Extending OOPSMP functionality

presented by Ioan Sucan, Rice University

Physics simulation

- define a model of a robot and its environment
- use a physics-based simulator to compute robot actions if certain controls are applied
- this essentially provides a forward integration routine
- we can do motion planning with this
- examples: Vortex, ODE, PhysX, Bullet, etc.

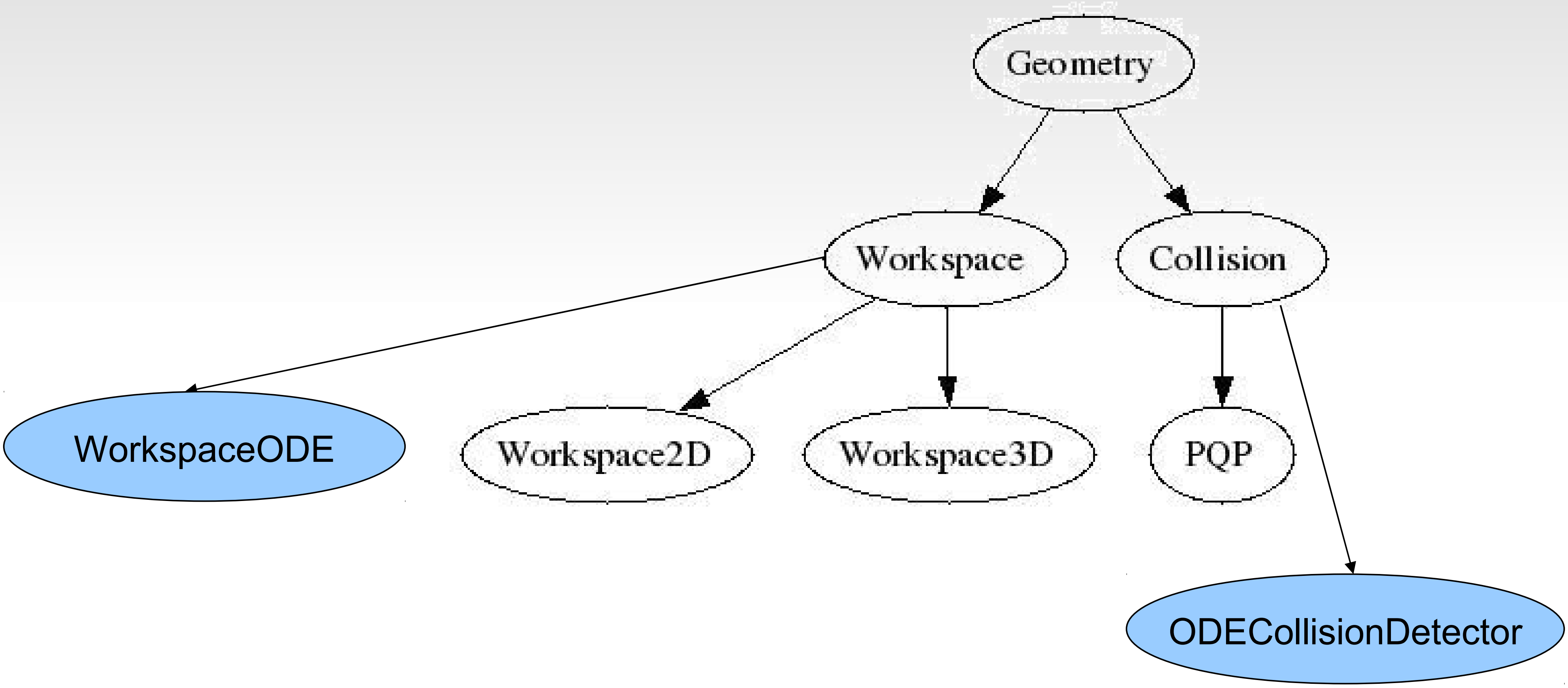
Some details

- Inherit from a base class and offer the same functionality with a different algorithm
- Declare a **Factory** class that can instantiate that class
- Register functions to export
- Create an XML file to load the new class

