MINT, AN ITER TOOL FOR INTERACTIVE VISUALIZATION OF DATA

L. Abadie, G. Carannante, I. Nunes, J. Panchumarti, S. D. Pinches, S. Simrock, M. Tsalas, ITER Organization, Saint-Paul Lez Durance, France
D. Makowski, P. Mazur, P. Perek Lodz University of Technology Department of Microelectronics and Computer Science Wolczanska, Lodz, Poland
A. Neto, Fusion For Energy, Barcelona, Spain
S. S. Kalsi, Tata Consultancy Services, Pune, India

Challenges/Requirement

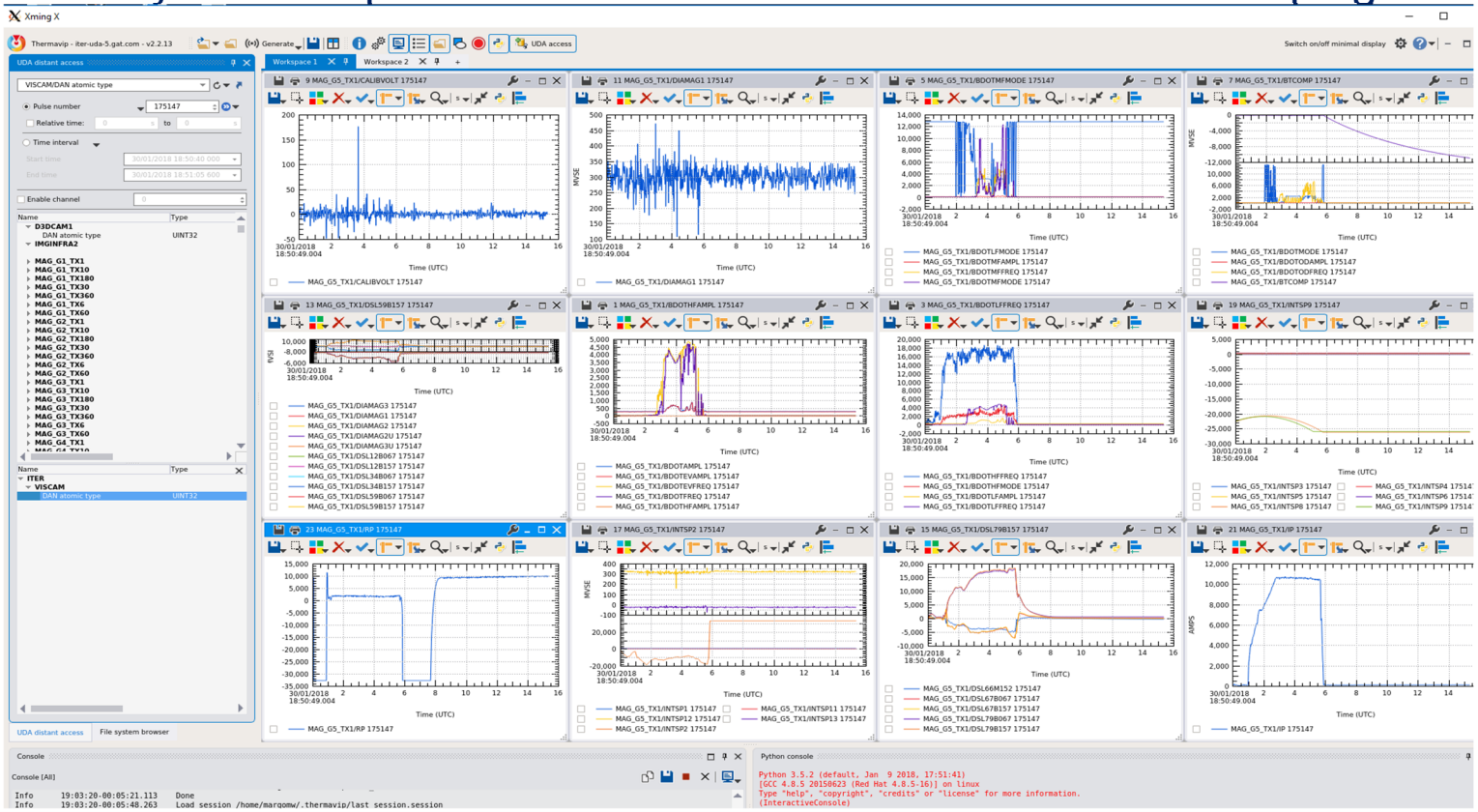
1. Different stakeholders, different needs
    1. Plant Engineers : system investigation, research, quick search
    2. Science : pulse analysis, research, quick search
    3. Operation team : pulse analysis, quick search

