ROS

Open Source Robot Operating System

Jürgen Hess, Felix Endres, Armin Hornung,
Bastian Steder und Jürgen Sturm

University of Freiburg, Germany

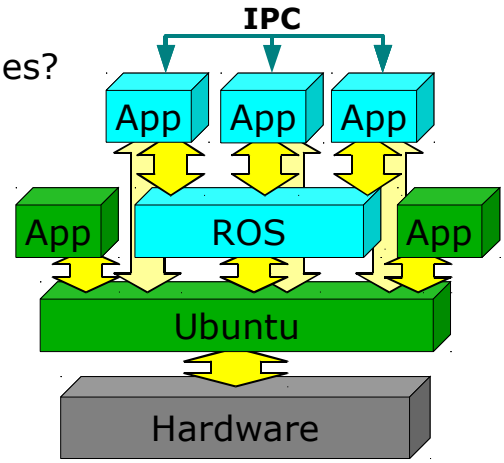
Agenda of the Overview Talk

- High Level Overview
  - What is ROS?
  - Who made it?
  - Why do we use it?
- Middleware Aspects
- Software Structure
- Client Libraries

High Level Overview

What is ROS?

- An Operating System like Windows, GNU/Linux?
- A Software Distribution?
- A Middleware?
- A Set of Libraries?



While ROS can be seen as an operating system – it "manages the computer hardware and provides common services for efficient execution of various application software". However, it is not on the same level as Windows or GNU/Linux, as it is actually based on Ubuntu and comes with additional drivers, functionality and a software infrastructure for the operation of robots. Therefore it is not an self-sustained OS, but an add-on to an existing OS.

ROS is of course also a software distribution. It comes with tools to install, remove and manage ROS applications and software projects. It can deal with dependencies on a binary or a source code level. ROS applications include a robot simulator, 3D Visualization Software and tools to analyze interprocess communication.

Another role of ROS is as a middleware for applications. See later slide.

ROS also provides a set of libraries and according APIs. A set of these is included in ROS programs to make the programs part of the ROS infrastructure, e.g., for using the interprocess communication or the parameter server.

From ros.org (http://www.ros.org/wiki/ROS/Introduction):
"ROS currently only runs on Unix-based platforms. Software for ROS is primarily tested on Ubuntu and Mac OS X systems, though the ROS community has been contributing support for Fedora, Gentoo, Arch Linux and other Linux platforms.

While a port to Microsoft Windows for ROS is possible, it has not yet been fully explored. You can find updates on our current progress, on the Windows page. "

High Level Overview

Who made ROS?

- Privatly Owned Company
- Based in Menlo Park, California
- Hardware: PR2, Texai
- Software: ROS (OpenCV, Player)
- Strong Open Source Commitment
- Extensive Internship Program

4

From willowgarage.com:

Who We Are

Willow Garage is a team of experts in robot design, control, perception, and machine learning, with both a strong theoretical background and a demonstrated drive to produce practical systems.

Approximately one third of our team is made up of robotics researchers whose specialties include: grasping/manipulation, human-robot interaction, motion planning, perception, and task planning. Researchers provide an institutional expertise to ensure that Willow Garage stays at the technological forefront of the many depth expertise areas represented in robotic technologies.

Open Source

We are committed to open source robotics software and the furtherance of the open source personal robotics community. We helped found, and continue to contribute heavily to, the robot operating system, ROS. The ROS software we contribute is BSD-licensed, making it completely free for anyone to use and change, and free for other companies to commercialize on. We see this open-source approach enabling robotics innovation, and helping to ensure that the adoption of robotic technologies is a transparent process with positive societal impact.

External Collaboration

Willow Garage is only one part of the open-source robotics community. We try to focus our efforts in ways that will make the entire community more productive. Willow Garage actively engages research labs and companies as partners, collaborators, customers and advisors in the development of both our hardware platform and open source software. Willow Garage also supports researchers who would not otherwise have the bandwidth or funding to open source their work.

High Level Overview

Why do we use ROS?

- Great functionality
  - Middleware (this session)
  - Development tools (2nd Session)
  - Advanced libraries (3rd Session)
  - Hardware drivers
- Large scientific community
  - A lot of state-of-the-art software available
  - Easy to exchange/integrate/build-upon existing projects
  - Open source (mostly BSD)
  - Actively developed by full-time staff
- To stop reinventing the wheel...

