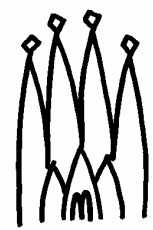
9
Associators

How to relate objects to each other

Create relations

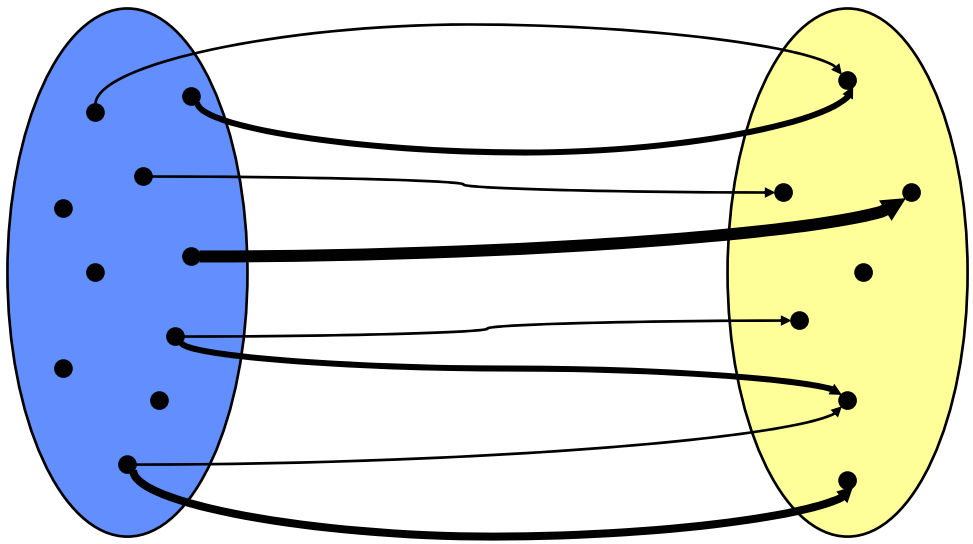
Save relations

Use relations: Associators



Relations between objects

- **Which type of objects**
  - Any object: int, double, complex class, keyed/contained objects…
  - Most interesting: two sets of contained objects
- **What is a relation?**



Types of relations

- One or two directional (1D / 2D)
  - But reverse relations can always be retrieved from direct relations
  - Hence, only 1D relations are made persistent
  - Advice: create only 1D relations, unless both usages are frequent
- Normal relations
  - Simple link between objects
  - Not necessarily between all objects of each set
  - Possibly several links from/to an object
- Weighted relations
  - The link carries additional information (can be any class)
    - An ordering should be possible on the *WEIGHT* class
    - Example: int, double
    - But could be complex class with the == and < operators defined

How to create relations

1. Instantiate the relation table (in the creation algorithm)

```
#include "MyAssociator.h"
. . . .
new Table* table;   // The type "Table" is defined in MyAssociator.h
```

2. Usually one loops on all objects in the FROM set

```
for( from_iterator frIt=from.begin(); from.end()!=frIt; frIt++) {
```

3. For each object, decide which objects of the TO set to link to, possibly which weight.

```
double weight = computeWeight( frIt, toIt );
if( weight > 0 ) {  // Example of how to decide
```

4. Establish the relation

```
table->relate( *frIt, *toIt [, weight]);
```

How to save relations (1)

- Once the table is filled
- Optionally apply filters (if weighted)

```
FromObj* from;
ToObj* to;
Weight threshold;
table->filterFrom( from, threshold, {false,true});
// Keeps only relations with weight > (true) or > than a threshold
```

- Optionally remove some relations (all)

```
table->removeFrom( from );
table->removeTo( to );
```

How to save relations (2)

- **Declare the relations table in the transient store**

```
StatusCode sc =
 eventSvc()->registerObject( outputData(), table);
// outputData() returns the location in TES
// it should be declared as a property of the algorithm
```

- **If the table should be discarded (e.g. in case of error)**
  - **Do not forget to**

```
delete table; // avoid memory leaks!
```

How to use relations?

- In order to use relations, the user algorithm should use a Gaudi tool called an Associator
- Generic Associator tool available
- Guidelines for Associators
  - Specialise the associator (for ease of use)
  - New class derived from the class Associator
  - For weighted Associators: class AssociatorWeighted
- Where does the tool look for the table?
  - The tool looks in the TES
  - If not found, it tries and get it from the PES
  - If not found, one can define a construction algorithm which should save the relations table in the TES (at the location they are expected!)

