Interaction and Undo

Ingmar Wegner

What should be known by now

- DataNodes are stored within DataStorage and are parsed during rendering. One datum is connected to one DataNode

- Visualization is done by VTK
- Image processing is done by ITK

- MITK is GUI independent, MITK ExtApp uses QT

- See hyperlinks to doxygen documentation for further reading

Contents

- Interaction
  - What's this?
  - Feature requests for MITK
  - Solution (state machines)
  - Example interaction sequence
- Undo (/ Redo)
  - Feature requests
  - Example undo sequence

What is interaction?

User can modify data during runtime using input devices.

Feature requests in the beginning 11/2002

Interaction:

- Representation of complex workflows
- Possibility to quickly modify the interaction
- Reuse of interaction
- Independent from GUI toolkit
- User- and developer-friendly
- Allows flexibility
- Not dependent on visualization (2D / 3D)

Undo:

- Offer flexible undo / redo functionality
- Save memory resources

Answers

Interaction:

- Representation of complex workflows
  - So use state machines (Mealy / More)
- Possibility to quickly modify the interaction
  - Without recompile? Then use a generic way of loading interaction patterns during startup.
- Reuse of interaction
  - Then one interactor per data object and the developer defines what it does. Several equivalent data objects use the same interaction pattern.

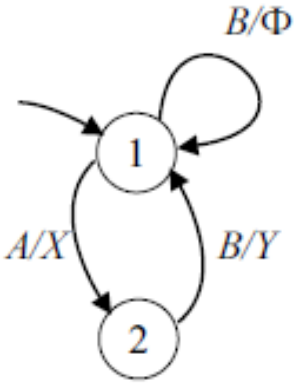
Answers

- Independent from GUI toolkit
  - ➢Create a layer in between
- User- and developer-friendly
  - ➢**Swiss army knife?* / *Eierlegende Woll-Milch-Sau* …*
    focus on user-friendly
- Allows flexibility
  - ➢Adapting patterns without recompile,
    data objects still accessible,
    equal interactors can have different patterns …
- Not dependent on visualization (2D / 3D)
  - ➢Change the data independent from visualization

State machine (Mealy)

A Mealy state machine consists of:

- States
- Events
- Transitions
- Actions



Traditional

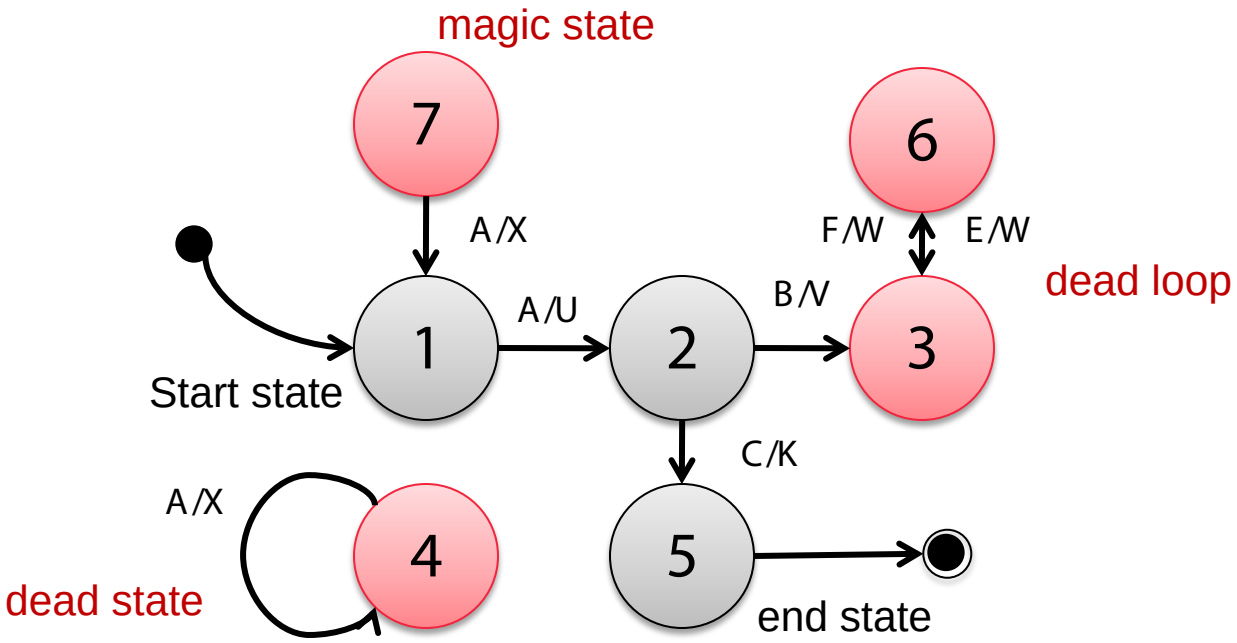
Mealy: State is passive, transition causes change of data.

