Architecture Design of IC-Service IPC

Author: Masanori Maruyama

Shall not use confluence comment, shall use Jira at https://jira.automotivelinux.org/browse/SPEC-3960

Fixed.

Table of Contents

1. Executive Summary

This document shows a design of program structure, the communication protocol, and the data of IPC between IC-Service and Cluster.

2. Indexes and tables

2.1 Change history

Revision	Date	Author	Description
0.1	2021/03/22		New entry.

2.2 Referenced documents

Tag	Document name	Location of documents
	IC-Service_API_rev0.4.docx	

	Size_list_of_Data_Pool.xlsx	
	IC-EG Architecture Overview - Draft_200803a.pptx	

2.3. Glossary

Term	Mean
-	-

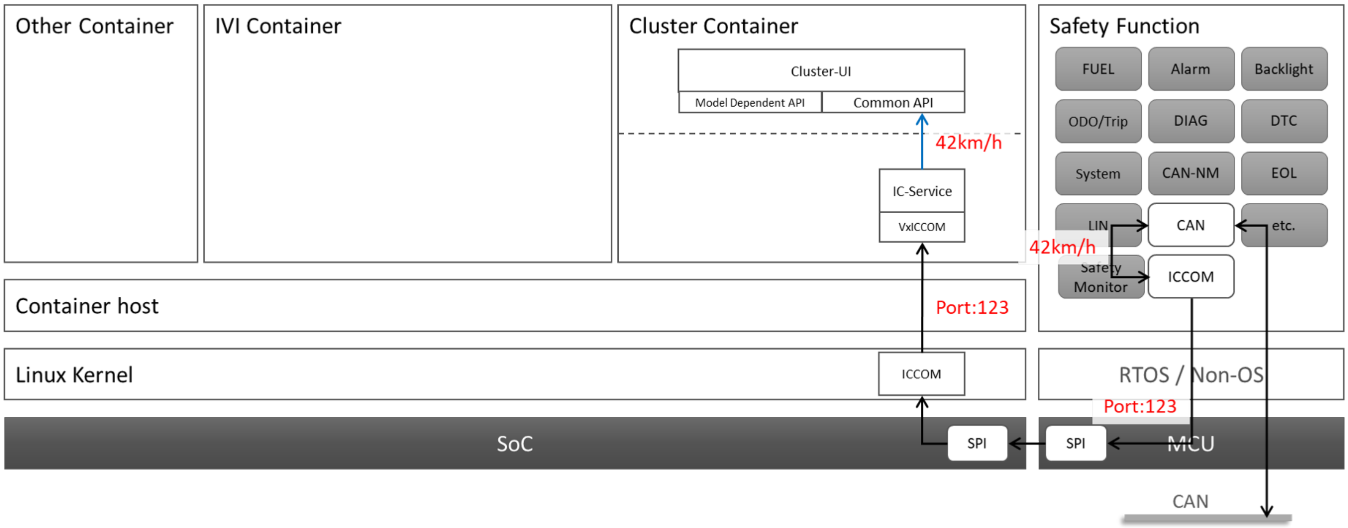
3. Scope

- This document is intended to be read by users of IC-Service IPC.

4. Use case

4.1. The whole system

This chapter describes the whole system including the developmental target of this document. The following picture is quoted from the system architecture, "IC-EG Architecture Overview - Draft_200803a.pptx".



The outline of the system is following:

- This linux system has Cluster UI display and Cluster UI applications based on the vehicle information received from CAN.
- External MCU receives vehicle information from CAN and sends it to IC-Service on SoC through SPI ICCOM. This is because CAN has to be started earlier and also it is related to functional safety.
- Cluster UI reads the vehicle information from IC-Service by calling Common API.

4.2. IC-Service IPC

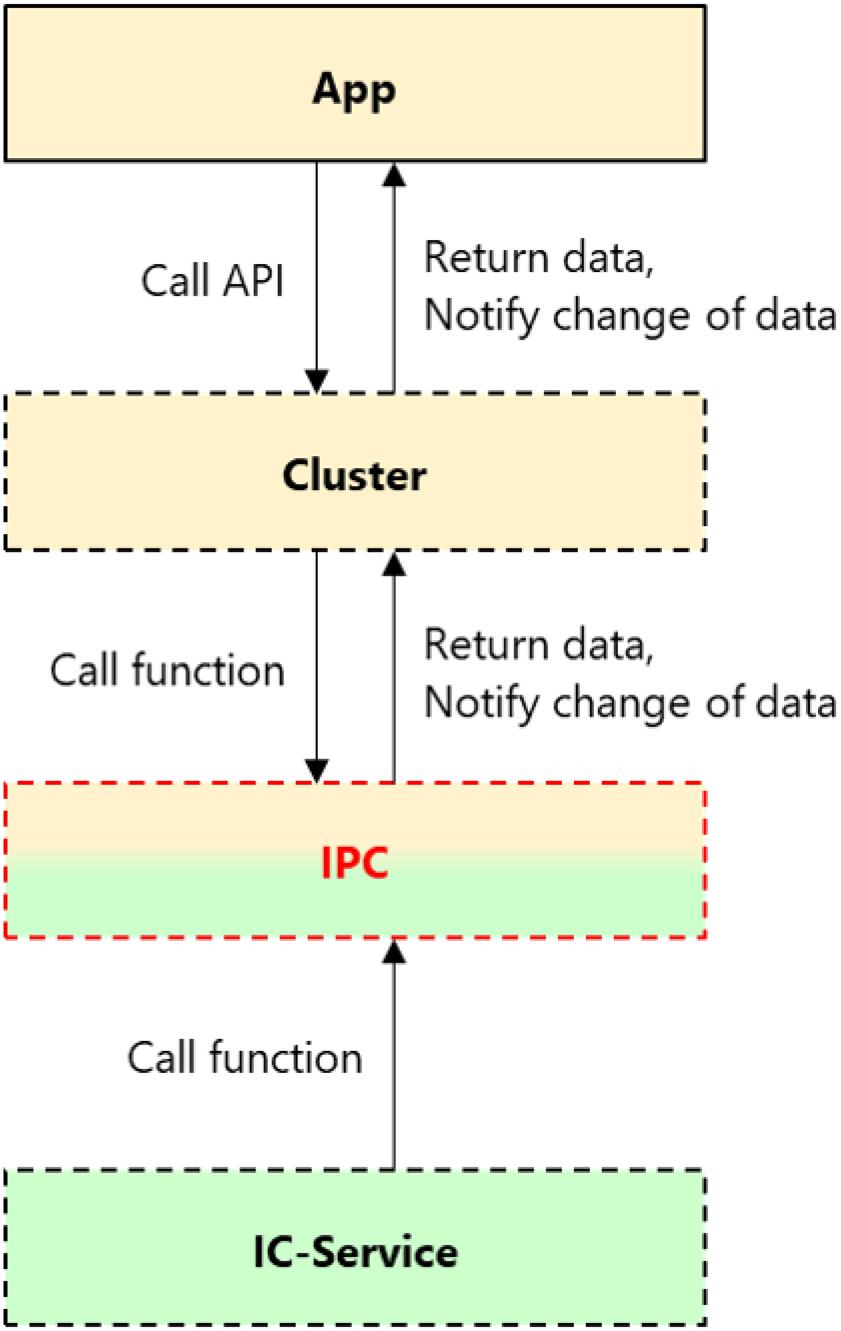
IC-Service IPC in the system shown in 4.1 has the following functionalities.

- Hide the implementation of communication between IC-Service and Cluster UI
- Define communication protocols for Cluster
- Abstract the CAN data received through ICCOM

Components of use cases are described in the following table.

Component	Description
App	App is a part of Cluster UI. App calls API functions for Cluster.
Cluster	Cluster is the library provides APIs to get data of IC-Service. It provides App the signal information received from IC-Service. Cluster runs as a part of App process because it is the layer called as function from App.

IPC	Inter Process Communication. This part provides functionalities to communicate to send signal information from IC-Service to Cluster or the reverse. It has data protocols and it is designed and implemented as the divertable part used by not only Cluster and IC-Service but also other domains.
IC-Service	IC-Service is a part to have vehicle internal information. It sends signal information to Cluster. It runs as another process from App process.

