Separation of Concerns in Component-based Robotics

Davide Brugali

Università degli Studi di Bergamo, Italy

Politecnico di Milano, 7-06-2010
Robot Control Architectures

- Typical functions implemented in software
  - Acquiring and interpreting input from sensors
  - Controlling the motion of all moving parts
  - Representing the robot’s and environment’s state
  - Managing all kinds of resources (power, CPU, memory, devices)
  - Planning future activities
  - Reacting to unpredictable events

- Typical control paradigms
  - Sense → Plan → Act
  - Behavior-based control
  - Layered control
Robot Software Architectures

• Typical aspects of software architectures
  – decomposition of the robot control system into a collection of software components
  – encapsulation of functionality and control activities into components
  – definition of data flow and control flow among components

• Typical software quality factors
  – Reusability
  – Portability
  – Interoperability
  – Maintainability
Control vs Software Architecture

Control vs Software Architecture

M. Kim, S. Kim, S. Park, M. Choi, M. Kim and H. Gomaa
Control vs Software Architecture
Control vs Software Architecture

Control vs Software Architecture

Control vs Software Architecture

M. Kim, S. Kim, S. Park, M. Choi, M. Kim and H. Gomaa
Control vs Software Architecture

M. Kim, S. Kim, S. Park, M. Choi, M. Kim and H. Gomaa
PUSH
Control vs Software Architecture

READ
Control vs Software Architecture

- The same sensor component is not reusable in two different robotic systems

- Where is the problem?

- Not in the control architecture
  - Both architectures indicate that the sensors supply data to the control modules

- Not in the software architecture
  - The interaction pattern (push / pull) depends on specific requirements of each application
Separation of concerns

- The implementation of the sensor components mixes two different aspects:
  - The component functionality
  - The interaction mechanism
Separation of concerns

• The implementation of the sensor components mixes two different aspects:
  
  – The component functionality is stable → should be reusable

  – The interaction mechanism
Separation of concerns

- The implementation of the sensor components mixes two different aspects:
  - The component functionality is stable → should be reusable
  - The interaction mechanism is variable → should be flexible
- IEEE RAM Tutorial on Component Based Robotic Engineering
  - Part I : December 2009
  - Part II : March 2010
Refactoring aims at restructuring a set of existing software libraries without affecting their external behavior in order to harmonize their architecture, data structures, and APIs.
Refactoring Mobile Manipulation libraries

CONFIGURATION

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSL</td>
<td>MSLVector: double array, includes size</td>
</tr>
<tr>
<td>MPK_{Kernel}</td>
<td>Configuration: (self-defined) vector of doubles, includes call to OpenGL</td>
</tr>
<tr>
<td>CoPP</td>
<td>Config: typedef for vector&lt;double&gt;</td>
</tr>
<tr>
<td>MPK_{Kit}</td>
<td>mpkConfig inherits from vector&lt;double&gt;, includes various functions</td>
</tr>
<tr>
<td>OpenRAVE</td>
<td>TPOINT includes vector&lt;dReal&gt;, additionally velocities and time</td>
</tr>
<tr>
<td>OOPSMP</td>
<td>State_t: typedef for double* (needs external storage of size)</td>
</tr>
<tr>
<td>OMLP</td>
<td>State: double array (needs external storage of size) and flags</td>
</tr>
</tbody>
</table>

C-SPACE

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSL</td>
<td>Model (and Problem) have upper/lower limits, and various more things covering control inputs and system simulation</td>
</tr>
<tr>
<td>MPK_{Kernel}</td>
<td>upper/lower limits are derived from joints</td>
</tr>
<tr>
<td>CoPP</td>
<td>upper/lower limits stored explicitly where needed</td>
</tr>
<tr>
<td>MPK_{Kit}</td>
<td>limits implicitly in planner</td>
</tr>
<tr>
<td>OpenRave</td>
<td>ConfigurationState includes limits and number of DoF</td>
</tr>
<tr>
<td>OOPSMP</td>
<td>StateSpace includes bounding box and various other functions. Many concrete implementations</td>
</tr>
<tr>
<td>OMLP</td>
<td>SpaceInformation includes: start and goal config, dimension, StateDistanceEvaluator, StateValidityChecker</td>
</tr>
</tbody>
</table>

