The problem

In control systems with communication, errors can happen. A transmission error or a receiver that is busy could be some of the explanations of why a communication could go wrong. In VIGO6, errors in communication result in invalidation of the data being received to avoid half packets of valid data. Communication in VIGO6 is whenever a component attempts to run a method, write data or fetch data, from outside the scope of the component itself. Even if the two components are located inside the same device. As an example, see the figure below.

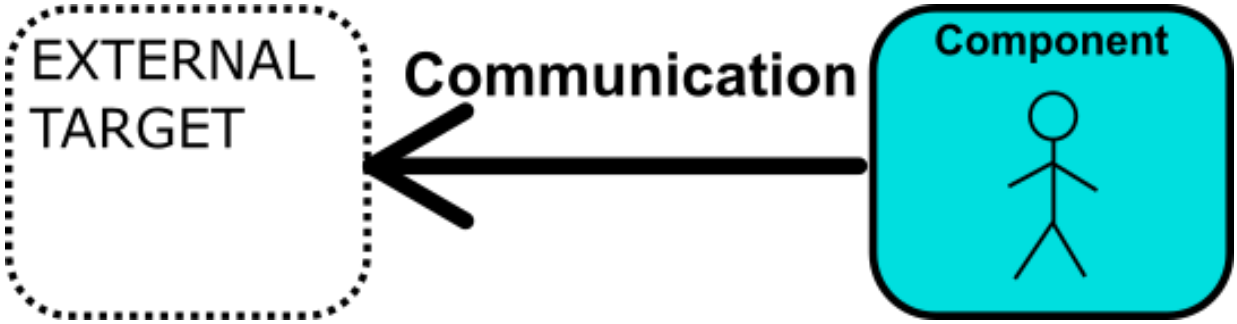

Figure 1 Communication example.

In the above example, the component under development is programmed to communicate with another component. This communication is done through a connector, which means that the external target can be located in a device anywhere across the world, seen in relation to the component being developed, or it can be located in the same device as the component being developed. In both cases, the communication can fail, and even though components located in the same device are less prone to fail, they still can.

The solution

The above description explains that it is very important to handle errors, as sporadic failures can happen. Luckily, COPP comes with a tool to make error handling easy. The name of the solution is called Method_state, which is located under system variables. Method_state can have the values OK or Error. Once Method_state turns from OK to Error, it stays in the error state until cleared by the programmer in the code. An example of usage can be seen below:

```
// Make sure that method state is OK
Method_state := OK

// Write data to target
Connector.External_target.Register := My_data

// Now check the method state
IF Method_state = OK
 THEN
  // The data was written to target successfully.
 ELSE // Error
  // Something went wrong while writing to target
  // Handle the communication error
```

Code Example 1 Method state usage.

Thus the recipe to use method state is:

A. Set method state to OK just before doing the external communication
B. Do the external communication
C. Check method state:
    a. If OK, handle a successful transaction
    b. If Error, handle a failed transaction

The value of method state can be used to see the full transaction. If in the above code example a method is called, which calls a second method that in turn calls yet a third method, and the last called method fails or invokes a method error, the method state will be error in all the methods that are part of calling another method. Let's look at another example:

The code example below is a component calling another component's method.

```
// Make sure that method state is OK
Method_state := OK

// Run method in external target
Connector.External_target.Run_method()

// Now check the method state
IF Method_state = OK
 THEN
  // The method was run successfully.
 ELSE // Error or Invalid
  // Something went wrong while running the method
  // Handle the communication error
```

Code Example 2 First method calling another component's method.

```
// Make sure that method state is OK
Method_state := OK

// Run method in external target
Connector.External_target_2.Run_method_2()

// Now check the method state
IF Method_state = OK
 THEN
  // The method was run successfully.
 ELSE // Error
  // Something went wrong while running the method
  // Handle the communication error
```

Code Example 3 Second method calling another component's method.

```
// Make sure that method state is OK
Method_state := OK

// Write data to target
Connector.DB_reg[APPEND] := My_data

// Now check the method state
IF Method_state = OK
 THEN
  // The data was written to target successfully.
 ELSE // Error
  // Something went wrong while writing to target
  // Handle the communication error
```

Code Example 4 Writing to a database.

By following the three code examples above, the first example calls a method in an external target. The method in the external target in turn calls a method in another external target. The third method attempts to write data to a database. If at any point any method is not executed correctly, the calling method will get a method error. If the last code example fails to write data to the database, the calling method in Code Example 3 will get a method error, which in turn will let Code Example 2 get a method error, meaning that all the levels know that something went wrong. This is why we know that Method_state = OK implicitly means that EVERYTHING went as it should.

A state machine will often require additional states to handle errors, operator intervention and restart.

When doing Boolean logic, invalid can cause Method_state errors. See example below:

```
// Make sure that method state is OK
Method_state := OK

// Is my variable true? If it is invalid, method_state is error
IF My_Variable = True
 THEN
  // something
 ELSE //False or Invalid
  // something else

// Now check the method state
IF Method_state = OK
 THEN
  // My_variable was not invalid
 ELSE // Error
  // My variable was invalid
```

Code Example 5 Method state to error in Boolean logic

In cases like above, it is a general good practice to set method_state to OK in the ELSE clause of the first if statement checking “My_Variable”.