Intelligent Robots Seminar

Topic:

Robotics Software Architectures

Outline

1. Background

   Description of Robotic Software Architecture

2. What is Robot Operating System?

   Terminologies used

   ROS Comunication Protocols

   Transformation and Navigation

3. Player / Stage

   Architecture (Device Addressing, Virtual Drivers, Message Passing)

4. Comparisons
5. Conclusion
6. References

Background

Robotics Software Architecture described from three points of view

-Robot functional architecture

system's control and data flows, data sources , data stores ,

data processing and data sinks (e.g. the actuators).

- Robot component architecture

software structured in components plus dependencies ,

communication, quality of service

- Robot runtime architecture

-Software components mapped onto processes.

Definition of Software Architecture

- Description of the subsystems
- Components of a software system
- Relationship between the components
- Specified in different views to show Functional and

  non –functional properties of the system

What is ROS / What it isn‘t

- A Peer to Peer distributed environment for robots
- A “meta” operating system for robots
- Provides a structured communications layer above operating system
- Can run on the robot itself or on a remote computer
- A collection of packaging, software building tools
- Simply, each logical unit holds its own service (process) which is accessible over the network.

ROS Graph Level Terminologies

- **Node** : processes that perform computations

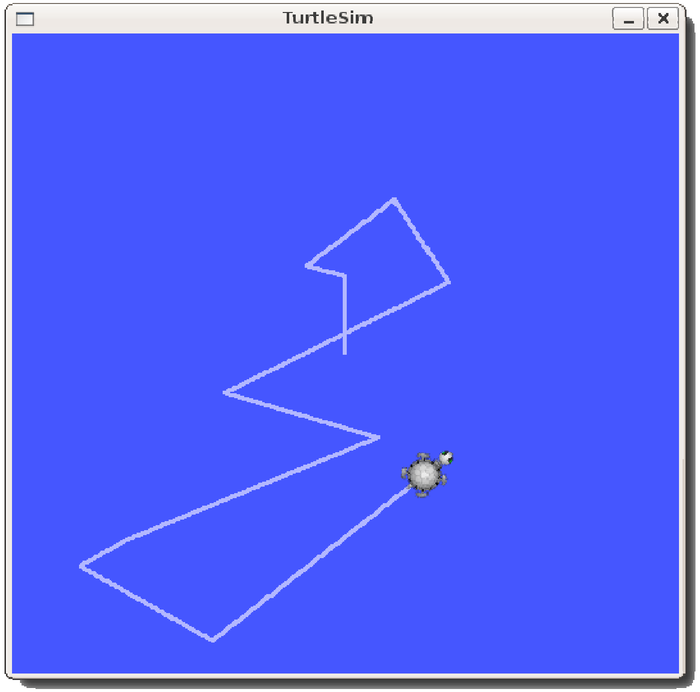
  Include Slave API, TCPROS, Commandline API

- **Master** : naming and registration services to the rest of the nodes, negotiate communication connections

- **Parameter** : Persistent data like config file, initialization setting stored on the Parameter server

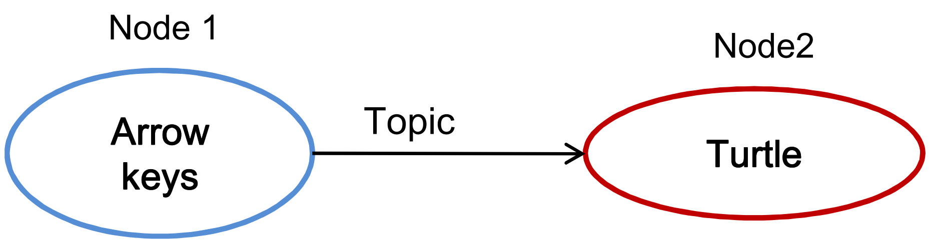
- **Parameter server** : Stores persistent configuration parameters and other arbitrary data

ROS Graph Level Terminologies



- **Topic** : An event handler for an asynchronous communication among multiple nodes.

  Uses XMLRPC to negotiate connection for data



ROS Graph Level Terminologies

- **Service** : A method interface for a synchronous communication between nodes.

