Acquila socket interface

General description : Acquila features a ZeroMQ based socket interface for scripting the scanner and interfacing with a remote client.

ZeroMQ version : >3.2.4 (see https://zeromq.org/)

ZeroMQ pattern used : Two TCP/IP sockets are created, an outbound **PUB** port and an inbound **SUB** port (from the perspective of Acquila).
Acquila **binds** to both sockets, the remote client should **connect** to them.

Default values :
Outbound : tcp://localhost:50000
Inbound : tcp://localhost:50001

The **outbound port** serves two goals :
- forward a "SEND command" event issued within Acquila to the remote client*
- forward "REPLY" events issued within Acquila to the remote client

The **inbound port** serves two goals:
- receive a command issued by a remote client to be broadcasted within Acquila
- receive REPLY events from the remote client and broadcast them in Acquila

Allowed commands:
This depends on your license and the operator level of the user logged on to Acquila. Contact TESCAN-XRE for more details.

* : this does not imply the command is meant to be executed by the client

Acquila socket interface

Formatting:

There are two types of command events within Acquila :

SEND events : broadcast a command to be executed by the responsible component

REPLY events : broadcasted by the component to give feedback on the progress of a command

Both SEND and REPLY events are passed on the socket interface using a JSON formatted string.

An example of a SEND event:

{"**component**":“name","**comp_phys**":“physical_name","**command**":"your_command","**arg1**":“your_arg1","**arg2**":“your_arg2","**reply**":“your_reply","**reply type**":"","**comp_type**":"other","**tick count**":1380210404,"**UUID**":26481}

Please note that UUID numbers are passed to JSON as unsigned 64-bit integers. Some JSON parsing codes (such as those using JavaScript) cast the value to standard IEEE float, thus losing the required precision to correctly store the UUID value. This will cause responses to have a wrong UUID value, leaving the structure listening for feedback waiting until a timeout occurs.

Acquila socket interface

Formatting:

An example of a SEND event:

{"**component**":“name","**comp_phys**":“physical_name","**command**":"your_command","**arg1**":“your_arg1","**arg2**":“your_arg2","**reply**":“your_reply","**reply type**":"","**comp_type**":"other","**tick count**":1380210404,"**UUID**":26481}

All fields are required!

component : the abstract name of the addressed component (tube, camera, rot_obj, etc.). It is up to the client to decide if this is a command it should execute, or if it should be ignored (for example if multiple clients are connected, they will all receive the command, but only one client should start executing it)

comp_phys : the physical name of the addressed component (as described in the Acquila settings file)

command : the command which was issued

... more on next page ...

Acquila socket interface

Formatting:

An example of a SEND event:

{"**component**":“name","**comp_phys**":“physical_name","**command**":"your_command","**arg1**":“your_arg1","**arg2**":“your_arg2","**reply**":“your_reply","**reply type**":"","**comp_type**":"other","**tick count**":1380210404,"**UUID**":26481}

All fields are required!

arg1 : the first argument field (optional, empty if no arguments are needed)

arg2 : the second argument field (optional, empty if no arguments are needed)

reply : always empty for SEND events (this field is used for the content of replies, see further for details)

… more on next page …

Acquila socket interface

Formatting:

An example of a SEND event:

{"**component**":“name","**comp_phys**":“physical_name","**command**":"your_command","**arg1**":“your_arg1","**arg2**":“your_arg2","**reply**":“your_reply","**reply type**":"","**comp_type**":"other","**tick count**":1380210404,"**UUID**":26481}

All fields are required!

reply type: empty for SEND events. For REPLY events the possible values are:

RCV (confirmation of reception of the command by the code that will execute the command)

FDB (optional intermediate feedback on the progress of execution)

ACK (confirmation of finishing execution of the command) **OR** ERR (notification that an error occurred while executing the command)

… more on next page …

Acquila socket interface

Formatting:

An example of a SEND event:

{"**component**":“name","**comp_phys**":“physical_name","**command**":"your_command","**arg1**":“your_arg1","**arg2**":“your_arg2","**reply**":“your_reply","**reply type**":"","**comp_type**":"other","**tick count**":1380210404,"**UUID**":26481}

All fields are required!

comp_type : possible values are tube, motor, camera or other

tick count : a tick count which holds the time when the command was issued by Acquila

UUID : a unique identifier number for each broadcasted command. When sending a command from the client to Acquila one can either leave this value to 0, then Acquila will assign a UUID when issuing the command on Acquila side or the client side can already fill in a UUID. But then the client

... more on next page ...