“The primary goal of ROS is to support code reuse in robotics research and development.” – ros.org

ROS – a Middleware for Robots

"Middleware is a software that connects software components or applications."

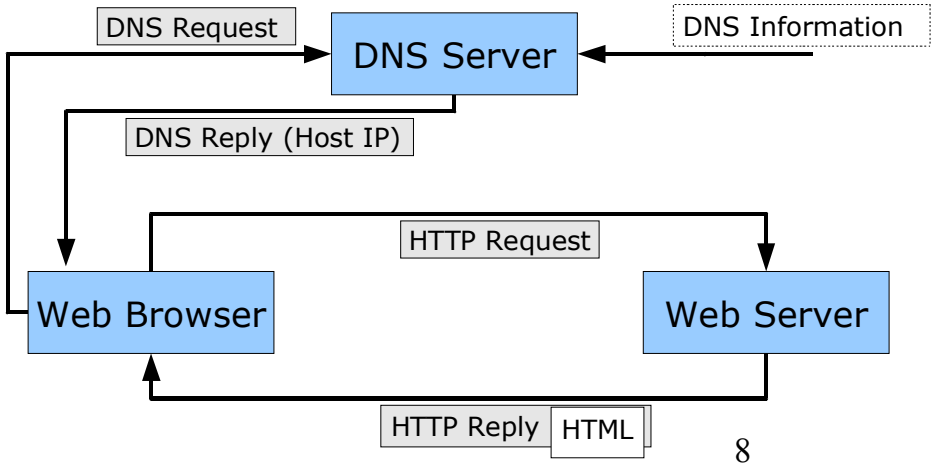
- Framework for interprocess communication
- Boosts modularization
- Enables transparent distribution of software in a network

In complex tasks (and robot operation is complex per se) the required functionality can increase tremendously: components for sensor control, actuator control, processing of sensor data, localization, mapping, reckognition and tracking of humans or objects, navigation and motion planning with collision avoidance.

Some of these tasks have to be executed on the robot itself, some (computation intensive, but low bandwidth i/o) may be distributed to computers connected by network.

ROS - a Middleware for Robots

Example: Communication on the Internet



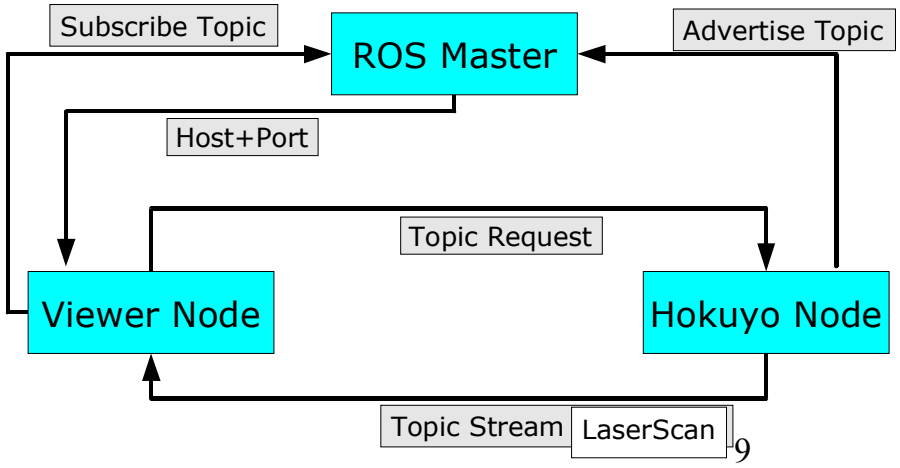
8

The user needs to know what information (which file) it wants from whom (the web address)

The web browser needs to know how to to find the web address and how to handle the information (how to display HTML)