2. Heterogeneous data to be visualized
    1. Time and profiles traces
    2. Spectrum
    3. Fluxes
    4. Images, videos
3. Architecture shall be flexible to cope with technology changes

-> 2 families : web-based tools and **desktop tools**

Desktop vis tools

- Investigate what tools are being used in other tokamaks
  - Most of them are IDL-based because permissive and great graphical support
  - Not an option for ITER as the IDL license is expensive
  - Study thermavip from CEA and we extended it with an UDA plugin



Desktop vis tools

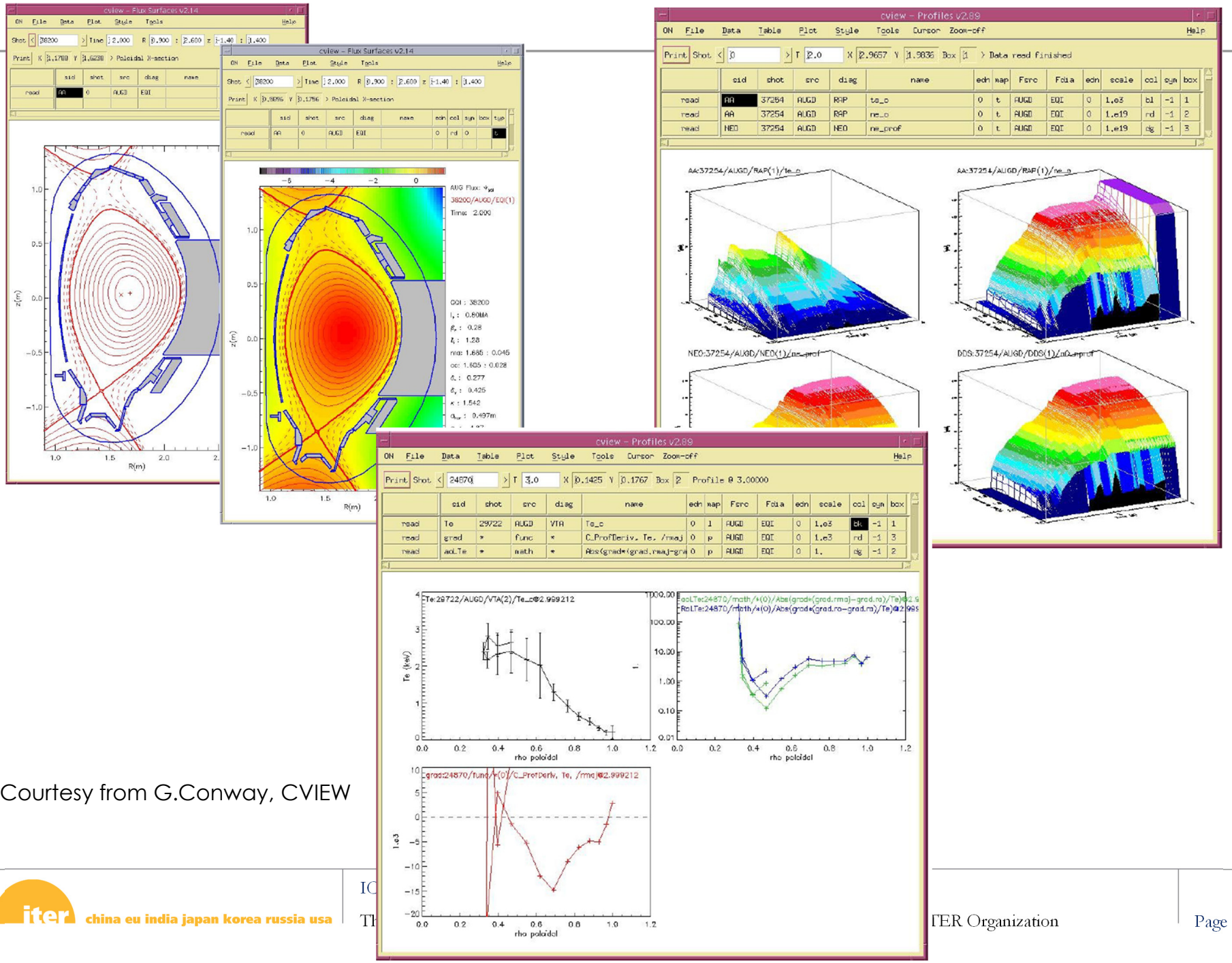
- Agile Development
- Focus on performance and resources check (CPU,mem)
- We made various demo of Thermavip to stakeholders
  - Some were enthusiastic but the main stakeholders were not…
  - So we failed
  - The plus : good way to start to capture the requirements!
- Start to study why we failed
  - Architecture was not thought carefully, the UDA integration was not nicely implemented, adding a new type of data source would have been painful
  - The human factors were not carefully thought : too many clicks for the most useful functions…
- Series of meetings with key stakeholders to better understand the full scope and needs

