Automation of the Warm magnet Interlock Controller Commissioning (WIC)

BALLEJOS Lilian

22 / 08 / 2023

Plan

- **Context**
  - Short presentation of what is a WIC system
  - What elements should be tested in a WIC
- **What has been done**
  - Algorithm created with MOMPO Richard and DUTRUEL Aymeric
  - CCDA and PyCCDA extension for WIC commissioning
  - Recovery of missing relationships
  - Python code to apply the different algorithms
  - Creating a GUI with PyQt to meet user needs (1/2)
- **What still needs to be done**
  - Add the missing information
  - Solve both problems related to Power Converter
  - Incorporate the WIC commissioning into AccTesting

Short presentation of what is a WIC system (1/2)

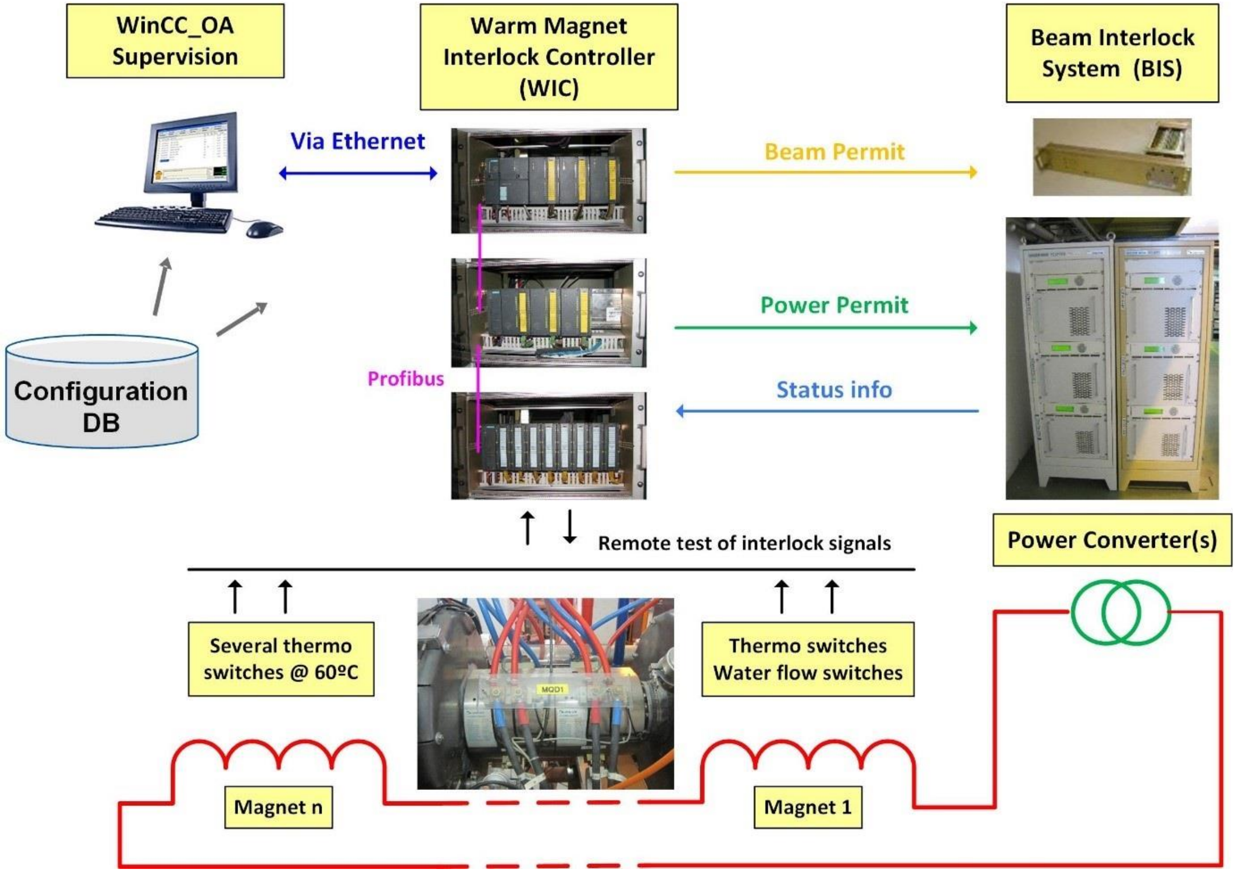
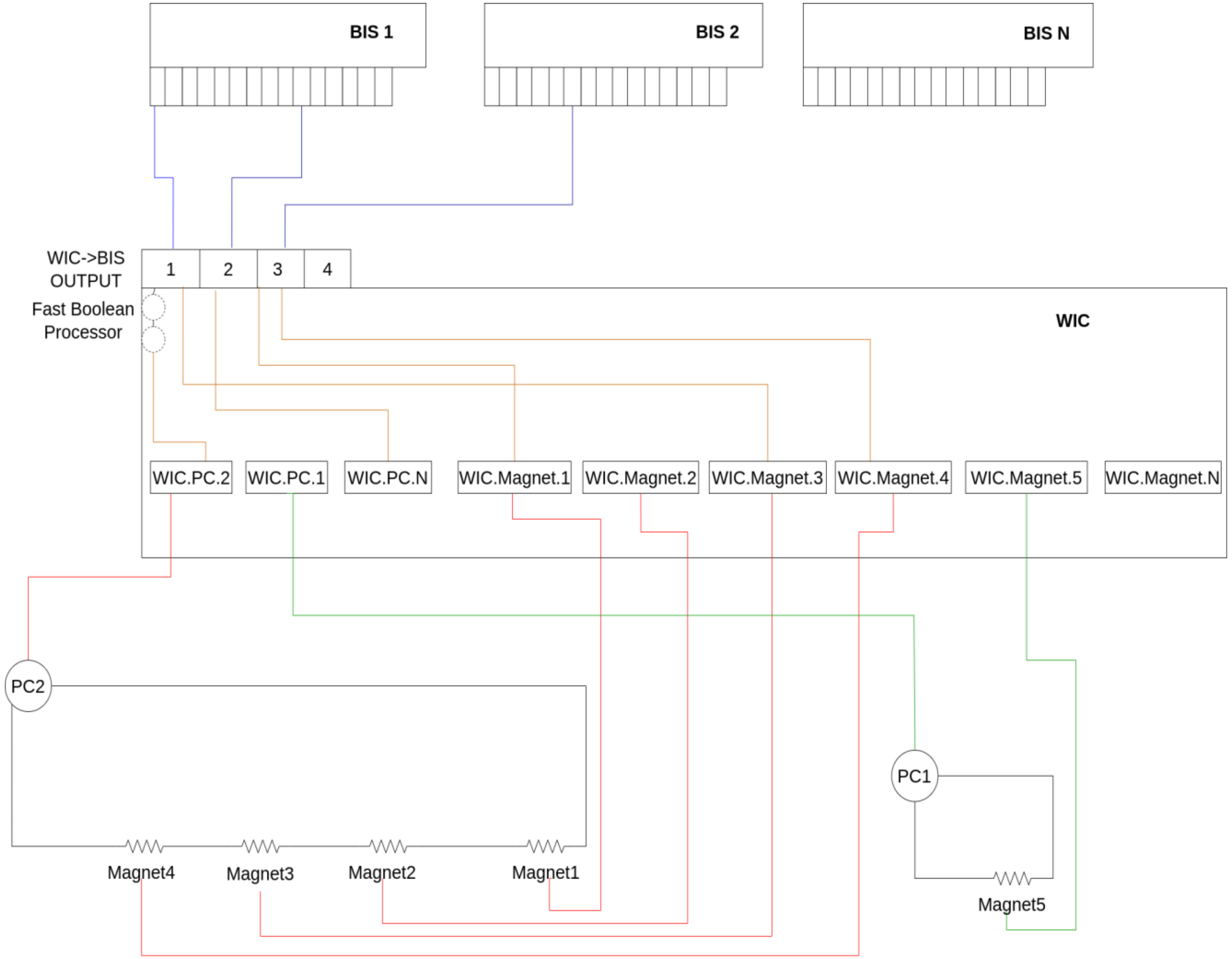

