Towards a Domain Specific Language for a Scene Graph based Robotic World Model

Sebastian Blumenthal, **Herman Bruyninckx**
Mechanical Engineering, KU Leuven, Belgium
sebastian.blumenthal @ mech.kuleuven.be

DSLRob-13, Tokyo, Nov 8, 2013

Outline

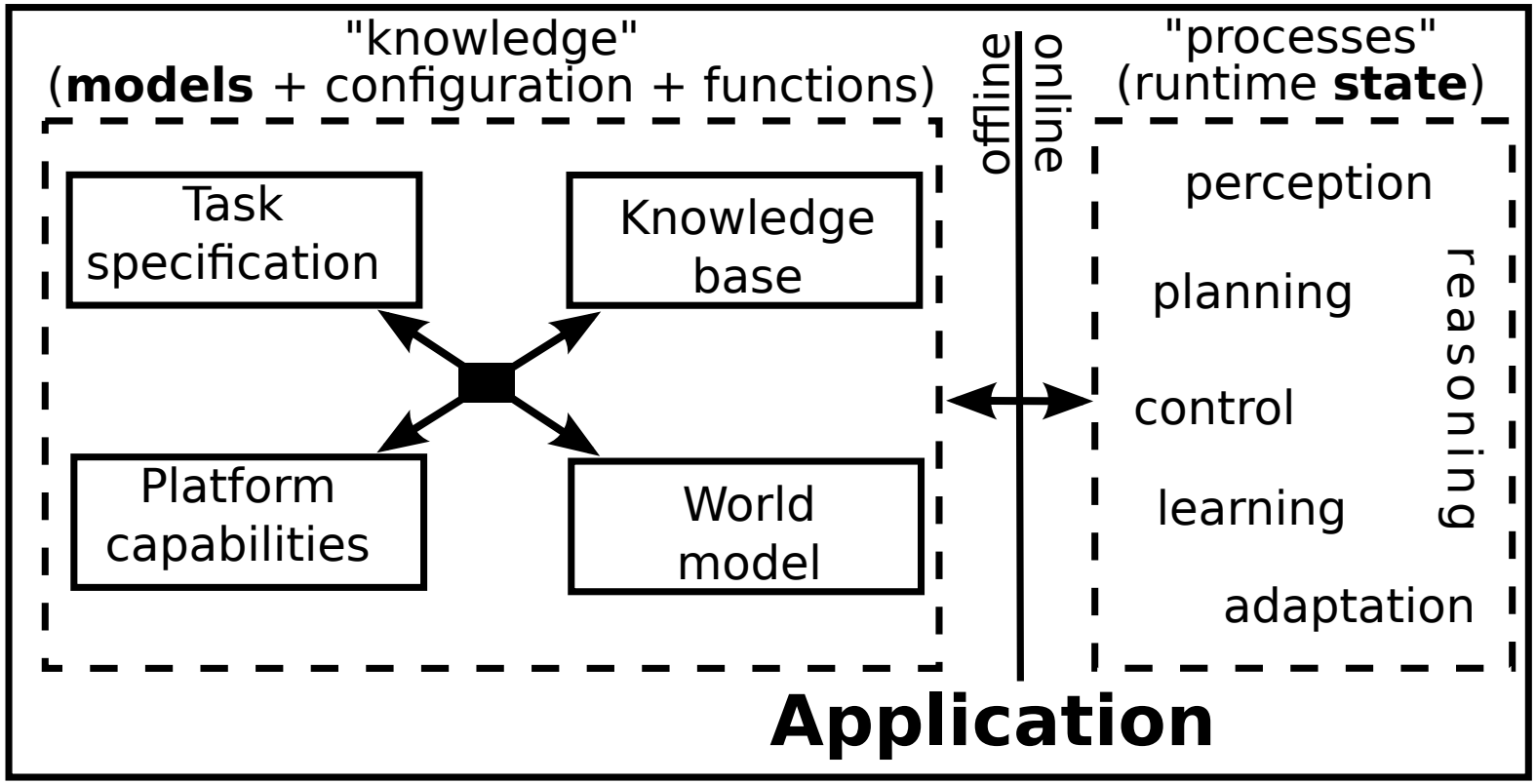
Introduction

The Robot Scene Graph: RSG

The RSG-DSL

Conclusion

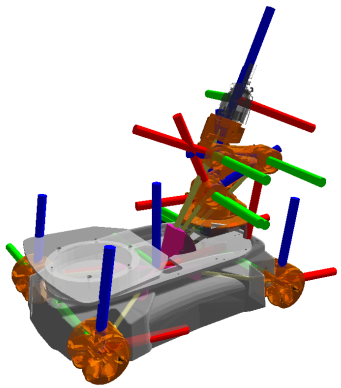
DSLs for robotic applications

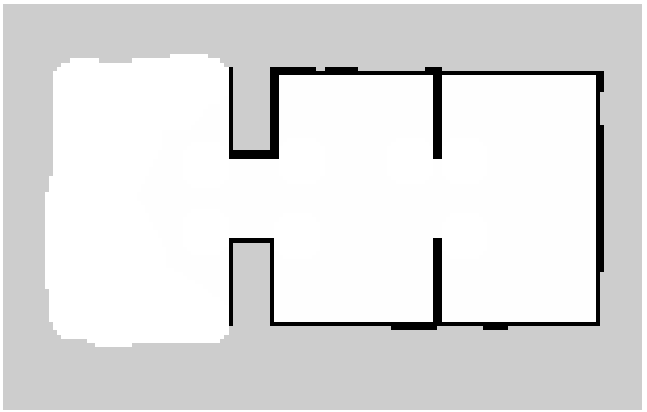


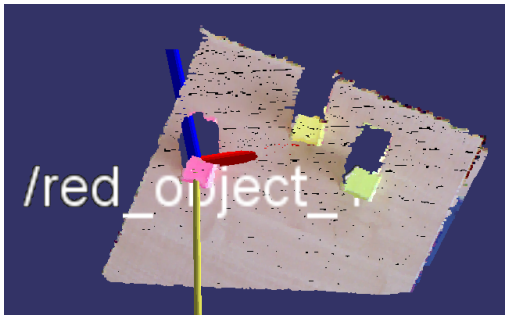
- (Prior) Knowledge
  - Robot, Task, ... **World Model**
- Algorithms
  - For various domains: planning, perception, control, ...
- Framework for execution and communication

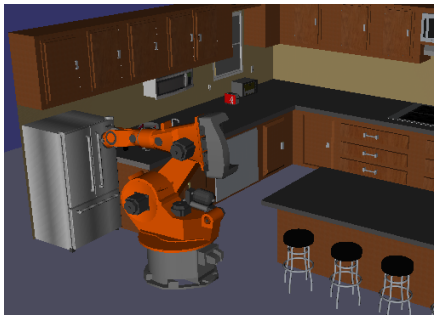
Existing World Models

- Control
  - Kinematic chains
  - Geometry
- Navigation
  - Grids
- Perception
  - 2D/3D images
  - Features
  - Poses
- Planning
  - (Triangle) Meshes