  Only one service provider at a time

- **Bag** = A logger for messages that are passed on a specific or multiple topics

ROS File System Level Terminologies

- **Message :** A structured data- type that is passed to the topic
- **Service message :** A Request- and- response messages that is passed to the service;
- **Package :** A named Organizational unit for nodes, configuration and more

ROS File System Level Terminologies

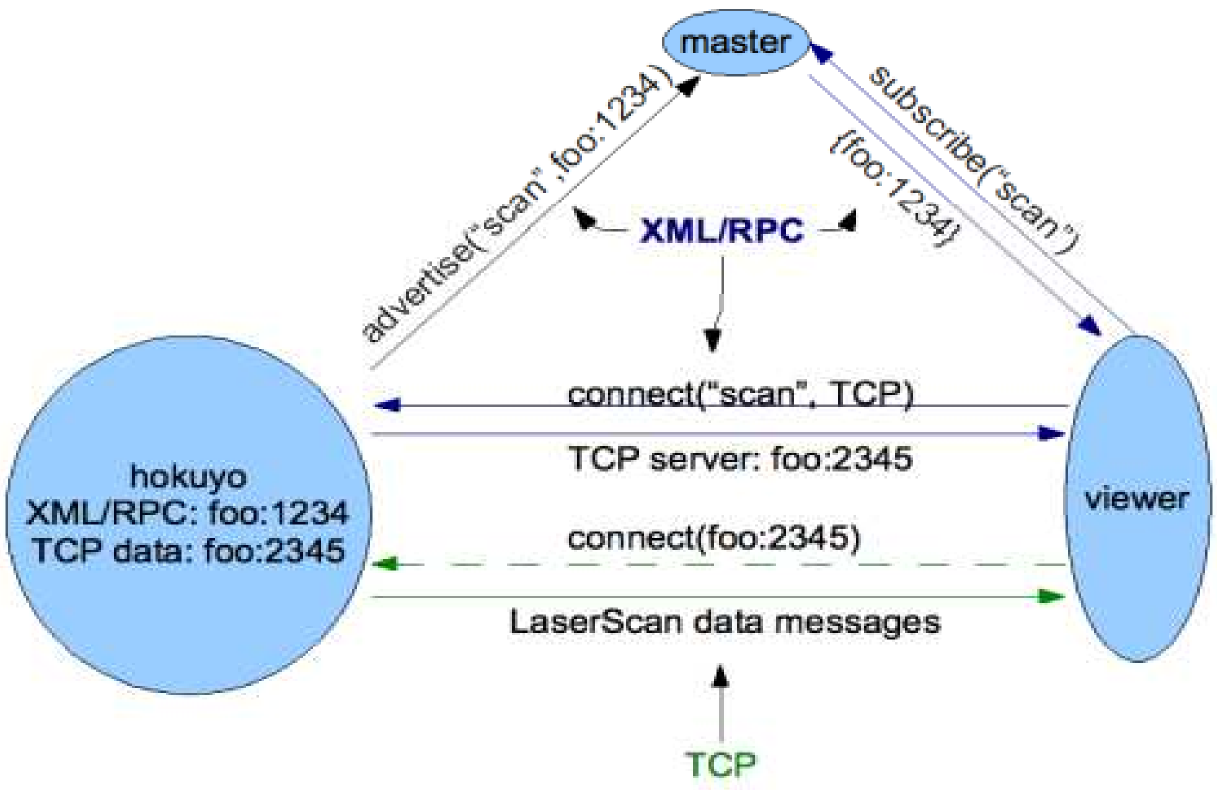
Manifests(manifests.xml) : metadata about a package like license and dependencies, language specific information

- **Stack** : A name collection of packages for distribution
- **Stack Manifests**(stack.xml): Provide data about a stack like License Info and its dependecies on other stacks

