Managing additional graphs in GroIMP

Tim Oberländer

Uni Göttingen - Ökoinformatik

Content

- Introduction
  - Motivation
  - Background / Project structure
- GraphObjects
  - Concept
  - Use cases
- SecGraph
  - Concept
  - Use cases

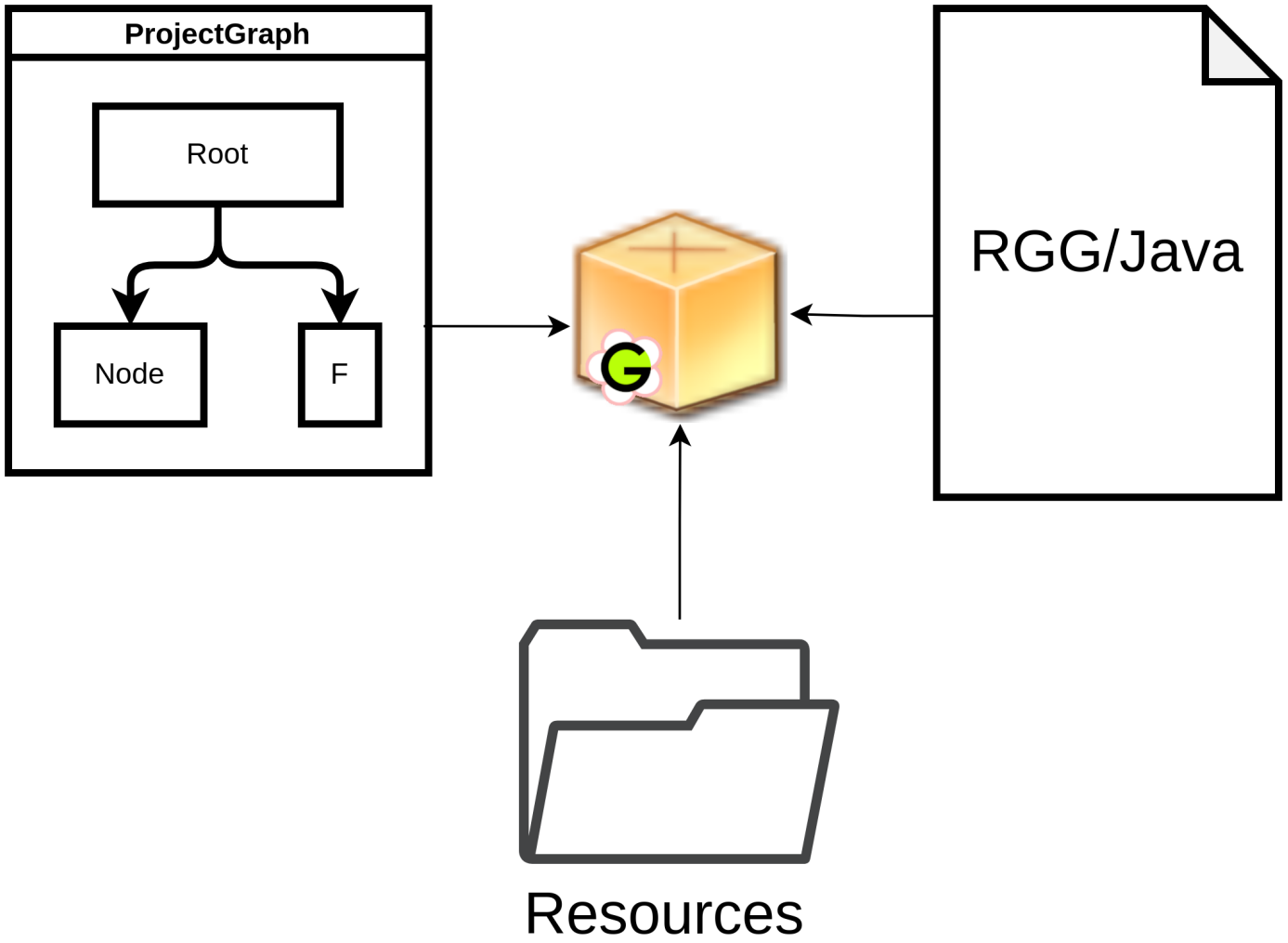
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