Outcome

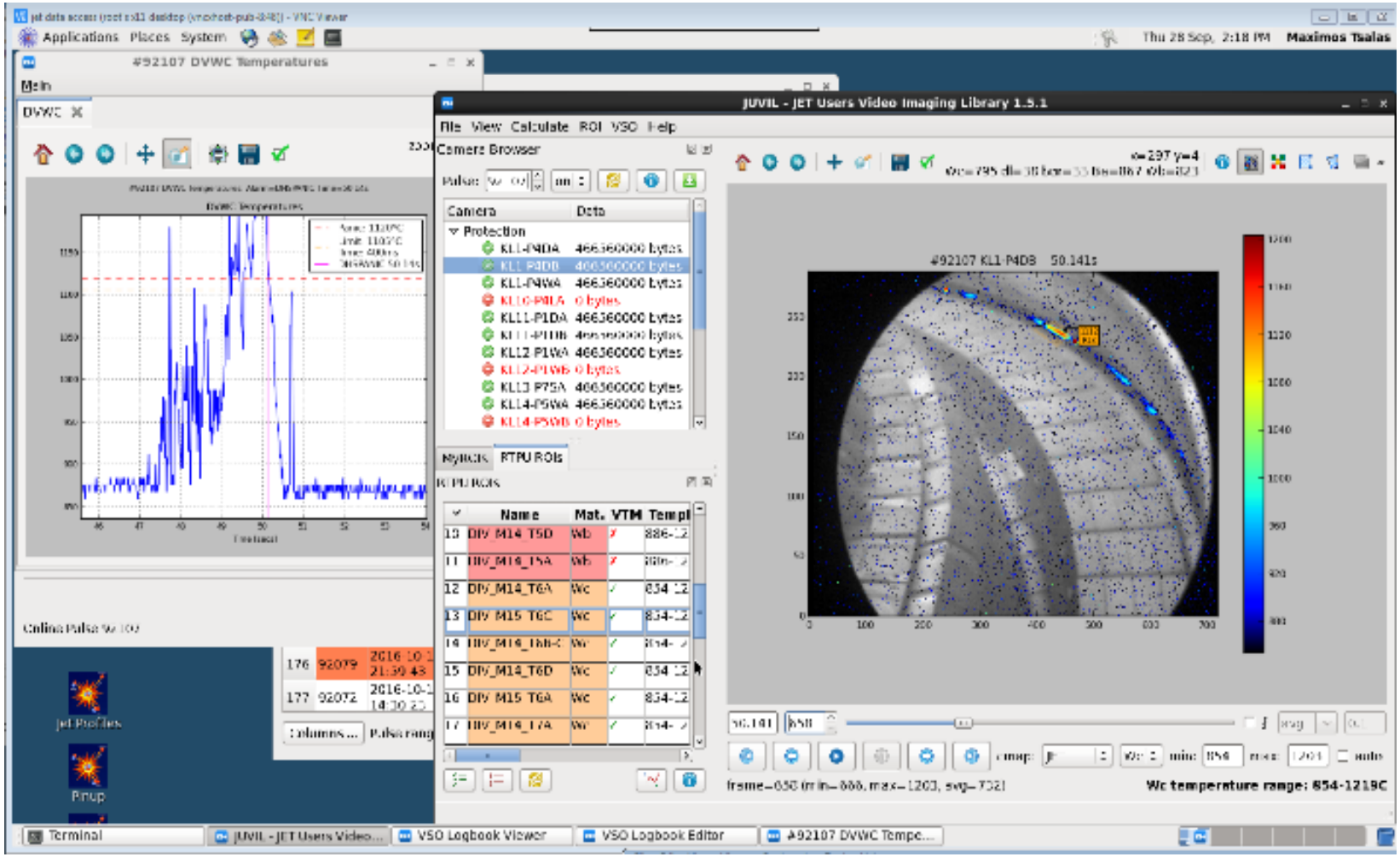
- Importance to have a clear architecture
  - Modular with an abstraction of the graphical library so that we can change it if needed (3-D is out of scope)
  - Develop a Python plotting library (but not up to OpenGL level) so that scientists can develop their own tools/applications
  - Use this library to develop the different tool in the control rooms
- Use of tabular view for plot configuration
  - Results of analyzing existing tools ReviewPlus, JETDSP, CView
- Detached/fully integrated panels
  - Industrial systems like Cooling, Cryo will prefer integrated panels
  - But not the others…Most tools used in tokamaks consist of detached panel to freely move them around, resized
- Community approach
  - Science, operation, heating, diagnostics