Associators

- **Naming conventions**
  - **Type of the Associator tool**

    ```
    class FromObj2ToObjOtherInfoAsct : public
       Associator[Weighted]<FromObj,ToObj[,Weight]> { . . . };
    ```

    OtherInfo is optional (should not reflect the method used but the content)

  - **If ToObj and FromObj can be "factorised", do not repeat the common part in ToObj**

    ```
    class Particle2MCWeightedAsct;
    class ITCluster2MCParticleAsct;
    ```

  - **Type for the relations table**

    ```
    FromObj2ToObjOtherInfoAsct::Table
    ```

  - **Type for the Associator tool interface**

    ```
    FromObj2ToObjOtherInfoAsct::IAsct
    ```

Declaring an associator

- **In MyAssociator.h (note that "Weighted" is only in case of weighted relations)**

```
#include "Relations/AssociatorWeighted.h"
. . . .
class Particle2MCWeightedAsct :
  public AssociatorWeighted<Particle,MCParticle,double>
{
public:
  // Define data types
  // Define the relations table, templated class
  typedef RelationWeighted1D<Particle,MCParticle,double> Table;
  // Defines the type of the base associator
  typedef OwnType                                  Asct;
. . . .
// Minimal constructor
 Particle2MCWeightedAsct(const std::string& type, const std::string& name,
                const IInterface* parent )
    : Asct( type, name, parent) { };
}
```

Declaring an associator (2)

- Declare types for retrieving ranges of objects
  - When getting objects related to a given From (To) object, one gets a “range”
  - A “range” can be seen as a list/vector of objects
  - A “range” has an iterator, with the usual begin() and end() methods
  - For ease of use, one can define meaning full types for ranges, e.g.

```
typedef Particle2MCWeightedAsct::FromRange     ParticlesToMC;
typedef Particle2MCWeightedAsct::FromIterator  ParticlesToMCIterator;
typedef Particle2MCWeightedAsct::ToRange       MCsFromParticle;
typedef Particle2MCWeightedAsct::ToIterator    MCsFromParticleIterator;
```

- DLL file for loading the tool
  - MyAssociators_dll.cpp

```
#include "GaudiKernel/LoadFactoryEntries.h"
LOAD_FACTORY_ENTRIES(PhysAssociators)
```

Declaring an Associator (3)

- A _load.cpp file must be defined to declare the necessary factories
  - MyAssociators_load.cpp:

```
#include "DaVinciAssociators/Particle2MCWeightedAsct.h”
// Declare factory for the associator
DECLARE_TOOL_FACTORY( Particle2MCWeightedAsct );
// Declare factory for the relations table
DECLARE_OBJECT_FACTORY( Particle2MCWeightedTable );
. .
DECLARE_FACTORY_ENTRIES( PhysAssociators ) {
  DECLARE_OBJECT( Particle2MCWeightedTable );  // Declare the Table object
  DECLARE_TOOL( Particle2MCWeightedAsct );     // Declare the Associator tool
  DECLARE_ALGORITHM( Particle2MCWeighted );    // Declare the construction algorithm
}
```

Retrieving an Associator

- **An instance of the tool should be created in the user algorithm**
  - **Returns a pointer to an Associator interface (type Iasct*):**

```
Particle2MCWeightedAsct::IAsct* m_pAsctWithChi2; ///< Pointer to associator with chi2
    as weight
. . . .
 // This is the Particle2MCWeighted tool
  sc = toolSvc()->retrieveTool( "Particle2MCWeightedAsct",
                                m_pAsctWithChi2); ["MyAssociator",]

/// "Particle2MCWeightedAsct" is the type of the tool (as in _load)
/// m_pAsctWithChi2 is a pointer to the interface used later on
/// ["MyAssociator",] is an optional private name to that tool
```

Using an Associator

- **Retrieve a range of ToObj given a FromObj**

```
Particle* part = . . . ;
. . .
MCsfromParticle mcParts = m_pAsctWithChi2->rangeFrom( part );
MCsfromParticleIterator mcPartsIt;
for( mcPartIt = mcParts.begin(); mcParts.end() != mcPartIt; mcPartIt++) {
. . .
  // CAUTION: *mcPartIt is not of type MCParticle!!!
  MCParticle* mcPart = mcPartIt->to();
  Weight weight = mcPartIt->weight();
}
```

- **Similarly one can retrieve a range of FromObj given a ToObj**

```
Particle* part = . . . ;
. . .
ParticlesToMC parts = m_pAsctWithChi2->rangeTo( mcPart );
```

Using an Associator (2)