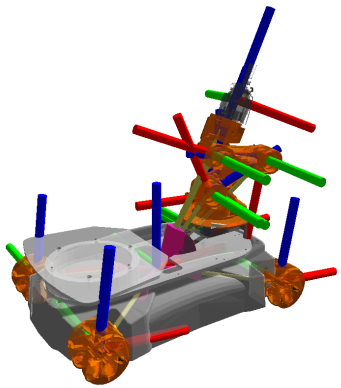


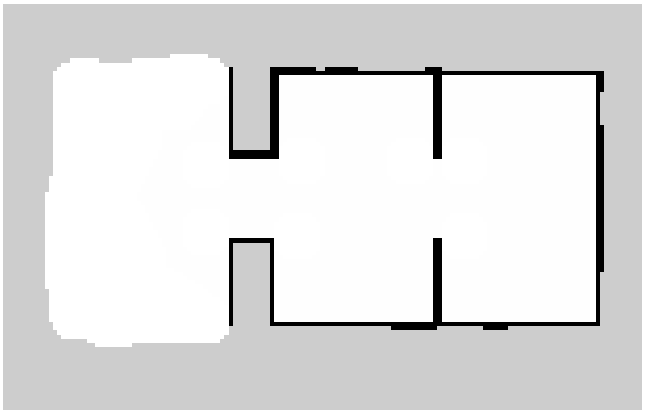


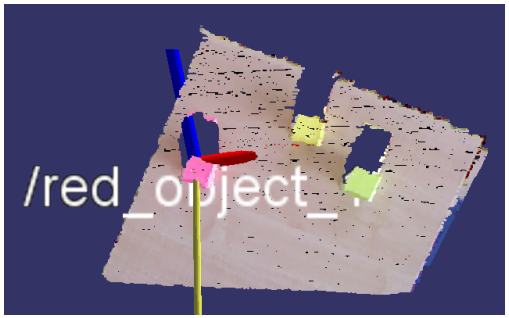


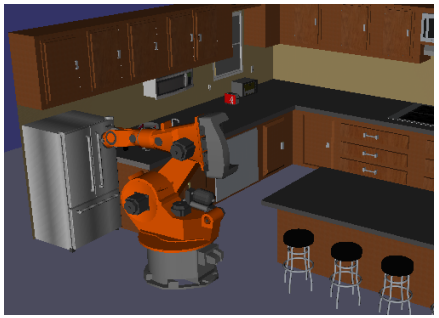
Existing World Models

- Control
  - Kinematic chains
  - Geometry
- Navigation
  - Grids
- Perception
  - 2D/3D images
  - Features
  - Poses
- Planning
  - (Triangle) Meshes
- Not well connected yet!

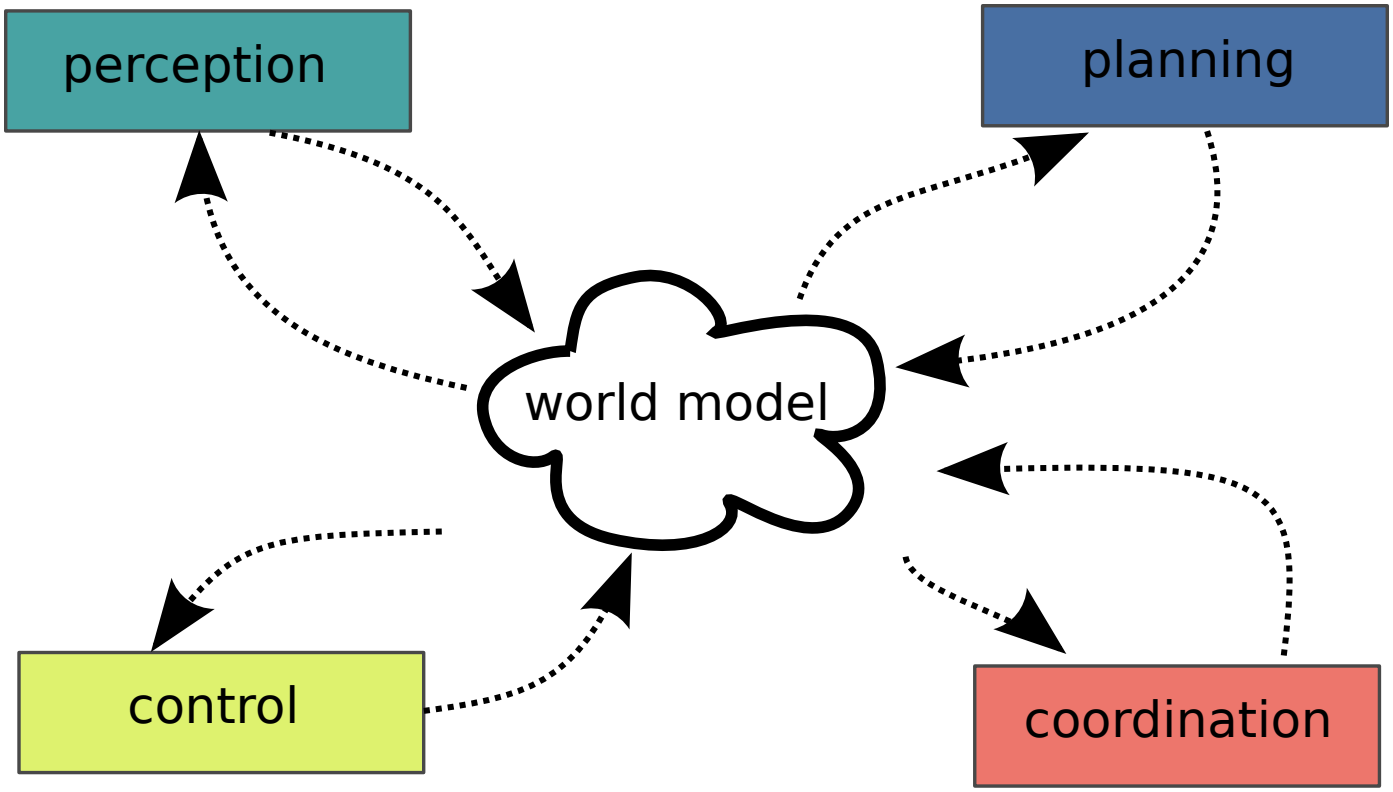






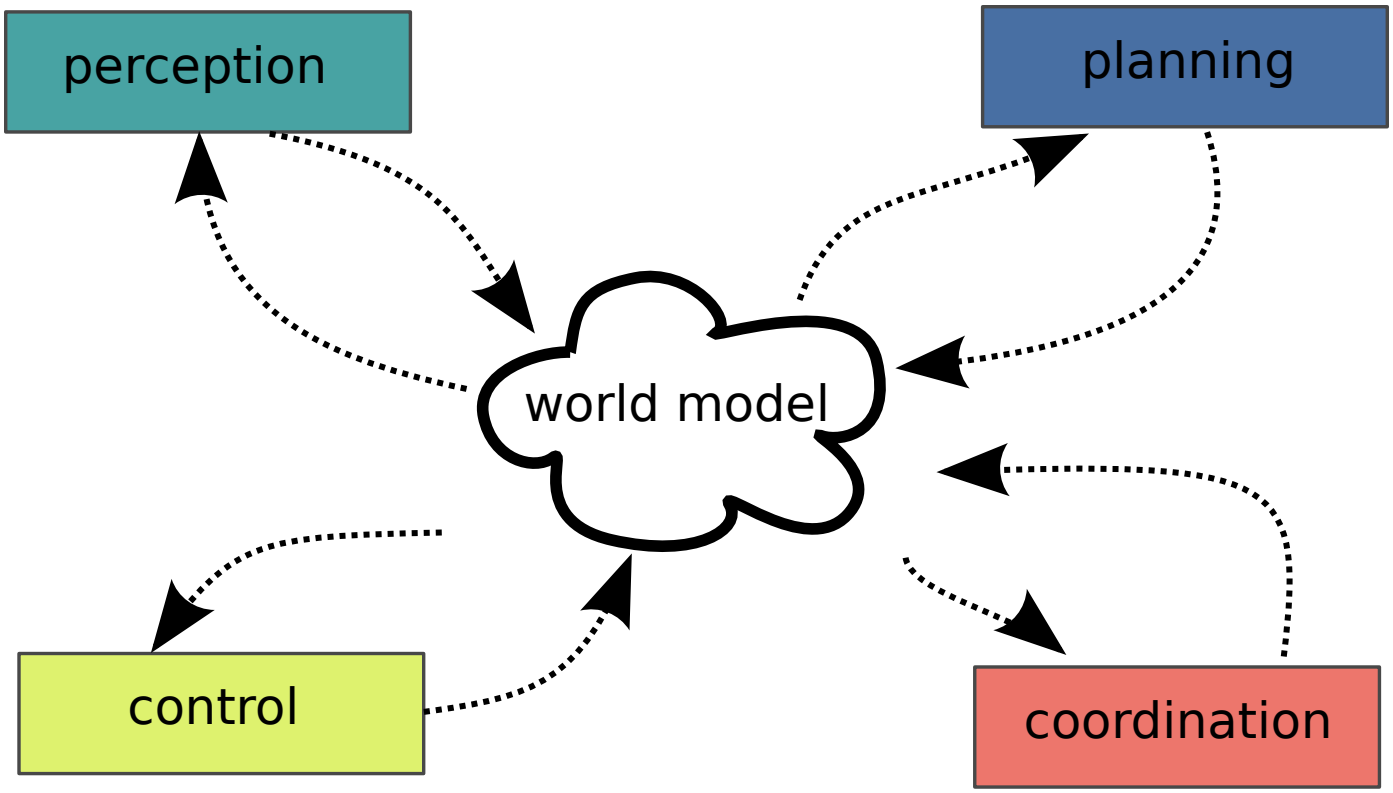


Shared world model for robot applications



- **World Model**: as shared (distributed) resource with dynamic scene, multi-resolution and uncertainty support