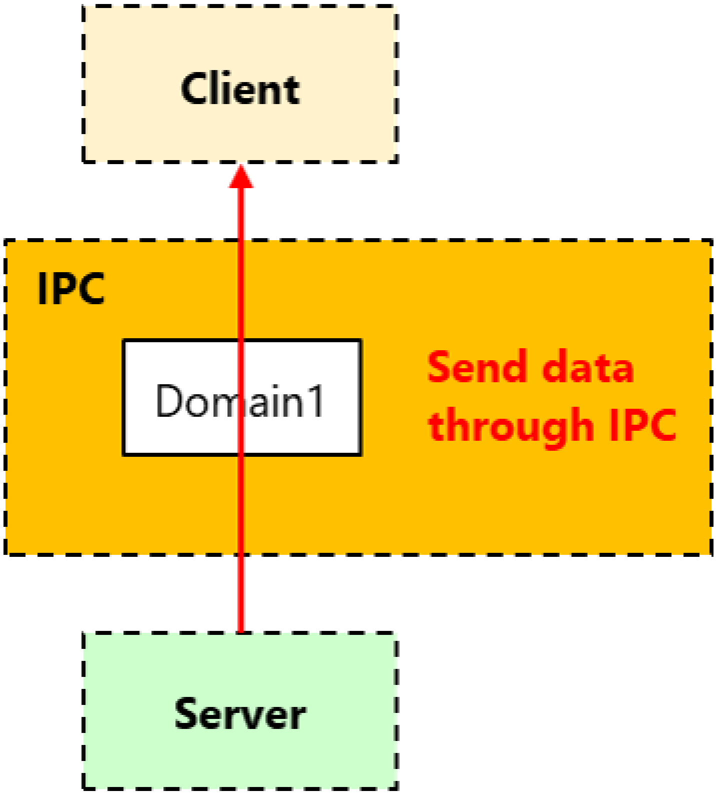


4.2.1. Use cases of sending data through IPC

IPC enables communication between client and server. Use cases of client-server communication are shown below.

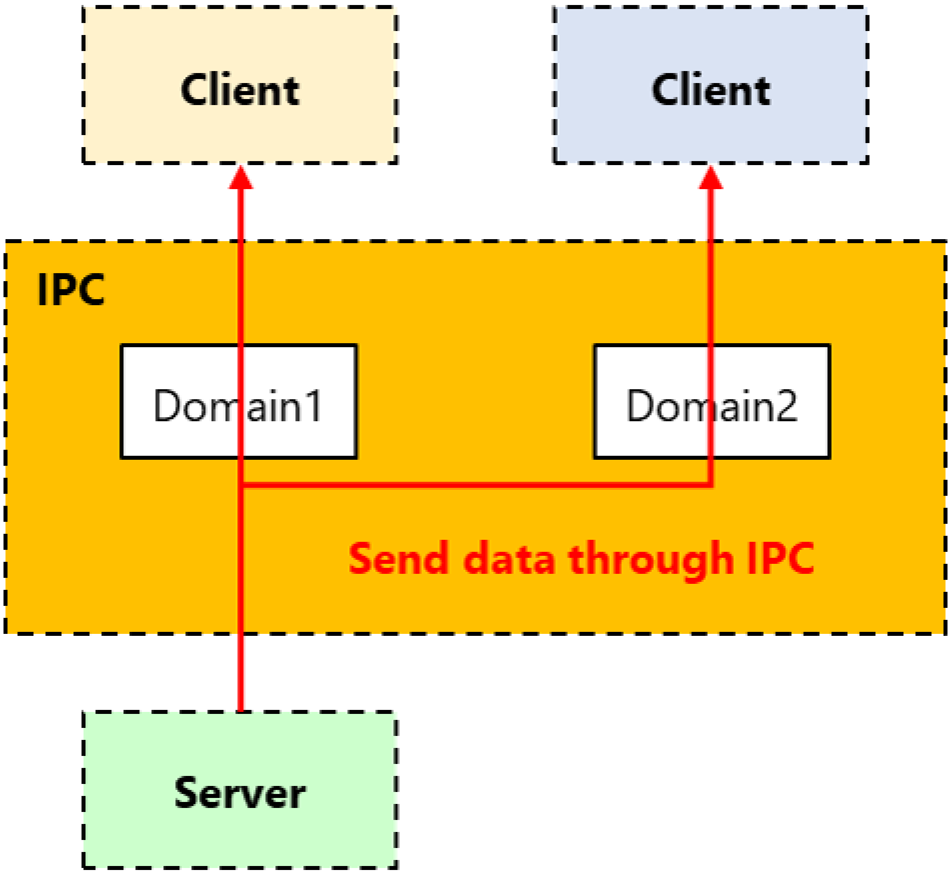
4.2.1.1. One server to one client

One server can send data to one client through IPC with one domain.



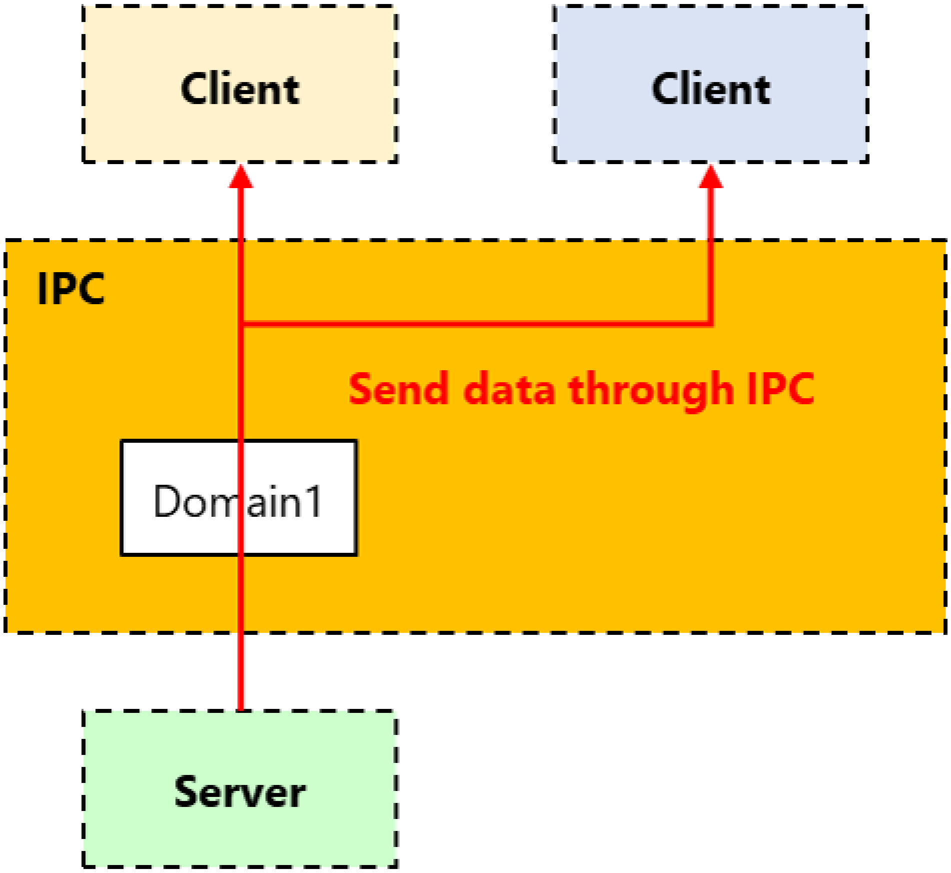
4.2.1.2. One server to multiple clients with multiple domains

One server can send data to multiple clients with different domains.



