



ACONA Interface Specification

Autor: Alexander Wendt

**Document history**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Changes | Authors |
| 0.1 | 2019-03-28 | Created Report | Alexander Wendt |
|  |  |  |  |
|  |  |  |  |

Table of contents

1. Introduction 3

2. Analysis 4

3. Conclusion 5

4. Todo 6

5. Template Chapter 7

Literature 10

Internet references 12

Annex A 13

Abbreviations

# Concepts

In the following, some concepts of ACONA are presented to introduce the handling of the interfaces.

## Agents and Functions



Each agent is defined by its functions. At any time, a function can be registered in an agent in-stance. To save development effort of custom functionality, each function offers the following features:

* A function offers services, which can be accessed through other functions, e.g. read or write from a database
* It runs code in a thread either scheduled in intervals or on demand
* Subscribed datapoints trigger custom code in the function, i.e. a listener
* A function access data provided by other functions or sources either through a request-response pattern order through a publish-subscribe pattern
* Provide the current state of the function of monitoring purposes

## Communication with Services



## Instantiation of Agents in Non-Java Languages

# Interface Specification

The following document specifies the communication interfaces of agent functions and codelets. This model can be used to write own ACONA clients in other languages different from Java, which has a native ACONA client.

## Agent Functions Standard

The following mqtt topics are handled by a function.

#### Description

The description of a function is usually only set once and provides a description of the function as a string in a Json message. The usage of this address is optional.

**Publish** to address: <[AgentName]>/[FunctionName]/**description**

e.g. <ACO3>/SAKE/description for the agent „ACO3“ and function „SAKE“

Json message example:

{

"ADDRESS": "SAKE/description",

"AGENT": "ACO3",

"VALUE": "[DESCRIPTION OF THE FUNCTION AS JSON PRIMITIVE]",

"timeStamp": 1553072211865

}

#### Configuration

Through the configuration datapoint, a configuration in Json format can be injected into the function. The usage of this address is optional.

**Subscribe** from address: <[AgentName]>/[FunctionName]/**config**

Json message example:

{

"ADDRESS": "SAKE/config",

"AGENT": "ACO3",

"VALUE": "[JSON OBJECT WITH KEY-VALUE FOR CONFIGURATION]",

"timeStamp": 1553072211865

}

#### Function State

The function state publishes the current state of the function. The following states are available in the system:

* BUILDING: At the creation of the function
* INITIALIZING: At the initliaization of the function variables
* ERROR: Error
* UNDEFINED: Also error
* RUNNING: Currently in action
* FINISHED: Running action has finished

This address is used by other functions to check the state of a function, e.g. to see if there are any errors. The usage of this address is **mandatory**.

**Publish** to address: <[AgentName]>/[FunctionName]/**state**

Json message example for publishing the state:

{

"ADDRESS": "SAKE/state",

"AGENT": "ACO3",

"VALUE": "FINISHED",

"timeStamp": 1553072212162

}

#### Result

The result address is a default return address for the thread in the function. It is used if the function shall return anything to a calling function, but the return address is not explicitly provided. Then it publishes its data to the result address. The usage of this address is optional.

**Publish** to address: <[AgentName]>/[FunctionName]/**result**

Json message example for publishing a result:

{

"ADDRESS": "SAKE/result",

"AGENT": "ACO3",

"VALUE": "Result is a string",

"timeStamp": 1553072211865

}

#### ReplyTo

The replyto address is offered by every agent function as a channel to receive responses from requests. Each function can send a request to another function and expect an answer. In the request message, the replyto address of the caller is provided. As a response is ready, the service function send the result to the replyto address. The usage of this address is **mandatory**.

