Low Level Control

• Robot H/W control Software (drivers):
  – RHeXLlib
  – Player
  – Ndirect, seriald (Nomadics)

• Simulation
  – RHeX SimSect?
  – Stage
  – Nclient, server
  – RD11
High Level Control

• Important when multi-tasking
• Especially in Multi-Robot settings
• Brief Historical note:
  – Subsumption Architecture (Rodney Brooks)
  – Behaviour based Architecture
  – Three Layer Architectures
    • Combining the above two plus some more 😊
Several Options

- Player/Stage (USC)
- Microsoft Robotics Developers Studio
- ROS (willow garage)
- ALLIANCE (L. Parker)
- RoboDevel/RHeXlib (U. Saranli)
- Robodaemon [RD11] (MRL product)
- CLARAty (JPL)
- CAMPOUT (JPL)
- SAPHIRA (Konolige)
- CARMEN (Thrun, Roy)
- EPICS (Junaed, J. Smith suggestion)
- Subsumption (Rodney Brooks)
- Three layer Architectures
- DCA (Christensen)
- Reid Simmons projects
  - TeamBots (Balch), Mission Lab (Arkin), Ayllu (Werger), ARIA (ActivMedia)
Sense Plan Act
Subsumption

- The Subsumption architecture is built in layers.
- Each layer gives the system a set of pre-wired behaviours.
- The higher levels build upon the lower levels to create more complex behaviours.
- The behaviour of the system as a whole is the result of many interacting simple behaviours.
- The layers operate asynchronously.

See: http://ai.eecs.umich.edu/cogarch0/subsump/index.html
Subsumption

Figure 1. A traditional decomposition of a mobile robot control system into functional modules.


Figure 2. A decomposition of a mobile robot control system based on task achieving behaviors.
Three-Layer Architectures

- The Controller (low level, tight coupling)
- The Sequencer (selecting low level behaviours)
- The Deliberator (time-consuming computations)

Player and Stage

- Following the bazaar/open_source model
- Player is the low level control interface
- Stage is a simulation engine (2D)
- Gazebo is a 3D simulation engine
Player

- TCP socket server
- Clients connect to the server and send/receive commands/data
- Sensor and actuator abstraction
Player

[Diagram showing a mapping computer, vision system, laser scanner, data logging server, and robots connected to a GUI & Debugging workstation]
Player Architecture
CARMEN

• Welcome to CARMEN, the Carnegie Mellon Robot Navigation Toolkit.
• CARMEN is an open-source collection of software for mobile robot control.
• CARMEN is modular software designed to provide basic navigation primitives including:
  – base and sensor control
  – logging
  – obstacle avoidance
  – localization
  – path planning
  – mapping

See: http://carmen.sourceforge.net/
Microsoft Robotics Developer Studio

- Concurrency and Coordination Runtime
- Decentralized Software Services
- Visual Programming Language (VPL)
- Physics based Simulation Engine
- Web-based Technology
- Not-Open Source

Concurrency and Coordination Runtime (CCR) is a managed code library, a Dynamically Linked Library (DLL), accessible from any language targeting the .NET Common Language Runtime (CLR).

- Service-oriented applications
- manage asynchronous operations
- deal with concurrency
- exploit parallel hardware and deal with partial failure.
- The software modules or components can be loosely coupled
- They can be developed independently and make minimal assumptions about their runtime environment and other components.
Decentralized Software Services (DSS) is a lightweight .NET-based runtime environment that sits on top of the Concurrency and Coordination Runtime (CCR):

- Lightweight
- state-oriented service model
  - Combines the notion of representational state transfer (REST) with a system-level approach for building high-performance, scalable applications.
- DSS services are exposed as resources which are accessible both programmatically and for UI manipulation.
- Integrating service isolation, structured state manipulation, event notification, and formal service composition
- Robustness
- Composability
- Observability
Graphical Programming
Physics based Simulation Engine
Web based Interface

Aqua in Barbados

MRDS

802.11n

pull

push

Custom Scripts

802.11n

Internet

Web-Server

Client

Presentation at St. Georges high school, Montreal

Underwater Swimmer

Camera

Depth (m)

0

0.14

-30

Roll Pitch Yaw

Leg 1

Leg 2

Leg 3

-11.96 -21.47 -6.94

1.12

1.93

1.68

Leg 4

Leg 5

Leg 6

Battery 11.61 V

0.38

0.60

0.41

LED: OFF

- School of Computer Science
- Centre for Intelligent Machines
- Mobile Robotics Lab
- AQUA Robot
ROS

- ROS is an open-source, meta-operating system for robots.
- It provides the services expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.
- ROS is similar in some respects to 'robot frameworks,' such as Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio.
- The ROS runtime "graph" is a peer-to-peer network of processes that are loosely coupled using the ROS communication infrastructure.
- ROS implements several different styles of communication, including synchronous RPC-style communication over Services, asynchronous streaming of data over Topics, and storage of data on a Parameter Server.

See: http://www.ros.org/wiki/ROS
ROS
CLARAty

- A two layer architecture
- Developed at NASA/JPL
- Supporting different h/w

Different Mobility platforms

(a) Skid Steering (no steering wheels)
(b) Tricycle (one steering wheel)
(c) Two -wheel steering
(d) Partially Steerable (e.g. Sojourner, Rocky 7)
(e) All wheel steering (e.g. MER, Rocky8, Fido, K9)
(f) Steerable Axle (e.g. Hyperion)
Approach

• Develop
  – Common data structures
  – Physical & Functional Abstractions
    • E.g. motor, camera, locomotor. Stereo processor, visual tracker
  – Unified models for the mechanism

• Putting it together
  – Start with top level goals
  – Elaborate to fine sub-goals
  – Choose the appropriate level to stop elaboration
  – Interface with abstractions
  – Abstractions translate goals to action
  – Specialize abstractions to talk to hardware
  – Hardware controls the systems and provide feedback

From: http://claraty.jpl.nasa.gov/main/overview/presentations/FY05/FY05_claraty_jtars.pdf
Two Layer Architecture

**THE DECISION LAYER:**
Declarative model-based
Global planning

**INTERFACE:**
Access to various levels
Commanding and updates

**THE FUNCTIONAL LAYER:**
Object-oriented abstractions
Autonomous behavior
Basic system functionality

Adaptation to a system
Behaviour Layer Base Loop

- **Check For Messages**
- **Sense (update sensor data)**
- **Reason (set velocities)**
- **Act**