(More: State is the active part)

State machine

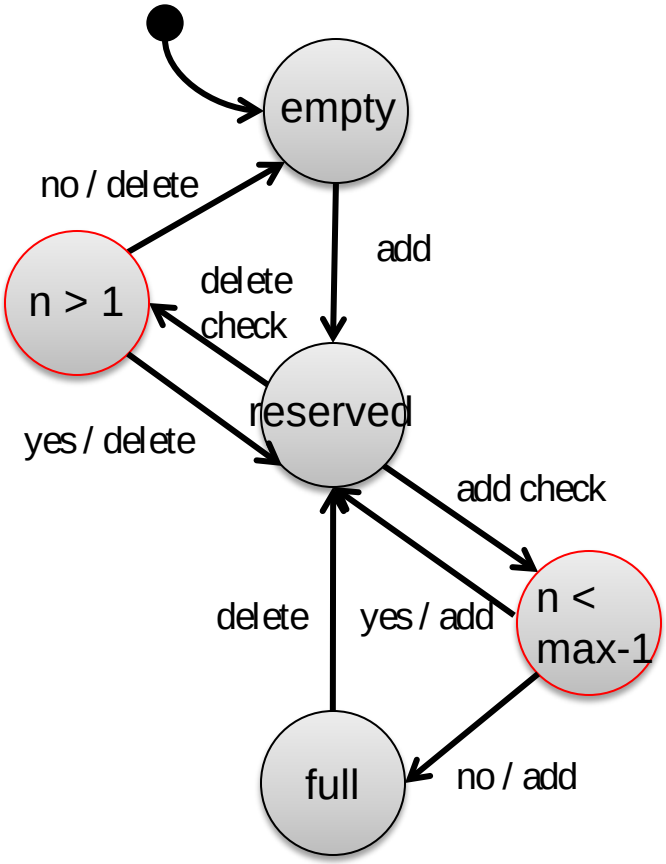
Has to have:

- One start state
- No dead state (not reachable)
- No magic state (transition leaves the state but can not be reached by others, not a start state)
- No dead loop
- Deterministic transitions (no equal transitions leading to different states)



State machine

Guard state: temporary state to check for a condition

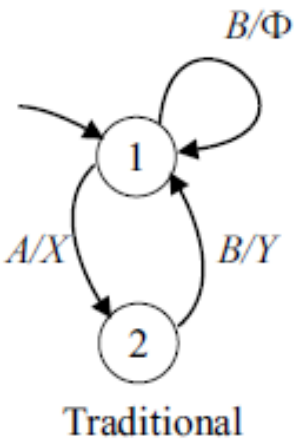


Classes in MITK to implement a state machine

mitkState

mitkTransition

mitkAction



Traditional

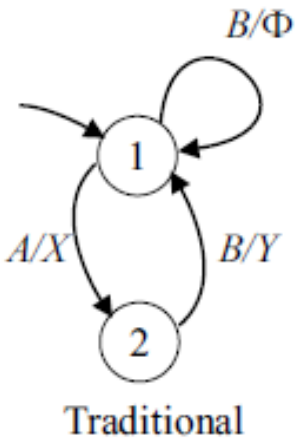
According to a description several objects are instantiated and connected to each other during startup. Objects of type **mitkTransition** connect two objects of type **mitkState** in one direction (e.g. from state 1 to state 2). They contain the information on which event a state change may be done (EventID). They contain several objects of class **mitkAction** that specify the actions that are done after a state change.

All objects together represent the so called **state machine pattern**

Generically loading of interaction patterns

So, a state machine pattern defines the workflow of a special interaction procedure (e.g. interacting with a set of points).

All available patterns are loaded by a **StateMachineFactory** during startup (StateMachine.xml)



