ASCLEPIOS - INRIA Sophia Antipolis

Herve Delingette, Barbara Andre

SOFA - Topology Module

Presentation
of the current implementation

Meeting in Lille

Monday, 18. February 2008

Mesh Topology is useful for :

- **Mesh Visualization**
- **Collision Detection**
- **Mechanical Modeling (deformation)**
- **Haptic Rendering**
- **Description of Scalar Fields (temperature, electric potential, …) or Vectorial Fields (speed, fiber orientation, …)**

} **Computational Mesh**

Geometry Description

Set of DOFs

(position of each vertex)

Topology Description

How the DOFs are connected ?

(edges, triangles, tetrahedron, …)

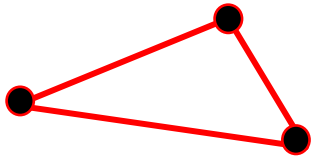
Outline

1. **Hierarchical Architecture**
2. **Topological Events**
3. **Dynamic Data Structures**
4. **Topological Mappings**

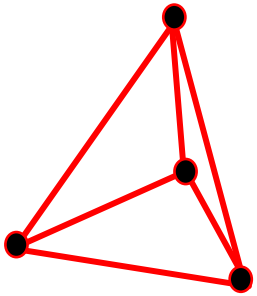
1. **Hierarchical Architecture**
2. Topological Events
3. Dynamic Data Structures
4. Topological Mappings

