Design and Implementation of the

ROBOTIC PLATFORM

Markus Loffler – Ph.D. Dissertation Defense

Outline

- **Introduction**
  - Previous Research
  - Course and Focus of This Research
- **Design and Implementation of the Robotic Platform**
  - Design Overview
  - Manipulator and Manipulator Related Classes
  - The Math Library
  - The Object Manager
  - The Concurrency Architecture
  - Control Implementation with QMotor
- **The User Interface**
  - Built-in GUI Framework
  - Applications
- **Programming Examples**
  - Generic Programming
  - Two Manipulators
- **Conclusions**

Introduction - Previous Research

Due to the limitations of proprietary robot control languages, most research investigates how to use a common programming language for robot control.

C Libraries:

- RCCL (Lloyd and Hayward, 1988)
- ARCL (Corke and Kirkham, 1992)

→ *Complex and not flexible due to procedural programming*

Object-Oriented Systems (using C++)

- RIPE (Miller and Lennox, 1991)
- OSCAR (C. Kapoor, 1996)
- ZERO++ (C. Pelich and F. Wahl, 1996)

→ *Complex due to the integration of inhomogeneous distributed hardware components and "full system architecture"*

No flexibility at the servo control level

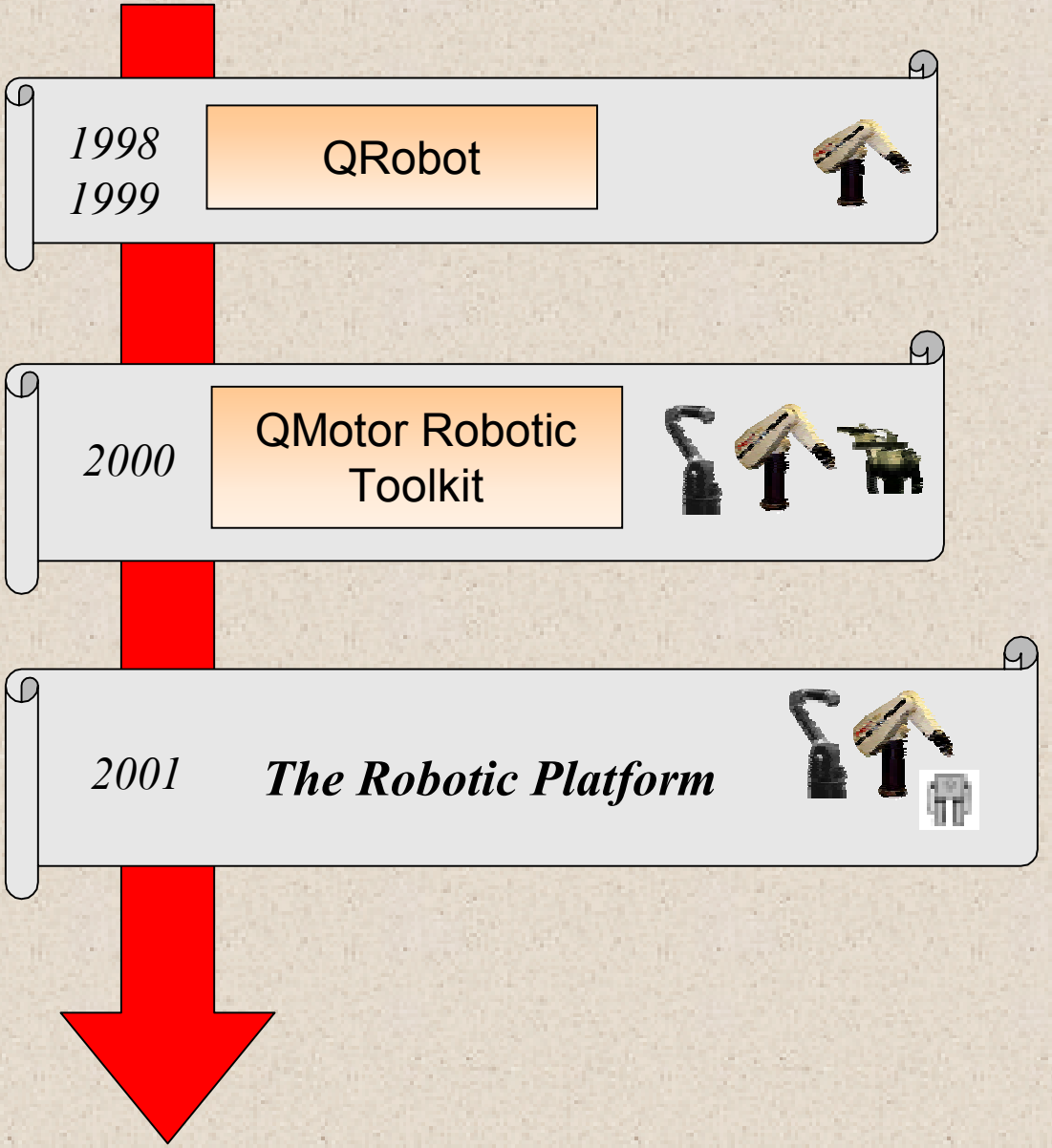
Extensibility claimed but never demonstrated

- QMotor Robotic Toolkit (Loffler, Dawson, Costescu, Zergeroglu, 2001)

→ *"Platform" Design*

→ *Only joint level functionality, no 3D graphical simulation*

Introduction - Course of this Research



Demonstrates that the PC is capable of integrating servo control, trajectory generation and graphical user interface for robot control

Introduces a homogeneous object-oriented platform and control tuning capabilities

This presentation...