Traditional

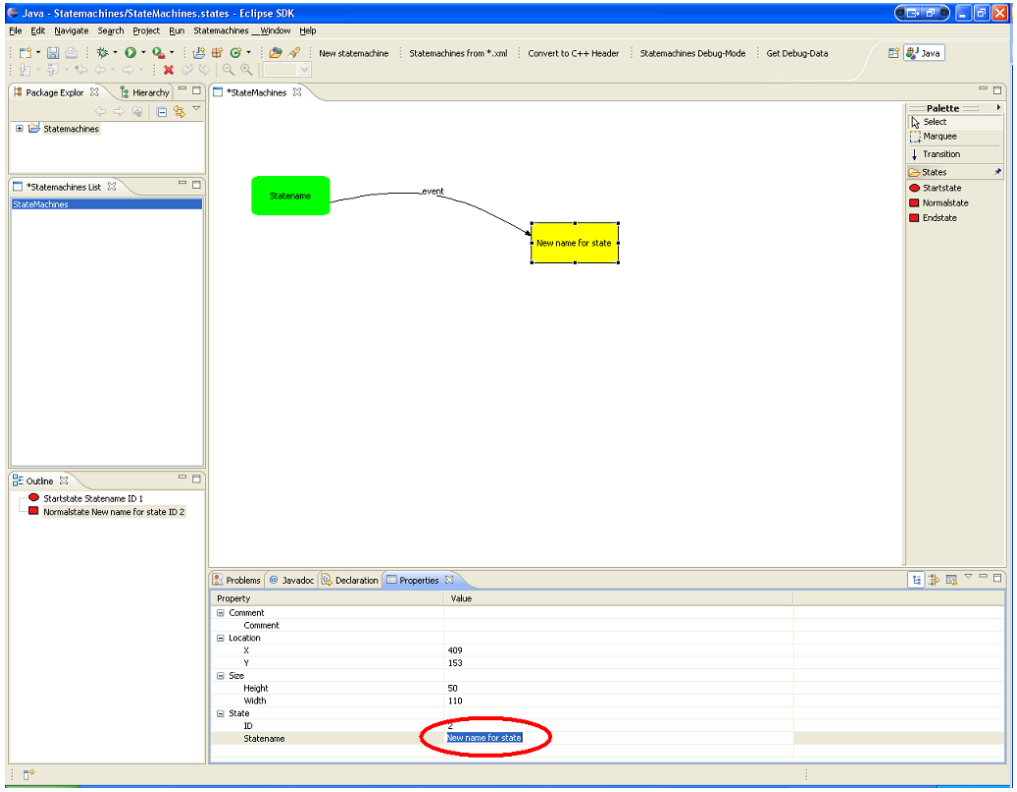
```
…
<stateMachine NAME=“dumm example of  the MITK state machine pattern xml syntax">
   <state NAME=“first" ID="1" START_STATE="TRUE">
      <transition NAME=“to2" NEXT_STATE_ID="2" EVENT_ID=“A">
         <action ID=“X" />
      </transition>
      <transition NAME=“to1" NEXT_STATE_ID=“1" EVENT_ID=“B">
         <action ID=“Φ" />
      </transition>
   </state>
   <state NAME=“second" ID=“2" >
      <transition NAME=“to1" NEXT_STATE_ID=“1" EVENT_ID=“B">
         <action ID=“Y" />
      </transition>
   </state>
</stateMachine>
…
```

Note: Also event IDs and action IDs are of type int in MITK

State Machine Editor

See related pages for section State Machine Editor:

graphical tool (eclipse plug-in) to create, modify and view state machine patterns



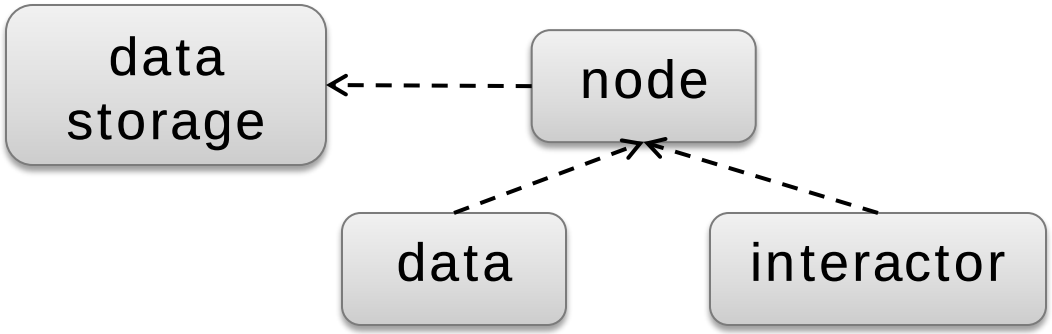
State machine logic

Class mitk::StateMachine implements all methods for the use of a state machine.

```
class StateMachine : public itk::Object, public mitk::OperationActor
{
...
public:
  virtual bool HandleEvent (StateEvent const *stateEvent);
...
protected:
  virtual bool ExecuteAction (Action *action, StateEvent const *stateEvent);
...
private:
  std::vector<State::Pointer> m_CurrentStateVector;
...
}
```

One interactor takes care of one data

mitk::Interactor is derived from mitk::StateMachine and adds dependency to one data.

