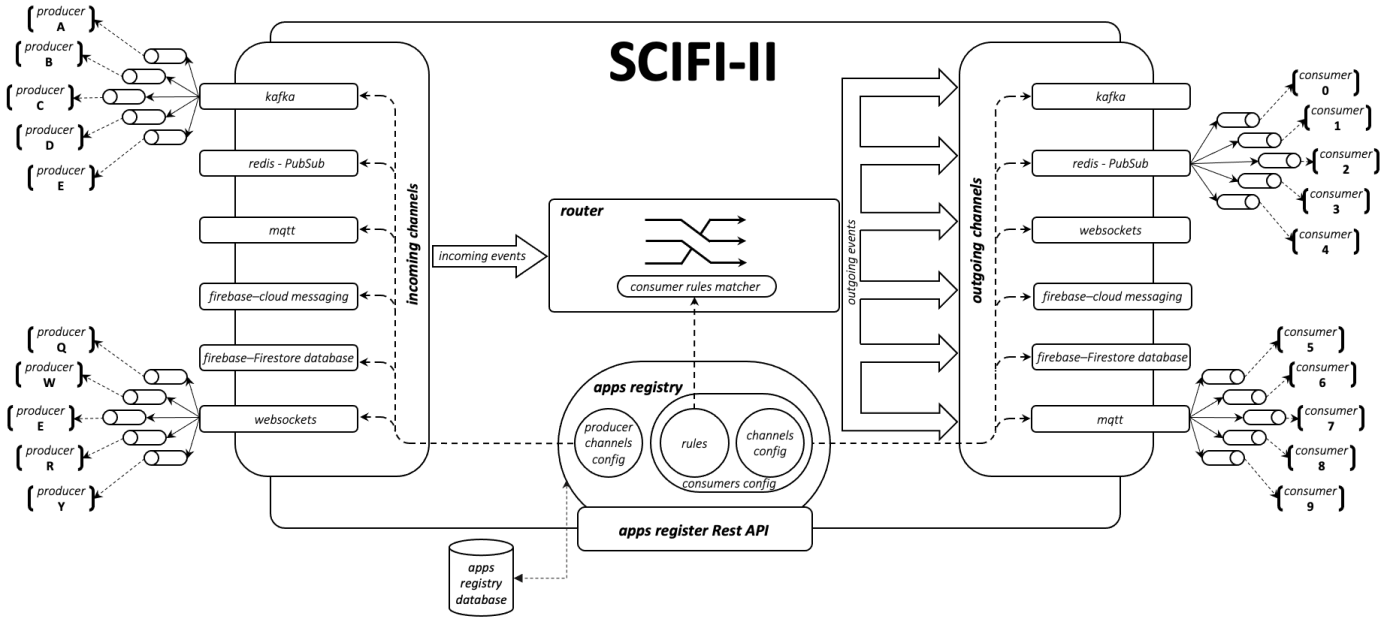

Fig. 3. SCIFI-II components.

Serverless.com event Gateway [9]. It is a serverless gateway that can redirect the events it receives (using HTTP request) to previously registered serverless applications. This solution allows to create sort of topics ("spaces") where event types are created. The way it works is as follows: when registering an application, it is necessary to indicate the event type and the space to which it subscribes, so that the events directed to these coordinates are routed to the registered serverless functions.

This solution is not an event mesh per se, although at the serverless level it can act as one. The main disadvantage found is that it is not an event centric solution since the subscription is made according to the destinations to which it is sent and not according to the characteristics of the events.

Other event mesh solutions provided with cloud platforms include the following: **Azure Event Grid** [10], **Amazon EventBridge** [11] or **Oracle Cloud Events Service** [12]. Their main limitation is that they are only cloud solutions designed to integrate the services of these platforms. Moreover, only Oracle Events Service and Azure Event Grid are CloudEvents compliant.

Table II summarizes the main characteristics of the analyzed platforms. As we can see, only the Argo Events platform is event centric, CloudEvent compliant, in which the redirection of events is done through rules specified by each consumer, and which supports multiple types of channels. Its main disadvantage is the excessive complexity with which these rules are described and the fact that it does not have channels for communication with mobile devices.

The following is a description of our SCIFI-II system that solves and simplifies the described limitations.

