OFMF Eventing

Michele Gazzetti (IBM Research Europe)

Christian Pinto (IBM Research Europe)

Agenda

1. Overview on Redfish subscriptions and events
2. OFMF/Agent subscription diagram (draft)
3. Gaps and Future steps

Resource Overview

- **Event Service**: contains properties for managing event subscriptions and generates the events sent to subscribers.
  - Has links to the actual collection of subscriptions (**event destinations**).
  - To subscribe, a clients POST to this collection specifying the events of interest
  - Contains attributes such as: status, retry information, etc..
- **Event Destination**: defines the target of an event subscription. This includes:
  - The list of resources or resource types of interest for the subscriber
  - The URI of the destination event receiver.
- **Events**: The message sent on the event destination, which is subscribed to the event. Includes:
  - data about events
  - descriptions
  - severity,
  - message identifier to a message registry that can be accessed for further information
- **Message Registries**: schema describing all possible messages generated. These are
  - Messages are indexed by keys (MessageId)
  - Each message entry describes:
    - severity,
    - Number, type and descriptions of the arguments. (these are variables used when rendering the message)
  - Each message registry has an Owning entity. This is the organization/company that publishes the message registry.

Redfish Eventing: https://www.dmtf.org/sites/default/files/Redfish%20School%20-%20Events_0.pdf
Redfish Resource and Schema guide: https://www.dmtf.org/sites/default/files/standards/documents/DSP2046_2022.2.pdf

Resource Overview

- **Event Service**: contains properties for managing event subscriptions and generates the events sent to subscribers.
  - Has links to the actual collection of subscriptions (**event destinations**).
  - To subscribe, a clients POST to this collection specifying the events of interest s
  - Contains attributes such as: status, retry information, etc..
- **Events**: The message sent on the event destination, which is subscribed to the event. Includes:
  - data about events
  - descriptions
  - severity,
  - message identifier to a message registry that can be accessed for further information
- **Message Registries**: schema describing all possible messages generated. These are
  - Messages are indexed by keys (MessageId)
  - Each message entry describes:
    - severity,
    - Number, type and descriptions of the arguments. (these are variables used when rendering the message)
  - Each message registry has an Owning entity. This is the organization/company that publishes the message registry.
- **Event Destination**: defines the target of an event subscription. This includes:
  - The list of resources or resource types of interest for the subscriber
  - The URI of the destination event receiver.

Redfish Eventing: https://www.dmtf.org/sites/default/files/Redfish%20School%20-%20Events_0.pdf
Redfish Resource and Schema guide: https://www.dmtf.org/sites/default/files/standards/documents/DSP2046_2022.2.pdf

Event


```
{
  "@odata.type": "#Event.v1_7_0.Event",
  "Id": "1",
  "Name": "Event Array",
  "Context": "ContosoWebClient",
  "Events": [ {
      "EventType": "Other",
      "EventId": "4593",
      "Severity": "Warning",
       "Message": "A cable has been removed from network adapter '1' port '1'.",
      "MessageId": "NetworkDevice.1.0.CableRemoved",
      "MessageArgs": [ "1", "1"  ],
      "OriginOfCondition": {
        "@odata.id": "/redfish/v1/Systems/1/EthernetInterfaces/1"
      },
      "LogEntry": {
        "@odata.id": "/redfish/v1/Managers/BMC/LogServices/EventLog/Entries/532"}
  } ]
}
```


Message and related severity level are defined in the Message Registry. MessageArgs are positional argument substituted before the sending of the message

Note: the event can be processed programmatically using **MessageId** and **MessageArgs. But** we can fetch the new resource state by querying the associated Redfish endpoint reported in **OriginOfCondition**

Resource generating the event

Subscribe to resources by type

```
{
  "@odata.context":
  "/redfish/v1/$metadata#EventDestination.EventDestination",
  "@odata.id": "/redfish/v1/EventService/Subscriptions/1",
  "@odata.type": "#EventDestination.v1_0_0.EventDestination",
  "Id": "1",
  "Name": "EventSubscription 1",
  "Destination": https://10.1.1.1:443,
  "Protocol": "Redfish",
  "Context": "Test_Context",
  "ResourceTypes" : [ "ComputerSystem", "Memory",…,]
}
```