Acquila socket interface

Formatting:

REPLY events:

When executing a command progress is reported back to the sender through REPLY events.

It is critical that both the tick count and UUID which arrived in the SEND event are copied to any REPLY event that relates to this particular command.

REPLY events should be issued as follows : first a RCV event (**obligatory!**), one or more FDB events **(optional)**, and finally **(obligatory!)** either an ACK event or ERR in case of an error.

For example :

{"component":“name","comp_phys":“physical_name","command":"your_command","arg1":"","arg2":"",
"reply":"","reply type":"RCV","comp_type":"other","tick count":1380210404,"UUID":26481}

{"component":“name","comp_phys":“physical_name","command":"your_command","arg1":"","arg2":"",
"reply":"your feedback here","reply type":"FDB","comp_type":"other","tick
count":1380210404,"UUID":26481}

{"component":“name","comp_phys":“physical_name","command":"your_command","arg1":"","arg2":"",
"reply":"your final reply here","reply type":"ACK","comp_type":"other","tick
count":1380210404,"UUID":26481}

Acquila socket interface

Formatting:

REPLY events:

It is strongly recommended for commands that take long to execute (>1s) to issue FDB events at least once a second to avoid timeouts at the sending side.

Acquila socket interface

Component types:

Acquila currently features four component types : tube, camera, motor and other.

Implementing a tube, motor or camera component:

Any component that acts as a tube, motor or camera should conform to the predefined set of commands and corresponding replies for that component type. This guarantees integration in the Acquila environment, and functionality of the GUI which processes these replies to display the scanner status

Within Acquila, the reply content formatting depends on the command and whether it is FDB or ACK (RCV normally does not contain reply content). When the reply content is something simple such as a single value or word, the format is a simple string. When the content is a more complex data structure such as a tube status cluster, it is flattened to a pseudo-binary string. When implementing a component in another programming environment this cannot be directly reproduced. For that reason, a parser was implemented to parse the Acquila formatted string to a more readable XML format. An extensive list of commands and corresponding replies can be found in Appendix A.

Implementing another component:

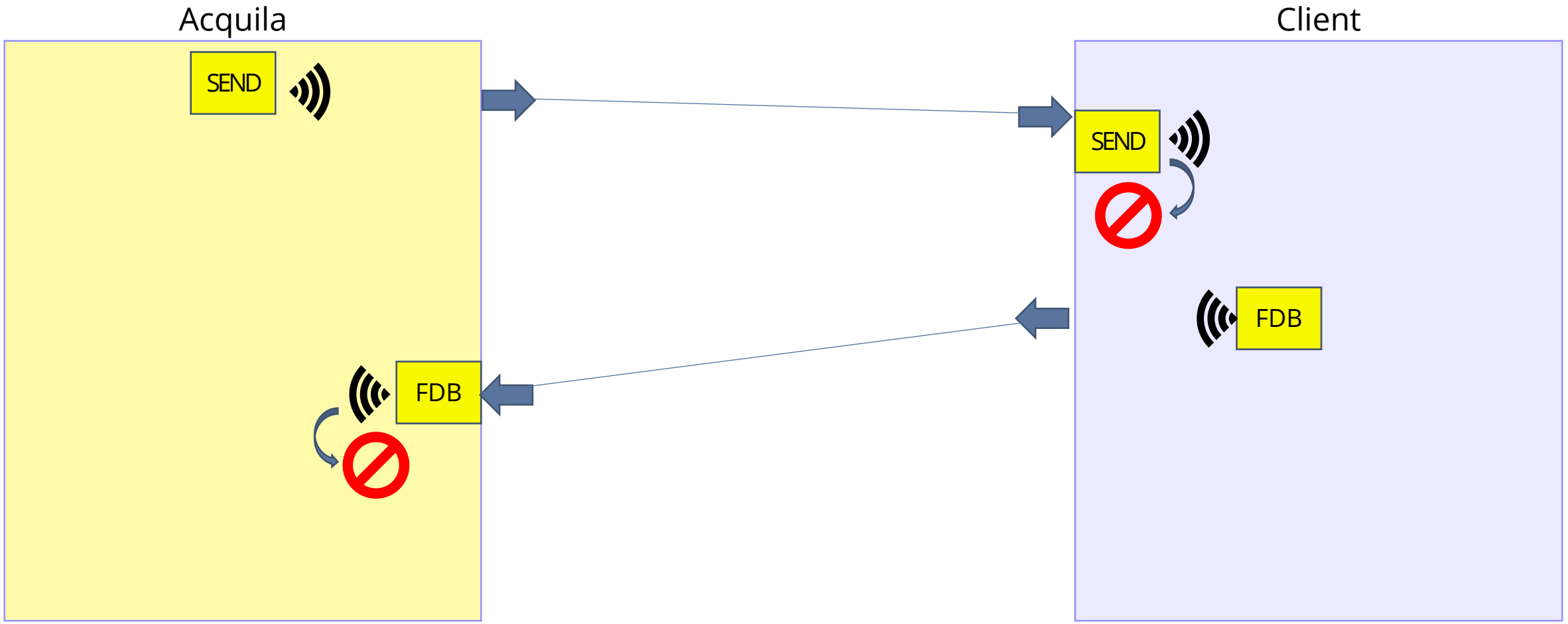
The developer is free to define commands and possible arguments or replies to enable the desired functionality of that component. It is clear however that Acquila or its integrated components cannot handle or interpret these events.

! The following component names are reserved for exclusive use by Acquila :

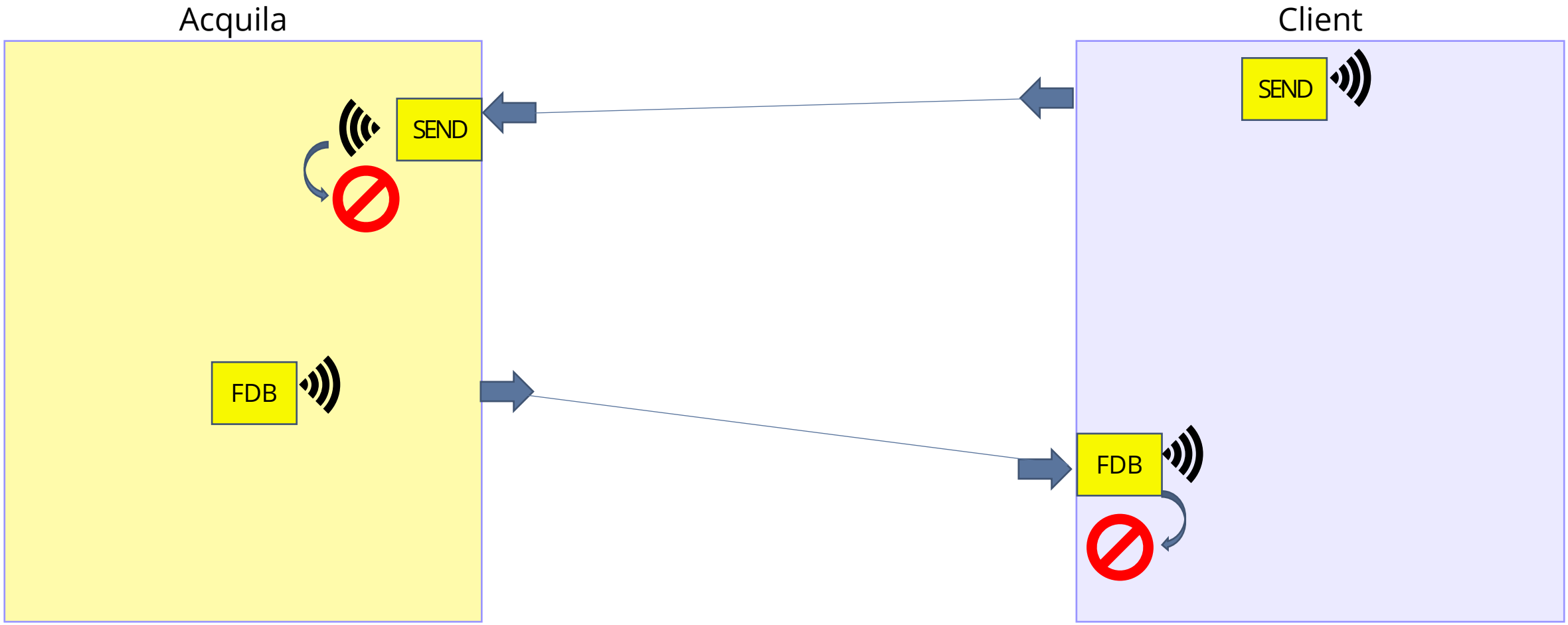
tube, camera, scanner, SC, mag*, rot*, tra*, ver*, *obj, *tube, *det

