Design

Examples, proposals and the final design to solve the challenge

- First proposal

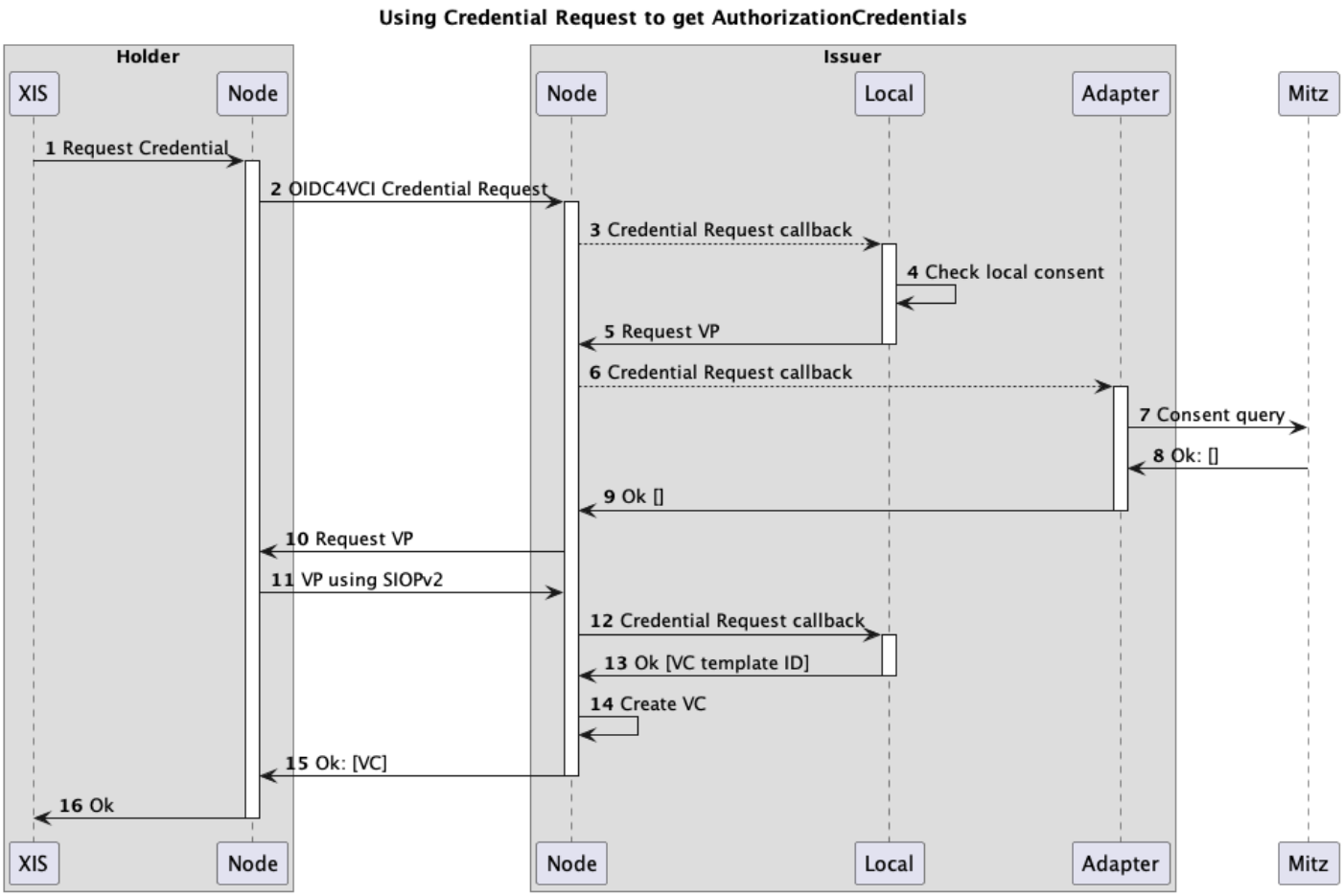
First proposal

This page contains a proposal that could solve the problem. It does not have all the details worked out yet.