ROS - a Middleware for Robots

Interprocess Communication Using ROS

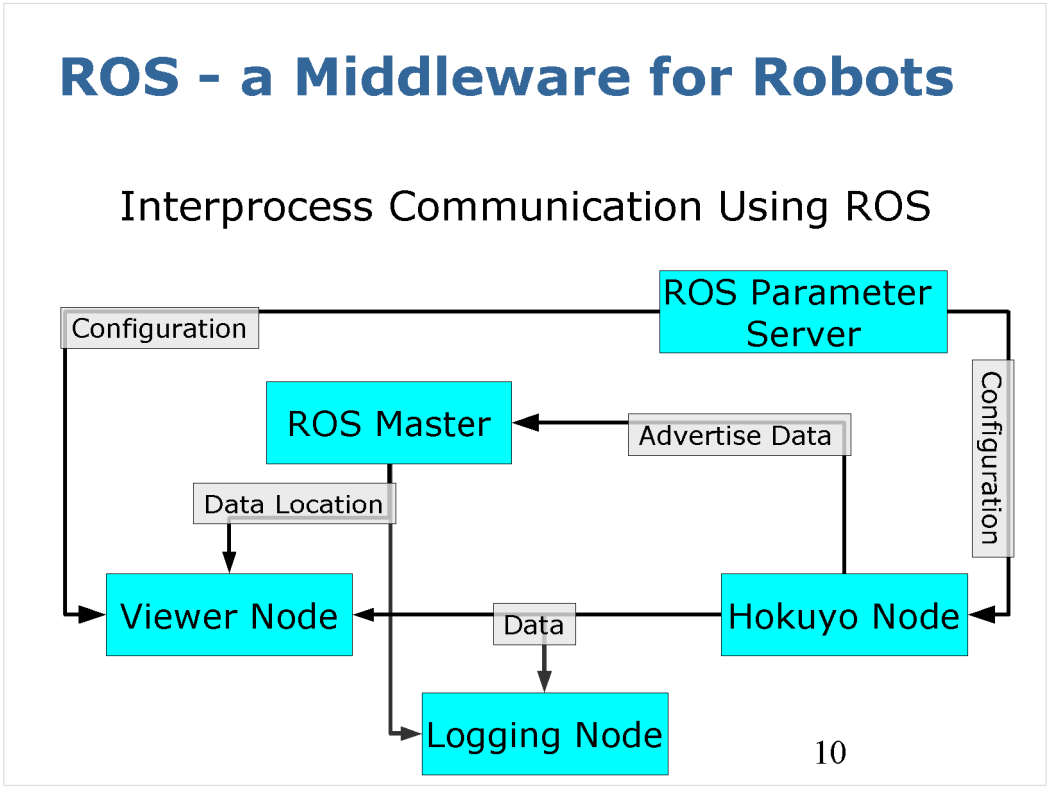


9

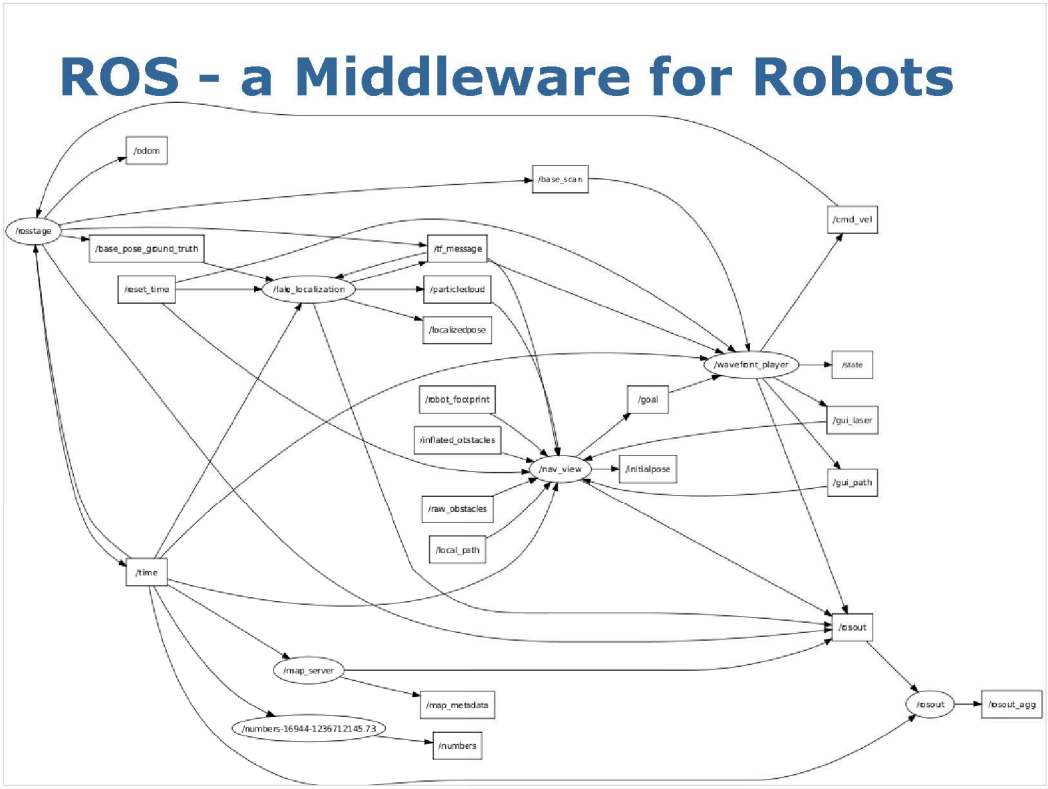
The programmer needs to know what information (the topic) it wants from whom (which program)

