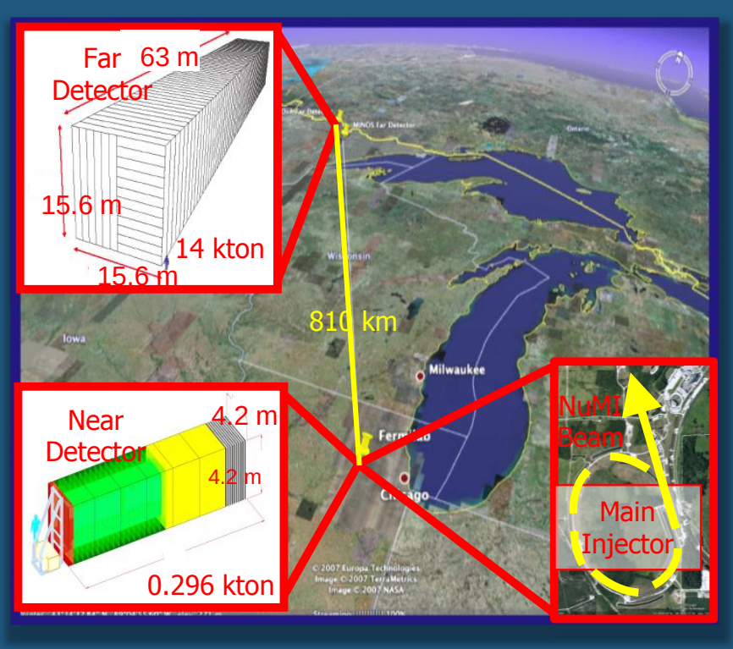
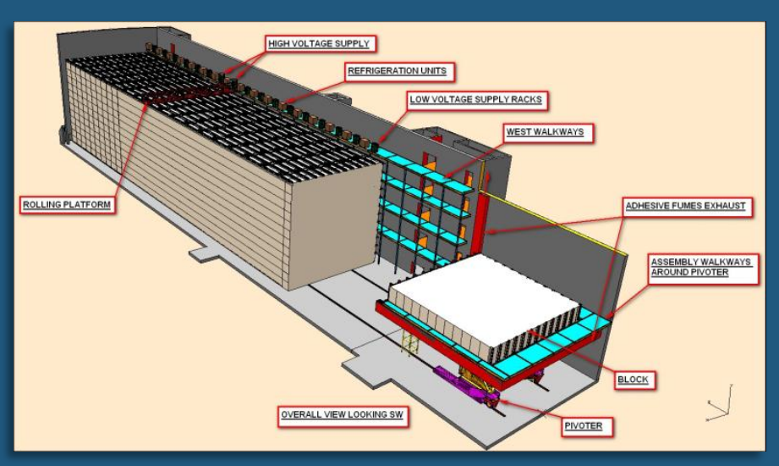
Message Service Analyzer and Error Handling in NOvA detector

Nitin Yadav[1], Qiming Lu[2]
[1]Indian Institute of Technology Guwahati, [2]Fermilab

