Modular Analysis with DatABriCxx

Oliver Schulz

MAX-PLANCK-GESELLSCHAFT

Max-Planck-Institut für Physik
(Werner-Heisenberg-Institut)

oschulz@mpp.mpg.de

Sino-German GDT Symposium, Oct. 21th, 2015

Algorithm modularity

Challenges:

- Flexible selection and combination of algorithms
- Plug-ins: Keep code with different dependencies separate

Algorithm modularity

Challenges:

- Flexible selection and combination of algorithms
- Plug-ins: Keep code with different dependencies separate
- Deal with frequent personnel turnover:
  Students and PostDocs come and go,
  how to combine their software and keep it alive?

Algorithm modularity

Challenges:

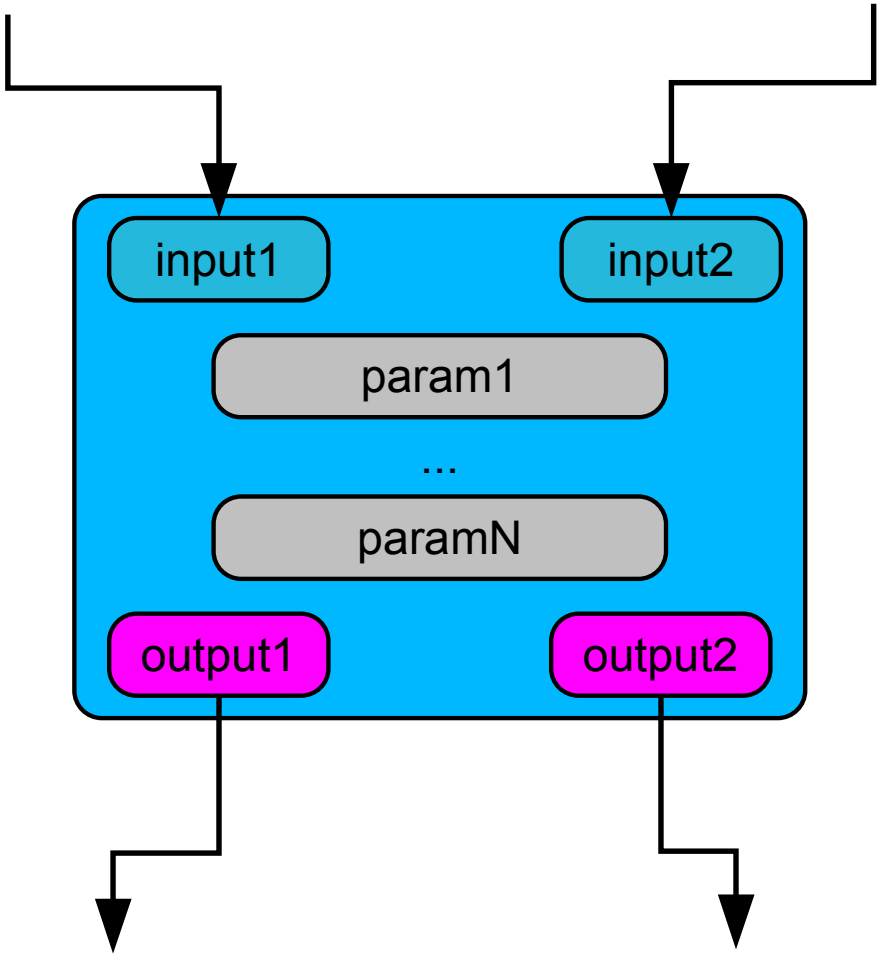
- Flexible selection and combination of algorithms
- Plug-ins: Keep code with different dependencies separate
- Deal with frequent personnel turnover:
  Students and PostDocs come and go,
  how to combine their software and keep it alive?
- Configuration/Parameter management
- Modularity *inside* loops over datasets, events, channels, ...
- Hard-coding is not the way: Lot's of boilerplate code, ...

Algorithm modularity

Challenges:

- Flexible selection and combination of algorithms
- Plug-ins: Keep code with different dependencies separate
- Deal with frequent personnel turnover:
  Students and PostDocs come and go,
  how to combine their software and keep it alive?
- Configuration/Parameter management
- Modularity *inside* loops over datasets, events, channels, ...
- Hard-coding is not the way: Lot's of boilerplate code, ...
- Consistent generalized framework?

What's is an algorithm?