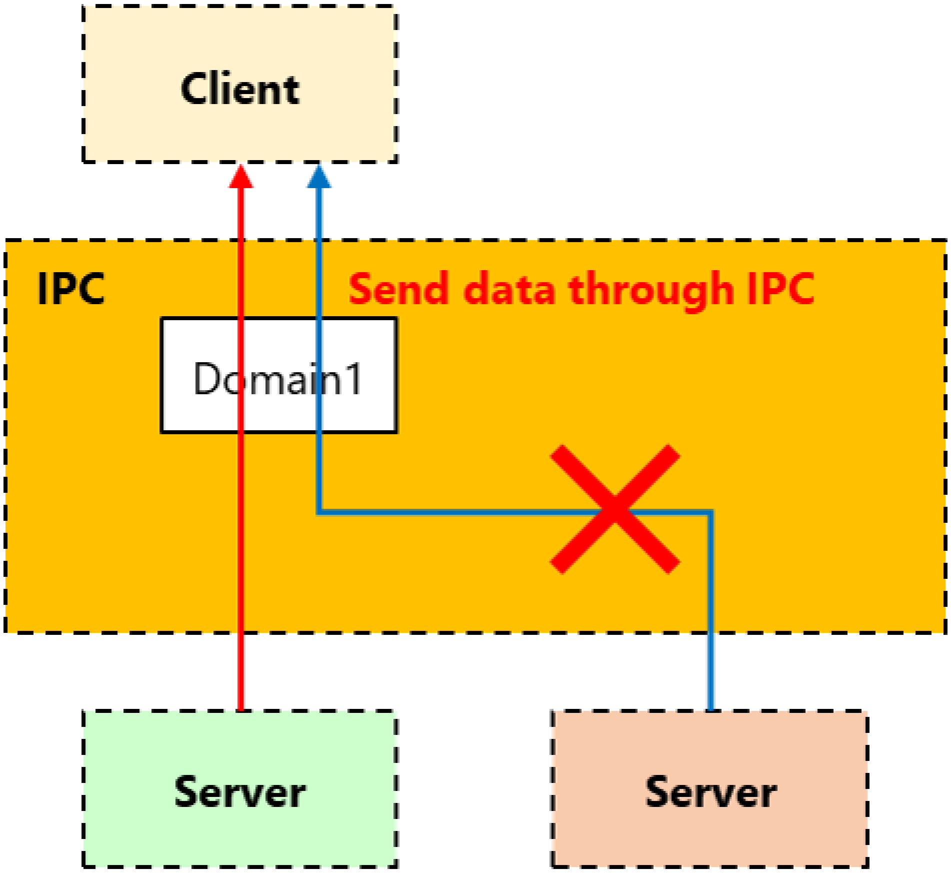
4.2.1.3. One server to multiple clients with single domain

One server can send data to multiple clients with the same domain.



4.2.1.4. Multiple server to one client with single domain

Multiple servers cannot send data to same client with the same domain.



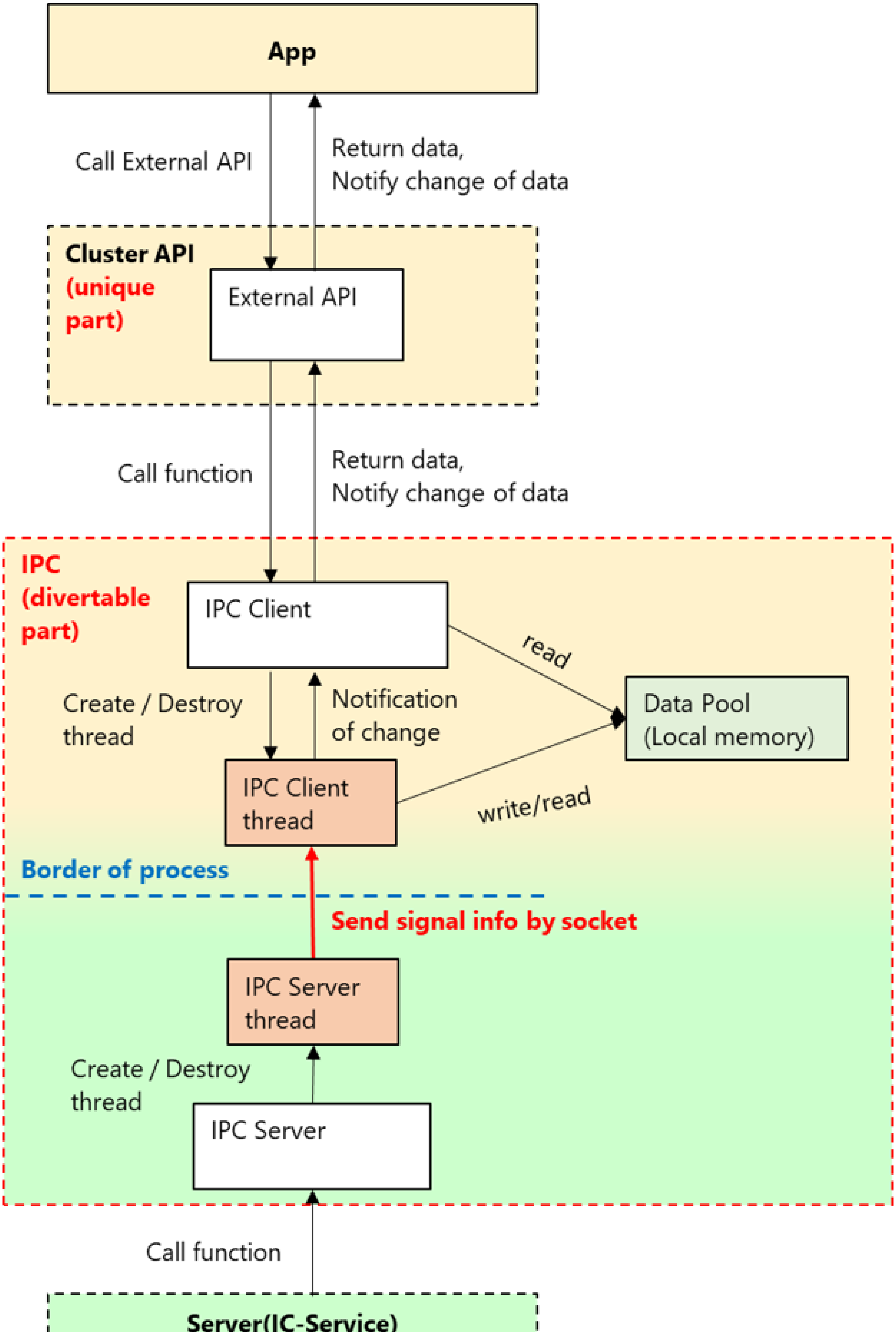
5. Requirements

The following requirements are derived from the use cases in 4.2. and the external API specification.

- Data protocol should be scalable.
- API abstracting part should be implemented separately in order to make this part divertable.
- IPC should enable client-server communication.
  - One server can communicate with one or more clients.
  - One client cannot communicate with multiple servers with the same domain.
    - It is allowed that One client communicates with multiple servers with different domains.
- Cluster UI in HMI Layer and IC-Service in Service Layer should communicate with IPC using Linux Domain Socket in the two ways of communication; polling and event.
  - Polling Cluster UI calls getData function to get value periodically regardless of change of the value.
  - Event Cluster UI registers the signal to IC-Service. When the value of the signal changes, IC-Service notifies Cluster UI with Notify function.
- IPC part should be divertable for other domains and communication protocols.

6. Architectural design

Architectual design of IPC and components associated with IPC is shown below.


App
Call External API
Return data,
Notify change of data
Cluster API
(unique part)
External API
Call function
Return data,
Notify change of data
IPC
(divertable part)
IPC Client
read
Data Pool
(Local memory)
Create / Destroy thread
Notification of change
IPC Client thread
write/read
Border of process
Send signal info by socket
IPC Server thread
Create / Destroy thread
IPC Server
Call function
Server(IC-Service)

6.1. Design Consideration

6.1.1. Outline of IPC design

Outline of IPC design is shown below.