WinCC_OA Supervision
Warm Magnet Interlock Controller (WIC)
Beam Interlock System (BIS)
Via Ethernet
Beam Permit
Power Permit
Status info
Profibus
Configuration DB
Remote test of interlock signals
Power Converter(s)
Several thermo switches @ 60ºC
Thermo switches Water flow switches
Magnet n
Magnet 1

Diagram from the WIC team website representing a WIC and all its entities

Magnet covered by a WIC system

Short presentation of what is a WIC system (2/2)

- 0 to 4 possible connections to BIS systems
- **1-1 relationship** between a WIC signal and the device it covers
- Each WIC signal can be connected or not to one of the 4 WIC-> BIS outputs
- On the WIC->BIS n°1 output only, there can be 0, 1 or 2 Fast boolean processors
- 1 Power converter can power multiple magnets
- 1 magnet can be powered by multiple Power Converters

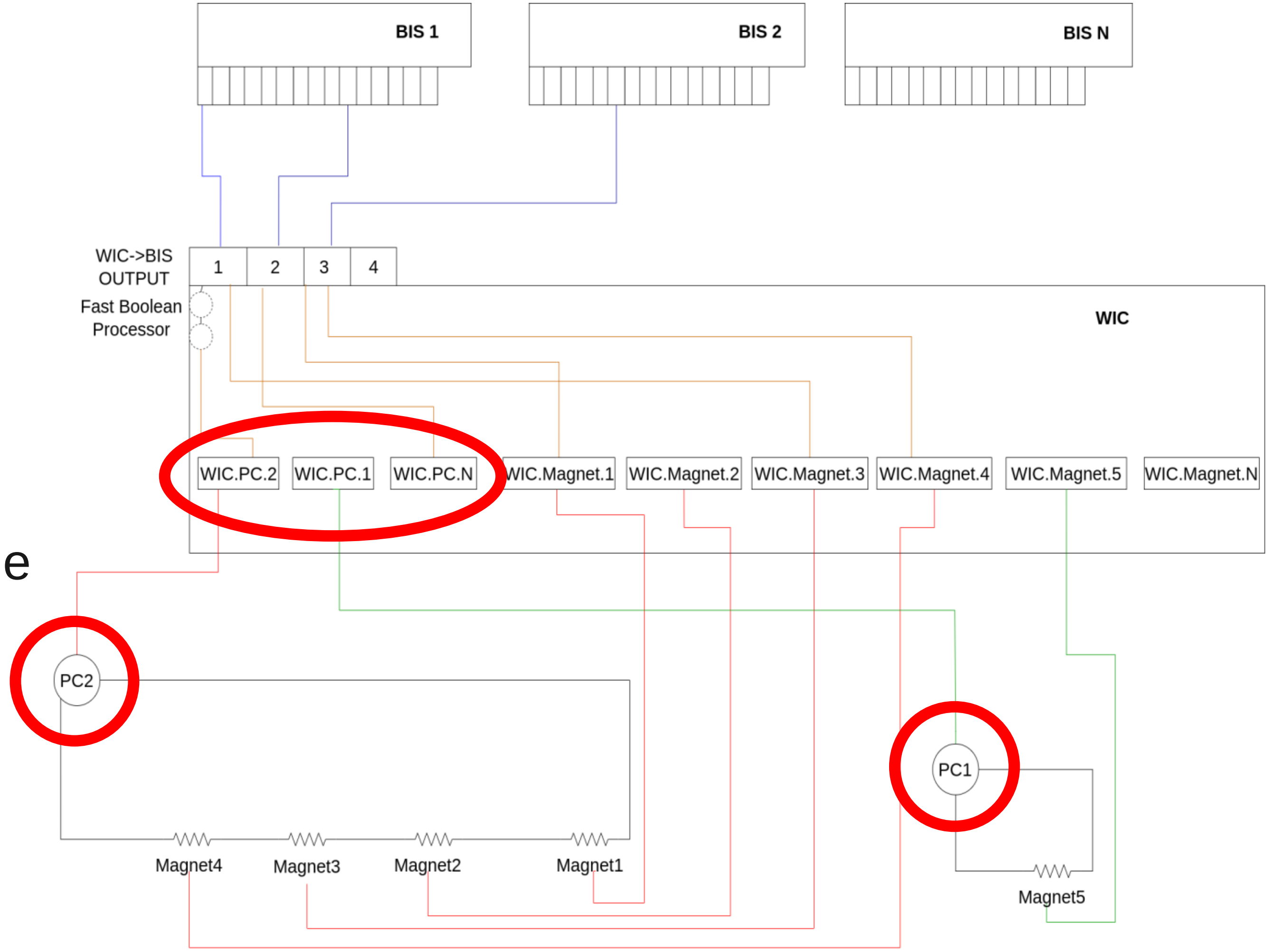


What elements should be tested in a WIC (1/3)

WIC Power Converter Signals (WIC PC)

- A WIC PC signal must **receive a fault signal from a Power Converter** if this one is in an internal fault

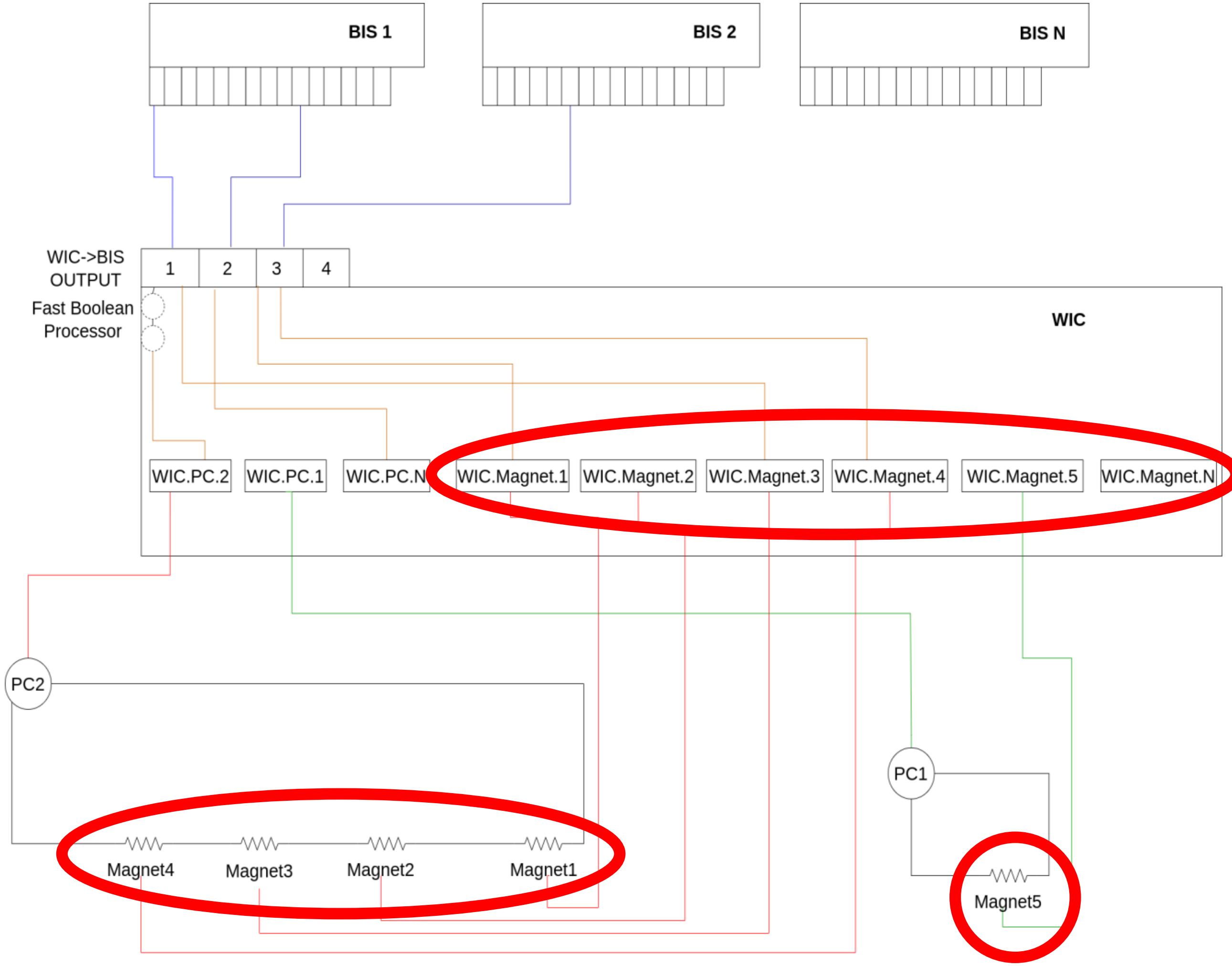
- A WIC PC signal must in case of a detected error and connection to a WIC->BIS output, send a **USER_PERMIT at False** to the connected BIS



