Use Case 1: VEN query the VTN to determine its capabilities

Use Case Description:
Registration may optionally begin with the VEN querying the VTN to determine what profiles, transports, and extensions it may support using the oadrQueryRegistration payload. The VEN needs to be configured out of band with the address of the VTN in order to initiate the query. The response to the query is the oadrCreatedPartyRegistration payload.

Preconditions:

The VEN has all of the necessary network connections available.
The VEN is configured out of band with the address of the VTN.
The VEN configuration enables a oadrQueryRegistration message.

Postcondition:

Minimal Guarantees:

The VEN does not process any invalid data.

Success Guarantees:

The VEN receives the needed VTN info.

Trigger:
The trigger for this use case if the VEN is switched on.

Main Success Scenario:

1) The VEN sends a oadrQueryRegistration message to the VTN.
2) The VTN receives the messages
3) The VTN transmits the message oadrCreatedPartyRegistration back to the VEN.

Example messages:
oadrQueryRegistration:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrQueryRegistration.xml

oadrCreatedPartyRegistration:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreatedPartyRegistration_Query.xml

Use Case 2: VEN register on the VTN side

Use Case Description:
Registration is always initiated by the VEN with the oadrCreatePartyRegistration payload. This payload provides the information on the profile and the transport method which the VEN has decided to use for communication with the VTN, in addition to other registration related information.
The VTN responds with an oadrCreatedPartyRegistration containing all the profiles and transports supported by the VTN, IDs, and other registration related information. The VTN returns a registrationID in its response payload, which is used for subsequent operations pertaining to this registration instance. This value is used to identify the registration instance for reregistration and cancellation purposes.

Preconditions:

The VEN has all of the necessary network connections available.
VEN configured out of band with the address of the VTN

Postcondition:

Minimal Guarantees:

The VEN does not process any invalid data.

Success Guarantees:

The VEN receives the needed VTN info and a RegistrationID

Trigger:
Use case 1 successful finished.

Main Success Scenario:

1) The VEN sends a oadrCreatePartyRegistration message to the VTN.
2) The VTN receives the messages
3) The VTN transmits the oadrCreatedPartyRegistration message back to the VEN.

Example messages:
oadrCreatePartyRegistration:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreatePartyRegistration_xmpp.xml

oadrCreatedPartyRegistration:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreatedPartyRegistration_xmpp.xml

Use Case 3: VEN re-register on the VTN side due registration information change

Use Case Description:
If the VEN's registration information changes, the VEN can re-register at any time using the oadrCreatePartyRegistration payload referencing the current registrationID. The same registrationID will be maintained across re-registrations until one of the parties cancels the registration.

Preconditions:
The VEN is successfully registered as defined in use case 2.

Postcondition:
Minimal Guarantees:
The VEN does not process any invalid data.
Success Guarantees:
The VTN knows about the changes on VEN side.

Trigger:
If on VEN side one of the following registration information changes:
ei:venID, oadr:oadrProfileName, oadr:oadrTransportName, oadr:oadrTransportAddress, oadr:oadrReportOnly, oadr:oadrXmlSignature,oadr:oadrVenName

Main Success Scenario:

1) The VEN sends an oadrCreatePartyRegistration message which contains the received RegistrationID from the use case 2 to the VTN.
2) The VTN receives the messages
3) The VTN adapts its VEN knowledge according the the oadrCreatePartyRegistration message.
4) The VTN transmits the oadrCreatedPartyRegistration message back to the VEN.

Example messages:
oadrCreatePartyRegistration:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreatePartyRegistration_xmpp.xml

oadrCreatedPartyRegistration:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreatedPartyRegistration_xmpp.xml

Use Case 4: The VTN informs VEN about registration change

Use Case Description:
If the VTN's registration information changes, the VTN can request the VEN to re-register using the oadrRequestReregistration payload. The response to this request is an oadrResponse acknowledgement followed by an asynchronous oadrCreatePartyRegistration request from the VEN.

Preconditions:
The VEN is successfully registered as defined in use case 2.

Postcondition:
Minimal Guarantees:
The VEN does not process any invalid data.
Success Guarantees:
The VEN knows about the changes on VTN side.

Trigger:
On VTN side registration information changes.

