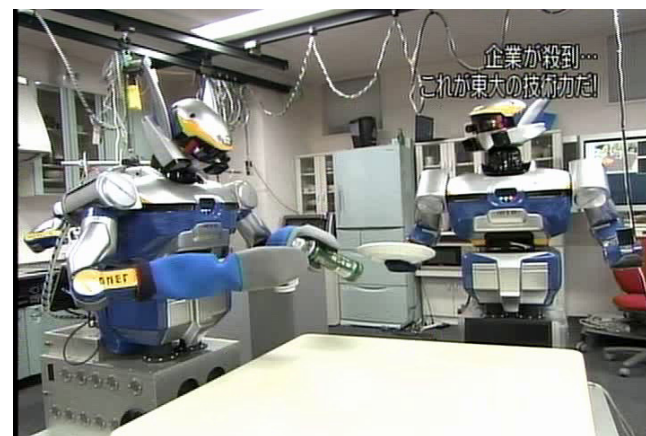


Building Interoperability between OpenRTM and ROS

The University of Tokyo / AIST

Kei Okada

k-okada@jsk.t.u-tokyo.ac.jp

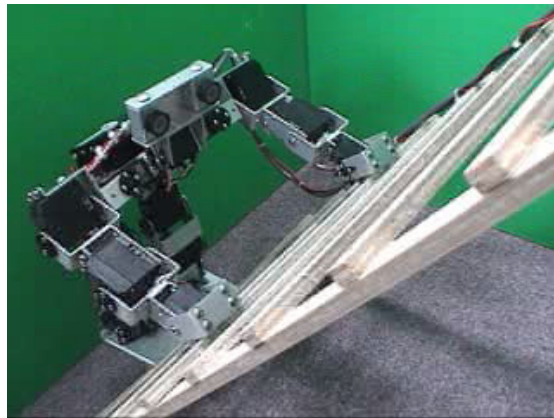


Outline

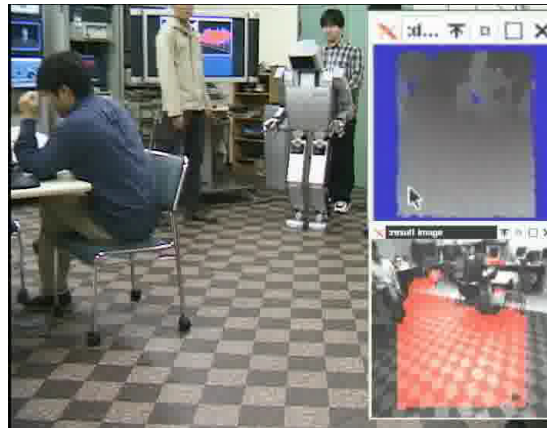
- Introduction
- OpenRTM and ROS Integration
- Using OpenRTM component for ROS users
- Using ROS node for OpenRTM users
- Conclusion

Robots in JSK Lab (2002)

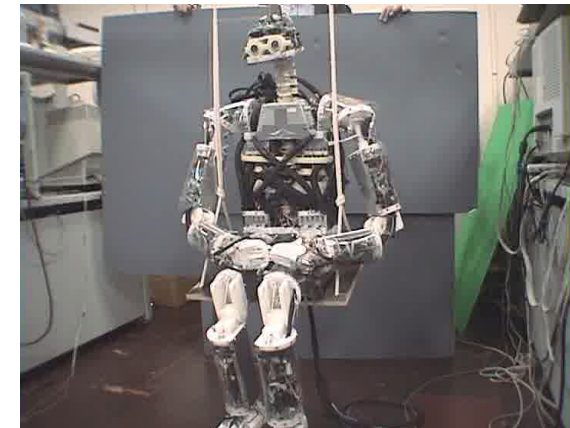
Remote-brained Robot



Life-sized Humanoid



Musculo-skeletal Humanoid



hard
ware

comp
uter

Middle-
ware

Top-level

Custom made ware

Micro computer + PC

custom made framework
(nervous: plugin-component
framework for humanoid)

Custom made hardware

Embedded PC

custom made framework
(RT-Linux kernel module →
nervous)

Custom made hardware

Distributed MPU + PC

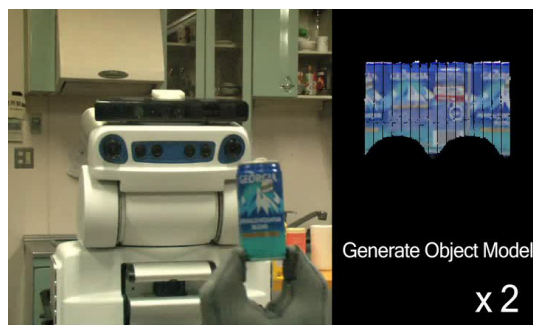
custom made framework
(nervous: plugin-component
framework for humanoid)

EusLisp robot programming environment

→ Towards uniform robotics software framework

Robots in JSK Lab (2012)

Mobile Manipulator



WillowGarage

PC Servers

ROS nodes

Life-sized Humanoid



Kawada Inc.

Embedded PC

RTM components on
OpenHRP3 and
hrpsys

Musculo-skeletal
Humanoid



Custom made

Distributed MPU + PC

Custom made
framework
(nervous)