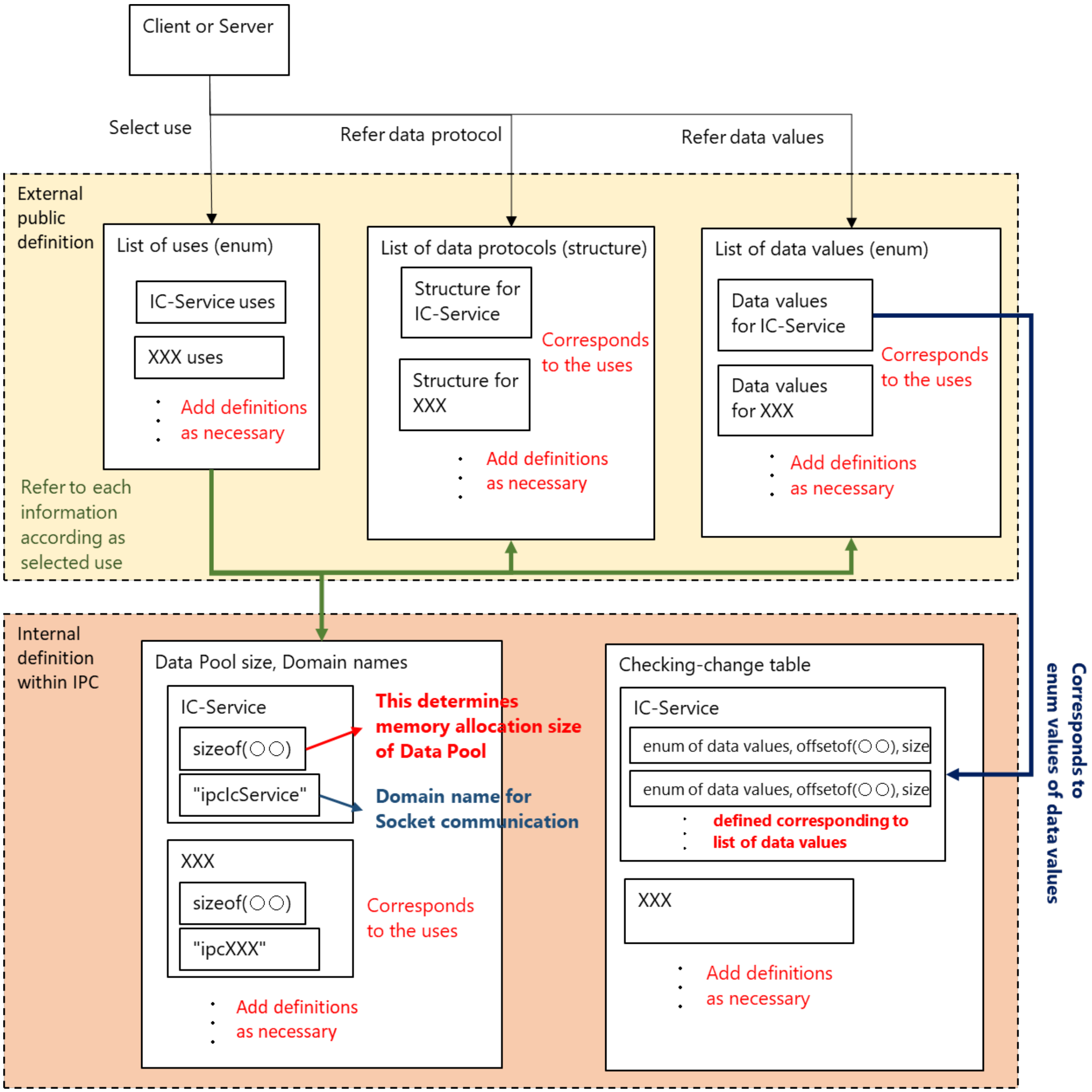
- Precondition
    - IPC communicates by Unix Domain Socket.
    - Use epoll for monitoring file descriptors.
    - IPC part is designed to be divertable.
    - IPC creates a shared library for both client and server.
    - IPC has one public header file. The header is included from both client and server.
- Specification of the public header file
    - The header is named as "ipc.h".
    - ipc.h has the functions and macros to provide IPC functionalities for client and server.
    - ipc.h includes "ipc_protocol.h".
        - In ipc_protocol.h defines data protocols and Unix Domain names for all usages.
        - One data protocol and one Unix Domain name are defined for each of usage.
    - Client and server select one from usage defined in ipc_protocol.h
- Specification of communication and data
    - Another thread for IPC communication is created in IPC. IPC Client and IPC Server communicate on the thread.
        - One thread is created for one process, even in the case that multiple domains are used.
        - IPC Client allocates memory for Data Pool at start of the thread.
    - The client-side thread receives the data sent from server and writes the data into Data Pool.
        - The data is all of the structure defined as the data protocol for the domain.
        - Then client also detects a change of Data Pool and notifies client-side App of the change by callback.
- Limitation
    - Communication is allowed as only one direction, sending from server to client.
    - It is not allowed for multiple servers to use same domain.

6.1.2. How to define data protocol

Data sent/received for each usage and Unix Domain names for communication are defined in ipc_protocol.h. When the usage of IPC divertable part is expanded, new definition of them are added to ipc_protocol.h.

The defined items in ipc_protocol.h are as follows:

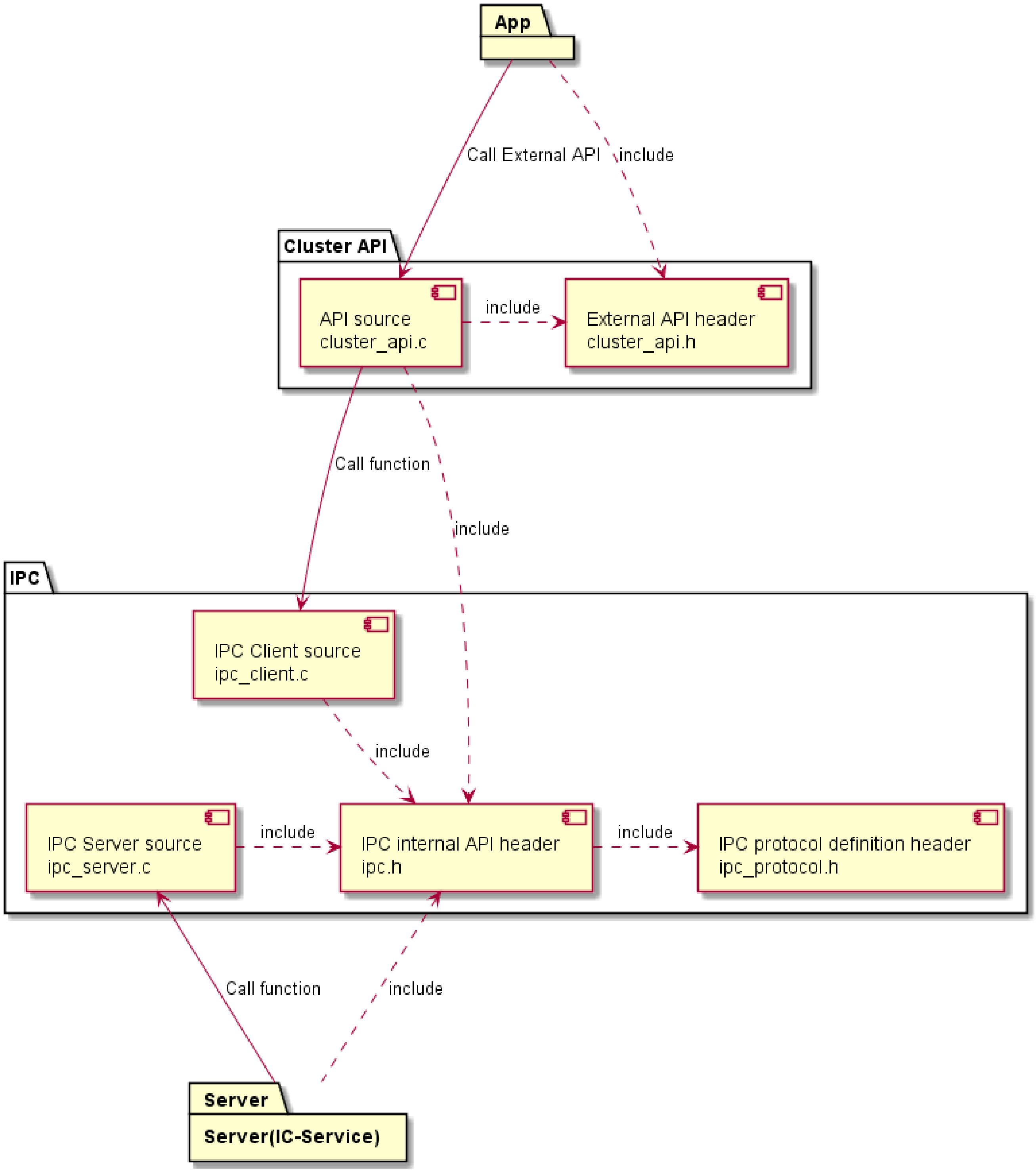
- enum definition for client/server to select an usage
- data protocols (structure) corresponding to the usage (referable from client/server)
- enum definition of data values corresponding to the usage (referable from client/server)
- size of Data Pool and Domain name corresponding to the usage (used within IPC)
- address (offset) table for checking the change of data corresponding to the usage (used within IPC)