NOνA Detectors

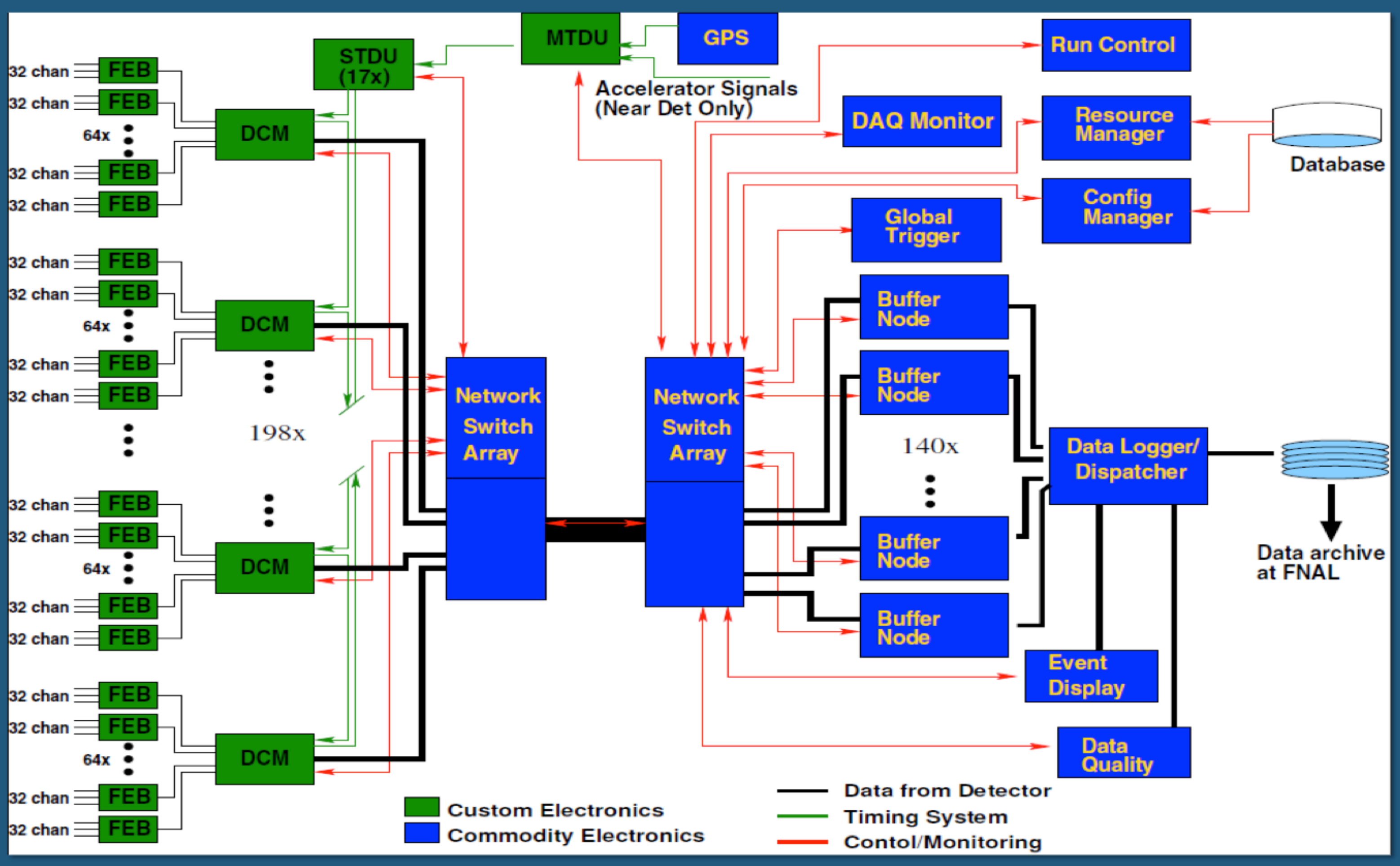




- NOνA is an *off-axis long-baseline* neutrino experiment using the NuMI neutrino beam and two detectors to look for ν_e appearance
- The Near Detector is located at Fermilab, 14 mrad off the NuMI beam axis, 100 m underground, 1 km downstream of the beam production target .
- The Far Detector is located at Ash River, in northern Minnesota, 14 mrad off the NuMI beam axis, 810 km downstream of the beam production target

The far(near) detector is composed of 28(6) blocks; each block is made of 32(24) scintillator PVC planes, giving 896(214) planes in total. The near detector also has a muon catcher which is composed of 12 scintillator PVC planes and 10 steel planes. NOνA is the world's biggest freestanding plastic structure.

