Core Software:
AthenaMT (was AthenaHive)

Charles Leggett

November 9 2015

ATLAS Software Technical Meeting

AthenaMT Status

- Moving build from mig1 to 20.8.X
  - will have real validation nightlies
  - just cut 20.8.1
  - about to move from gcc48 to gcc49
- Incorporates recent changes to propagation of Algorithm and AlgTool data dependencies
  - automatic for VarHandles
  - explicit declarations for StoreGate::retrieve/record:

```
MyAlg.ExtraInputs = [('MyTrackClass','TrackKey1'), ('MyHitClass','key2')]
```

  - Shared base class for Atlas / Gaudi
- SGInputLoader now managed via generic jobOptions instead of a specialized Algs:
  - `SGInputLoader.Load = [('Class1','key1'), ('Class2','key2'),... ]`
- tutorial wiki updated:
  - https://twiki.cern.ch/twiki/bin/view/AtlasComputing/AthenaHiveTutorial201509

Upcoming Gaudi Changes

- Merge of Hive with trunk - v27
  - should have a discussion as to when to use this in regular Athena builds
- new concurrency states:
  - non-concurrent
  - cloneable
  - re-entrant
- decision made NOT to have enforced const access to Public tools as a stepping stone
  - all Tools must be Private or migrated to a Service

DataHandles

- Desired features:
- Automatic declaration of data dependencies of Algorithms and AlgTools
  - dependencies of chained classes are propagated to parent
- Provide generic mechanism to inform Scheduler when an object in the store is ready for downstream use
- Allow manipulation of attributes in the jobOptions
- Syntax consistent with other Handles
- Share functionality between Athena and Gaudi

DataHandles

- Using VarHandles in an AthAlgorithm

```
class MyAlg : public AthAlgorithm {
…
private:
   SG::ReadHandle<MyTrack> m_track;

   ToolHandle<ITrackRefitter> m_trackTool;
};

MyAlg::MyAlg(const std::string& name, SvcLocator* pSvcLocator ) :
  AthAlgorithm( name, pSvcLocator ),
  m_track("TheDefaultKey"),
  m_trackTool("TheTrackRefitter",this) {

  declareProperty("Track", m_track, "the track");
  declareProperty("TrackTool, m_trackTool, "the track tool");
}

StatusCode MyAlg::initialize() {
   if (m_trackTool.retrieve().isFailure()) { OOPS! do something }
}

StatusCode MyAlg::execute() {
   if ( ! m_track.isValid() ) { OOPS! do something; return; }

   int nhits = m_track->hits();
   m_trackTool.refit();
}
```

DataHandles

- Using VarHandles in a Tool

```
class MyAlg : public AthAlgorithm {
…
private:
   SG::Re

   ToolHa
};

MyAlg::My
  AthAlgo
  m_track
  m_track

  declare
  declare
}

StatusCod
   if (m_
}

StatusCod
   if ( !m_track.isValid() ) { oops! }

   int nhits = m_track->hits();
   m_trackTool.refit();
}
```

```
class TrackRefitter : public AlgTool {
…
private:
   SG::ReadHandle<Track> m_origTrack;
   SG::WriteHandle<RefittedTrack> m_refit;
};

TrackRefitter::TrackRefitter(type,name,parent):
   AlgTool(type,name,parent), m_origTrack("TheDefaultKey"),
   m_refit("TheRefittedKey") {

   declareProperty("Track", m_origTrack, "orig track");
   declareProperty("RefittedTrack", m_refit, "refit track");
}

void TrackRefitter::refit() {
   m_refit = CxxUtils::make_unique< RefittedTrack >
             ( m_origTrack.refit(somePar) );
}
```

DataHandles

- Setting up the jobOptions

```
class MyAlg : public AthAlgorithm {
…
private:
   SG::Re
   ToolHar
};
```

```
class TrackRefitter : public AlgTool {
…
private:
   SG::ReadHandle<Track> m_origTrack;
   SG::WriteHandle<RefittedTrack> m_refit;
```

```
# jobOptions

from MyPackage.MyPackageConf import MyAlg
if ( we_need_to_update_the_key_for_MyAlg_Track ) :
   MyAlg.Track = "NewKey"
   MyAlg.TrackTool.RefittedTrack = "NewRefittedTrackKey"
```