Using credential request

The credential request flow is part of the OIDC4VCI specification and will be implemented by the Nuts node. It's also used in some of the other working groups.

The main idea is that when a care organization already knows care is given by another care organization, it can ask that care organization for an authorization. The *other* care organization acts as an credential issuer and will need to respond to the request. The answer to the request depends on any available legal basis and if the requester submitted the correct data.



The flow below shows a basic example.

1. A XIS/ECD sends a credential requests to its Nuts node. The request could look like this for eOverdracht:

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://nuts.nl/credentials/v1"
  ],
  "types": [
    "VerifiableCredential",
    "NutsAuthorizationCredential"
  ],
  "issuer": "did:nuts:123",
  "credentialSubject": {
    "id": "did:nuts:456",
    "purposeOfUse": "eOverdracht",
    "case#": "10475098459138475"
  }
}
```

or this for same organization access:

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://nuts.nl/credentials/v1"
  ],
  "types": [
    "VerifiableCredential",
    "NutsAuthorizationCredential"
  ],
  "issuer": "did:nuts:123",
  "credentialSubject": {
    "id": "did:nuts:456",
    "purposeOfUse": "same-org",
    "patient_identifier": "123456782"
  }
}
```

2. The node will find the correct endpoint and forwards the request to another node. Nodes can act as both a wallet and issuer. The node will embed the request in the

`authorization_details` field:

```
[{
  "type":"openid_credential",
  "format": "ldp_vc",
  "credential_definition": {
    "@context": [
      "https://www.w3.org/2018/credentials/v1",
      "https://nuts.nl/credentials/v1"
    ],
    "types": [
      "VerifiableCredential",
      "NutsAuthorizationCredential"
    ],
    "issuer": "did:nuts:123",
    "credentialSubject": {
      "id": "did:nuts:456",
      "purposeOfUse": "eOverdracht",
      "case#": "10475098459138475"
    }
  }
}]
```

3. The received request will be forwarded to any configured *adapter* using the same format as seen in step 2. Multiple adapters can be contacted in this way.
4. The adapter will apply business logic to determine the correct answer. In this example the local consent registry only stores consent based on logical organization identifiers.
5. Due to missing context data, the local adapter will respond with a *400 - not enough data* and a `presentation_definition`:

```
{
 "presentation_definition": {
  "id": "first simple example",
  "input_descriptors": [
   {
    "id": "A specific type of VC",
    "name": "A specific type of VC",
    "purpose": "We want a VC of this type",
    "constraints": {
     "fields": [
      {
```