INTERFACES FOR COLLISION DETECTION

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSL</td>
<td>Geom with derived class for PQP CollisionDetectorBase. Universe has an array of Mesh which can model various objects.</td>
</tr>
<tr>
<td>MPK_{Kernel}</td>
<td>ObjectSet. Base class Geom stores a position, with inherited classes for triangles and convex objects. Geometric objects are attached to kinematic structures by pointing to their transformation.</td>
</tr>
<tr>
<td>CoPP</td>
<td>mpkColl1Dist1Algo uses PQP or own collision detector</td>
</tr>
<tr>
<td>MPK_{Kit}</td>
<td>CollisionCheckerBase. KinBody includes TRIMESH and GEOMPROPERTIES for modelling triangle meshes</td>
</tr>
<tr>
<td>OpenRAVE</td>
<td>CollisionDetector. Workspace holds list of Part, support of polygons</td>
</tr>
<tr>
<td>OOPSMP</td>
<td>Based on ROS with interfaces of CollisionSpace and various geometry messages</td>
</tr>
</tbody>
</table>

PATH

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSL</td>
<td>list&lt;MSLVector&gt;: list of vectors</td>
</tr>
<tr>
<td>MPK_{Kernel}</td>
<td>PathBase: abstract base class, and PA_Points with vector&lt;Configuration&gt;</td>
</tr>
<tr>
<td>CoPP</td>
<td>Path: own data type encapsulates list of configs and time</td>
</tr>
<tr>
<td>MPK_{Kit}</td>
<td>vector&lt;mpkConfig&gt;: vector (or list) of configurations</td>
</tr>
<tr>
<td>OpenRAVE</td>
<td>Trajectory includes vector of points and segments, in addition elements for dynamic motion control</td>
</tr>
<tr>
<td>OOPSMP</td>
<td>Path includes interfaces for times, splitting and more. Base class with various implementations</td>
</tr>
<tr>
<td>OMLP</td>
<td>Path: points to a SpaceInformation, derived classes include array of State</td>
</tr>
</tbody>
</table>
Component Interface & Implementation

```
<Interface>
PathUpdating
+updatePath()
</Interface>

<Interface>
PlannerSetup
+selectGlobalPlanner(in criteria : string, in param : vector<T>)
+selectUpdater(in criteria : string, in param : vector<T>)
+selectLocalPlanner(in criteria : string, in param : vector<T>)
+selectSampler(in criteria : string, in param : vector<T>)
</Interface>

<Interface>
PathPlanning
+getPath(in start : Config, in end : Config) : Path
+nextConfig(in currentConfig : Config) : Config
</Interface>

PlanerComponent

Sampler
+getSample() : Config

UniformSampler

AdvancedSampler
+Instantiate()

LocalPlanner
+connect(in from : Config, in to : Config) : Path

VariantDescriptors

BinaryConnector

PathUpdater
+updatePath()

<<use>>

Path

CommaiteLeg

PathLeg

Configuration

GlobalPlanner
+getPath(in start : Config, in end : Config) : Path
+nextConfig(in currentConfig : Config) : Config

<<use>>

PathUpdater

ElasticStripUpdater

StaticUpdater

FADPRMUpdater

PCDPlanner

PRMPlanner

RRTPlanner
```
Configuration

Sequential Components

Path Planner
Collision Checker
Collision Checker
Collision Checker
Collision Checker
Collision Checker
Collision Checker
Collision Checker
Collision Checker
Cartesian Space

Service Components

Map Updater
Navigator

Container Component

Resource Manager
QoS Negotiation
Scheduler

QoS Negotiation
Communication

Distributed Middleware

Path Planner

Navigator

Potential Field

Obstacle Avoider

Event Manager

Container

Event Manager

Container
Communication

Path Planner → Navigator

Motor Driver → Locomotor

Container → Stub

Container → Skeleton

Distributed Middleware
Communication

**Assembly**

Path Planner → Navigator

**Assembly**

Motor Driver → Locomotor

**Contained**

Stub

**Contained**

Skeleton

**Distributed Middleware**
Coordination

Connector

Assembly

Path Planner

Navigator

Container

Stub

Motor Driver

Locomotor

Container

Skeleton
Coordination

![Diagram of coordination](image)
BRICS Research Camps on Mobile Manipulation

Malaga, Costa de Sol, Spain - October 24 - 29, 2010

- Invite best Ph.D. students AND PostDocs from all over the world
- We will provide
  - travel grants (1250 EUR for European students, 2000 EUR for international students)
  - the latest and coolest pieces of robot hardware in mobile manipulation
  - a DVD with best practice software for mobile manipulation
  - a fast Internet access
  - typical mobile manipulation tasks
- We expect in return
  - a competitive solution to the given tasks either using the provided or self-developed algorithms for mobile manipulation demonstrated in two competitions on the last day of the research camp
  - critical feedback and revisions of the provided hardware and software
Thank you for your attention!

Davide Brugali
Università degli Studi di Bergamo, Italy
Politecnico di Milano, 7-06-2010