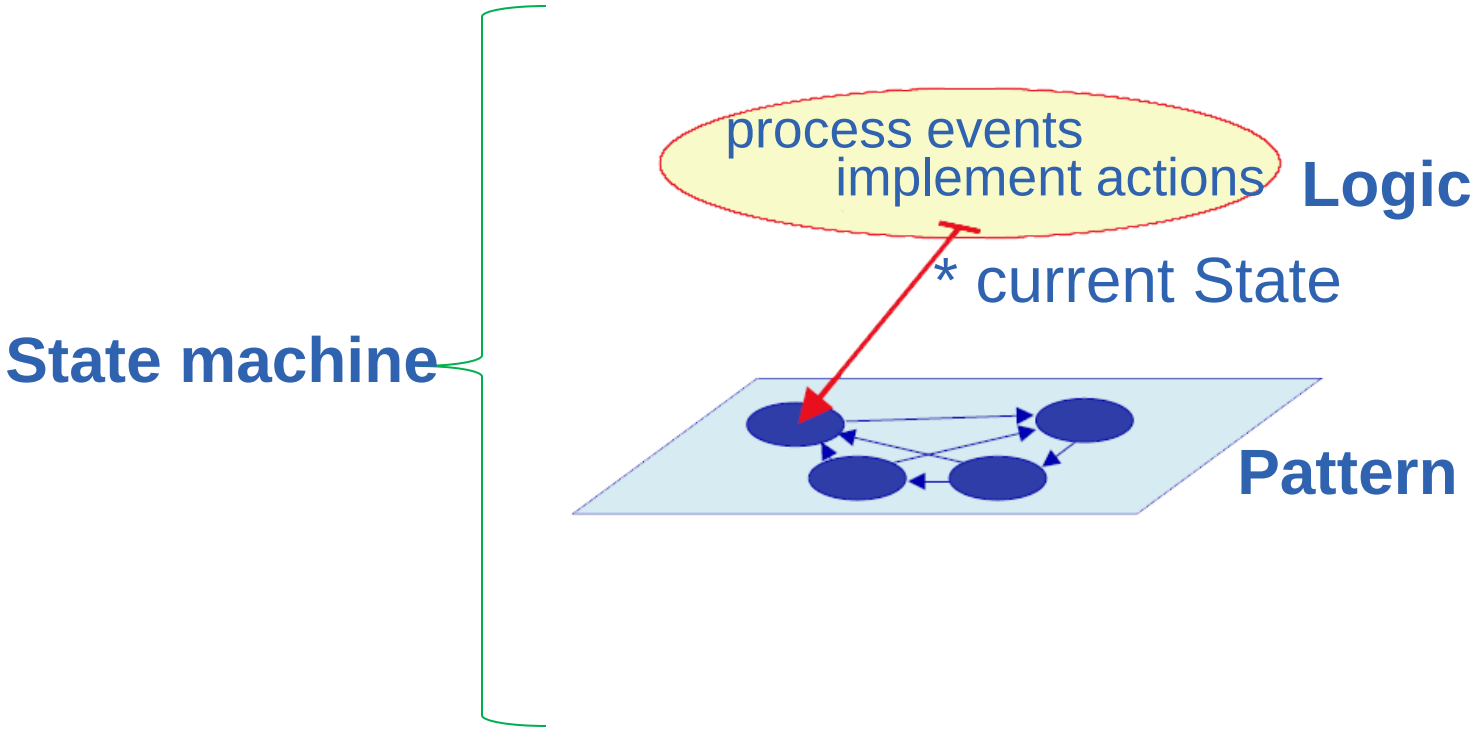


Derived classes

From mitk::Interactor derived classes add the implementation of actions that will lead to a change of data. Example:

```
class LightSwitch : public StateMachine
{
public:
   mitkClassMacro(LightSwitch, StateMachine);
   LightSwitch(const char*);
   bool DoSwitchOn(Action*, const StateEvent*);
   bool DoSwitchOff(Action*, const StateEvent*);
}
LightSwitch::LightSwitch(const char* type) :StateMachine(type)
{
   CONNECT_ACTION( AcSWITCHON, DoSwitchOn );
   CONNECT_ACTION( AcSWITCHOFF, DoSwitchOff );
}
bool LightSwitch::DoSwitchOn(Action*, const StateEvent*)
{
   MITK_INFO<< "Enlightenment \n";
}
bool LightSwitch::DoSwitchOff(Action*, const StateEvent*)
{
    MITK_INFO << "Confusion \n";
}
```

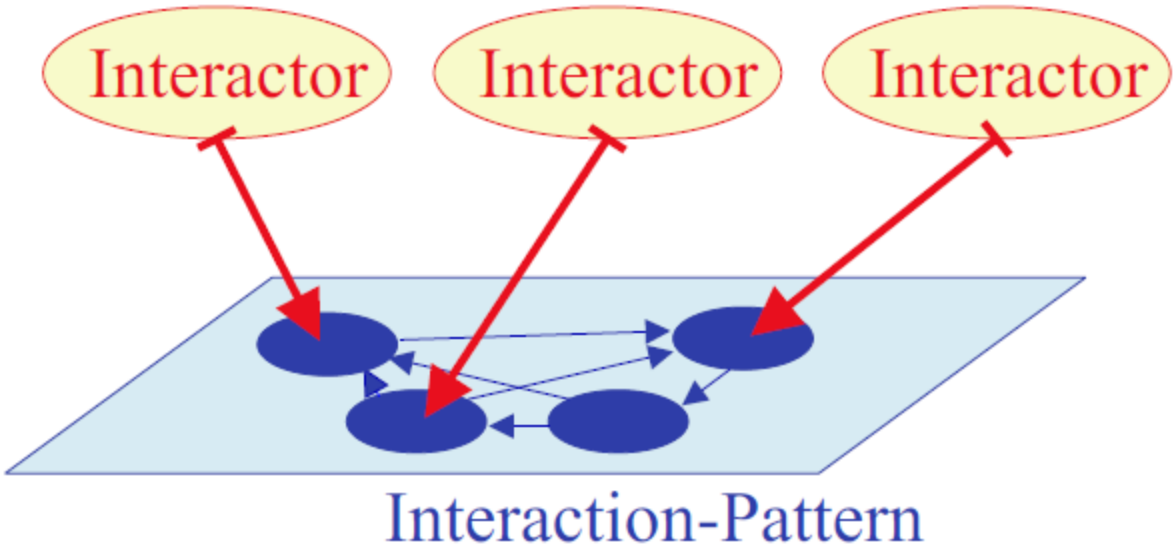
All together it‘s a state machine!