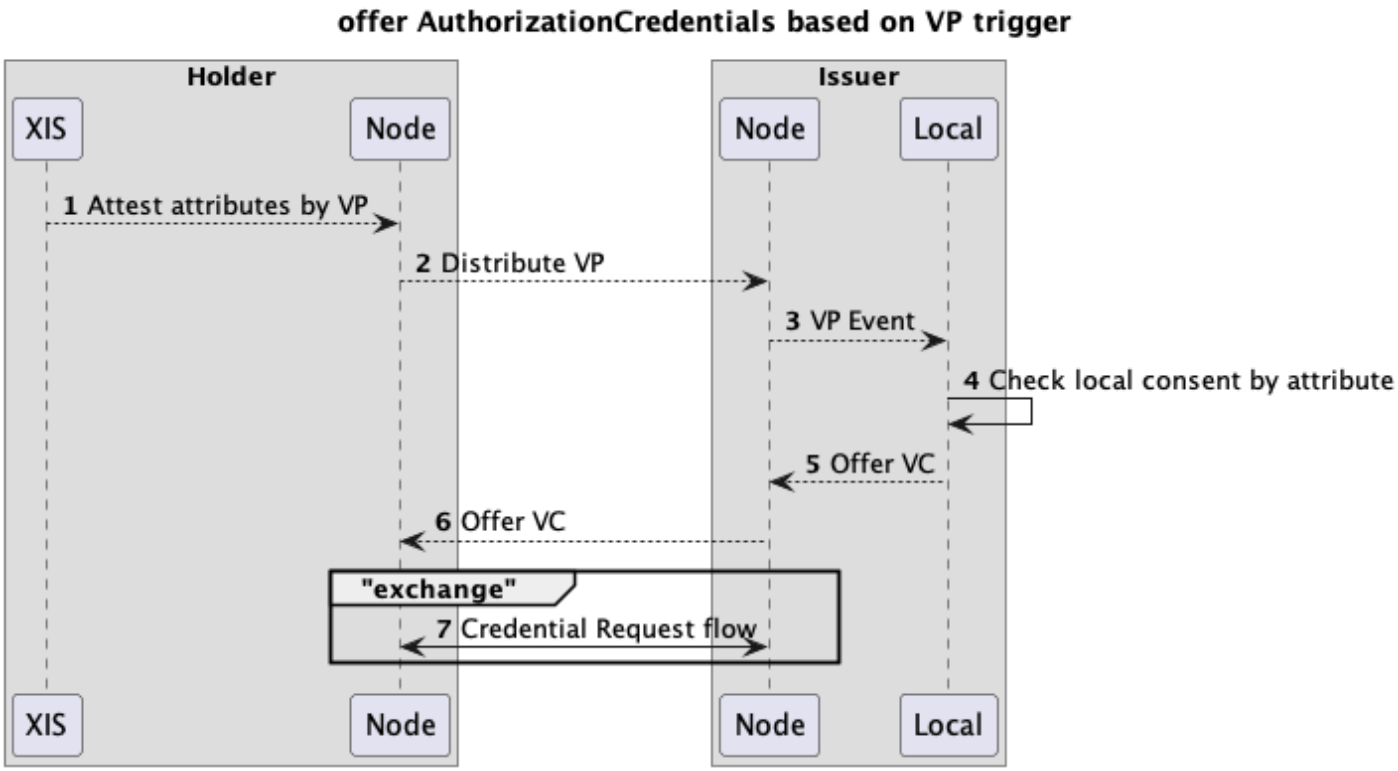
```
            "path": [
              "$.type"
            ],
            "filter": {
              "type": "string",
              "pattern": "NutsOrganizationCredential"
            }
          },
          {
□□□ "path": [
□□□□"$.organization.identifier"
□□□ ],
□□□ "filter": {
□□□□"type": "object",
□□□□"minProperties": 1
□□□ }
□□□},
        ]
      }
    }
  ]
 }
}
```

The presentation definition tells the issuer node it should acquire a verifiable presentation containing a *NutsOrganizationCredential* 6. same as step 3 7. The Mitz adapter contacts Mitz using a FHIR consent query. 8. Nothing registered, empty response. 9. This adapter returns an empty result. No rules to issue a credential on. 10. The presentation definition from the local adapter is used to perform acquire a `Verifiable Presentation` from the requestor using `SIOPc2`. 11. If the requestor holds a credential that matches the presentation definition, it can respond with a verifiable presentation. 12. The issuer node submits the request again, but now with a verifiable presentation in the `vp_token` field. The request now fulfils the requirements of the local adapter. 13. The adapter responds with a template of a credential that may be issued by the issuer node. 14. A credential is created, it's identifier stored. 15. The credential is returned to the wallet of the holder. 16. complete

Using credential/presentation trigger

In the case the holder does not know where to get an authorization, the issuer could send a *credential offer* to the holder based on information the holder publishes. For example: a newly published *NutsOrganizationCredential* could trigger an issuer to offer a credential based on logical identifier.