NOνA DAQ



A schematic overview of the NOνA DAQ system. The data stream is from left to right. There are almost 200 DCMs and 200 buffer nodes. The data acquisition (DAQ) system of NOνA is complex, consisting of more than 400 distributed but closely-interacting components. For the Far Detector, 12,036 front-end boards are used to continuously read out 385,152 detector channels. The raw data are first concatenated, based on the geometrical location of the detectors, in Data Concentrator Modules (DCMs). They are then pushed downstream to a buffer farm. There are nodes dedicated to management and coordination of the system, e.g., run control, application manager, message server, and DAQ online monitor. Therefore, to maintain a healthy running state of the DAQ system, it is necessary to have a means of continuously monitoring each participating component and for detecting and reacting, with minimum delay, to abnormalities that might put the system or the quality of data in jeopardy. The system which takes care of it is called as *Automatic Error Recovery System (AERS).*

Automatic Error Recovery System – AERS

❖ The AERS is composed of three functional parts:

❑ Status Report: Status report from each component and from dedicated monitoring nodes.
*Status reports are issued, routed, collected and stored though a distributed message service package **MESSAGE FACILITY** developed by ssi@Fermilab.*

❑ Message Analyzer : Decision making engine performs the correlation analysis based on the status reports.
*Message analyzer is a rule engine that processes **Event-Condition-Action** rules **(ECA):*** if *"DCM heart attack"* and *" during data taking"* then send an *"email to the shifter".*

❑ SUPERVISOR: Error handling supervisor reacts to situations and carries out the action.
The action part of an ECA rule submits the detected error to the supervisor. Supervisor takes proper action based on error submitted and supervisor's knowledge about this error.

Message Analyzer

- ❖ **It is a real time event correlation analysis tool based on log messages.**
- ❖ **It implements a forward chaining inference engine.**
- ❖ **It deals with two level of cascading *ECA* rules:**

1. Fact Extraction: Driven by incoming messages to extract facts from status logs. Performs a series of tests on incoming message based on following attributes:
(A) Issuer, severity and category.
(B) Regex pattern matching.
(C) Occurrence/frequency.
Output of fact extraction is a *BOOLEAN FLAG.*
Ex: During data taking NOνA run control sends out a heartbeat check every second to components and expects a response otherwise emits a status message. A Fact that "DCM has a heart attack" is asserted as:

- ✓ Test whether the sender is "RunControl".
- ✓ Test whether the severity is "Warning" or higher.
- ✓ Test whether the category is "RegularCheck".
- ✓ Test whether the body matches regex expression "DCM missed heartbeats".
- ✓ Test whether the message has been seen over 10 times in the past 60 seconds.

If passes all five test, a flag of TRUE is marked under the fact "DCM has a heart attack".

2. Event Identification: Correlation analysis (rule) based on 'facts' to identify events. Use logically correlated facts to identify events:
"event of dcm_failure is asserted when both dcm_heartattack and dcm_selfcheck_failure flagged during data_taking or hardware_config stage".

Cascading rules: When a rule is triggered for querying its condition on an incoming event, executing its action may in turn trigger new rules for further assessment.

Extended Approach

It will be complicated to write facts and rules for each connecting component, which differ slightly. To cope with this issue, we have a 'grouping' approach called the Extended Approach. This extends a primitive fact into a collection of related facts by distinguishing ***sources*** and ***targets*** of a message. Collapse similar facts using wildcards and add a restriction clause.