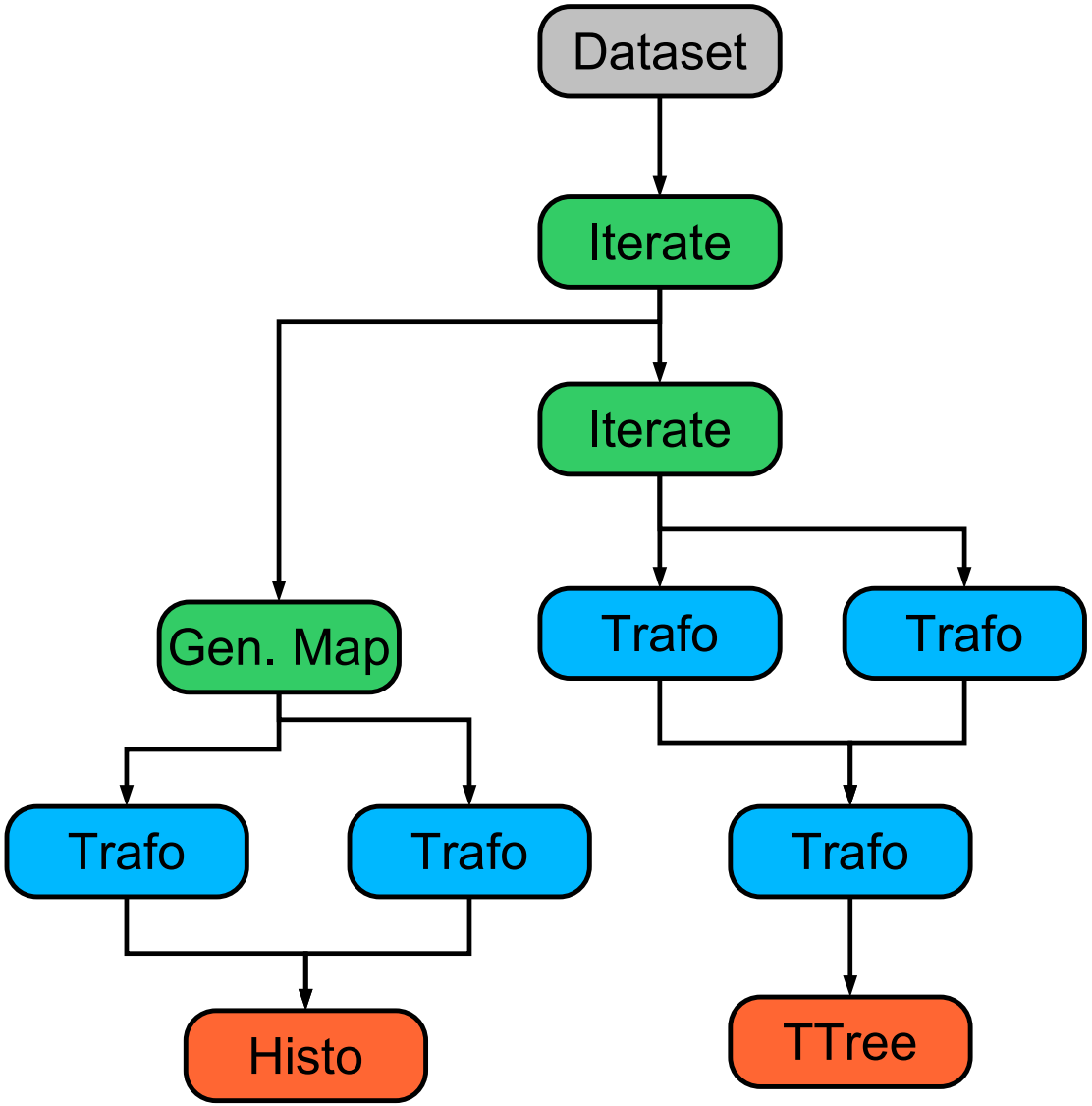
- Treat as black box with inputs, outputs and parameters
- Let's call this a Bric(k)

Input to Output Multiplicity

Simplified Map/Reduce-like approach covers all we need:

- 1 to n: Mapper
  (e.g. iteration over files / events / channels, cuts, ...)
- 1 to 1: Transform
  (e.g. waveform filters, fitters, ...)
- n to 1: Reducer
  (e.g. histogramming, file output, ...)

The Bric Graph



Brics form a directed, acyclic graph (DAG)

DatABriCxx

- Brics as analysis building blocks
- Mapper, transform and reducer brics
- Modern: Use full power of C++11 and ROOT-6
- Brics can form arbitrary directed, acyclic graph (DAG), automatic topological sorting and execution scheduling
- ROOT-based class loading:
  Keep your brics separate from framework
- Basic mappers: Loop over TTrees, text files, vectors, ...
- Basic reducers: Build ROOT histograms, write text files, ...
- Error handling
- ...