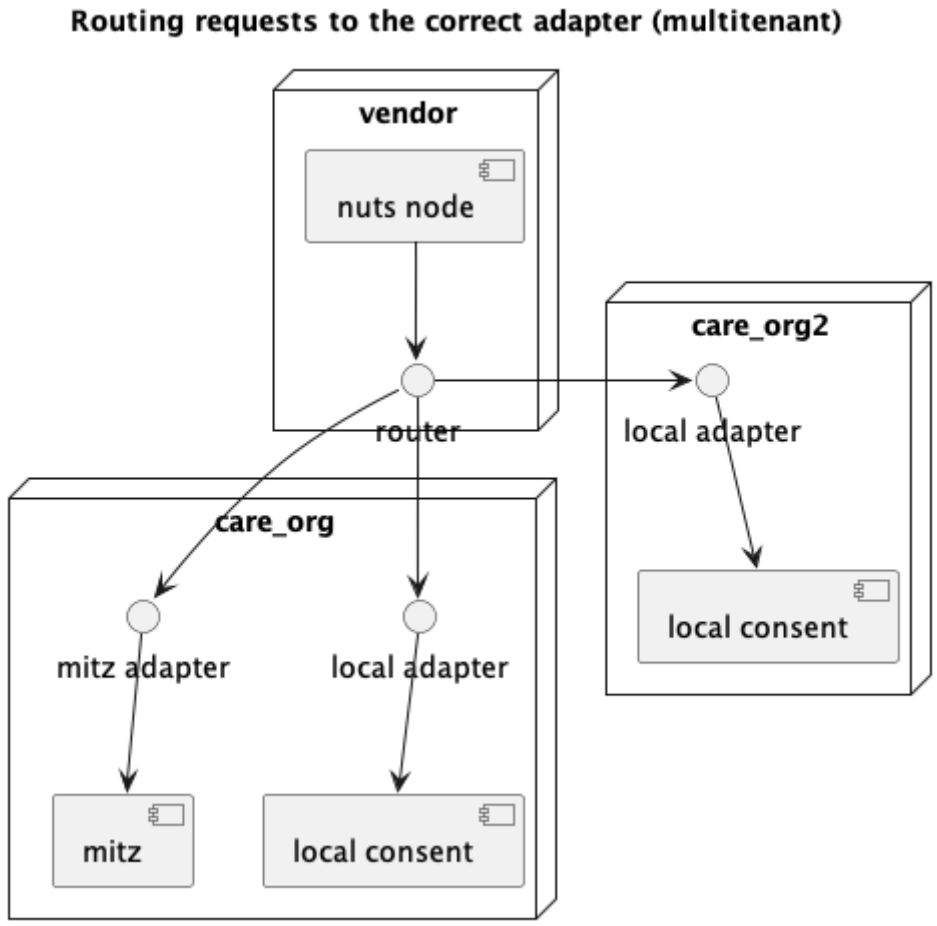
The flow could look like this:



1. A XIS/ECD creates a new wallet in a node and loads a credential. This could be a *NutsOrganizationCredential* or any other credential containing identifying information.
2. The credential is synchronized over the network.
3. The Nuts node of the issuer emits an event for the newly perceived credential.
4. The local consent store listens to these type of events and receives the published credential. The logical identifier is used to query its local DB on *active* consent records.
5. For all records that match an offer is sent through the Nuts node (batching is possible).
6. The Nuts node gets the correct endpoint and forwards the offer to the Nuts node of the holder.
7. The holder node/wallet will initiate a flow similar to the previous paragraph.

Routing requests to the correct adapter

There's an additional challenge that surfaced: the Nuts node is hosted per vendor, but local and/or other adapters might be specific per care organization. This might mean that some sort of routing is needed from the Nuts node to the actual adapter. The adapter might be from a different vendor than the Nuts node. Any security configuration between node and adapter has to be arranged by the vendors.



The router should be able to map DIDs to adapter endpoints. The only reason for not having a router is when all adapters, the wallet, local DBs and node are managed by the same vendor. Since this would restrict the freedom of choice, we'll have to assume this is not the case.