- **Often, relations are one-to-one between the two sets**
  - **Possibly no linked object, but never 2 or more**
  - **Shortcut to directly access the object:**

```
MCParticle* mcPartChi2;
double chi2;
mcPartChi2 = m_pAsctWithChi2->associatedFrom( *part[, chi2]);
if( mcPartChi2 ) {
  // There was an associated MCParticle
} else {
  // There was no associated MCParticle OR there was not relations table
}
```

Using an Associator (3)

- **Advanced usage of weighted associators**
  - One can retrieve relations which have a weight larger (smaller) than a threshold

```
Particle* part = . . . ;
. . .
MCsfromParticleChi2 mcParts =
  m_pAsctWithChi2->rangeWithHighCutFrom( part, maxChi2 );
// This will return a range containing only associated MCParticles
// if the weight (I.e. the chi2) is smaller than maxChi2
```

  - No one-to-one retrieval method with cut, but trivially

```
double chi2;
mcPartChi2 = m_pAsctWithChi2->associatedFrom( *part, chi2);
if( mcPartChi2 && chi2 < maxChi2) {
  // There was an associated MCParticle with chi2 < maxChi2
}
```

Using an Associator (4)

- **Miscellaneous features**
  - Testing if the relations table is present

```
if( false == m_pAsctChi2->tableExists() ) {
  // The table doesn't exist
} else {
  // One can retrieve information safely
}
```

  - Getting a status code when retrieving a range

```
Range range;
StatusCode sc = m_pAsct->rangeFrom( from, range);
if( sc.isSuccess() ) {
  // One can use range safely
}
```

JobOptions for Associators

- **Properties of the base class**
  - **No default: to be defined in the constructor using**
    set_property(name, value);
  - **Location of the relations table in the TES**
    Toolsvc.Particle2MCWithChi2Asct.Location = "Phys/Relations/Particle2MCWithChi2";
  - **Convention for the location name:**
    - **Root: the TES branch of the "To" objects**
    - **/Relations**
    - **Leaf: name of the Associator**
  - **Creation algorithm**
    ToolSvc.Particle2MCWithChi2Asct.AlgorithmType = "Particle2MCWithChi2";
    ToolSvc.Particle2MCWithChi2Asct.AlgorithmName = "Particle2MCWithChi2";
  - **Note: one can give an alternate name to the Associator and/or to the algorithm… The same Associator can be used twice with different settings**

JobOptions for Associators (2)

- **Example of dual usage of a single associator**
  - **In the code, retrieve the same tool with two different names**

```
// First the WithChi2 associator (of type Particle2MCWeightedAsct)
sc = toolSvc()->retrieveTool( “Particle2MCWeightedAsct”, “WithChi2Asct”, m_pAsctWithChi2);
// This is another type of Particle2MC tool, differentiated by jobOptions
sc = toolSvc()->retrieveTool(“Particle2MCWeightedAsct”, "LinkAsct", m_pAsctLinks);
```

  - **In the JobOptions file, declare different locations and algorithms**

```
// first associator using chi2 as weight
Toolsvc.WithChi2Asct.Location      = "Phys/Relations/Particle2MCWithChi2";
ToolSvc.WithChi2Asct.AlgorithmType = "Particle2MCWithChi2";
ToolSvc.WithChi2Asct.AlgorithmName = "Particle2MCWithChi2";
// alternate associator using stored links
Toolsvc.LinkAsct.Location      = "Phys/Relations/Particle2MCLinks";
ToolSvc.LinkAsct.AlgorithmType = "Particle2MCLinks";
ToolSvc.LinkAsct.AlgorithmName = "Particle2MCLinks";
```

Further features

- **Location of the table**
  - **Advise:**
    - **Use as for containers a static const definition in the .h file**

```
static const std::string& Particle2MCAsctLocation = "Phys/Relations/Particle2MC";
```

    - **Define the Associator property (in the Associator constructor)**

```
setProperty( location, Particle2MCAsctLocation );
```