Introduction

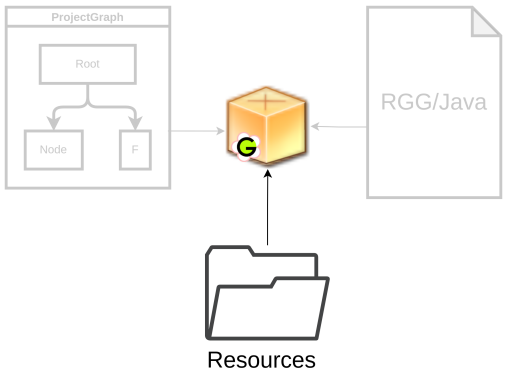
Motivation

- Part of my Ph.D.
  - Understanding rules as graphs
- Integrate additional graphs in GroIMP
- Object-oriented approach to graph management

Background - Project



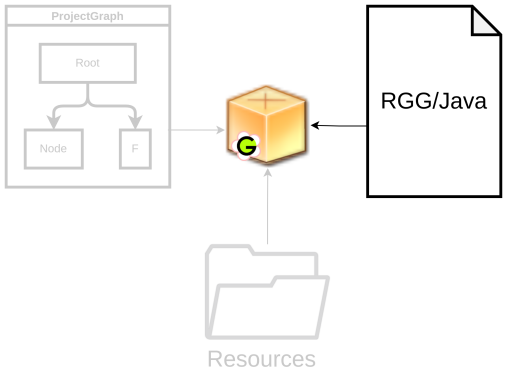
Background - Project



Resources

- All additional information:
  - Shaders, Images, Datasets, Functions
- Independent from Simulation & Compilation
- Serialized in either files or meta-graph nodes

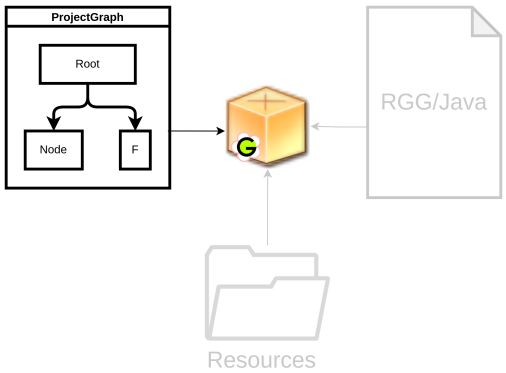
Background - Project



(Compiled) Code

- Module definitions
  - Required to create the nodes in the graph
- XL Rules
  - Analyzing and manipulating the graph
- Referencing to resources
- Java calculation

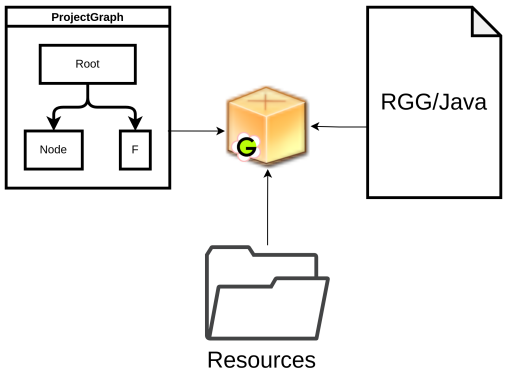
Background - Project



Project graph

- State of the simulation
- Construed out of nodes, edges, and modules
  - Requires compiled code of modules
- Each node has an id (persistence)
- Meta graph
  - Instances of the compiled classes
  - Smaller resources (eg. RGBAShader)

Background - Project



Registry

- Holds it all together
- Core structure of GroIMP
- Project registry inherits from root registry
- Only project-related part is stored in the .gs file