Main Success Scenario:

1) The VTN sends an oadrRequestReregistration message to the VEN.
2) The VEN acknowledges the receipt with an oadrResponse message.
3) The VEN transmits an synchronous oadrCreatePartyRegistration request which contains the received RegistrationID from the use case 2.
4) The VTN sends an oadrCreatedPartyRegistration message which contains the new VTN information.

Example messages:
oadrRequestReregistration:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrRequestReregistration.xml

oadrResponse:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrResponse.xml

oadrCreatePartyRegistration:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreatePartyRegistration_xmpp.xml

oadrCreatedPartyRegistration:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreatedPartyRegistration_xmpp.xml

Use Case 5: VEN cancel the registration

Use Case Description:
The VEN cancel an active registration using the oadrCancelPartyRegistration payload, referencing the registrationID. The other party responds with an oadrCanceledPartyRegistration payload.

Preconditions:
The VEN is successfully registered as defined in use case 2.

Postcondition:
Minimal Guarantees:
The VEN does not process any invalid data.
Success Guarantees:
Both sides cancel the active registration

Trigger:
The Colibri Semantic Core gives the command to cancel the registration OR the VEN connector itself wants to cancel the registration.

Main Success Scenario:

1) The VEN sends an oadrCancelPartyRegistration message which contains the received RegistrationID from the use case 2.
2) The VTN acknowledges the cancellation with an oadrCanceledPartyRegistration message.
3) Both sides cancel the active registration.

Example messages:
oadrCancelPartyRegistration:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCancelPartyRegistration.xml

oadrCanceledPartyRegistration:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCanceledPartyRegistration.xml

Use Case 6: VTN cancel the registration

Use Case Description:
The VTN cancels an active registration using the oadrCancelPartyRegistration payload, referencing the registrationID. The other party responds with an oadrCanceledPartyRegistration payload.

Preconditions:
The VEN is successfully registered as defined in use case 2.

Postcondition:
Minimal Guarantees:
The VEN does not process any invalid data.
Success Guarantees:
Both sides cancel the active registration

Trigger:
As soon as the VEN receives an oadrCancelPartyRegistration from the VTN

Main Success Scenario:

1) The VTN sends an oadrCancelPartyRegistration message which contains the received RegistrationID from the use case 2.
2) The VEN acknowledges the cancellation with an oadrCanceledPartyRegistration message.
3) Both sides cancel the active registration.

Example messages:
oadrCancelPartyRegistration:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCancelPartyRegistration.xml

oadrCanceledPartyRegistration:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCanceledPartyRegistration.xml

Use Case 7: The VTN informs the VEN about new electricity prices in a PUSH based manner

Use Case Description:
The VTN informs the VEN about the upcoming electricity prices with an oadrDistributeEvent message. Such price information are covered in events. The oadrDistributeEvent payload contains one or more events described by the oadrEvent element. Some events require a response and others do not as indicated by the oadrResponseRequired element in the event description. As a transport level acknowledgement the VEN responses with an empty iq stanza to oadrDistributeEvent. If a response is required, the VEN acknowledges asynchronously its opt-in or out-out disposition by responding with an oadrCreatedEvent payload containing eventResponse elements matching each oadrEvent. If no response is required, the VEN MUST NOT reply with oadrCreatedEvents (or oadrCreateOpt) payloads for this event. (according to the specification Conformance Rule 12)

Preconditions:
The VEN is successfully registered as defined in use case 2.
Registration in use case 2 defines a XMPP communication

Postcondition:
Minimal Guarantees:
The VEN does not process any invalid data.
Success Guarantees:
The VEN knows the upcoming energy prices

Trigger:
As soon as the VEN receives an oadrDistributeEvent from the VTN

Main Success Scenario:

1) The VTN sends a oadrDistributeEvent message which contains the new energy prices.
2) The VEN acknowledges the receipt with an empty iq stanza message.
3) The VEN indicates its opt-in disposition to the events by responding with an oadrCreatedEvent payload containing eventResponse elements matching each oadrEvent.
4) The VTN acknowledges the receipt with an oadrResponse message.