Introduction - Focus of this Research

A New Level Of Flexibility and Extensibility

- *All* components are flexible and extensible (including servo control)
- Extensions without modification and knowledge of source code
- Extensions integrate with generic system components

A New Level Of Ease of Use

- Accelerated development of robot control programs
- Advanced testing, simulation and tuning capabilities
- Integration of the Graphical User Interface
- Easy installation and (re-)configuration
- Low complexity

Object-Oriented Architecture

One must consider run-time requirements!

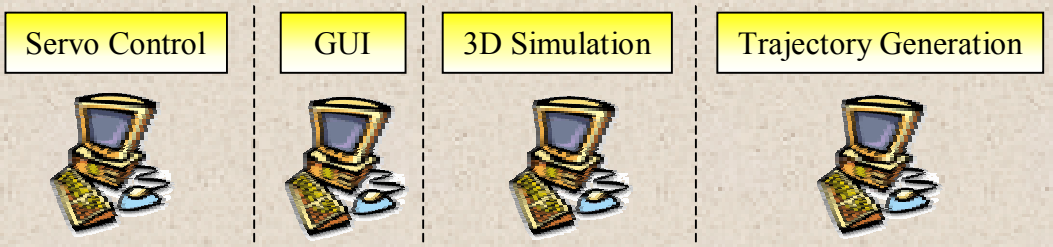
Introduction - Run-Time Requirements

For a robot control platform, it is essential to address the run-time architecture as well, especially the following issues:

- Concurrency
- Deterministic real-time
- Hardware interfacing
- Graphical user interface and accelerated 3D graphics
- Performance issues

Traditional Architecture:

- Multiple computing platforms
- Distributed, networked
- Proprietary hardware (e.g., DSP)



Defeats the Advantages of Object-Oriented Design And Increases Complexity

NEED HOMOGENEOUS DESIGN

"One-Box" Solution

Implemented on:

- A Single PC
- A Single Operating System
- A Single Programming Language

Introduction - The Three Design Goals

A New Level Of Flexibility and Extensibility

- *All* components are flexible and extensible (including servo control)
- Extensions without modification and knowledge of source code
- Extensions integrate with generic system components

A New Level Of Ease of Use

- Accelerated development of robot control programs
- Advanced testing, simulation and tuning capabilities
- Integration of the Graphical User Interface
- Easy installation and (re-)configuration
- Low complexity

Homogeneous Object-Oriented Architecture

"One-Box" Solution

Implemented on:

- A Single PC
- A Single Operating System
- A Single Programming Language

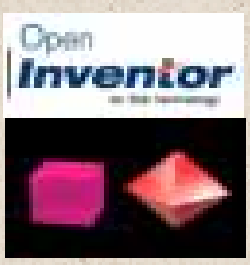
Design - The Basis: Powerful Tools and Technologies

PC Technology

- Processing Power
- Inexpensive Components
- Many Interface Board

The QNX Real-Time Platform

- Microkernel Real-Time Operating System
- Graphical User Interface with 3D Capabilities
- Free for Non-Commercial Use



