Inter-module RPC usage

There are two methods implemented for inter-module RPC: `moduleMethod` and `moduleMethodResponse`. When using `moduleMethod` there is one huge difference to all other RPC methods: The call is asynchronous. This means, you won't get a response directly to `moduleMethod`. Instead when the request is finished, `moduleMethodResponse` is called as a request on the originating client.

Warning

There is never a response received to `moduleMethod`. Instead `moduleMethodResponse` is called on the originating module.

The reason for this implementation is, that a request might take a long time to process. This hanging request then blocks a lot of processing threads on multiple processes or services. This again might cause Sensaru Cloud to hang.

Routing

Routing packets within Sensaru Cloud is not done by IP address, but by module and module client ID. `moduleMethod` and `moduleMethodResponse` are used as a container for the actual RPC request. So these methods are sort of our equivalent of an IP header. For module to module communication, routing requires only the module ID. Clients connected to modules (i. e. module clients) additionally require the module client ID for routing. So we have four possible routing combinations:

Direction	Description
Module to module	When `moduleMethod` is called from a module on another module, we only need the source module ID and the destination module ID for routing.
Module to module client	When `moduleMethod` is called from a module on a client of another module, we need the source module ID, the destination module ID and the destination module client ID for routing.
Module client to module	When `moduleMethod` is called from a module client on a module, we need the source module ID, the source module client ID and the destination module ID for routing.

Direction	Description
Module client to module client	When `moduleMethod` is called from a module client on another module client, we need the source module ID, the source module client ID, the destination module ID and the destination module client ID for routing.

One important thing to note is that a module client is not allowed to put the source module client ID in `moduleMethod` if it is in an unsecure location or cannot be trusted. In this case this must be done in the module the client is connected to. Here the module client ID can be verified and placed into `moduleMethod`.

moduleMethod

Calling module method

Whenever you want to call a RPC method on another module or module client, you need to execute `moduleMethod`. `moduleMethod` has the following signature:

```
void moduleMethod(
    String destinationModuleId,
    String destinationModuleClientId = "",
    String senderModuleId,
    String senderModuleClientId = "",
    Struct userId = [],
    Struct modulePrincipalId = {},
    Struct homeClientUsers = {},
    Struct homeClients = {},
    String methodName,
    Array parameters = [],
    String ticketId,
    Integer qos
)
```

The signature (obviously) is the same on the sender and receiver side (except in `homegear-cloudconnect` which has a simplified signature). The sender does not need to fill all parameters though, as most of them are filled in transit by other services.

Parameter	Set by	Optional	Type	Description
`destinationModuleId`	Sender	no	`String`	The ID of the module to call the RPC method in (e. g. `c1-device-management`)
`destinationModuleClientId`	Sender	yes	`String`	

Parameter	Set by	Optional	Type	Description
				The ID of the module client to call the RPC method in (e. g. `d1faa8d0-2db4-11ea-af75-674069e60b74_1000.1.1_1`). A module client as a service connected to a module, e. g. edge clients connected to `c1-proxy`.
`senderModuleId`	c1-core	no	`String`	The ID of the sending module. This parameter is filled (or overridden) in `c1-core` with the ID of the sending module.
`senderModuleClientId`	Module	yes	`String`	The ID of the sending module client. Needs to be set to a verified value by the module. This can't be checked by `c1-core`. The value is only required for requests from module clients.
`userId`	Sender	yes	`Struct`	Filled with the ID of the sending user. For requests coming from UIs to `c1-core` this is filled with the logged in user. This parameter can be set by any sender. Just make sure the information is verified, as the validity cannot be checked by `c1-core`.
`modulePrincipalId`	c1-proxy / c1-core	yes	`Struct`	Filled with the edge client's principal ID for requests coming from edge clients and with and with the module's principal ID for requests coming from modules. The latter is relevant, because modules can be associated to a principal and access to other

Parameter	Set by	Optional	Type	Description
				principals must be denied in this case.
`homeClientUsers`	c1-core	yes	`Struct`	Filled with the users associated to the edge client specified with `homeClientId`.
`homeClients`	c1-core	yes	`Struct`	Filled with the edge client IDs associated to the user specified with `userId`.
`methodName`	Sender	no	`String`	The RPC method to call.
`parameters`	Sender	yes	`Array`	The parameters to pass to the RPC method.
`ticketId`	Sender	no	`String`	This parameter is like a password. Fill this parameter with fully random string of appropriate length. **Every request requires a unique ticket ID**. When receiving `moduleMethodResponse`, the ticked ID must be matched.
`qos`	Sender	no	`Integer`	`0` to disable `QoS`. `1` causes the RPC call to be cached until the destination acknowledges the request.