Shared world model for robot applications



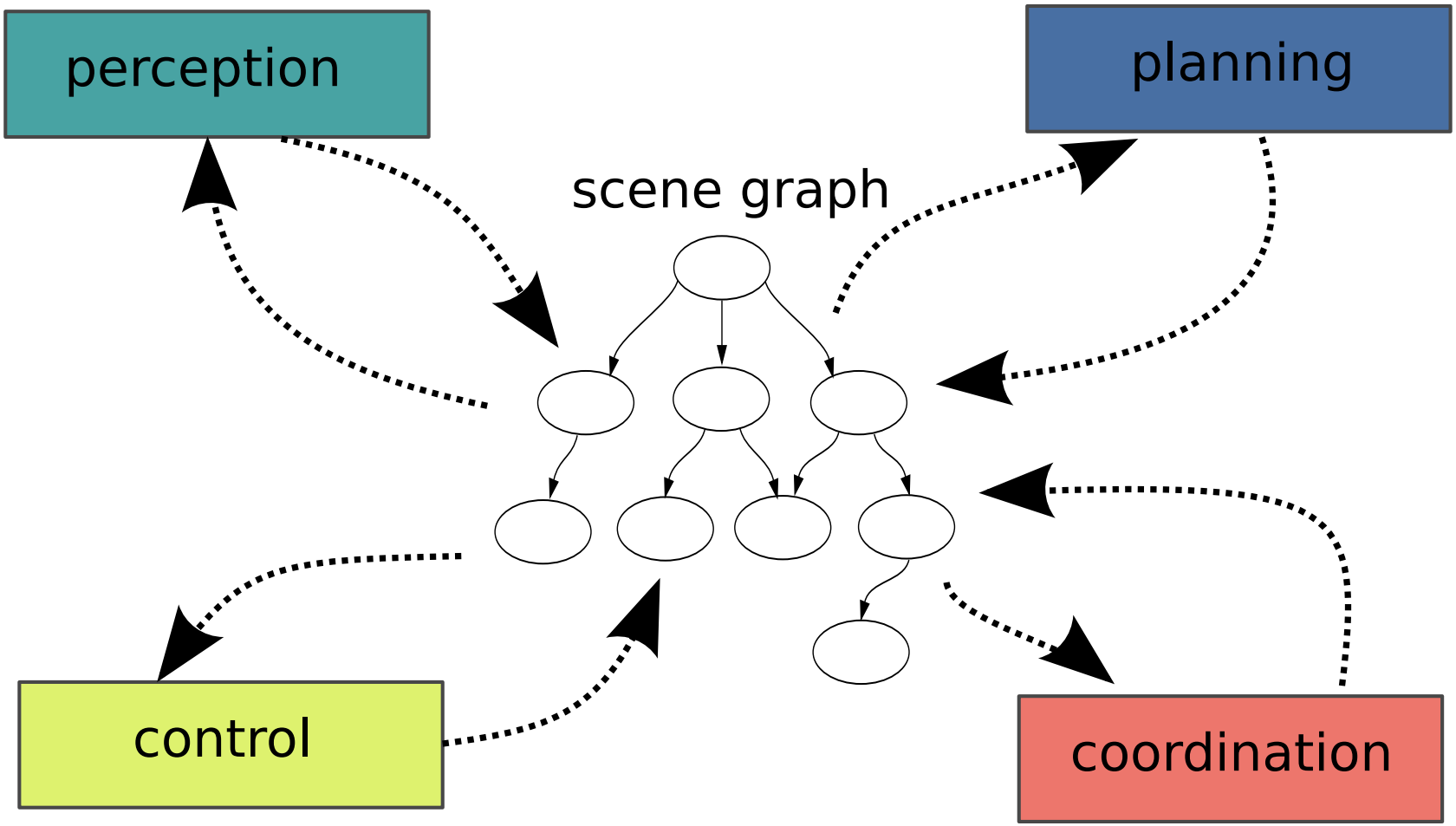
- **World Model**: as shared (distributed) resource with dynamic scene, multi-resolution and uncertainty support
- **Problem**: Does not exist yet!
  - Concept
  - DSL

Contributions

- M3: Ecore framework
- **M2: Xtext based DSL for a world model**
- M1: Editor via Eclipse toolchain
- M0: *Prior work:* C++ implementation (BRICS_3D library), **code generation from M2 level**

