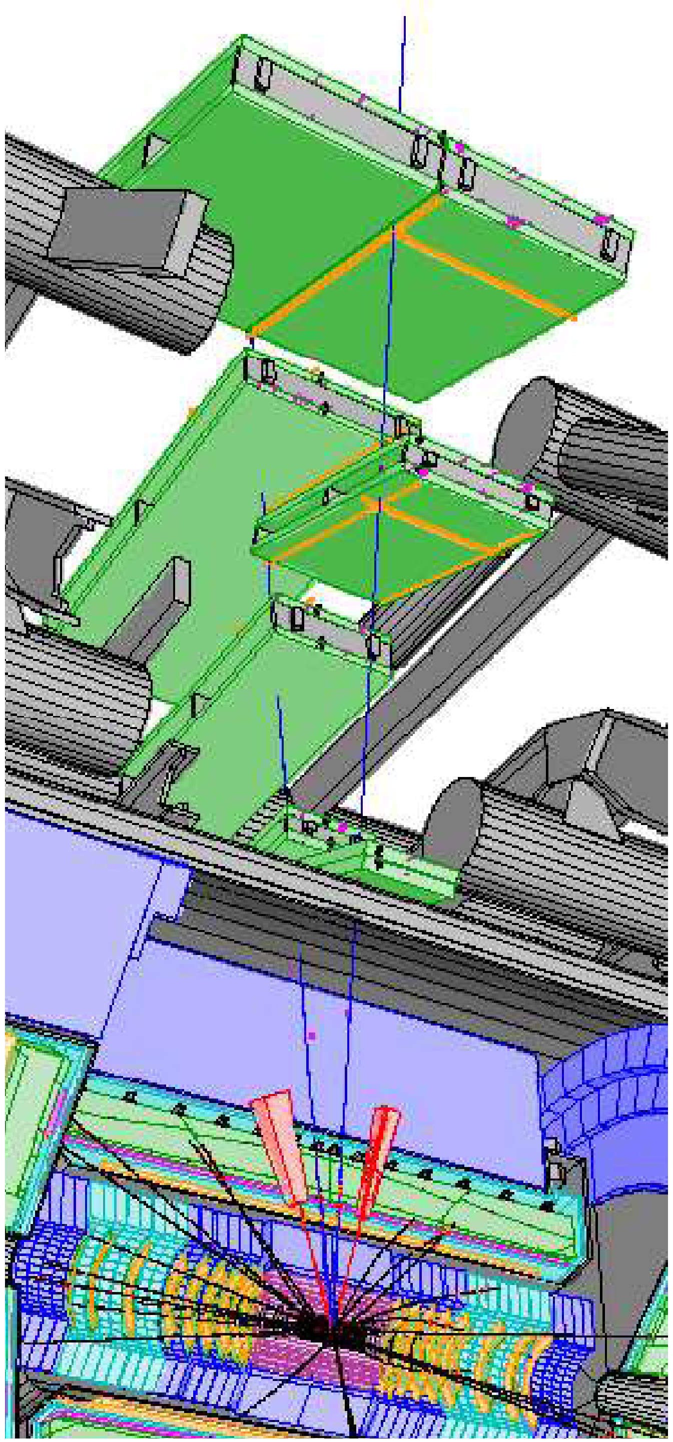


The New TrigDecision

Nicolas Berger, Till Eifert, Ricardo Gonçalo
Physics Analysis Tools session
ATLAS Software Workshop – Munich, March 2007

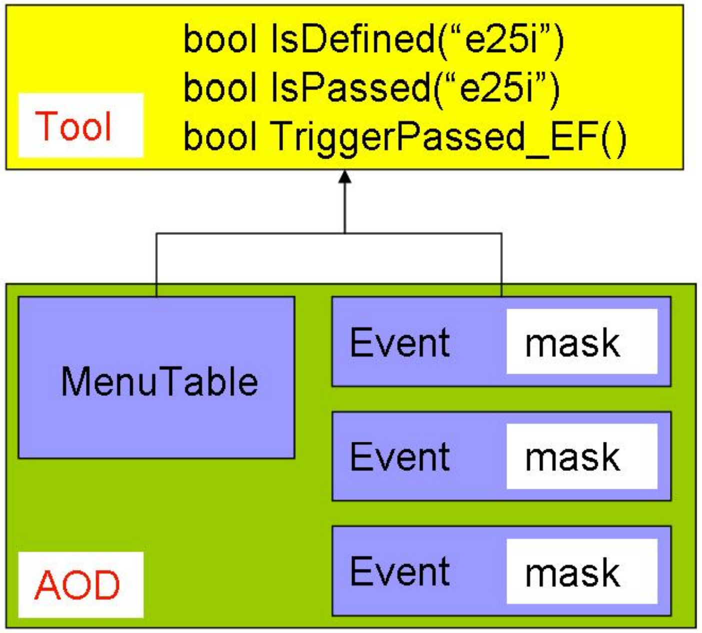
The old TriggerDecision

- Up to 12.0.6: `TrigSteeringEvent/TriggerDecision`
  - ☑ Caters for needs of most physics analyses
    - ☑ Good reception from physics-analysis users
  - ☑ Simple interface
  - ☑ Self-contained object (simple map<string, bool> for each level)
  - ✖ Overloads event with configuration information (simple map<signature, state> for each level; signature same for all run)
  - ✖ Slightly clunky in trigger re-run from POOL AOD:
    - ✖ More than one TriggerDecision object per event retrieved "by hand"
  - ✖ No access to trigger EDM objects or prescale information

