Separate Calibrations for Simulation and Reconstruction

Leon R.

Core Software Meeting

24-October-2006

Rationale

- Our calibrations are approximate.
- We want to model this by having simulation and reconstruction calibrations that don't quite match.
- We can use calibration flavors to implement this.
  - For production:
    - Simulation and reconstruction are different jobs, so there's no issue.
  - For development, test, user jobs, etc:
    - A two-step job is inconvenient.
    - Display of MC event from mc.root is compromised.

TKR Calibration Strategy

- Each calibration is handled by a service, which is passed a pointer to a CalibData object. (A service can handle more than one calibration type, but this is not relevant for the discussion.) Each service has an update() method that can be used to pass a pointer, and signal any local processing.
- Once per event, a tracker Algorithm, TkrCalibAlg, checks to see if all the calibrations are in their valid ranges, and if not, prompts CalibSvc for a new calibration, and updates the appropriate service with the new pointer.
  - This is invoked in a Gaudi sequence, but there is currently only one member.

```
...
“Sequencer/CalibUpdate”
...
CalibUpdate.Members = {“TkrCalibAlg”};
```

TkrCalibAlg
(in TkrUtil package)

- Controls all the calibrations
- Initialized from jobOptions
  - All calibration types can be set to the same flavor

```
TkrCalibAlg.calibFlavor = “DC2”;
// Default = “ideal”
```

  - The common assignment can be over-ridden for each constant type individually

```
TkrCalibAlg.deadStripsFlavor = “myTest”;
// Default = “notSet”
```

Multiple Calibrations: Simple Case (example: TkrToTSvc in TkrUtil)

In this case, constants are retrieved from the calibration file by the service, without any further processing:

- Put an instance of TkrCalibAlg into each of two Gaudi sequences:

```
...
SimCalibUpdate.Members = {“TkrCalibAlg/TkrSimCalib”};
RecCalibUpdate.Members = {“TkrCalibAlg/TkrRecCalib”};
...
```

```
 ...
“Sequencer/SimCalibUpdate”,
 ... Digi stuff…
...
“Sequencer/RecCalibUpdate”,
 ... Recon stuff…
...
```

- Each instance is initialized separately:

```
TkrSimCalib.calibFlavor = “vanilla”;
TkrRecCalib.calibFlavor = “vanilla”;
TkrRecCalib.deadStripsFlavor = “myTest”;
```

- The algorithm must update the data pointer every time it’s called, because it has no way of knowing what the other instance (if any) has done.

Multiple Calibrations: More Complicated Case (example: TkrBadStripsSvc in TkrUtil)

- In this case, the service processes the data in the calibration file and produces locally stored information.
- Two approaches:
  - Duplicate the service
  - Duplicate the local storage
- I've chosen the latter (it seemed more maintainable…)

Add a method to the service, called by TkrCalibAlg

Set a flag in the service, depending on called name of the TkrCalibAlg.

```
// look for "Rec" in the name
 if(name().find("Rec")!=std::string::npos) {
     m_pTkrBadStripsSvc->setCalType(ITkrBadStripsSvcCalib::REC);
     m_pTkrFailureModeSvc->setCalType(ITkrFailureModeSvcCalib::REC);
 } else {
     m_pTkrBadStripsSvc->setCalType(ITkrBadStripsSvcCalib::SIM);
     m_pTkrFailureModeSvc->setCalType(ITkrFailureModeSvcCalib::SIM);
 }
```

- The service uses the flag to choose one of the sets of locally maintained data to return values, and to update if necessary.
- The service needs to be updated only if the pointer changes, unless the service tracks the changes itself. (The latter is probably better.)
- This code is backwards-compatible with the standard Gaudi sequence.