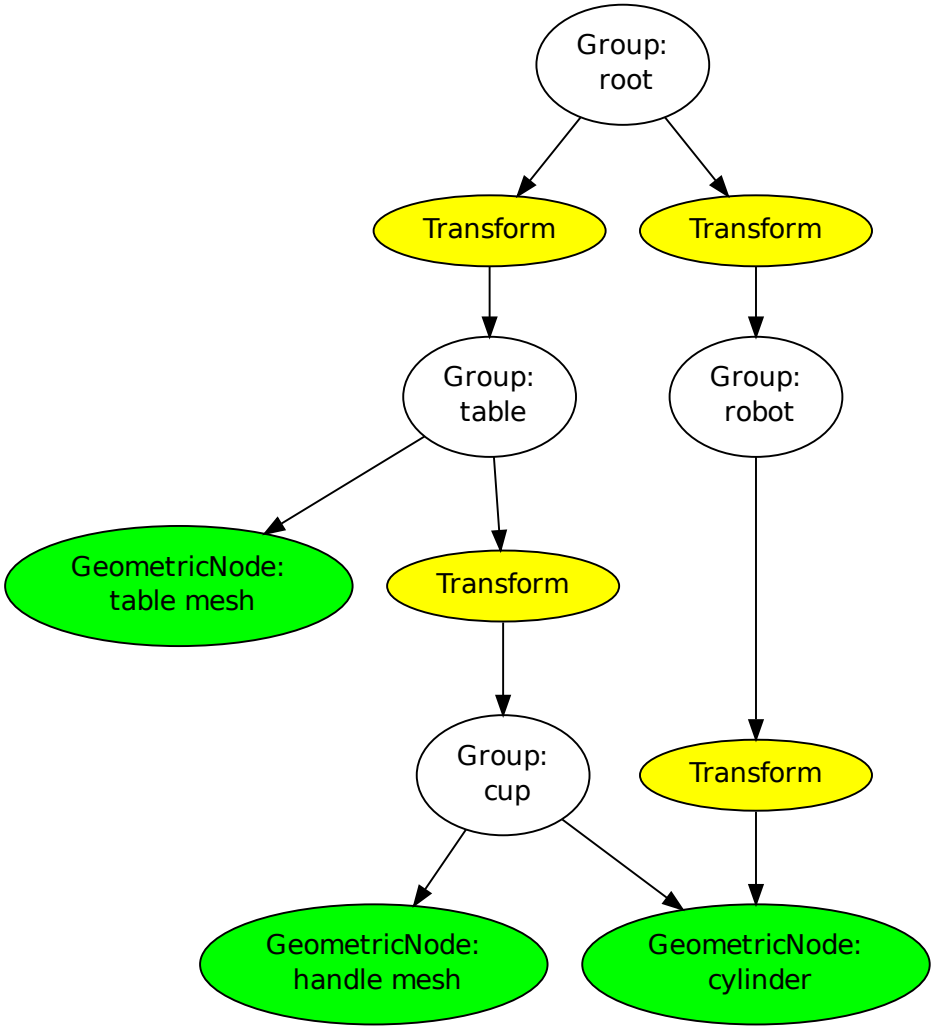
The Robot Scene Graph: RSG

RSG: Robot Scene Graph



RSG: Robot Scene Graph

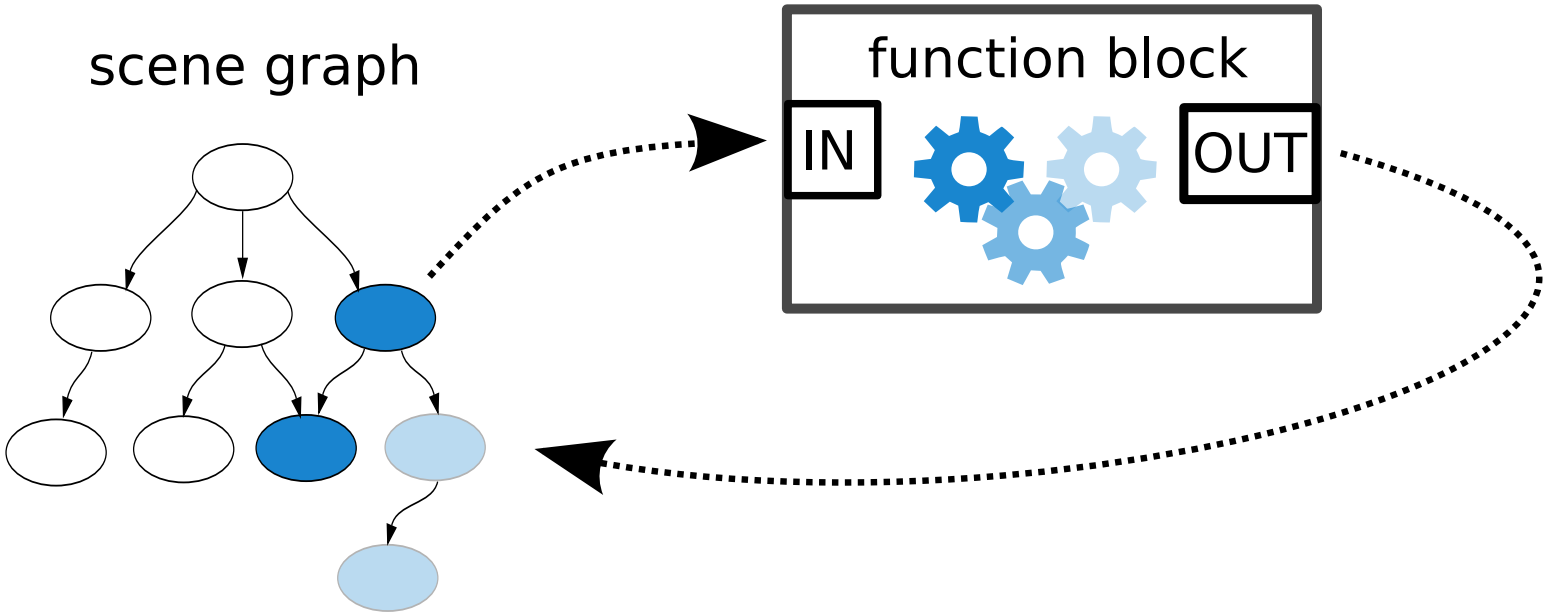
- RSG contains:
  - **Objects**
    - Low level like a point cloud
    - High level *scene objects*
  - **Relations**
  - Both organized in a *Directed Acyclic Graph (DAG)*
- Example graph
  - Scene with table
  - Cup on table
  - Robot observes cup



RSG: Nodes

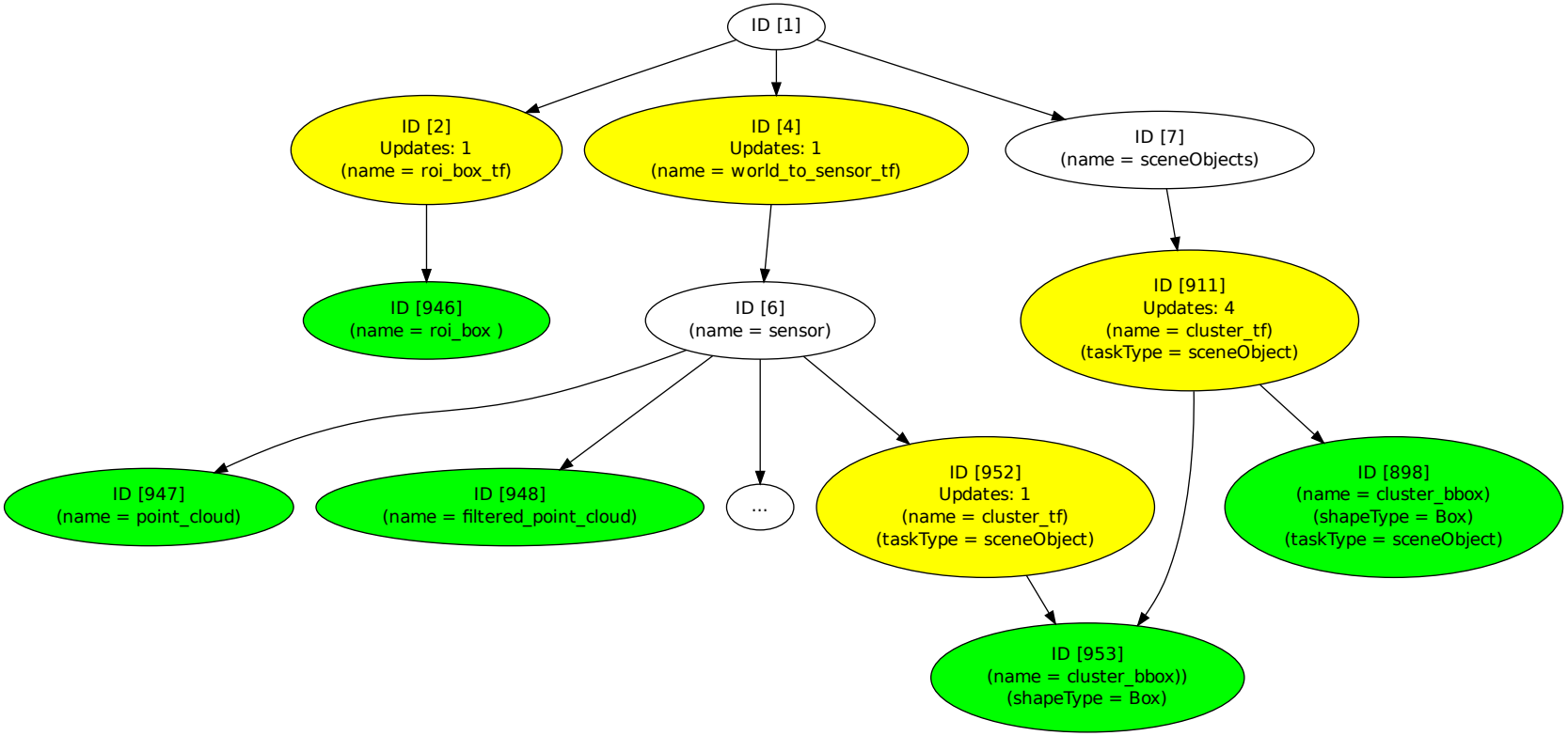
- Nodes
  - Unique IDs
  - Attributes
- GeometricNodes
  - Point Cloud
  - Mesh
  - Box
  - Cylinder
- Groups
- Transforms
  - Cache with time stamps

RSG: Function blocks



- A function block is a generic computation hosted by the world model
- Consumes a part of the world model and might perform updates or add new data
- Inputs and outputs are based on (unique) IDs