Example messages:
oadrDistributeEvent:
see Appendix “OadrDistributeEvent message” (without an eiResponse element) and https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrDistributeEvent.xml

oadrCreatedEvent:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreatedEvent.xml

oadrResponse:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrResponse.xml

Use Case 8: The VTN sends the VEN direct load control signals in a PUSH based manner

Use Case Description:
The VEN user accepts a contract to control the load in a certain manner beforehand. The VTN informs the VEN about the upcoming direct load control signals with an oadrDistributeEvent message. Such information are called events. The oadrDistributeEvent payload contains one or more events described by the oadrEvent element. Some events require a response and others do not as indicated by the oadrResponseRequired element in the event description. As a transport level acknowledgement the VEN responses with an empty iq stanza to oadrDistributeEvent. If a response is required, the VEN acknowledges asynchronously its opt-in or out-out disposition by responding with an oadrCreatedEvent payload containing eventResponse elements matching each oadrEvent. If no response is required, the VEN must not reply with oadrCreatedEvents (or oadrCreateOpt) payloads for this event.

Preconditions:
The VEN is successfully registered as defined in use case 2.
Registration in use case 2 defines a XMPP communication

Postcondition:
Minimal Guarantees:
The VEN does not process any invalid data.
Success Guarantees:
The VEN knows how it has to balance the load.

Trigger:
As soon as the VEN receive an oadrDistributeEvent from the VTN

Main Success Scenario:

1) The VTN sends a oadrDistributeEvent message which contains the direct load control signals.
2) The VEN acknowledges the receipt with an empty iq stanza message.
3) The VEN indicates its opt-in disposition to the events by responding with an oadrCreatedEvent payload containing eventResponse elements matching each oadrEvent.
4) The VTN acknowledges the receipt with an oadrResponse message.

Example messages:
oadrDistributeEvent:
see Appendix “OadrDistributeEvent message” (without an eiResponse element) and https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrDistributeEvent.xml

oadrCreatedEvent:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreatedEvent.xml

oadrResponse:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrResponse.xml

Use Case 9: The VTN informs the VEN about new electricity prices in a PULL based manner

Use Case Description:
The XMPP transport uses a PUSH model, although VENs can still make requests with the oadrRequestEvent message. The rest is equal to the use case 7 description

Preconditions:
The VEN is successfully registered as defined in use case 2.
Registration in use case 2 defines a XMPP communication

Postcondition:
Minimal Guarantees:
The VEN does not process any invalid data.
Success Guarantees:
The VEN knows the upcoming energy prices

Trigger:
The Colibri Semantic Core requests the latest event information OR
The VEN Connector itself requests the latest event information to forward it to the Colibri Semantic Core.

Main Success Scenario:

1) The VEN requests the latest event info with an oadrRequestEvent message.
2) The VTN sends an oadrDistributeEvent message which contains the new energy prices.
3) The VEN acknowledges the receipt with an empty iq stanza message.
4) The VEN indicates its opt-in disposition to the events by responding with an oadrCreatedEvent payload containing eventResponse elements matching each oadrEvent.
5) The VTN acknowledges the receipt with an oadrResponse message.

Example messages:
oadrRequestEvent:
see Appendix “OadrRequestEvent message” and
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrRequestEvent.xml

oadrDistributeEvent:
see Appendix “OadrDistributeEvent message” and
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrDistributeEvent.xml

oadrCreatedEvent:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreatedEvent.xml

oadrResponse:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrResponse.xml

Use Case 10: The VEN informs the VTN about its report capabilities

Use Case Description:
This use case describes how the VEN party's reporting capabilities are registered with the VTN party. Report registration is performed after completion of party registration. In addition, any party may send report registrations at any time after the initial registration. The source party sends its reporting capabilities with the oadrRegisterReport message to the target party. The source party's reporting capabilities are specified using a special well-known report profile called the METADATA report, which is exchanged using the same schema as any other report. The target party responds with the oadrRegisteredReport payload. The target party's response may contain an oadrReportRequest object requesting which reports the source party should generate. If there are reports that the target party knows that it wants to receive from the source party then it should make those requests as part of this step. If the target party requests that the source party creates any reports as part of the step then the source party responds with the oadrCreatedReport payload. This message contains an oadrPendingReports element which must contain a list of reportRequestID element values that include all reports that are scheduled for future delivery.
Colibri can report e.g. the current power consumption.

Preconditions:

The VEN is successfully registered as defined in use case 2.

Postcondition:

Minimal Guarantees:

The VEN does not process any invalid data.

Success Guarantees:

The VTN knows the report capabilities from the VEN. e.g. report for the current power consumption.

Trigger:
As soon as the VEN is successfully registered as defined in use case 2.