DatABriCxx

- ...
- Integrated configuration and parameter handling
- PropVal: Fast in-memory representation of untyped data: bridge between C++ types and JSON (and similar formats)
- Algebra on PropVal allows adding, diffing and merging: cascading configuration,
  exchange partial state with outside world (e.g. GUI), etc.
- Fast and comfortable logging system
- Universal string formatting (e.g. for exception messages)
- ... and more

Bric Configuration

- Bric graph is specified via JSON config file:

```
{
   "inputFileReader": {
     "type": "dbrx::RootFileReader",
     "input": "input.root"
   },
   "inputTreeReader": {
     "type": "dbrx::RootTreeReader",
     "input": "&inputFileReader.content.someTree",
     "nEntries": 1000,
     "firstEntry": 42
   },
   "histBuilder" : {
      "type": "dbrx::RootHistBuilder<double>",
      "input" : "&inputTreeReader.entry.energy",
      "histName": "energySpectrum",
      "histTitle": "Energy Spectrum",
      "nBins" : 3000,
      "xlow" : 0,
      "xup" : 3000
   }
}
```

Mapper Brics

- Produces output n times for each new input
- Methods to implement: processInput(), nextOutput()

```
class SimpleMapper: public dbrx::MapperBric {
protected:
    std::vector<double>::const_iterator m_iter;

public:
    Input< std::vector<double> > input{this};
    Output<double> output{this};

    void processInput() override
        { m_iter = input->begin(); }

    bool nextOutput() override {
        if (m_iter != input->end()) {
            output = *m_iter;
            ++m_iter;
            return true;
        } else return false;
    }

    using MapperBric::MapperBric;
};
```

Transform Brics

- Produces output exactly once for each new input
- Most frequent bric type
- Methods to implement: processInput()

```
class SimpleTransform: public dbrx::TransformBric {
public:
    Input<float> a{this, "a"};
    Input<float> b{this, "b"};

    Param<double> factor{this, "factor", "Some factor", 2.0};

    Output<double> c{this, "c"};

    void processInput() {
        c = factor * (a+b);
    }

    using TransformBric::TransformBric;
};
```

Reducer Brics

- Produces exactly one for all input
- Methods to implement: newReduction(), processInput()

```
class SimpleReducer: public dbrx::ReducerBric {
public:
    Input<double> input{this};
    Output<TH1D> output{this};

    Param<Int_t> nBins{this, "nBins", "Number of bins", 20};
    Param<Double_t> xlow{this, "xlow", "Low edge of first bin", 0};
    Param<Double_t> xup{this, "xup", "Upper edge of last bin", 100};

    void newReduction() override {
        output.value() = std::unique_ptr<TH1D>(
            new TH1D("hist", "Histogram", nBins, xlow, xup)
        );
    }

    void processInput() override {
        output->Fill(input);
    }

    using ReducerBric::ReducerBric;
};
```

Configuration and Execution

- Running the included example:

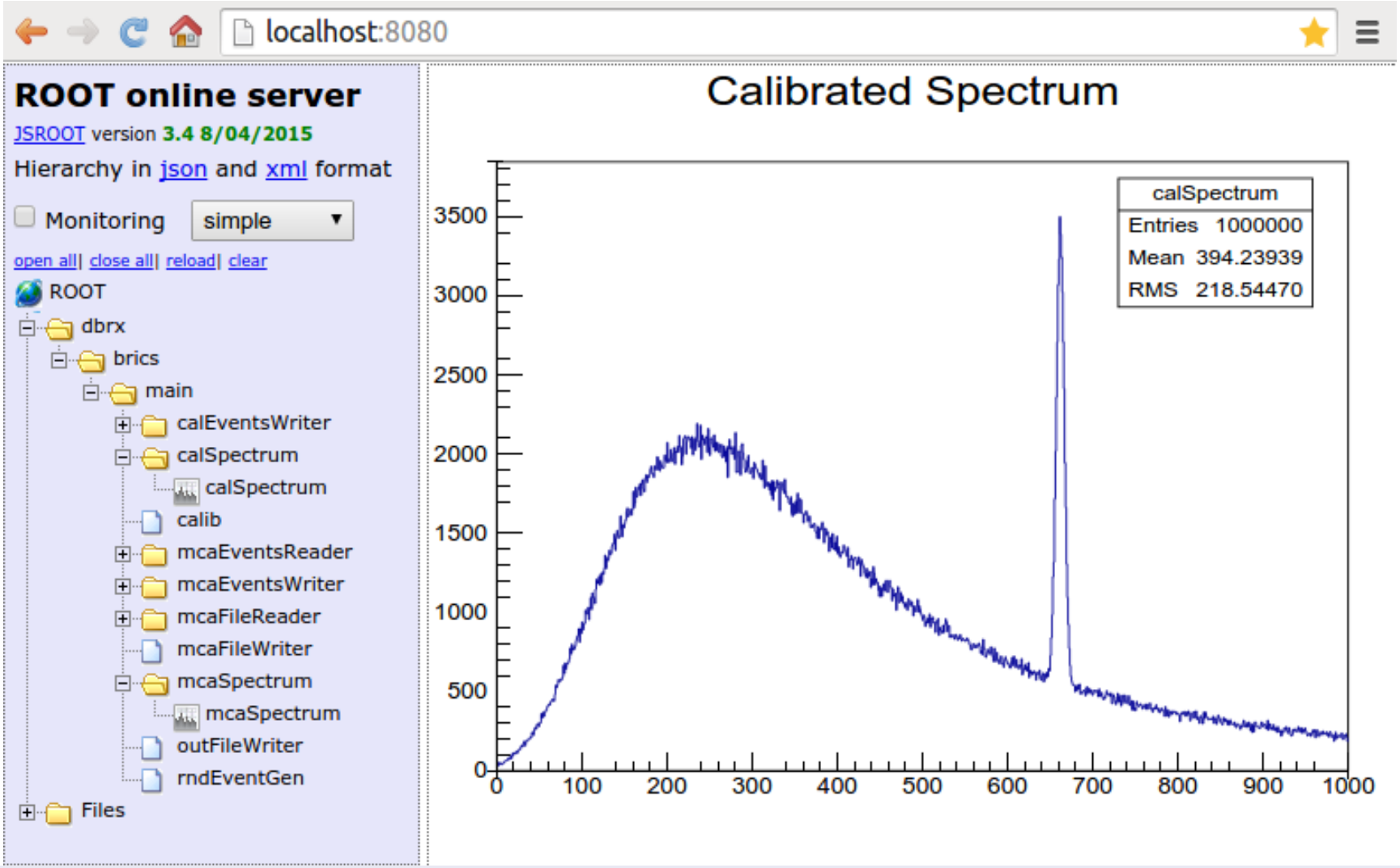
```
# dbrx run mca-calib.json
```

- Cascading configuration and variable substitution:

```
offset=-19.157
dbrx run -Vslope=0.10403 mca-calib.json mca-calib-vars.json
```

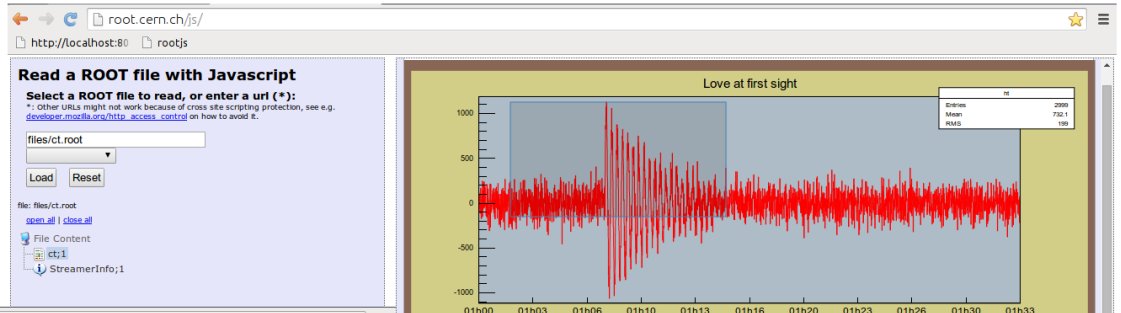
- Multiple configuration files are merged:
  - Allows for "default" configuration and specialization in separate files
  - Allows for modular configuration
  - Variable substitution