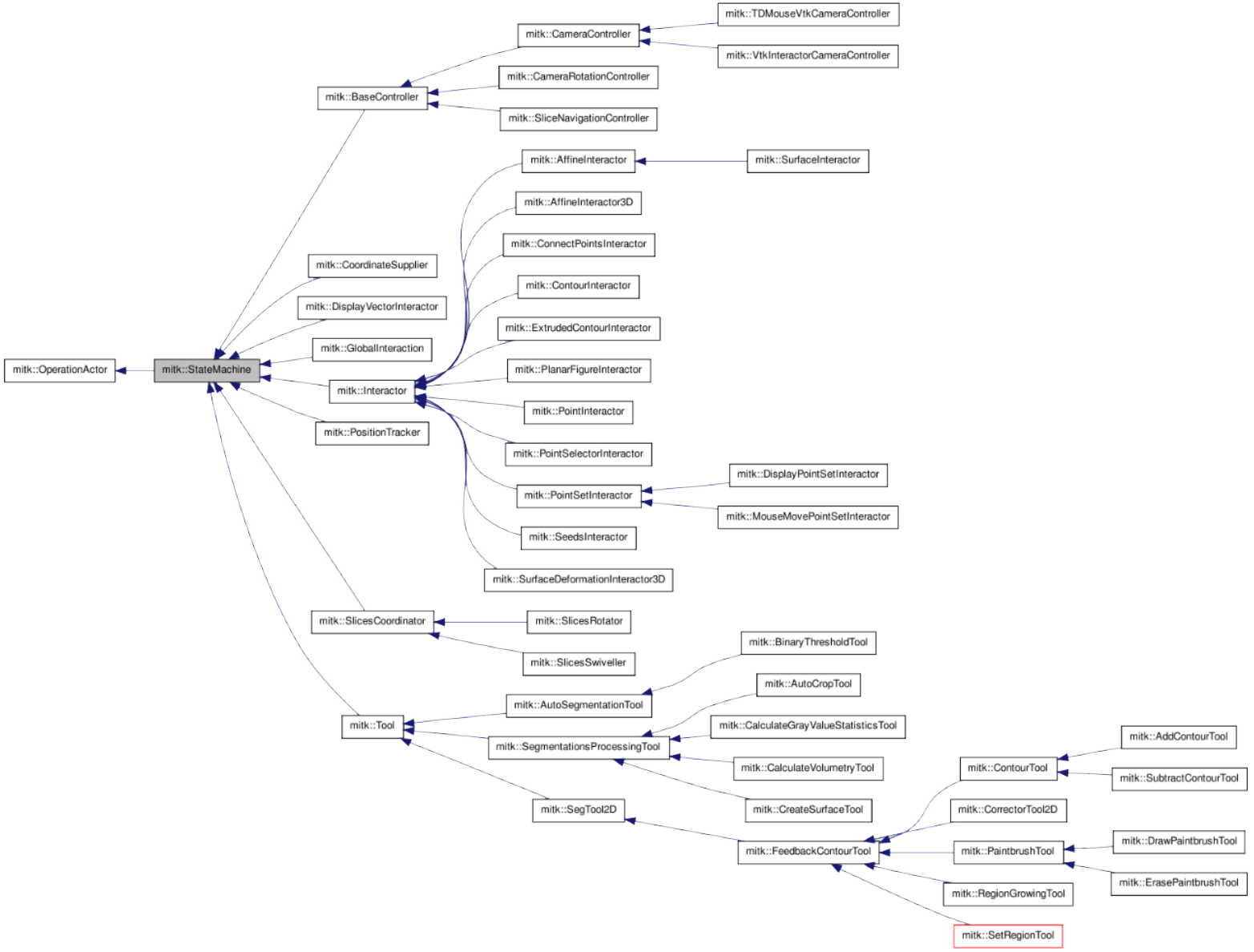

```
mitk::MyStateMachine::Pointer myStateMachine =
        mitk::MyStateMachine::New(“myPattern“, nodeOfData);
```

Reuse of patterns

Because each object of type StateMachine pointers to one state of the specified state machine pattern, many objects can reuse one pattern.