Different tools instead of one big tool

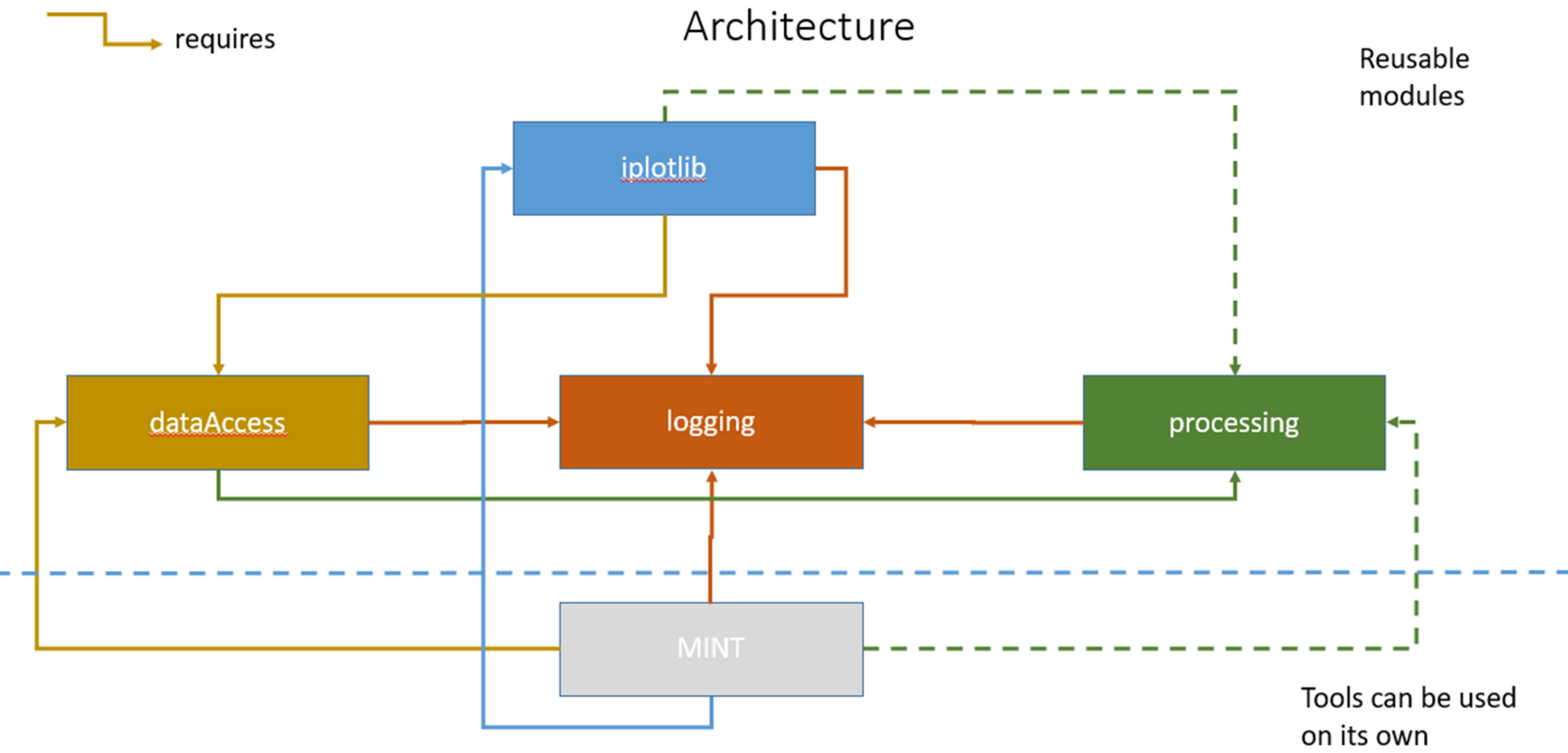
- Modular architecture
  - Can be started individually and assembled via plugin approach
- **MINT - Trends**
  - View time-series data, by time range or by pulse, include oscilloscope view
  - Profiles : Plots as function of space at selected time(s)
  - Can subscribe to the live stream if needed
- **Image viewer**
  - Mainly for camera data (infra, vis) – requires machine description data
  - More specialized function related to image processing
  - Can be mixed with 1-D data for correlation
- **Magnetic flux surfaces**
  - Requires geometry information (coil position for instance)
  - 2-D plot in (R,Z) at a selected time(s)