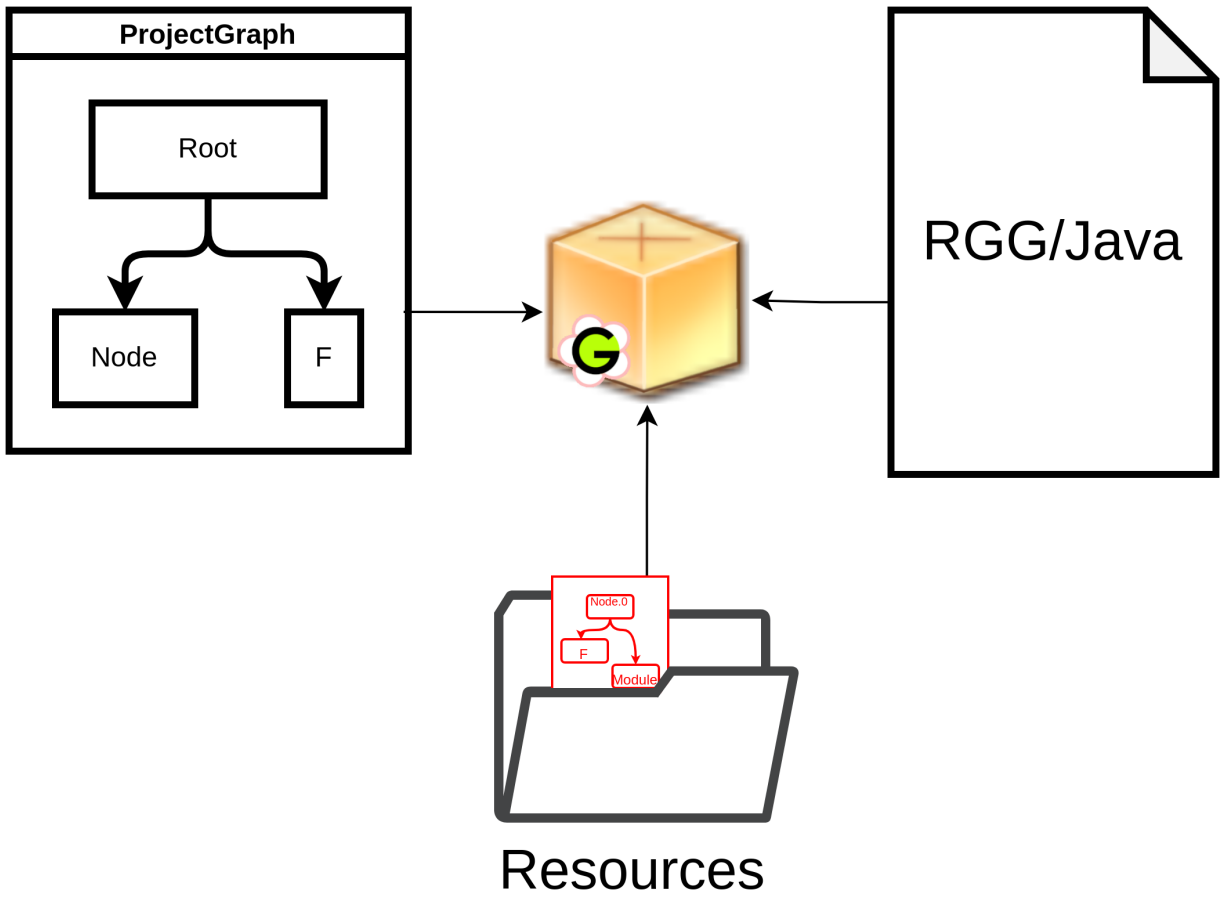
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

GraphObject

Idea

- Using graphs as resources
- Read-only
- Own GraphManager $\Rightarrow$ Own Persistence
  - Independent from project-graph
- Stored in original files
- Lazily loaded

Idea



Application

- Usage through the GraphObjectExplorer
- Import files similar to object/import
- Viewable in 2D and 3D
- Use the file in RGG with the GraphObjectRef class

```
public void load(){
    GraphObjectRef gr= new GraphObjectRef("myleaf.ply");
    [
        A ==> gr.cloneGraph(); // clones the graphObject
        B ==> gr; // creates an instantiation of the graphObject
    ]
}
```

Use case

Start simulation from measurement

- Deconstruction or further growth

```
protected void init (){
    GraphObjectRef gr = new GraphObjectRef("semantic_tree.qsm");
    [Axiom ==> gr.cloneGraph();]
}
public void fakeGrow()[
    f:F,(empty((*f --> Node*)))==> f F(f.length*0.9,f.diameter*0.9);
]
```

Graph structures as organs

- As static graph extensions

```
module Branch(float len) extends M(len) ==> GraphObjectRef("branch.obj");
```