Resource types (schema names) that correspond to the OriginofCondition

Subscribe to all subordinate resources

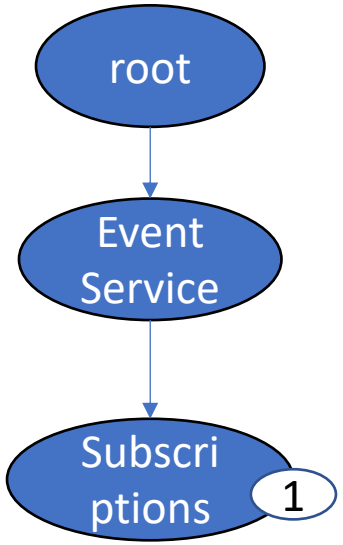
```
{
  "@odata.context":
  "/redfish/v1/$metadata#EventDestination.EventDestination",
  "@odata.id": "/redfish/v1/EventService/Subscriptions/1",
  "@odata.type": "#EventDestination.v1_0_0.EventDestination",
  "Id": "1",
  "Name": "EventSubscription 1",
  "Destination": https://10.1.1.1:443,
  "Protocol": "Redfish",
  "Context": "Test_Context”,
  "OriginResources" : ["/redfish/v1/Fabric/{FabricId}"],
  " SubordinateResources ": true,
}
```

resources for which the service sends only related events

If true, sends events related to OriginResoures and all their subordinates.

Agent’s Subscriptions at boot (step 1)

- Agent starts with pre-existing EventDestination reporting:
  - **Destination**: OFMF Endpoint
  - **ResourceTypes**: [“Fabric”]
- **Result**:
  - The Agent is aware of the OFMF server endpoint
  - At boot the Agent notifies the presence of a new Fabric via an Event
  - At OFMF receives an Event reporting the creation/enablement of a new Fabric
  - The OFMF can scan the Agent’s Redfish tree and collect information on the new resources.

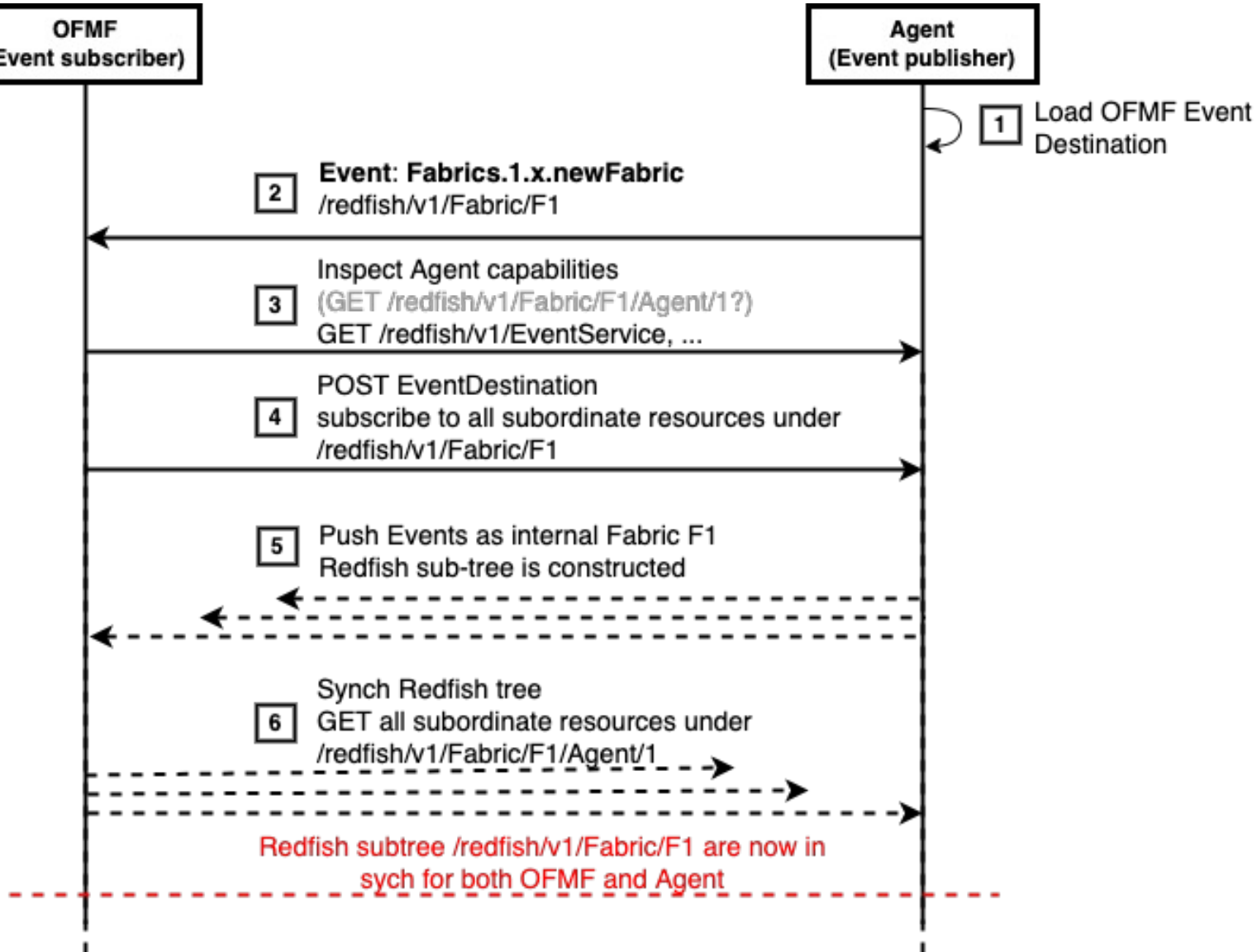



```
"@odata.context":
  "/redfish/….EventDestination",
  "@odata.id":
…/EventService/Subscriptions/1",
  "@odata.type": ….EventDestination",
  "Id": "1",
  "Name": "EventSubscription 1",
  "Destination": https://ofmf.infa:443,
  "Protocol": "Redfish",
  ”ResourceTypes" : [ ”Fabric”]
```