III. SCIFI-II

SCIFI-II is a reactive event mesh implemented using the Quarkus framework and therefore it is a cloud native system compatible with Microprofile Reactive Messaging standard specification. It allows dynamic registration of consumer and producer events applications. When registering a producer application, the event broker through which it emits the events must be indicated. As will be discussed later, SCIFI-II supports many types of event brokers. These events must be described using the CloudEvent specification. A consumer application must specify during its registration which events it wishes to receive and through which event broker. In order to specify the events, it wishes to receive, the consumer application must provide a set of rules. These rules must be enforced by incoming events so that they can be forwarded by the event mesh to the consumer. These rules are defined from the properties of the events, mainly from their metadata. SCIFI-II is therefore a CloudEvent compliant and event centric event mesh. A schematic of the different components of SCIFI-II is shown in the Fig. 3.

To dynamically register all applications, SCIFI-II includes a REST api. JSON is the format used to configure the application parameters. The application configuration data is stored in a Google Cloud Firestore database that feeds the apps registry module. This module is responsible for dynamically creating the source and sink connector instances linked to the incoming and outgoing channels of the applications. Additionally, it creates the rules that determine when an event must be redirected to a consumer application. All events received by the incoming channels are processed by the router module. This

module checks for each event the compliance with the rules defined by the applications. If a rule is fulfilled, router redirects the incoming event to the sink connector corresponding to the application that owns the rule.

As discussed above, in their registration, producer and consumer applications (or those playing both roles) must provide different data. Fig. 4 shows the json schema of the document to be provided in the registration of an application.

When a publisher is registered, the *incoming-config* property must indicate the channel on which it broadcasts its events. This attribute contains properties to define the necessary parameters for SCIFI-II to connect to that channel. When a consumer is registered, it is necessary to include two properties: *outgoing-config* and *rules*. Outgoing-config specifies the parameters required for SCIFI-II to connect to the channel to which the consumer is linked. On the other hand, the rules property defines the rules that the events must comply with in order to be routed to the consumer.

```
{
 "$schema": "http://json-schema.org/draft-07/schema",
 "$id": "http://scifi.upsa.es/register.schema.json",
 "type": "object",
 "definitions":   "config": {"type": "object",
                             : : : the object's structure depends of channel
                            }
                }
 "properties": { "id":   {"type": "string"},
                 "name": {"type": "string"},
                 "incoming-config": {"$ref": "#/definitions/config"},
                 "outgoing-config": {"$ref": "#/definitions/config"},
                 "rules": {"type": "array", "items": {"type": "string"}, "minItems": 1}
               },
 "anyOf": [ {"required": ["id", "name", "incoming-config"]},
            {"required": ["id", "name", "outgoing-config", "rules"]}
          ],
 "dependencies": { "outgoing-config": ["rules"],
                   "rules": ["outgoing-config"]
                 },
 "aditionalProperties": true
}
```

Fig. 4. Publisher or Consumer application register schema.

Fig. 5 shows the structure of the JSON documents needed to register both publisher and consumers. Of course, if an application plays both roles, the JSON would be a merge of both. SCIFI-II allows consumer applications to include additional properties which, as will be seen later, can be used in the definition of the rules.

Publisher

```
{
  "id": "id",
  "name": "name",
  "incoming-config": { channel definition }
}
```

Consumer

```
{
  "id": "id",
  "name": "name",
  "outgoing-config": { channel definition },
  "rules": ["rule-1", "rule-2", … "rule-n"]

  : : :  additional properties
}
```

Fig. 5. Publisher and consumer register.

The connectors of the different event-brokers supported by SCIFI-II are presented below.

A. Channels

This section explains the channels currently supported by SCIFI-II. They are classified according to the type of application that uses them. For each channel, the data to be provided in its registration is specified.

- ***sensors/actuators***: In IoT applications, MQTT is the main protocol used [13], in fact this protocol has become the lingua franca in the IoT world. The parameters that need to be included in incoming-config or outgoing-config for this channel are shown in Fig. 6.

```
{
  "type": "object",
  "properties": { "connector":          {"type": "string", "const": "scifi-mqtt"}
                  "mqtt.broker":        {"type": "string", "format": "uri"},
                  "mqtt.client-id":     {"type": "string"},
                  "mqtt.topic":         {"type": "string"},
                  "mqtt.qos":           {"type": "integer", "enum": [0, 1, 2]},
                  "mqtt.clean-session": {"type": "boolean"}
                },
  "required":["mqtt.broker","mqtt.client-id","mqtt.topic","mqtt.qos","mqtt.clean-session"]
}
```

Fig. 6. MQTT channel properties schema.