Communication Protocols in ROS

1. ROS Topics

   - Asynchronous "stream-like" communication



Communication Protocols in ROS

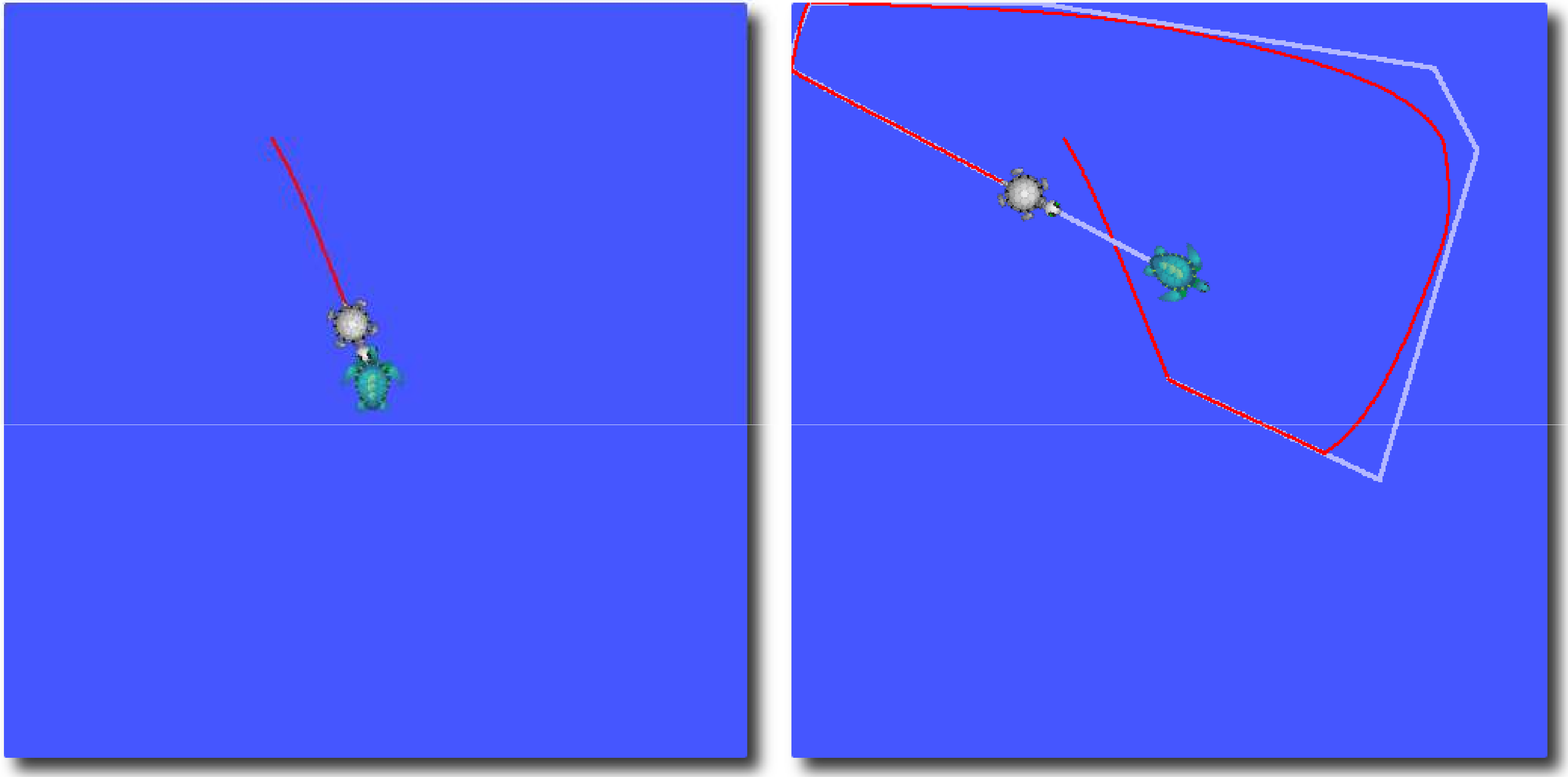
2. ROS Services

- Synchronous “function-call-like” communication
- TCP/IP or UDP Transport
- Register with Master, client look up service
- Client creat TCP/ IP to service
- Exchange Connection header, send message.
- If no service registered, client lib provide „wait for service“ API that poll Master

Transformation tf

- need to track spatial relationships – mobile robot and fixed frame of reference for localization
- 3D co-ordinate frame system changes over time
- Tranformation tree defines offsets in terms of translation and rotation between different coordinate frames