**Subscribe** from address: <[AgentName]>/[FunctionName]/**replyto**

Json message example of an incoming response:

{

"correlationid": "49b220f2-3aea-4fa0-a4df-c7af21bf73d2",

"replyto": "<ACO3>/SAKE/replyto",

"result": "OK"

}

Another Json message example of an incoming response:

{

"correlationid": "2860bd20-9786-4feb-88ef-ecf6b725f493",

"replyto": "<ACO3>/SAKE/replyto",

"result": {

"workingmemoryaddress": "workingmemory",

"internalmemoryaddress": "internalmemoryaddress"

}

}

#### Command

Every agent function offers a possibility to run its thread externally, i.e. to control the function externally. For that purpose, the function offers a command service. A request that reaches the command service sets a certain command. The following commands as strings are allowed:

* START: Start the thread
* STOP: Stop or reset the thread
* PAUSE: Pause the execution of the thread without resetting it
* EXIT: Shut down the function

As this address represents a service,

The usage of this address is optional.

**Subscribe** from address: <[AgentName]>/[FunctionName]/**command**

Json message example of an incoming command request:

{

"ADDRESS": "Controller/command",

"AGENT": "ACO3",

"VALUE": "PAUSE",

"timeStamp": 1553072212255

}

## Additional Interfaces for Codelet Handlers

A codelet handler is a controller that executes registered codelets either in parallel or in serial depending on their configurations. The codelet handler offer some services for that.

#### Register Codelet

All codelets that are used in the system are registered in a codelet handler. An incoming request consists of

**Subscribe** from address: <[AgentName]>/[FunctionName]/register

Incoming request as Json message as an example. The codelet with address "<ACO3>/SAKE registers itself on runorder 2 (it runs as 2nd in serie)

{

"correlationid": "2860bd20-9786-4feb-88ef-ecf6b725f493",

"replyto": "<ACO3>/SAKE/replyto",

"parameter": {

"caller": "<ACO3>/SAKE",

"order": 2

}

}

The response of the Codelet Handler is

{

"correlationid": "2860bd20-9786-4feb-88ef-ecf6b725f493",

"replyto": "<ACO3>/SAKE/replyto",

"result": {

"workingmemoryaddress": "workingmemory",

"internalmemoryaddress": "internalmemoryaddress"

}

}

In case of an error, the message would contain an error object instead of a result object.

#### Unregister Codelet

Just as a codelet registers, it can unregister.

**Subscribe** from address: <[AgentName]>/[FunctionName]/unregister

Incoming request as Json message as an example.

{

"correlationid": "2860bd20-9786-4feb-88ef-ecf6b725f493",

"replyto": "<ACO3>/SAKE/replyto",

"parameter": {

"caller": "<ACO3>/SAKE"

}

}

The response has the following format:

{

"correlationid": "08d1c79e-4b70-4bf6-81c7-08d201de4d78",

"replyto": "<ACO3>/SAKE/replyto",

"result": "OK"

}

#### Execute Codelet Handler

If the execute codelet handler is executed, it executes all its registered codelets.

**Subscribe** from address: <[AgentName]>/[FunctionName]/execute

Incoming request as Json message as an example. Notice that no parameters are necessary.

{

"correlationid": "fd9fb1d4-8b85-4197-af12-0a13970bb425",

"replyto": "<ACO3>/Controller/replyto",

"parameter": {}

}

The response has the following format:

{

"correlationid": "08d1c79e-4b70-4bf6-81c7-08d201de4d78",

"replyto": "<ACO3>/SAKE/replyto",

"result": "OK"

}

#### Set State

To know the state of the running codelets and to know, when they have finished their runs, the set state service is used.

**Subscribe** from address: <[AgentName]>/[FunctionName]/setstate

Incoming request as Json message as an example. The codelet "<ACO3>/SAKE” sets its state to running.

{

"correlationid": "214e2295-7db1-4c50-8d27-715246f1c14a",

"replyto": "<ACO3>/SAKE/replyto",

"parameter": {

"caller": "<ACO3>/SAKE",

"state": "RUNNING"

}

}

The response has the following format:

{

"correlationid": "08d1c79e-4b70-4bf6-81c7-08d201de4d78",

"replyto": "<ACO3>/SAKE/replyto",

"result": "OK"

}

#### Reset

The rest function has not been implemented yet.