Main Success Scenario:

1) The VEN sends an oadrRegisterReport message to the VTN.
2) The VTN responses with an oadrRegisteredReport message which contains an oadrReportRequest object requesting that the VEN should generate the report “current power consumption”.
3) The VEN acknowledges the receipt with the oadrCreatedReport message.
4) VTN responds with oadrResponse message

Example messages:
oadrRegisterReport:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrRegisiterReport.xml

oadrRegisteredReport:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrRegisteredReport.xml

oadrCreatedReport:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreatedReport.xml

oadrResponse:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrResponse.xml

Use Case 11: The VTN requests the report “current power consumption”

Use Case Description:
VTN requests reports from the VEN party. Note that any reports that are being requested must correspond to one of the reports that were previously specified in the METADATA report that was previously sent by the VEN party.
The VTN party requests a report from the target party by using the oadrCreateReport payload. That payload contains a set of reportSpecifierID's that correspond to report capabilities in the METADATA report that was previously sent by the target party as part of the previously oadrRegisterReport interaction. As part of the report request the source party specifies a set of reportRequestID's that is used in subsequent operations on this report request.

Preconditions:

The VEN is successfully registered as defined in use case 2.
The VTN successfully receives the METADATA report from the VEN party as defined in use case 10.
The VTN didn't request the report “current power consumption” as defined in use case 10.

Postcondition:

Minimal Guarantees:

The VEN does not process any invalid data.

Success Guarantees:

The VEN cyclically transmits the report “current power consumption” to the VTN side.

Trigger:
As soon as the VEN receives an oadrCreateReport message from the VTN side.

Main Success Scenario:

1) The VEN receives an oadrCreateReport message.
2) The VEN acknowledges the receipt with the oadrCreatedReport message.

Example messages:
oadrCreateReport:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreateReport.xml

oadrCreatedReport:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreatedReport.xml

Use Case 12: VEN transmits the report “current power consumption” to the VTN side.

Use Case Description:
The VEN party sends a report to the VTN party. This operation can be performed by the VEN party only after a previous report request interaction is performed. This operation uses the same oadrReport object as the report registration operation did, but it is used to exchange a report with actual data elements as opposed to the METADATA report used in the registration process. The reports sent in the oadrUpdateReport payload use the EiReport schema with the report-RequestID as defined by the target party in the previous report request interaction that prompted the sending of this report. The response sent by the target party uses the oadrUpdatedReport payload to acknowledge receipt of the report.

Preconditions:

The VEN is successfully registered as defined in use case 2.
The VTN successfully receives the METADATA report from the VEN party as defined in use case 10.
The VTN requests the defined report “current power consumption” as defined in use case 10 or use case 11.

Postcondition:

Minimal Guarantees:

The VEN does not process any invalid data.

Success Guarantees:

The VTN knows the latest “current power consumption” report.

Trigger:
The defined time interval to transmit the report is over.

Main Success Scenario:

1) The VEN transmits an oadrUpdateReport message.
2) The VEN acknowledges the receipt of the report with the oadrUpdatedReport message.

Example messages:
oadrUpdateReport:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrUpdateReport.xml

oadrUpdatedReport:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrUpdatedReport.xml

Use Case 13: VEN transmits the report “current power consumption” to the VTN side and VTN wants to cancel the report

Use Case Description:
The beginning is equal to the use case 12 description. As part of the oadrUpdatedReport response the target party cancels the sending of any future reports using the optional oadrCancelReport object, which contains a list of reportRequestIDs. There is no confirmation of the cancellation if included in the oadrUpdatedReport payload, but if the report continues to be sent, the receiving party should use oadrCancelReport to cancel the report. In the case of periodic report, if reportToFollow is set to true in the oadrCancelReport object by the target party, the source party is expected to send one final additional report.

Preconditions:

The VEN is successfully registered as defined in use case 2.
The VTN successfully receives the METADATA report from the VEN party as defined in use case 10.
The VTN requests the defined report “current power consumption” as defined in use case 10 or use case 11.

Postcondition:

Minimal Guarantees:

The VEN does not process any invalid data.

Success Guarantees:

The VTN knows the latest “current power consumption” report.
Depending on the reportToFollow: true: one last report will be transmitted,
false: no further report will be transmitted

Trigger:
The defined time interval to transmit the report is over.