What elements should be tested in a WIC (2/3)

WIC Magnet Signals (WIC Magnet)

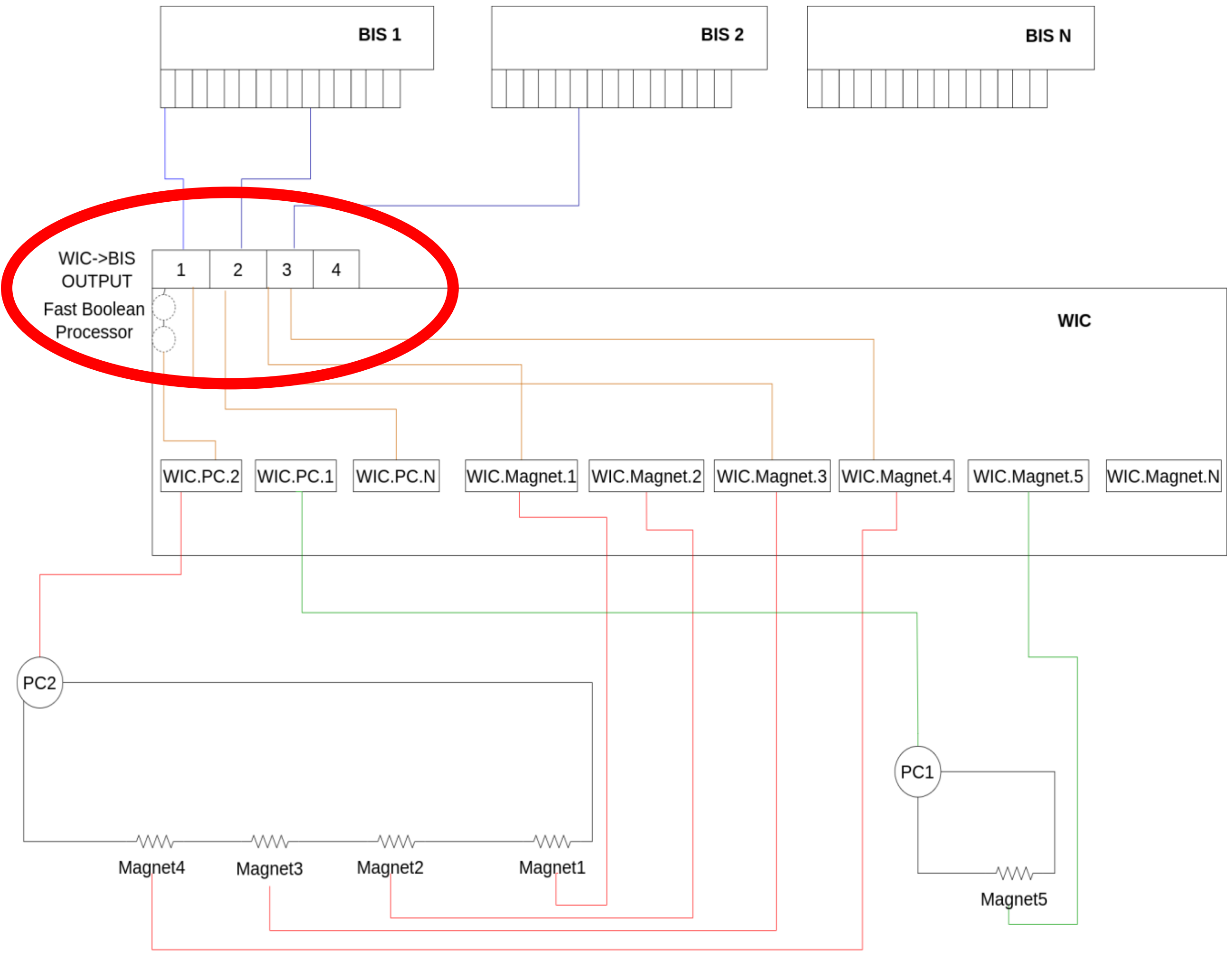
- **A WIC magnet signal must receive a fault signal from a magnet** if this one is in a faulty state
- A WIC magnet signal must put all **Power Converter connected** to the covered magnet in **external fault** (FAST_ABORT)
- For safety, the error sent to a Power Converter is accomplished by **two independent operating relays connected in series**
- A WIC magnet signal must in case of a detected error and connection to a WIC->BIS output, send a **USER_PERMIT at False** to the connected BIS



What elements should be tested in a WIC (3/3)