* stands for one or more characters

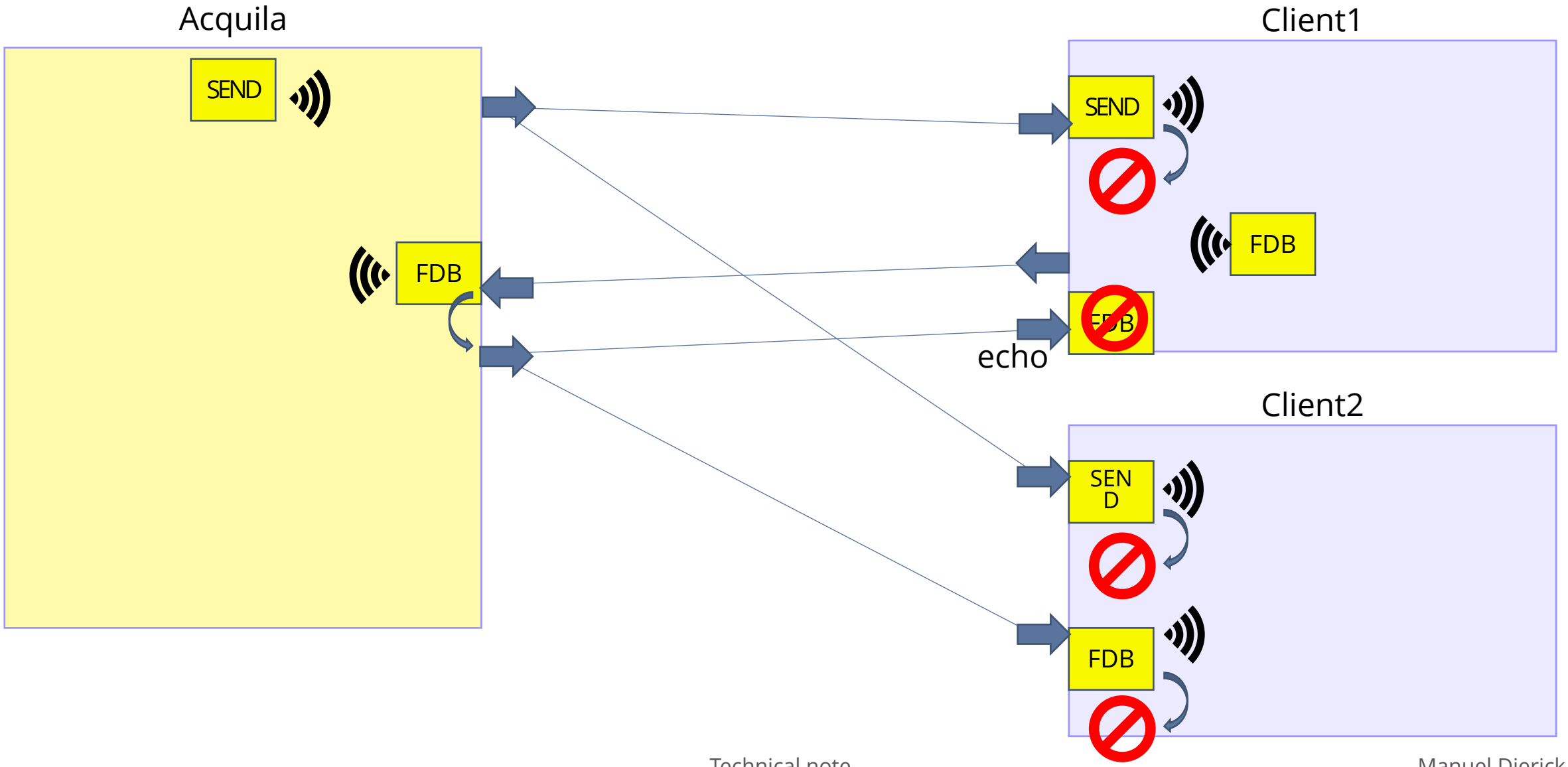
single server/single client mode: no echoes
blocklist contains events received through tunnel, both in server side tunnel and client side tunnel