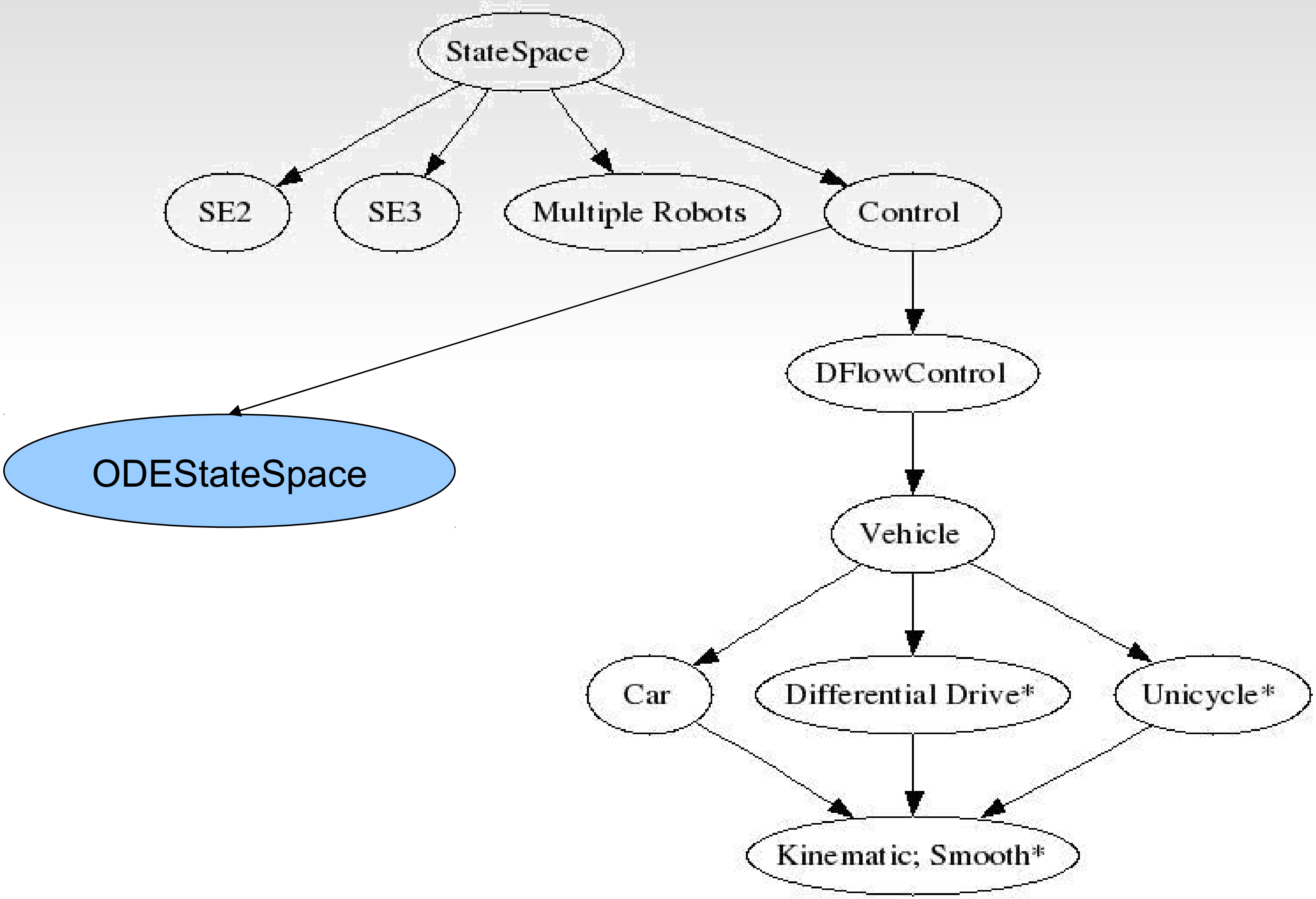
Example

- Adding physics-based simulation
  - Motion planning with the Open Dynamics Engine (ODE)
- Implement a new states space (ODEControlStateSpace), derived from ControlStateSpace
- Implement a new CollisionDetector (ODECollisionDetector)
- Implement a new Workspace (ODEWorkspace)