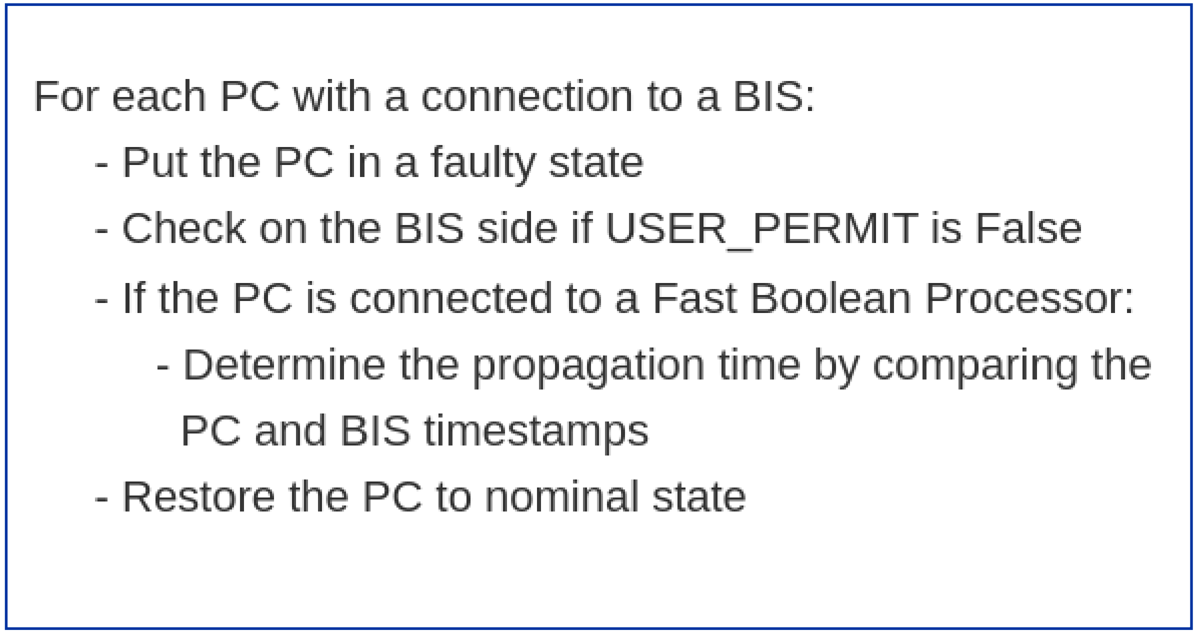
WIC->BIS relations

- If there is a fast boolean processor on output 1, the following rule is respected: **more than 12 Power Converters covered by the WIC: 2 fast boolean processors, otherwise 1**
- Reaction time with a fast boolean processor is around **6µs** and around **50ms** without
- Only USER_PERMIT from WIC Power Converter signals use fast boolean processors
- For security, the USER_PERMIT sent to the BIS are duplicated



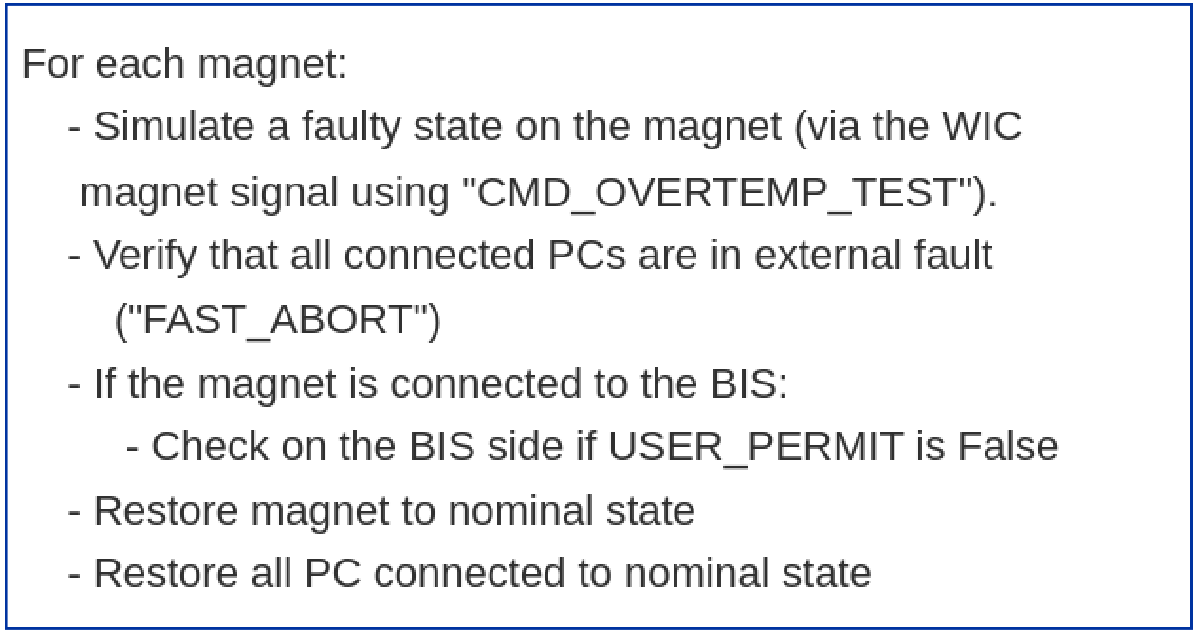
Algorithm created with MOMPO Richard and DUTRUEL Aymeric (1/2)

Stage 1: Checking the Power Converters



```
For each PC with a connection to a BIS:
    - Put the PC in a faulty state
    - Check on the BIS side if USER_PERMIT is False
    - If the PC is connected to a Fast Boolean Processor:
        - Determine the propagation time by comparing the
          PC and BIS timestamps
    - Restore the PC to nominal state
```

Stage 2: Checking the magnets



```
For each magnet:
    - Simulate a faulty state on the magnet (via the WIC
     magnet signal using "CMD_OVERTEMP_TEST").
    - Verify that all connected PCs are in external fault
       ("FAST_ABORT")
    - If the magnet is connected to the BIS:
        - Check on the BIS side if USER_PERMIT is False
    - Restore magnet to nominal state
    - Restore all PC connected to nominal state
```

Algorithm created with MOMPO Richard and DUTRUEL Aymeric (2/2)

Stage 3: Checking USER_PERMIT redundancy

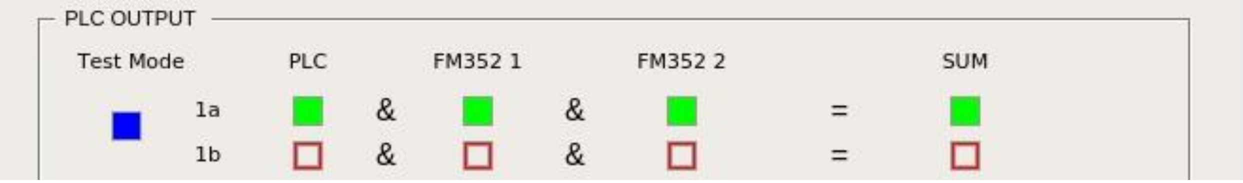
For each magnet:

- Simulate a faulty state on the magnet (via the WIC magnet signal using "CMD_OVERTEMP_TEST").
- Verify that all connected PCs are in external fault ("FAST_ABORT")
- If the magnet is connected to the BIS:
    - Check on the BIS side if USER_PERMIT is False
- Restore magnet to nominal state
- Restore all PC connected to nominal state

Stage 4: Checking relays redundancy

For each WIC signal PC:

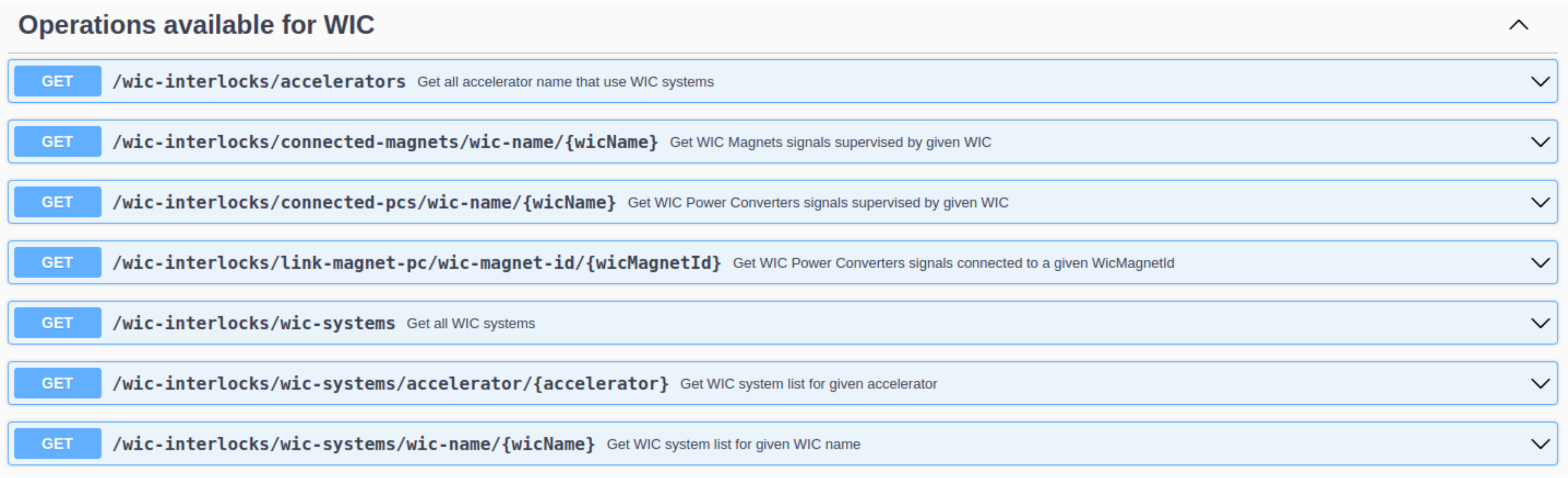
- Open relay A on WIC side connected to the covered PC
- See if the PC covered is in external fault (FAST_ABORT)
- Close relay A
- Restore the PC to nominal state
- Do the same with relay B