Client or Server
Select use
Refer data protocol
Refer data values
External public definition
List of uses (enum)
IC-Service uses
XXX uses
Add definitions as necessary
List of data protocols (structure)
Structure for IC-Service
Structure for XXX
Corresponds to the uses
Add definitions as necessary
List of data values (enum)
Data values for IC-Service
Data values for XXX
Corresponds to the uses
Add definitions as necessary
Refer to each information according as selected use
Internal definition within IPC
Data Pool size, Domain names
IC-Service
sizeof(○○)
"ipcIcService"
This determines memory allocation size of Data Pool
Domain name for Socket communication
XXX
sizeof(○○)
"ipcXXX"
Corresponds to the uses
Add definitions as necessary
Checking-change table
IC-Service
enum of data values, offsetof(○○), size
enum of data values, offsetof(○○), size
defined corresponding to list of data values
XXX
Add definitions as necessary
Corresponds to enum values of data values

6.2. External view

6.2.1. Component structure of IPC

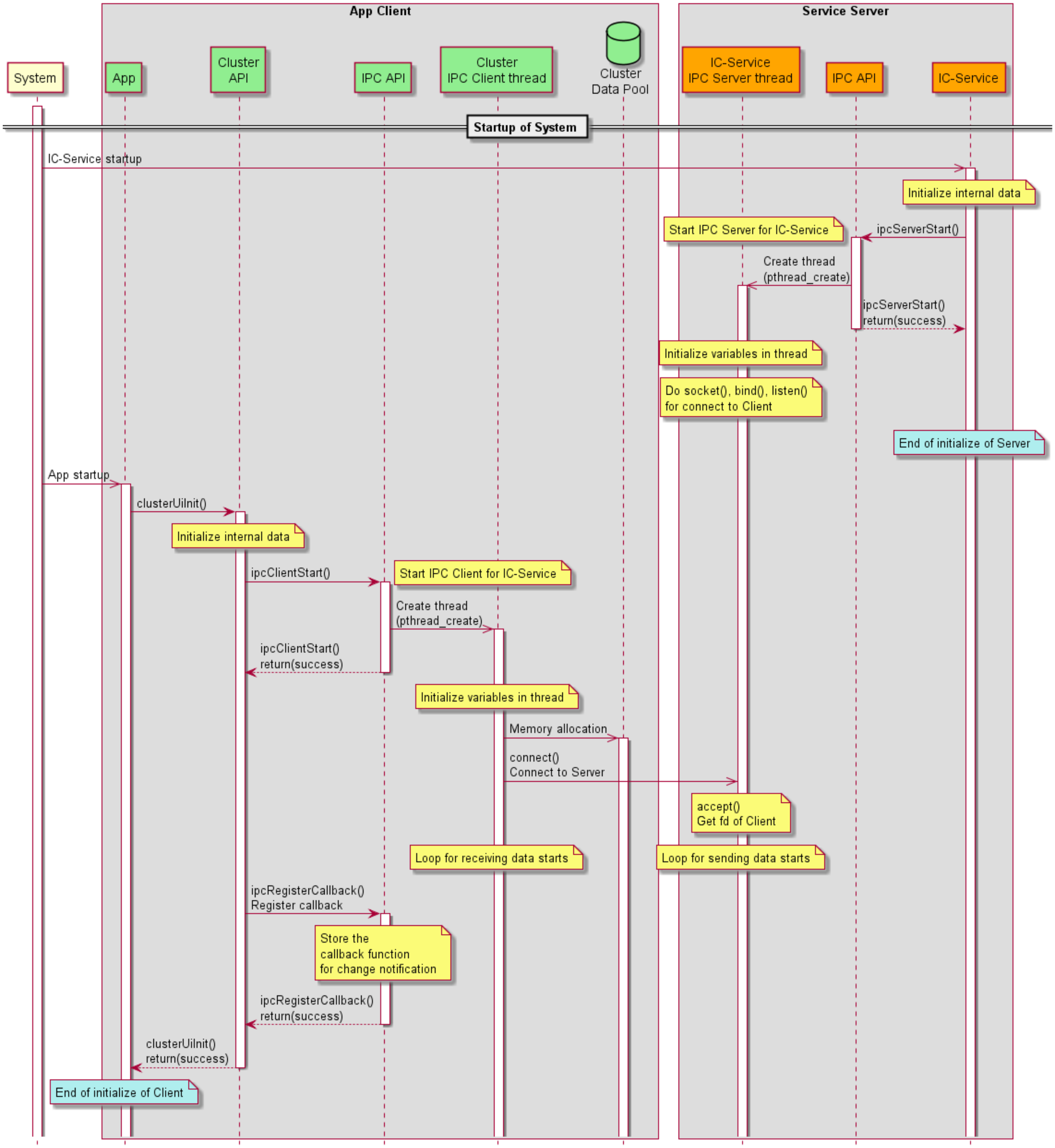
Source and header files in the components associated with IPC is shown below.



6.2.2. Inter-component sequence of IPC

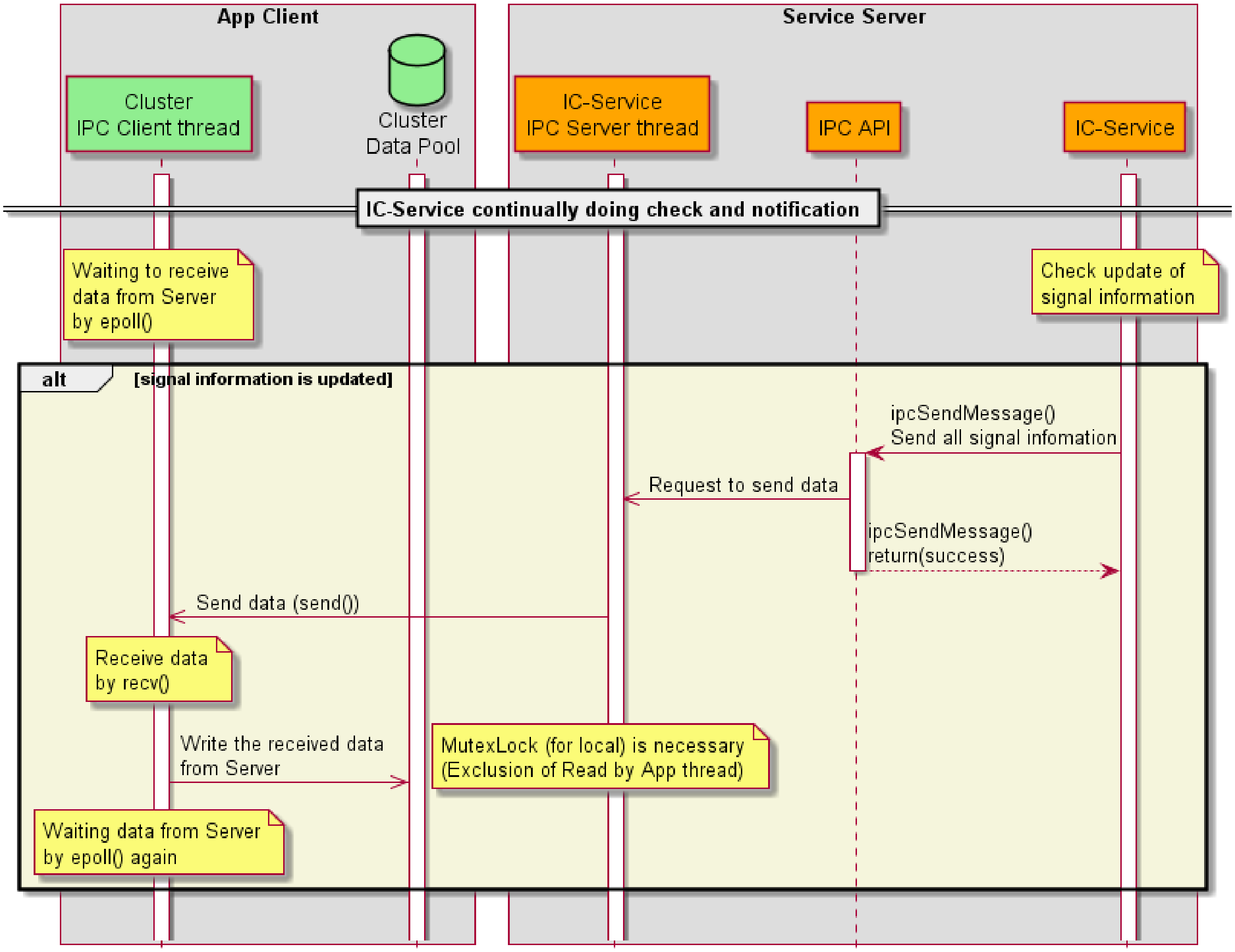
This chapter shows sequences between component elements of the architecture in 6. About APIs between App and Cluster API in the sequences, refer to the external API specification. About APIs between Cluster API or IC-Service and IPC API, refer to 7.2.