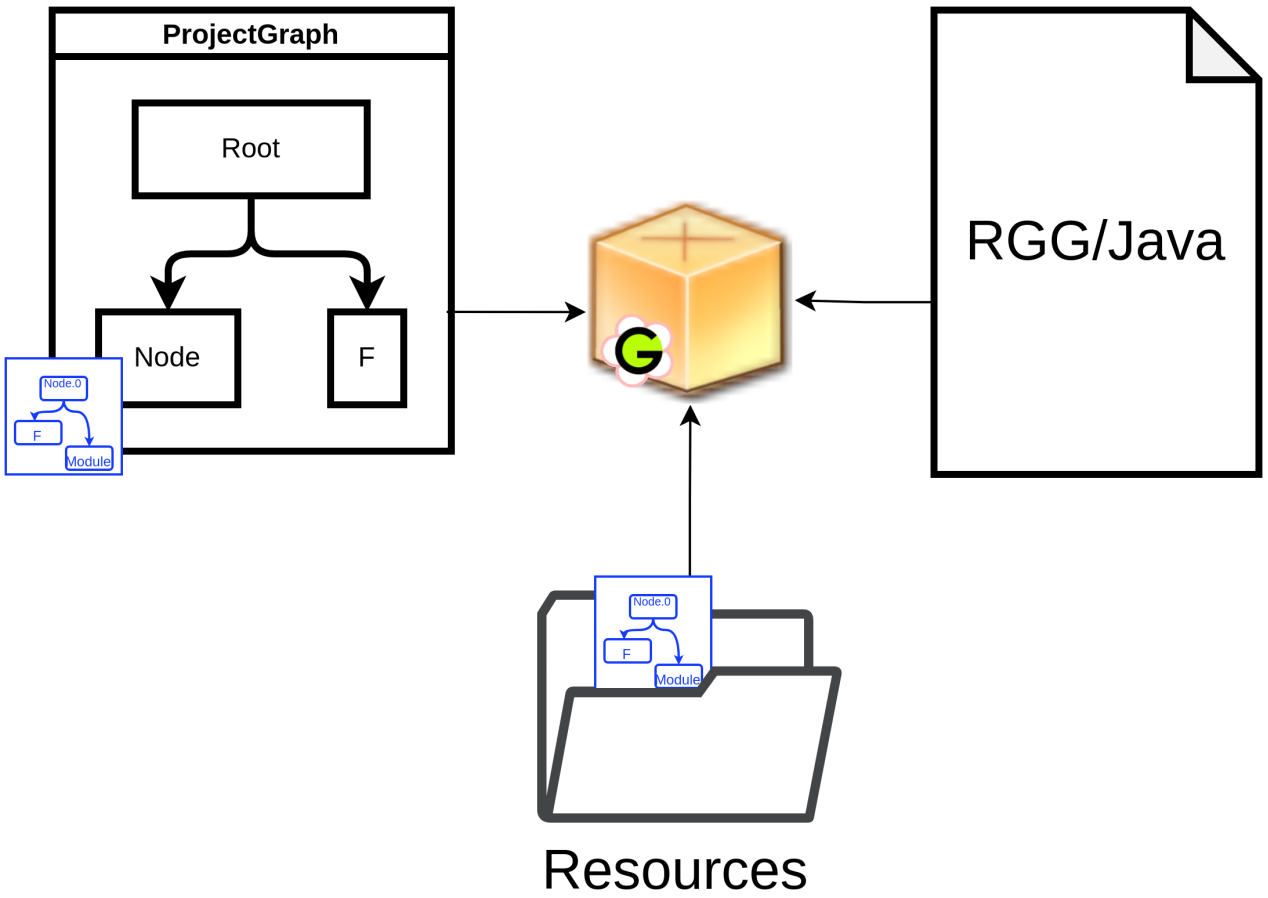
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