Topology Elements

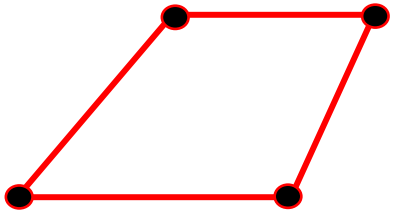
Edge



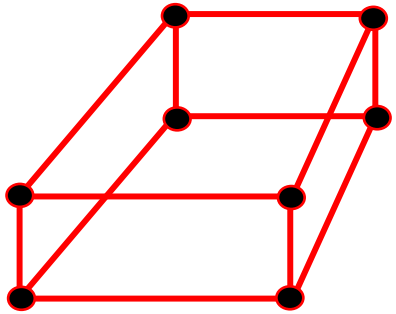
Triangle



Tetrahedron

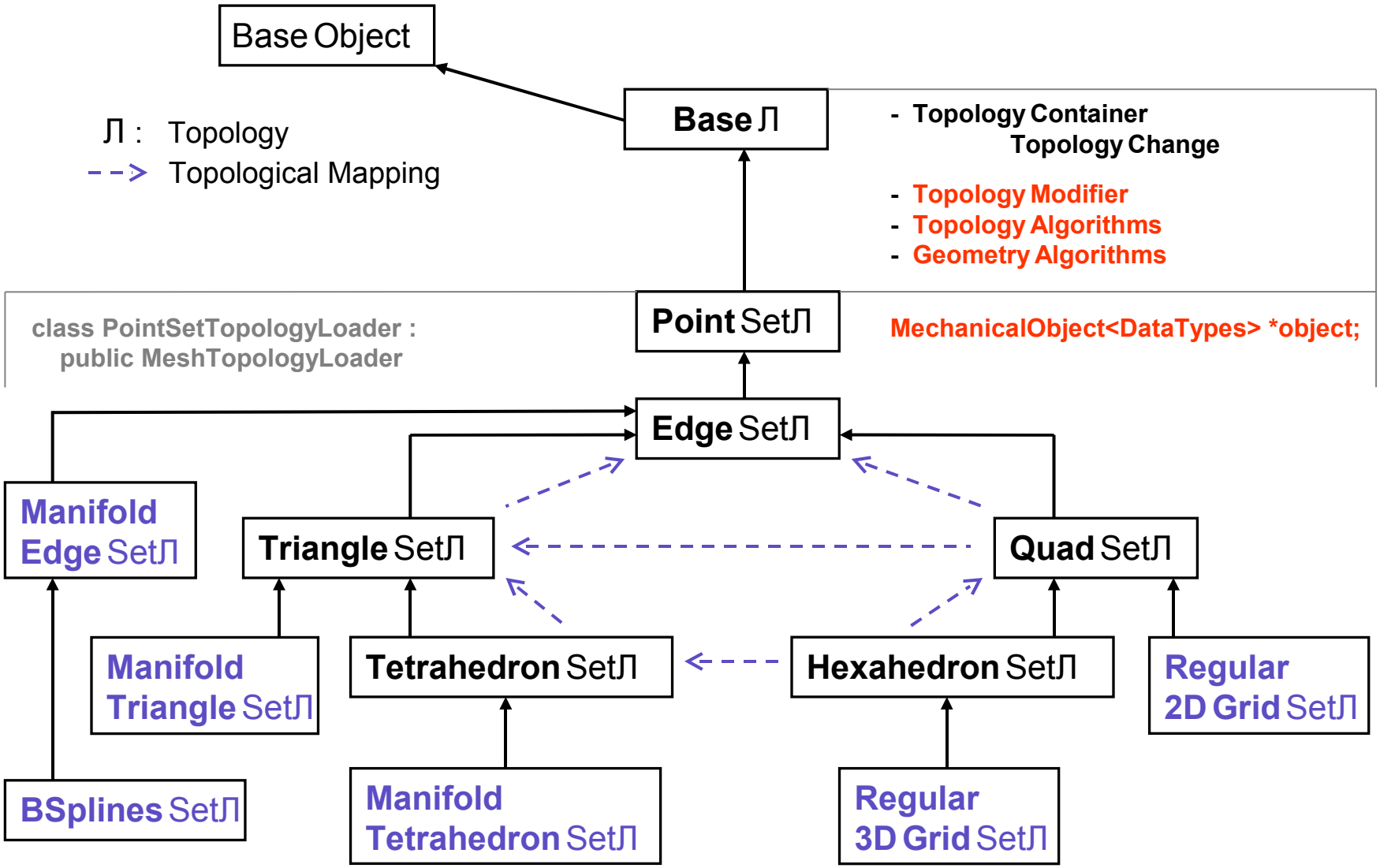


Quad



Hexahedron

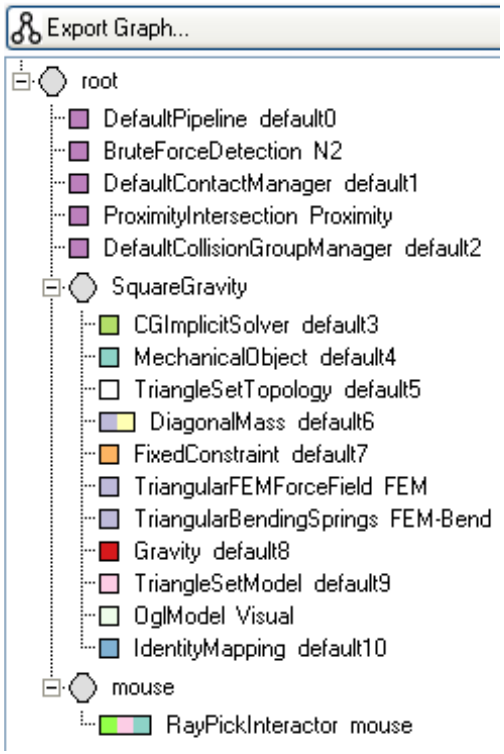
1. Hierarchical Architecture

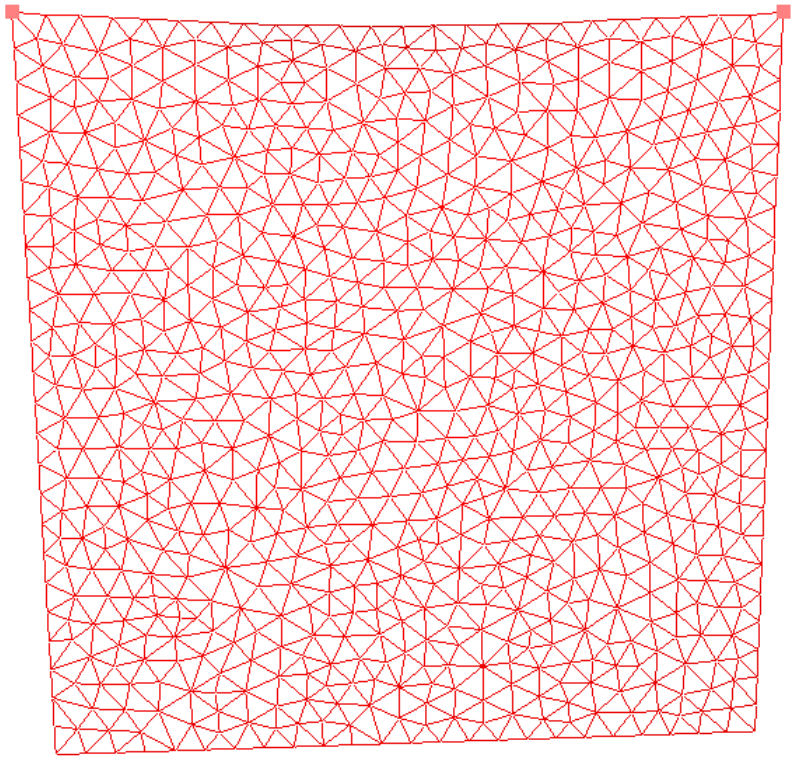


```
Class BaseTopology<DataTypes> {

        // A container for info to be stored and methods to access adjacency :
        // - Adjacency Information is only computed when needed
        // - Non template class
        // - Store TopologicalChange list
        TopologyContainer *container ;

        // A modifier for low-level methods to change topology :
        // - Cannot be accessed from user
        // - Modifier also changes the DOFs in the Mechanical Object
        // - Low level methods to add or to remove an item
        TopologyModifiers<DataTypes> *modifier ;

        // TopologyAlgorithms for high-level methods to change topology (user access) :
        // - Accessed from the user
        // - High level algorithms to refine, cut mesh
        TopologyAlgorithms<DataTypes> *topologyAlgorithms ;

        // Geometry Algorithms methods to get geometry information :
        // - Compute geometric information (normal, curvature, area, length)
        GeometryAlgorithms<DataTypes> *geometryAlgorithms ;
};
```

Use Case 1

Simulate a hanging soft tissue as a triangular surface





Example of TriangleSetTopology :

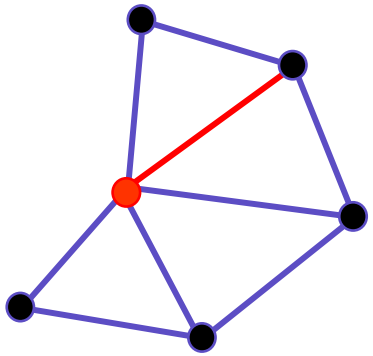
Container

getTriangleArray ( )
getTriangleEdgeArray ( )

getTriangleVertexShell (i)
getTriangleEdgeShell (i)

getEdgeArray ( )

getEdgeVertexShell (i)



Modifier

load ( )

addTrianglesWarning (I)
addTrianglesProcess (I)

removeTrianglesWarning (I)
removeTrianglesProcess (I)

Geometry

computeTriangleArea (i)
computeTriangleNormal (i)

computeSegmentTriangleIntersection (…)
computeIntersectedPointsList (…)

Algorithms

removeTriangles (I)

InciseAlongPointsList (I)
InciseAlongEdge (i)

1. Hierarchical Architecture
2. Topological Events
3. Dynamic Data Structures
4. Topological Mappings

Order to respect when adding or removing an item (low-level methods)