MyAlg Dependencies:
Input: ('**Track**','**NewKey**')
Output: ('**RefittedTrack**','**NewRefittedTrackKey**')

```
   int nhits = m_track-
   m_trackTool.refit();
}
```

DataHandles

- Declaring non-VarHandle dependencies

```
class MyAlg : public AthAlgorithm {
…
```

Dependencies:
ClassID (int) or ClassName
python tuple of (ClassID,Key)

```
er : public AlgTool {
```

```
# joboptions

from MyPackage.MyPackageConf import MyAlg
if ( we_need_to_update_the_key_for_MyAlg_Track ) :
   MyAlg.Track = "NewKey"
   MyAlg.TrackTool.RefittedTrack = "NewRefittedTrackKey"


# add some non-VarHandle dependencies
MyAlg.ExtraInputs  = [ ('Track','SomeKey'), ('7777','Key') ]
MyAlg.ExtraOutputs = [ ('Track','SomeKey'), ('Track','AnotherKey') ]
```

```
   if ( !m_track.isValid() ) { OOPS! do something, return, }

   int nhits = m_track->hits()
   m_trackTool.refit();
}
```

In theory, we can have multiple objects in the store with the same ClassID but different Keys

DataHandles

- Preload Disk Resident data

```
class MyAlg : public AthAlgorithm {
…
private:
   SG::Re
   ToolHar
```

```
class TrackRefitter : public AlgTool {
…
private:
```

```
# jobOptions
```

Force an ERROR if there are unmet dependencies

```
                                  import MyAlg
                                  ey_for_MyAlg_Track ) :
   MyAlg.Track = "NewKey"
   MyAlg.TrackTool.RefittedTrack = "NewRefittedTrackKey"

# add some non-VarHandle dependencies
MyAlg.ExtraInputs  = [ ('Track','SomeKey'), ('7777','Key') ]
MyAlg.ExtraOutputs = [ ('Track','SomeKey'), ('Track','AnotherKey') ]

# preload Disk data
from SGComps.SGCompsConf import SGInputLoader
topSequence += SGInputLoader(ShowEventDump=False)
topSequence.SGInputLoader.Load = [ ('EventInfo','McEventInfo') ]

svcMgr.ForwardSchedulerSvc.CheckDependencies = True
```

GaudiKernel/DataHandle

```
class DataHandle {
public:

  enum Mode {
    Reader = 1<<2,
    Writer  = 1<<4,
    Updater = Reader | Writer
  };

  DataHandle(const DataObjID& k, Mode a=Reader, IDataHandleHolder* owner=0):

  virtual void setOwner(IDataHandleHolder*);
  virtual IDataHandleHolder* owner() const;

  virtual Mode accessType() const;

  virtual void setKey(const DataObjID& key);
  virtual void updateKey(const std::string& key);

  virtual const std::string& objKey() const;
  virtual const DataObjID& fullKey() const;

  virtual void reset(bool);
  virtual StatusCode commit();

};
```

StoreGate/VarHandle

```
class VarHandleBase : public IResetable, public Gaudi::DataHandle {

  bool isConst() const;
  bool isInitialized() const;

  virtual bool isSet() const override final;
  StatusCode setState() const;

  virtual CLID clid() const =0;
  std::string store() const;
};



template <class T>
class ReadHandle : public SG::VarHandleBase {

  const T* operator->() const;
  const T& operator*() const;

  virtual bool isValid() const;

};
```

Un-templated base class for Property manipulation

Gaudi/DataObjectHandle

```
class DataObjectHandleBase : public Gaudi::DataHandle {

  bool isOptional() const;

  const std::vector<std::string> & alternativeDataProductNames() const
  void setAlternativeDataProductNames(const std::vector<std::string> &)

  bool initialized() const;
  bool wasRead() const;
  bool wasWritten() const;

  bool isValid() const;

  void setRead(bool wasRead=true)
  void setWritten(bool wasWritten=true);
  void init();

};

template <typename T>
class DataObjectHandle : public DataObjectHandleBase {

  T* get() { return get(true); }
  T* getIfExists() { return get(false); }
  bool exist() { return get(false) != NULL; }
  T* getOrCreate();
  void put (T* object);

};
```