On the other hand, SCIFI-II also provides a connector for Kafka. Kafka is currently the leading distributed event streaming platform on the market. The configuration parameters are shown in Fig. 7.

```
{
 "type": "object",
 "properties": { "connector":         {"type": "string", "const": "scifi-kafka"}
                 "bootstrap-servers": {"type": "string"},
                 "application-id":    {"type": "string"},
                 "topics":            {"type": "string"},
                 "key":               {"type": "string",
                                       "enum": ["void",  "byte-array", "short",  "integer", "long",
                                                "float", "double",     "string", "uuid"]
               },
 "required":["connector", "bootstrap-servers", "application-id", "topics", "key"]
}
```

Fig. 7. Kafka channel properties schema.

- ***Mobile apps***: Mobile applications are an indispensable component in the development of IoT solutions for Smart cities [14] through the Mobile CrowSensing paradigm. In this sense, SCIFI-II provides connectors for different communication channels specific to mobile technology. One of them is Firebase Cloud Message. Through this channel it is possible to receive and send messages through the XMPP protocol to specific mobile devices (Fig. 8).

```
 "type": "object",
 "properties": {
   "connector":                  {"type": "string", "const": "scifi-firebase"}
   "firebase.type":              {"type": "string", "const": "messaging-xmpp"},
   "firebase.url":               {"type": "string", "format": "uri"},
   "firebase.project-id":        {"type": "string"),
   "firebase.api-key":           {"type": "string"),
   "firebase.credentials.file":  {"$ref": "#/definitions/credentials.file"),
   "firebase.recipientRules":    {"type": "array",
                                  "items": {"type": "object",
                                            "properties": {"test":      {"type": "string"),
                                                           "recipient": {"type": "string"}
                                                          },
                                            "required": ["test", "recipient"]
                                           },
                                  "minitems": 1
                                 },
   "required" : ["connector", "firebase.type", "firebase.credentials.file", "firebase.url",
                 "firebase.project-id", "firebase.api-key"]
```

Fig. 8. Firebase Cloud Message channel properties.

A message could also be sent to a topic. In this case all the mobile devices subscribed to it receive the message. For this use case, the channel could only be of type outgoing-config being their properties those shown in Fig. 9.

```
{
  "type": "object",
  "properties": {
                  "connector":        {"type": "string", "const": "scifi-firebase"}
                  "firebase.type":    {"type": "string", "const": "messaging-sdk"},
                  "firebase.url":     {"type": "string", "format": "uri"},
                  "firebase.credentials.file":   {"$ref": "#/definitions/credentials.file"},
                  "firebase.topic":   {"type": "string"}
                 },
  "required" : ["connector" , "firebase.type", "firebase.credentials.file", "firebase.url",
                "firebase.topic"]
}
```

Fig. 9. Firebase Cloud Message (topic target) outgoing channel properties schema.

Another of the connectors available to SCIFI-II in this area is to Google Cloud Firestore. Firestore is one of the main noSQL databases in the cloud that is used by all types of apps (android, Apple, or web apps). When an event is sent to this channel its event data contains in json format the new data to be added to a collection. The consumers of this channel receive events every time there is a change in the data of a collection (either when it is added, updated, or deleted). Its configuration properties are described in Fig. 10.

```
{
 "type": "object",
 "properties ": {
                  "connector":        {"type": "string", "const": "scifi-firebase"}
                  "firebase.type":    {"type": "string", "const": "database-firestore"},
                  "firebase.url":     {"type": "string", "format": "uri"},
                  "firebase.credentials.file":       {"$ref": "#/definitions/credentials.file"},
                  "firebase.firestore.collection":   {"type": "string"}
                 },
 "required":  ["connector", "firebase.type",  "firebase.credentials.file",  "firebase.url",
               "firebase.topic"]
}
```

Fig. 10. Google Cloud Firestore channel properties schema.

- ***Proxies***: In an application it is common to find components running before or after the data processor that perform data filtering and transformation tasks. Since one of the most used channels in these situations is Redis Pub/Sub, SCIFI-II also has a connector for it (Fig. 11).

```
{
  "type": "object",
  "properties": { "connector":     {"type": "string", "const": "scifi-redis"}
                  "redisauri":     {"type": "string", "format": "uri"},
                  "redis.channel": {"type": "string"},
                 },
  "required":  ["connector", "redisachannel", "redisauri"]
}
```

Fig. 11. Redis Pub/Sub channel properties schema.

- ***Web applications***: A typical use case in this context is the development of web applications to monitor and manage devices in real time. For this type of applications SCIFI-II provides a connector for WebSocket because it allows to establish a full-duplex asynchronous connection without the need for long polling (Fig. 12).

```
{
 "type": "object",
 "properties": {
                  "connector": {"type": "string", "const": "scifi-wsocket"}
                  "uri":       {"type": "string", "format": "uri"}
                },
 "required": ["connector", "uri"]
}
```

Fig. 12. WebSocket channel properties schema.