Dynamic
Humanoid

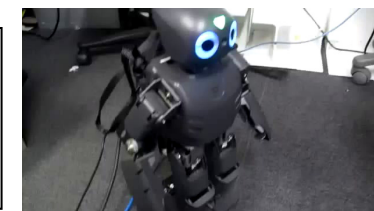


Custom made

Embed. PC

Custom made

Humanoid



ROBOTIS

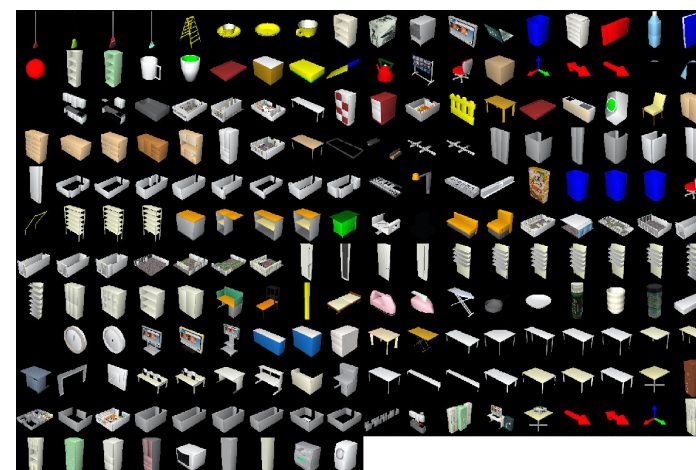
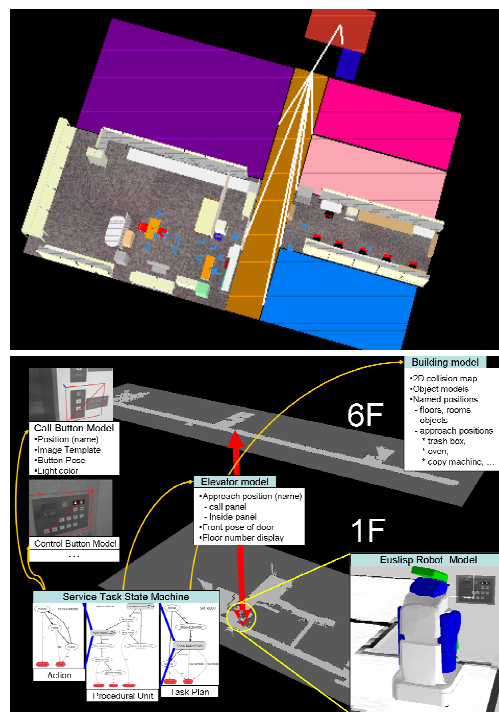
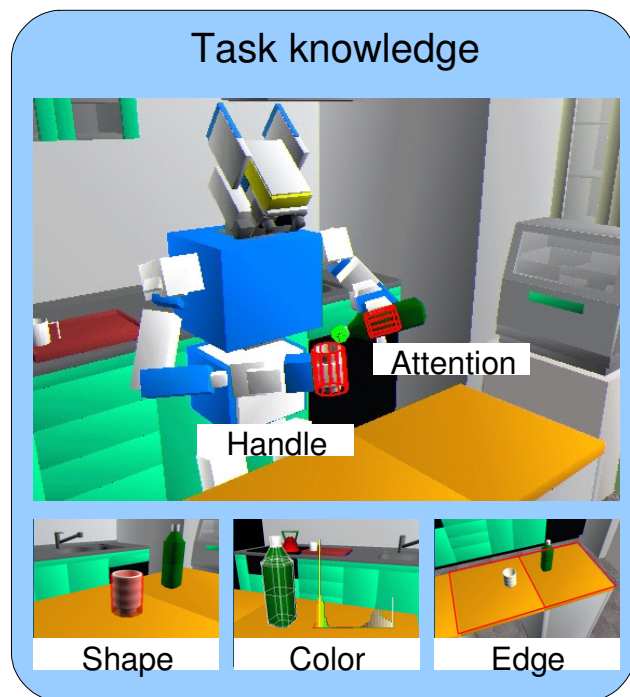
Embed. PC

Darwin-OP

EusLisp robot programming environment

EusLisp programming environment

- 1986, Toshihiro Matsui@AIST
- Interpreter with solid modeling functions
- Modeling robots and objects
+ scene or task knowledge



Daily assistive tasks of HRP2JSK



HRP2JSK(2002-)
Tasks generated
based on humanoid
motion planning
technology



Tool manipulation

Clean up

In the kitchen

Play ;-)

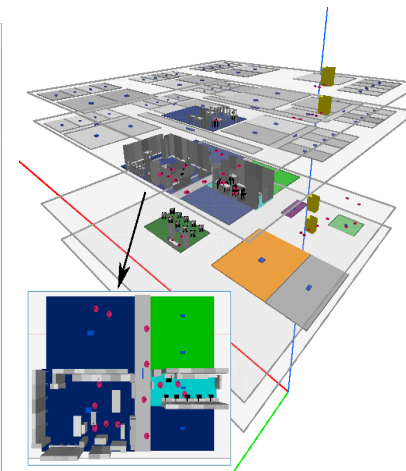
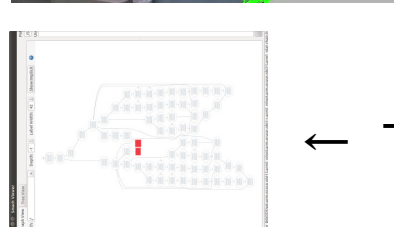
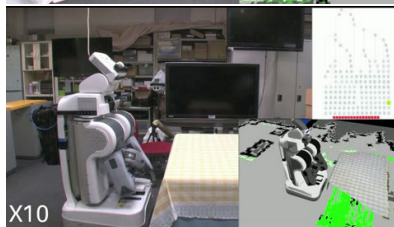
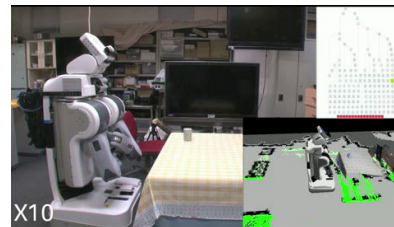
ROS based framework for integrate JSK and IAS software

PR2 getting a sandwich

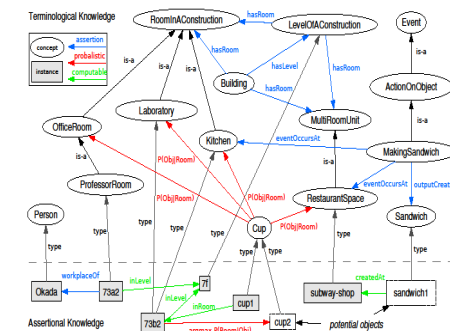
JSK, The University of Tokyo
&
IAS, Technische Universität München