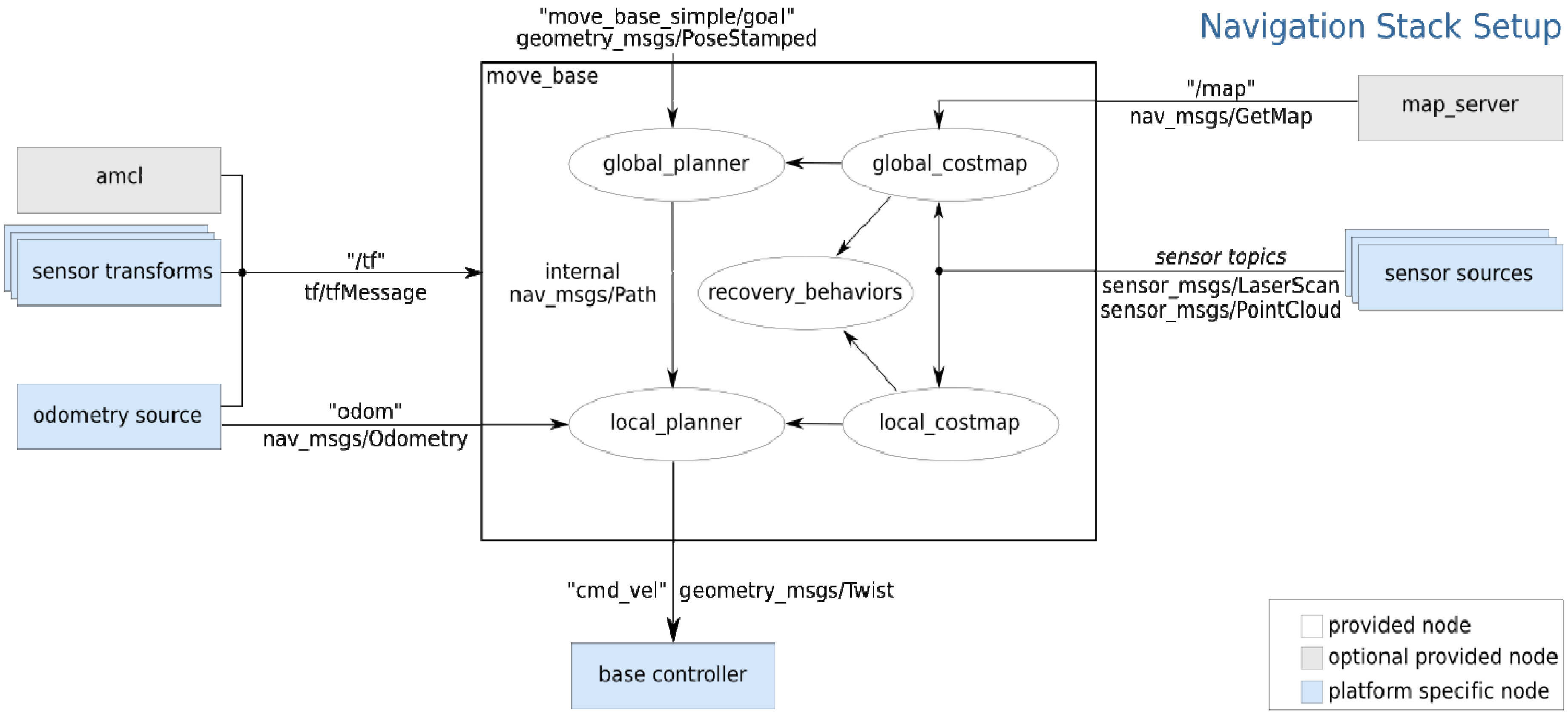
Transformation tf



Transformation tf

- tf library to create three coordinate frames: a world frame, a turtle1 frame, and a turtle2 frame.
- Adds transformation tree, each node is a coordinate frame.
- **tfbroadcaster** publishes transforms, and a node to **tflistener** to compute difference in turtle frames and move one turtle after another.
- Additional tf tools available like tf_echo, view_frames, rviz

Navigation in ROS



Navigation in ROS

- Robot must be publishing information about relationship btn coordinate frames - tf
- ACML

  Provides config option for localization performance
- Local Planner

  computing velocity commands to send to the robot's mobile base
- Global costmap

  Long term plans over entire environment

Navigation in ROS

- **Local costmap**(freq, height, resolution, obstacle range)

  Local planning and obstacle avoidance

  Point maps to sensor topics to listen for updates

- **Base Controller**

  Subscribes to cmd_vel topic, taking (vx, vy, vtheta) velocities

  converting them into motor commands to send to a mobile base.