The ROS master provides the means to find and connect to those programs.

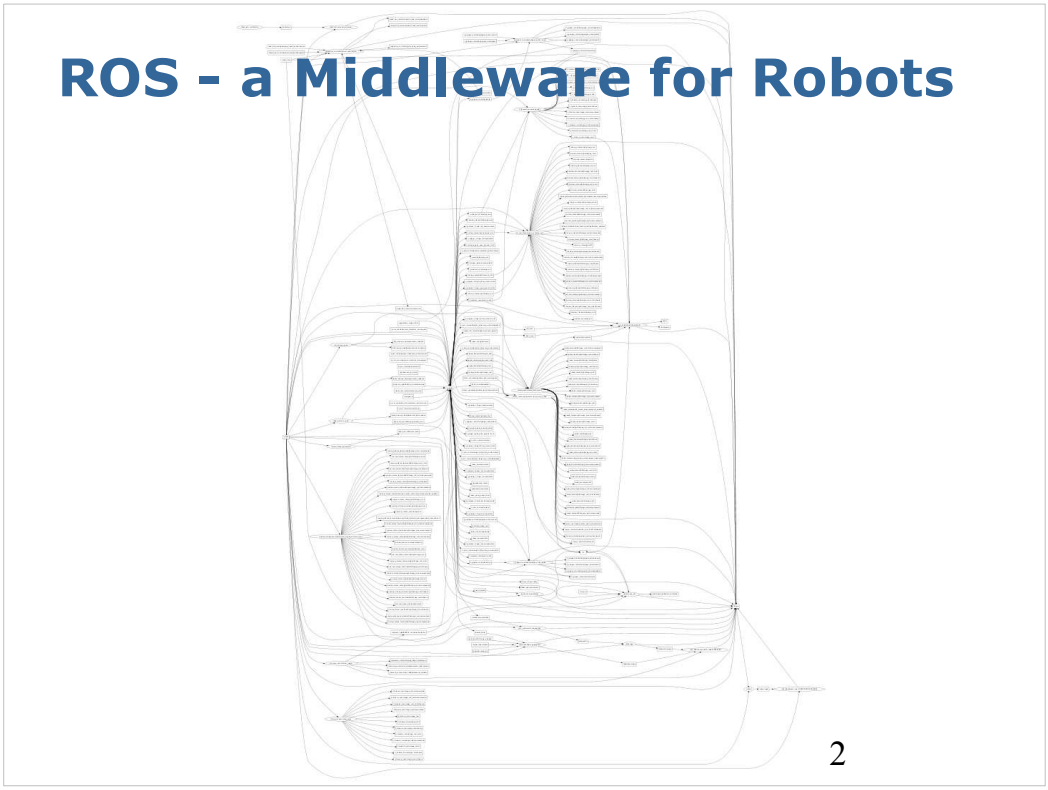
ROS also defines how interfaces (data types) are defined, and provides functionality for serialization and transportation between different nodes (programs). Nodes can be located on different computers and written in different programming languages.



This diagram is more abstract and includes the parameter server, that is used as a central location to configure the behaviour of nodes. The logging node at the bottom can save data for later offline (i.e. without the sensor) playback. As this example shows, topics can be directed to more than one receiver. Also more than one sender could send out data on a topic.