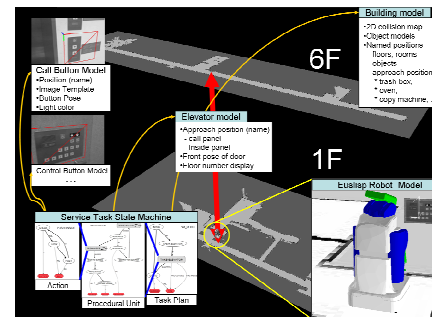
- Inferring likely location for a sandwich using web based semantic environment models



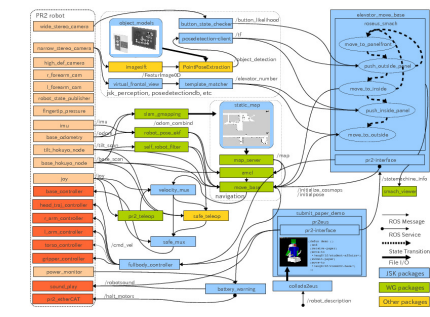
IAS semantic map



IAS knowrob map



JSK task knowledge

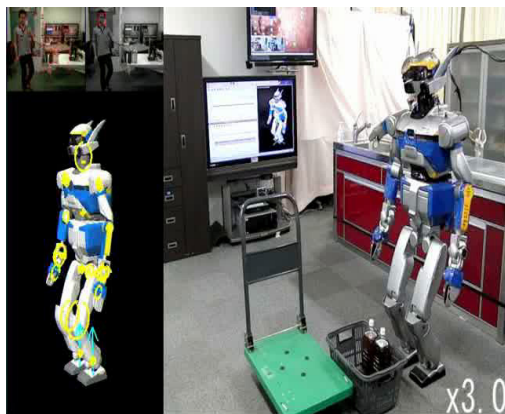
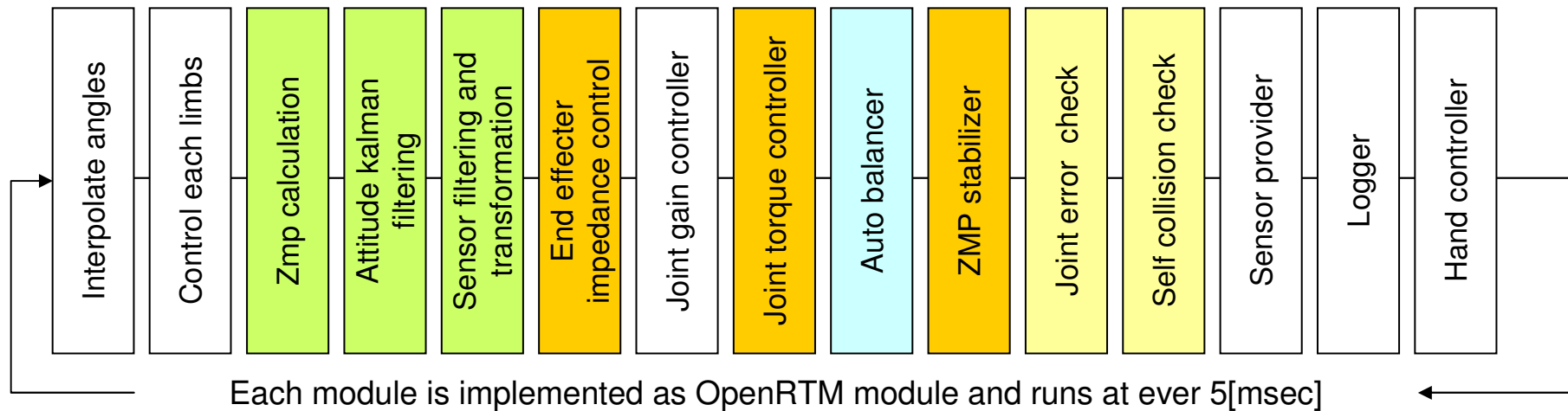


ROS nodes conn.

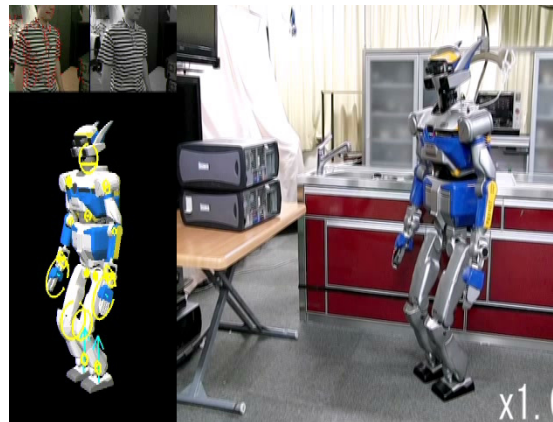
← Task planner to generate behavior executives

OpenRTM components on HRP Humanoids

Feed forward motion
Sensor feedback motion
Interface
Fail-safe softwares
Sensor data processing



Force sensor monitoring
(StateProvider, StateProvider2
ZMPsensor, Torquecontrol)



CoG modification based on Fullbody motion generation
Force sensor heedback Based on Joint Torque control
(AutoBalancer) (Torque Control)



OpenRTM and ROS integration

What is ROS exactly?

