PAT core development status and plans

Author: *Sebastien Binet*
Institute: *LAL/IN2P3*
Date: *2010-11-30*

Outline

- versioned objects (aodfix)
- userdatasvc
- thinningsvc and thinning truth
- associations/matching

StoreGateSvc and versioned objects

- ability to record and retrieve different versions of an object, with the **same** key
  - need and specs described in `Reco Task Force` document (`soft-2003-010`)
  - Paolo implemented this a few years ago
- main methods:
  - `StoreGateSvc::retrieveHighestVersion`
  - `StoreGateSvc::retrieveAllVersions`
- a few clients already
  - `MetaDataSvc`
  - `EventBookkeeperMetaDataTool`
  - `LumiBlockMetaDataTool`
  - `IOVDbMetaDataTool`
  - `MonteCarloReactTools`

Recording multiple versions

- in a nutshell:

```
SG::VersionedKey myKey("aVersObj", 77);
evtStore()->record(new Foo(data), (std::string)myKey);

SG::VersionedKey my2Key("aVersObj", 88);
evtStore()->record(new Foo(data2), (std::string)my2Key);
```

- note that the usual `retrieve` will return the **first** recorded object, independant of its version
  - ie: version `77` in the above example
- note that all versions of an object will be written on disk
  - not sure if this can be disabled
- note that “only” 100 versions are supported (0-99)

Retrieving multiple versions / last version

- to retrieve the last version:

```
SG::ObjectWithVersion<Foo> highest;
const std::string baseKey("aVersObj");
evtStore()->retrieveHighestVersion(highest, baseKey);
assert(highest.versionedKey.version() == 88);
highest.dataObject->someFooMethod();
```

- to retrieve all versions:

```
std::list<SG::ObjectWithVersion<Foo> > allVersions;
evtStore()->retrieveAllVersions(allVersions, baseKey);
```

- the list is ordered by **ascending** version number
  - ie: `allVersions().back().versionedKey.version()==88`

Status and plans

- “basic” functionality implemented
- already well tested in the field (lumiblock,metadata,...)
- plan is to have the full `AODFix` use case supported
  - track which version was made with which fixing-tool
  - store this in some kind of metadata
    - how ?
    - in what form ?

UserDataSvc

- `twiki:Atlas/UserDataSvc`
- a service to add user-defined informations to an object, an event and/or a file
- the user data is stored in a `TTree` in the same file than the decorated data
- decoration data can be:
  - builtins (`float, double, int, ...`)
  - class instances (for which a proper `Reflex` dict. has been generated)

- in a nutshell:

```
svc->decorateEvent("evtlvl_deco", udata).isSuccess();
svc->decorateFile("filelvl_deco", udata).isSuccess();
svc->decorateElement(*ibarcode,    // IAthenaBarCode
                     "objlvl_deco",
                     udata).isSuccess();
```

Retrieving user-data decoration

- in a nutshell:

```
svc->getFileDecoration ("filelvl_deco", udata).isSuccess();
svc->getEventDecoration("evtlvl_deco",  udata).isSuccess();
svc->getElementDecoration(*ibarcode,
                          "objlvl_deco",
                          udata).isSuccess();
```

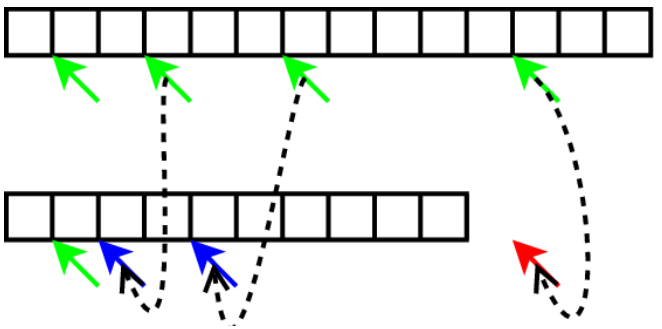
- note: `getInMemXyz` methods to get file/event/element level decoration from the current in-memory userdata created during the current job