Un-templated base class for Property manipulation

IDataHandleHolder

- Algorithm and AlgTool inherit from IDataHandleHolder

```
class GAUDI_API IDataHandleHolder : virtual public INamedInterface {
public:
   virtual ~IDataHandleHolder() {};

   virtual std::vector<DataHandle*> inputHandles()  const = 0;
   virtual std::vector<DataHandle*> outputHandles() const = 0;

   virtual const DataObjIDColl& extraInputDeps()  const = 0;
   virtual const DataObjIDColl& extraOutputDeps() const = 0;

   virtual void accept(IDataHandleVisitor*) const = 0;

   virtual void commitHandles() = 0;

protected:
   virtual void declareInput (DataHandle*) = 0;
   virtual void declareOutput(DataHandle*) = 0;

};

class IDataHandleVisitor {
public:
  virtual void visit(const IDataHandleHolder*) = 0;
};
```

for deps that are not DataHandles

called after component has finished executing

Algorithm.h

```
class GAUDI_API
Algorithm: public implements4<IAlgorithm, IDataHandleHolder, IProperty, IStateful> {

 public:
  virtual std::vector<IDataHandle*> inputHandles()  const;
  virtual std::vector<IDataHandle*> outputHandles() const;

  virtual const DataObjIDColl& extraInputDeps()  const;
  virtual const DataObjIDColl& extraOutputDeps() const;

  virtual void accept(IDataHandleVisitor*) const ;

  const DataObjIDColl& inputDataObjs()  const;
  const DataObjIDColl& outputDataObjs() const;

 protected:
  virtual void declareInput (IDataHandle* im);
  virtual void declareOutput(IDataHandle* im);

 private:
  std::vector<IDataHandle*> m_inputHandles, m_outputHandles;
  DataObjIDColl m_inputDataObjs, m_outputDataObjs;

  DataObjIDColl m_extInputDataObjs, m_extOutputDataObjs;
```

for non-DataHandle dependencies

same goes for **AlgTools**

Algorithm.h and AlgTool.h

```
public:
  template <class T>
  Property* declareProperty(const std::string& name, ToolHandle<T>& hndl,
                            const std::string& doc = "none" ) const {
    // ugh!
    Algorithm* a = const_cast<Algorithm*>(this);
    a->declareTool(hndl).ignore();

    return m_propertyMgr->declareProperty(name, hndl, doc);

  }


  template<class T>
  StatusCode declareTool(ToolHandle<T> &handle, std::string toolTypeAndName = "",
                         bool createIf = true) {
    if (toolTypeAndName == "") toolTypeAndName = handle.typeAndName();

    StatusCode sc = handle.initialize(toolTypeAndName,
                                      (handle.isPrivate() ? this : 0 ), createIf);
    m_toolHandles.push_back(&handle);

    if (sc.isSuccess()) {
      debug() << "Handle for " << (handle.isPublic() ? "public":"private") << " tool "
              << toolTypeAndName  << " successfully created and stored." << endmsg;
    } else {
      error() << "Creating Handle for tool " << toolTypeAndName << endmsg;
    }
    return sc;
  }
```

Algorithm.cpp

```
StatusCode Algorithm::Algorithm(name, svcLoc, version) {

 …

/// declare non-DataHandle inputs

  declareProperty( "ExtraInputs",  m_extInputDeps)
  declareProperty( "ExtraOutputs", m_extOutputDeps)
…
}
```

Algorithm.cpp

```
StatusCode Algorithm::sysInitialize() {

 …
  DataHandleHolderVisitor avis(m_inputDataObjs, m_outputDataObjs);
  accept(&avis);

  log << MSG::DEBUG << "Data Deps for " << name();
  for (auto h : m_inputDataObjs)  { log << "\n  + INPUT  " << h; }
  for (auto h : m_outputDataObjs) { log << "\n  + OUTPUT " << h; }
  log << endmsg;

  return sc;
}

void Algorithm::accept(IDataHandleVisitor *vis) const {
  vis->visit(this);

  // loop through tools
  for (auto tool : tools()) {
    AlgTool* at = dynamic_cast<AlgTool*>(tool);
    vis->visit(at);
  }

  // loop through sub-algs
  for (auto alg : *subAlgorithms()) {
    vis->visit(alg);
  }

}
```