- **Odometry Information**

  Published using tf library

Navigation in ROS

- **Sensor sources**

  Help avoid obstacles in the real world.

  Tansformations in sensor orientations improve map

  Specify, does sensor add obstacle info or clear it from map

  Include lasers

- **Map Server**

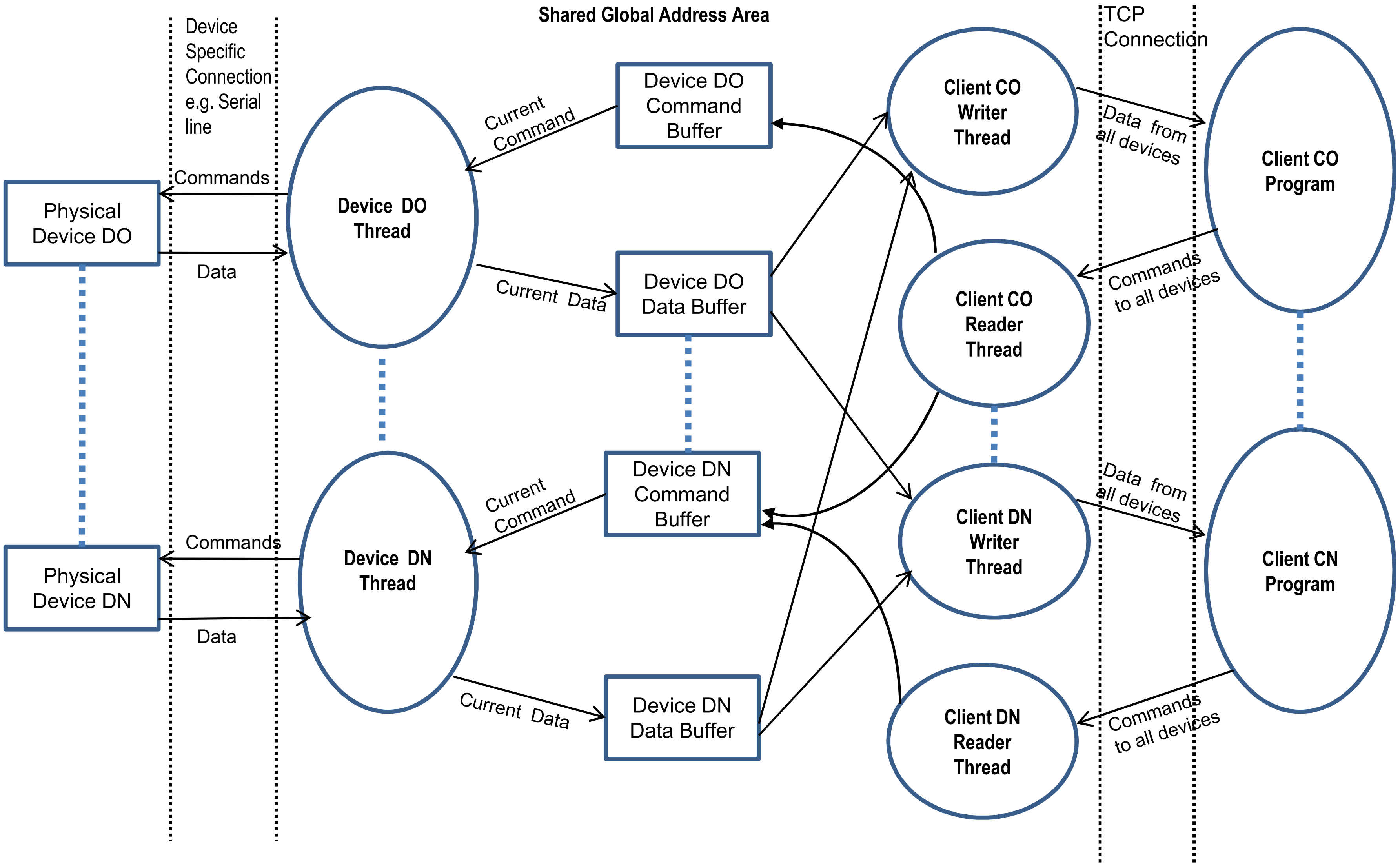
  Need to create your own map of your environment.