RSG: Example

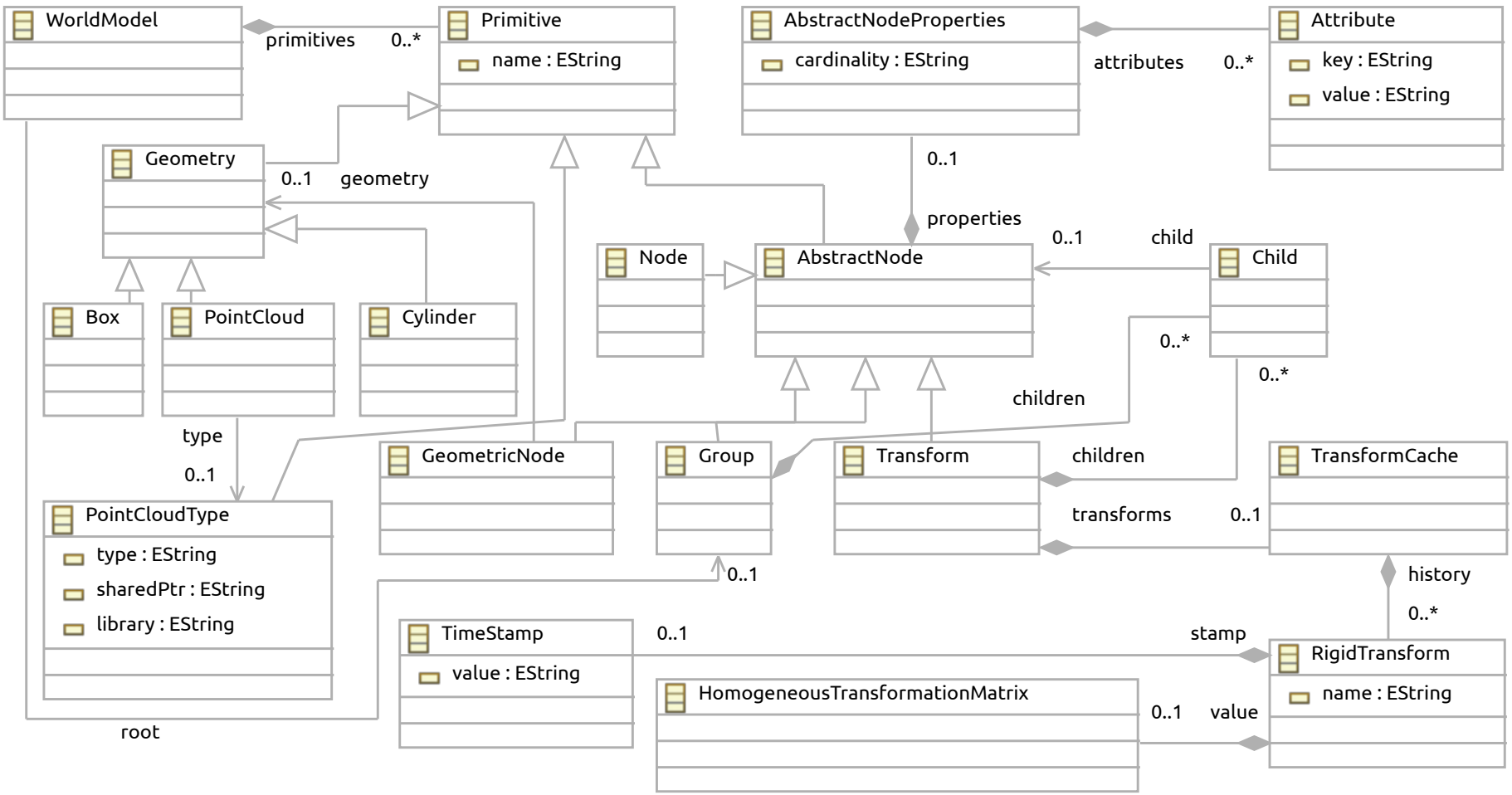

ID [1]
ID [2]
Updates: 1
(name = roi_box_tf)
ID [4]
Updates: 1
(name = world_to_sensor_tf)
ID [7]
(name = sceneObjects)
ID [946]
(name = roi_box)
ID [6]
(name = sensor)
ID [911]
Updates: 4
(name = cluster_tf)
(taskType = sceneObject)
ID [947]
(name = point_cloud)
ID [948]
(name = filtered_point_cloud)
...
ID [952]
Updates: 1
(name = cluster_tf)
(taskType = sceneObject)
ID [898]
(name = cluster_bbox)
(shapeType = Box)
(taskType = sceneObject)
ID [953]
(name = cluster_bbox))
(shapeType = Box)

The RSG-DSL

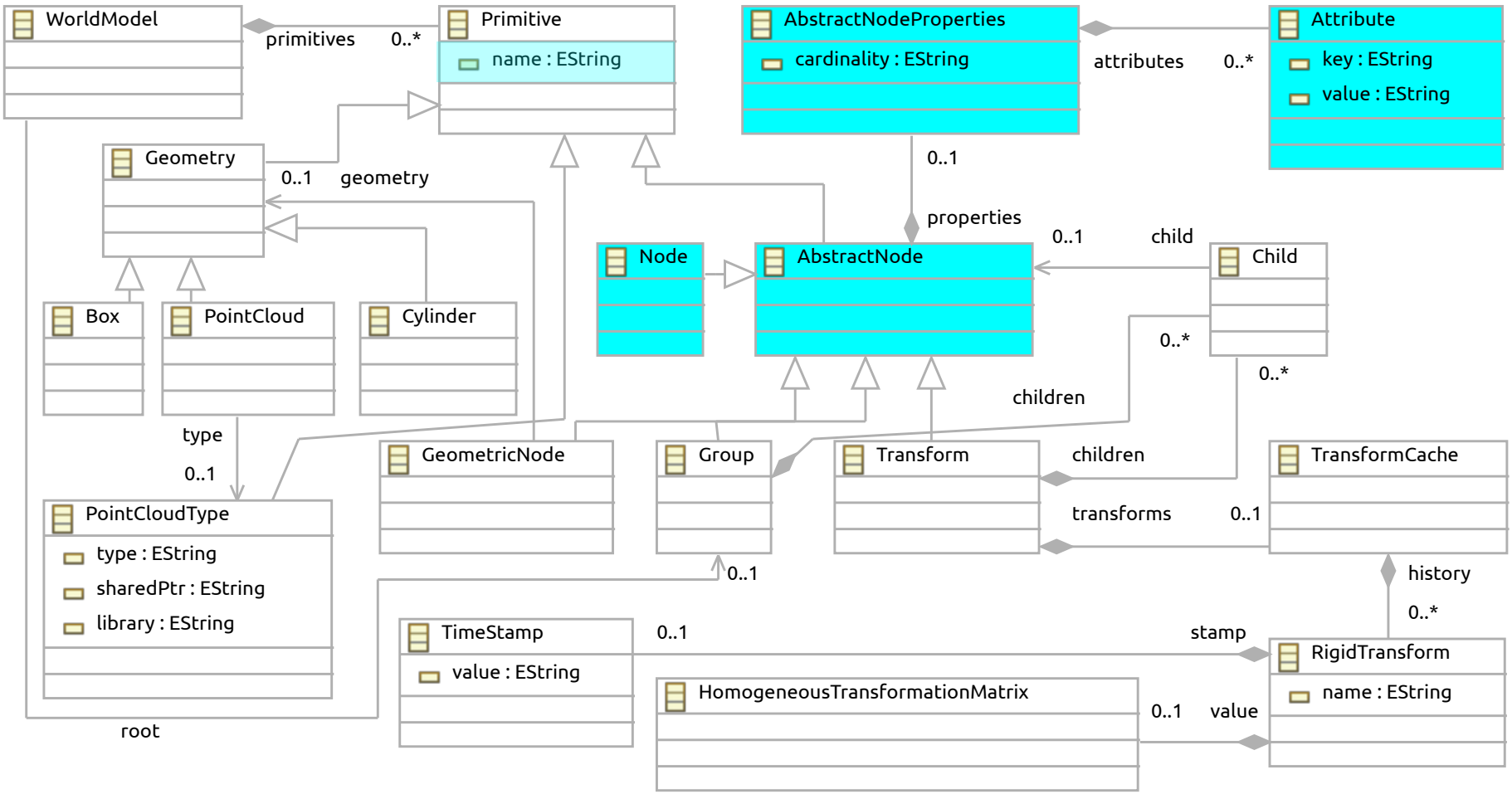
RSG-DSL Overview

- **Primitives**: Node types, function blocks
- **Relationships**: DAG structure
- **Transformations**: Model-to-text (C++)