This is the output of *rxgraph*, a ROS tool to visualize the communication network. Ellipses represent nodes and rectangles represent topics. Note that there are topics with more than one sender (e.g. rosout at the bottom right) and more than one subscriber (e.g. time on the lower left). Also some topics have no subscribers and some have no senders.



On a real robot, the communication network soon becomes complex

ROS - a Middleware for Robots

There is more to it...

- (De-)Serialization under the hood
- Central multi-level logging facilities
- Service calls, preemptible actions
- Central time

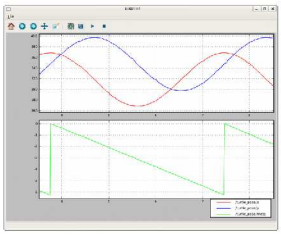
The Serialization procedures even provide a feature checking for compatibility of the message definition.

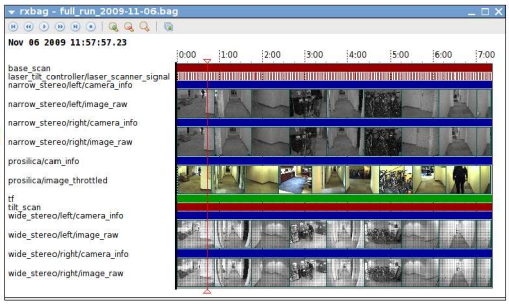
The central time is important in this context, as it lets you play back real data from some other point in time while in simulation.

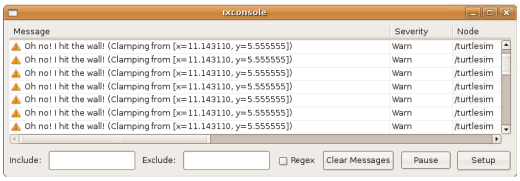
ROS - a Middleware for Robots

There is more to it...

- Tools for analysis, debugging and visualization of IPC
- Live message view
- Recording and playback
- Many more...







Examples for the tools are *rxgraph* (see last slide), *rostopic*, *rosservice*, *rosnode*, *rosparam*, *rxconsole*, *rxplot*

Recording and playback of data can be achieved with *rosbag:*ros.org/wiki/rosbag

Agenda of the Overview Talk

- High Level Overview
  - What is ROS?
  - Who made it?
  - Why do we use it?
- Middleware Aspects
- Software Structure
- Client Libraries

15

Software Structure

The first slide of the middleware section states how a middleware helps to build modular software for easier reuse of components. Also it states that the number of required components can grow tremendously.

While the modularization protects us from highly complex monolithic applications that are difficult to understand, debug, adapt or reuse, the complexity of the task is still present in the number of components. Therefore, as in operating systems, it is important that the software components are structured in a comprehensible way.

Client Libraries

The last section will explain what makes a program a ROS-node.

ROS – Software Distribution

Software Hierarchy

- Release – collection of stacks and packages
- Stacks – a full application suite
- Package – software (and interface definition) to solve a specific task
- Node – An executable with some useful functionality
- Message/Service/Action – Interface definitions

16

A rough Comparison:

KDE 4.3 – Release

Kontact – Stack

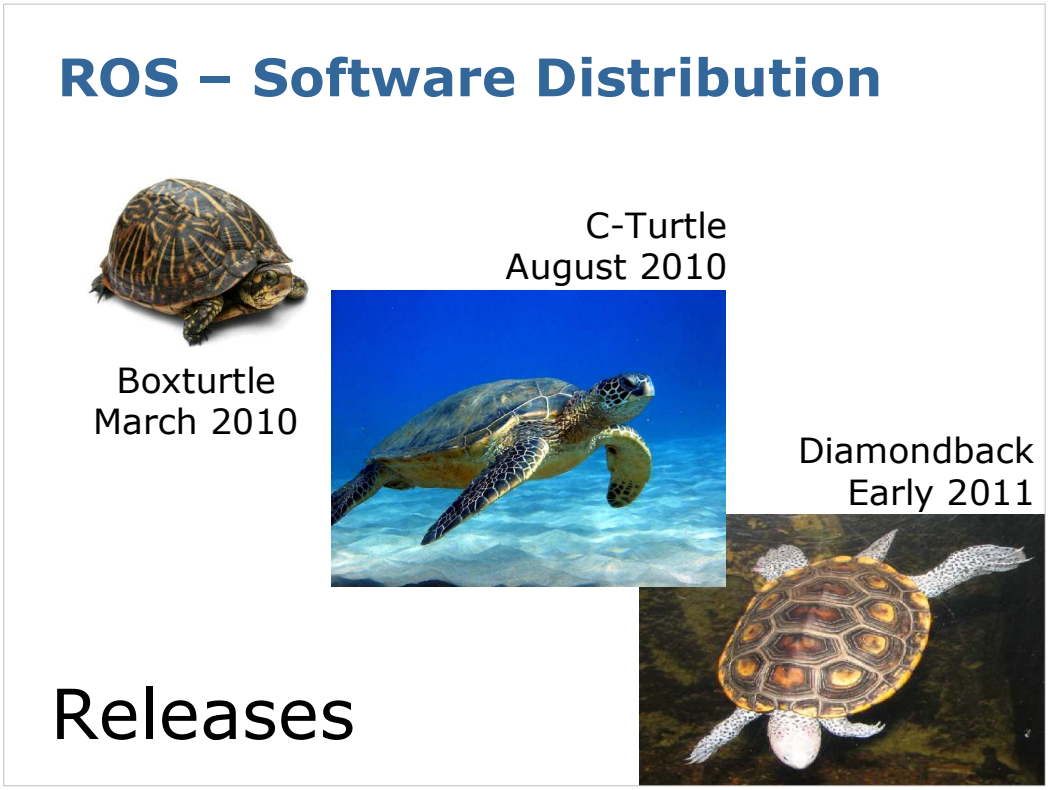
Kmail – Package

IMAP-plugin – Node

IMAP RFC – Message/Service Definition

In contrast to desktop application development, separable components are not integrated into a monolithic binary (or loaded as plugins), but implemented as individual executables that communciate via the middleware. This is more related to the unix philosophy of having small specialized tools, that perform a particular task very well, e.g. ls, sed, wget, tar...

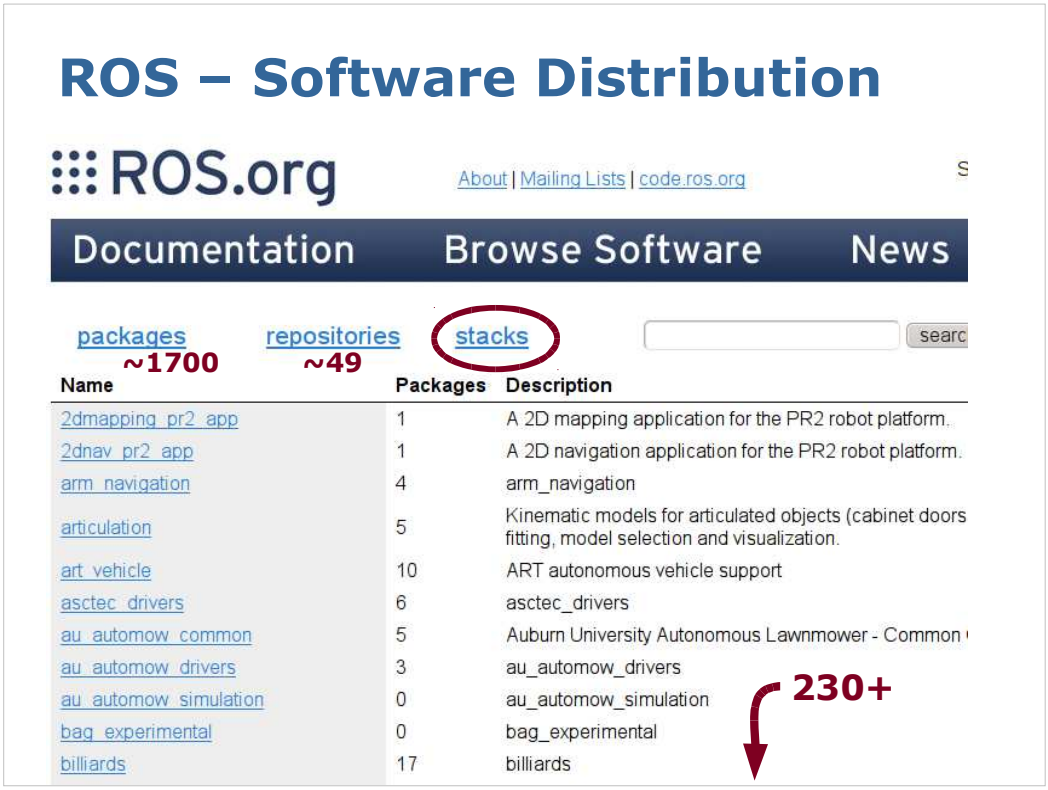
This makes it easier to exchange small parts to adapted the software to a new task or a new state-of-the-art in research.