6.2.2.1. Startup/Initialization

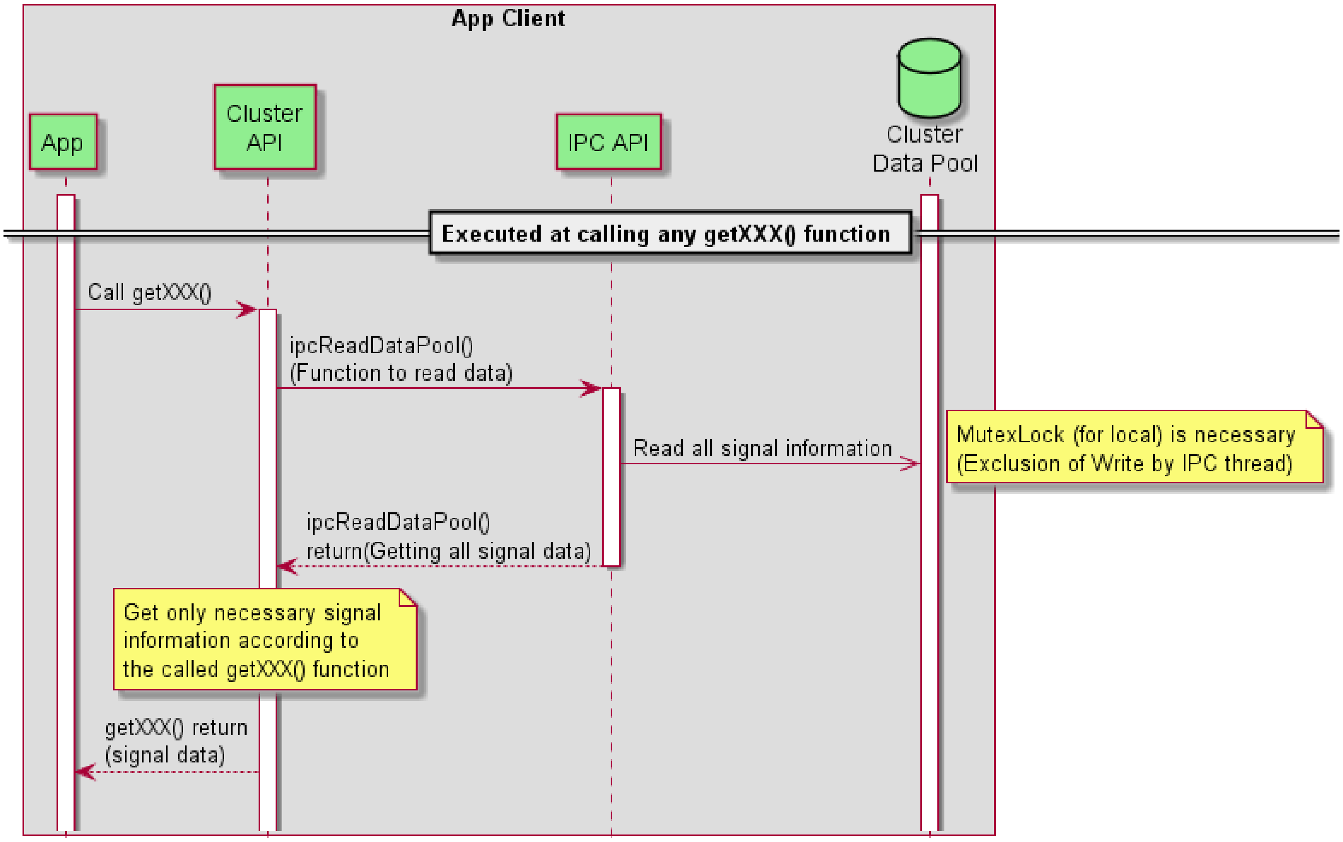

App Client
Service Server
System
App
Cluster API
IPC API
Cluster IPC Client thread
Cluster Data Pool
IC-Service IPC Server thread
IPC API
IC-Service
Startup of System
IC-Service startup
Initialize internal data
Start IPC Server for IC-Service
ipcServerStart()
Create thread (pthread_create)
ipcServerStart() return(success)
Initialize variables in thread
Do socket(), bind(), listen() for connect to Client
End of initialize of Server
App startup
clusterUiInit()
Initialize internal data
ipcClientStart()
Start IPC Client for IC-Service
Create thread (pthread_create)
ipcClientStart() return(success)
Initialize variables in thread
Memory allocation
connect() Connect to Server
accept() Get fd of Client
Loop for receiving data starts
Loop for sending data starts
ipcRegisterCallback() Register callback
Store the callback function for change notification
ipcRegisterCallback() return(success)
clusterUiInit() return(success)
End of initialize of Client

