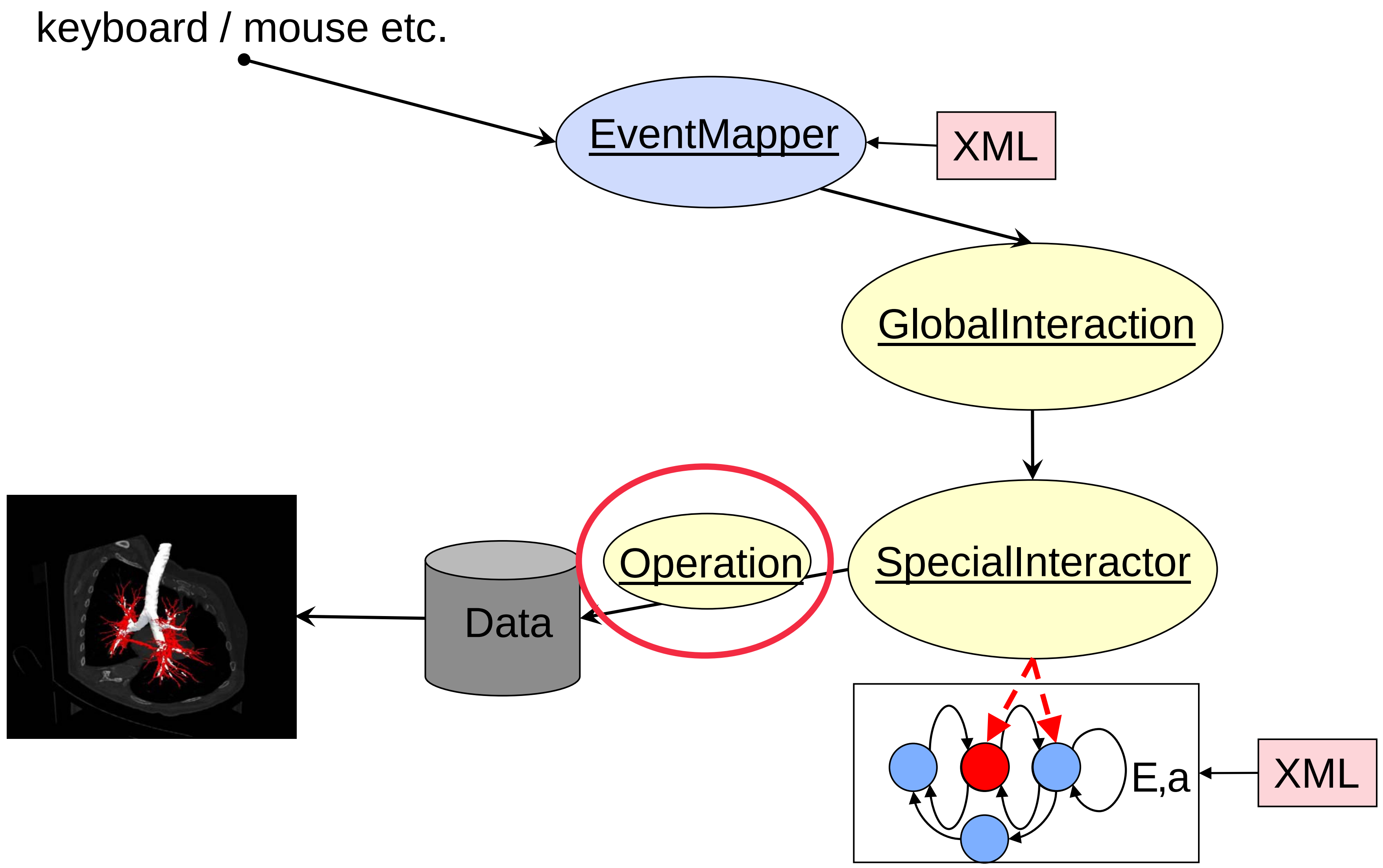
MITK

Undo / Redo

Ingmar Wegner

Reminder: Interaction Sequence



Operations

Class mitk::Operation is a container for all information important for a change of data. Example:

…within MySpecialInteractor::ExecuteAction(…)

```
mitk::Point3D itkPoint = theEvent->GetWorldPosition();
PointOperation* doOp = new mitk::PointOperation(
      OpINSERT, timeInMS, itkPoint, pointSet->Size());
```

