Events – deep dive

Piotr Kliczewski
Red Hat
23 July 2015

Agenda

- Motivation
- What's new
- Event Flow
- Engine usage
- Vdsm usage
- Where it is used today
- Future plans

Motivation – issues solved

- One side responsible for initiating communication
- Periodic information exchange based on quartz
- High resource utilization
- Increased network traffic

What's new

- Expose communication asynchronicity in the engine
- Json-rpc 2.0 notification format
- Bi-directional data exchange
- Broker “ready” - topology still open – mini broker in use in vdsm
- Implementation of org.reactivestreams in the engine
- Partial contract by using subscription ID

Asynchronous communication

New way of running a command:

```
VDSAsyncReturnValue asyncRetVal =
ResourceManager.getInstance().runAsyncVdsCommand(VDSCommandType.GetImageInfo,
                    new GetImageInfoVDSCommandParameters(storagePoolId,
                            storageDomainId,
                            diskImage.getId(),
                            diskImage.getImageId()));
        if (asyncRetVal != null && asyncRetVal.isRequestCompleted()) {
            Object retVal = asyncRetVal.getReturnValue();
            // process retVal here
        }
```

Complete example: https://gerrit.ovirt.org/#/c/39374

Event format

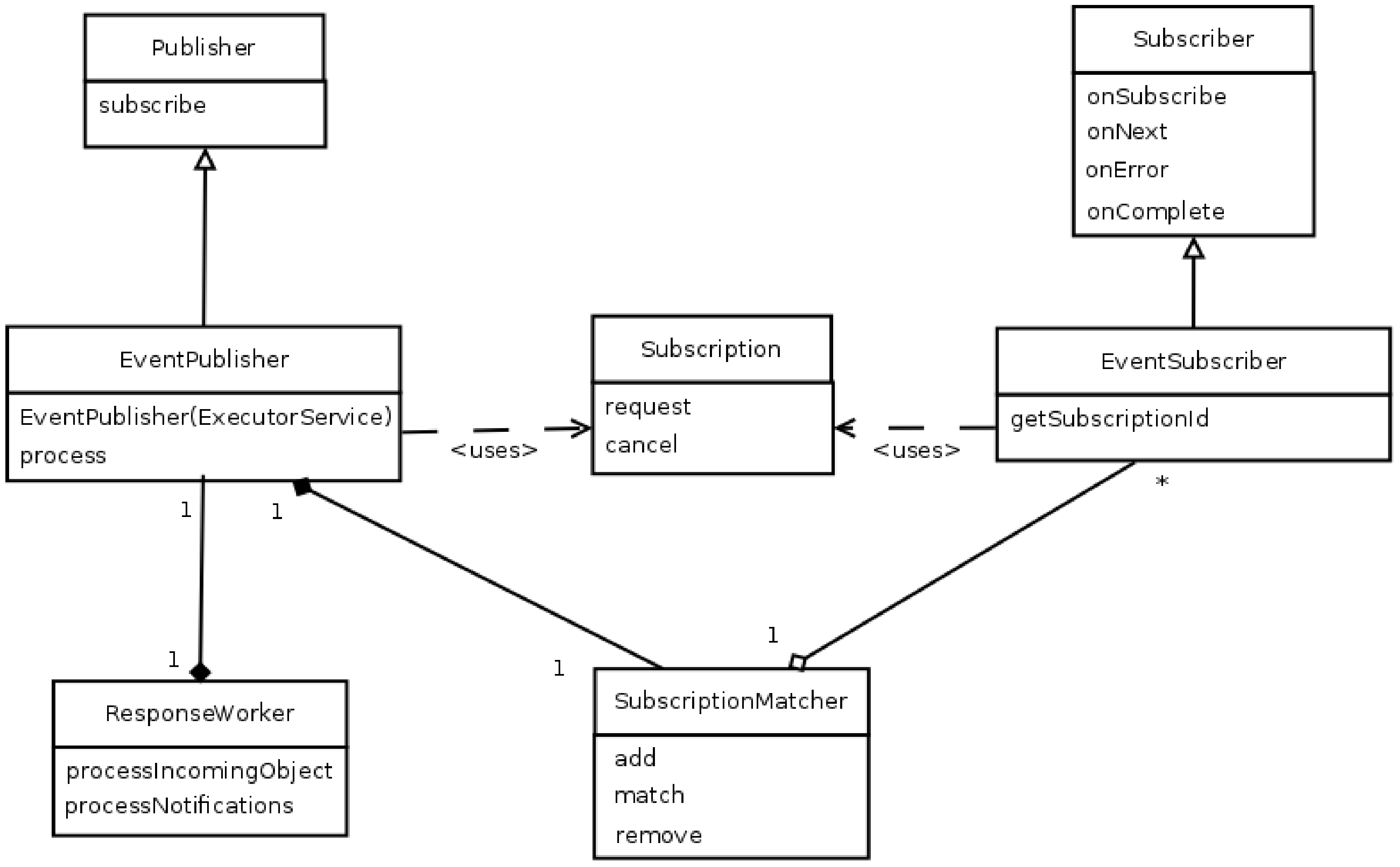
- Based on Notification from jsonrpc 2.0 specification

```
SEND
destination: <queue/topic>
content-type:text/json
content-length: <length>
{
     "jsonrpc": "2.0",
     "method": "<receiver>|<component>|<operation_id>|<unique_id>",
      params": {
       <contents>
    }
}
^@
```