6.2.2.2. Update DataPool

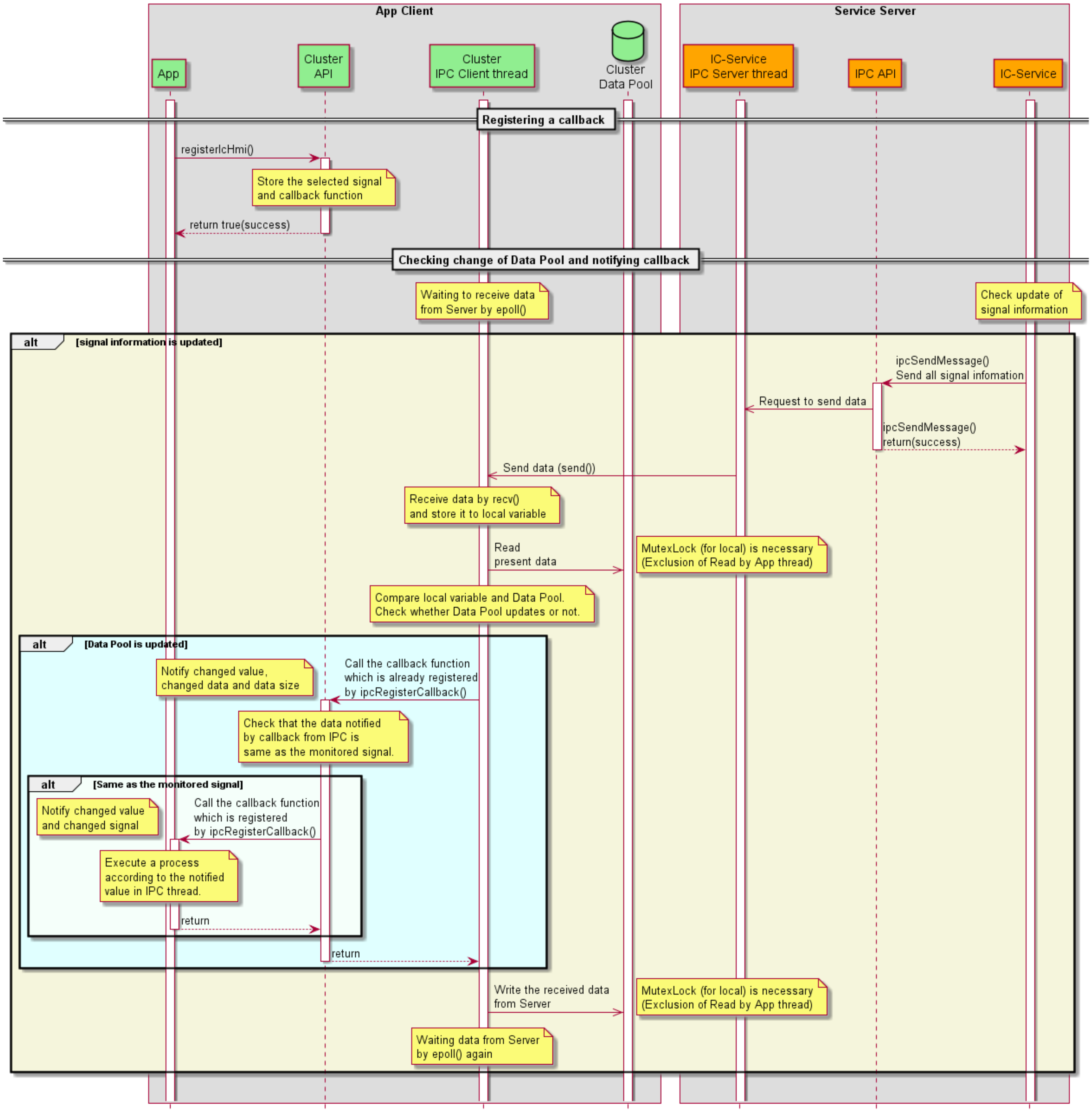
Broken image



6.2.2.3. Process of getter API


App Client
App
Cluster API
IPC API
Cluster Data Pool
Executed at calling any getXXX() function
Call getXXX()
ipcReadDataPool()
(Function to read data)
Read all signal information
MutexLock (for local) is necessary
(Exclusion of Write by IPC thread)
ipcReadDataPool()
return(Getting all signal data)
Get only necessary signal
information according to
the called getXXX() function
getXXX() return
(signal data)

6.2.2.4. Register/Notify



App Client
Service Server
App
Cluster API
Cluster IPC Client thread
Cluster Data Pool
IC-Service IPC Server thread
IPC API
IC-Service
Registering a callback
registerIcHmi()
Store the selected signal and callback function
return true(success)
Checking change of Data Pool and notifying callback
Waiting to receive data from Server by epoll()
Check update of signal information
alt
[signal information is updated]
ipcSendMessage()
Send all signal infomation
Request to send data
ipcSendMessage()
return(success)
Send data (send())
Receive data by recv() and store it to local variable
Read present data
MutexLock (for local) is necessary (Exclusion of Read by App thread)
Compare local variable and Data Pool. Check whether Data Pool updates or not.
alt
[Data Pool is updated]
Notify changed value, changed data and data size
Call the callback function which is already registered by ipcRegisterCallback()
Check that the data notified by callback from IPC is same as the monitored signal.
alt
[Same as the monitored signal]
Notify changed value and changed signal
Call the callback function which is registered by ipcRegisterCallback()
Execute a process according to the notified value in IPC thread.
return
return
Write the received data from Server
MutexLock (for local) is necessary (Exclusion of Read by App thread)
Waiting data from Server by epoll() again

6.2.3. Pkg-config

IPC and Cluster API prepare pkg-config. Names of metadata files are the followings:

- IPC: ipc.pc
- Cluster API: cluster_api.pc

7. Interface definition

7.1. External API

Refer to "IC-Service_API_rev0.4.docx".

7.2. Internal API

IPC has the following internal apis. Details of the apis and how to use them in case of IC-Service are described in 8.2.1.

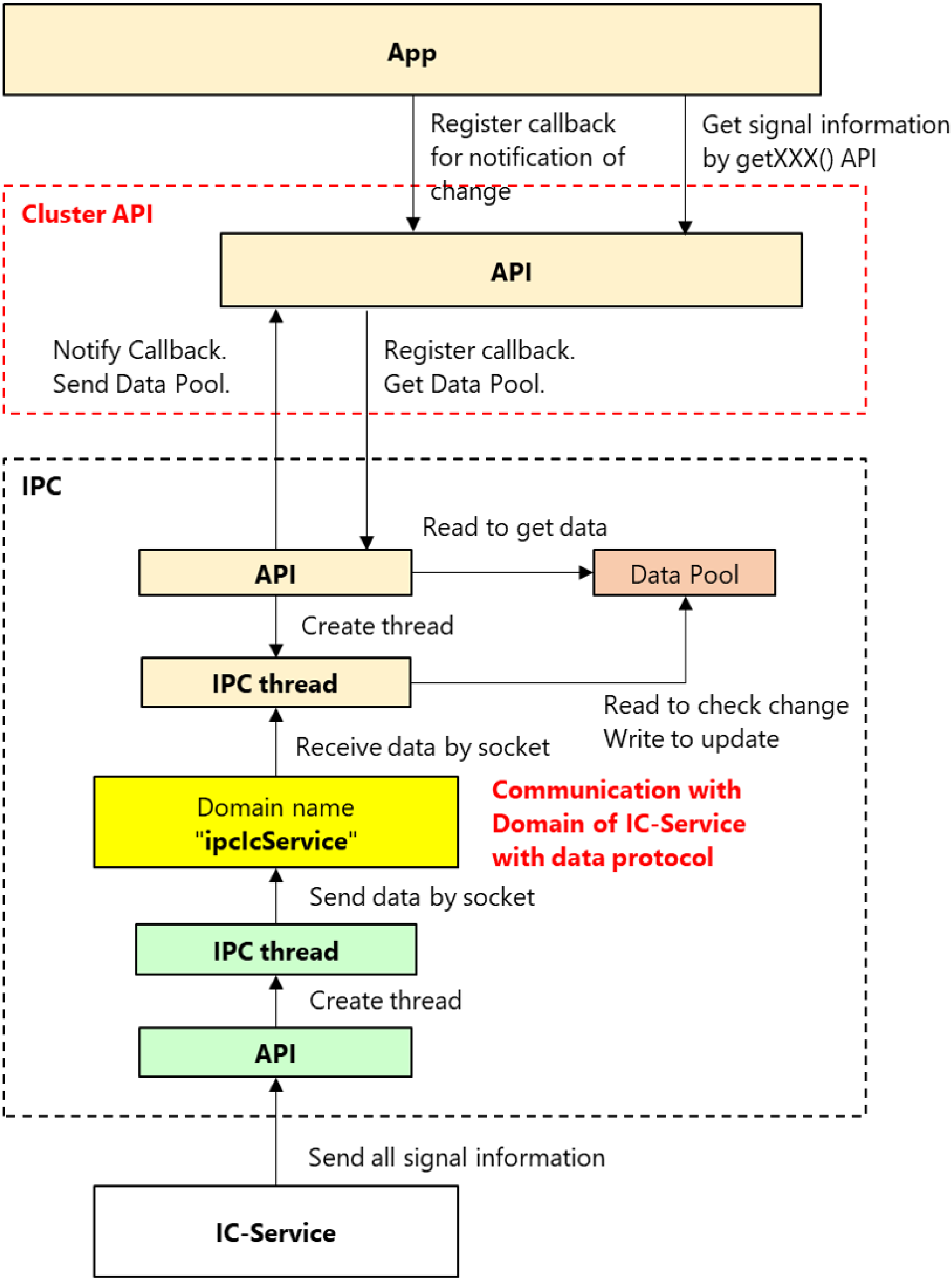
Function name	Explanation	User	Section
ipcServerStart()	Start IPC Server.	Server	8.2.1.1
ipcSendMessage()	Send data to Client.	Server	8.2.1.3
ipcServerStop()	Stop IPC Server.	Server	8.2.1.5
ipcClientStart()	Start IPC Client and connect to the IPC Server which has same Domain.	Client	8.2.1.1
ipcReadDataPool()	Read the Data Pool which stores data received from Server.	Client	8.2.1.4
ipcRegisterCallback()	Register a callback function for notification of change of Data Pool.	Client	8.2.1.2
ipcClientStop()	Stop IPC Client.	Client	8.2.1.5

7.3. Communication protocol

7.3.1. Communication protocol design

This chapter shows the communication data between IC-Service and Cluster.