cview – Flux Surfaces v2.14
cview – Profiles v2.89
Courtesy from G.Conway, CVIEW

(JUVIL from JET)



Architecture (1/2)



Architecture (2/2)

- Easier to add a new data source for data access
- Easier to switch to a new plotting library:
  - clear abstraction via iplotlib
  - Select 2 libraries Matplotlib and VTK to validate the library
  - None of the tools make an import of the underlying plotting library
  - Can be called with users' own data
- Processing layer can be used in conjunction with data access or not

MINT – Features supported

Features
View one or several 1-D trends (X axis is the time)
Support for pulse-based and time range data access (all the plots show the same time range/pulse)
Support for time range or pulse specialization at row level (allow comparing two plots side by side)
Support for pulse overlay
Support for interactive zoom
Support for crosshair (over multiple plots)
Support for envelope display
Support for pan interactive
Support for preferences update {color of a signal,draw-style {interpolated,stair}, display points with markers}
Add a plot title
Support for automatic X-axis scale (if I zoom or pan on one plot, all the plots will be rescaled to the same time window).
Import/Export list of variables
Save/load configuration a canvas
Support for streaming
Support for full-mode
Support for basic processing of one signal
Support for Y and X-axis label edition
Generate a screenshot given a preference file in an headless environment (to allow report automation)