Status and plans/ideas - UserDataSvc

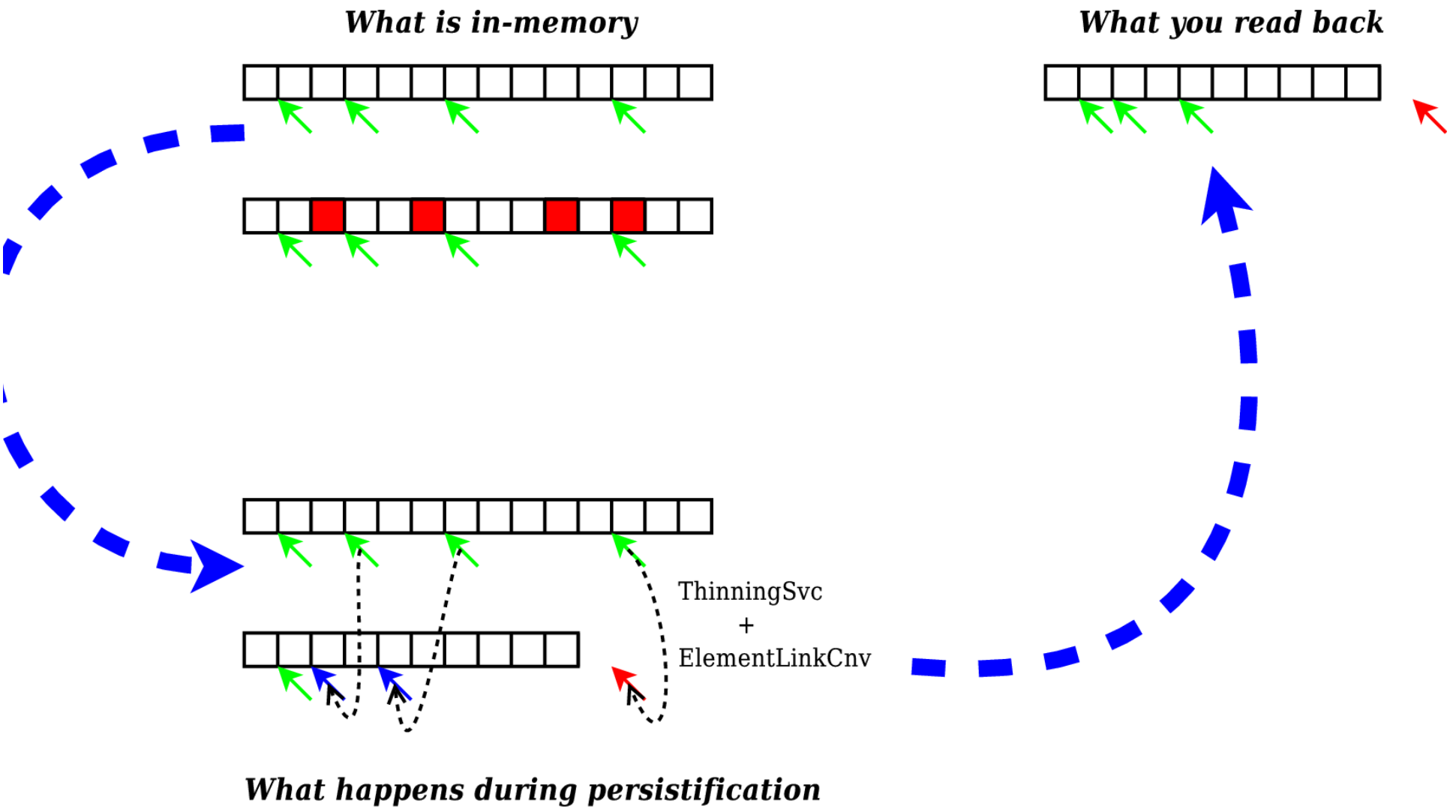
- create/read user data in `Athena`
- read user data from `PyROOT/Py-ARA`
- request to also be able to read user data from `C++ARA`
- request to simplify the API
  - have `getInMemXyz` and `getInFileXyz`
  - `getXyz` would first try `getInMemXyz` and then `getInFileXyz`
  - reducing/simplifying the `IUserDataSvc` interface would also allow a not insane `PyAthena.IUserDataSvc`
- file merging should also work when a `UserDataTree` is present
  - usual issue of handling the merging of file-level user data
- ability to apply element-level decoration is tied to the element actually implementing the `IAthenaBarCode` interface
  - need to implement `IAthenaBarCode` for those classes (e.g. `CaloCluster`, `TrackParticle`)
  - streamline the implementation of `IAthenaBarCode`