Main Success Scenario:

1) The VEN transmits an oadrUpdateReport message.
2) The VEN acknowledges the receipt of the report with the oadrUpdatedReport message which contains the element oadrCancelReport to cancel the the report “current power consumption”

Example messages:
oadrUpdateReport:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrUpdateReport.xml

oadrUpdatedReport:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrUpdatedReport_W_Cancel.xml

Use Case 14: VTN cancels the report “current power consumption”

Use Case Description:
This interaction is used by the VTN party to cancel ongoing report that are being generated by the VEN party. The source party uses the oadrCancelReport payload with the appropriate reportRequestID that was specified by the VEN party in a previous request report interaction. Upon receiving the oadrCancelReport payload the VEN party stops generating and sending the reports corresponding to the reportRequestID.
The response to the oadrCancelReport payload is the oadrCanceledReport payload that is sent by the VTN party to acknowledge the canceling of the report.
Note that both oadrCanceledReport and oadrCreatedReport return a list of pending reports in the oadrPendingReports object, which includes all reports that are scheduled for future delivery. When cancellation of periodic report is requested by the VEN party, if reportToFollow is set to true in oadrCancelReport, the source party is expected to send one final additional report.

Preconditions:

The VEN is successfully registered as defined in use case 2.
The VTN successfully receives the METADATA report from the VEN party as defined in use case 10.
The VTN requests the defined report “current power consumption” as defined in use case 10 or use case 11.

Postcondition:

Minimal Guarantees:

The VEN does not process any invalid data.

Success Guarantees:

Depending on the reportToFollow: true: one last report will be transmitted,
false: no further report will be transmitted

Trigger:
As soon as the VEN receives an oadrCancelReport message from the VTN side.

Main Success Scenario:

1) The VEN receives an oadrCancelReport message.
2) The VEN acknowledges the receipt with an oadrCanceledReport message.

Example messages:
oadrCancelReport:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCancelReport.xml

oadrCanceledReport:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCanceledReport.xml

Use Case 15: VEN is temporary unavailable to comply to a specific event.

Use Case Description:
The oadrCreateOpt message is used by VEN to set temporary availability schedules that differ from the default. The element eiTarget is used to specify to a specific target. In our use case this is the VEN defined by its venID. If only the venID is specified in eiTarget, then the opt schedule applies to all of the resources associated with that VEN. The eventID element specifies the event which the VEN can not apply. oadrCreateOpt payload includes an optID, which can be used in subsequence operations to reference this schedule. The VTNs response to oadrCreateOpt is an oadrCreatedOpt payload that includes an optID, which can be used in subsequence operations to reference this schedule.

The reason is defined in the optReason element. There are some predfined see oadr_ei_20b.xsd beginning at line 604 and if nessasary there is the possibility to define new ones with a “x-” prefix. This example use the reason “notParticipating”.

Preconditions:
The VEN is successfully registered as defined in use case 2.

Postcondition:
Minimal Guarantees:
The VEN does not process any invalid data.
Success Guarantees:
The VTN knows that the VEN is unavailable for a specific event.

Trigger:
As soon as the VEN connector recognizes that it is temporary unavailable for an event.

Main Success Scenario:

1) The VEN transmits an oadrCreateOpt message which contains only the venID in the eiTarget element and the eventID element contains the event identifier.
2) The VTN acknowledges the receipt with an oadrCreatedOpt message.

Example messages:
oadrCreateOpt:
see Appendix “OadrCreateOpt event message” and
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreateOpt_Schedule.xml

oadrCreatedOpt:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCreatedOpt.xml

Use Case 16: VEN is available to comply to all contracts again.

Use Case Description:
The VEN cancels a temporary availability schedule by using the oadrCancelOpt with an optID referencing the schedule to be canceled

Preconditions:
The VEN is successfully registered as defined in use case 2.
The VTN is informed about the VEN's unavailability as defined in use case 15

Postcondition:
Minimal Guarantees:
The VEN does not process any invalid data.
Success Guarantees:
The VTN knows that the VEN is available for all previous eiTargets.

Trigger:
As soon as the VEN recognizes that it is available for all eiTargets again.

Main Success Scenario:

1) The VEN transmits an oadrCancelOpt message.
2) The VTN acknowledges the receipt with an oadrCanceledOpt message.