single server/single client mode: no echoes
blocklist contains events received through tunnel, both in server side tunnel and client side tunnel

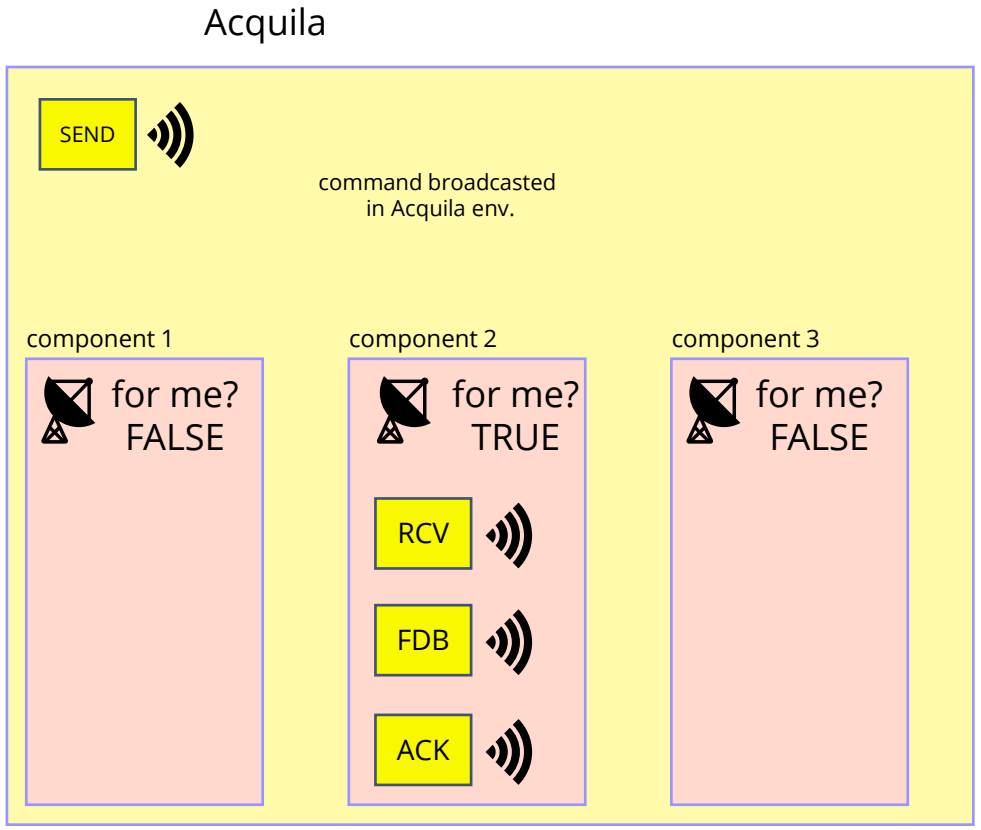


single server/multiple client mode: transparent server, client not only blocks events issued on behalf of tunnel but also echoes from events forwarded on tunnel