VDSM as a broker

- Legacy mode for 3.5 (based on old queue naming convention)
- Standard mode
- Stomp Broker

ReactiveStreams Implementation

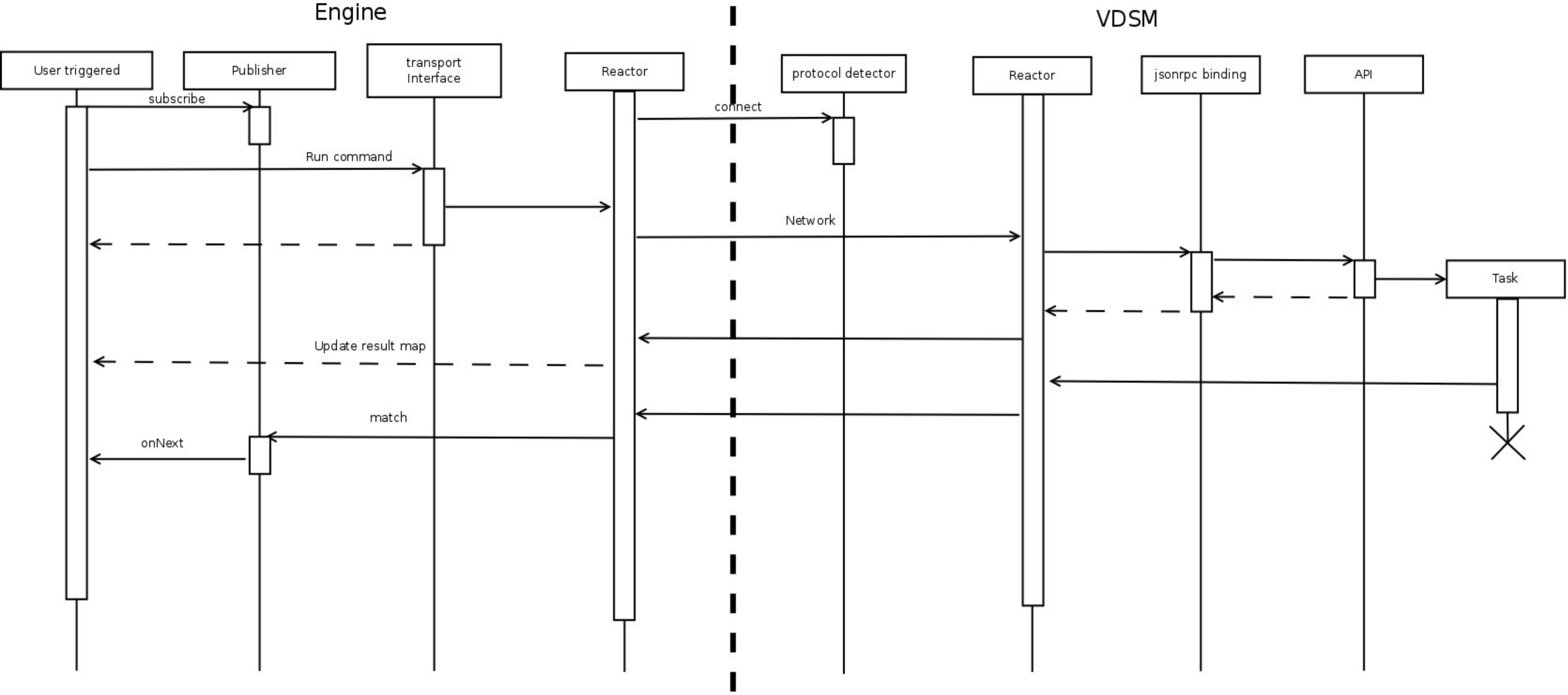


Subscription ID

It is used as contract between vdsm and the engine code to uniquely identify events.

- Receiver - contains a hostname, and it is provided by the client side when an event is received
- Component - contains information about which component generated an event
- Operation id - contains information about the operation, currently mapped to API.py verbs
- Unique id - contains information about the object on which an operation is performed

Event flow


Engine
VDSM
User triggered
Publisher
transport Interface
Reactor
protocol detector
Reactor
jsonrpc binding
API
subscribe
connect
Run command
Network
Task
Update result map
match
onNext

How to receive an event

We need to register our implementation of EventSubscriber

```
        this.resourceManager.subscribe(new EventSubscriber(manager.getVdsHostname() + "|*|VM_status|*") {
            @Override
            public void onSubscribe(Subscription sub) {
                subscription = sub;
                subscription.request(1);
            }
            @Override
            public void onNext(Map<String, Object> map) {
                try {
                    List<Pair<VM, VmInternalData>> changedVms = new ArrayList<>();
                    List<Pair<VM, VmInternalData>> devicesChangedVms = new ArrayList<>();
                    convertEvent(changedVms, devicesChangedVms, map);
                    if (!changedVms.isEmpty() || !devicesChangedVms.isEmpty()) {
                        getVmsMonitoring(changedVms, devicesChangedVms).perform();
                    }
                } finally {
                    subscription.request(1);
