OOPSMP code structure

presented by Ioan Sucan, Rice University

Previous presentation - OOPSMP for the user:

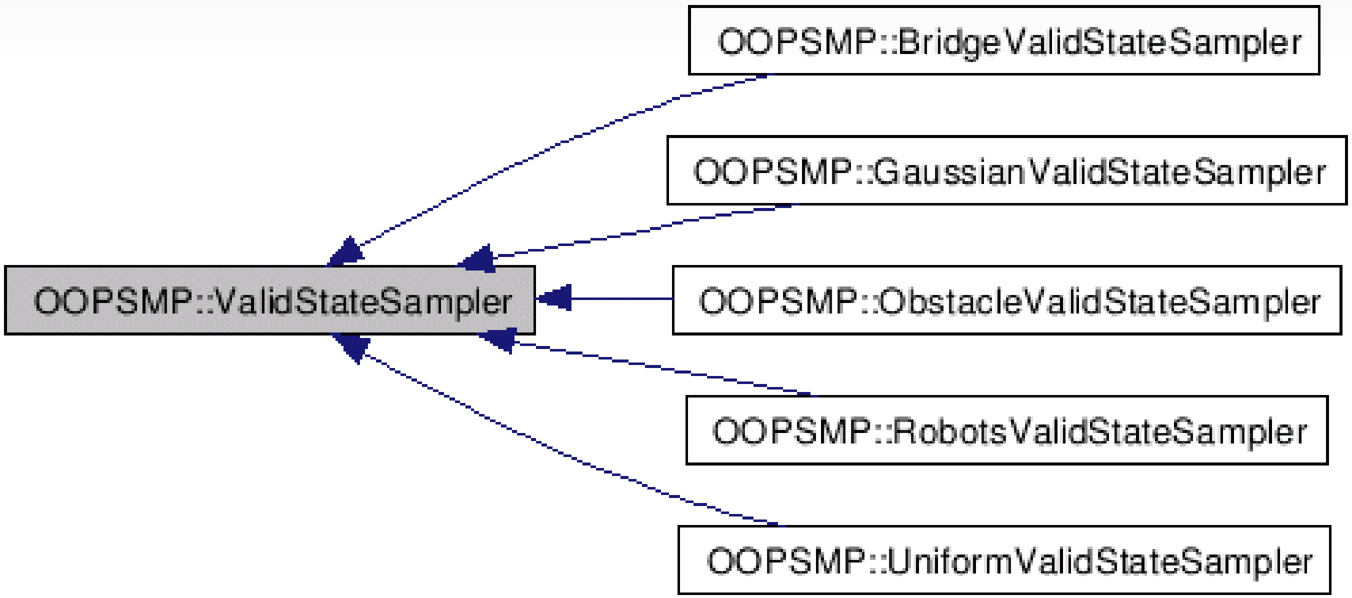
- General purpose utilities and data structures
- Implementations of popular motion planning algorithms
- Solution of problems involving one or multiple robots

OOPSMP for the developer

- First, code structure
- Plug-and-play functionality

Basic organization of OOPSMP:

- Set of classes compiled in dynamically linked libraries
- Semantically related classes are grouped in the same library
- The typical paradigm
  - One abstract base class
  - Different implementations of the base class are available



Basic organization of OOPSMP:

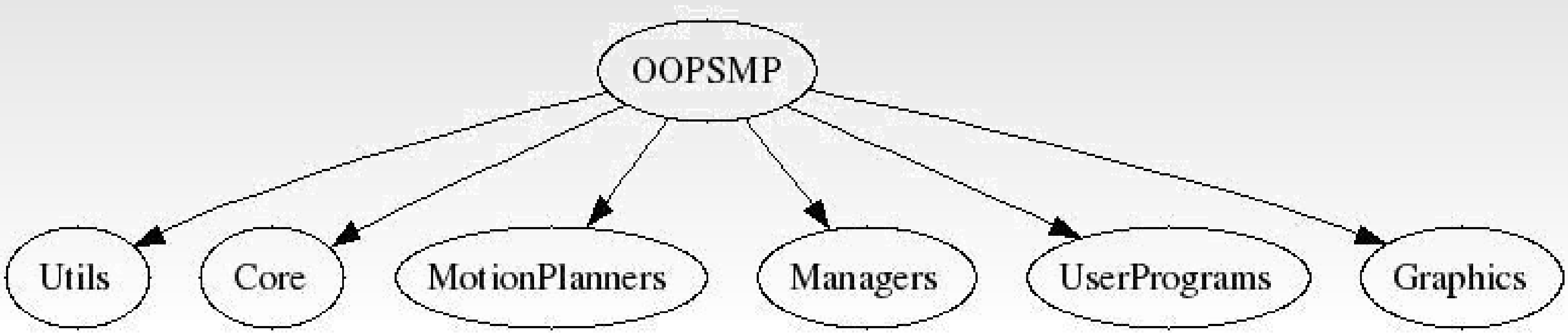
- One executable
  - Loads the needed libraries
  - Parses XML input file
  - Instantiates the user selected class implementations
  - Calls functions from the instantiated classes

```
<factory instance="Example2">               # Example2 *ex2 = new Example2();

  <call fn="fn1">                           #
  </call>                                   # ex2->fn1();

  <call fn="fn2">                           #
    <arg type="double">1.0</arg>            #
    <arg type="int">4</arg>                 #
  </call>                                   # ex2->fn2(1.0, 4);

</factory>
```

Solving a motion planning problem:

- Pick a set of components: the class implementations
  - Sampler, local planner, planning algorithm
- Set the planning queries and attempt to solve the problem
- The problem may not be necessarily solved
  - Can a different sampler do better?
  - Can a different planning algorithm do better?
- Change some of the selected class implementations
  - What are the available ones?
  - Overview of the available implementations follows!

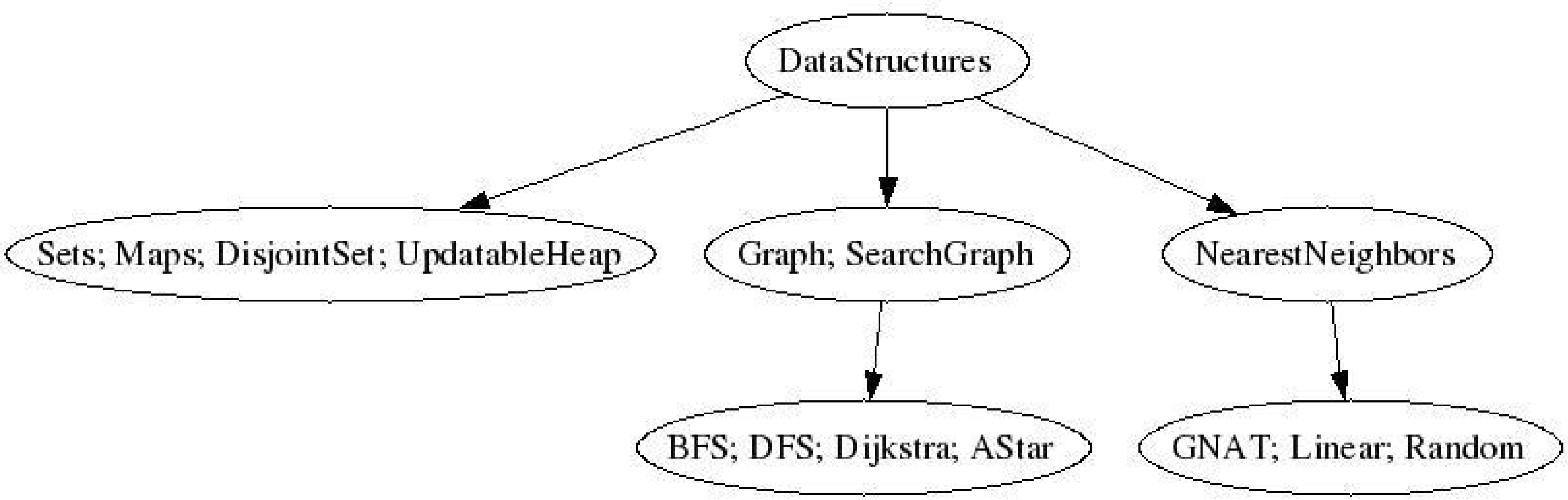
Top-level organization:



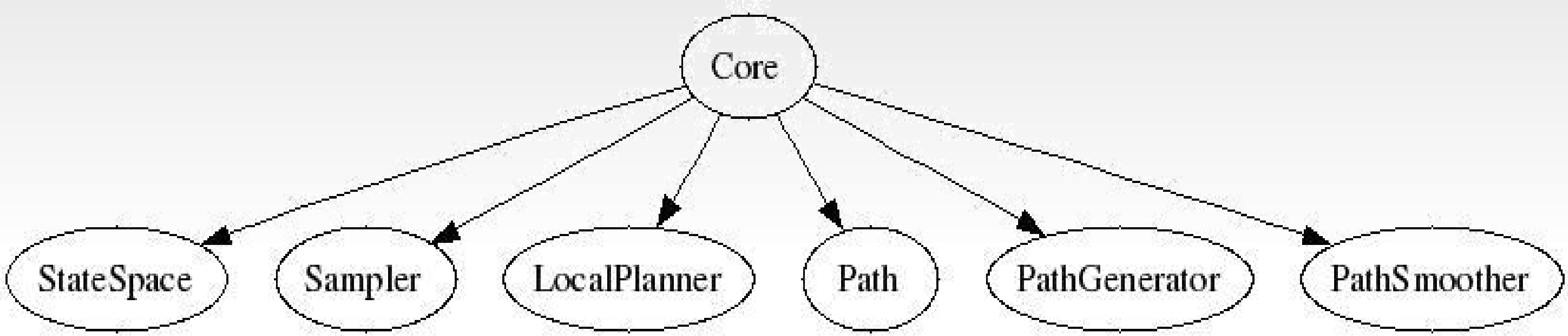
- The directory structure follows the code organization
- Look at components one by one

Utilities:

- Random number generation
- Topology representation: SE(2), SE(3), ...
- Numerical integration: Runge-Kutta, ...
- Search algorithms
- Data structures
- ...



Core:

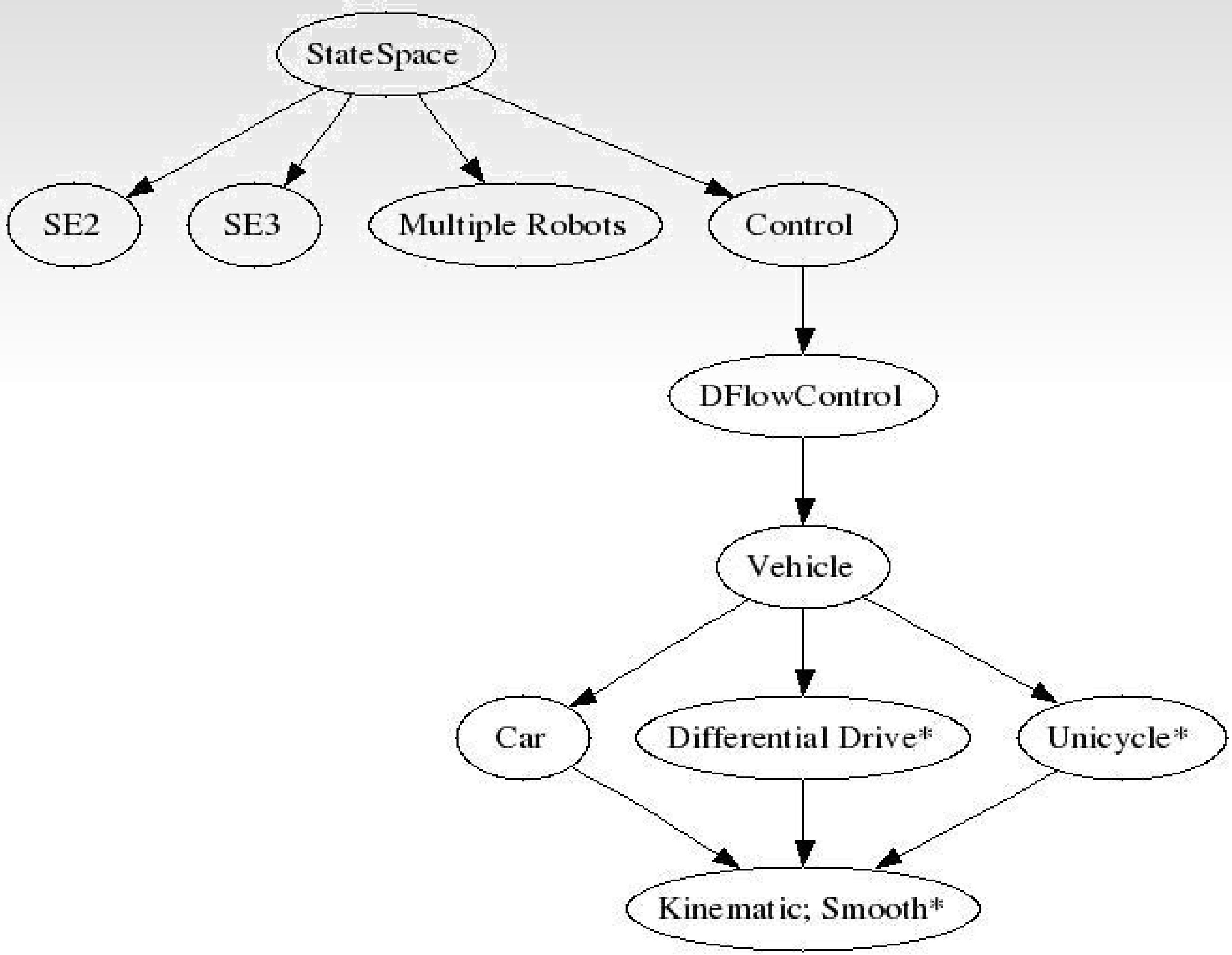


- Essential set of components for motion planning

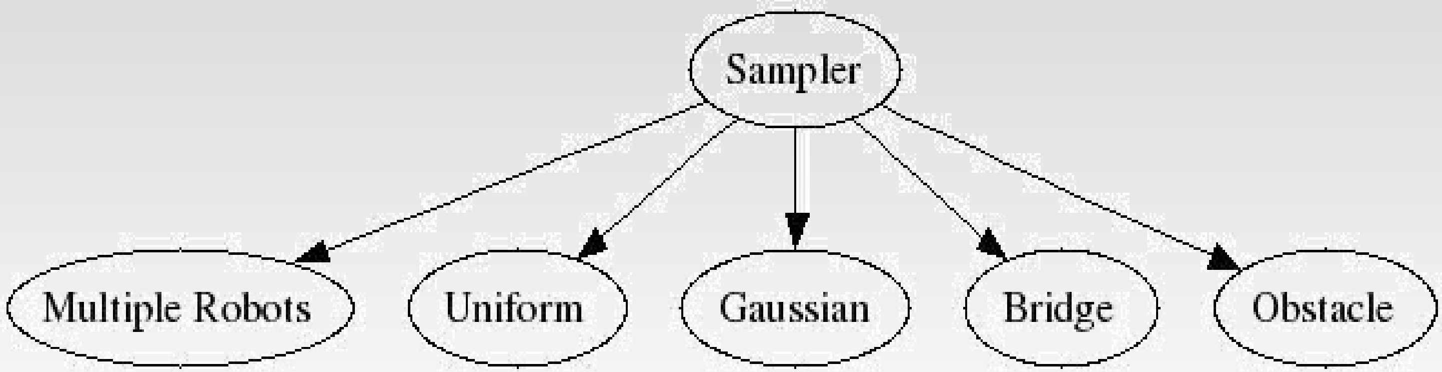
THIS SLIDE IS DISABLED
What is available:

- Utilities
    - Mathematics
        - ODE solvers
        - Topology representation: SO(2), SO(3), SE(2), SE(3)
        - Matrix operations
    - Geometry
        - Collision detection for 2D and 3D
    - Data structures
        - Heaps, hashes, graphs, nearest neighbors
- Motion planners
    - Roadmap based:
        - PRM
    - Tree based:
        - RRT, EST, Bi-directional
- Graphics
    - Display environments, states, paths

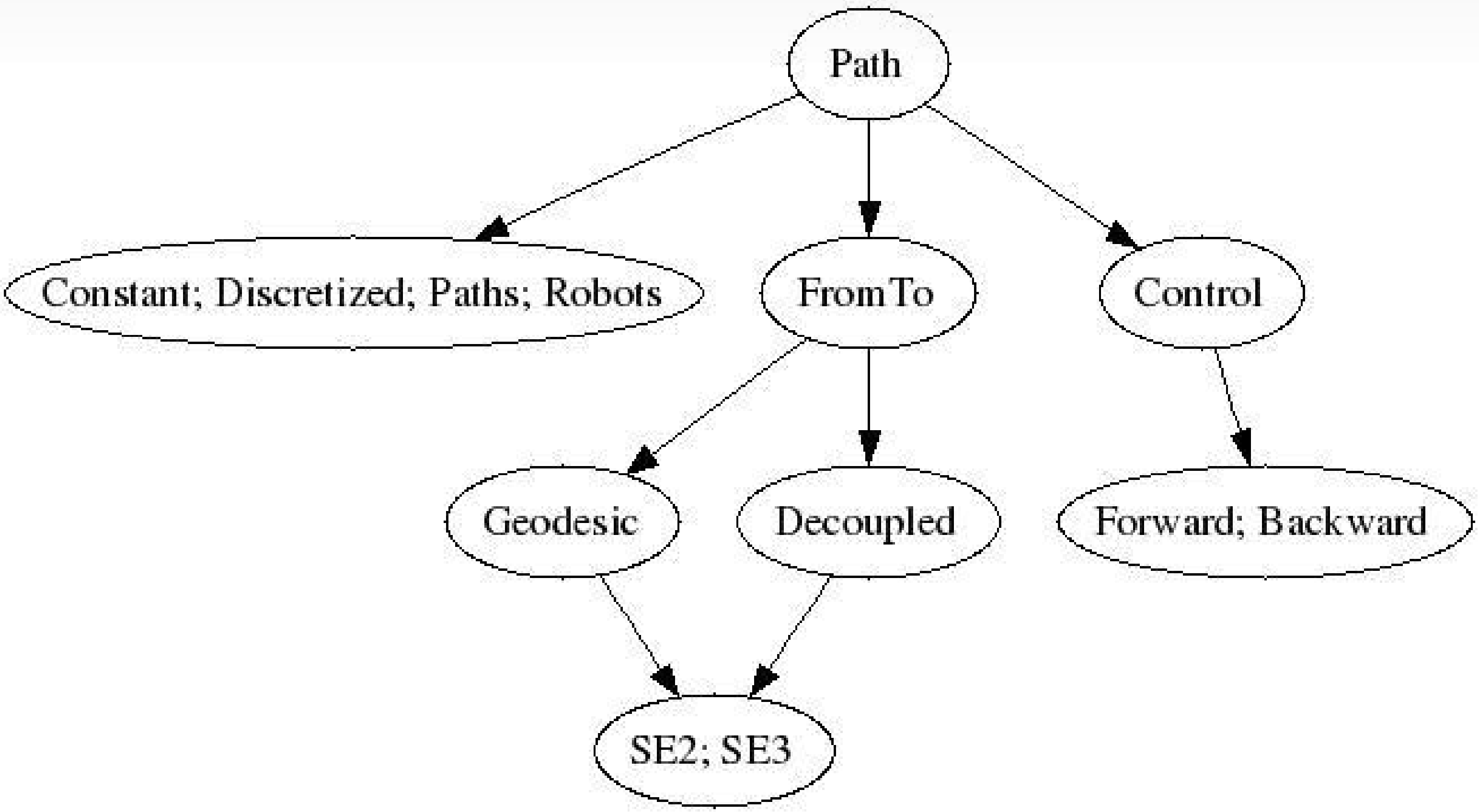
The state space (configuration space) definition:



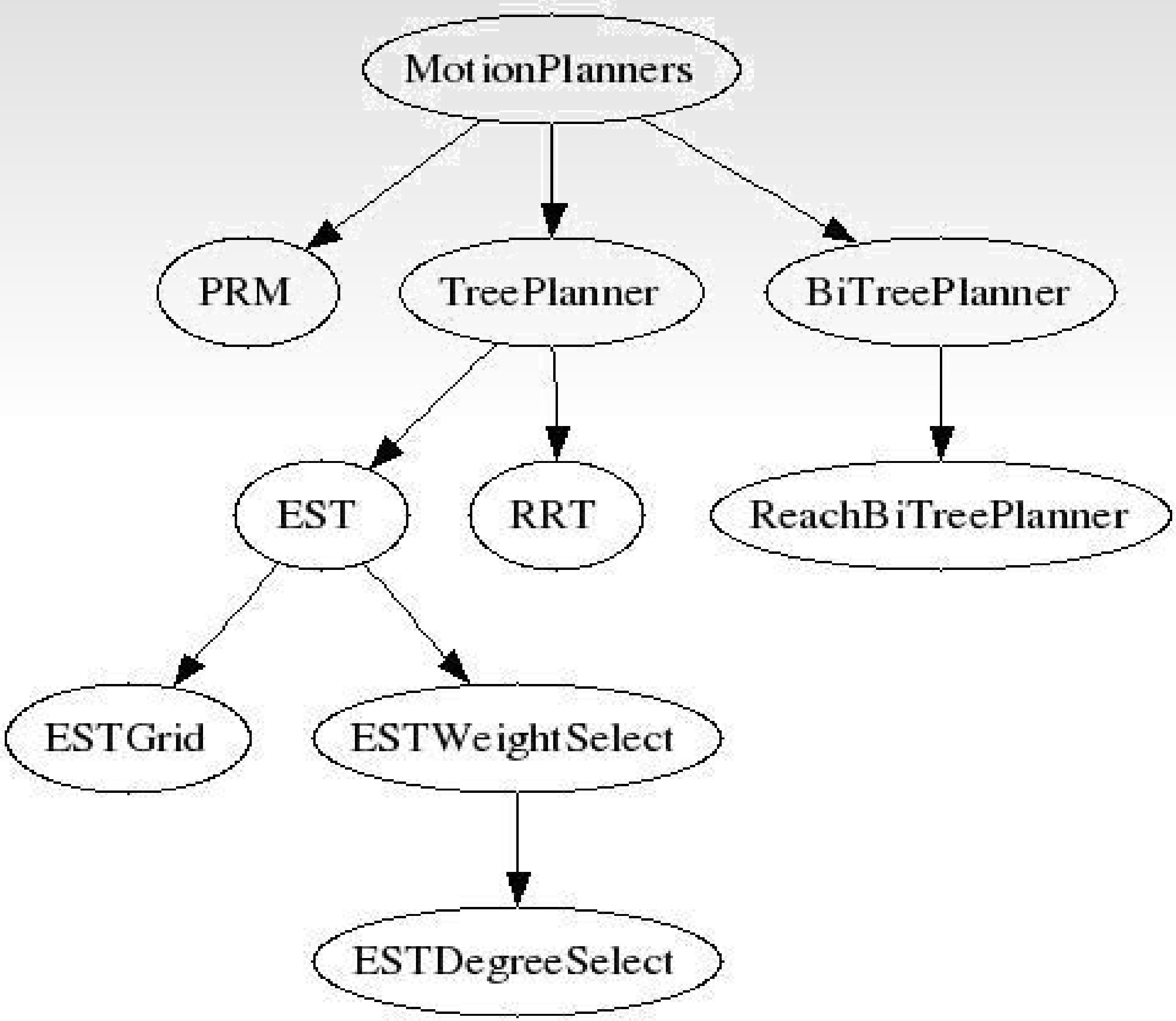
State space sampler:



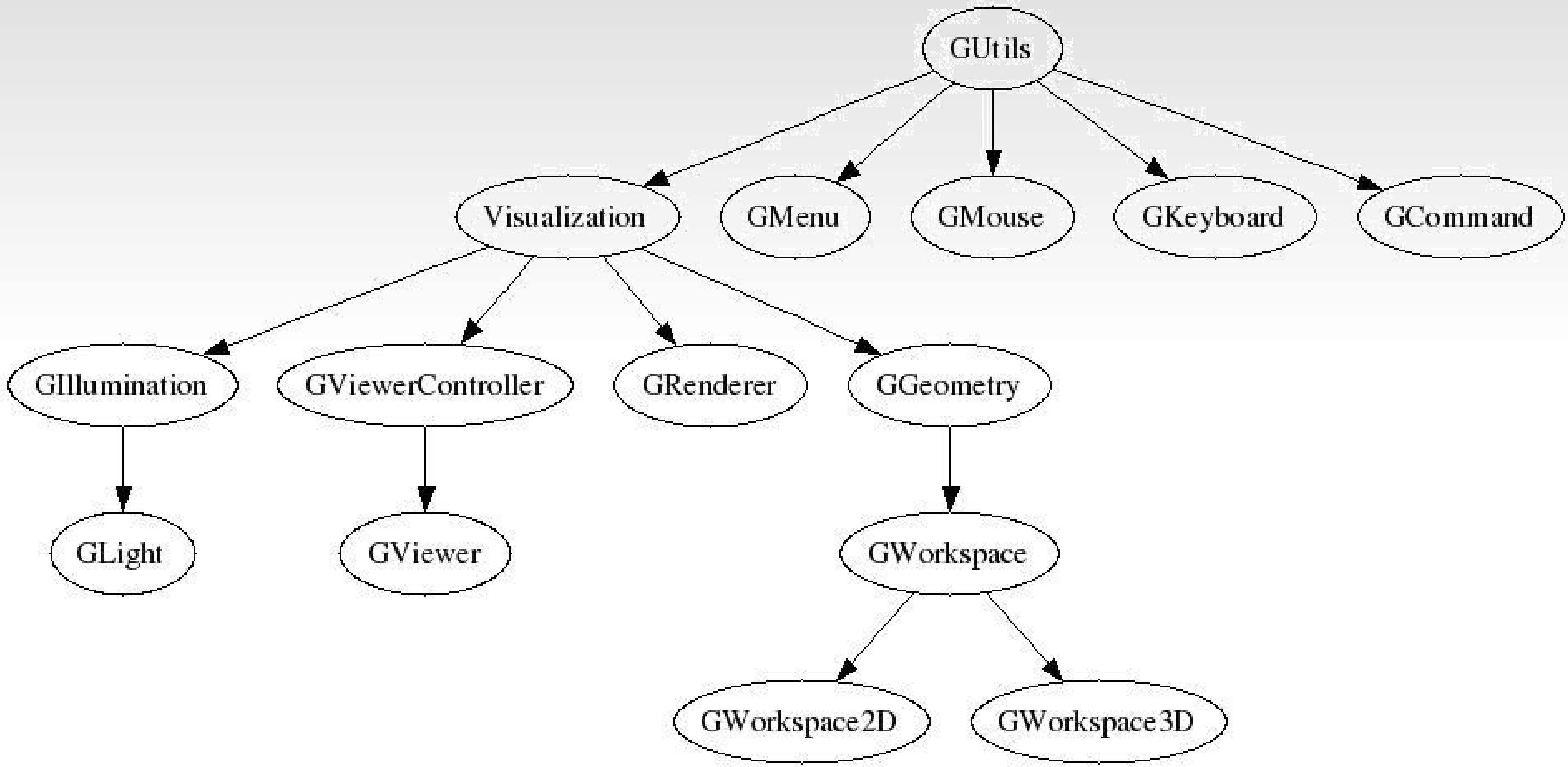
Paths:



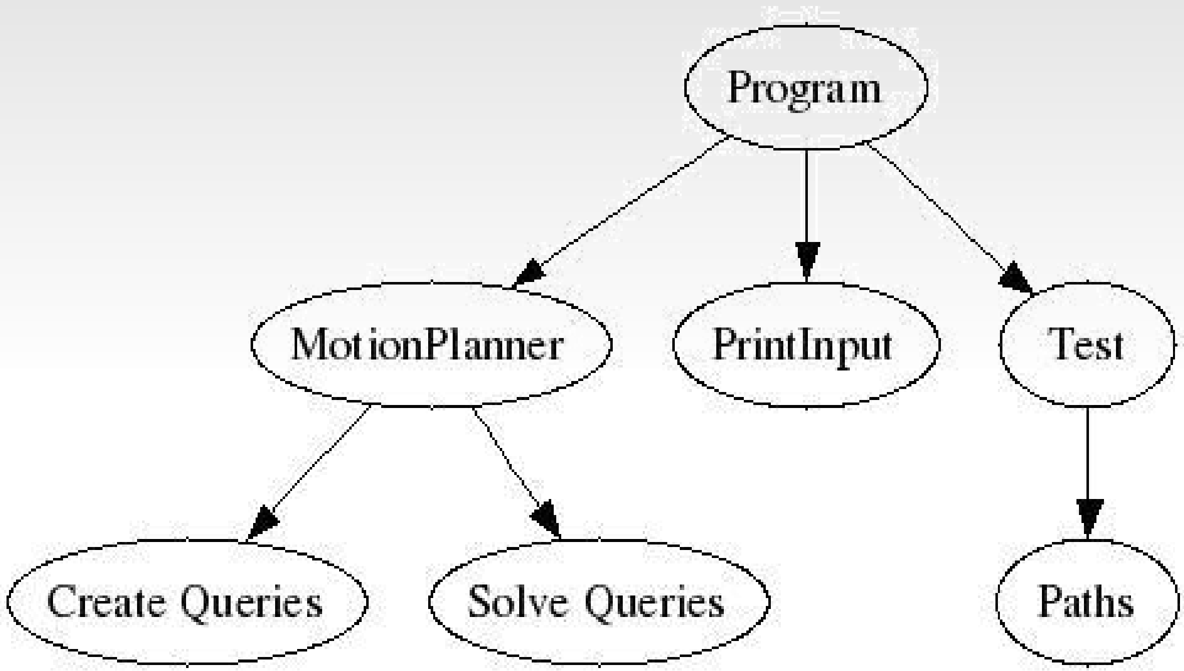
Sampling-based motion planners:



Graphics utilities:



User programs:



The **Program** class:

- Needs to implement the run() function
  - This does the useful computation

- There exist default implementations (user programs):
  - MotionPlannerProgram – solves a set of queries
  - GMotionPlannerProgram – solves a set of queries and provides a graphical interface
  - ...

- A special **Manager** class instantiates and executes a program

What needs to be instantiated to solve a motion planning problem:

- **CoreRobotData** – this is the component everything connects to
  - CoreRobotsData
- **Workspace** – this is where collision detection will be done
  - Workspace2D, Workspace3D
- **CollisionDetector** – the method for collision detection
  - PQPCollisionDetector2D, PQPCollisionDetector3D
- **StateSpace**
  - SE2StateSpace, SE3StateSpace, ControlStateSpace, etc
- **ValidStateSampler**
  - UniformValidStateSampler, ObstacleValidStateSampler, etc
- **PathGenerator**
  - SE2GeodesicPathGenerator, ControlPathGenerator, etc
- **LocalPlanner**
  - IncrementalLocalPlanner, SubdivisionLocalPlanner
- **MotionPlanner**
  - PRM, RRT, EST, etc
- **Queries** – define what problems to solve

Summary:

- A set of classes grouped into libraries
- Different functionality available
- Flexible to execute, no need to recompile code for testing various combinations
- Easy to extend
- Free for academic use