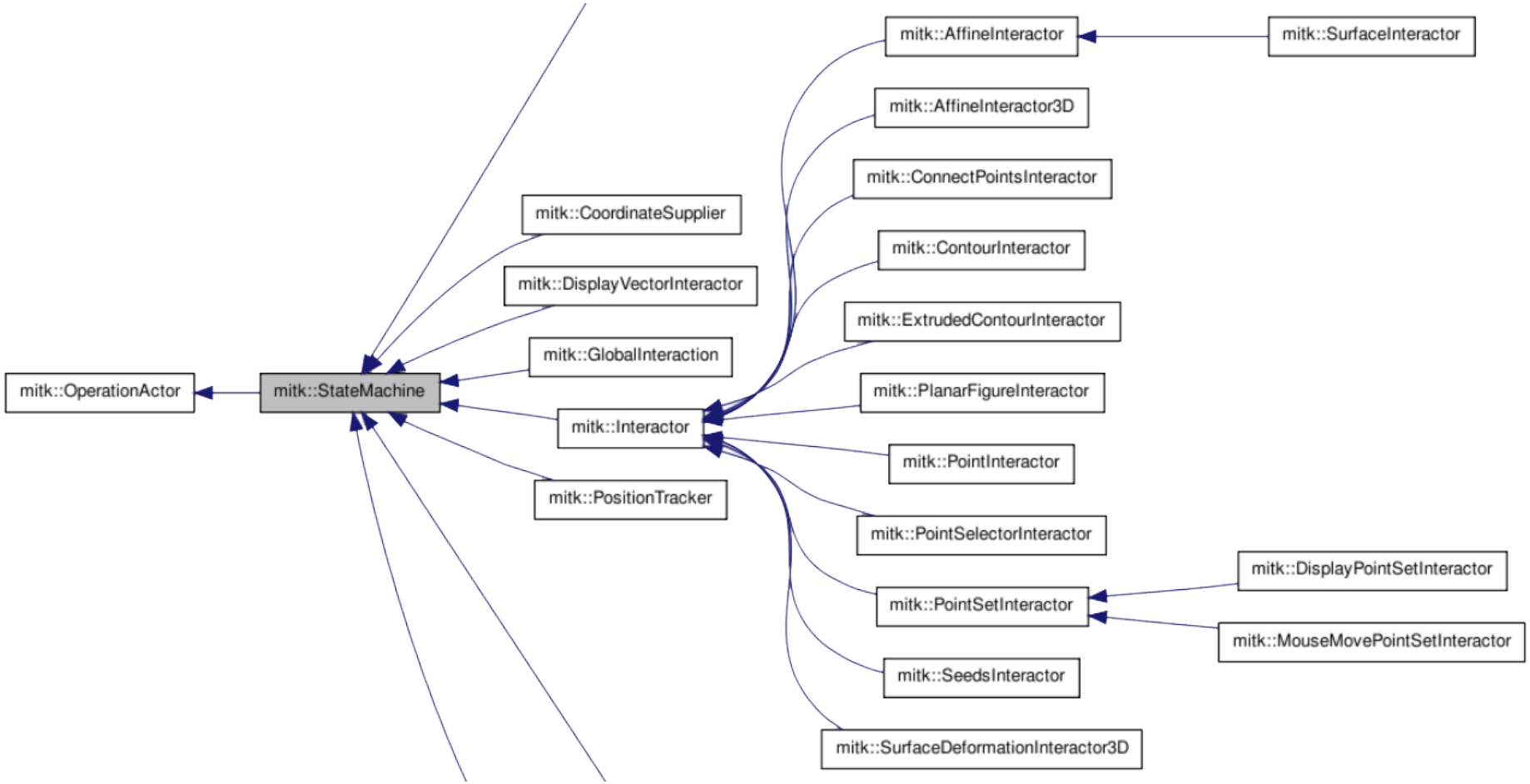


Doxygen mitk::StateMachine



by 9/17/2010

Detail from Doxygen mitk::StateMachine

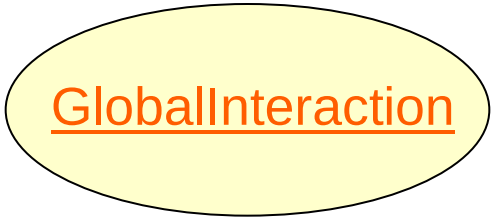


by 9/17/2010

So many interactors!

How to administrate several interactors?

- Let one be the leader!