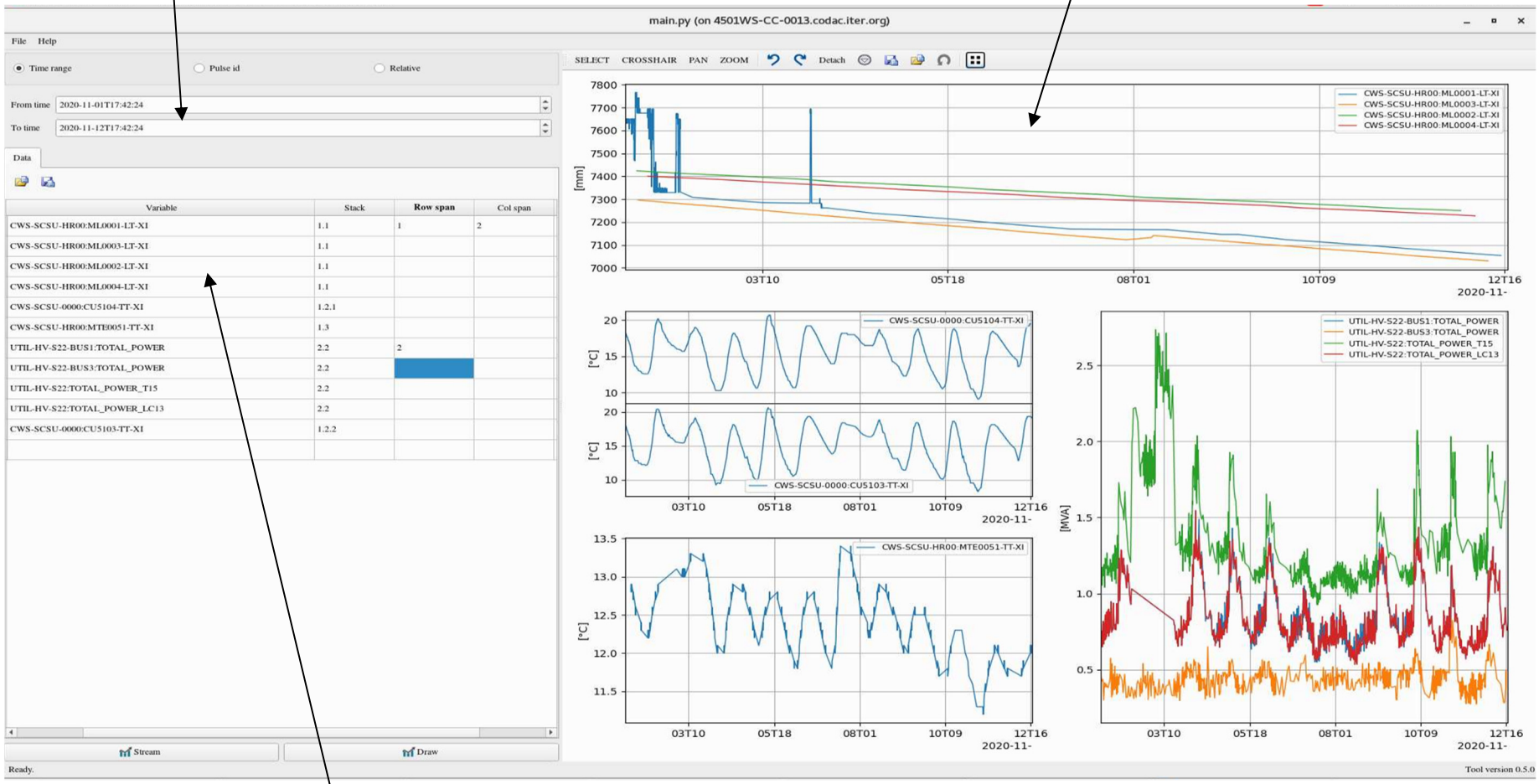
Technologies

- Main tests done on RHEL8
- Development with Python 3.8, PySide2
- Assumption is that the graphical library needs to support Qt backend (which is rather common)
- Qt offers a wide range of widgets and allow building complex UI, support various platforms and good support for screen resolution via env. Variables.
- Python is appreciated by scientists and offers a wide range of modules
- Use of GIT to store the code
- Implement unit tests whenever possible
- Zoom/Pan are interactive
  - Means if the data to be retrieved is too big, only a few points are retrieved (link to screen resolution)
  - Whenever a zoom/pan is triggered , a new request is triggered if (not all data have been retrieved)
- Memory footprint and CPU of MINT are monitored.