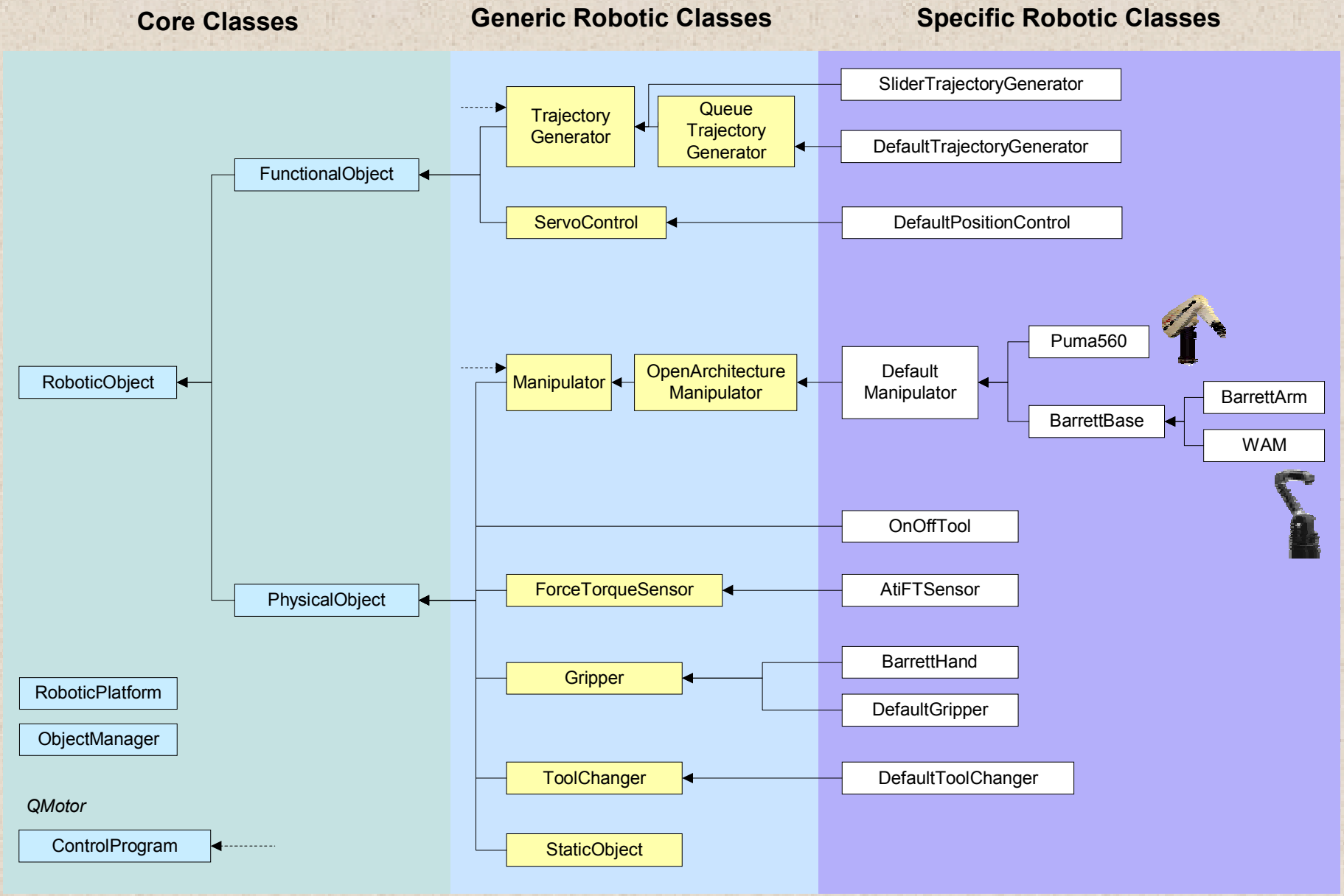
Open Inventor

- C++ Library for Creating 3D Graphics
- Supports VRML format
- Animation of 3D Graphics

QMotor

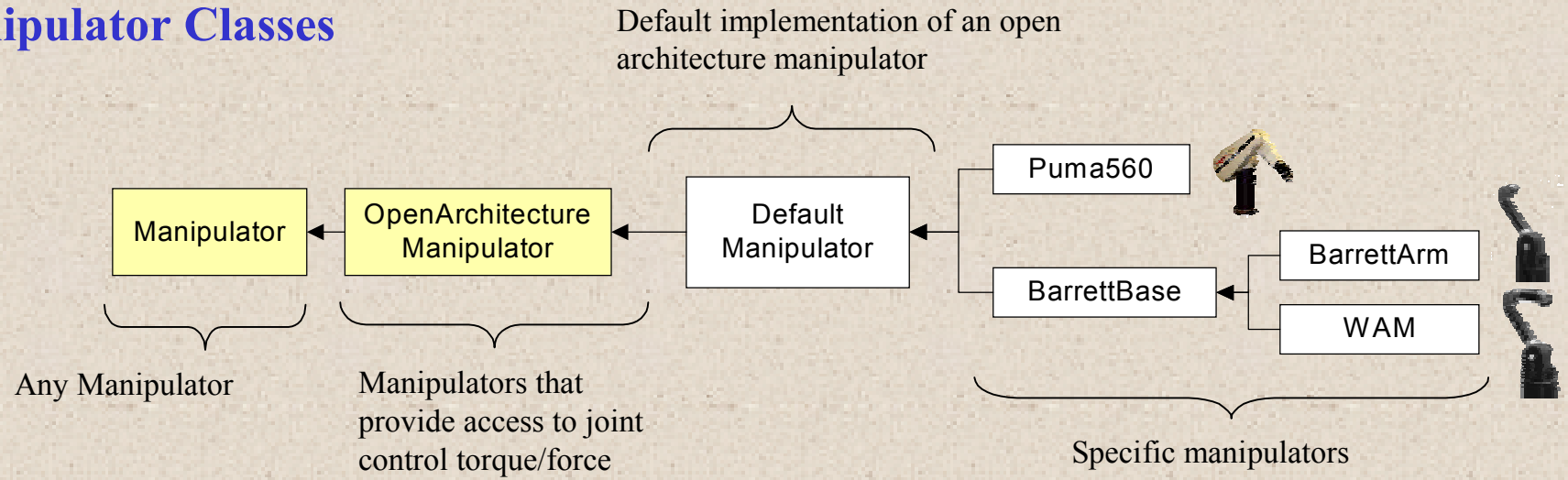
- Control Loop Implementation in C++
- Data Logging, Plotting and Control Tuning