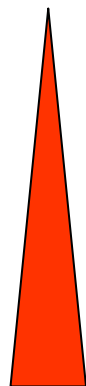
ROS = plumbing + tools + capabilities + ecosystem
B. Gerkey, Dec 06 '11. answers.ros.org

- Application
- Modules
- Library
- Simulator
- Communication
- Device Drivers
- Tools



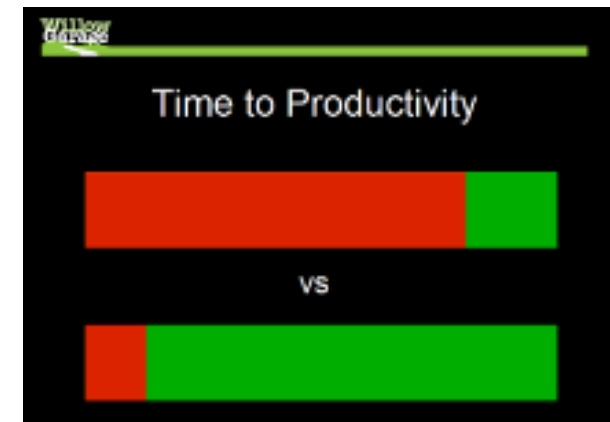
Research

Target of OpenRTM project



Tools

ROS provides extensive set in this layer



Red indicates time to build tools, and green shows the research. Current PhD student spend most of their time to build tools. ROS is designed to provide efficient tools for researchers to concentrate on the “research” (Steve Cousins speaking at Robo Development: <http://www.willowgarage.com/blog/2008/11/17/steve-cousins-speaking-robo-development-tuesday>)

→ Building OpenRTM-ROS environment on ROS-tools

- Connecting OpenRTM and modules developed in all over the world.
- Efficient development and maintenance

<http://rtm-ros-robotics.googlecode.com>

1. Compile OpenRTM component with rosmake
 - rtmbrid/cmake/rtmbuild.cmake
 - Compile and link OpenRTM idl files
2. Invoke OpenRTM component with roslaunch
 - openrtm/scripts/rtmlaunch.py
 - Add <rtconnect> and <rtactivate> tag to support component management in launch file.

CMakeLists.txt example (hrpsys/CMakeLists.txt)

```
cmake_minimum_required(VERSION 2.4.6)
include($ENV{ROS_ROOT}/core/rosbuild/rosbuild.cmake)
rosbuild_find_ros_package(rtmbrid)
include(${rtmbuild_PACKAGE_PATH}/cmake/rtmbuild.cmake) # include rtmbuild.cmake
rtmbuild_init() # this calls rosbuild_init() inside
rtmbuild_genidl() # compile idl
rtmbuild_add_executable(AbsTransformToPosRpy src/AbsTransformToPosRpy.cpp
                        src/AbsTransformToPosRpyComp.cpp) # compile component
```

Launch file example (openrtm/launch/MobileRobot_2DSimulator_Example.launch)

```
<node name="rtmlaunch" pkg="openrtm" type="rtmlaunch.py" args="$(find
  openrtm)/launch/MobileRobot_2DSimulator_Example.launch" />
<rtconnect from="TkJoyStick0.rtc:pos" to="TkMobileRobotSimulator0.rtc:vel" />
<rtactivate component="TkJoyStick0.rtc" />
<rtactivate component="TkMobileRobotSimulator0.rtc" />
```

OpenRTM-ROS Example

```
# Install RTMROS environment
```

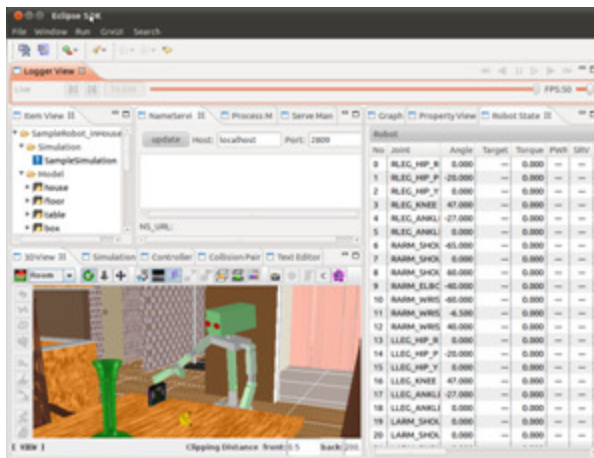
```
$ rosinstall ~/prog/rtm-ros-robotics http://rtm-ros-robotics.googlecode.com/svn/tags/latest/agentsystem\_ros\_tutorials/rtm-ros-robotics.rosinstall
```

```
# Compile RTMROS environment
```

```
$ rosmake hrpsys_ros_bridge
```

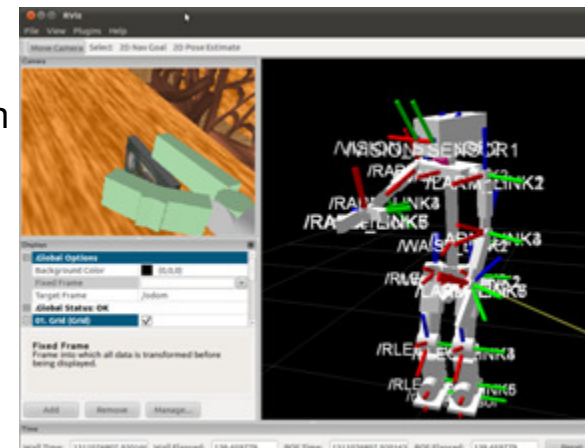
```
# Launch RTMROS environment
```

```
$ roslaunch hrpsys_ros_brikdge samplerobot.launch
```