                }
            }
            @Override
            public void onError(Throwable t) {
            }
            @Override
            public void onComplete() {
            }
});
```

How to send an event

We need an instance of clientIF and call notify.

```
stats = {}
# collect stats
self._notify('VM_status', stats)

def _notify(self, operation, params):
    sub_id = '|virt|%s|%s' % (operation, self.id)
    self.cif.notify(sub_id, **{self.id: params})
```

Failure cases

- When no matches on the engine an event is dropped
- If no-one is subscribed to jms.queue.events queue no events are sent
- There is no guarantee that an event is delivered so it is important to poll for information after a timeout

Usage in 3.6

- VM monitoring
- DHCP IP assignment (investigated)

VM monitoring (data)

- notify_time – Time when an event as triggered (added by infrastructure)
- status – new vm status
- hash – device hash. Used to understand whether any device has changed
- exit_code, exit_message, exit_reason – additional information for 'Down' status

VM monitoring (gains)

- Reduce polling

  Number of calls for 200 hypervisors

  - 3.5 # requests per minute
    - getAllVMStats (poll) – 800
    - getVMList (poll) - 3200
    - getStats (poll) – depends on # of vm status changes
  - 3.6 # requests per minute
    - getAllVMStats (poll) – 800
    - Vm status event (incoming) – depends on # of vm status changes
- Improve user experience

Future plans

- Back pressure
- Aggregation/throttling
- Schema and versioning
- Widespread use (storage, virt and network)
- Broker

Summary

- Functionality provided as part of event changes
- New architecture of communication layer
- How to send and receive events
- Current usage and future plans

THANK YOU !

pkliczew@redhat.com

@pkliczewski