What to do? When? Change to? Where?

```
pointSet->ExecuteOperation(doOp);
```

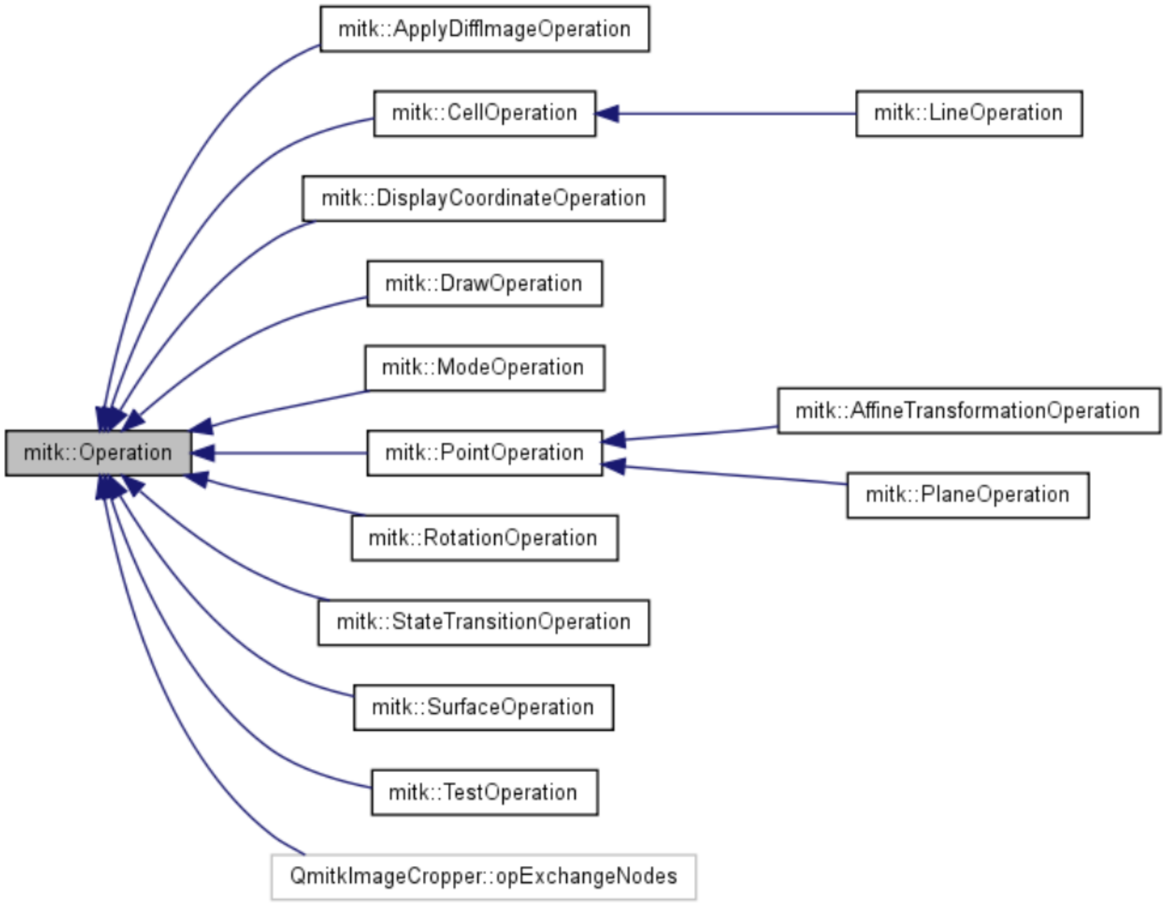
Public Member Functions | Protected Attributes

mitk::Operation Class Reference
[Undo Classes]

Base class of all Operation-classes. More...

`#include <mitkOperation.h>`

Inheritance diagram for mitk::Operation:



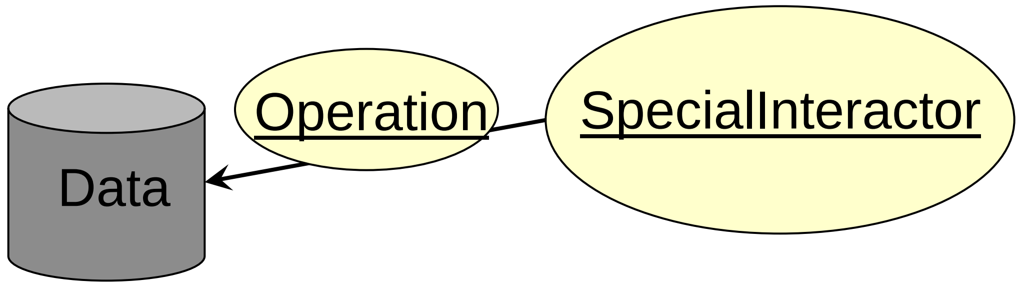
[legend]

List of all members.

