LIGO

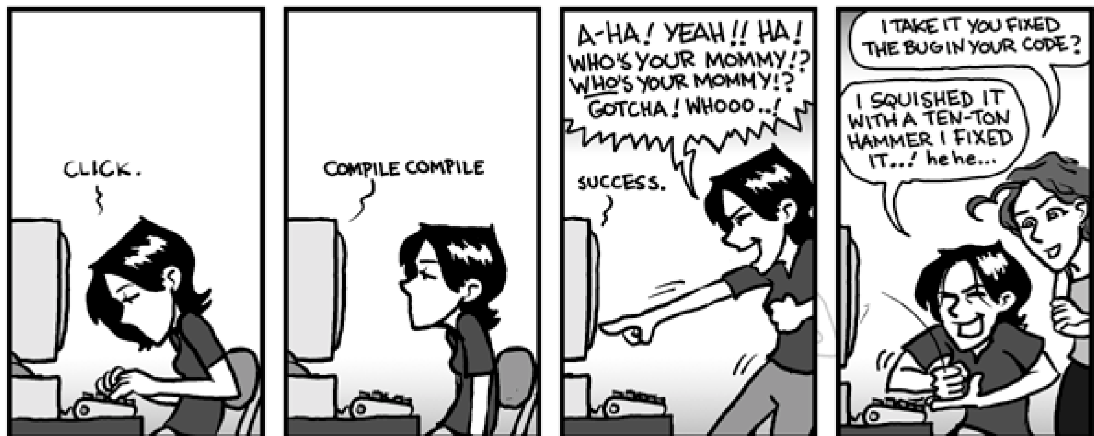
Easy DMT: Using the chInterface Class to Create/Upgrade DMT Monitors

Rauha Rahkola, Univ. of Oregon

LSC Meeting, August 19-22, 2002
LIGO Hanford Observatory
Richland, WA

Introduction to the chInterface utility class

- The design of a typical DMT monitor
- The chInterface model
- Simple examples: RMSWatch and Multi-RMSWatch
- Real-life example: absGlitch
- Other benefits of chInterface Features
- Looking beyond S1



LIGO

Design of the typical DMT Monitor

1. Read a configuration file
    1. configure monitor parameters
    2. add channels (w/ preferences for filters, thresholds, etc.)
    3. create a list of channels w/ their filters, thresholds, etc.
2. Run through the data
    1. for each channel, either:
        - gather some statistics, or
        - check if the data exceeds a threshold
    2. log the results (log file, trigger, etc.)
    3. repeat
3. Final report