SecGraph

Idea

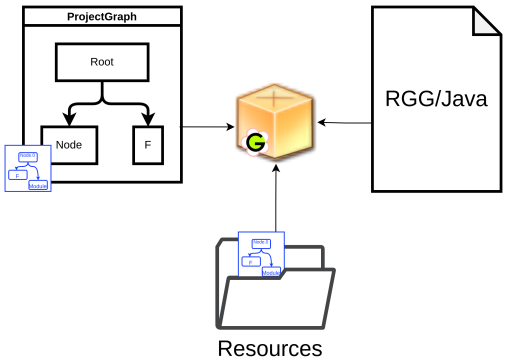
- Using an updatable graph as a resource or an attribute
- Extends GraphObject
- Stored in XML format
- Can be imported, created, or extracted/cloned
- Can be manipulated through RGG/XL or the GUI

Idea


ProjectGraph
Root
Node
F
Node.0
F
Module
RGG/Java
Node.0
F
Module
Resources

Application - Types

- As reference (SecGraphRef)
  - Shown in the SecGraphExplorer
  - Independent from compilation
- As a variable (SecGraphImpl)
  - Only stored if the variable is an attribute of a Node
  - Stored in a hidden directory



Application - Create SecGraph

- Import (similar to GraphObject)

Create on the fly

- Using the same syntax as Instantiations

```
SecGraph eins = new SecGraphRef("eins");
eins ==> F [RL(90)F][RL(-90)for(int i=0;i<10;i++)(Box(1))];
```

Copy an existing graph structure

- Clone the sub-graph below the provided node

```
SecGraph sg1 = new SecGraphImpl(workbench(),first((*RGGRoot*)));
```

- Clone a SecGraph/GraphObject in another SecGraph

```
SecGraphRef zwei = new SecGraphRef("zwei",sg1);
```

Application - Change SecGraph

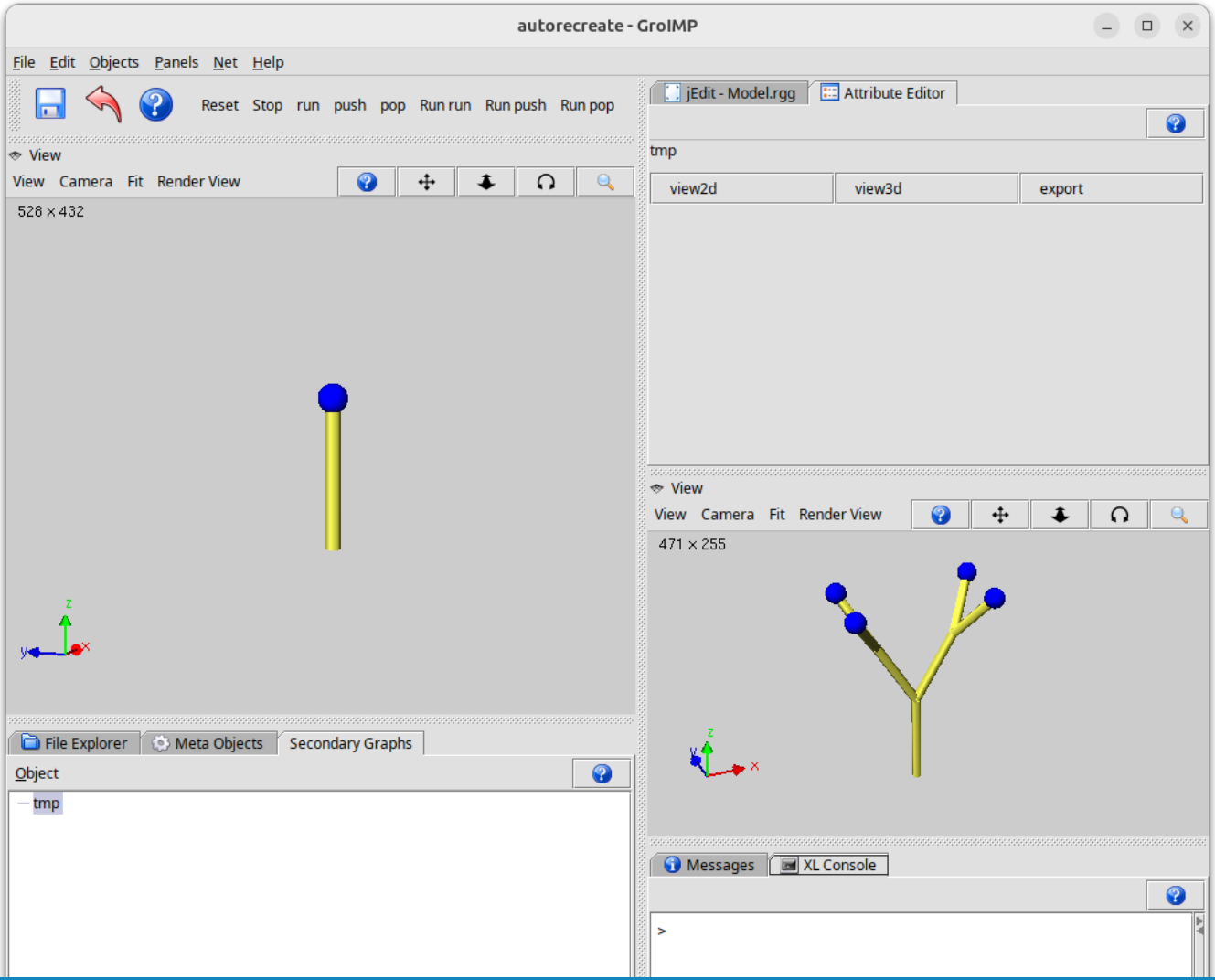
- A SecGraph object can be turned into the current used RGGGraph
- Apply whole RGG function on a SecGraph: `sg1.apply("run");`
- Set and release the current lock by hand:

```
public void changeF(){
    sg1.setCurrent();
        [F ==> F RL(30) F;] // on SecGraph sg1
        println((*F*));
    sg1.releaseCurrent();
    F ==> M; // on main graph
}
```