From ros.org (http://www.ros.org/wiki/ROS/Introduction)

Releases

The core ROS system, along with useful tools and libraries are regularly released as a ROS Distribution. This distribution is similar to a Linux distribution and provides a set of compatible software for others to use and build upon. There are currently two releases: C Turtle (August 2010) and Box Turtle (March 2010).



Examples for ROS Applications:

- Visualization
- Calibration
- Simulation
- Synchronous Localization and Mapping (SLAM)
- Visual SLAM
- Articulated Objects
- Object Recognition
- Grasping
- PR2 billiard
- PR2 “beer me”

ROS – Software Distribution

- Binary distribution via dpkg/apt
- Source distribution via version control systems
- Tools for dependency resolution
  - Fetch binaries from apt-repository
  - Fetch source code from version control
  - Recursively build dependencies
- Central documentation wiki: ros.org/wiki

Installation instructions & download mirror:
http://ros.informatik.uni-freiburg.de

Willow Garage's ROS package repository for Ubuntu is at
http://code.ros.org/packages/ros/ubuntu

We have a mirror at

http://ros.informatik.uni-freiburg.de/packages/ros/ubuntu

See the mentioned web page for instructions.

There is a Howto for the ROS wiki: http://www.ros.org/wiki/About%20ros.org

From ros.org (http://www.ros.org/wiki/ROS/Concepts)

- Distributions: ROS Distributions are collections of versioned stacks that you can install. Distributions play a similar role to Linux distributions: they make it easier to install a collection of software, and they also maintain consistent versions across a set of software.

- Repositories: ROS relies on a federated network of code repositories, where different institutions can develop and release their own robot software components.

- The ROS Wiki: The ROS community Wiki is the main forum for documenting information about ROS. Anyone can sign up for an account and contribute their own documentation, provide corrections or updates, write tutorials, and more.

ROS – Client Libraries

- What makes a program a ROS node?
  → Usage of a client library
  - C++ and Python
  - Octave/Matlab, Lisp, Java and Lua (experimental)
  - You can implement your own
- A client library embeds a program in the ROS interprocess communication network (i.e. attaches it to the middleware)

At a minimum, a client library MUST:

* implement the slave side of the master/slave API. This involves managing an XML/RPC server, and issuing and responding to XML/RPC requests.

* handle node-to-node transport negotiation and connection setup. A client library may only support a subset of the possible transport mechanisms, most likely ROS/TCPROS

* handle transport-specific serialization and deserialization of messages.

A client library SHOULD also:

* parse command-line Remapping Arguments. A node that cannot do this will not be reconfigurable.

* Subscribe to a simulated Clock

* publish debugging messages to rosout

In addition, client libraries generally offer other features, such as:

* object representation of message types

* event loop for connection servicing

* user callback invocation on message receipt

–

From http://www.ros.org/wiki/Implementing%20Client%20Libraries

ROS – Client Libraries

How does it look like? In Python:

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('beginner_tutorials')
import rospy
from std_msgs.msg import String
```

```
def talker():
    pub = rospy.Publisher('chatter', String)
    rospy.init_node('talker')
    while not rospy.is_shutdown():
        str = "hello world %s"%rospy.get_time()
        rospy.loginfo(str)
        pub.publish(String(str))
        rospy.sleep(1.0)
```

```
if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException: pass
```

Yellow Block: Use ROS Client Library

Blue Block: Definition of the Node

Red Block: Start the Node

The first two lines with the exclamation mark are responsible for making the basic ROS functionality available for the remaining python code.

The second exclamation mark signifies the lines connecting the program to ROS

More details about programming with ROS will be given in a later talk.