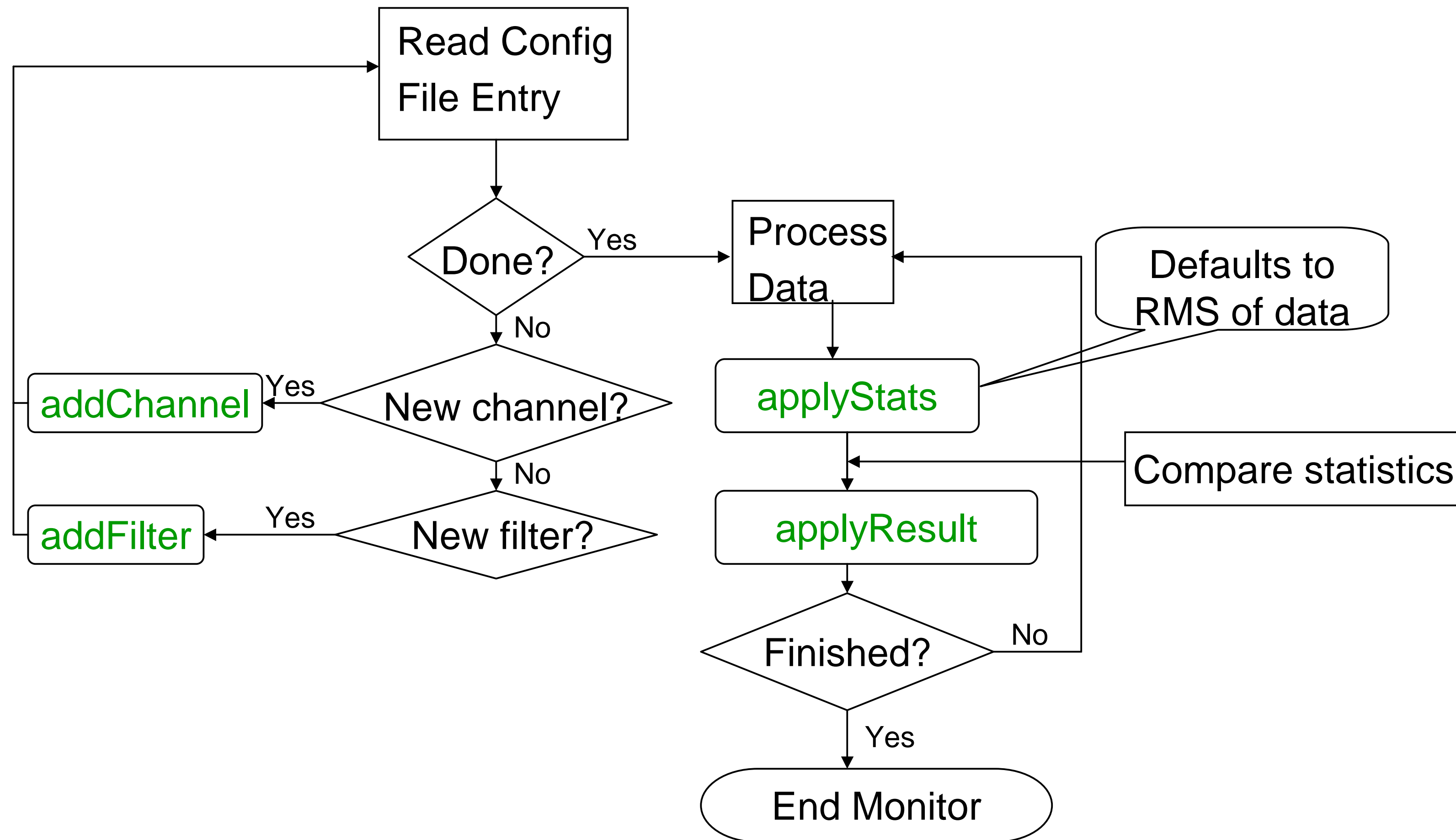
LIGO

The chInterface Model

- Add:
  - channels with addChannel
  - (multiple!) filters to channel with addFilter
  - statistic type with addStatistic
  - glitch-checking functionality with addGlitch
  - a method to report results with addResult
- Apply one or all:
  - filters to the data (or filtered data!)
  - glitch-checking to the (filtered) data
  - statistics-calculating to the (filtered) data
  - result-logging to the results of the above

A More Complex Example: Multi-Watch

A Simple Example: RMS Watch



Real-Life Example: absGlitch

```
void absGlitch::ProcessData(void) {
  fCurrentTime = getDacc().getCurrentTime();
  if ( Debug() > 1 ) cout <<"Processing Data: "<<fCurrentTime.getS()<<endl;
  //======================== If data is continuous, update stats & check event
  if ( fChannels.begin()->isTSContinuous() ) { // check continuity
    for (CHiter ch = fChannels.begin(); ch != fChannels.end(); ch++) {
      //----------- If the OSC, if used, is satisfied...
      if ( ch->getOSC().empty() || mOSC.satisfied(ch->getOSC().c_str()) ) {
        //--------- convert TSeries to double (note: should take 0 time, but
        //---------  we'll be safe (if slow) by doing it here.  Note that
        //---------  the reason we need this is IIRFilter only uses doubles
          (*ch->getData())->Convert(DVector::t_double);
        //--------- filter data
          ch->applyFilters();
        for (unsigned int i=0; i < ch->FBank.size(); i++) {
          list<string>::iterator j;
        //--------- update the filtered data statistics
            for (j=statList.begin(); j != statList.end(); j++)
              ch->applyStats( *j, ch->FBank[i] );
          //--------- check for event
            for (j=glitchList.begin(); j != glitchList.end(); j++)
              ch->applyGlitch( *j, ch->FBank[i] );
        }
      } // if ( ch->getOSC().empty() || mOSC.satisfied...
    } // for (CHiter ch = fChannels.begin(); ...
  //=======================  Otherwise there's a continuity error
  } // if (fChannels.begin()->IsTSContinuous())
  else fLogFile<<"TimeSeries Continuity Error "
               <<(*fChannels.front().getData())->getStartTime().getS()<<endl;
  //=======================  Output stats, if need be
  //  OutputLog(true);
  //=======================  Check if we've reached the end of the run
  fCount++; // increment # of ProcessData executed
  if (fEndRuntime) {
    if (fCount >= fEndRuntime) finish();
  }
}
```

Other Benefits of chInterface Features

- Modular forms of nnnConstruct classes
  - modify a monitor to use a different glitch-finding scheme
  - statistical tests become plug-ins
- Multiple functions through a single monitor
  - a step closer to on-the-fly monitor configuration
  - new tests for coincidences/clustering different kinds of “glitches” w/ Event Tool
- Heirarchy of filters
  - create a band-pass filter using high- and low-pass filters
  - check results in intermediate stages of filtering

Looking Beyond S1

- During S1:
  - implement the GlitchConstruct equivalent of the absGlitch glitch-finding scheme
  - include the chInterface class in the GDS CVS tree
- Between S1 and S2:
  - absGlitch and eqMon monitors will implement chInterface after S1
  - with the help of others, the repertoire of GlitchConstruct, StatisticsConstruct, and ResultConstruct classes will grow
- After S2:
  - much functionality of the existing (non-GUI) monitors will be replicated in the nnnConstruct classes

LIGO

In Summary

- chInterface shoulders the burden common to most (all?) DMT monitors
- chInterface GlitchConstruct, StatisticsConstruct, and ResultConstruct are base classes from which can be derived the functional equivalents to existing monitors
- DMT monitors using chInterface can be multi-functional

More info: check http://www.ligo-wa.caltech.edu/~rrahkola/chInterface/
(until ~S2 begins, when chInterface is implemented in GDS)