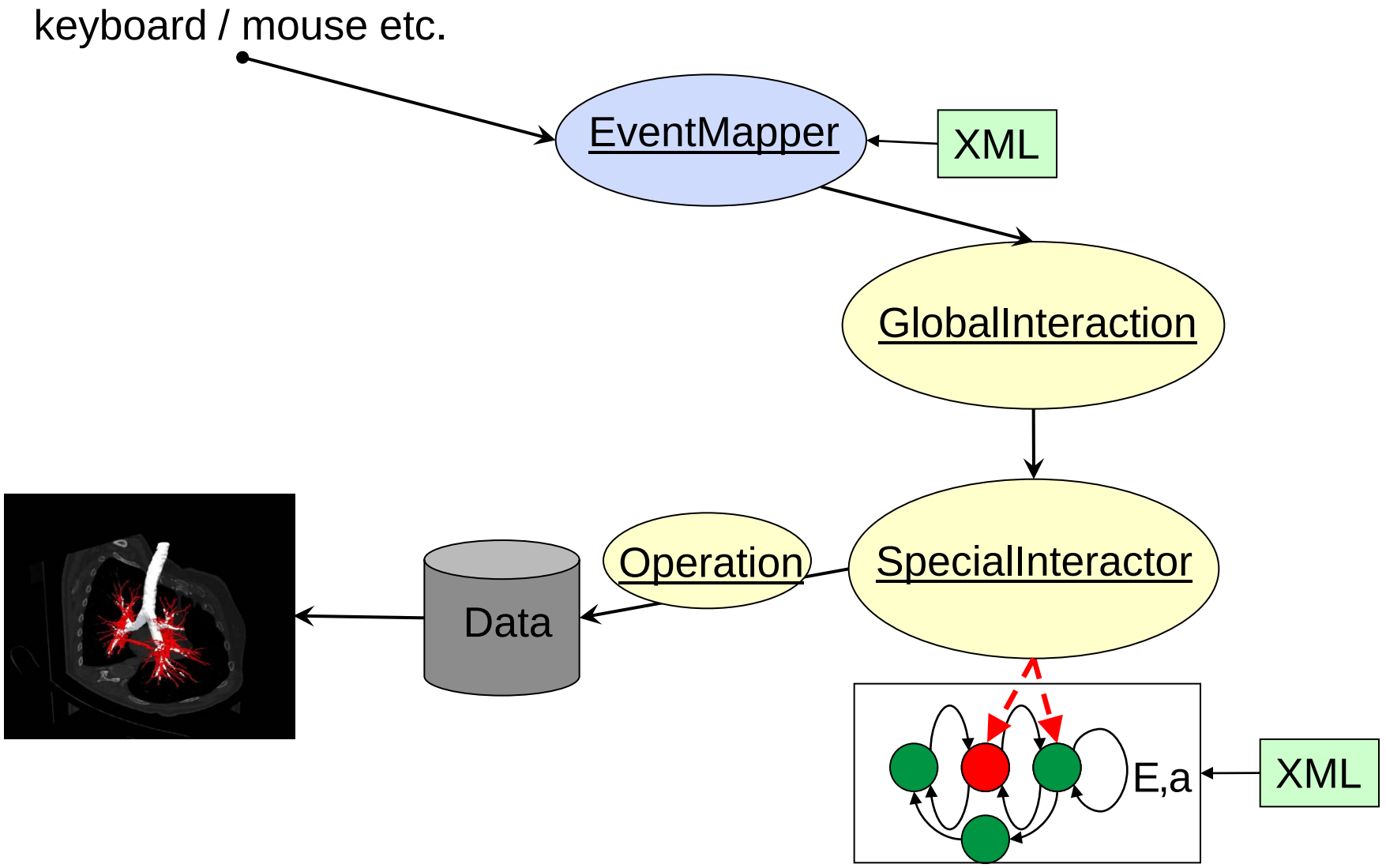
GlobalInteraction

- Administrates several state machines:
  - Interactors: Set onto one DataNode and thus associated to one data (e.g. PointSetInteractor to PointSet)
  - *„Listeners“*: StateMachines that modify visualization, not data (e.g. CorrdinateSupplier for StatusBar)

```
class GlobalInteraction : public StateMachine
{
   public:   ...
      void AddInteractor(Interactor* interactor);
      bool RemoveInteractor(Interactor* interactor);
      void AddListener(StateMachine* listener);
      bool RemoveListener(StateMachine* listener);
...}
```

An event is sent to all Listeners and only to the one Interacor, that float Interactor::CanHandleEvent(...) the best.

Interaction sequence

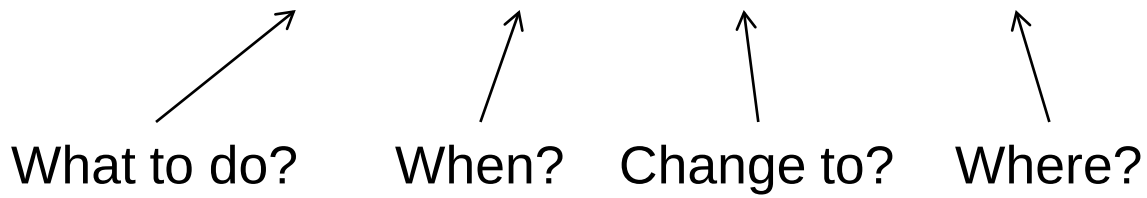


Operations

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());
```



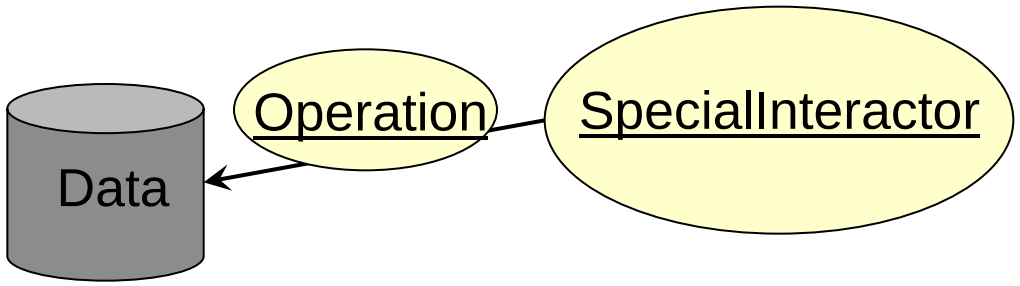

