Robotics Programming Laboratory

Bertrand Meyer
Jiwon Shin

Lecture 10: Software Architecture in Robotics
Control and navigation architecture

Serial architecture

Parallel architecture
Sense-Plan-Act

Architecture

- Sense the environment.
- Plan the next move based on the goals.
- Execute the plan through the actuators.

Properties

- Easy to execute a plan
- Must generate a plan and model the world
- No feedback: insufficient to handle environmental uncertainty and unpredictability.

Subsumption architecture

Architecture

- Divide the control into different behaviors, where the higher level behavior subsumes the lower level behaviors.
- Let the arbiter pick the appropriate behavior for the given condition.

Subsumption Architecture

Properties

- Each layer/behavior as a small finite state machine
  - Each behavior achieves a single goal
  - No cooperation of different behaviors
  - Deduce the best next action based on the current sensor readings
- Reactive: rapidly responds to environmental changes
- No global representation nor world model
- No planning nor meta-reasoning
- Not taskable
  - Does not remember the current goals
Three-layer architecture

Architecture

- **Deliberator**: perform high-level computations
- **Sequencer**: select which primitive behavior the controller should use at a given time and supply parameters for the behavior.
- **Controller**: Perform primitive behaviors, with tight coupling of sensors to actuators

Properties

- Avoids the bottleneck problem
- Can plan and learn
- Can operate in dynamic environment

Tiered robot architecture examples

Three-tiered architecture

- Planning
- Executive
- Real-time controller
- Behavior 1
- ...  
- Behavior n
- Motion control

Perception → Action

Robot hardware

Two-tiered architecture with offline planning

- Executive
- Real-time controller
- Behavior 1
- ...  
- Behavior n
- Motion control

Perception → Action  

Robot hardware

Tiered robot architecture examples

Three-tiered architecture with episodic planning

- Global knowledge
- Executive
- Planning
- Local knowledge
- Real-time controller
- Behavior 1
- ...
- Behavior n
- Motion control
- Perception
- Action
- Robot hardware

Two tiered with integrated planning

- Executive
- Planning
- Global knowledge
- Real-time controller
- Behavior 1
- ...
- Behavior n
- Motion control
- Perception
- Action
- Robot hardware
It’s all good but ...

Why reinvent the wheel?
Robotics Frameworks

- Ease the development of control software for robots.
- Provide standards, principles, applications, and libraries to support common tasks.
- Exemplary frameworks
  - Yet Another Robot Platform (YARP)
  - Universal Robotic Body Interface (URBI)
  - Mission-Orientated Operating Suite (MOOS)
  - Microsoft Robotics Development Studio
  - Robot Operating System (ROS)
Model-View-Controller (MVC)

- A central hub coordinates the communication.
- The modules read parameters and maps from a centralized model repository.
- The modules communicate over the network with a publish/subscribe pattern.
- The modules are arranged in layers.
# CARMEN modules

<table>
<thead>
<tr>
<th>Layer</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Layer</strong></td>
<td>• High-level tasks, e.g. tour giving, interaction, etc.</td>
</tr>
<tr>
<td><strong>Navigation Layer</strong></td>
<td>• Localization, planning, mapping, visual processing, logging, and simulation</td>
</tr>
<tr>
<td><strong>Base Layer</strong></td>
<td>• Hardware management and communication</td>
</tr>
<tr>
<td></td>
<td>• Collision detection</td>
</tr>
<tr>
<td><strong>Non-autonomous Layer</strong></td>
<td>• Display modules, editors, etc.</td>
</tr>
</tbody>
</table>
Observer pattern

- Special port objects deliver messages to other observers/ports.
- Every connection can take place using a different protocol.
- Every port belongs to a process.
- Ports are located on the network by symbolic names.
- A name server maps the names into the IP address, port number, and interface name.
YARP components

libYARP_OS
- Interface with operating system(s)
- Data streaming across many threads across many machines

libYARP_sig
- Signal processing tasks (visual, auditory)
- Easy interface with commonly used libraries

libYARP_dev
- Interface with common devices used in robotics: framegrabbers, digital cameras, motor control boards, etc.
Client-server architecture

- Several clients can interact with the server simultaneously.
- Remote objects can also connect to the server.
URBI components

**UObject**
- C++ component library that comes with a robot standard API to describe motors, sensors and algorithms.

**UrbiScript**
- Orchestration scripting language
- Runs on top of Urbi Virtual Machine
- Glues components together and describes high level behaviors
- Supports parallel and event-driven programming
Representational State Transfer (REST) pattern

- A program interacts with a robot through multiple software services.

- A distributed messaging system enables services to communicate on the same computer over the network.

- A configuration manifest file defines the interaction of services in a particular control system.
MRDS core features

Coordination and Concurrency Runtime (CCR)
- Handles state updates and message processing

Decentralized Software Services Protocol (DSSP)
- Launches services from its manifest descriptions
- Provides for partnering
- Facilitates communications between message ports on individual services

Generic contracts for common elements of a robotic system
- New services can specify to which contracts it conforms
- A discovery service lists all currently running services that conform to a certain contract.
A star topology with layered architecture

- A central server with a database of messages
- Each client bundles its messages and sends them to the server.
- All communication between the client and the server is instigated by the client.
  - A client subscribes for messages of the right type.
  - The client picks up the messages whenever it connects to the server.
- No peer-to-peer communication.
MOOS client

- **Application Layer**
  - Applications

- **Essentials Layer**
  - Commonly used functionality such as control and logging

- **Communication Layer**
  - Connects clients (e.g. sensors, actuators, processes, etc.) through a network with a star topology
Peer-to-peer network architecture

- A central naming service to allow nodes to find other nodes
- Publish-subscribe model for asynchronous transactions
  - A node can publish and subscribe to topics.
  - Many nodes can publish and subscribe to a single topic.
- Service for synchronous transactions
  - Only one node can advertise a service.
  - A response follows a request.