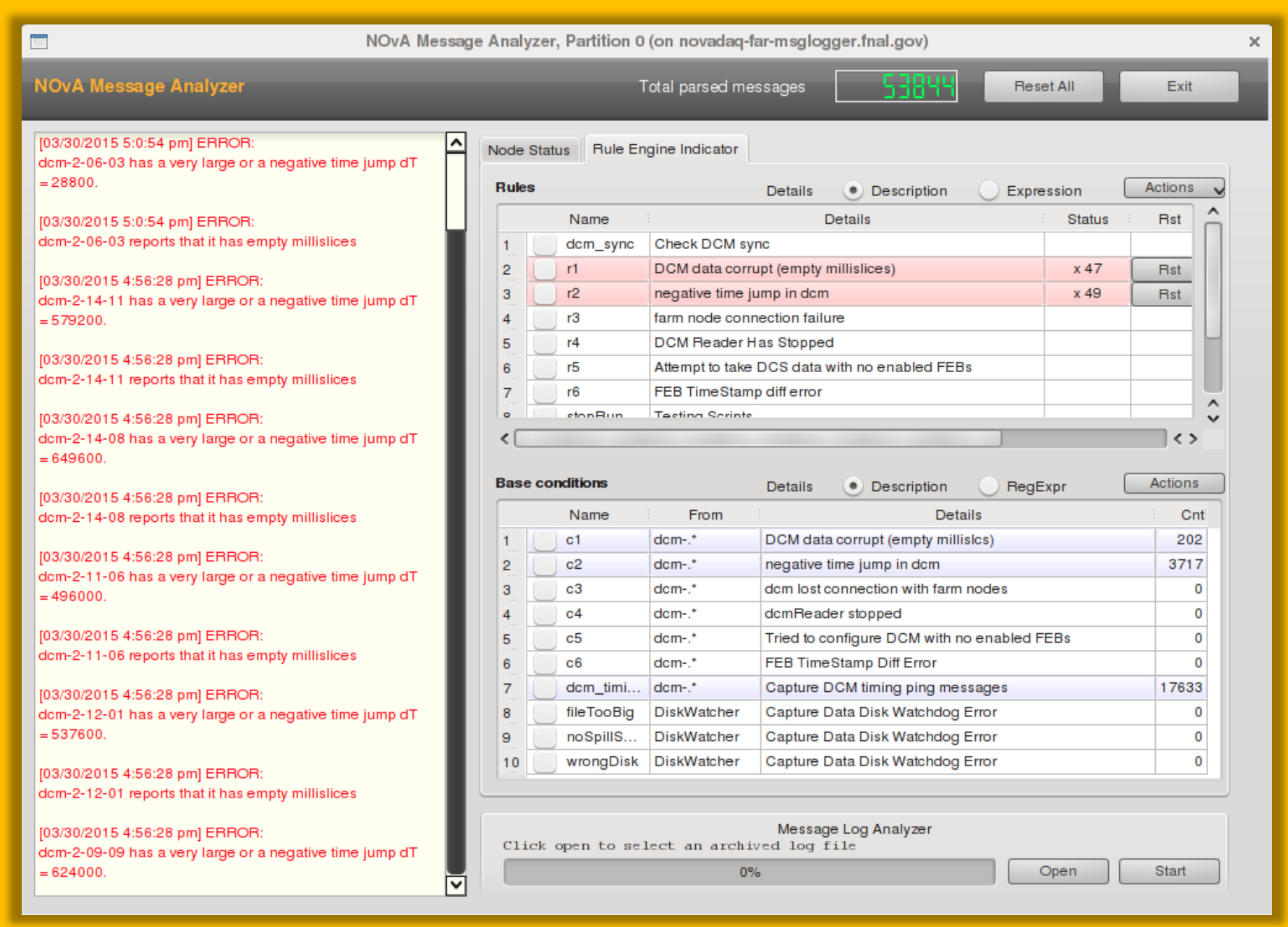
- Fact1: F1: RC:dcm-?? Missed heartbeats
- Fact2: F2: dcm-??:I'm not happy
- Rule with restriction clause:
  R1:=F1&&F2 *where the target of F1 is same as the source of F2.*

Facts & Rules in Configuration Language

The facts and rules form the knowledge base of the Message Analyzer, provided in the form of a configuration file. One simple rule is shown below:

```
Rule_simple :
{
  type        : simple
  description : "this is an example of a basic rule"

  # filtering conditions
  severity    : warning
  sources     : ["dcm.*", "rcWindow"]
  categories  : ["*"]

  # match by regex, or match by 'if contains', not both
  regex       : "corruption\s.*\sdcm-03-\d{2}-\d{2}"

  # frequency count can be for each source
  rate        : { occurrence      : 10
                  timespan  : 60   # in seconds
                 }
  # granularity
  granularity : { pre_source : true }

  # actions can be empty
  actions     : { alert : { level:warning message:"%s corrupted" }
                  popup : { message : "…" }
                  RunControl : { … } }
}
```