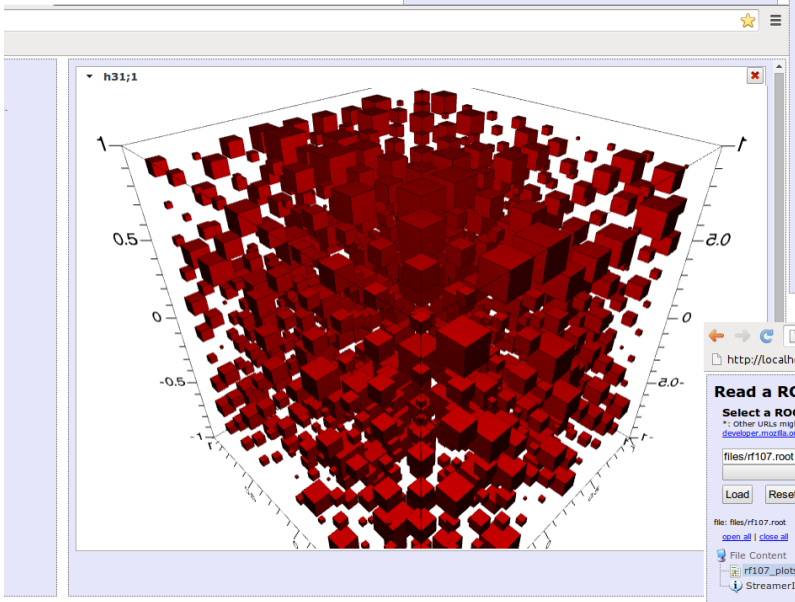
Simple GUI included (via THttpServer)

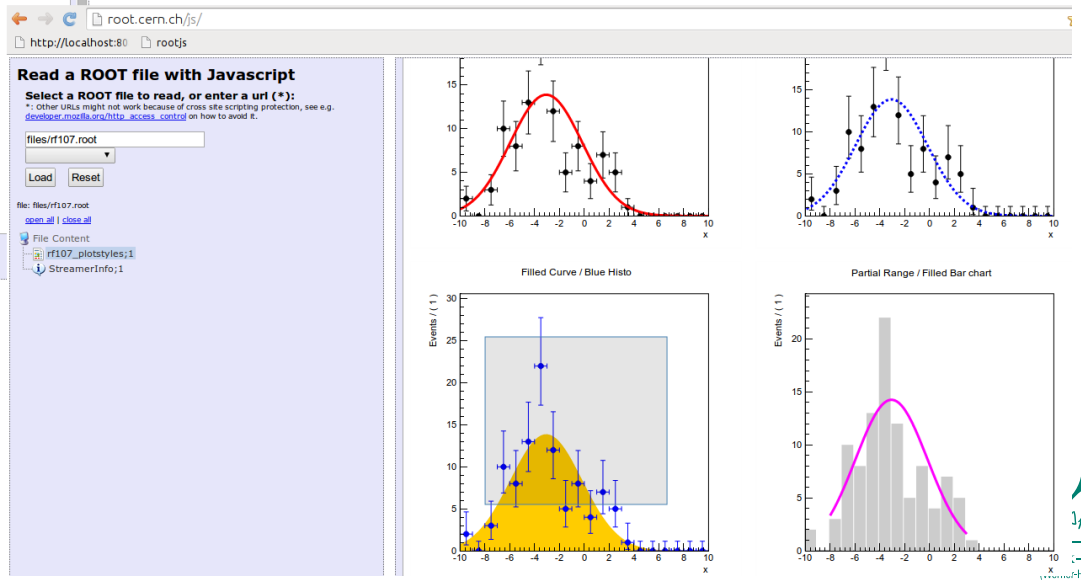



```
# dbrx run -k -w -p 8080 mca-calib.json
```

JSRootIO provides good basis ...







... for future GUI plans

- Web-GUI ideal for narrow-bandwidth remote work (compute clusters, etc.)
- No software installation on client
- Add custom visualizations based on d3.js
- May use different HTTP server in future for event sourcing, etc.
- Build future snappy SPA Web-GUI, e.g. based on AngularJS (using JSRootIO, etc.)

Summary and Outlook

- New framework DatABriCxx (Data Analysis Bricks in C++)
- Initial users: GeDet, CRESST, GERDA (partially), COBRA also interested

Summary and Outlook

- New framework DatABriCxx (Data Analysis Bricks in C++)
- Initial users: GeDet, CRESST, GERDA (partially), COBRA also interested
- Batch-operation ready (modulo bugs)
- No API docs yet, but first official examples
- Many improvements in the pipeline
- Primitive GUI now, plans for fancier GUI

Summary and Outlook

- New framework DatABriCxx (Data Analysis Bricks in C++)
- Initial users: GeDet, CRESST, GERDA (partially), COBRA also interested
- Batch-operation ready (modulo bugs)
- No API docs yet, but first official examples
- Many improvements in the pipeline
- Primitive GUI now, plans for fancier GUI
- Long-term support intended
- Public, open-source (LGPL)
- Get it here: https://github.com/mppmu/databricxx