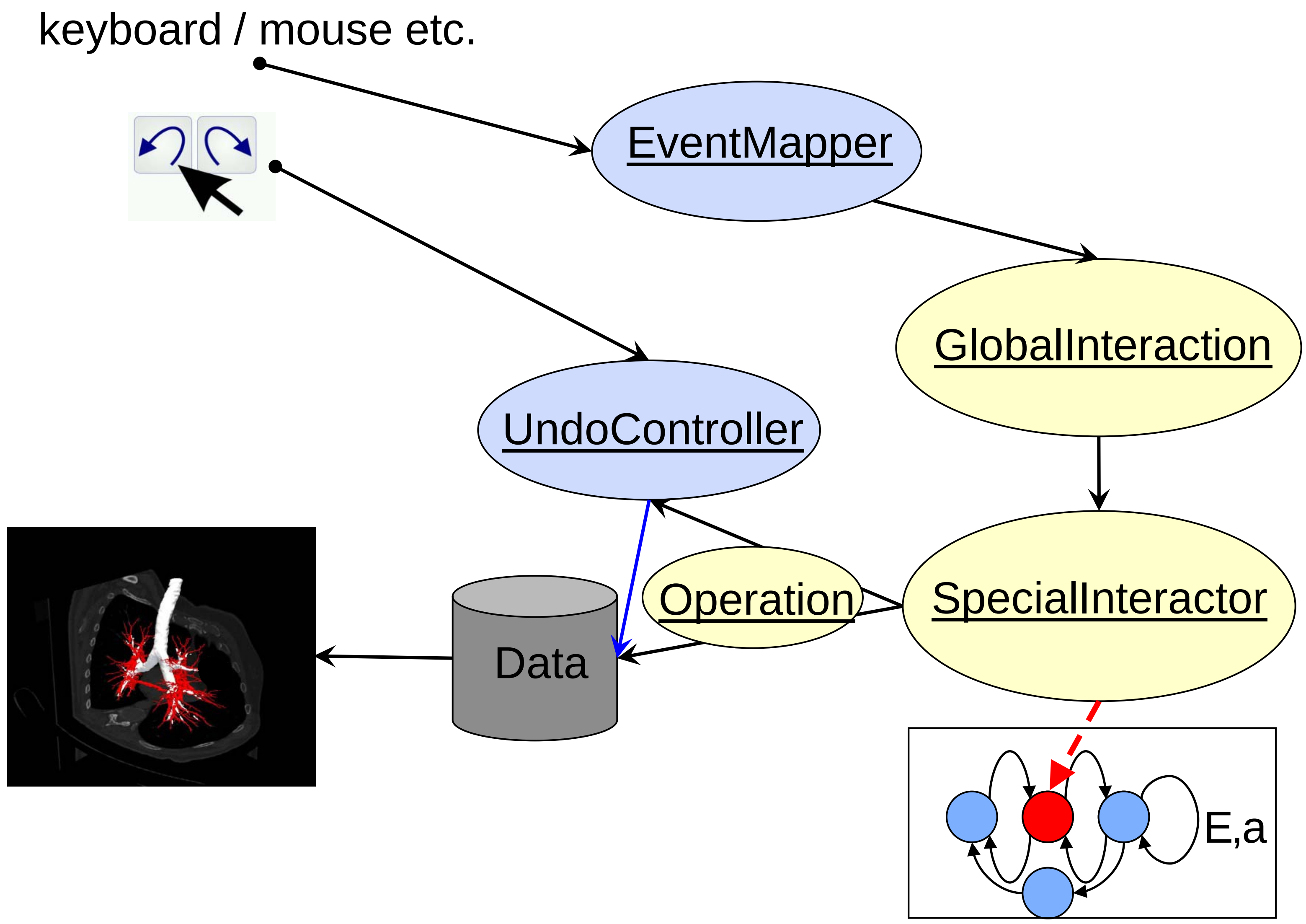
Why Operations?

Undo / Redo functionality!

Represents an extra layer between
interaction classes taking care of changing data
and data.



Undo Sequence



Undo operations

```
//within MySpecialInteractor::ExecuteAction(…)

mitk::Point3D itkPoint = theEvent->GetWorldPosition();
PointOperation* doOp = new mitk::PointOperation(
      OpINSERT, timeInMS, itkPoint, pointSet->Size());

pointSet->ExecuteOperation(doOp);

if (m_UndoEnabled) //protected member of mitk::StateMachine
{
   PointOperation *undoOp = new mitk::PointOperation(
         OpREMOVE, timeInMS, itkPoint, pointSet->Size());
   OperationEvent *operationEvent =
         new OperationEvent(pointSet, doOp, undoOp, "Add point");
   m_UndoController->SetOperationEvent(operationEvent);
}
else
   delete doOp;

//OperationEvent and Operations are kept within and deleted in UndoModel
```

Feature Requests in the very beginning 11/2002

Undo:

- Offer flexible undo / redo functionality
  - ➢Can be enabled and disabled. Thorough programming includes undo, rapid prototyping doesn't care about undo.
- Save memory resources
  - ➢Only store parameters how operations can be undone

```
PointOperation *undoOp = new mitk::PointOperation(
    OpREMOVE, timeInMS, itkPoint, pointSet->Size());
```

  - ➢If impossible (e.g. image filters), store backups if necessary

DO‘s and DONT‘s

DO:

- provide Undo functionality!
- reuse constants in mitkInteractionConst.h
- divide your information into small pieces and send them via operations to data:
  OpADD, OpSELECT rather than OpADDSELECTED
- if image filter operation is invertible,
  then store invert parameters only

```
MyFilterOperation *undoOp = new mitk::MyFilterOperation(
        OpDEFAULT, timeInMS, invertParameters);
```

- if not, save backup of image on disk

DON’T

- store big data in operations; SmartPointers onto images in operations will hold memory until UndoStack is cleared