```
pointSet->ExecuteOperation(doOp);
```

Why Operations?

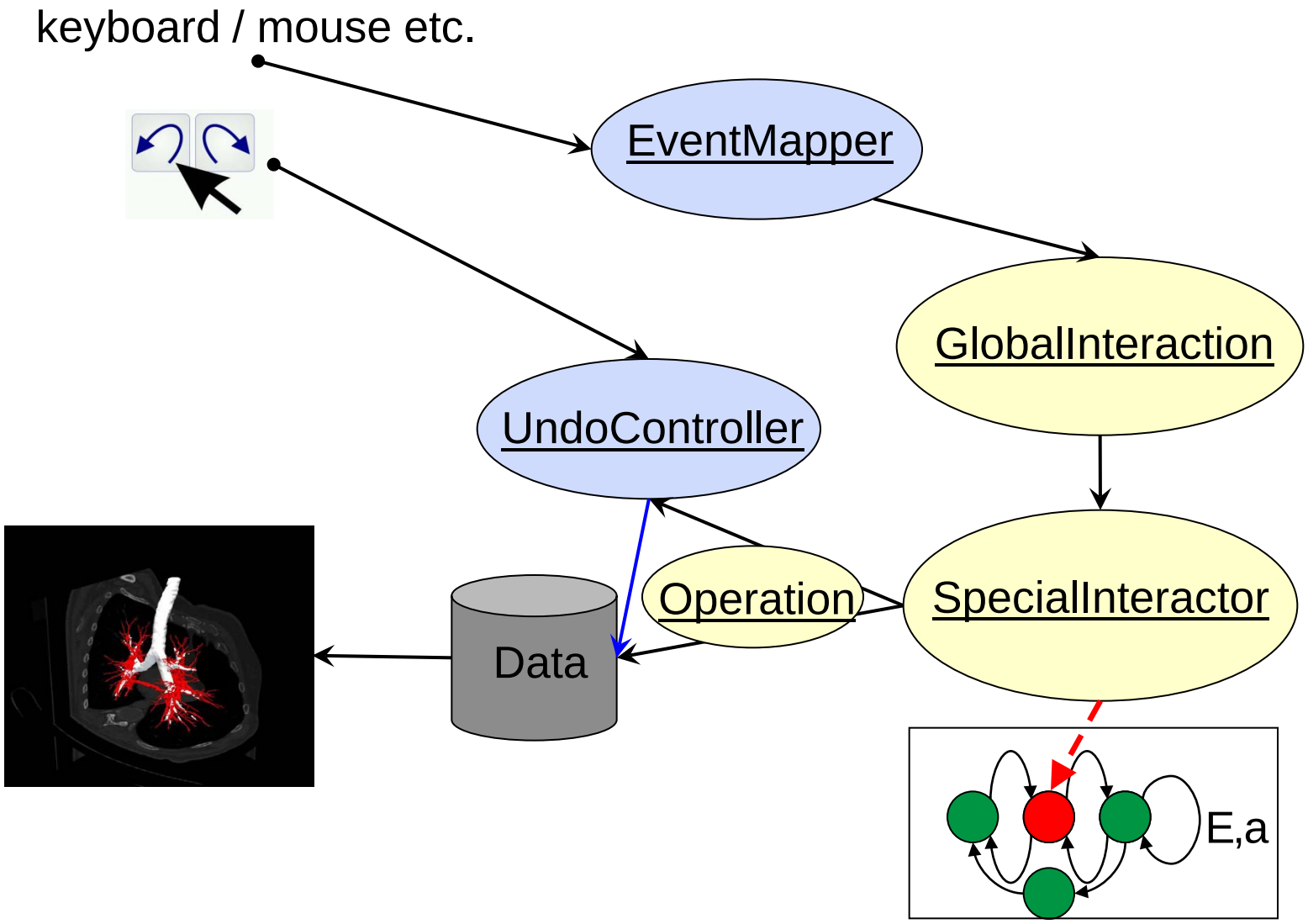
Undo / Redo functionality!

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());

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

pointSet->ExecuteOperation(doOp);
```

Feature Requests in the 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

Further reading:

- http://www.mitk.org/wiki/Interaction_concept
- Doxygen documentation on mitkGlobalInteraction, mitkStateMachine
- Tutorial Step10 shows what to modify to add a new interactor