DataHandleHolderVisitor

```
class DataHandleHolderVisitor : public IDataHandleVisitor {
public:

  DataHandleHolderVisitor(DataObjIDColl& ido, DataObjIDColl& odo);

private:
  DataObjIDColl &m_ido, &m_odo;
};

void
DataHandleHolderVisitor::visit(const IDataHandleHolder* idhh) {

  if (idhh == 0) return;

  // add all the deps that are DataHandles
  for (auto h : idhh->inputHandles()) {
    m_ido.insert(h->fullKey());
  }
  for (auto h : idhh->outputHandles()) {
    m_odo.insert(h->fullKey());
  }

  // deal with DataObjs that aren't DataHandles
  m_ido.insert(idhh->extraInputDeps().begin(),  idhh->extraInputDeps().end());
  m_odo.insert(idhh->extraOutputDeps().begin(), idhh->extraOutputDeps().end());

}
```

GaudiAlgorithm

```
public:
  using Algorithm::declareProperty;
  template <class T>
  Property* declareProperty ( const std::string&       name,
                              DataObjectHandle<T>&     hndl,
                              const std::string& doc = "none" ) const {

    if ( hndl.mode() & Gaudi::DataHandle::Reader ) {
      // ugh!
      (const_cast<GaudiAlgorithm*>(this))->Algorithm::declareInput(&hndl);
    }


    if ( hndl.mode() & Gaudi::DataHandle::Writer ) {
      // ugh!
      (const_cast<GaudiAlgorithm*>(this))->Algorithm::declareOutput(&hndl);
    }

    if (hndl.owner() == 0) {
      hndl.setOwner((const_cast<GaudiAlgorithm*>(this)));
    }

    return m_propertyMgr->declareProperty(name, hndl, doc);
  }
```

similarly for **AthAlgorithm** and **AthAlgTool**

Properties

- SG::VarHandle
  - VarHandleProperty class
    - provides `Gaudi::Parsers::parse(VarHandleBase&, string&)`
- GaudiKernel/DataObjectHandle
  - DataObjectHandleProperty class
    - provides `Gaudi::Parsers::parse(DataObjectHandleBase&, string&)`
  - python manipulation stuff in GaudiKernel/python to allow setting of other DataObjectHandle attributes
    - **`Path`**
    - **`Mode`**
    - **`AlternativePaths`**
    - **`Optional`**

DataHandles: Next Step

- Full functionality for DataHandles, data dependency propagation, and Property managing is now in place
  - 20.8.X

- Need to migrate user code and jobOptions to VarHandles
  - start with Calorimeter hive testbed
    - ~10 Algorithms
    - ~30 AlgTools
    - ??? jobOption files....
  - looking for volunteers to help
    - so far Scott, Jovan and I have signed up

Tasks

- clang build for static analyzer [SCOTT]
- migrate static analysis to gcc [SCOTT]
- Package migrations
  - split up by Reco domain
  - VarHandles
  - public Tools -> private / const / Service
  - thread safety
  - start with CaloHive testbed [JOVAN, SCOTT, CHARLES]
    - muons?
  - talk to reco managers [GRAEME]
- Service thread safety
  - Random numbers [JOHN,PAOLO]
  - Mag Field [Started: ELMAR]
  - Geometry
  - FileMgr [CHARLES]
  - Persistency [PETER, DAVID]
- **Conditions**
  - **GeoModelSvc**
  - **investigate implications of implementation style**
  - **multiple instances of detector store**
  - **event processing synchronization barrier**
  - **IOVSvc and callbacks**
  - **interactions with MetaData**
- Shared cache object [CHARLES]
  - lifetime of lumi/run/job
  - automatic merge at end
- Event Views [BEN]
  - extension / subclassing EventContext
- How to remove SGInputLoader [PETER, PAOLO]
  - preload all objects at start of event that have unmet READ dependencies?
- Rewrite ToolSvc : public tools have only const access [CHARLES]
- I/O
  - shared reader / writer
- IncidentSvc: Schedulable Incidents
  - FileIncidents
  - interaction with EventView
  - migrate Begin/EndEvent Incident to State Transition
    - open JIRA ticket [CHARLES, PETER]
    - Marco: transform these types of Incidents to "Processing Phases", handle all Algs/clients in one place