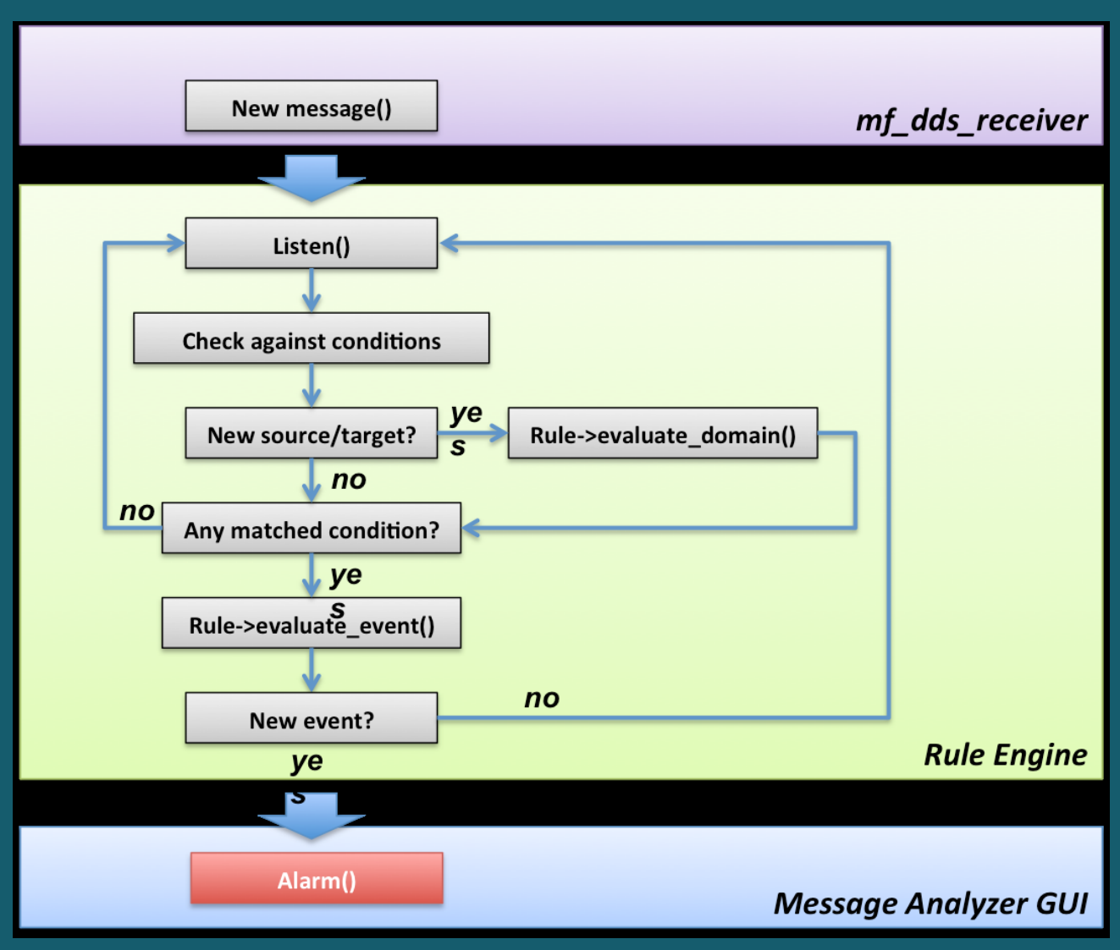
Message Analyzer GUI at NOνA



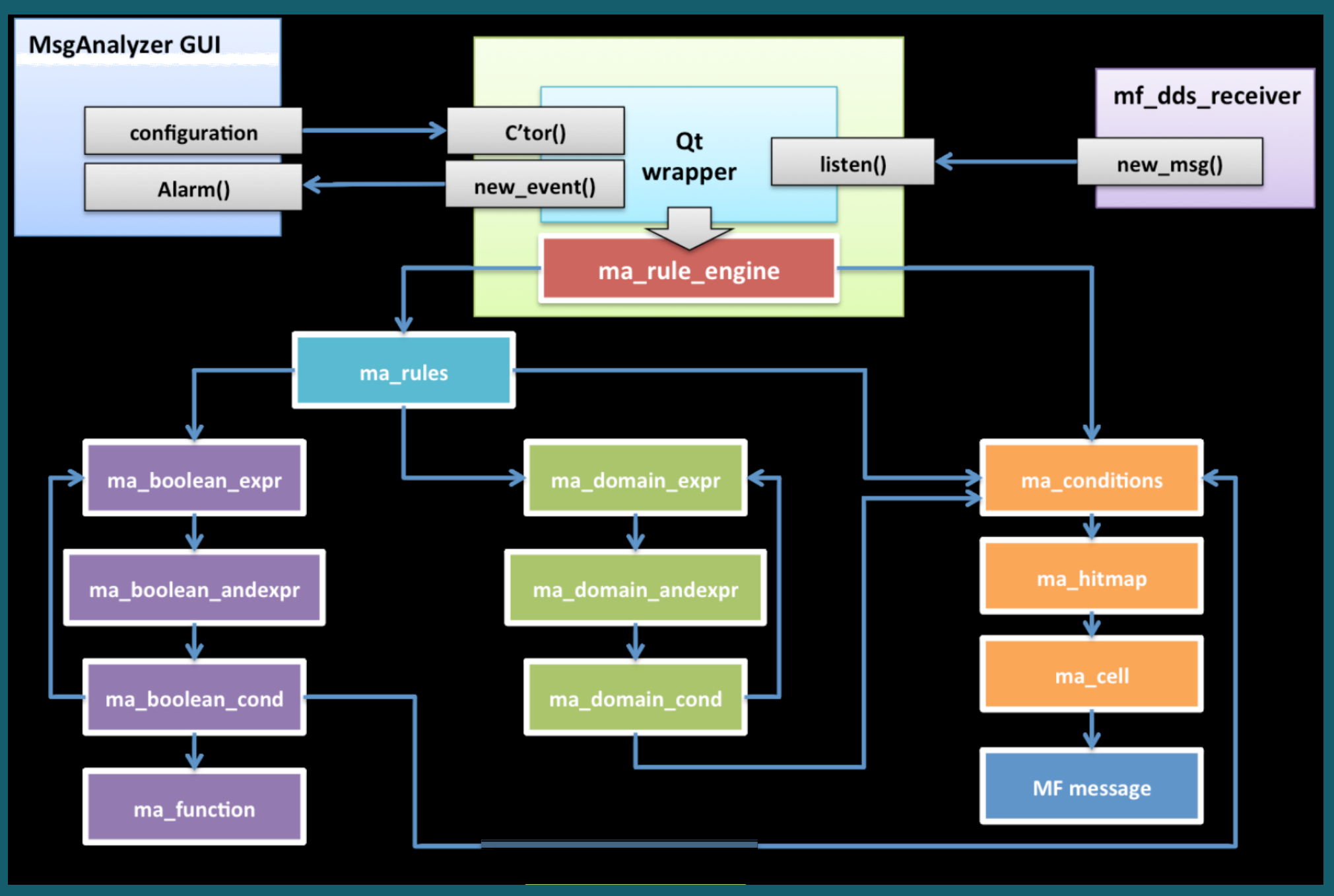
User defined function

Sometimes, more complex situations can not be handled just by the above rules. We have in this case user-defined functions which enable the Message Analyzer to handle a great deal of more complex situations using customized logic implemented in a generic programming language such as C++.

Message Analyzer: *Execution flow chart*



Message Analyzer: *class diagrams*



Summary and Outlook

- ❖ Message Analyzer is a light-weight correlation analysis tool with great flexibility and extensibility.
- ❖ It is being used for monitoring run state and data quality in the NOνA DAQ system.
- ❖ Separation of the system knowledge (the rules) from the software implementation. Composing facts/rules using a Domain-specific Rule Language.
- ❖ The package has been made generic and portable. Easy migration to other experiments.

CHEP2015 OKINAWA, Japan
21st International Conference on Computing in High Energy and Nuclear Physics CHEP2015 Okinawa Japan: April 13 - 17, 2015
Fermilab
U.S. DEPARTMENT OF ENERGY
Fermi Research Alliance
Indian Institute of Technology Guwahati
Paper read at