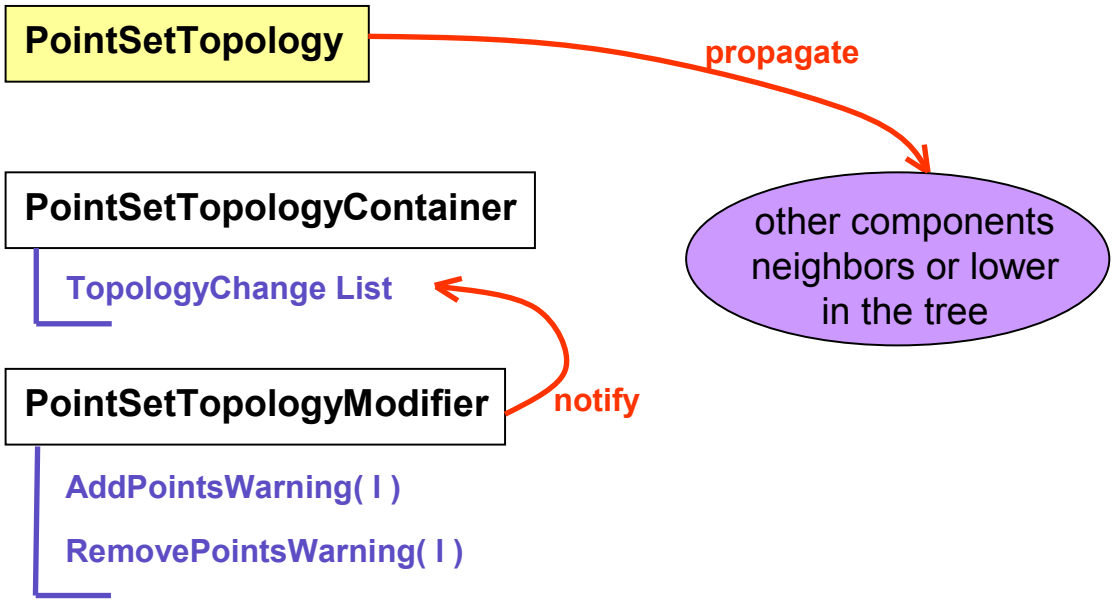
ADD a sequence of items	REMOVE a sequence of items
1. ADD	1. NOTIFY
2. NOTIFY	2. PROPAGATE
3. PROPAGATE	3. REMOVE

TOPOLOGICAL CHANGE EVENT : add or delete a list of items

NOTIFY : add the current topological change event in the List of Topological changes

PROPAGATE : traverse the simulation tree with a TopologyChangeVisitor to send the events of the current TopologicalChanges List to every components beneath (Force Fields, Constraints, Mass, …)

```
enum TopologyChangeType {

                ENDING_EVENT,
POINTSINDICESSWAP,
POINTSADDED,
POINTSREMOVED,
POINTSRENUMBERING,
                …
}
```



Important : **All topological elements are stored in arrays**

Advantage

Contiguous storage in memory ⇒ **efficiency**

Drawback

Difficulty to update when topological changes occur :

array resizing + index swapping issues

1. Hierarchical Architecture
2. Topological Events
3. **Dynamic Data Structures**
4. Topological Mappings

Dynamic Data Structures

Example Component : TriangularBendingSprings

DATA description

INDICES container equipped with listener

```
template<class DataTypes>
class TriangularBendingSprings :
{

class EdgeInformation
{
        Mat3 DfDx;   // the spring stiffness matrix

        int  m1, m2;  // indices of the two mass vertices : extremities of the spring

        double  ks;   // spring stiffness
        double  kd;  // damping factor

        double  restlength;  // rest length of the spring

        bool is_activated;
        bool is_initialized;
}

EdgeData< EdgeInformation > edgeInfo;
```

Equivalence : edgeInfo [i] stores information about the edge indexed by i in the topology

Example Component : TriangularBendingSprings

To handle topological change in each specific component,programmer only fills callback functions to add or remove an item :

```
template<class DataTypes>
class TriangularBendingSprings :
{

static void TriangularBSEdgeCreationFunction (

                int edgeIndex,
                void* param,
                EdgeInformation &ei,
                const Edge& , const sofa::helper::vector< unsigned int > &, const sofa::helper::vector< double >&
);

static void TriangularBSTriangleCreationFunction(

                const sofa::helper::vector<unsigned int> &triangleAdded,
                void* param, vector<EdgeInformation> &edgeData
);

static void TriangularBSTriangleDestructionFunction (

                const sofa::helper::vector<unsigned int> &triangleAdded,
                void* param, vector<EdgeInformation> &edgeData
);
```