7.3.1.1. Data and communication framework


App
Register callback for notification of change
Get signal information by getXXX() API
Cluster API
API
Notify Callback. Send Data Pool.
Register callback. Get Data Pool.
IPC
Read to get data
API
Data Pool
Create thread
IPC thread
Read to check change Write to update
Receive data by socket
Domain name "ipcIcService"
Communication with Domain of IC-Service with data protocol
Send data by socket
IPC thread
Create thread
API
Send all signal information
IC-Service

7.3.1.2. Data protocol

- The Domain name for IC-Service used by IPC is "ipcIcService" and is defined in ipc_protocol.h.
- IC-Service is Server and Cluster is Client. IC-Service sends all signal information to Cluster API.
- All signal information is entered into one structure. The structure is sent/received through IPC.
    - The structure for IC-Service is defined in ipc_protocol.h.
    - IC-Service sends all signal information each time, even in case that only a portion of the signals are updated.
- Sending cycle from IC-Service to Cluster is assumed to be about 10 msec as default.
- When App calls getXXX() function, App can get the data stored in Data Pool at the time.
    - App can get not all data but only the data correspoinding to getXXX().
- App registers signal and callback function with registerIcHmi().
- According to specification of notifyIcHmi(), only updated signal information is sent to App by callback function.
- Callback notification to App is executed only in case that Cluster API get notification of the change of data from IPC and the changed signal is the signal registered with registerIcHmi().

7.3.2. Data Pool design

- As described in 7.3.1.2, IPC Client stores all signal information sent from IC-Service in Data Pool.
- As described in "Size_list_of_Data_Pool.xlsx", the necessary size of Data Pool for Domain name "ipcIcService" is the following:
    - 276 bytes in 32-bit environment
    - 296 bytes in 64-bit environment
- For checking changes of signal information, the all signal information to be compared with Data Pool is stored locally in IPC thread.
    - Size of the local data is 276 or 296 bytes, same as Data Pool.

8. Component details

8.1. Other component

Refer to 6.2.2.

8.2. Designed component

8.2.1. Detailed design of internal APIs

8.2.1.1. IPC start for Client/Server

ipcServerStart() and ipcClientStart() are called with usage (enum) as argument. For IC-Service, these APIs are called as the following:

- IPC Server (called prior to IPC Client) ipcServerStart(IPC_USAGE_TYPE_IC_SERVICE);
    - Executes socket(), bind() and listen() as Server for IC-Service. Waits until connected by Client for the same usage.
    - Creates a thread for monitoring the connection with Client.
        - Monitor connection state by epoll.
        - Connect to Client by accept().
        - Close connection with Client by close().
- IPC Client ipcClientStart(IPC_USAGE_TYPE_IC_SERVICE);
    - Create Data Pool area for storing received data. Size of the Data Pool depends on the usage.
    - Execute socket() and connect() as Client for IC-Service. Connect to Server of the same usage.

8.2.1.2. Callback function for Client

ipcRegisterCallback() is called with usage (enum) and callback function. It is called only by Client, which calls ipcClientStart(). If the process which does not call ipcClientStart() calls this function, it returns an error. The specified callback function is used for notification of the change of a data received from Server. For IC-Service, for example, these APIs are executed as the following:

- ipcRegisterCallback(IPC_USAGE_TYPE_IC_SERVICE, changeNotifyCb)
    - Enables caller to receive callback notification about all of the data monitored for the usage.
    - In case of IC-Sevice, changes of the parameters only written in "3-2-1 TellTale" of "IC-Service_API_rev0.4.docx" are monitored.
        - For example, the value corresponding to getSeatbelt() in Telltale changes, then the change is notified. Whereas, the change of the value corresponding to getFrontRightSeatbelt(), not in 3-2-1 TellTale, is not notified.
- Memorize one callback function for every usage.
    - The callback function is overwritten when ipcRegisterCallback() is called for the same usage again.
- Refer to 6.1.2 for specification of callback.

8.2.1.3. Send message from Server

ipcSendMessage() is called with usage (enum) and sending data (address, size). It is called only by Server, which calls ipcServerStart(). If the process which does not call ipcServerStart() calls this function, it returns an error. For IC-Service, for example, these APIs are executed as the following:

- ipcSendMessage(IPC_USAGE_TYPE_IC_SERVICE, &dataIcService, sizeof(dataIcService));
    - The second argument is the start address of sending data.
    - The third argument is the size of sending data.
    - IPC_DATA_IC_SERVICE_S struct is used as sending data for IC-Service. In the above example, dataIcService is intended to be the instance of this structure.
    - Send data to the Client (or all of the Clients) specifying the same usage.

8.2.1.4. Read received data by Client

ipcReadDataPool() is called with usage (enum), address to store received data and the size enough to store the received data. It is called only by Client, which calls ipcClientStart(). If the process which does not call ipcClientStart() calls this function, it returns an error. For IC-Service, for example, these APIs are executed as the following:

- ipcReadDataPool(IPC_USAGE_TYPE_IC_SERVICE, &dataIcService, &size);
  - The data received from Server is stored in Data Pool.
  - When this function is called, it reads all of the data of Data Pool stored in IPC Client.
  - The second argument is an address to store data of Data Pool. In the above example, dataIcService is the instance of IPC_DATA_IC_SERVICE_S structure.
  - The third argument is used for input and output.
    - For input, this argument is the size of the storing area specified by the second argument.
    - For output, the size of the stored data is set in this argument.

8.2.1.5. IPC stop for Client/Server

ipcServerStop() and ipcClientStop() are called with usage (enum) as argument. For IC-Service, these APIs are called in the following way:

- IPC Client (preferably called prior to Server side)
  - ipcClientStop(IPC_USAGE_TYPE_IC_SERVICE);
    - Close the connection.
    - Release Data Pool area.
- IPC Server
  - ipcServerStop(IPC_USAGE_TYPE_IC_SERVICE);
    - Disconnect from all of the connected Clients by close().
    - Close own socket.

8.2.2. Detection of change of data and callback notification

When Server sends data to Client, then the Client detects change of the data and call a callback function for notification of the change.

- How to detect change (Client-side IPC thread process)
  - Prepares the "checking-change table" for each usage that lists offset addresses in a structure and sizes of the checking data. (in order to process any structure in same way)
  - When Server sends data to Client, then stores the received data to a local variable in a thread.
  - Compares each data of the local variable with that of Data Pool according to the checking-change table.
  - Detects changed value and then notifies Client of the value by callback.
- Callback specification
  - App registers callback function(s) beforehand.
  - The following is a definition of a callback function. Callback function is intended to be used for any usage.

```
typedef void (*IPC_CHANGE_NOTIFY_CB)(void * pData, signed long size, int kind);
    - pData: an address to store a changed data
    - size: size of data
    - kind: usage of data (enum value corresponding to the usage)
```

- Making checking-change table
  - It is easy to make an offset table of data in a structure by using offsetof().

8.2.3. Setting path of socket file

When ipcServerStart() is executed, a socket file for Server-Client communication is generated because of using Unix Domain Socket. This socket file is generated in the location an application for Server runs in default. However, it is possible to specify a path of the socket file by the environmental variable, which is IPC_DOMAIN_PATH. For example, If IPC_DOMAIN_PATH is set to "/tmp", the socket file for IC-Service usage is generated in the path and the name of socket file is "/tmp/ipcIcService".

8.3 Deployment view

None in particular.