PLC OUTPUT

Test Mode		PLC		FM352 1		FM352 2		SUM
🟦	1a	🟩	&	🟩	&	🟩	=	🟩
	1b	□	&	□	&	□	=	□

CCDA and PyCCDA extension for WIC commissioning



Operations available for WIC

GET `/wic-interlocks/accelerators` Get all accelerator name that use WIC systems

GET `/wic-interlocks/connected-magnets/wic-name/{wicName}` Get WIC Magnets signals supervised by given WIC

GET `/wic-interlocks/connected-pcs/wic-name/{wicName}` Get WIC Power Converters signals supervised by given WIC

GET `/wic-interlocks/link-magnet-pc/wic-magnet-id/{wicMagnetId}` Get WIC Power Converters signals connected to a given WicMagnetId

GET `/wic-interlocks/wic-systems` Get all WIC systems

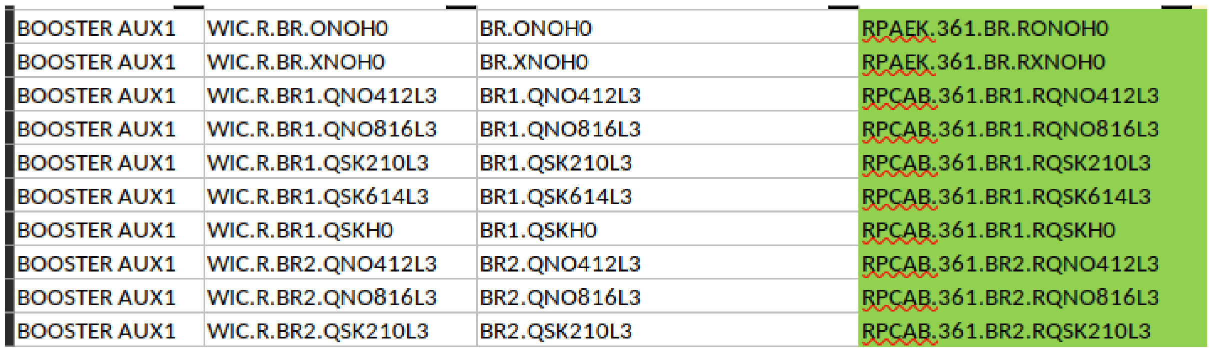
GET `/wic-interlocks/wic-systems/accelerator/{accelerator}` Get WIC system list for given accelerator

GET `/wic-interlocks/wic-systems/wic-name/{wicName}` Get WIC system list for given WIC name

- Modification of views to retrieve the necessary information for the commissioning of the WIC from the CCDB
- Creation of endpoints to retrieve this information via the CCDA
- Updating the PyCCDA Python module to cover the newly created endpoints
- Management of these available endpoints start from PyCCDA release 1.18.2

Recovery of missing relationships

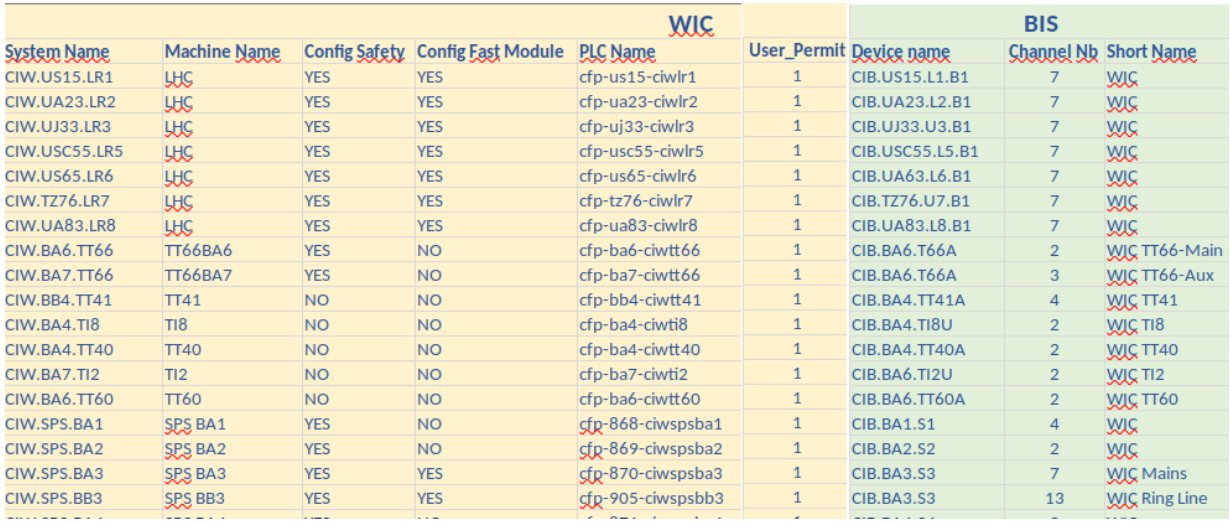
Relation between WIC PC signals and covered PCs



BOOSTER AUX1	WIC.R.BR.ONOH0	BR.ONOH0	RPAEK.361.BR.RONOH0
BOOSTER AUX1	WIC.R.BR.XNOH0	BR.XNOH0	RPAEK.361.BR.RXNOH0
BOOSTER AUX1	WIC.R.BR1.QNO412L3	BR1.QNO412L3	RPCAB.361.BR1.RQNO412L3
BOOSTER AUX1	WIC.R.BR1.QNO816L3	BR1.QNO816L3	RPCAB.361.BR1.RQNO816L3
BOOSTER AUX1	WIC.R.BR1.QSK210L3	BR1.QSK210L3	RPCAB.361.BR1.RQSK210L3
BOOSTER AUX1	WIC.R.BR1.QSK614L3	BR1.QSK614L3	RPCAB.361.BR1.RQSK614L3
BOOSTER AUX1	WIC.R.BR1.QSKH0	BR1.QSKH0	RPCAB.361.BR1.RQSKH0
BOOSTER AUX1	WIC.R.BR2.QNO412L3	BR2.QNO412L3	RPCAB.361.BR2.RQNO412L3
BOOSTER AUX1	WIC.R.BR2.QNO816L3	BR2.QNO816L3	RPCAB.361.BR2.RQNO816L3
BOOSTER AUX1	WIC.R.BR2.QSK210L3	BR2.QSK210L3	RPCAB.361.BR2.RQSK210L3