  „Video presentation“ ??

What is Player / Stage?

- Player Server: Simply an interface for communicating with a robot‘s resources(sensors and actuators) over an IP network
- Player is a network server for robot control
- Runs on your Robot and on PC – Linux, Solaris OS
- client program talks to Player over a TCP socket
- modular architecture makes it easy to support new hardware
- Player allows multiple devices to present the same interface

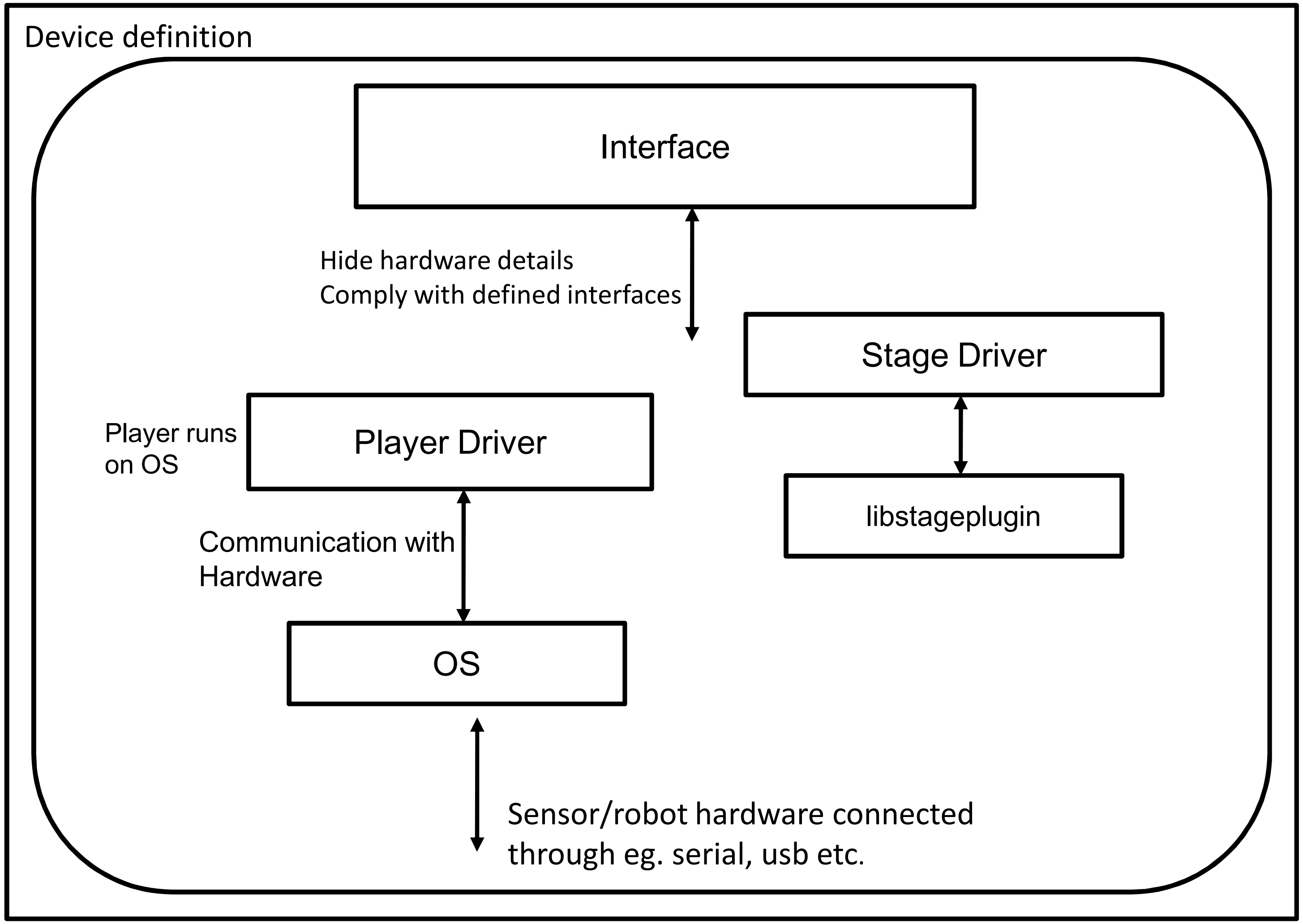
Player Architecture



Player Architecture

- Communication btn servers and clients through buffer on server
- Asynchronous architecture allowed clients and devices own speeds of operation
- Used 2 threads for client(1 reader, 1 write), 1 for each active device and 1 for listening to new clients
- Internally devices communicated through Global shared memory
- Each device had 2 buffers to store latest commands and data

Key concepts in Player



Key Architectural Concepts

1. **Interface:**

   define the syntax of how to issue commands to actuators and how to read inputs from sensors through messages.

2. **Driver:**

   - software that talks with the actual hardware

   - translates its I/O to conform to relevant interface

   -**Virtual Drivers** – special purpose functionality

3. **Device:**

   - topmost abstraction in Player for the hardware

   - used through a fully-qualified device address

Device Addressing

- fully-qualified device address consists of 5 parts

  - the key is a unique name,

  - the host and robot fields = Player server listens,

  - the interface – is the interface name of the device,

  - the index - interface specific sequence number

Example:

a driver providing front and back IR sensors as two different

ir devices can use: *front:::ir:0 and back:::ir:1*

.

Message Passing System

- Player is a queue based message passing system

Message Types

- **Data Messages -** used by drivers to publish sensor readings plus changes in device state, such as motor stalls.
- **Command Messages** – Sent by clients ordering driver to change stateof a specific device it controls
- **Configuration messages** – provide a way for clients to configure device properties say poses of individual sensors