Design – Design Overview / Class Hierarchy



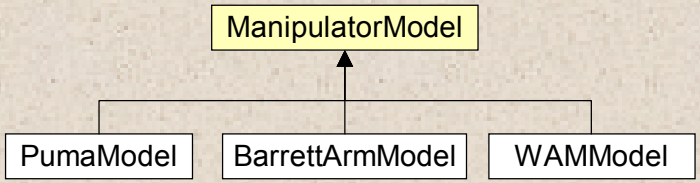
Design - Manipulator Classes

Manipulator Classes



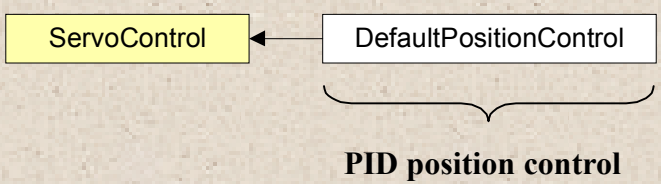
Manipulator Model Classes

- Kinematics
- Jacobian
- Dynamics
- Joint Limits
- Friction
- Gravity

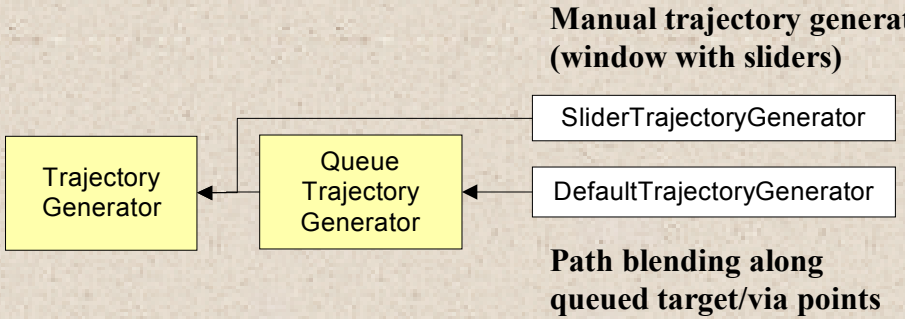


Design - Servo Control and Trajectory Generator Classes

Servo Control Classes

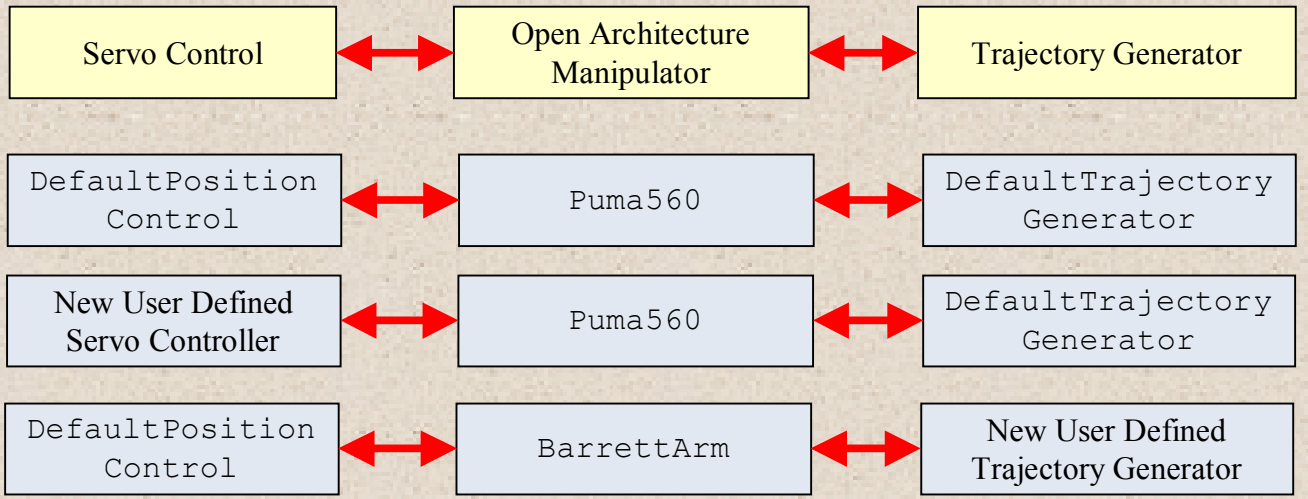


Trajectory Generator Classes



Object Relationships

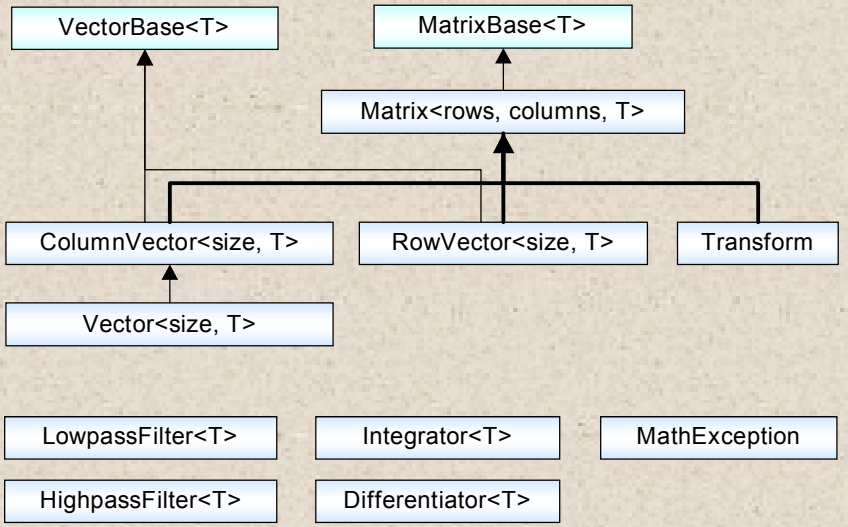
Three objects work together: The manipulator object, the servo control object, and the trajectory generator.



Design - The Math Library

A Set of C++ Classes for Matrices, Vectors, Transformations, Filters, Integration, and Differentiation

- Real-time capable
- Intuitive programming like in MATLAB
- Drastically simplifies programming of robotic applications
- Class **`Transform`** for kinematic calculations




```
Matrix<2,3> A;
ColumnVector<3> b;

A = 2, 5, -1,
    0, 1, 0.5;

b = -1, 0, 1;

cout << A * b;
```


```
Transform Z = translation(0, 0, 0.7);
Transform E = translation(0, 0, 0.1);
Transform W = translation(1, 0.2, 0.3)
              * xRotation(M_PI);
Transform P = translation(-0.5, 0, 0);

// Solve Z*T6*E == W*P
Transform T6;
T6 = inverse(Z) * W * P * inverse(E);
```

Design - An Example Robot Control Program

Example User Robot Control Program: Picking up an object

```
#include "RoboticPlatform.hpp"

void main(int argc, char *argv[])
{
  RoboticPlatform::init(argc, argv);          // Initialize the framework

  Puma560 robot("leader");                    // Create the manipulator

  DefaultGripper gripper;                     // Create the gripper
  gripper.setConnection(robot);

  DefaultTrajectoryGenerator<6> tragen;       // Create the trajectory
  robot.setTrajectoryGenerator(tragen);       // generator and connect it

  Vector<6> pos;
  pos = -1.57, 0.5, -2.7, 0, 0, 0;            // Define target position

  tragen.move(pos);                           // Move to the target
  tragen.stop(1)

  gripper.close();                            // Close gripper

  pos = 0, 1.57, -1.57, 0, 0, 0
  tragen.move(pos);                           // Pick up the object
}
```