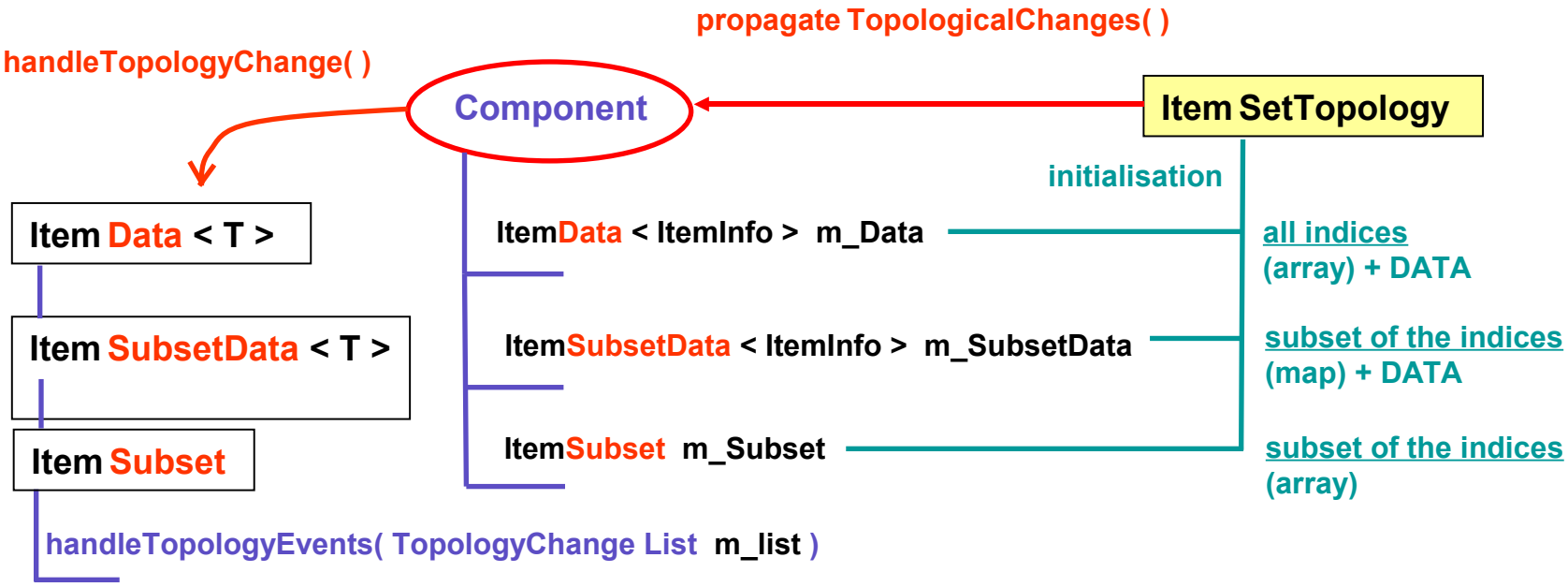
Example Component : TriangularBendingSprings

```
template<class DataTypes>
void TriangularBendingSprings<DataTypes>::init( )
{
        ...

        edgeInfo.setCreateFunction ( TriangularBSEdgeCreationFunction );
        edgeInfo.setCreateTriangleFunction ( TriangularBSTriangleCreationFunction );
        edgeInfo.setDestroyTriangleFunction ( TriangularBSTriangleDestructionFunction );

        edgeInfo.setCreateParameter ( (void *) this );
        edgeInfo.setDestroyParameter ( (void *) this );
}
```

```
template <class DataTypes>
void TriangularBendingSprings<DataTypes>::handleTopologyChange( )
{
        BaseTopology *topology = static_cast< BaseTopology *>( getContext( ) -> getMainTopology( ) );

        std::list<const TopologyChange *>::const_iterator itBegin = topology -> firstChange( );
        std::list<const TopologyChange *>::const_iterator itEnd = topology -> lastChange( );

        edgeInfo.handleTopologyEvents ( itBegin, itEnd );

        ...
}
```

Dynamic Data Structures are listening to topological change events :

Component : Force Field, Constraint, Mass, …
Item : Point, Edge, Triangle, …

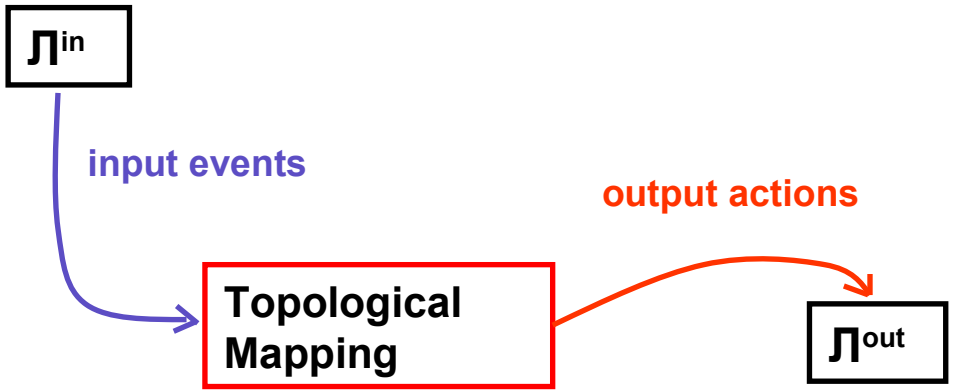


1. Hierarchical Architecture
2. Topological Events
3. Dynamic Data Structures
4. **Topological Mappings**

What is a Topological Mapping for ?

Creating a consistent topological object from a previous one,
such that this consistency remains under topological changes.

Λ^{in} : Topology of INPUT mesh
Λ^{out} : Topology of OUTPUT mesh



What is a Topological Mapping for ?

Application 1

- Provide multiple topological descriptions of an object, with the same degrees of freedom.

A quad

A quad
Decomposed into 2 triangles for FEM models

A quad
Decomposed into 6 edges for spring-mass models

Л^{out} is an alternative mesh from mesh **Л**^{in}

- Quad → Triangle
- Hexahedron → Tetrahedron

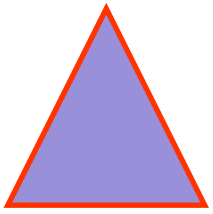
What is a Topological Mapping for ?

Application 2

- Create the topology of the boundary mesh,
  on which distinct forces, constraints, … can be applied