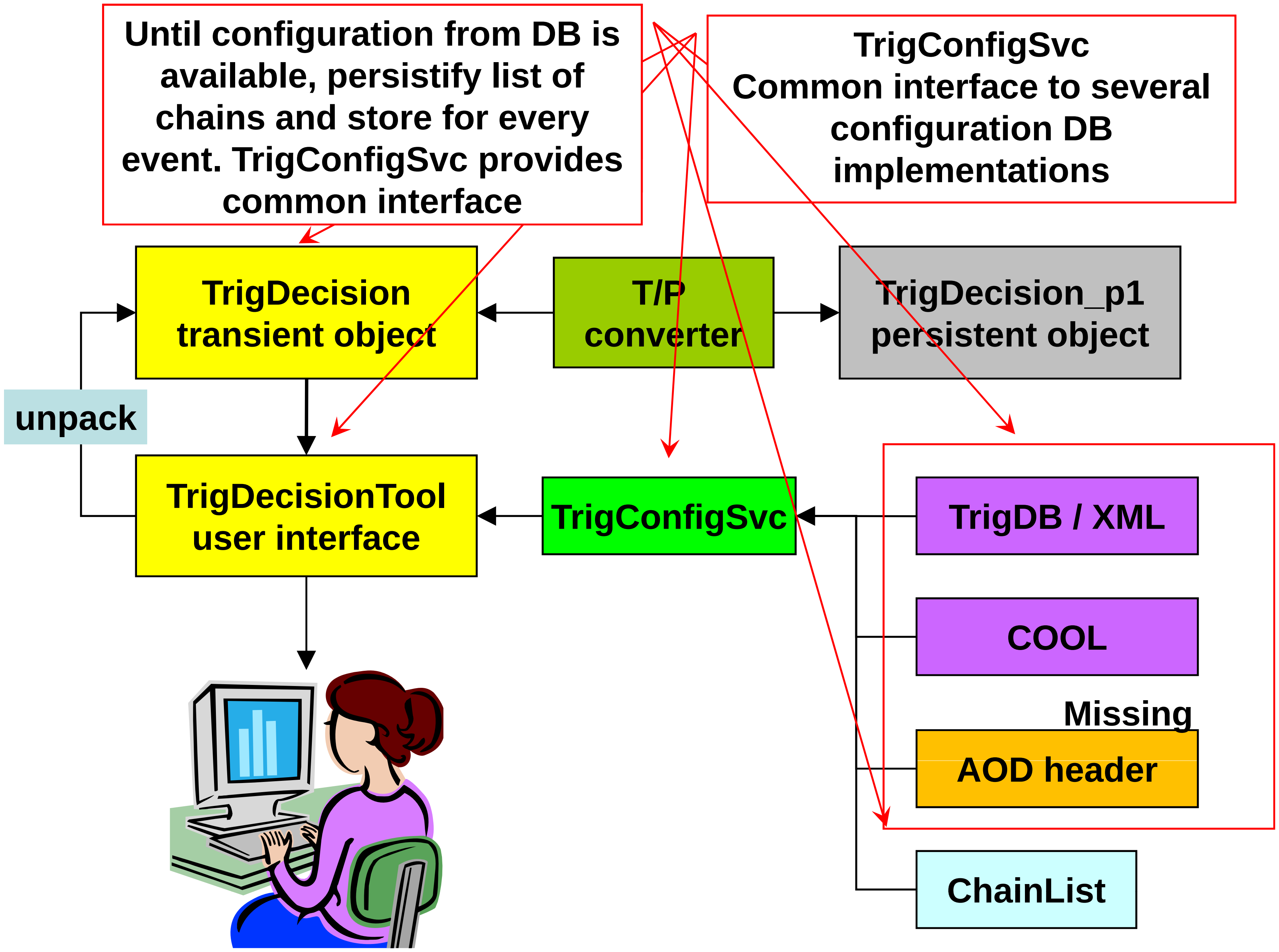
The new TrigDecision

- In 13.x.x new `TrigEvent/TrigDecision` package

  - ☑ Provides functionality of old TriggerDecision
  - ☑ Designed to work with new Steering
  - ☑ Provides more functionality:
    - ☑ Access to configuration: LVL1CTP::Lvl1Items, HLT::Chains, HLT::Signature, prescale factors
    - ☑ Access to trigger objects through Navigation: triggerElements and features (tracks, TrigElectron, …)