Example

To for example call `getVersion()` on a edge client a request might look like this:

```
{
  "id": 12,
  "method": "moduleMethod",
  "params": [
    "c1-proxy",
    "d1faa8d0-2db4-11ea-af75-674069e60b74_1000.1.1_1",
    "",
    "",
    {},
```

```
      {},
      {},
      {},
      "getVersion",
      [],
      "abcd",
      0
   ]
}
```

To call this method using `c1-module-proxy`, execute:

```
$ curl -X POST -H 'Content-Type: application/json' -d '{"id":12,"method":"moduleMethod","params":["c1-proxy", "d1faa8d0-2db4-11ea-af75-674069e60b74_1000.1.1_1", "", "", [], "", {}, {}, "getVersion", [], "abcd", 0]}' http://localhost:8080
```

The response will be:

```
{"id":12,"jsonrpc":"2.0","result":null}
```

After this, `moduleMethodResponse` is called (as a request!).

Warning

When you're using `c1-module-proxy`, the response is returned directly in `moduleMethod` and `moduleMethodResponse` is not called.

Receiving `moduleMethod`

When `moduleMethod` is called on a module (not a module client), `c1-core` invokes the actual method within the `moduleMethod` call. This means, you will not receive a call to `moduleMethod`. So when another service is calling the following on your module:

```
{
   "id": 12,
   "method": "moduleMethod",
   "params": [
      "c1-proxy",
      "d1faa8d0-2db4-11ea-af75-674069e60b74_1000.1.1_1",
      "",
      "",
      {},
      {},
      {},
      {},
      "getVersion",
      [],
      "abcd",
      0
```

```
    ]
}
```

Then you will receive the following in your module:

```
{
   "id": 12,
   "method": "getVersion",
   "params": [
      <verified metadata>
   ]
}
```

`moduleMethod` and module clients

You are only required to implement `moduleMethod` within your module, when the module has module clients. In this case `c1-core` does not invoke the wrapped method but passes `moduleMethod` to your module. In your module you can then identify the endpoint and invoke the wrapped RPC method directly as the routing information is not required anymore.

> **Note**
>
> When receiving `moduleMethod` you will notice that all or most of the parameters have been filled.

moduleMethodResponse

`moduleMethodResponse` is sent after processing of a call to `moduleMethod` has finished. It has the following signature:

```
void moduleMethodResponse(
    String destinationModuleId,
    String destinationModuleClientId = "",
    String senderModuleId,
    String senderModuleClientId = "",
    Struct userId = {},
    Struct modulePrincipalId = {},
    Struct homeClientUsers = {},
    Struct homeClients = {},
    Mixed response,
    String ticketId,
    Integer qos
)
```

As you can see, most parameters are the same as for `moduleMethod`. The only difference is that `methodName` and `parameters` are replaced by `response`. The latter contains the result of the call to the RPC method specified by `methodName` in `moduleMethod`. The ticked ID must be set to the one from the `moduleMethod` call. Apart from that the same rules as for `moduleMethod` apply.

Example

An example request looks like this:


```
{
   "id": 6,
   "jsonrpc": "2.0",
   "method": "moduleMethodResponse",
   "params": [
      "c1-proxy",
      "d1faa8d0-2db4-11ea-af75-674069e60b74_1000.1.1_1",
      "c1-device-management",
      "",
      {},
      {},
      {},
      {},
      "Homegear 0.8.0-3348",
      ""
   ]
}
```


The response looks like this:


```
{"id":6,"jsonrpc":"2.0","result":null}
```