Example messages:
oadrCancelOpt:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCancelOpt.xml

oadrCanceledOpt:
https://github.com/EnerNOC/oadr2-ven-python/blob/master/test/xml_files/2.0b_spec/Sample_oadrCanceledOpt.xml

Appendix:

OadrRequestEvent message:

```
<?xml version="1.0" encoding="UTF-8"?>
<oadr:oadrRequestEvent xmlns:oadr="http://openadr.org/oadr-2.0a/2012/07"
xmlns:ei="http://docs.oasis-open.org/ns/energyinterop/201110"
xmlns:emix="http://docs.oasis-open.org/ns/emix/2011/06" xmlns:pyld="http://docs.oasis-
open.org/ns/energyinterop/201110/payloads" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
   <pyld:eiRequestEvent>
      <pyld:requestID />
      <ei:eventID />
      <emix:marketContext>http://crunchGeorg:8080/oadr2-vtn-
groovy/program/c</emix:marketContext>
      <ei:venID>colibriVen</ei:venID>
      <pyld:eventFilter>all</pyld:eventFilter>
      <pyld:replyLimit>0</pyld:replyLimit>
   </pyld:eiRequestEvent>
</oadr:oadrRequestEvent>
```

OadrDistributeEvent message:

```
<?xml version="1.0" encoding="UTF-8"?>
<oadrDistributeEvent xmlns="http://openadr.org/oadr-2.0a/2012/07"
xmlns:ei="http://docs.oasis-open.org/ns/energyinterop/201110"
xmlns:emix="http://docs.oasis-open.org/ns/emix/2011/06"
xmlns:ical="urn:ietf:params:xml:ns:icalendar-2.0" xmlns:pyld="http://docs.oasis-
open.org/ns/energyinterop/201110/payloads" xmlns:strm="urn:ietf:params:xml:ns:icalendar-
2.0:stream" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <!-- VTN response to the VEN oadrRequestEvent message -->
   <ei:eiResponse>
      <!-- code 200 means everything is okay -->
      <ei:responseCode>200</ei:responseCode>
      <pyld:requestID>4f030fa7-ba42-4a68-a4e1-66242e1e88c9</pyld:requestID>
   </ei:eiResponse>
   <pyld:requestID>f0546dd0-37ac-4182-bf7a-1c09a1640ba9</pyld:requestID>
   <ei:vtnID>vtnID</ei:vtnID>
   <oadrEvent>
      <!-- defines an event ... -->
      <ei:eiEvent>
         <ei:eventDescriptor>
            <!-- … with this id ... -->
            <ei:eventID>283d9ad0-81bc-48e0-af42-198c519ae434</ei:eventID>
            <ei:modificationNumber>1</ei:modificationNumber>
            <ei:priority>3</ei:priority>
            <ei:eiMarketContext>
               <!-- This identifies a particular program which belongs to an event. -->
               <emix:marketContext>http://crunchGeorg:8080/oadr2-vtn-
groovy/program/c</emix:marketContext>
            </ei:eiMarketContext>
            <!-- The time when the payload which contains the event was created. -->
            <ei:createdDateTime>2016-06-01T06:58:41.784Z</ei:createdDateTime>
            <!-- The status of the event, indicating if the event is “near”, “far”,
                 “active” or “cancelled”. -->
            <ei:eventStatus>near</ei:eventStatus>
            <ei:testEvent>False</ei:testEvent>
            <ei:vtnComment />
         </ei:eventDescriptor>
         <ei:eiActivePeriod>
            <ical:properties>
               <ical:dtstart>
                  <!-- which starts at 8:00 on the 2016-06-01 UTC time (+2 for CEST) -->
                  <ical:date-time>2016-06-01T08:00:00.000Z</ical:date-time>
               </ical:dtstart>
               <ical:duration>
                  <!-- The duration is two hours-->
                  <ical:duration>P0Y0M0DT2H0M0S</ical:duration>
               </ical:duration>
               <ical:tolerance>
```

```
                  <ical:tolerate>
                     <!-- Event start times can be randomized using the tolerance object
                          in the eiActivePeriod. The subelement startafter defines a
                          randomization time window used by the VEN to select a random
                          value that is added to the start time of the event. If the
                          start time of a one hour event is 3:00pm and the randomization
                          window is 5 minutes, if the VEN selected 3 minutes as the
                          random value then the event would start at 3:03pm and would
                          end at 4:03pm
                     -->
                     <ical:startafter>P0Y0M0DT0H0M0S</ical:startafter>
                  </ical:tolerate>
               </ical:tolerance>
               <ei:x-eiNotification>
                  <ical:duration>P0Y0M0DT0H0M0S</ical:duration>
               </ei:x-eiNotification>
               <!-- Possibility to define different phases of an event -->
               <ei:x-eiRampUp>
                  <ical:duration>P0Y0M0DT0H0M0S</ical:duration>
               </ei:x-eiRampUp>
               <ei:x-eiRecovery>
                  <ical:duration>P0Y0M0DT0H0M0S</ical:duration>
               </ei:x-eiRecovery>
            </ical:properties>
            <ical:components xsi:nil="true" />
         </ei:eiActivePeriod>
         <!-- Events are divided into event signals -->
         <ei:eiEventSignals>
            <!-- This event has only one signal ... -->
            <ei:eiEventSignal>
               <strm:intervals>
                  <ei:interval>
                     <ical:duration>
                        <!-- which lasts for the whole duration 2h. -->
                        <ical:duration>P0Y0M0DT2H0M0S</ical:duration>
                     </ical:duration>
                     <ical:uid>
                        <ical:text>1b6597f0-4148-4cbd-8c87-93595068cace</ical:text>
                     </ical:uid>
                     <ei:signalPayload>
                        <ei:payloadFloat>
                           <!-- During this time the energy price
                                raises about 42 currency/kWh -->
                           <ei:value>42.0</ei:value>
                        </ei:payloadFloat>
                     </ei:signalPayload>
                  </ei:interval>
               </strm:intervals>
               <ei:signalName>simple</ei:signalName>
               <!-- This is a delta change to the existing price of electricity -->
               <ei:signalType>priceRelative</ei:signalType>
               <ei:signalID>71fea88d-1b4c-4373-84fc-96aa1f8de0cd</ei:signalID>
               <!-- The event currentValue element is optional for the B service,
                 it is the payloadFloat value of the event interval
                 currently executing -->
               <ei:currentValue>
                  <ei:payloadFloat>
                     <ei:value>0.0</ei:value>
                  </ei:payloadFloat>
               </ei:currentValue>
            </ei:eiEventSignal>
         </ei:eiEventSignals>
         <ei:eiTarget />
      </ei:eiEvent>
      <!-- Does the VEN have to response with an oadrCreatedEvent?
           never=No, always=yes -->
      <oadrResponseRequired>always</oadrResponseRequired>
   </oadrEvent>
</oadrDistributeEvent>
```