Design - An Example Robot Control Program

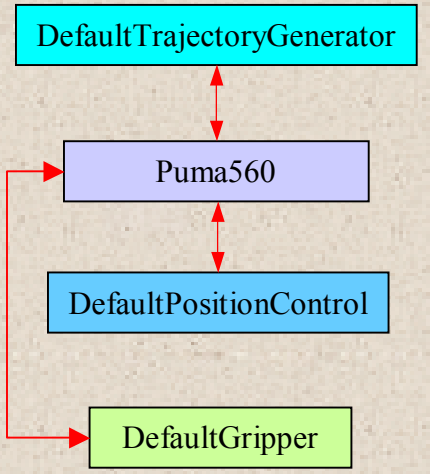
User Robot Control Program

```
#include "RoboticPlatform.hpp"

void main(int argc, char *argv[])
{
  RoboticPlatform::init(argc, argv);

  Puma560 robot("leader");

  DefaultGripper gripper;
  gripper.setConnection(robot);

  DefaultTrajectoryGenerator<6> tragen;
  robot.setTrajectoryGenerator(tragen);

  Vector<6> pos;

  pos = -1.57, 0.5, -2.7, 0, 0, 0;
  tragen.move(pos);
  tragen.stop(1)

  gripper.close();

  pos = 0, 1.57, -1.57, 0, 0, 0;
  tragen.move(pos);
}
```

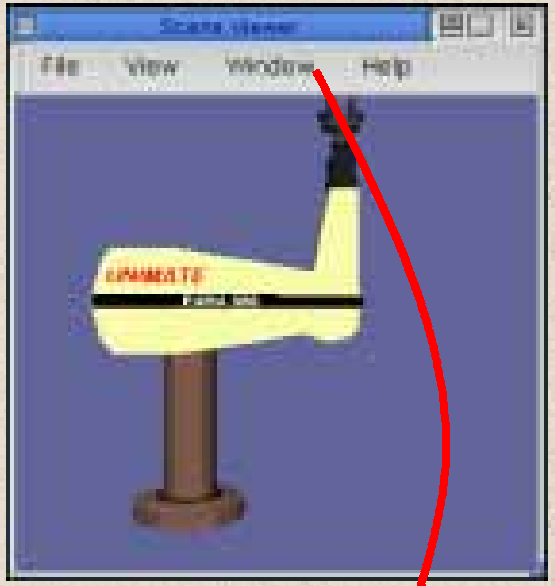
Global Configuration File

```
[leader]
class Puma560
position -1 0 0
simulationMode off

[gripper]
class DefaultGripper
```