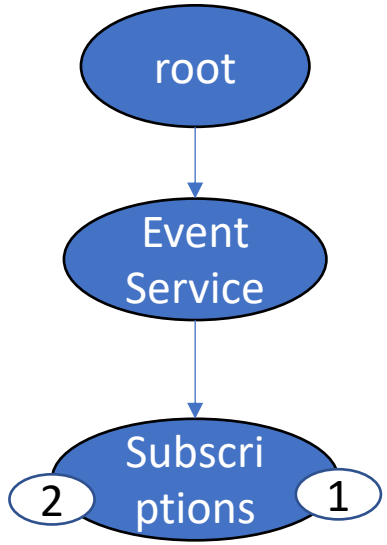
OFMF/Agent subscription diagram

Note: OFMF is a subscriber for this scenario. In some cases, the OFMF can be a publisher of events. For instance, external users can subscribe to the OFMF to receive hw infrastructure related events.

All cases follow the Redfish Eventing specification.



Agent’s Subscriptions at boot (step 2)



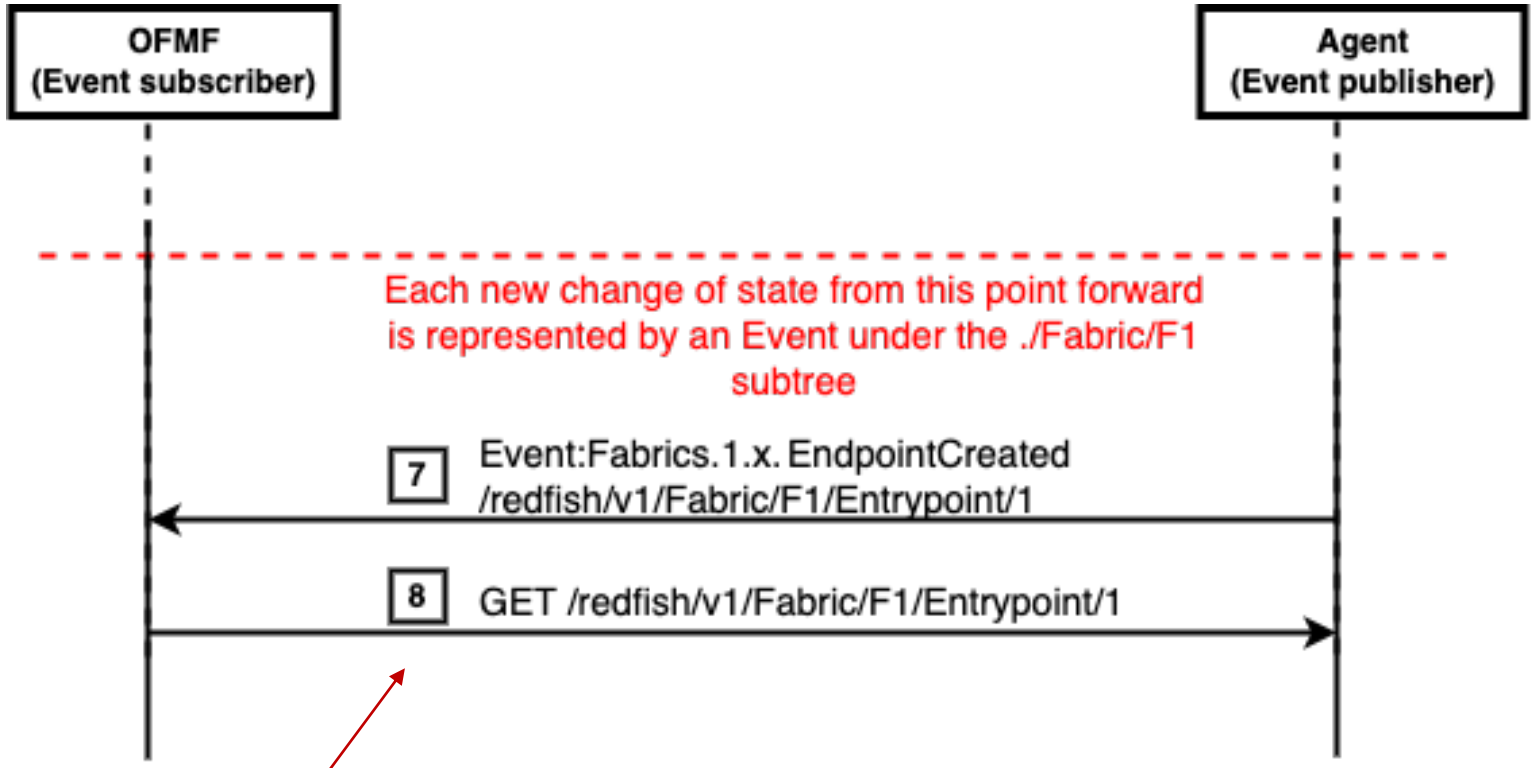

```
"@odata.context":
  "/redfish/….EventDestination",
  "@odata.id":
…/EventService/Subscriptions/1",
  "@odata.type": ….EventDestination",
  "Id": "1",
  "Name": "EventSubscription 1",
  "Destination": https://ofmf.infa:443,
  "Protocol": "Redfish",
  "OriginResources" :
["/redfish/v1/Fabric/F1"],
  " SubordinateResources ": true,
```

Notify the OFMF of any event generated by resources in the Fabric sub-tree

```
"@odata.context":
  "/redfish/….EventDestination",
  "@odata.id":
…/EventService/Subscriptions/1",
  "@odata.type": ….EventDestination",
  "Id": "1",
  "Name": "EventSubscription 1",
  "Destination": https://ofmf.infa:443,
  "Protocol": "Redfish",
  ”ResourceTypes" : [ ”Fabric”]
```

Notify the OFMF of any event related the Fabric objects

Events handling at runtime



Note: step 8 might be necessary (or not) depending on the message comprehensiveness

Gaps and Future steps

1. Formalize events/messages generated during the lifecycle of a Fabric
   - (Current Redfish Message Registry Guide can be found in **DSP2065)**
2. Explore the need to extend/update existing message registries with new entries related to Fabric events
   - Allows us to interpret MessageArgs and retrieve useful metadata information
3. Extend schema with information about the Agent (if necessary).
4. Explore pros/cons of various approaches to asynchronous event handling
   1. Is the current EventService scalable enough?
   2. Do we need third-party solutions? (persistency, scalability, ease of integration, etc..)
      - Message brokers (i.e. RabbitMq)
      - K/V Stores (i.e. Redis,Etcd,…)
      - Etc..

Questions