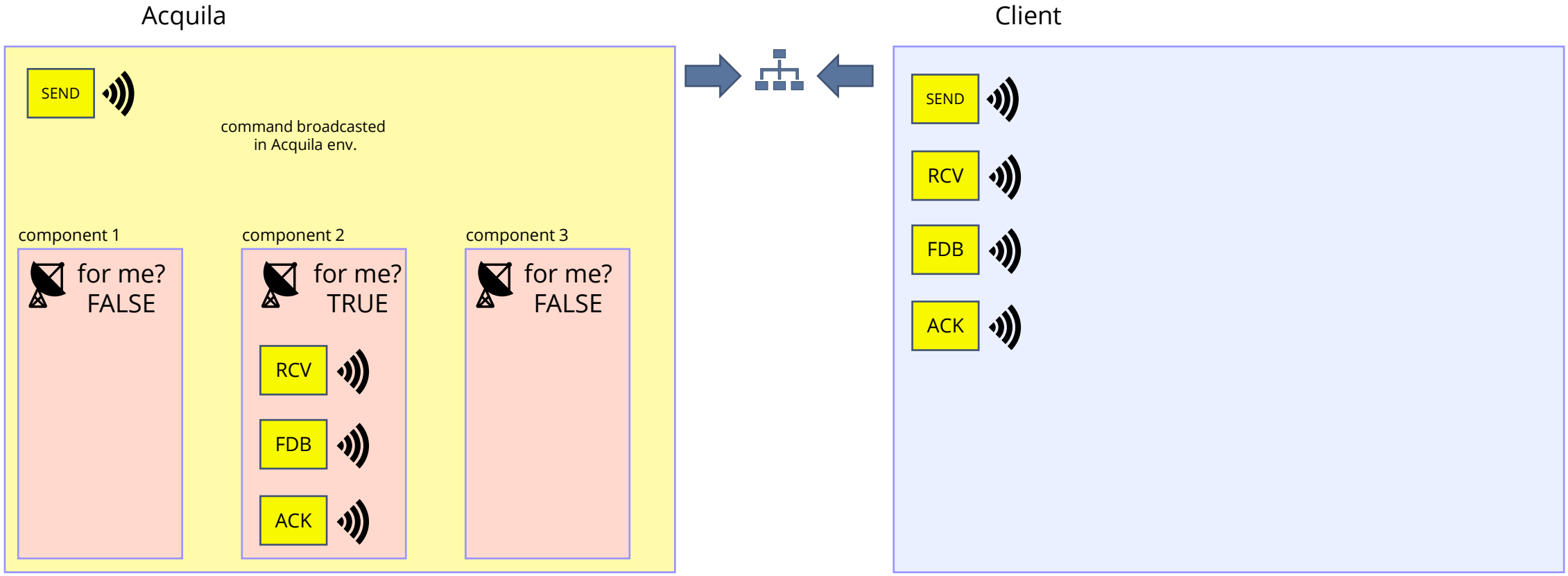


problem with REPLY events:
RCV, FDB and ACK are all events with same UUID.
Using the UUID alone as identifier on the blocklist will prevent FDB and ACK from getting out!
Temporary solution: adding tickcount and Reply type still risky when multiple FDB events are issued within same ms