- Recovered approximately 90% of the 1279 relationships to be found using a Python script that searches for correlations between the names of WIC PC signals and PCs.
- Rest completed by hand by Richard Mompo
- Identified specific cases to be addressed

Relation between WIC systems and BIS systems

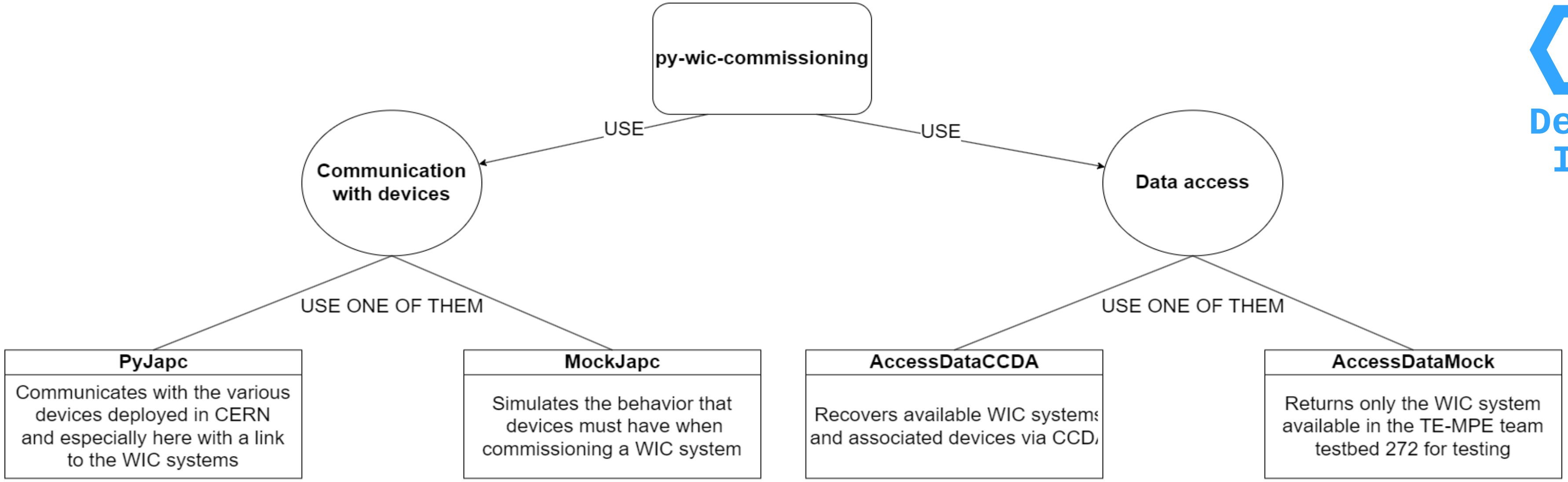


WIC						BIS		
System Name	Machine Name	Config Safety	Config Fast Module	PLC Name	User_Permit	Device name	Channel Nb	Short Name
CIW.US15.LR1	LHC	YES	YES	cfp-us15-ciwlr1	1	CIB.US15.L1.B1	7	WIC
CIW.UA23.LR2	LHC	YES	YES	cfp-ua23-ciwlr2	1	CIB.UA23.L2.B1	7	WIC
CIW.UJ33.LR3	LHC	YES	YES	cfp-uj33-ciwlr3	1	CIB.UJ33.U3.B1	7	WIC
CIW.USC55.LR5	LHC	YES	YES	cfp-usc55-ciwlr5	1	CIB.USC55.L5.B1	7	WIC
CIW.US65.LR6	LHC	YES	YES	cfp-us65-ciwlr6	1	CIB.UA63.L6.B1	7	WIC
CIW.TZ76.LR7	LHC	YES	YES	cfp-tz76-ciwlr7	1	CIB.TZ76.U7.B1	7	WIC
CIW.UA83.LR8	LHC	YES	YES	cfp-ua83-ciwlr8	1	CIB.UA83.L8.B1	7	WIC
CIW.BA6.TT66	TT66BA6	YES	NO	cfp-ba6-ciwtt66	1	CIB.BA6.T66A	2	WIC TT66-Main
CIW.BA7.TT66	TT66BA7	YES	NO	cfp-ba7-ciwtt66	1	CIB.BA6.T66A	3	WIC TT66-Aux
CIW.BB4.TT41	TT41	NO	NO	cfp-bb4-ciwtt41	1	CIB.BA4.TT41A	4	WIC TT41
CIW.BA4.TI8	TI8	NO	NO	cfp-ba4-ciwti8	1	CIB.BA4.TI8U	2	WIC TI8
CIW.BA4.TT40	TT40	NO	NO	cfp-ba4-ciwtt40	1	CIB.BA4.TT40A	2	WIC TT40
CIW.BA7.TI2	TI2	NO	NO	cfp-ba7-ciwti2	1	CIB.BA6.TI2U	2	WIC TI2
CIW.BA6.TT60	TT60	NO	NO	cfp-ba6-ciwtt60	1	CIB.BA6.TT60A	2	WIC TT60
CIW.SPS.BA1	SPS BA1	YES	NO	cfp-868-ciwspsba1	1	CIB.BA1.S1	4	WIC
CIW.SPS.BA2	SPS BA2	YES	NO	cfp-869-ciwspsba2	1	CIB.BA2.S2	2	WIC
CIW.SPS.BA3	SPS BA3	YES	YES	cfp-870-ciwspsba3	1	CIB.BA3.S3	7	WIC Mains
CIW.SPS.BB3	SPS BB3	YES	YES	cfp-905-ciwspsbb3	1	CIB.BA3.S3	13	WIC Ring Line

- Filled by hand by Richard Mompo
- Indicates for each WIC system which WIC ->BIS output is used
- Indicates which channel of which BIS is connected to each WIC output.

Python code to apply the different algorithms

- Used the Python module **PyJapc** to communicate with different devices related to WIC systems
- Used the Python **PyCCDA** module for data retrieval related to WIC systems
- Researched and documented a way to use dependency injection in Python to manage the different behaviours created: module **Dependency Injector**
- I created for each behaviour implemented a simulated version to perform the necessary tests
- Performed all tests on a WIC system present in the testbed in 272



PyJapc
Communicates with the various devices deployed in CERN and especially here with a link to the WIC systems

MockJapc
Simulates the behavior that devices must have when commissioning a WIC system