**Subscribe** from address: <[AgentName]>/[FunctionName]/reset

## Additional Interfaces for Codelets

A codelet is an independently running program that is controlled by a codelet handler.

#### Execute Codelet

The codelet handler executes the codelet by triggering the execute codelet service. By doing that, the codelet first sets its service state to “RUNNING”.

**Subscribe** from address: <[AgentName]>/[FunctionName]/execute

{

"correlationid": "fd9fb1d4-8b85-4197-af12-0a13970bb425",

"replyto": "<ACO3>/Controller/replyto",

"parameter": {}

}

It returns a response that it is running

{

"correlationid": "08d1c79e-4b70-4bf6-81c7-08d201de4d78",

"replyto": "<ACO3>/SAKE/replyto",

"result": "OK"

}

# System Processes

For implementing ACONA compatible software, some processes are helpful.

## Process of Reading and Writing Values

If you want to write a custom function that reads from and writes to an agent, the following processes are used.

### Read

1. Create a read request with one parameter: The address you want to read from. The address can have two formats: either [agentname]:[localaddress] or <[agentname]>/[localaddress]. The latter is the true MQTT address on the server.

{

"correlationid": "6309e0b8-1e31-427a-9af7-b4bf50a3addf",

"replyto": "<ACO3>/SAKE/replyto",

"parameter": {

"param": "ConrodSimulator:OP\_3/processedconrod"

}

}

1. Publish the request to the following address: <[Agentname]>/dataaccess/read. Wait for a response either through blocking your system or to use a listener
2. As a result, you receive a response with a matching correlation id with the requested information. This is the part of the result

{

"correlationid": "6309e0b8-1e31-427a-9af7-b4bf50a3addf",

"replyto": "<ACO3>/SAKE/replyto",

"result": {

"timestamp": "1553093653082",

"name": "Conrod",

"usecase\_description": "Use case description 1",

"machine\_state": "RUNNING",

"featuremap": {

"KAobenlngs": {

"name": "KAobenlngs",

"unit": "mm",

"description": "DM KA oben 0o",

"value": 23.403655561939125

}

}

}

}

### Write

1. Create a request to access the write service at the agent. The green part of the message is a datapoint in Json format.

{

"correlationid": "9c63331d-9382-44e8-9dd3-aba97ee50349",

"replyto": "<ACO3>/SAKE/replyto",

"parameter": {

"param": {

"ADDRESS": "OP\_3/machinecommands",

"AGENT": "ConrodSimulator",

"VALUE": {

"Decision": "ReplaceDriller",

"Command": "replace",

"DrillerNumber": 2

},

"timeStamp": 1553093259458

}

}

}

1. As response, a confirmation is returned that the value was written into the data storage

{

"correlationid": "9c63331d-9382-44e8-9dd3-aba97ee50349",

"replyto": "<ACO3>/SAKE/replyto",

"result": "OK"

}

## Process of Running a Codelet

For the implementation of a codelet without extending the ACONA codelet, the following process has to be used:

1. Initialization
   1. Initialize the codelet by setting state to “INITIALIZING” with “set state”
   2. Register the codelet in a codelet handler <[AgentName]>/[FunctionName]/register
   3. Set the state to “FINISHED”
2. Execution
   1. Implement the execute service for codelets and wait for a request. On request:
   2. Return OK to the caller, which is a codelet handler
   3. Set the function state to “RUNNING”
   4. Do the stuff you want
   5. Send a request with “FINISH” to the codelet handler at <[AgentName]>/[FunctionName]/setstate
   6. Set the function state to “FINISH”
3. Shut down
   1. Unregister the codelet in a codelet handler <[AgentName]>/[FunctionName]/unregister

Literature

Internet references