Layout of the tool

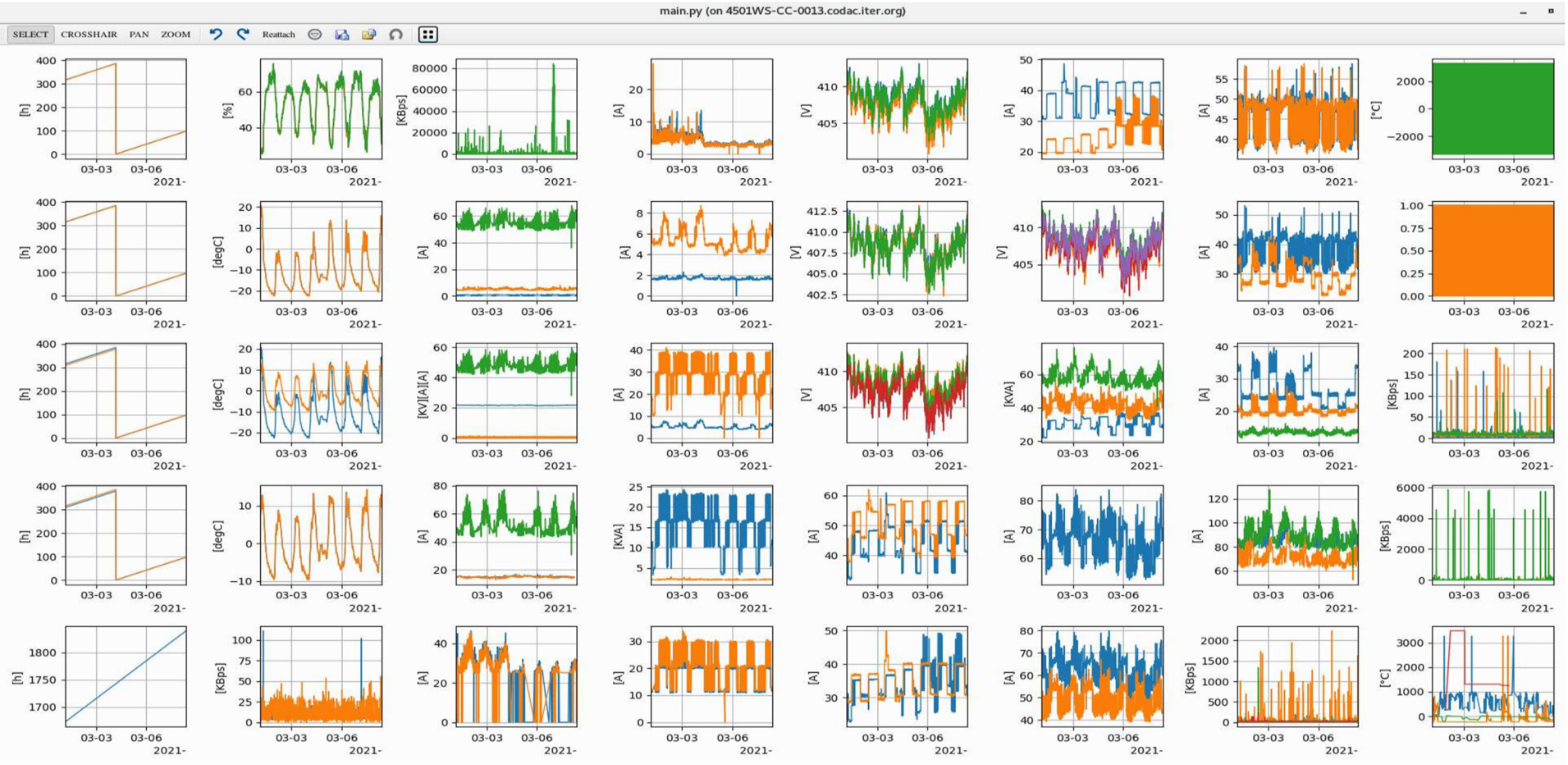
Time range {absolute, relative by pulse}