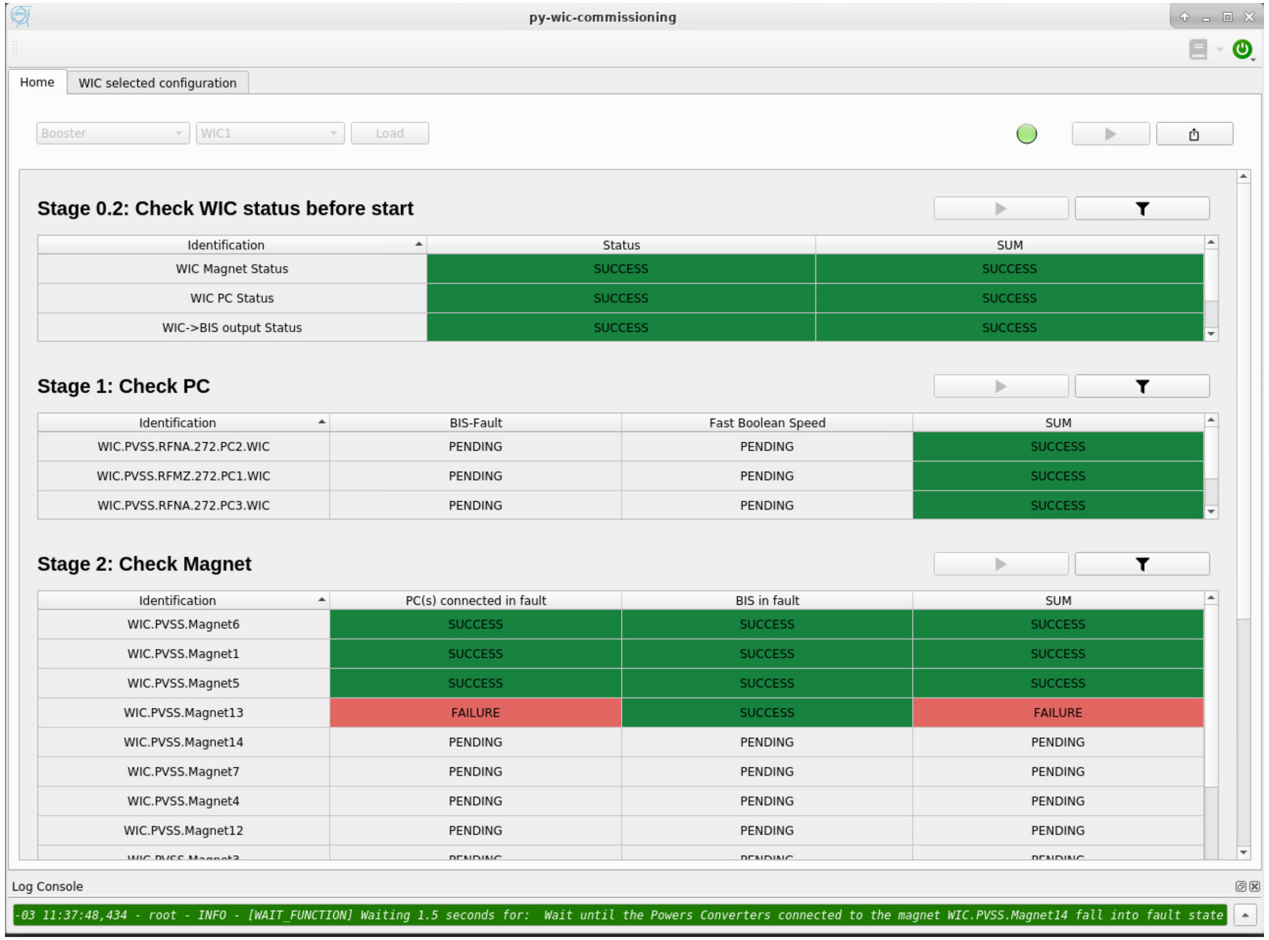
AccessDataCCDA
Recovers available WIC systems and associated devices via CCDA

AccessDataMock
Returns only the WIC system available in the TE-MPE team testbed 272 for testing

Dependency Injector

Creating a GUI with PyQt to meet user needs (1/2)

Appearance of the GUI



py-wic-commissioning

Home | WIC selected configuration

Booster | WIC1 | Load

Stage 0.2: Check WIC status before start

Identification	Status	SUM
WIC Magnet Status	SUCCESS	SUCCESS
WIC PC Status	SUCCESS	SUCCESS
WIC->BIS output Status	SUCCESS	SUCCESS

Stage 1: Check PC

Identification	BIS-Fault	Fast Boolean Speed	SUM
WIC.PVSS.RFNA.272.PC2.WIC	PENDING	PENDING	SUCCESS
WIC.PVSS.RFMZ.272.PC1.WIC	PENDING	PENDING	SUCCESS
WIC.PVSS.RFNA.272.PC3.WIC	PENDING	PENDING	SUCCESS

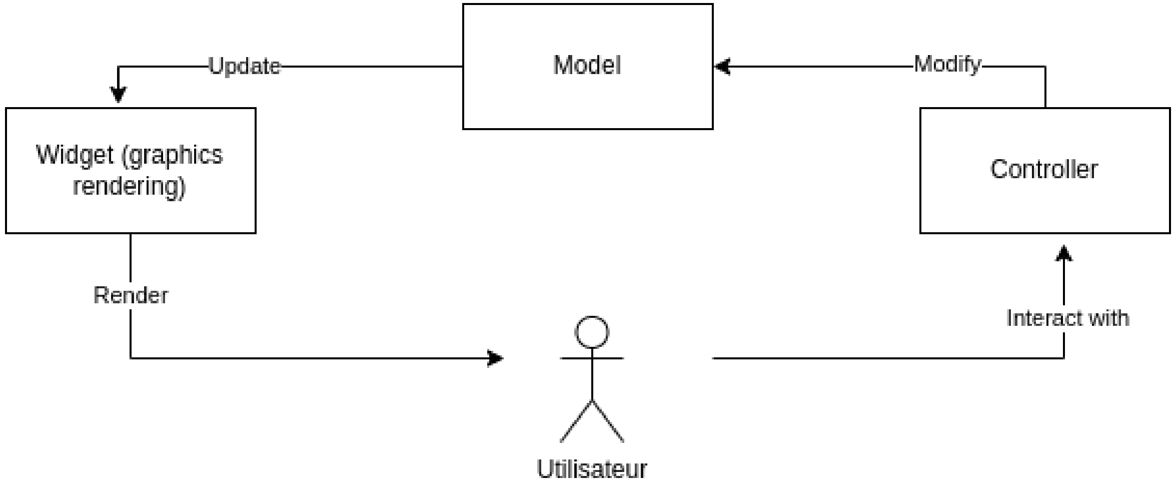
Stage 2: Check Magnet

Identification	PC(s) connected in fault	BIS in fault	SUM
WIC.PVSS.Magnet6	SUCCESS	SUCCESS	SUCCESS
WIC.PVSS.Magnet1	SUCCESS	SUCCESS	SUCCESS
WIC.PVSS.Magnet5	SUCCESS	SUCCESS	SUCCESS
WIC.PVSS.Magnet13	FAILURE	SUCCESS	FAILURE
WIC.PVSS.Magnet14	PENDING	PENDING	PENDING
WIC.PVSS.Magnet7	PENDING	PENDING	PENDING
WIC.PVSS.Magnet4	PENDING	PENDING	PENDING
WIC.PVSS.Magnet12	PENDING	PENDING	PENDING
WIC.PVSS.Magnet3	PENDING	PENDING	PENDING

Log Console

```
-03 11:37:48,434 - root - INFO - [WAIT_FUNCTION] Waiting 1.5 seconds for:  Wait until the Powers Converters connected to the magnet WIC.PVSS.Magnet14 fall into fault state
```

Logic behind the GUI



Model

Widget (graphics rendering)