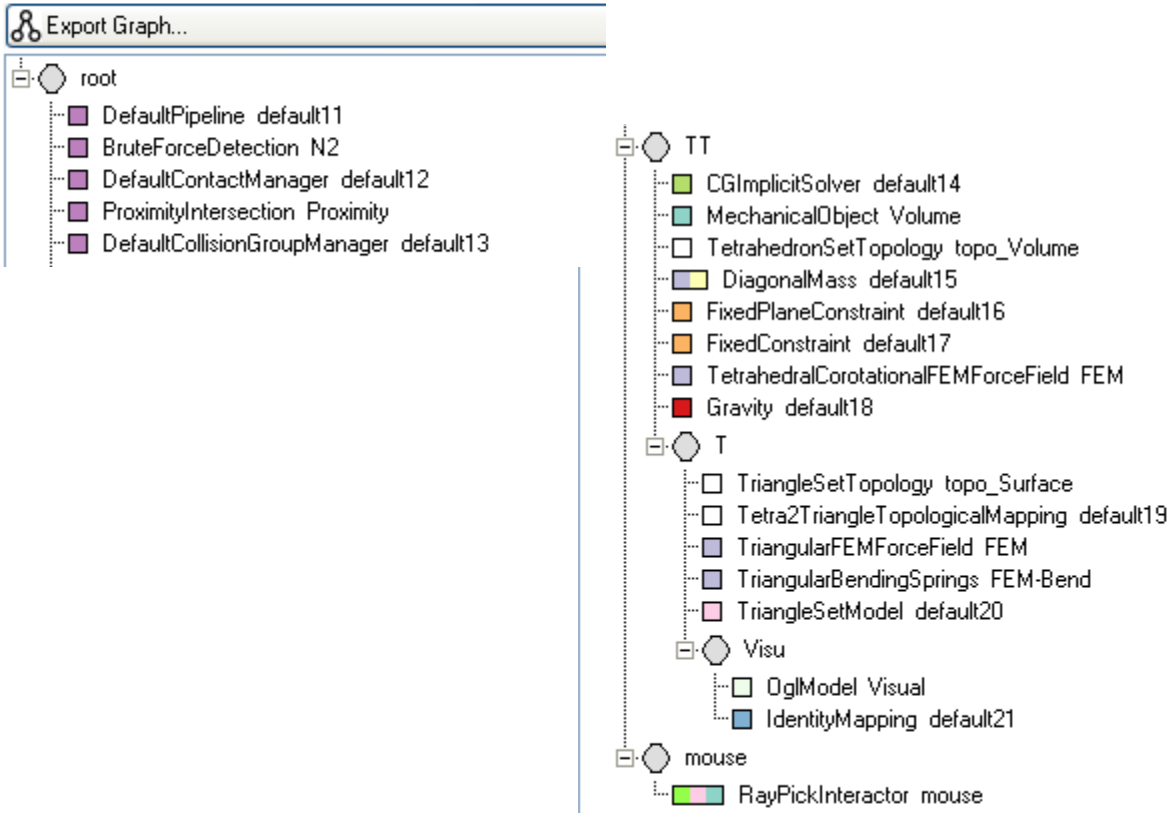
Л^{out} is a boundary mesh from mesh **Л**^{in}

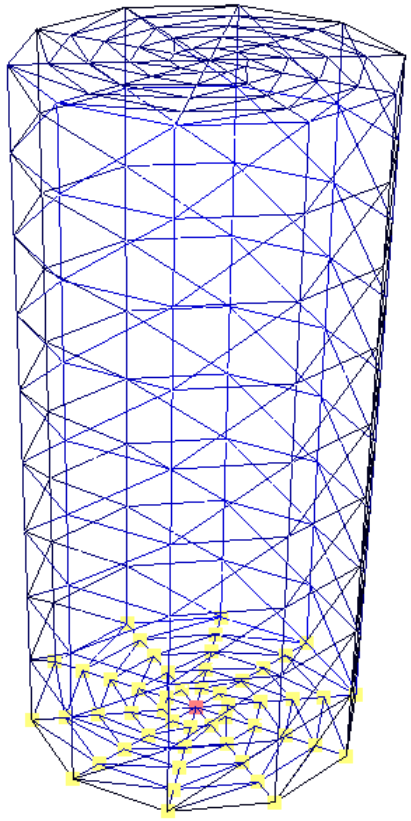
- Tetrahedron → Triangle
- Hexahedron → Quad
- Triangle → Edge
- Quad → Edge