Plotting area



Allow to create your canvas

100 signals, 1 week of data ~2mn



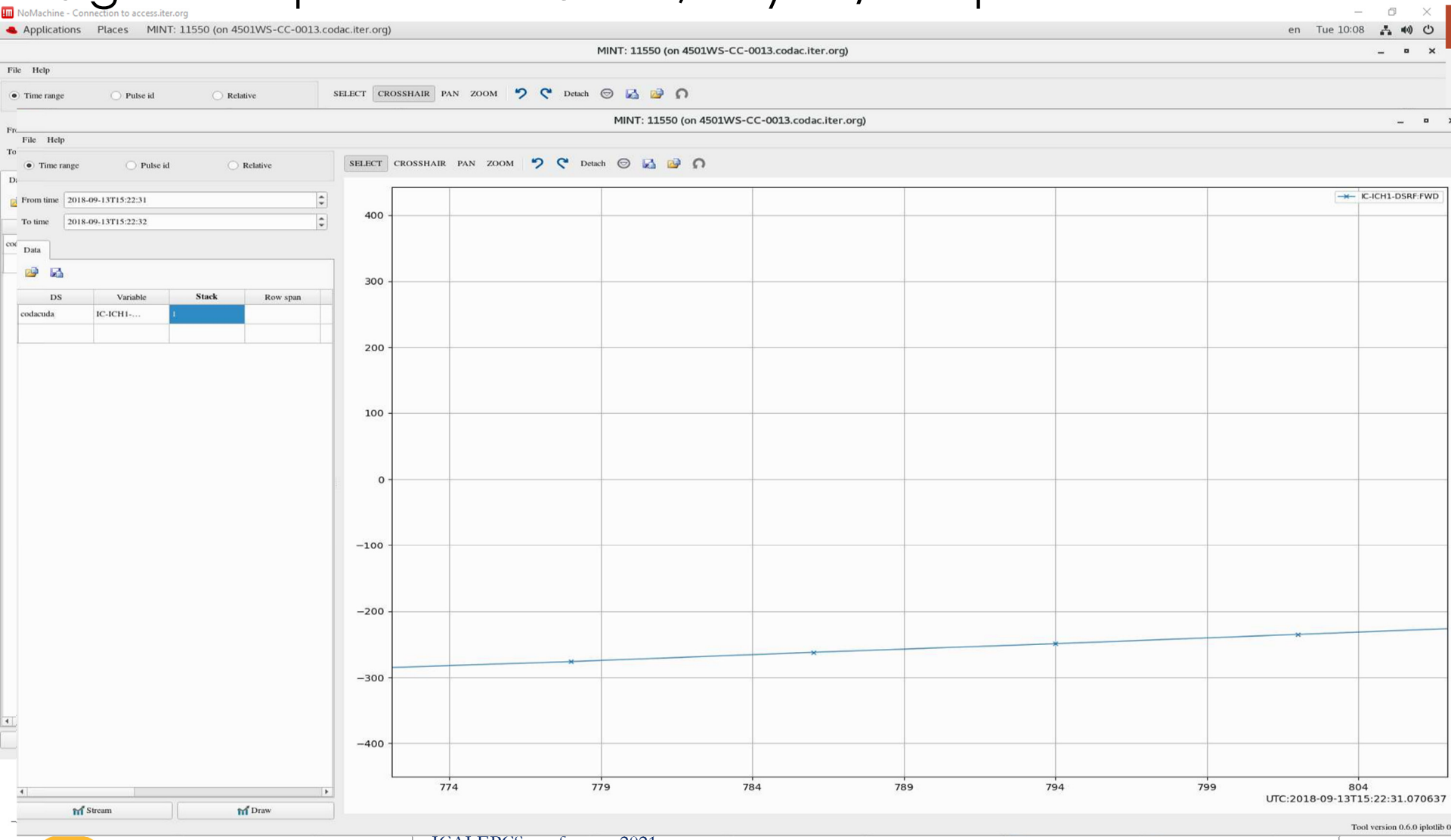
Plotting library abstraction

```
43
44
45  signal1 = DataAccessSignal(datasource="codacuda", varname="IC-ICH1-DSRF:FWD", pulsenb=123456, ts_relative=True)
46  plot1 = PlotXY(grid=True, axes=[LinearAxis(is_date=True), LinearAxis()])
47
48  plot1.add_signal(signal1)
49  plot1.add_signal(signal3)
50
51  plot2 = PlotXY()
52  plot2.add_signal(signal2)
53
54  canvas = Canvas(rows=2)
55  canvas.add_plot(plot1)
56  canvas.add_plot(plot2)
57
58  canvas.set_mouse_mode(Canvas.MOUSE_MODE_PAN)
59
60  QStandaloneCanvas(implementation="MATPLOTLIB", canvas=canvas).run()
61  # QStandaloneCanvas(implementation="GNUPLOT", canvas=canvas).run()
62
```

API -Backward compatibility will need to be carefully managed!

Handling nanoseconds precision

Signal acquired at 125MHz, 2bytes/sample



Trends tool : Next steps

- Improve the code quality (refactoring on-going)
  - Try to automate as much as possible canvas creation
- Extend the tool with more features
  - Support processing of multiple variables (Careful thought will need to put on realigning the time base and syntax definition)
  - A new window to allow browsing/searching the variables and its structure definition
  - Plot something else than time in X-axis (e.g. profile)
  - New plot types {surfaces, histogram, contours and XY with slider}
  - Support for overlay in case of XY (allow highlighting a specific region)

Conclusions

- Good progress on the data visualization tool mainly one trend (the most urgent one)
  - Implement the remaining features
  - Next one will be the image viewer and analyzer
  - Then flux surfaces viewer
  - Also work with Science to make sure that the plotting interface matches their expectation
- Design and implementation managed according to priorities and users' need
- **Improve the performance of the data access to give the best experience to the users**
- Support users to build their own expert tool based on iplotlib