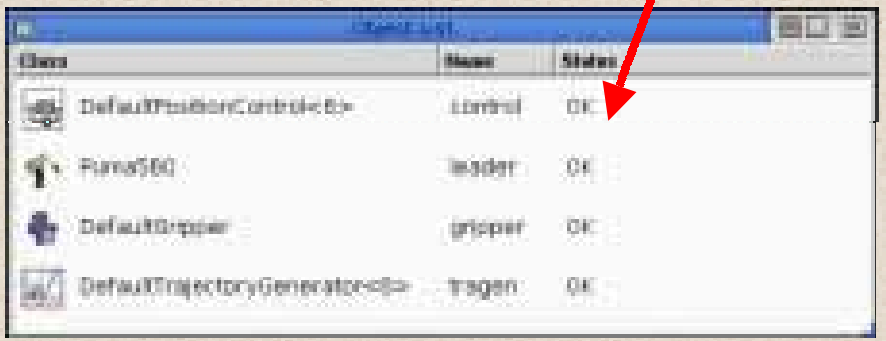
The Object Manager



The Graphical User Interface



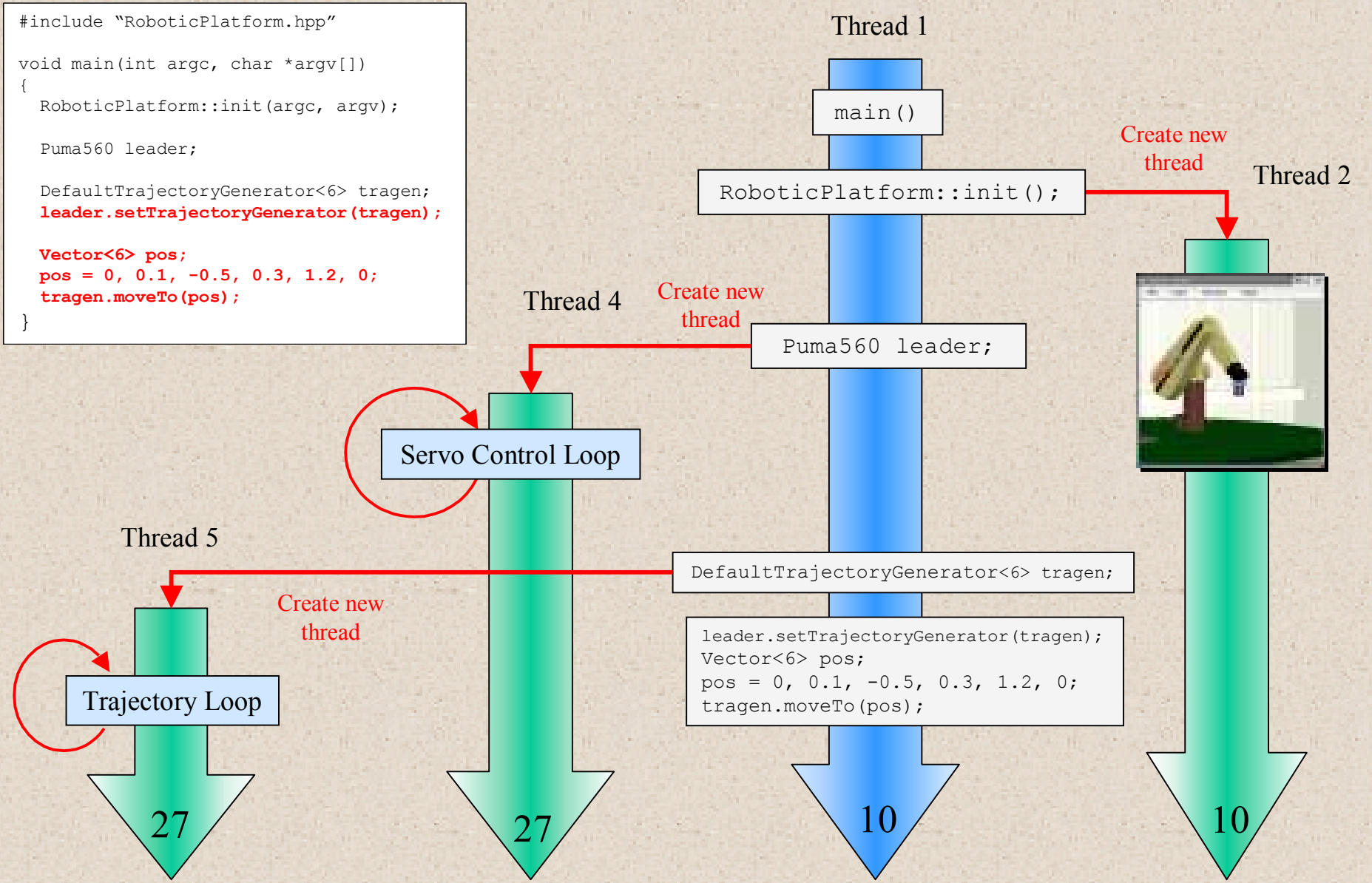
Scene Viewer



Object List Window

Design - Concurrency Architecture

```
#include "RoboticPlatform.hpp"

void main(int argc, char *argv[])
{
  RoboticPlatform::init(argc, argv);

  Puma560 leader;

  DefaultTrajectoryGenerator<6> tragen;
  leader.setTrajectoryGenerator(tragen);

  Vector<6> pos;
  pos = 0, 0.1, -0.5, 0.3, 1.2, 0;
  tragen.moveTo(pos);
}
```



Design - Control Loop Implementation with QMotor

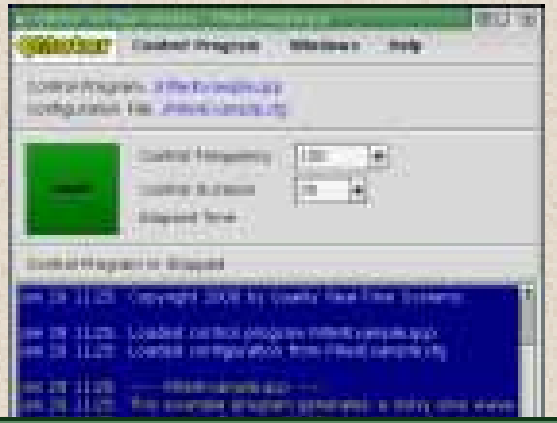
The QMotor Framework

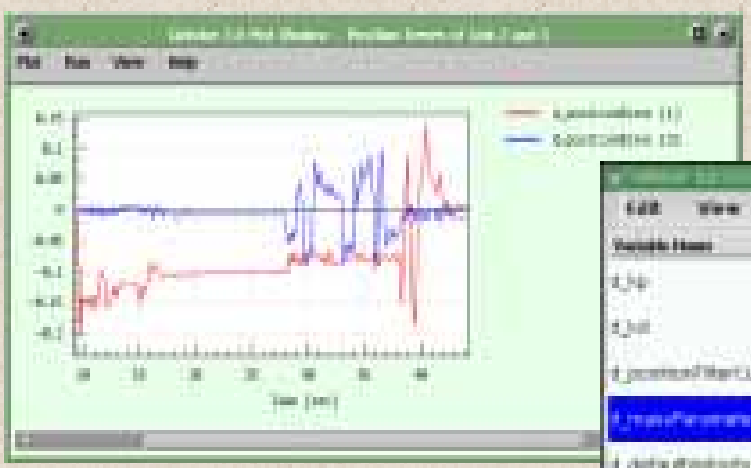
- Provides accurate execution of the real-time control loops
- Classes that require a control loop are derived from the `ControlProgram` class

The QMotor Graphical User Interface

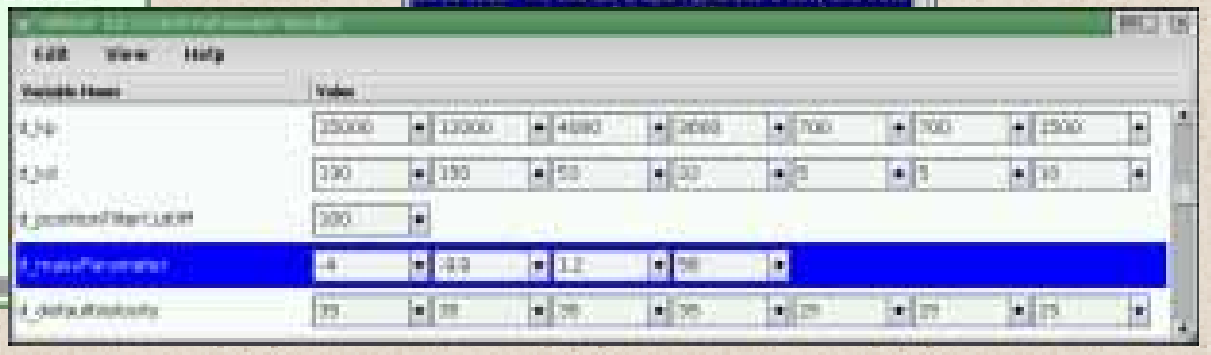
- Provides an intuitive user interface
- Provides flexible real-time data plotting
- Provides control tuning