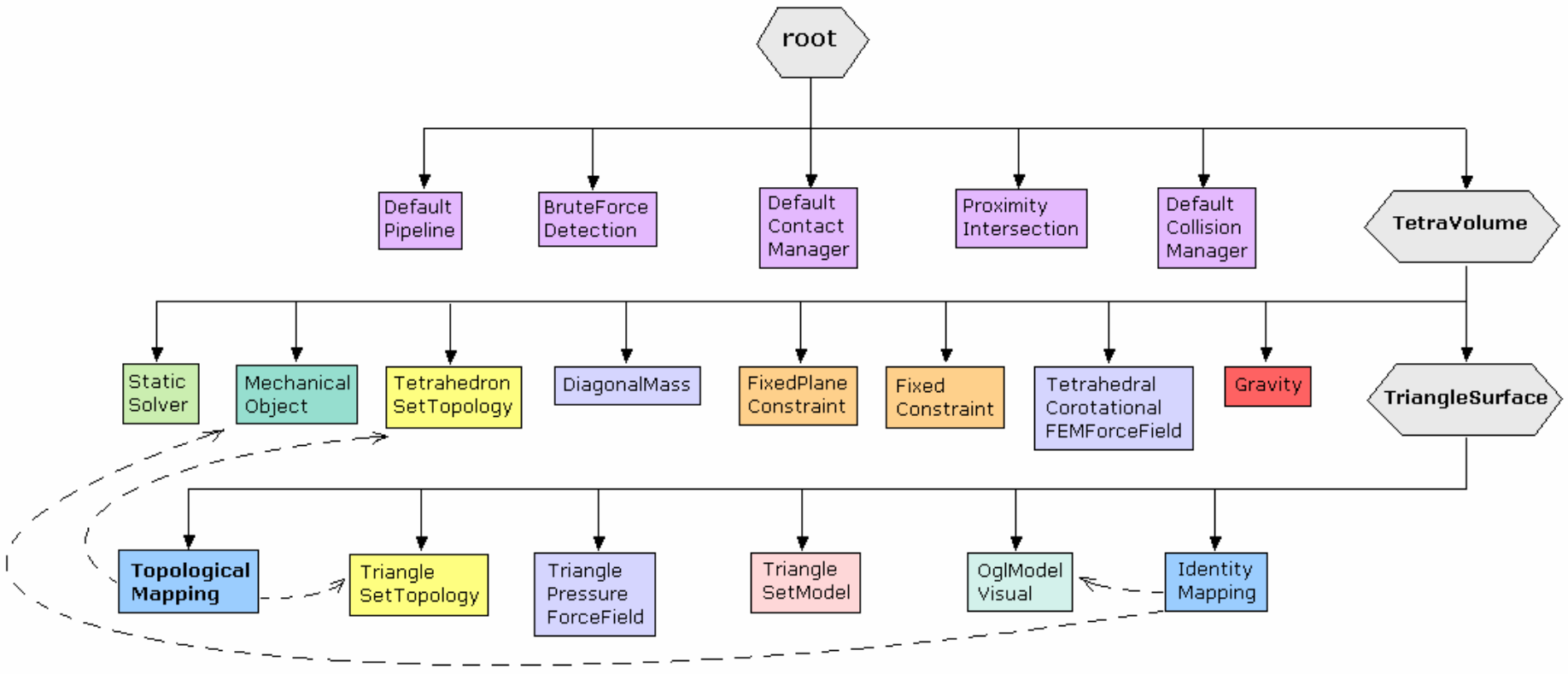
Use Case 2_

Simulate a bending cylinder as a tetrahedral volume and as a triangular surface (the cylinder's membrane)

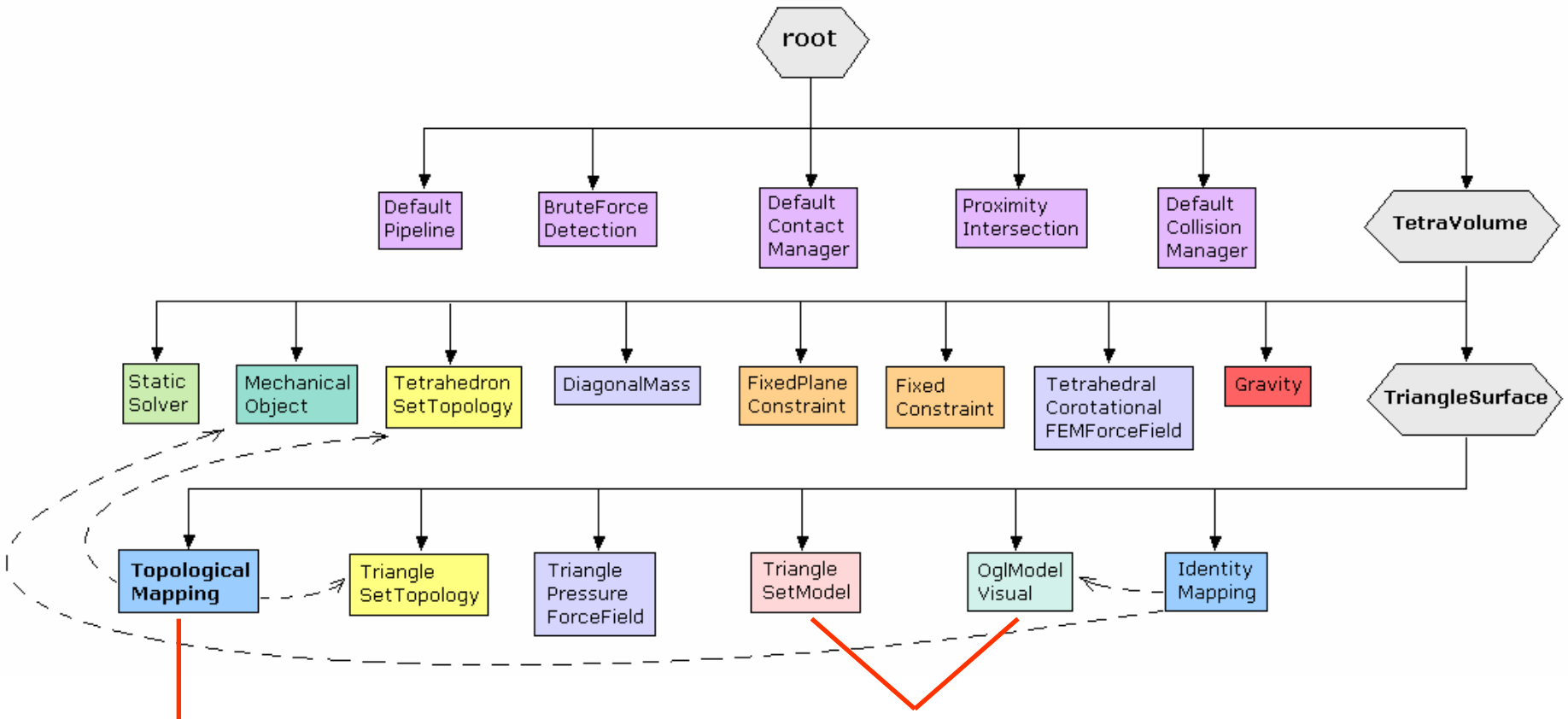




Scene Graph illustrating a Topological Mapping from a TetrahedronSetЛ to a TriangleSetЛ



Scene Graph illustrating a Topological Mapping from a TetrahedronSet Л to a TriangleSet Л



Note :
first written to be first initialized !

Advantage :
only work with surface triangles !