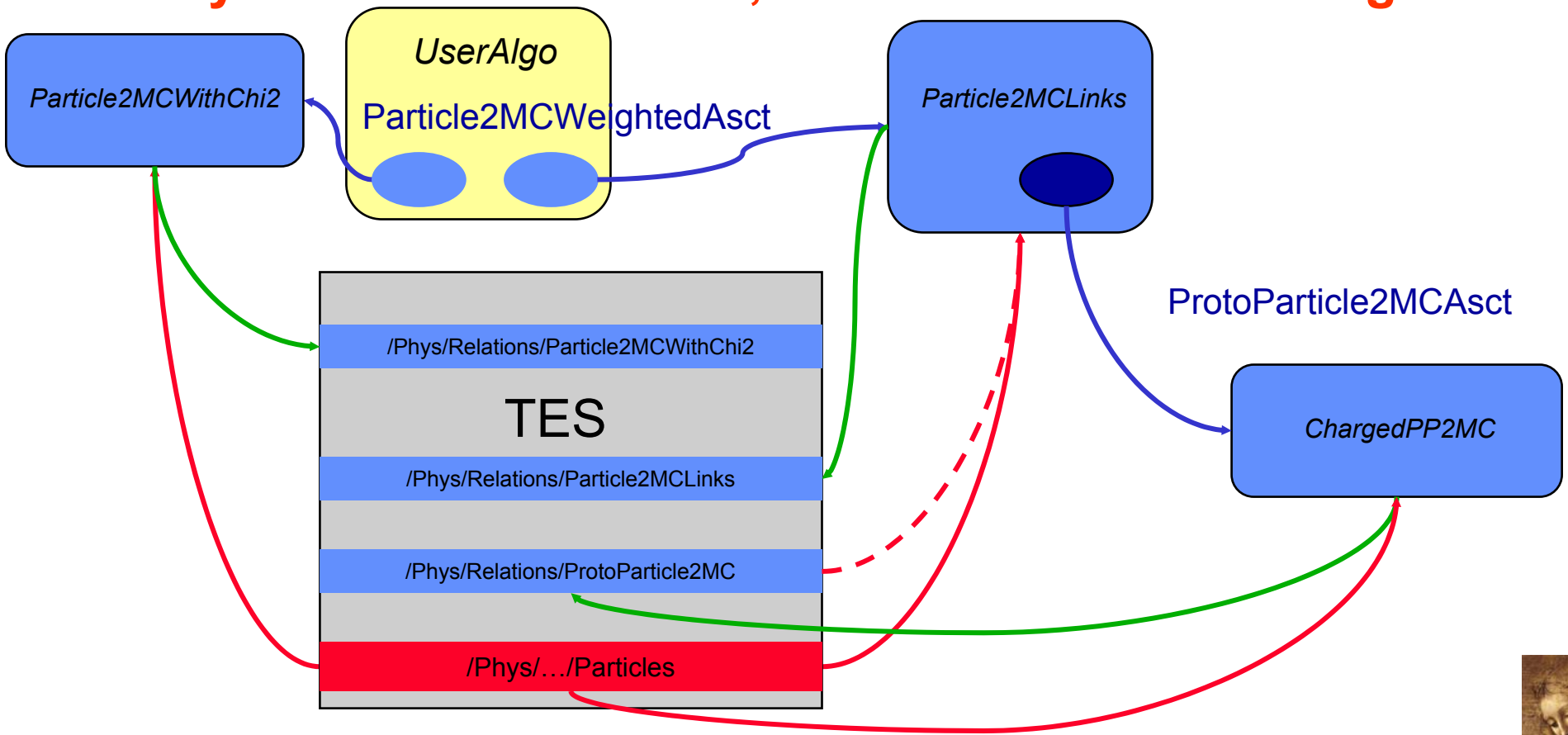
    - **Use for registering in the TES *OutputTable* as property of the algorithm**

```
declareProperty( "OutputTable", m_outputTable = Particle2MCAsctLocation );
. . .
 StatusCode sc = eventSvc()->registerObject( outputTable(), table);
```

    - **The associator automatically sets the *OutputTable* property of the algorithm**

DaVinci Associators

- **Package Phys/DaVinciAssociators**
- **Many associators defined, with various creation algorithms**



DaVinciAssociators 2

- **Algorithm properties**
  - **Particle2MCWithChi2.opts**

```
Particle2MCWithChi2.InputData ={"Phys/Production/Particles”};
Particle2MCWithChi2.OutputTable = "Phys/Relations/Particle2MCWithChi2";
Particle2MCWithChi2.FillHistos = true;
```

  - **Particle2MCChi2.opts**

```
Particle2MCChi2.InputData  = {"Phys/Production/Particles”};
Particle2MCChi2.OutputTable = "Phys/Relations/Particle2MC";
Particle2MCChi2.Chi2Cut = 100.;
```

  - **Particle2MCLinks.opts**

```
Particle2MCLinks.InputData  = {"Phys/Production/Particles”};
Particle2MCLinks.OutputTable = "Phys/Relations/Particle2MCLinks";
```

Summary

- Associators and relations tables are very powerful means for linking indirectly objects
  - No explicit link in the data model
  - Relations are external and can be serialized or re-created
  - Exemples:
    - Particle to MCParticle
    - Clusters to MCParticle
    - Vertex to Particles (not implemented that way, but could be)
- A generic tool exists, could be used as such
- For physics studies, we suggest to follow guidelines described in this presentation
- Other users could follow them as well…