OadrCreateOpt event message:

```
<?xml version="1.0" encoding="UTF-8"?>
<oadr:oadrCreateOpt ei:schemaVersion="2.0b" xsi:schemaLocation="http://openadr.org/oadr-
2.0b/2012/07 oadr_20b.xsd" xmlns:emix="http://docs.oasis-open.org/ns/emix/2011/06"
xmlns:power="http://docs.oasis-open.org/ns/emix/2011/06/power"
xmlns:ei="http://docs.oasis-open.org/ns/energyinterop/201110"
xmlns:pyld="http://docs.oasis-open.org/ns/energyinterop/201110/payloads"
xmlns:oadr="http://openadr.org/oadr-2.0b/2012/07"
xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xcal="urn:ietf:params:xml:ns:icalendar-2.0">
	<ei:optID>OPT_1234</ei:optID>
	<!-- The optType may have a value of “optIn” or “optOut” to indicate the VENs
		disposition for a given event.-->
	<ei:optType>optOut</ei:optType>
	<ei:optReason>x-schedule</ei:optReason>
	<emix:marketContext>http://www.drprogram.com</emix:marketContext>
	<ei:venID>VEN_1234</ei:venID>
	<ei:qualifiedEventID>
		<ei:eventID>event1</ei:eventID>
		<ei:modificationNumber>3</ei:modificationNumber>
	</ei:qualifiedEventID>
	<ei:createdDateTime>2001-12-17T09:30:47Z</ei:createdDateTime>
	<!-- Although requestID is a mandatory payload element for oadrCreateOpt and
		oadrCancelOpt, it MAY be left as an empty string. However, if a value is
		specified for requestID, the VTN MUST return that value in its
		oadrCreatedOpt or oadrCanceledOpt payload. -->
	<pyld:requestID>REQ_12345</pyld:requestID>
	<ei:eiTarget>
		<ei:venID>colibriVEN</ei:venID>
	</ei:eiTarget>
</oadr:oadrCreateOpt>
```