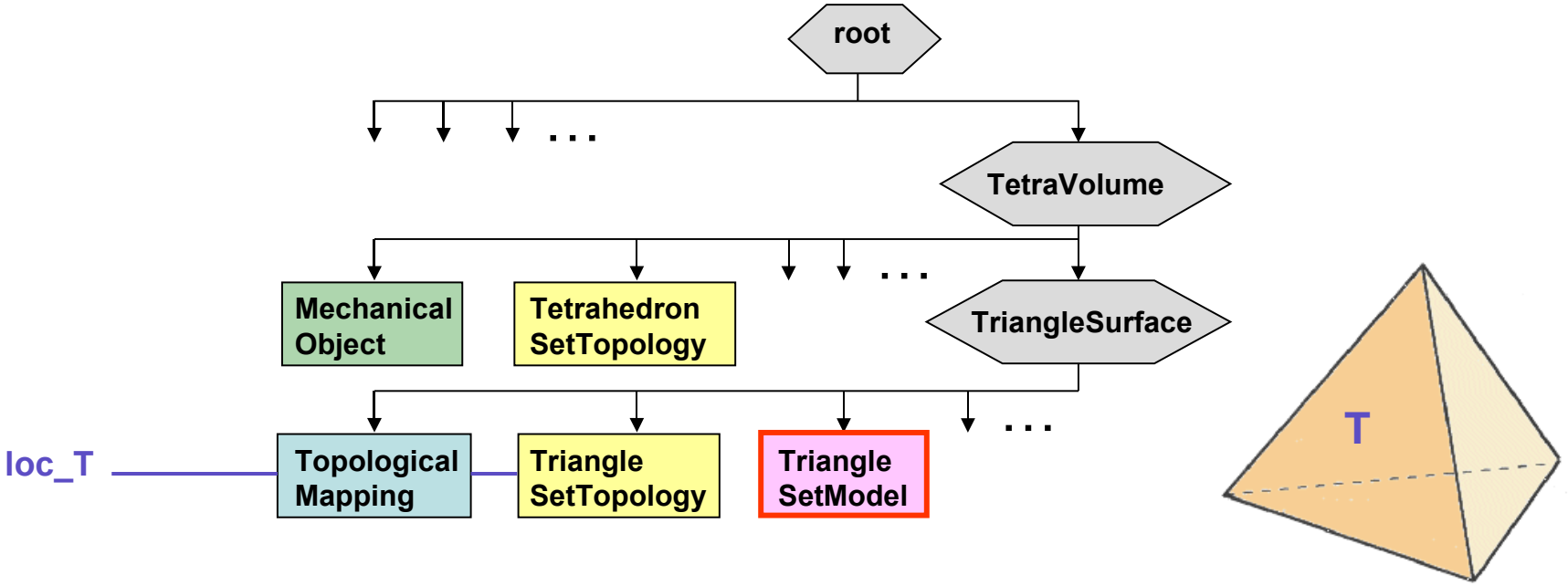
Scenario - 1)

User right-clicks on visible triangle T in the scene :

Interface "RayPickInteractor" **launches collision detection**

T detected by the Collision Model

T indexed by loc_T in the triangular surface mesh



Scenario - 2)

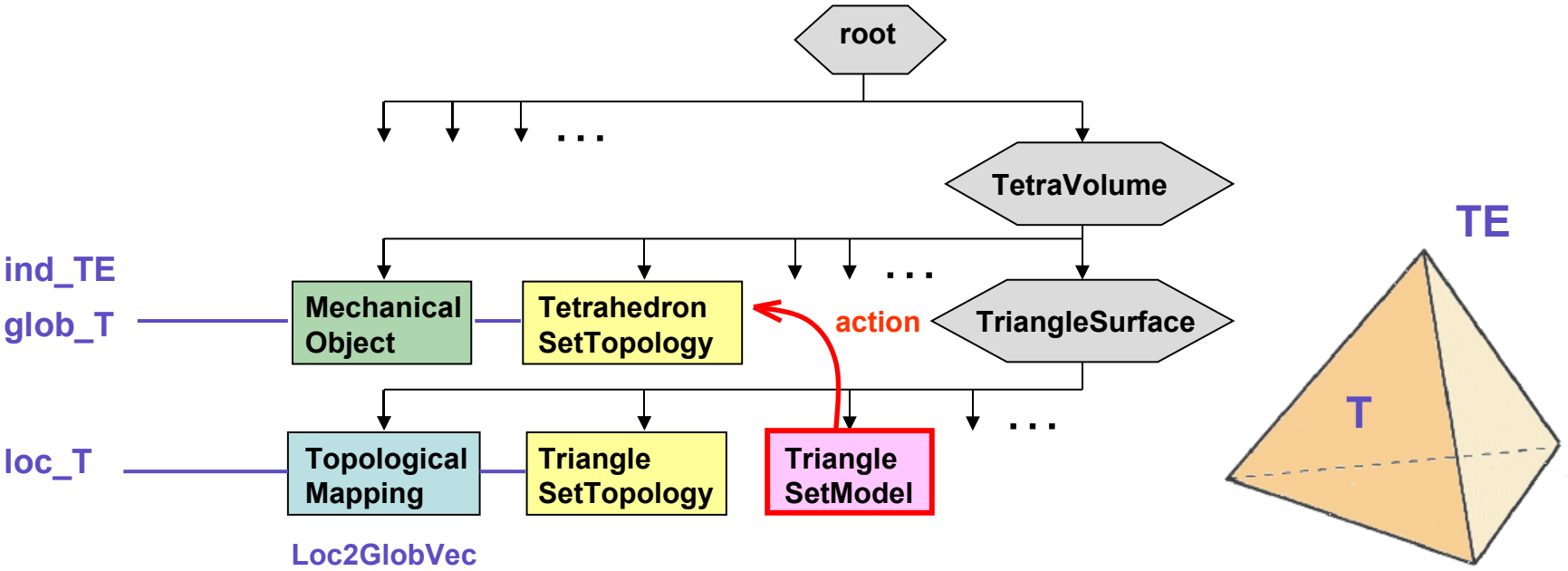
If Topological Mapping of type (input = TetrahedronSetЛ , output = TriangleSetЛ) :