B. Rules

In SCIFI-II consumer applications must describe which events they wish to receive. To do so, they must indicate what features they should have. In the event these features are generally included in their context, which in the CloudEvents specification represents the event metadata.

During the registration of a consumer application, rules must be specified that incoming events must comply with in order to be forwarded to that consumer. These rules are defined by means of boolean expressions written with the Jakarta Expression Language (EL) syntax. For each incoming event, all the rules defined in the consumer applications are evaluated. The positive evaluation of any rule will result in the redirection of the event to the consumer application. In this way, an incoming event can be redirected to many consumer applications. Likewise, an incoming event may not be redirected to any consumer application.

In these ELs, the context bean "event" can be used to reference through it all the attributes included in the event metadata of the event. For example, the expression "event.type" shall refer to the value of the mandatory type attribute in the CloudEvent specification. For example, the rule in Fig. 13 will determine that the consumer application will be interested in events of type "es.upsa.scifi.dtwins.put" issued by "http://scifi.upsa.es/dtwins".

```
     (event.source eq 'http://scifi.upsa.es/dtwins')
and  (event.type   eq 'es.upsa.scifi.dtwins.put')
```

Fig. 13. Consumer rule with "event" bean context.

In the same way, these ELs can also reference the context bean "self". Through it, all the attributes included in the consumer application registry can be accessed. Remember that the json schema of this registry (see Fig. 4) allows additional attributes to be included as needed. For example, if the "from" attribute had been added to the consumer application record to represent the URI of the publisher app from which it expected to receive events, the rule in Fig. 13 could also be expressed as shown in Fig. 14.

```
{
  "id": "id",
  "name": "name",
  "outgoing-config": { channel definition },
  "rules": ["(event.source eq self.from) and (event.type eq 'es.upsa.scifi.dtwins.put')"],
  "from": "http://scifi.upsa.es/dtwins"
}
```

Fig. 14. Rule with "self" bean context.

The eventDataAsJsonObject() and eventDataAsJsonArray() functions can also be used in the rules. Through these functions it is possible to define rules based on the event data. The former is evaluated as the JSON Object contained in the event data of the event. The function checks that the CloudEvent event contains an event data and its datacontenttype is "application/json". Thus, through the value returned by this function, the properties contained in the event data can be accessed. The second function is like the first one, but in this case, it is evaluated as a json array, so each of its items can be accessed individually. Fig. 15 shows an example where a rule is created based on the "source" and "type" attributes of the CloudEvent context and the "temperate" attribute included in the json object representing the event payload.

```
     (event.source  eq self.from)
and  (event.type    eq 'es.upsa.scifi.dtwins.put')
and  (eventDataAsJsonOb ject() .temperature.doubleValue()   > 14.5)
```

Fig. 15. Consumer rule with eventDataJsonObject() function.

IV. Conclusion

This paper presents the SCIFI-II framework. This framework is an event mesh that allows the connection of multiple event brokers. Its use facilitates the development of distributed applications based on microservices and the integration of heterogeneous applications in a reliable and simple way. It adds a software layer that allows decoupling the event publisher and consumer components of a common event broker. In this way, each component can freely determine the infrastructure that suits it best.

Comparing this framework with others on the market, it is the most versatile as it is the only one with the following characteristics: event centric; CloudEvents compliant; event redirection is based on rules; the rules are represented through Expression Language, a powerful language that simplifies the creation of complex expressions based on the payload and context (or metadata) of the event; it also has a wide catalogue of channels from which to receive or send events, including those oriented towards mobile devices.

The SCIFI-II framework can be used for the development of applications based on event-driven architectures that can be deployed in both cloud and edge environments, as well as in legacy contexts.

References

[1] J. He, J. Wei, K. Chen, Z. Tang, Y. Zhou and Y. Zhang, "Multitier Fog Computing With Large-Scale IoT Data Analytics for Smart Cities," IEEE Internet of Things Journal, vol. 5, no. 2, pp. 677-686, 2018.

[2] Y. Sahni, J. Cao, S. Zhang and L. Yang, "Edge Mesh: A New Paradigm to Enable Distributed Intelligence in Internet of Things," IEEE Access, vol. 5, pp. 16441-16458, 2017.

[3] M. Rescati, M. De Matteis, M. Paganoni, D. Pau, R. Schettini and A. Baschirotto, "Event-driven cooperative-based Internet-of-Things (IoT) system," 2018 International Conference on IC Design Technology (ICICDT), pp. 193-196, 2018.