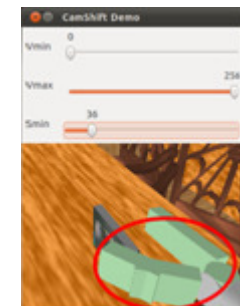
RTM

- Dynamics simulation
- Robot Model
- Controller
- Sequence
- Sensor Holder



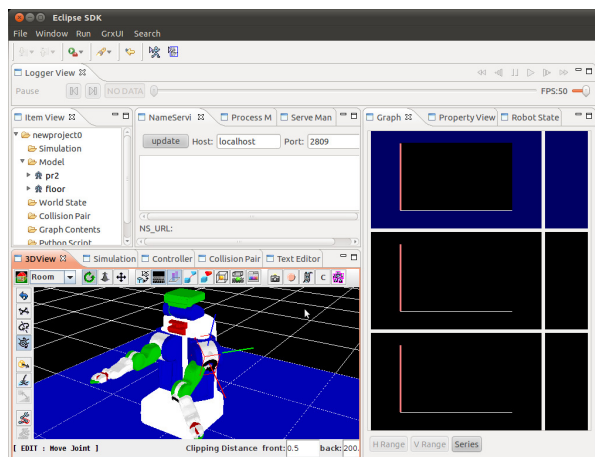
ROS

- Sensor Viewer
- Image Processin

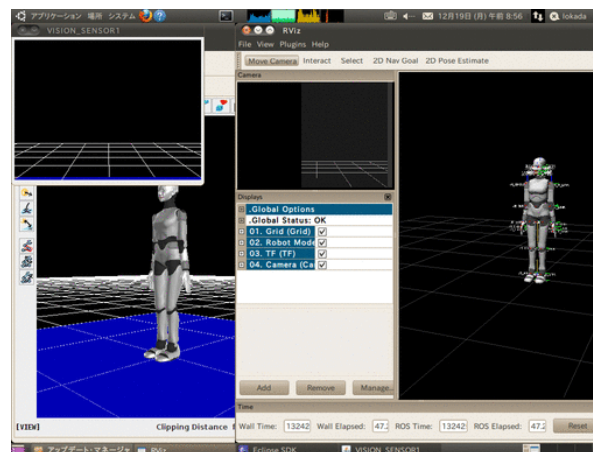


Interoperability of Robot Model between OpenRTM and ROS

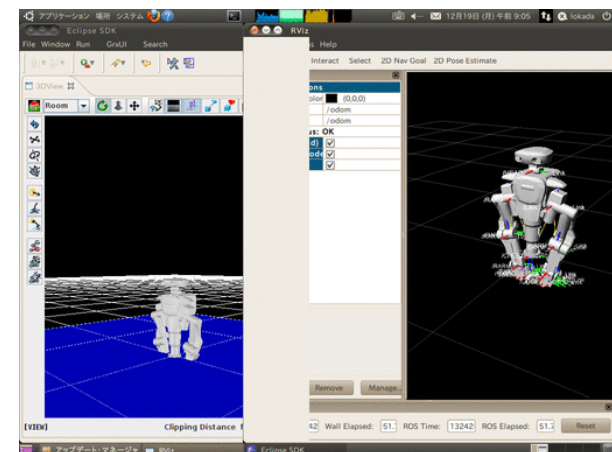
- Collada support on both OpenRTM and ROS
 - Convert OpenRTM (OpenHRP3 wrl format) to Collada (openhrop3/export-collada)
 - Convert ROS (URDF format) to Collada (robot_model support collada natively)
- ROS package for OpenRTM software (OpenHRP3, hrpsys, Choreonoid, OpenVGR, OpenINVENT)



PR2 in OpenHRP3 simulator



Simulating in OpenHRP3 and display the robots in rviz



Using OpenRTM for ROS users

- `rtmbuild.cmake` provides following cmake macros

```
rtmbuild_init()  
rtmbuild_genidl  
rtmbuild_genbridge()
```

- Compile IDL file into ROS service messages file and OpenRTM/ROS source file

```
module SimpleService {  
  typedef sequence<string> EchoList;  
  typedef sequence<float> ValueList;  
  interface MyService  
  {  
    string echo(in string msg);  
    EchoList get_echo_history();  
    void set_value(in float value);  
    float get_value();  
    ValueList get_value_history();  
  };  
};
```

SimpleService IDL File



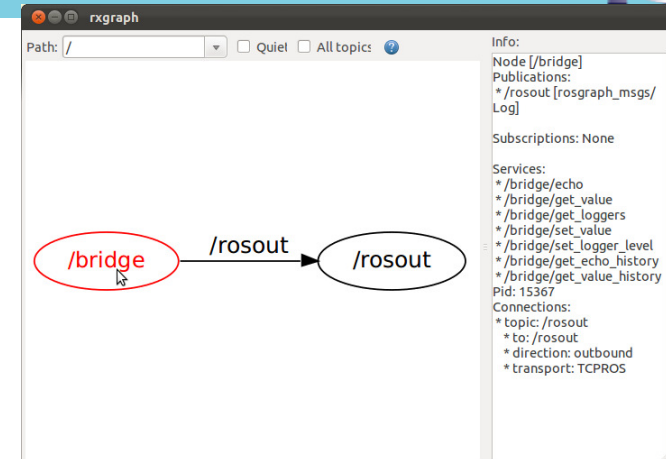
```
$ ls rc_gen/  
MyServiceROSBridge.cpp  
MyServiceROSBridge.h  
MyServiceROSBridgeComp.cpp
```

```
$ ls srv  
SimpleService_MyService_echo.srv  
SimpleService_MyService_get_echo_history.srv  
SimpleService_MyService_get_value.srv  
SimpleService_MyService_get_value_hisotory.srv  
SimpleService_MyService_set_value.srv
```

Generated Service messages

SimpleService example

- Calling OpenRTM Simple Service example from rosservice



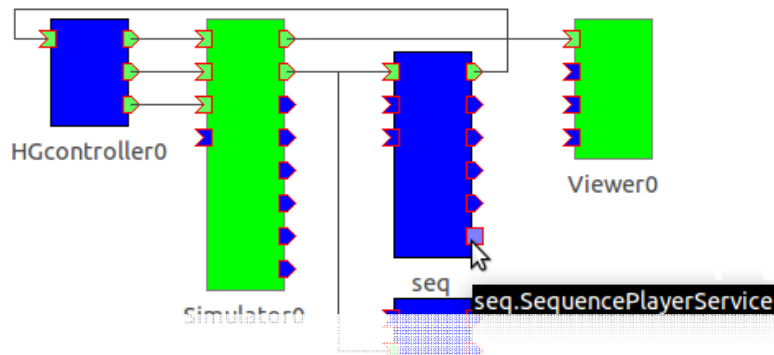
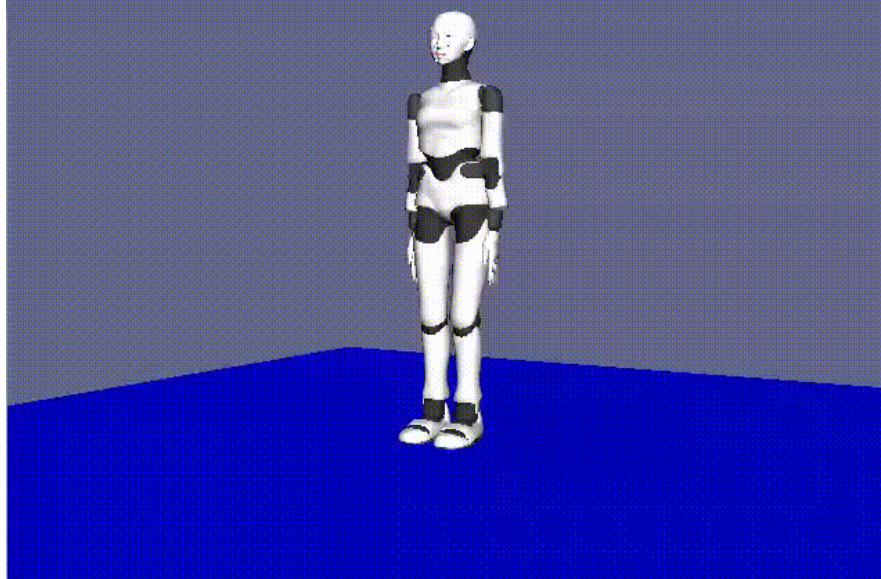
The screenshot shows the RT System Editor interface with the following components:

- Command list (left):** A list of commands for the `MyServiceConsumerComp` and `MyServiceProviderComp` components. The `MyServiceConsumerComp` commands include `echo [msg]`, `set_value [value]`, `get_value`, `get_echo_history`, and `get_value_history`. The `MyServiceProviderComp` commands include `echo [msg]`, `set_value [value]`, `get_value`, `get_echo_history`, and `get_value_history`.
- System Diagram (center):** A diagram showing the components `MyServiceConsumer0`, `MyServiceProvider0`, and `MyServiceROSBridge0` connected in a loop.
- Properties (right):** A table showing the properties of the `MyServiceROSBridge0` component.
- Terminal (bottom):** A terminal window showing the execution of the `rosservice list` and `rosservice call` commands.

Property	Value
Path URI	localhost/M
Instance Name	MyServiceRi
Type Name	MyServiceRi
Description	
Version	1.0.0
Vendor	
Category	
State	ACTIVE
owned	
ExecutionConf	
ID	0
State	RUNNING
Kind	PERIODIC

```
k-okada@kokada-t420s:~$ rosservice list
/bridge/echo
/bridge/get_echo_history
/bridge/get_loggers
/bridge/get_value
/bridge/get_value_history
/bridge/set_logger_level
/bridge/set_value
/rosout/get_loggers
/rosout/set_logger_level
k-okada@kokada-t420s:~$ rosservice call /bridge/echo 'hello, this is echo sample'
```

OpenRTM simulation



Launch OpenRTM humanoid example

```
$ roslaunch hrpsys hrp4c.launch
```

```
$ rosrn openhrp3 rtsysedit.sh
```

Launch ROS-OpenRTM bridge component

```
$ roslaunch hrpsys hrp4c-rosbridge.launch
```

Example code to send walking pattern from rosservice command

```
$ rosservice call /SequencePlayerServiceROSBridgeComp/loadPattern "`rospack find hrpsys`/share/hrpsys/samples/HRP-4C/data/walk2m" 1
```

```
$ rosservice call /SequencePlayerServiceROSBridgeComp/waitInterpolation
```

RTM/ROS Bridge Component/Node

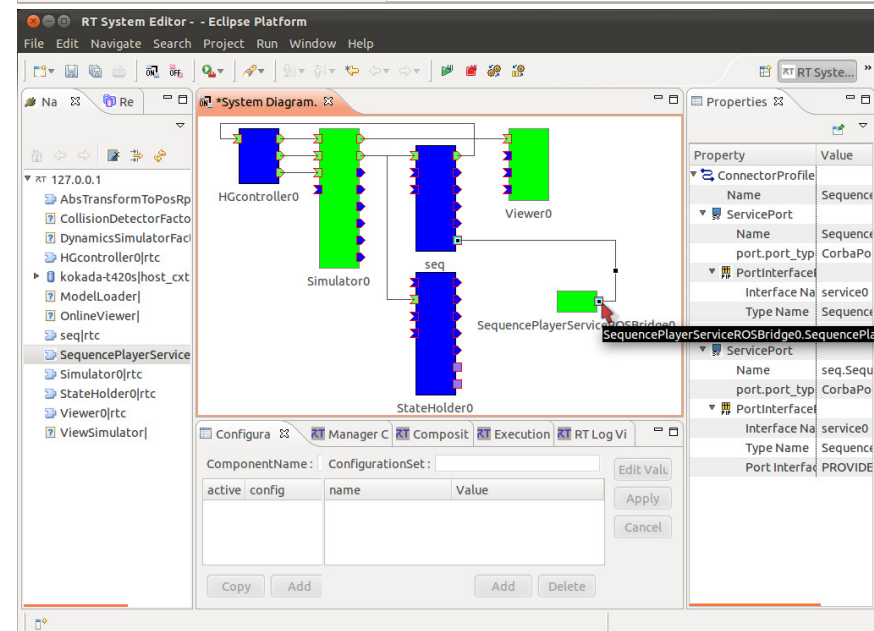
Publications:
* /rosout [rosgraph_msgs/Log]

Subscriptions: None

Services:
* /SequencePlayerServiceROSBridgeComp/setInterpolationMode
* /SequencePlayerServiceROSBridgeComp/setJointAngles
* /SequencePlayerServiceROSBridgeComp/loadPattern
* /SequencePlayerServiceROSBridgeComp/get_loggers
* /SequencePlayerServiceROSBridgeComp/isEmpty
* /SequencePlayerServiceROSBridgeComp/playPattern
* /SequencePlayerServiceROSBridgeComp/setBaseRpy
* /SequencePlayerServiceROSBridgeComp/setJointAngle
* /SequencePlayerServiceROSBridgeComp/setBasePos
* /SequencePlayerServiceROSBridgeComp/setZmp
* /SequencePlayerServiceROSBridgeComp/clear
* /SequencePlayerServiceROSBridgeComp/waitInterpolation
* /SequencePlayerServiceROSBridgeComp/clearNoWait
* /SequencePlayerServiceROSBridgeComp/setJointAnglesWithMask
* /SequencePlayerServiceROSBridgeComp/set_logger_level
Pid: 6818

