ROS 1.0 – Orocos 2.0 – GenoM 3.0

Peter Soetens <peter@thesourceworks.com>  OROCOS RTT
Dejan Pangercic <dejan.pangercic@cs.tum.edu> ROS
Anthony Mallet <anthony.mallet@laas.fr> GenoM

March 12, 2010
DONOSTIA – SAN SEBASTIAN
Relation to BRICS

- Best-Practices-In-Robotics
- Part: Software Components for Robotics
  - BRICS does not pick *one* 'robotics framework'
    - Just like this session
  - BRICS brings robotics projects together
    - Just like this session
  - BRICS is trying to give robot scientist access to the best tools
    - Just like this session
About me

• 2005, PhD (OROCOS RTT) at KU Leuven, Belgium
• 4 years at FMTC, closeto/into industry
• Founded The SourceWorks in 2009 to support Orocos development
• Currently the Orocos RTT maintainer
About Dejan

- PhD Candidate at IAS Group, TU Munich
- Working on Robotic Perception and Reasoning
- ROS User and Co-developer of tum-ros-pkg

Federated Repository
About Anthony

- Research engineer at LAAS, Toulouse
- More than a decade of experience in generating software components
- Current maintainer of GenoM 2, developing GenoM 3
Why Orocos?

- Software which supports all advanced controllers for robotics and machine tools.
  - Focus: hard realtime & component based.
  - And: via interoperability with other projects!
- To provide industry-friendly licenses & support.
Why ROS?

• Simple use backed-up by powerful command line tools (rosxy commands)
• More and more hardware supported
• Easy testing and scaling (rviz, rxgraph)
• Very well documented (200+ tutorials)
• It comes with a Robot (PR2 beta program)

10 x =

• tones of problems tackled
• more and more algorithms
• strong feedback and robustness
Why GenoM 3?

- Validation and supervision of applications
- Writing Platform/Framework independent code
- Code generation

- BUT: Work in progress (ready for ICRA)
Quick-Facts

• Orocos
• ROS
• GenoM
An Orocos Component’s Life Cycle

- Allows non real-time configuration and cleanup
- Starting is only allowed once configured correctly
- Extended with basic error states
Robotics Component break-down

Software component for control

Read - calculate - Write

- Provides
- Requires
- Plugin
- Flow Port

Input Ports

Data Flow

Output Ports

Data Flow

Algorithm executed by a 'thread'
OROCOS Component break-down

- Provides
- Requires
- Plugin
- Flow Port

Operations

Methods

Services Interface

maps to

C/C++ functions:
- Callbacks
- Algorithms

Dynamic functions:
- State Charts
- Program scripts

Input Ports

Output Ports

Providing & Requiring Services
OROCOS Component break-down

Operations

Methods

- Provides
- Requires
- Plugin
- Flow Port

Services Interface

Maps to

C/C++ functions:
- Callbacks
- Algorithms

Dynamic functions:
- State Charts
- Program scripts

Configuration Interface

Input Ports

Data Flow

Output Ports

Data Flow

Properties
OROCOS Component break-down

- **Services Interface**
  - **Operations**
  - **Methods**
    - Provides
    - Requires
    - Plugin
    - Flow Port

- **Input Ports**
  - Data Flow
    - C/C++ functions:
      - Callbacks
      - Algorithms
  - Dynamic functions:
    - State Charts
    - Program scripts

- **Output Ports**
  - Data Flow

- **Configuration Interface**
  - Scripting
  - Properties
  - Marshalling
ROS concepts can be divided into three layers:

- **computation graph**: how programs run (master, node, topic, service, parameter, bag)
- **file system**: how program files are organized and built (package, stack, app, msg, srv)
- **repositories**: how files are distributed online
ROS Concepts
ROS in Action – Segmentation of Planes and Clusters
(Orocos) Example Application
Problem statement:
Roslaunch file (code to be released by end of March 2010):
(http://tum-ros-pkg.svn.sourceforge.net/viewvc/tum-ros-pkg/mapping/cloud_algos)

```xml
<launch>
  <!-- Point Cloud Msg generator -->
  <node pkg="point_cloud_mapping" type="pcd_to_msg_node" name="pcd_to_msg_node" output="screen" respawn="true">
    <args>$(optenv pcd teapot_handle_side-denoise.pcd) $(optenv period 5)</args>
  </node>

  <!-- Rotational Object Estimator -->
  <node pkg="cloud_algos" type="rotational_estimation_node" name="rotational_estimation_node" output="screen" respawn="true">
    <remap from="~cloud_pcd" to="/cloud_pcd"/>
    <param name="output_cloud_outliers" type="string" value="/depth_image_triangulation_node/cloud_pcd"/>
  </node>

  <!-- Point Cloud Triangulation Program -->
  <node pkg="cloud_algos" type="depth_image_triangulation_node" name="depth_image_triangulation_node" output="screen" respawn="true"/>

  <!-- Topic Concatenator -->
  <node pkg="topic_tools" type="relay" name="relay_rotational_estimation">
    <args>/rotational_estimation_node/cloud_normals /triangular_mesh_to_vtk_node/mesh" respawn="false" output="screen"/>
  </node>

  <node pkg="topic_tools" type="relay" name="relay_depth_image_triangulation">
    <args>/depth_image_triangulation_node/cloud_triangulated /triangular_mesh_to_vtk_node/mesh" respawn="false" output="screen"/>
  </node>

  <!-- TriangularMesh to VTK Writer -->
  <node pkg="cloud_tools" type="triangular_mesh_to_vtk" name="triangular_mesh_to_vtk_node" output="screen" respawn="true">
    <param name="input_mesh_topic" type="string" value="mesh"/>
    <param name="output_vtk_file" type="string" value="object.vtk"/>
    <param name="nr_of_mesh_publishers" type="int" value="1"/>
  </node>
</launch>
```
(ROS) Example Application – Visualize Results in rviz

Courtesy Jonathan Kleinehellefort
(tum grasp simulator)
GenoM3 at a glance

Anthony Mallet, LAAS - CNRS
Toulouse, France
anthony.mallet@laas.fr

San Sebastian, March 12, 2010
- **Openrobots** - LAAS Open source software for robots
- **robotpkg** - Packaging system for Openrobots
  - About 200 packages (~120 from LAAS)
  - **GenoM** - Component generator
• 1996 - GenoM

• 2004 - GenoM2, open-source

• 2010 - GenoM3, middleware independance
- pocolibs - posix-based RT communication libraries.
Jido - mobile robot with arm

base motion
human perception
manipulation
object recognition

hardware sensors and actuators
Components are tied to a middleware

A generic component will probably look like this:
GenoM 3 - Big picture & Workflow

Highly reusable

Formal Description

Library

Just a few are needed

Generic template suited to a middleware

Component

Generated code

Middleware

Library

OS
A sample service definition

```plaintext
service AcquireImage(in bank_name, in nimages) {
    doc: "Acquire camera images";

    codel start: acquire_start(in nimages, in bank_name);
    codel main: acquire_main(outport image_bank);
    codel end: acquire_end(out nimages);
    codel inter: acquire_end();

    errors: NO_SUCH_BANK, NOT_CONFIGURED, IMAGE_GEOMETRY;
}
```