Loc2GlobVec requested to give glob_T

T indexed by glob_T in the tetrahedral volume mesh

TetrahedronTriangleShell gives ind_TE = index of the tetrahedron TE containing T

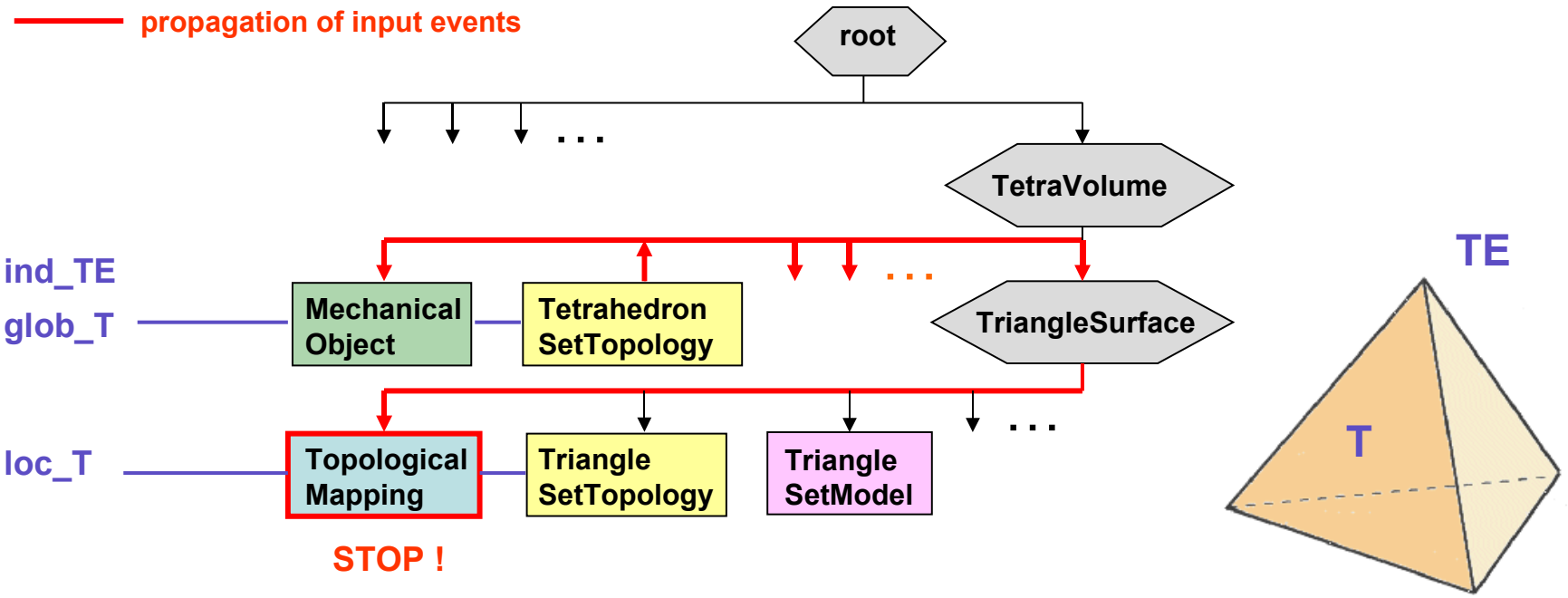
RemoveTetrahedra(< ind_TE >) action called on the input topology



Scenario - 3)

Tetrahedron
Triangle (s)
? Edge (s)
? Point (s)

} removal events are notified from TetrahedronSet Л



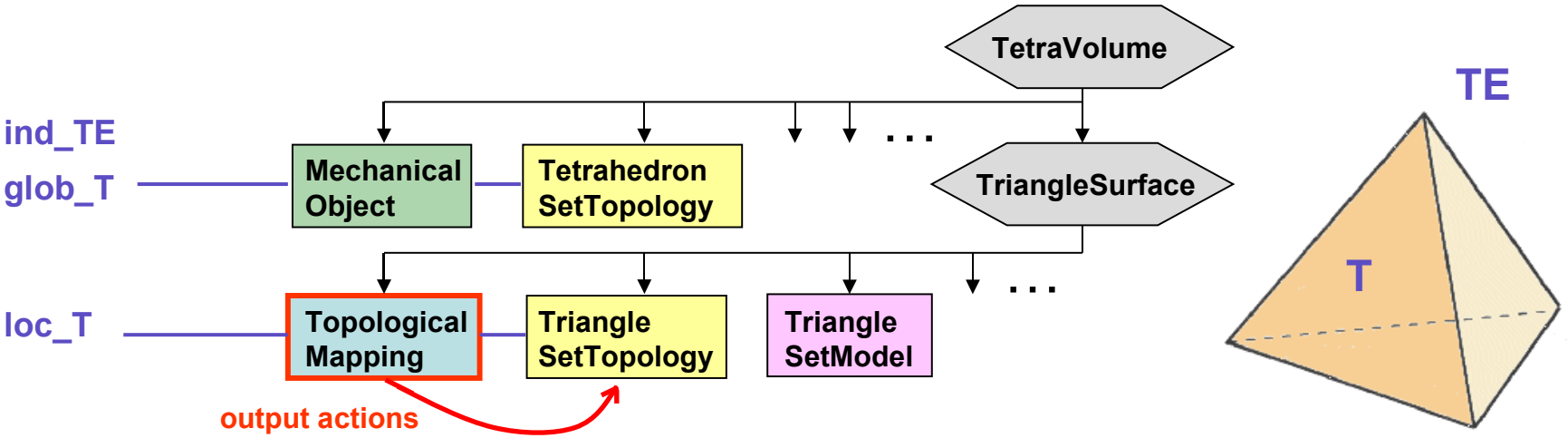
Scenario - 4)

Propagation reaches a Topological Mapping strictly lower in the scene graph, then it stops.

Tetra2TriangleTopologicalMapping translates input events into output actions on TriangleSet Л :

Tetrahedron removal → AddTriangles (< indices of new visible triangles >)
Triangle removal → RemoveTriangles (< indices of destroyed triangles >, removeDOF = false)

Index maps (Loc2GlobVec, Glob2LocMap, In2OutMap) are requested and updated to maintain the correspondence between the items indices in input and output topologies.



Scenario - 5)

Triangle (s)
? Edge (s)
? Point (s)
} removal events are notified from TriangleSet Л