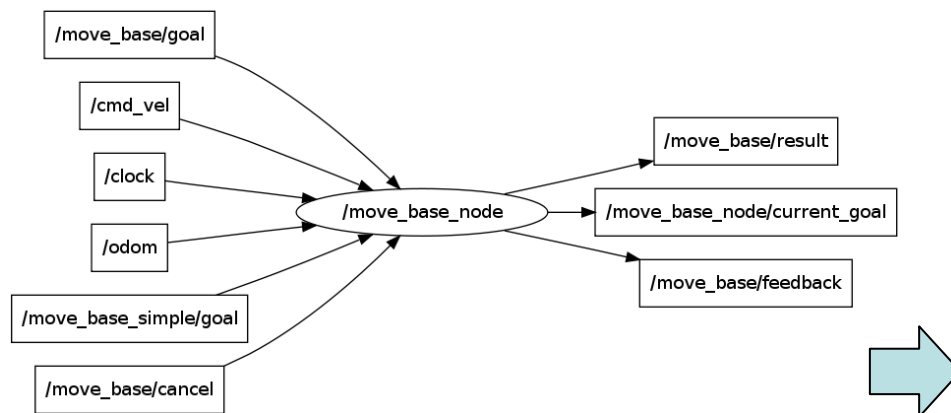
Connections:

* topic: /rosout
* to: /rosout
* direction: outbound

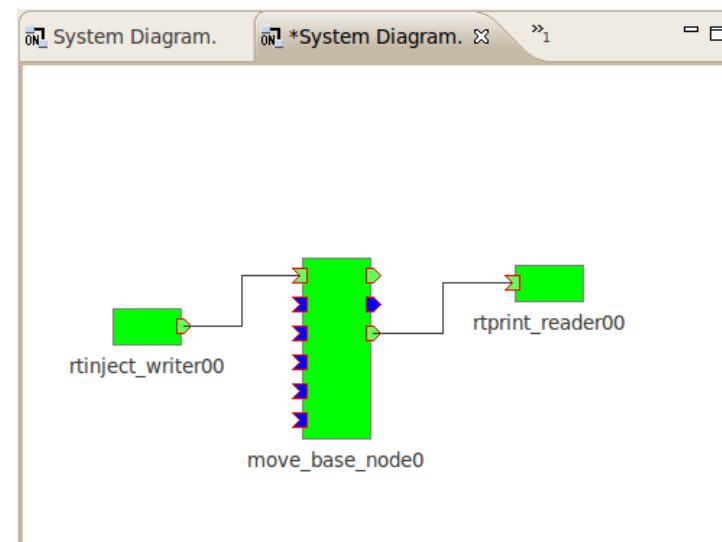


Using ROS node for OpenRTM users

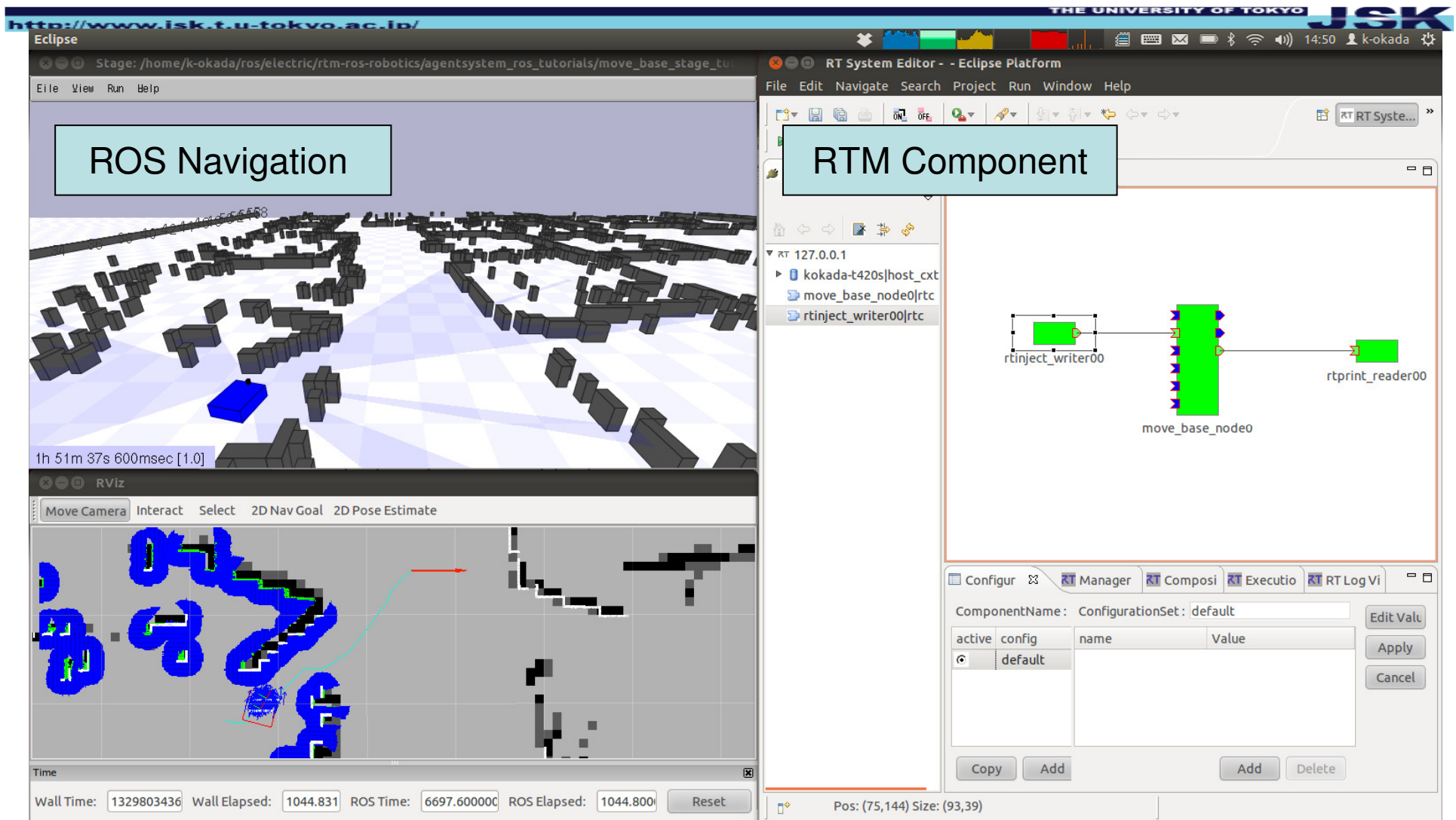
- Generate OpenRTM component from ROS node in operation
 - \$ `roslaunch rosnode_rtc stage_sample.launch`
 - Analyze input/output message of the ROS node
 - Generate corresponding IDL file for OpenRTM
 - Create OpenRTM component with the IDL file to provide transformation between ROS message and OpenRTM data port