ROS compared to Player / Stage

Disadvantages of ROS:

- Can only be run on a Unix platform
- Takes time to learn the effectively use ROS structure

Advantages

- A language-independent architecture (c++, python)
- Open-source under permissive BSD licenses (ros core libraries)
- Designed to be used by any robot
- Easy to create and find packages, nodes, and messages
- Player offers more hardware algorithms but ROS offers more implermentation of algorithms

ROS compared to Player / Stage

Advantages of Player

- Supports a wide range of devices like Lasers, sonars, camera etc
- Programs written for the simulated environment can work on the actual robot
- Sensor models and Odometry are the same as those used in actual robot
- Open source, supports a number of programming languages, Windows
- Virtual drivers allow for error correction in hardware

Disadvantages:

- Latency in client/Server model
- Interface/Driver model

In Conclusion

- Both ROS and Player are relatively good tools for development of Robot Software.
- ROS has however in recent years been the more popular tool than any other.
- ROS has great potential in an industrial connection
- companies don't need to start from scratch every time and instead are able to reuse existing components
- But who is responsible if a sub-component from ROS causes an error in the system?
- So very important to quality assure the components before use

References

- Software Architecture in Practice (2nd Edition) – Lens Bass, Paul Clemens, Rick Kazman
- Introduction to ROS distribution, build system and infrastructure – Jonathan Bohren (JHU)
- www.wiki.ros.org/ROS
- Player/Stage – Player Driver Implementation for ESRP Journal – By Bue Petersen and Jonas Fonseca
- ROS: an open-source Robot Operating System - Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust ,Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, Andrew N.
- Journal on Three Layer Architectures – Erann Gat Jet Propulsion Laboratory- California Inst. Of Technology
- www. http://playerstage.sourceforge.net/index.php?src=player – Visited on 15th Nov. 2013
- Journal: On device abstractions for portable, reusable robot code - Richard T. Vaughan, Brian P. Gerkey, Andrew Howard (University of Southern California,)