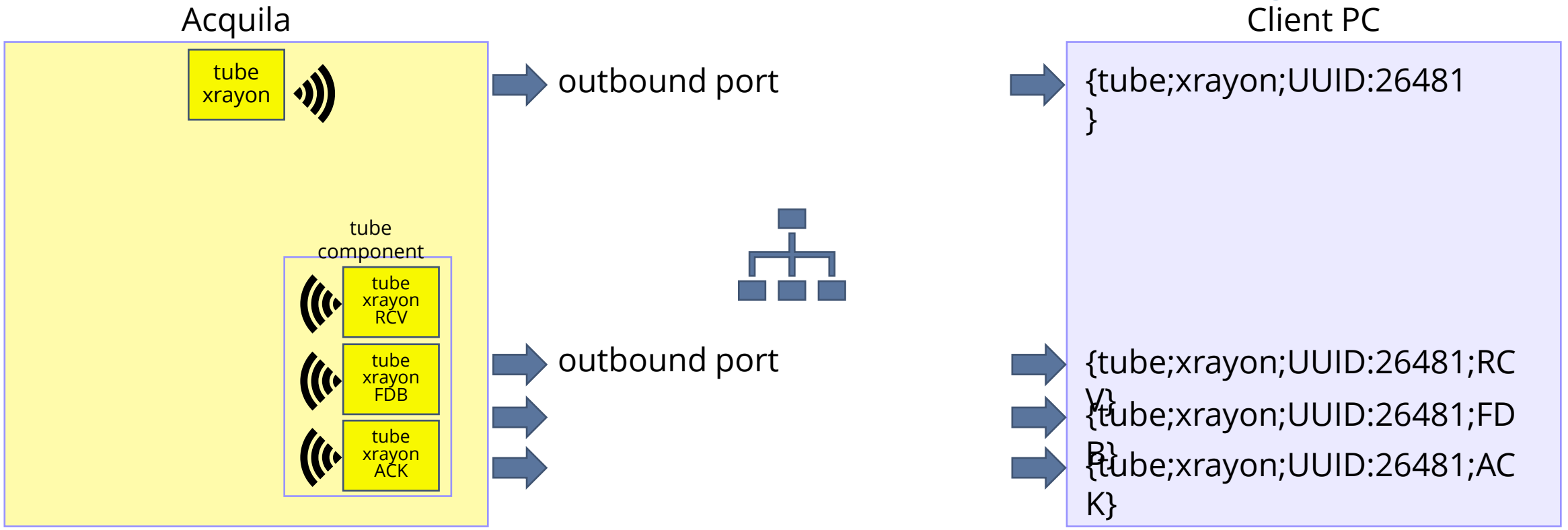
Acquila concept



Acquila concept

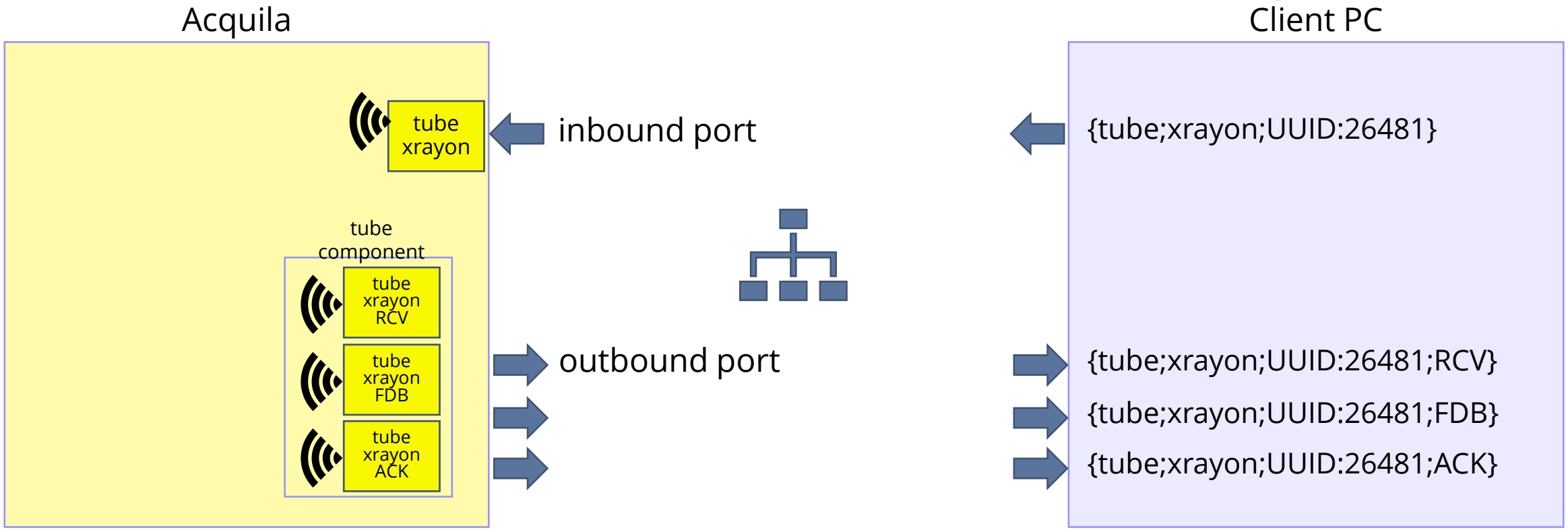


Typical example : listening in on what happens at scanner side



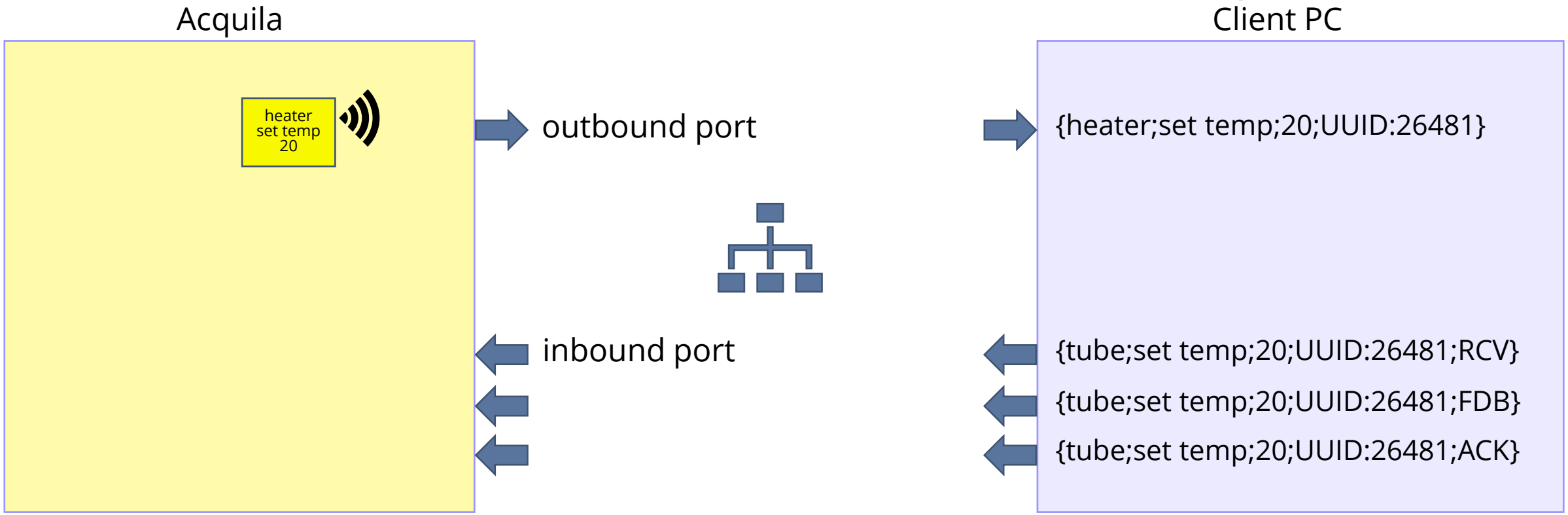
If the customer wants to react on certain events at the scanner side, say for example when the xray tube is switched ON he can subscribe to the outbound port on which Acquila publishes events (formatted as an XML string). The client will then hear the command being sent (e.g. when the operator clicks the xrayon button), and will hear the replies from the xraysource component. Typically this is first a confirmation of reception (RCV) , intermediate feedback to report on the progress of execution (FDB) and finally confirmation that execution finished (ACK).

Typical example : sending a command to the scanner side



If the customer wants to send a command to the scanner, say for example "tube xrayon" he can write a command to the inbound port, appropriately formatted as an JSON string. This will be broadcasted by Acquila and handled by the addressed component.
The client will then hear the replies on the outbound port. For a list of possible commands please contact XRE support.

Typical example : sending a command to a client application



If the customer has a client application that can perform actions, say for example a heater to control the temperature of his sample, and he wants Acquila to interface with his client app. In that case the customer can define a command set his app understands and include those commands in Acquila scripts. When the Acquila script runner encounters such commands they will be broadcasted as any other internal Acquila command and forwarded on the ZMQ port. The client will pick this up, execute the command and return appropriately formatted replies so Acquila can interpret the progress.

Acquila socket interface

Manuel Dierick