New classes that need to be added to support ODE geometry representation:



New state space class:



Remember:

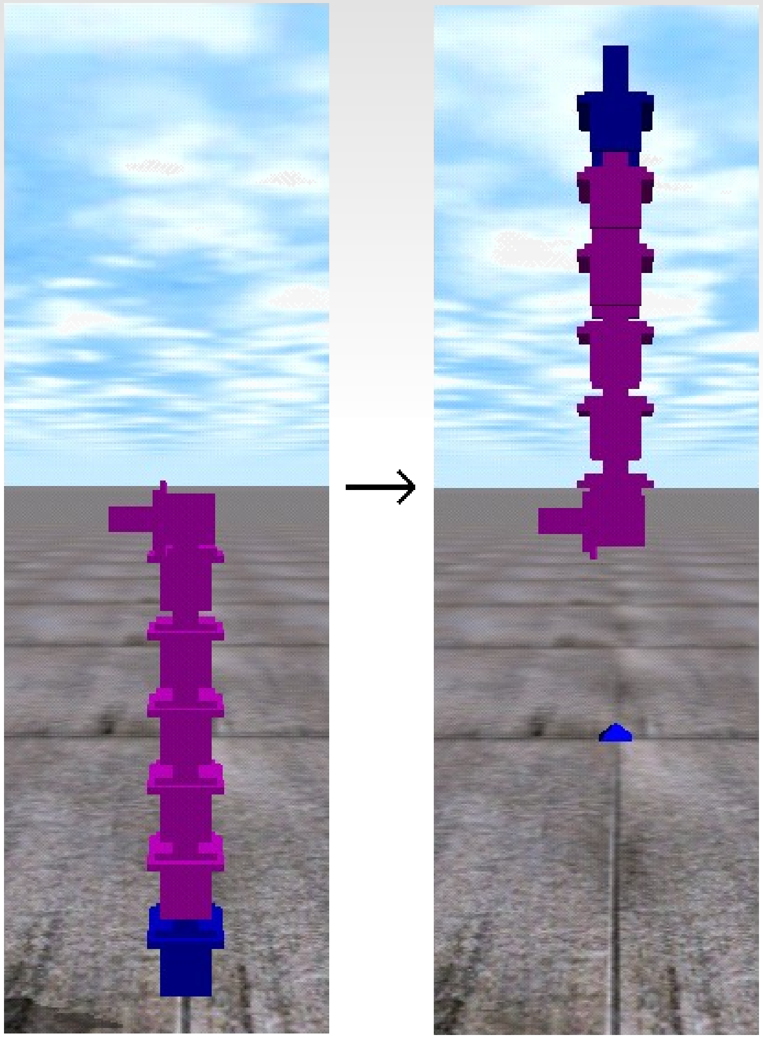
- When adding new classes, they need to be registered, so the user can load them from XML.
- Functions to be called externally, need to be registered as well

```
// header file of ClassName class
DeclareInstanceFactory(ClassName, BaseClassNameFactory);
//source file of ClassName class
BeginImplementInstanceFactory(ClassName, BaseClassNameFactory);
RegisterFnFactory(fn1Name, fn1_arg_types);
RegisterFnFactory(fn2Name, fn2_arg_types);
...
EndImplementInstanceFactory(ClassName)
```

```
<factory instance="ODECollisionDetector">
</factory>
```

Why do we need this?

- run existing motion planners on problems with physics simulation
- example: the CKBot problem



Conclusions

- Adding complex new functionality is possible
- Code reuse is maximized
  - Existing components can use the newly added components
  - The same motion planners can be now used without change