Controller

Utilisateur

Update

Modify

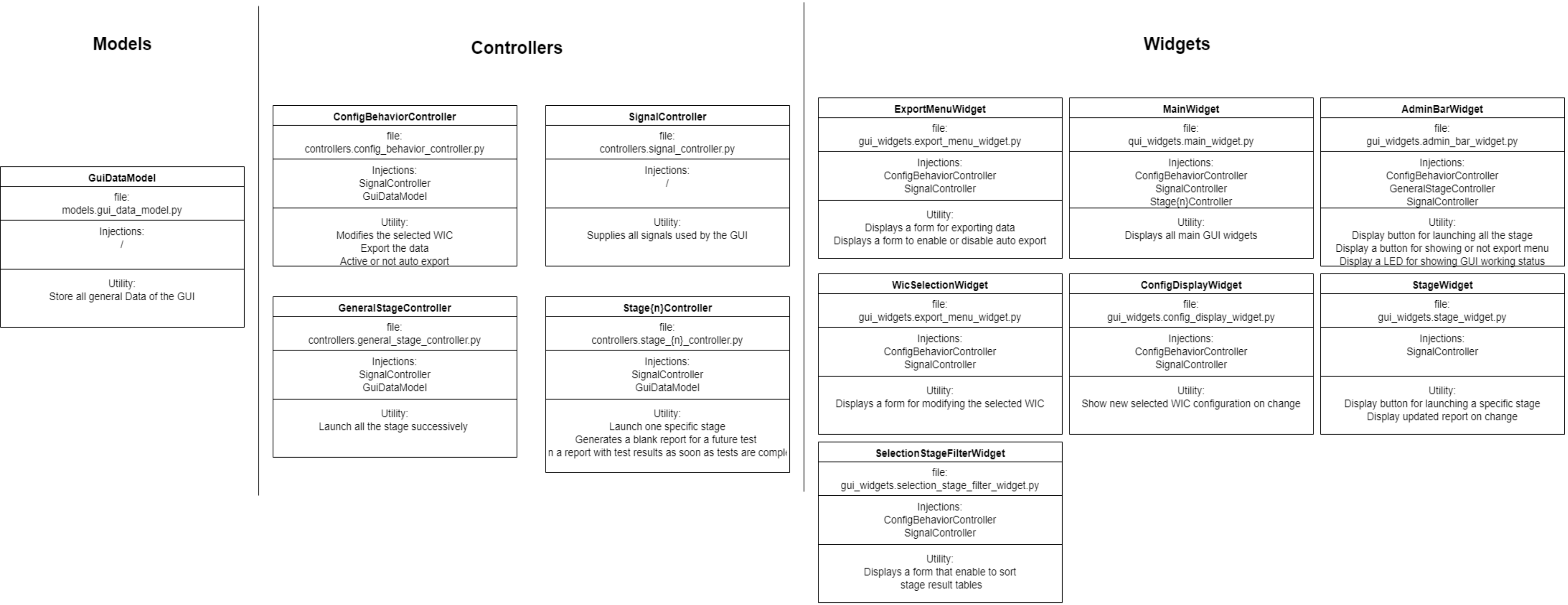
Render

Interact with

- Researched the best way to organize a GUI in PyQt
- Documented each of my choices in the code for the future of the team
- Shared my new discoveries about PyQt to the team

Creating a GUI with PyQt to meet user needs (2/2)

What has been implemented



Models

GuiDataModel
file:
models.gui_data_model.py
Injections:
/
Utility:
Store all general Data of the GUI

Controllers

ConfigBehaviorController
file:
controllers.config_behavior_controller.py
Injections:
SignalController
GuiDataModel
Utility:
Modifies the selected WIC
Export the data
Active or not auto export

SignalController
file:
controllers.signal_controller.py
Injections:
/
Utility:
Supplies all signals used by the GUI

GeneralStageController
file:
controllers.general_stage_controller.py
Injections:
SignalController
GuiDataModel
Utility:
Launch all the stage successively

Stage{n}Controller
file:
controllers.stage_{n}_controller.py
Injections:
SignalController
GuiDataModel
Utility:
Launch one specific stage
Generates a blank report for a future test
n a report with test results as soon as tests are compl

Widgets

ExportMenuWidget
file:
gui_widgets.export_menu_widget.py
Injections:
ConfigBehaviorController
SignalController
Utility:
Displays a form for exporting data
Displays a form to enable or disable auto export

MainWidget
file:
gui_widgets.main_widget.py
Injections:
ConfigBehaviorController
SignalController
Stage{n}Controller
Utility:
Displays all main GUI widgets

AdminBarWidget
file:
gui_widgets.admin_bar_widget.py
Injections:
ConfigBehaviorController
GeneralStageController
SignalController
Utility:
Display button for launching all the stage
Display a button for showing or not export menu
Display a LED for showing GUI working status

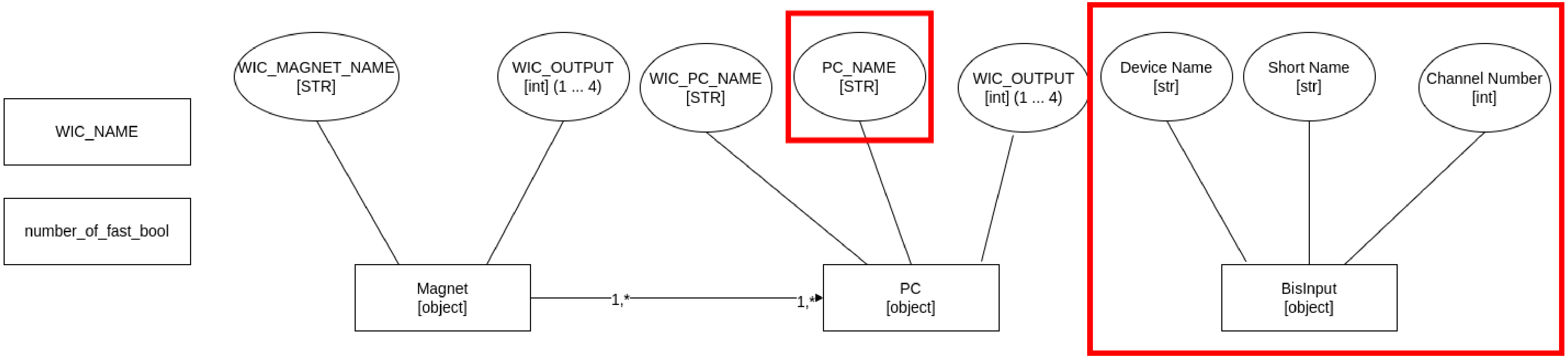
WicSelectionWidget
file:
gui_widgets.export_menu_widget.py
Injections:
ConfigBehaviorController
SignalController
Utility:
Displays a form for modifying the selected WIC

ConfigDisplayWidget
file:
gui_widgets.config_display_widget.py
Injections:
ConfigBehaviorController
SignalController
Utility:
Show new selected WIC configuration on change

StageWidget
file:
gui_widgets.stage_widget.py
Injections:
SignalController
Utility:
Display button for launching a specific stage
Display updated report on change

SelectionStageFilterWidget
file:
gui_widgets.selection_stage_filter_widget.py
Injections:
ConfigBehaviorController
SignalController
Utility:
Displays a form that enable to sort
stage result tables

Add the missing information



Create new tables with missing information inside → Update CCDA → Update PyCCDA

Solve both problems related to Power Converter

Find a stable way to find non-FGC Power Converters

- Some old Power Converters covered by WIC systems have no FGC layer
- This means that if they are set in external fault by simulating a fault on a magnet, they cannot be restored to normal conditions remotely
- They must absolutely be excluded from testing
- Some of these non-FGC Power Converters are not even referenced in the CCDB

Fix in the Powers Converters test the problem with the recovery of timings

- To calculate the time elapsed between the moment when a Power Converter goes into fault and the moment when the USER PERMIT passes in False on the BIS side, I retrieve the timestamp of the change of status on the BIS side and the time stamp of the change of status on the Power Converter side and performs the subtraction
- I noticed aberrant results in some cases (negative elapsed time) that needs to be investigated

Incorporate the WIC commissioning into AccTesting

- My project enabled us to clarify the organization and usage of WIC systems at our section level. It also allowed WIC experts to gain a better understanding of their true needs.
- During the implementation of my Python program, I ensured to properly isolate the testing logic of my program from the GUI part so that it can be easily extracted.
- We could now extract the logic of my Python code to automate the WIC commissioning to incorporate it in a more general project developed by our team
- The next step of this project is to implement the testing logic on a server in order to integrate the WIC commissioning into AccTesting.