[4] P. Bellini, D. Nesi, P. Nesi and M. Soderi, "Federation of Smart City Services via APIs," 2020 IEEE International Conference on Smart Computing (SMARTCOMP), pp. 356-361.

[5] Knative, "Knative Eventing,". Accessed: Mar. 06, 2022. [Online]. Available: https://knative.dev/docs/eventing/.

[6] "Cloudevents" Accessed Mar. 06 03 2022 [Online]. Available: https://

cloudevents.io.
[7] Solace, "Solace PubSub+". Accessed Mar. 06 03 2022 [Online]. Available: https://solace.com/.
[8] "Argo Events". Accessed Mar. 06 03 2022 [Online]. Available: https://argoproj.github.io/argo-events.
[9] serverless.com, "Serverless.com - Event Gateway". Accessed Mar. 06 03 2022 [Online]. Available: https://github.com/serverless/event-gateway.
[10] Microsoft Azure, "Azure Event Grid". Accessed Mar. 06 03 2022 [Online]. Available: https://azure.microsoft.com/en-in/services/event-grid.
[11] Amazon Web Services, "Amazon EventBridge". Accessed Mar. 06 03 2022 [Online]. Available: https://aws.amazon.com/eventbridge.
[12] Oracle Corp., "Oracle Cloud Events Service". Accessed Mar. 06 03 2022 [Online]. Available: https://www.oracle.com/cloud-native/events-service.
[13] B. Mishra and A. Kertesz, "The Use of MQTT in M2M and IoT Systems: A Survey," IEEE Access, vol. 8, pp. 201071-201086, 2020
[14] A. Capponi, C. Fiandrino, B. Kantarci, L. Foschini, D. Kliazovich and P. Bouvry, "A Survey on Mobile Crowdsensing Systems: Challenges, Solutions, and Opportunities," IEEE Communications Surveys Tutorials, vol. 21, no. 3, pp. 2419-2465, 2019.

Roberto Berjón

Roberto Berjón received his PhD. in Computer Science from the Universidad de Deusto in 2006. At present he is Professor at the Universidad Pontificia de Salamanca (Spain). He has been a member of the organizing and scientific committee of several international symposiums and has authored papers published in a number of recognized journals, workshops and symposiums. Nowadays he is member of the research group MARATON (Mobile Applications, inteRnet of things, dAta processing, semanTic technologies, OpeN data) where he currently focuses his work on IoT and mobile environments. At present time he is Program Director of the Master in Mobile Applications at the Universidad Pontificia de Salamanca.

Montserrat Mateos

PhD in Computer Science from Universidad de Salamanca in 2006. At present, she is a Professor at the Universidad Pontificia de Salamanca (Spain), and also, she is member of the research group MARATON (Mobile Applications, inteRnet of things, dAta processing, semanTic technologies, OpeN data) where she develops her research works in areas such as mobile technologies, IoT and Information Retrieval. She has been a member of the organizing and scientific committee of several international symposiums and has authored papers published in a number of recognized journals, workshops and symposiums. On other hand, she is external internship coordinator in Faculty of Computer Science.

M. Encarnación Beato

Mª Encarnación Beato (PhD.). Received a PhD. in Computer Science from the University of Valladolid in 2004. She is professor at the Universidad Pontificia de Salamanca (Spain) since 1997. At present she is a member of the MARATON (Mobile Applications, inteRnet of things, dAta processing, semanTic technologies, OpeN data) research group at the Universidad Pontificia de Salamanca. She has been a member of the organizing and scientific committee of several international symposiums and has co-authored papers published in a number of recognized journals, workshops and symposiums.

Ana Fermoso

PhD in Computer Science and Computer Engineering from the University of Deusto. She is currently Professor of Software Engineering at the Faculty of Computer Science of the Universidad Pontificia de Salamanca. She is a member of the MARATON research group, where she works on research lines of this group related to data retrieval, integration and processing, semantic technologies and open data, as well as mobile technologies and IoT. As a researcher, she is author and co-author of numerous scientific publications indexed in the main reference rankings (JCR and SCOPUS), she has participated as a presenter and has been a member of the scientific committee of numerous national and international scientific conferences. She has also participated in competitive research projects as principal investigator and collaborator and from which have derived intellectual property registrations of the software products developed in them. In addition, at present she is the Program Director of the Master's degree in IT project management and technological services. In regards to the topic of the master's degree, she has several certifications in the area. She is PMP (Professional Project Management) certified by the PMI (Project Management Institute), Scrum Master (PSM I) accredited by Scrum.org and by European Scrum, as well as ITIL4 Foundations certification for IT service management.