Main Window



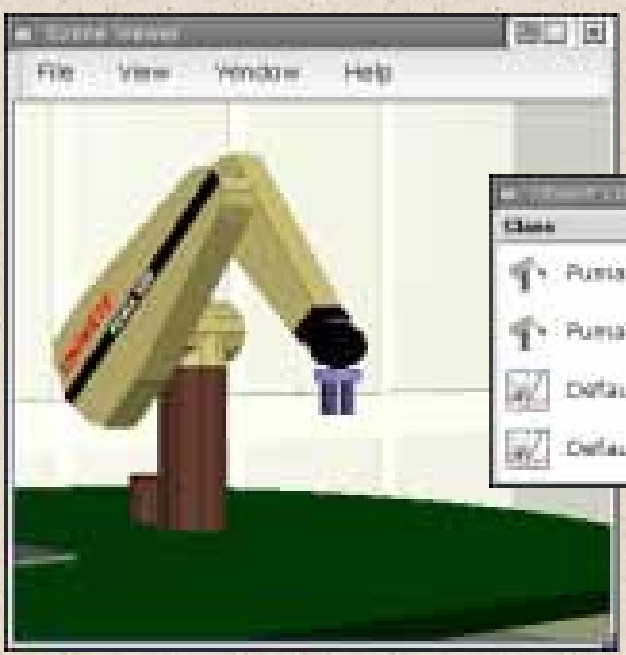


Plot Window



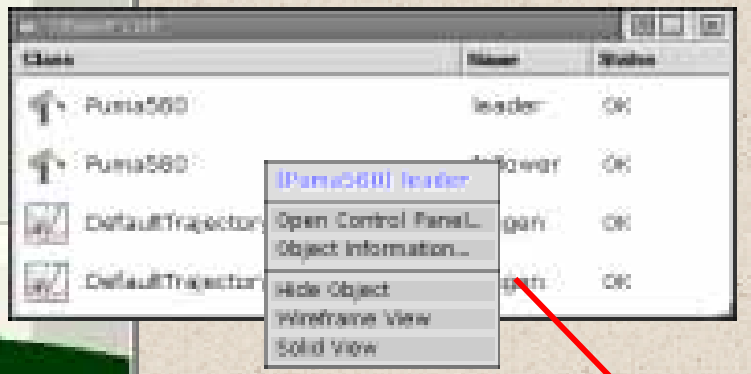
Control Parameter Window

User Interface - Built-in GUI Framework

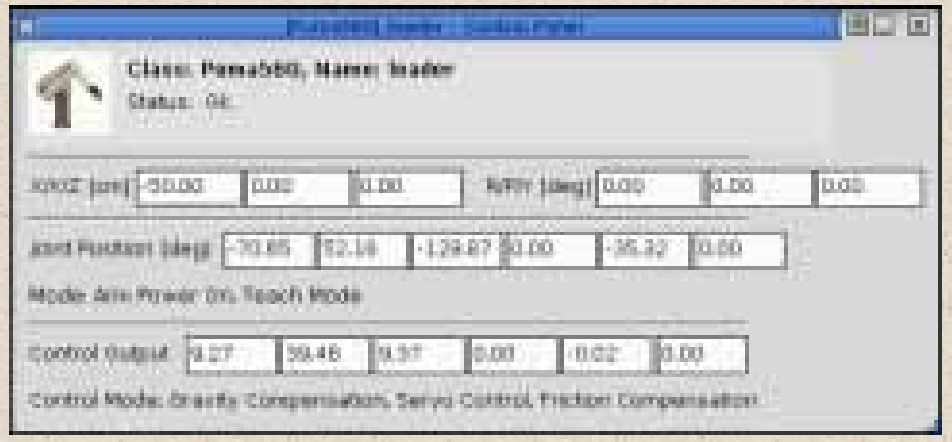


Scene Viewer Window

Object List Window



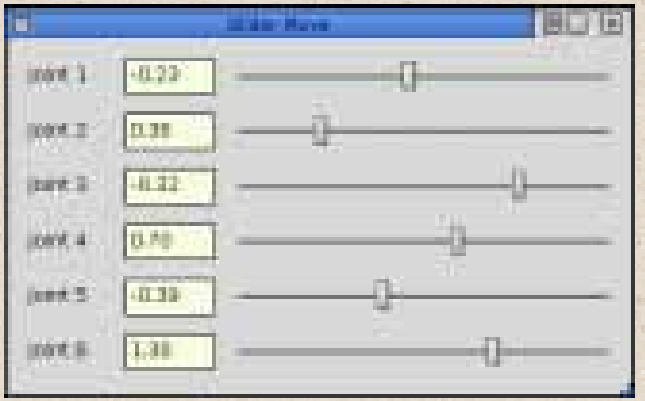
Object Popup Window



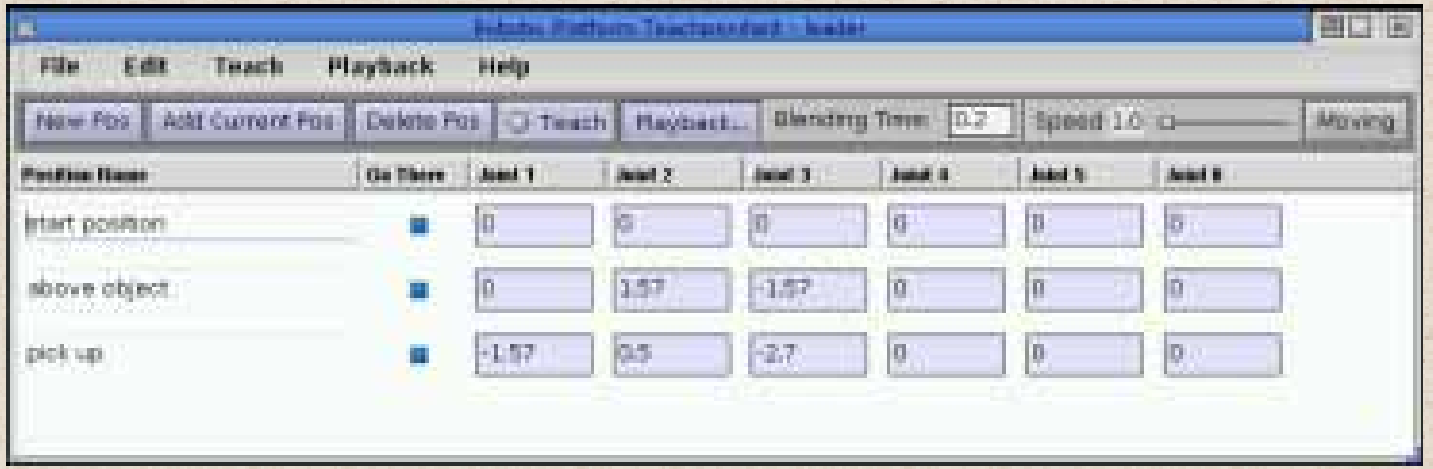
Control Panel

User Interface - Applications

Slider Move Utility



Teachpendant



Programming Examples - Generic Programming

Specific Code

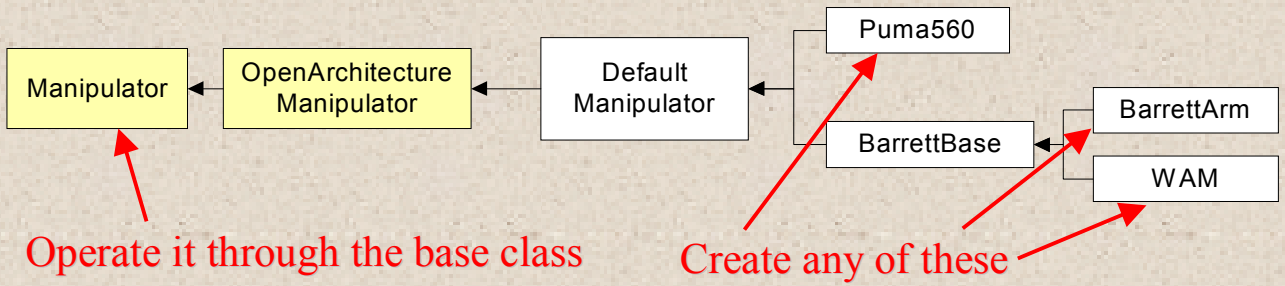
- Need to specify which implementation to use

```
Puma560 manipulator;

manipulator.setTeachModeOn();
```

```
WAM manipulator;

manipulator.setTeachModeOn();
```

Generic Code

- Abstracts from the actual implementation, e.g., to write a trajectory generator that can be used with different manipulator



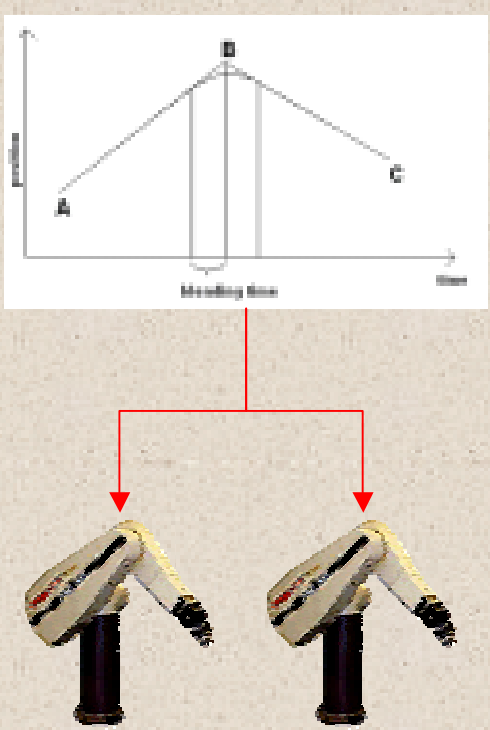

```
Manipulator *manipulator;
ObjectManager om;

manipulator = om.createDerivedObject<Manipulator>("follower");

manipulator-> setTeachModeOn();
```

Global Configuration File

```
[leader]
class Puma560
simulationMode off

[follower]
class WAM
simulationMode on

[gripper]
class DefaultGripper
```

Programming Examples – Two Manipulators

Forwarding the same trajectory to a simulated and a real manipulator




```
#include "RoboticPlatform.hpp"

void main(int argc, char *argv[])
{
  // Initialize the Robotic Platform framework
  RoboticPlatform::init(argc, argv);

  // Create the robot objects
  Puma560 puma(1);
  Puma560 pumaSimulated(2);

  // The second robot is simulated
  pumaSimulated.setSimulationModeOn();

  // Connect both to the same trajectory generator
  DefaultTrajectoryGenerator tragen;
  puma.setTrajectoryGenerator(tragen);
  pumaSimulated.setTrajectoryGenerator(tragen);

  Vector<6> target;

  target = 0, 1.57, -1.57, 0, 0, 0;
  tragen.move(target);

  target = -1.57, 0.5, -2.7, 0, 0, 0;
  tragen.move(target);
}
```

Conclusions

Novel Architecture

- Homogeneous object-oriented design
- “One-box” solution: Uses a single computing platform, a single operating system, and a single programming language

Advantages

- Only a single PC is required (cost advantage)
- Flexibility and extensibility of all components
- Lower complexity
- Easier to install, configure, and use

Other novelties

- Real-time math library
- Adaptive 3D simulation and graphical user interface