ThinningSvc

- `twiki:Atlas/ThinningSvc`
- allow to remove elements from `DataVectors` and `IDC` collections
  - while preserving the consistency of `ElementLinks`
  - w/o duplicating those collections (ie: one filtered+one pristine)
- all the magic happens at `T->P`
  - objects marked for removal are removed from the collection
    - and backed up on a temporary address location
  - the collection is compacted and serialized
  - other objects holding `ElementLinks` to objects from within that collection are automagically modified by the `T/P` converter of the `ElementLinks`



Thin-on-write



Usage

- in a nutshell:

```
std::vector<bool> filter_mask;
// fill filter_mask according to some predicate/selection
svc->filter(particles, filter_mask).isSuccess();

// applying an OR of the previous+current mask
svc->filter(particles, filter_mask,
            IThinningSvc::Operator::Or).isSuccess();
```

Status and plans - ThinningSvc

- works both in `Athena` and `PyAthena`
- works with multiple output streams
  - no cross-talk
- no easy-way to thin `TruthParticleContainer` with the `ThinningSvc`
  - current recipe is to thin directly the `McEventCollection` with the `McParticleTools` toolkit
  - would be great to have `TruthParticleContainer` directly thin-able
  - would be great to have this also integrated-with/steered-by the `MCTruthClassifier`
    - ⋆ also handle decay-trees of selected particles
- integration with `D3PDMakers` ?
  - register a `D3PDOutputStream` with the `ThinningSvc`
  - have the d3pd dumper tool honor the thinning of collections
    - ⋆ not sure if this can be done centrally and automagically for all d3pd tools
- integration with `AssociationTools` and `INav4MomAssocs` ?

Associations and matching

- map of persistifiable pointers to collection of persistifiable pointers
  - `DataModel/AssociationMap.h`
- beautification for the “special” case of `INavigable4Momentum`
  - `NavFourMom/INav4MomAssocs.h`
- examples:
  - `PhysicsAnalysis/AssociationBuilder/AssociationComps`
  - `PhysicsAnalysis/AtlfastConverters/AtlfastConversionTools`
- in a nutshell:

```
typedef ElementLink<INavigable4MomentumCollection>
        INav4MomLink_t;
INav4MomAssocs m;
INav4MomLink_t mc_ele ("mckey",  mc_ele_idx);
INav4MomLink_t rec_ele("reckey", rec_ele_idx);
m.addAssociation( mc_ele.getDataPtr(),  mc_ele.index(),
                  rec_ele.getDataPtr(), rec_ele.index());
```

- old class, API/implementation a bit crufty
  - this mostly comes from manipulating `ElementLinks`...

Status and plans - Associations

- t/p cnv for `INav4MomAssocs`
  - can handle any `INav4Mom` association
  - other use cases ? (ie: something not inheriting from `INav4Mom`)
- API to iterate over associations, add associations
  - capable
  - but cumbersome: definitely place for improvement(s) - efficiency-wise too (expose `ElementLink` would reduce silly overheads)
  - adding an association should be as easy as:

```
m[mc_ele_link].push_back(rec_ele_link);
```

- allow to automatically (or provide helper functions to) create a `ThinningSvc` filter-mask from any `AssociationMap` ?
- integrate with `D3PDMaker` tools ?
- somewhat some overlap b/w `UserDataSvc`, `ThinningSvc` and `AssocMap`
  - would be interesting to think on ways on how to unify/consolidate a bit more these tools

Other core aspects

- `PyAthena` and `Py-StoreGate`
  - no (known to me) needed feature
  - apart from improving runtime speed
  - to address this:
    - a `python` -> `C` compiler (`cython`) is being integrated into `PyAthena`
    - Wim is also working on a `Just-In-Time` compiler (`PyPy`) to do the same)
- anything else ?