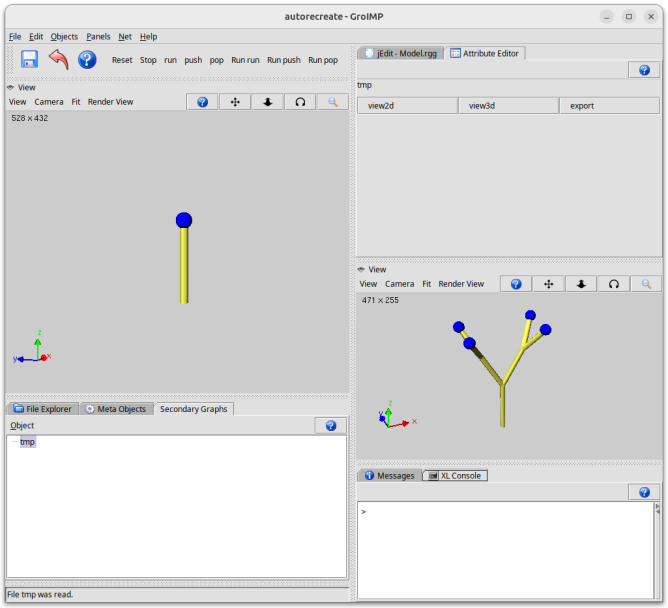
Application - Change SecGraph II

```
module A(float len) extends Sphere(0.1)
{
    SecGraphImpl sgi= new SecGraphImpl(workbench());
    {
        setShader(GREEN);
        sgi ==>F(0.2) RV(0) Parallelogram(1,1);
    }
} ==> sgi;

...

public void hanging()[
    a:A ::> {
        a.sgi.setCurrent();
            [RV(x) ==> RV(x+0.01*a.len);]
        a.sgi.releaseCurrent();
    }
    {repaintView3D();}
]
```

Application - Change SecGraph III



Application - Change SecGraph III




```
import parameters.*;
import de.grogra.graph.object.sg.impl.*;

module A(float len) extends Sphere(0.1){
 {setShader(BLUE);}
}
protected void init ()[
 Axiom ==> A(parameters.length);
]

public void run ()[
 A(x) ==> F(x) [RU(30) RH(90) A(x*0.8)]
               [RU(-30) RH(90) A(x*0.8)];
]

public void push() {
 SecGraphRef tmp = new SecGraphRef("tmp",first((*RGGRoot*)));
 tmp.setAutoRecreate(true);
}
public void pop() {
 SecGraphRef    tmp = new SecGraphRef("tmp");
 [A ==> tmp.cloneGraph();]
}
```

Use case

- Store simulation results of multiple runs
- Node-based multi-scale structures
- Brake out graphs for rewriting