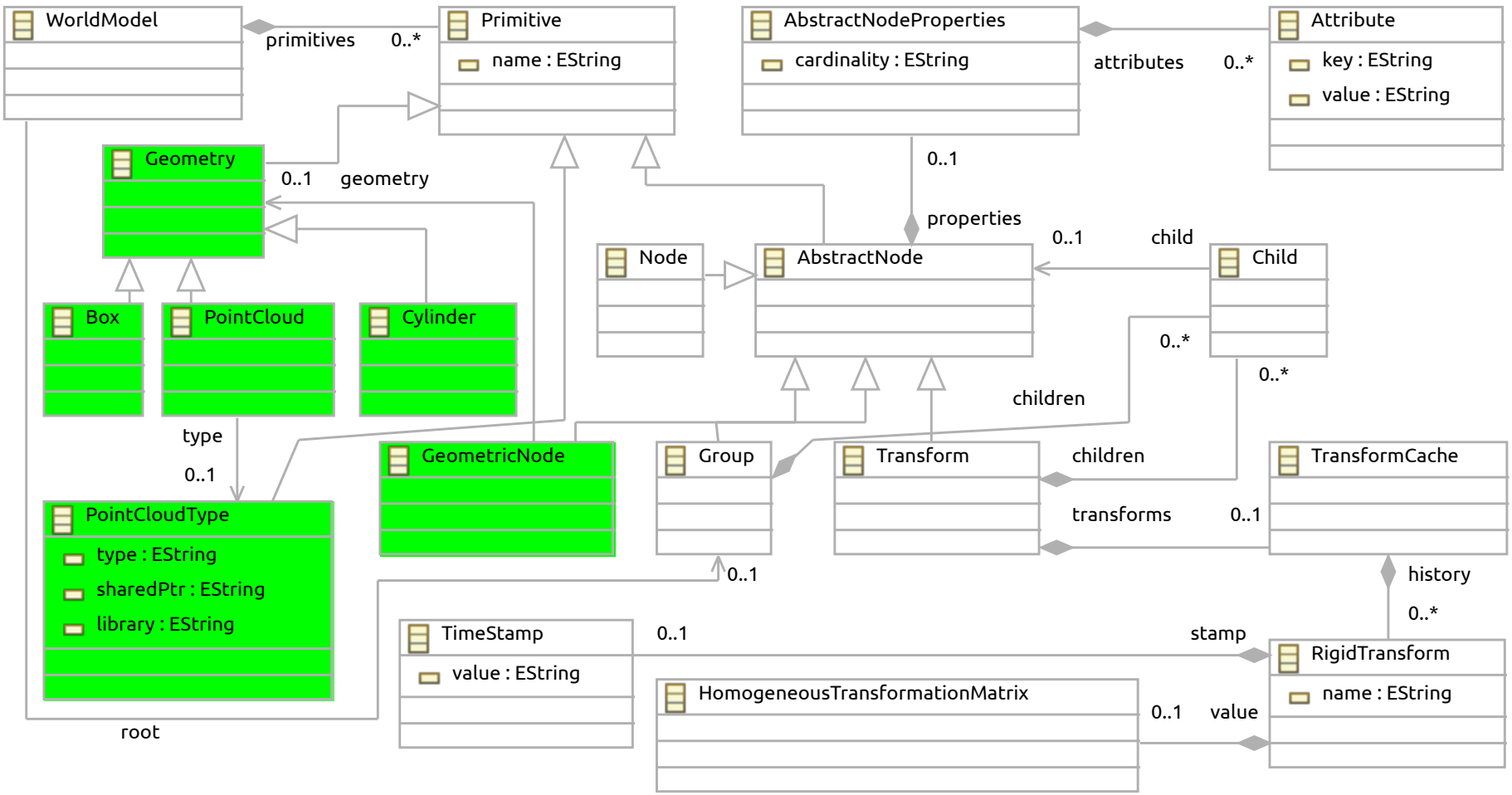
DSL Primitives for RSG node types



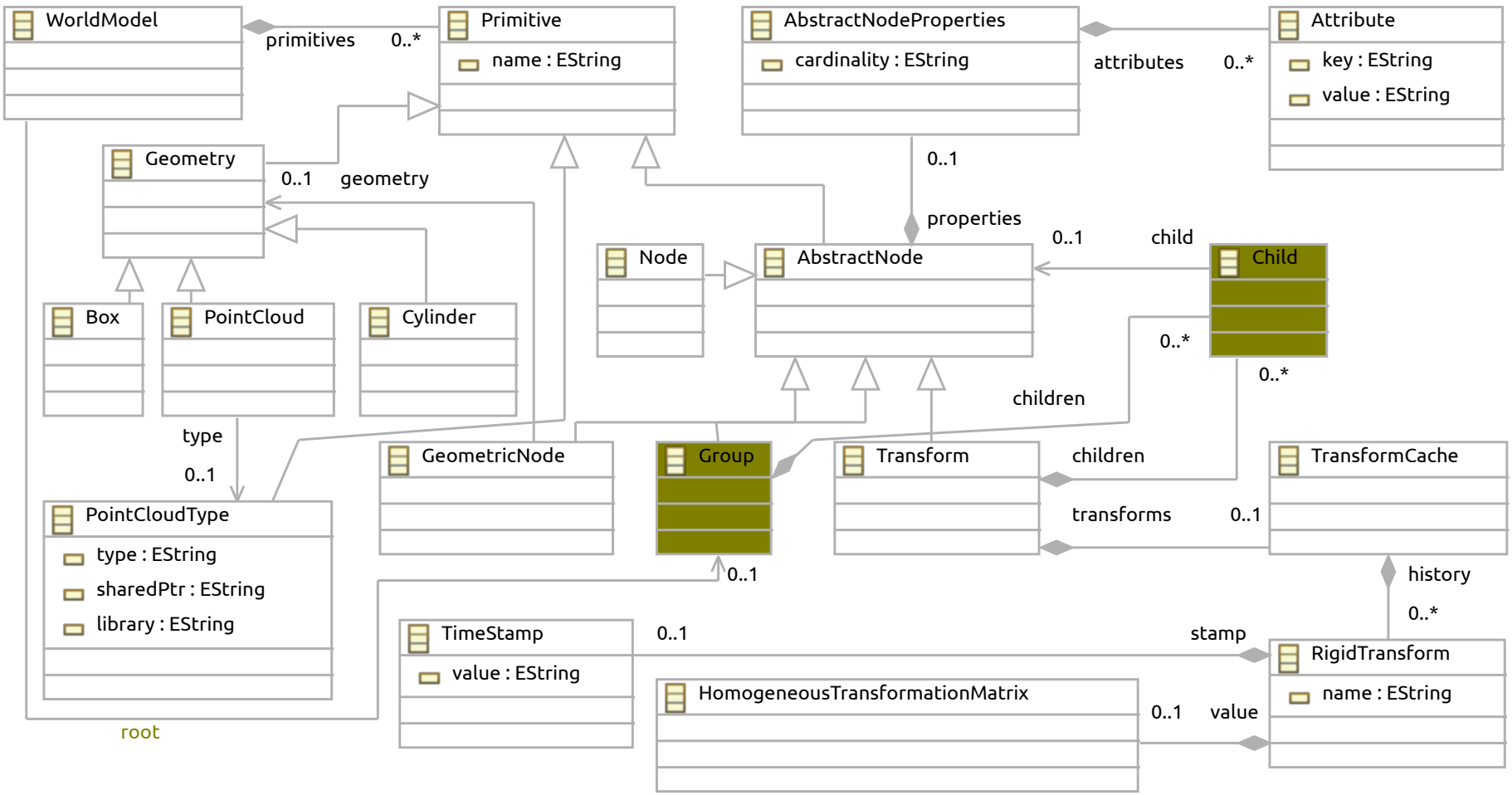
DSL Primitives for RSG node types



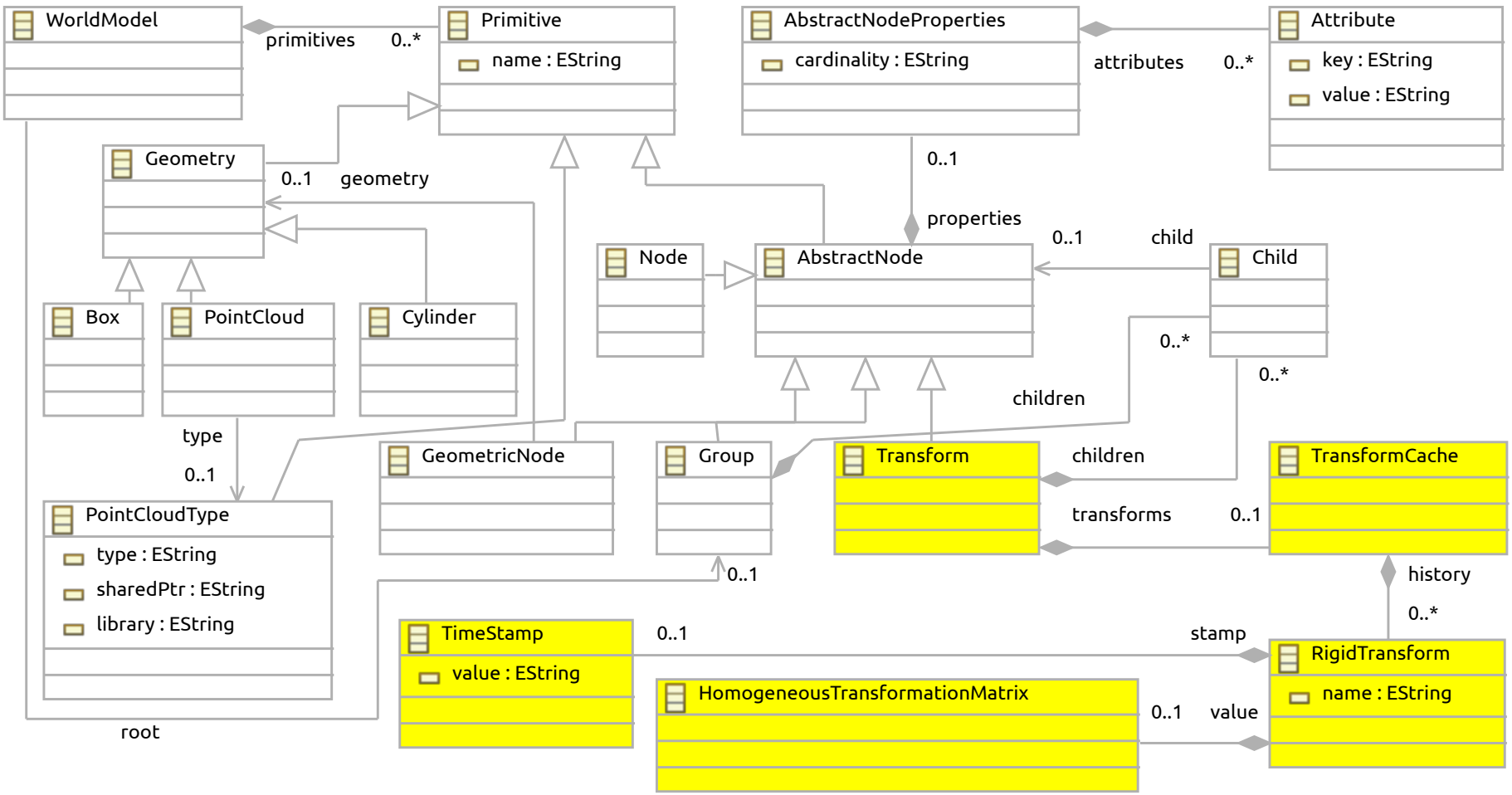
DSL Primitives for RSG node types



DSL Primitives for RSG node types



DSL Primitives for RSG node types



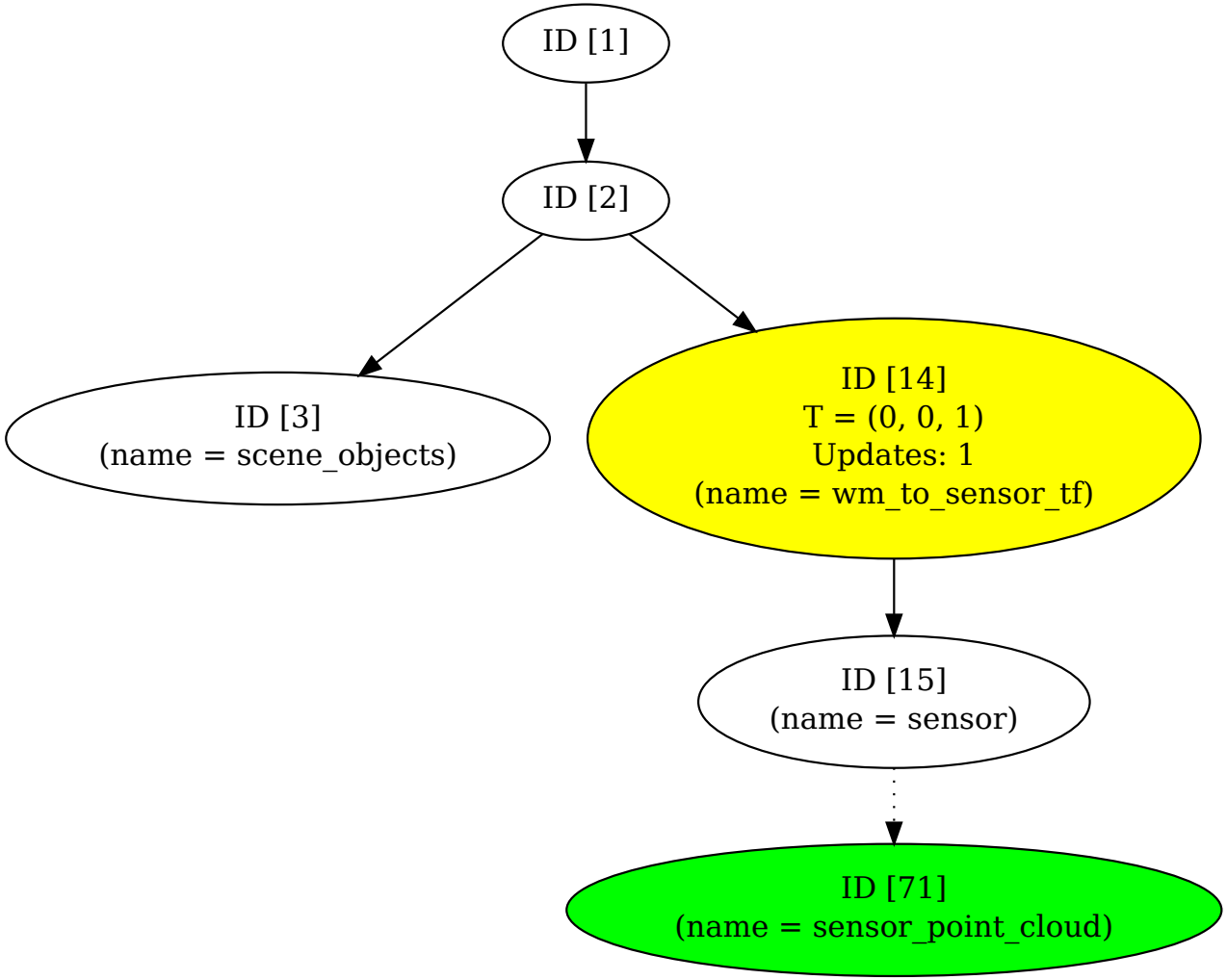
Example scene setup

```
root rootNode // application scene

Group rootNode {
  child group1
  child worldToCamera
}

Group group1 {
  Attribute ("name", "scene_objects")
  child kitchenTable
}

Transform worldToCamera {
  Attribute ("name", "wm_to_sensor_tf")
  child sensor
  transforms {
    RigidTransform t1 {
      stamp TimeStamp ( 0.0 s)
      value HomogeneousTransformationMatrix (
      [1.0, 0.0, 0.0, 0.0 m ],
      [0.0, 1.0, 0.0, 0.0 m ],
      [0.0, 0.0, 1.0, 1.0 m ],
      [0.0, 0.0, 0.0, 0.0   ])
    }
  }
}
Group sensor {
  Attribute ("name", "sensor")
}
```