  - ☑ Configuration information stored per file/luminosity block, not per event
  - ✘ Interface more complicated than before
    - ☑ Working on implementing/simplifying accessors
  - ✘ No longer a self-contained object
    - ✘ Needs Tool to interpret it
    - ☑ Tool configuration can be used to hide clunky retrieving of decision in trigger re-run



Until configuration from DB is available, persistify list of chains and store for every event. TrigConfigSvc provides common interface

TrigConfigSvc
Common interface to several configuration DB implementations

TrigDecision transient object
T/P converter
TrigDecision_p1 persistent object
unpack
TrigDecisionTool user interface
TrigConfigSvc
TrigDB / XML
COOL
Missing
AOD header
ChainList

How to use it: example

- Instantiate TrigDecisionTool: this will “point” at a certain TrigDecision transient object – TrigDec::Last in the example (*) – to accommodate re-running the trigger on POOL files
  - IoVSvc used by TrigDecisionTool to point at the right TrigDecision object for each new event.

```
MyAlgorithm::initialize() {
  StatusCode sc = toolSvc()->retrieveTool(“TrigDecisionTool”,
                                           “TrigDecisionTool”,
                                          m_trgTool, TrigDec::Last); // (*)
...
```

(*) currently being implemented

Summary and plans

- Trigger-user interface rewritten for new Steering
- Also need a self-contained form for use in SAN/pAOD: TrigDecisionSAN
  - Mimics TrigDecisionTool but is self-contained
  - Like old TriggerDecision but with TrigDecisionTool accessors
- Now compiling in nightlies; soon to be usable for tests
  - Some methods still missing
  - Accessors to cover most common use cases being written
    - e.g. to retrieve all features of some class T using single method: this can use other accessor methods already in place
  - To be fully debugged in rel.13.0.0

TrigDecisionTool accessors

```
bool isPassed(TrigLevel lvl);
bool isPassed(TrigLevel lvl, unsigned int chain_counter);
bool isPassed(std::string chain_name);

bool isConfigured(TrigLevel lvl);
bool isConfigured(TrigLevel lvl, unsigned int chain_counter);
bool isConfigured(std::string chain_name);

const LVL1CTP::Lvl1Item* getLvl1Item(unsigned int item_counter);
const LVL1CTP::Lvl1Item* getLvl1Item(std::string item_name);
const HLT::Chain* getHLTChain(TrigLevel lvl, unsigned int chain_counter);
const HLT::Chain* getHLTChain(std::string chain_name);

const std::vector<LVL1CTP::Lvl1Item*>& getConfiguredItems();
const std::vector<HLT::Chain*>& getConfiguredChains(TrigLevel lvl);

int getFullChainPrescaleFactor(TrigLevel lvl, unsigned int chain_counter);
int getFullChainPrescaleFactor(std::string chain_name);

const HLT::Signature* getChainSignature(TrigLevel lvl, unsigned int chain_counter, int step = -1);
const HLT::Signature* getChainSignature(std::string chain_name,  int step = -1);

const std::vector<const HLT::TriggerElement*>* getChainTEs(TrigLevel lvl, unsigned int chain_counter, int step = -1);
const std::vector<const HLT::TriggerElement*>* getChainTEs(std::string chain_name, int step = -1);

std::string getTELabel( const HLT::TriggerElement* te ) const;

HLT::Navigation* getNavigation();

template<class T> HLT::ErrorCode getFeature(const HLT::TriggerElement* te, const T*&  feature, cost std::string& label = "");
template<class T> HLT::ErrorCode getFeatures(const HLT::TriggerElement* te, std::vector<const T*>&  feature, const std::string& label = "");

bool isTriggerPassed();
```

NOT FINAL INTERFACE

How to use it: example 2

- Retrieve feature and look at eta phi …

```
...
MyAlgorithm::execute() {
  if ( m_trgTool->isConfigured("tau25_xE30_L2") ) {
   // retrieve handle to Trigger chain:
   const HLT::Chain* chain1 = m_trigDec->getHLTChain("tau25_xE30_L2");
   // retrieve all TriggerElements (TE) of last step of chain (= -1)
   const std::vector<const HLT::TriggerElement*>*
                             comb = m_trigDec->getChainTEs("tau25_xE30_L2", -1);
   // loop over all TEs
   for (std::vector<const HLT::TriggerElement*>::const_iterator
        te = comb->begin(); te != comb->end(); ++te) {

      // retrieve RoI descriptor feature attached to this TE
      std::vector<const TrigRoiDescriptor*> rois;
      if (m_trigDec->getFeatures(*te, rois) != HLT::OK) continue;
      // print eta, phi for each RoI descriptor object
      for (std::vector<const TrigRoiDescriptor*>::const_iterator
           roi = rois.begin(); roi != rois.end(); ++roi) {
          log << MSG::INFO << "eta: " << *roi)->eta0() << ", phi: "
              << (*roi)->phi0() << ") ";
      }

...
```