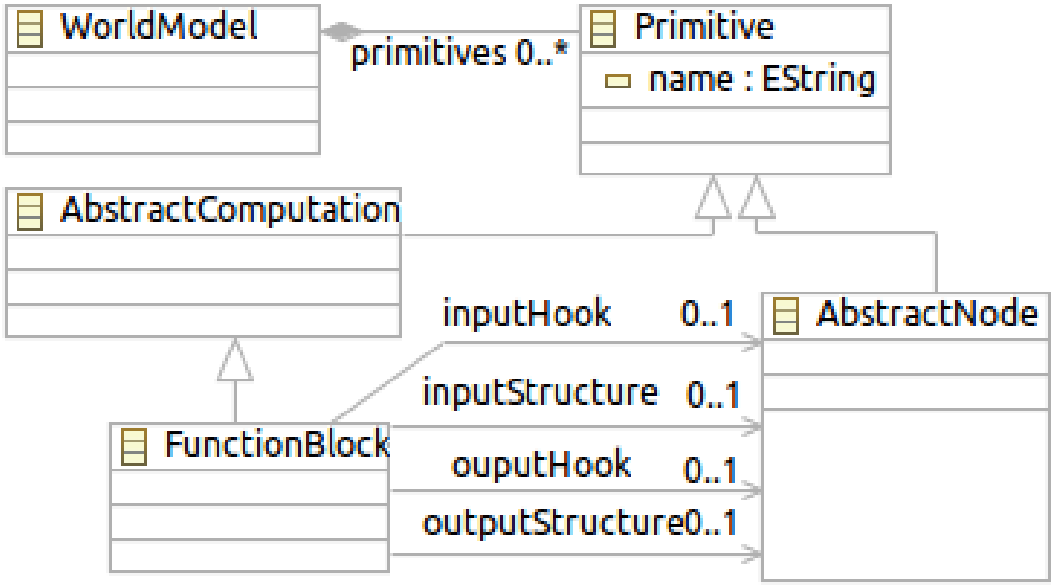

Generated code (excerpt)

```
1  std::vector<rsg::Attribute> attributes; // Instantiation of list of attributes.
2  unsigned int rootNodeId;        // IDs correspond to names in model on M1 level.
3  unsigned int group1Id;
4  unsigned int worldToCameraId;
5  // [...]
6
7  /* Add group1 as a new node to the scene graph */
8  attributes.clear();
9  attributes.push_back(Attribute ("name", "scene_objects"));
10 wm->scene.addGroup(rootNodeId, group1Id, attributes); // group1Id = output
11 // [...]                                              // -> returns a unique ID.
12
13 /* Add worldToCamera as a new node to the scene graph */
14 attributes.clear();
15 attributes.push_back(Attribute ("name", "wm_to_sensor_tf"));
16 brics_3d::IHomogeneousMatrix44::IHomogeneousMatrix44Ptr worldToCameraInitialTf(
17   new brics_3d::HomogeneousMatrix44(
18     1.0, 0.0, 0.0,
19     0.0, 1.0, 0.0,
20     0.0, 0.0, 1.0,
21     0.0 * 1.0, 0.0 * 1.0, 1.0 * 1.0  // Values are scaled to SI unit [m].
22 ));
23
24 wm->scene.addTransformNode(rootNodeId, worldToCameraId, attributes,
25   worldToCameraInitialTf, brics_3d::rsg::TimeStamp(0.0, Units::Second)
26 );       // Value is scaled to SI unit [s].
```


Resulting scene graph



DSL Primitives for RSG function blocks



Example function block

```
PointCloudType PointCloudPCL {
  type "pcl::PointCloud<PointType>"
  sharedPtr "pcl::PointCloud<PointType>::Ptr"
  library "pcl"
}

PointCloud inputCloud type PointCloudPCL
PointCloud planeCloud type PointCloudPCL

GeometricNode pointCloud {
  Attribute ("name", "point_cloud")
  geometry inputCloud
}

Group planes {
  Attribute ("name", "planes")
  child tfToPlaneCentroid
}
```

```
GeometricNode horizontalPlane {
  Attribute ("name", "plane")
  geometry planeCloud
}

FunctionBlock horizontalPlaneSegmentation {
  inputStructure pointCloud
  inputHook sensorPointCloud
  outputStructure planes
  outputHook sensor
}
```


Generated code (excerpt)

```
1  /* Interface stub for HorizontalPlaneSegmentation function block */
2  class HorizontalPlaneSegmentation : public rsg::IFunctionBlock {
3  public:
4
5    /* used input IDs */
6    unsigned int pointCloudId;
7
8    /* used output IDs */
9    unsigned int planesId;
10   unsigned int cloudCenterToPlaneCenterId;
11   unsigned int horizontalPlaneId;
12
13   HorizontalPlaneSegmentation(brics_3d::WorldModel* wmHandle) : IFunctionBlock
         (wmHandle) {};
14   virtual ~HorizontalPlaneSegmentation(){};
15
16   virtual void configure(brics_3d::ParameterSet parameters){}
17   virtual void execute(){}
18
19 protected:
20
21 /* data conventions */
22 const static unsigned int horizontalPlaneSegmentationInputHookId  = 0;
23 const static unsigned int horizontalPlaneSegmentationOutputHookId  = 1;
```


Generated code (excerpt)

```
1   pcl::PointCloud<PointType>::Ptr getInputCloudPointCloud() {
2     pcl::PointCloud<PointType>::Ptr inputCloudData (new pcl::PointCloud<
          PointType>());
3
4     /* Query the world model based on the input id pointCloudId  */
5     brics_3d::rsg::TimeStamp resultTimeStamp;
6     brics_3d::rsg::Shape::ShapePtr shape;
7     wm->scene.getGeometry(pointCloudId, shape, resultTimeStamp);
8
9     /* Resolve (raw) data type known by model */
10    brics_3d::rsg::PointCloud<pcl::PointCloud<PointType> >::PointCloudPtr
          inputCloudContainer(new brics_3d::rsg::PointCloud<pcl::PointCloud<
          PointType> >());
11    inputCloudContainer = boost::dynamic_pointer_cast<brics_3d::rsg::PointCloud<
          pcl::PointCloud<PointType> > >(shape);
12    inputCloudData = inputCloudContainer->data;
13
14    return inputCloudData;
15  }
16
17 };
```


Conclusion

- Robot applications require models for different aspects $\Rightarrow$ the **world model** is one of them
- *Different* existing world models for various domains $\Rightarrow$ RSG is shared world model
- RSG-DSL
  - Primitives: Nodes, function blocks
  - Relations: DAG structure
  - Transforms: Model-to-text (C++)
- Expresses
  - Prior knowledge: *scene setups*, known objects
  - Interfaces for function blocks
- Future works
  - Uncertainty, multi resolution support, triggers
  - Composition with other DSLs