ROS navigation node



Auto generated OpenRTM component, that provides interface to ROS navigation node



Launch ROS navigation tutorial

```
$ roslaunch move_base_stage_tutorial robot.launch
```

```
$ rosrn rviz rviz -d `rospack find movebase_stage_tutorial`/config/rviz.vcg
```

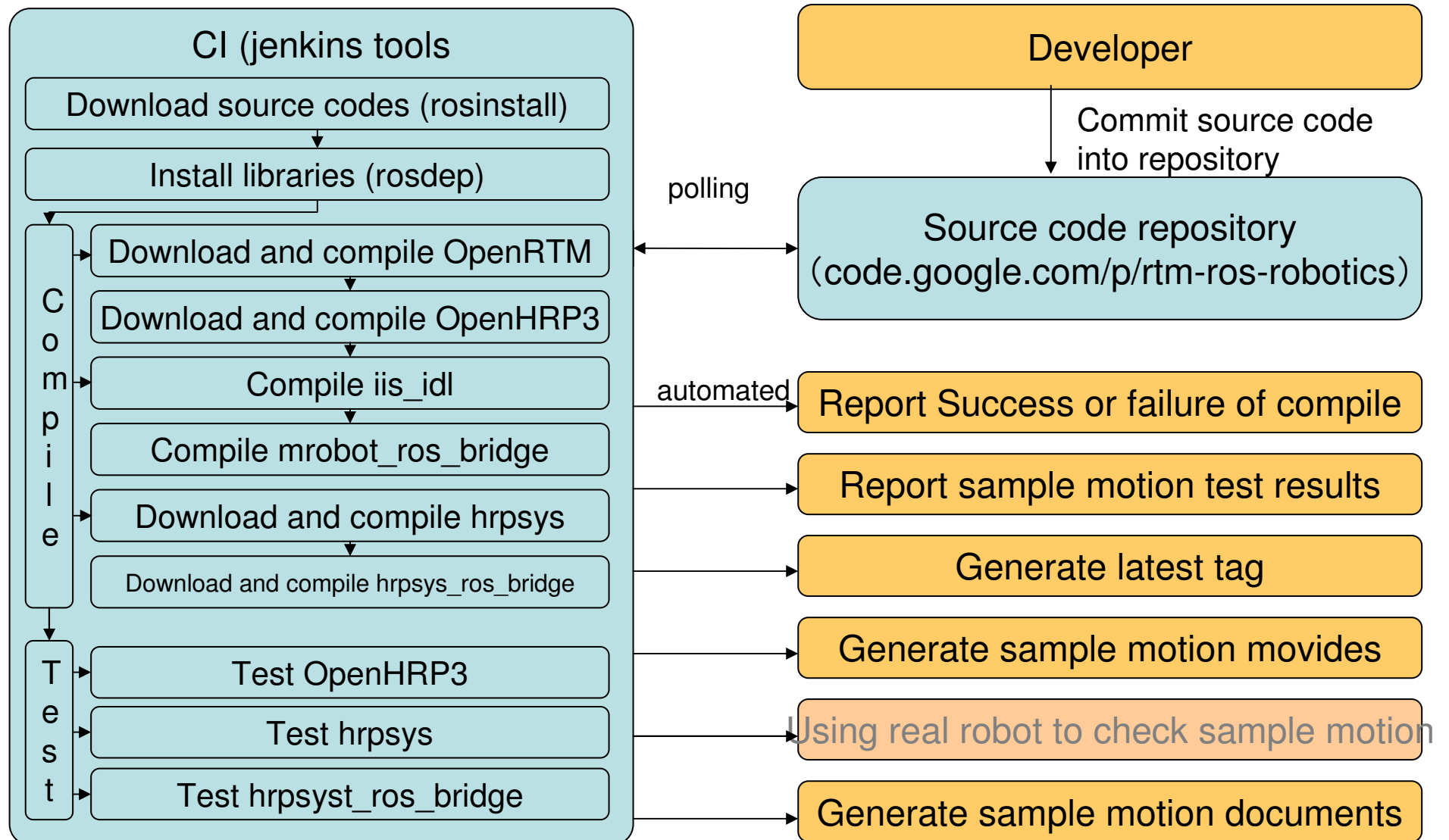
Generate OpenRTM component from ROS node

```
$ roslaunch rosnode_rtc stage_sample.launch
```

Example code to control ROS navigation through RTM data port

```
$ rosrn rosnode_rtc stage_sample_send_goal.sh
```

Documentation



Auto roslaunch documentation

- Add extra <sphinxdoc> tags into launch file that automatically generate documentation.

```
<launch>
  <include file=" ... " />
  <include file=" ... " />
  <sphinxdoc><![CDATA[
.. code-block:: bash
  rosrun roseus roseus `rospack find
  hrpsys_ros_bridge`/scripts/hrp4c-pickup.1
```

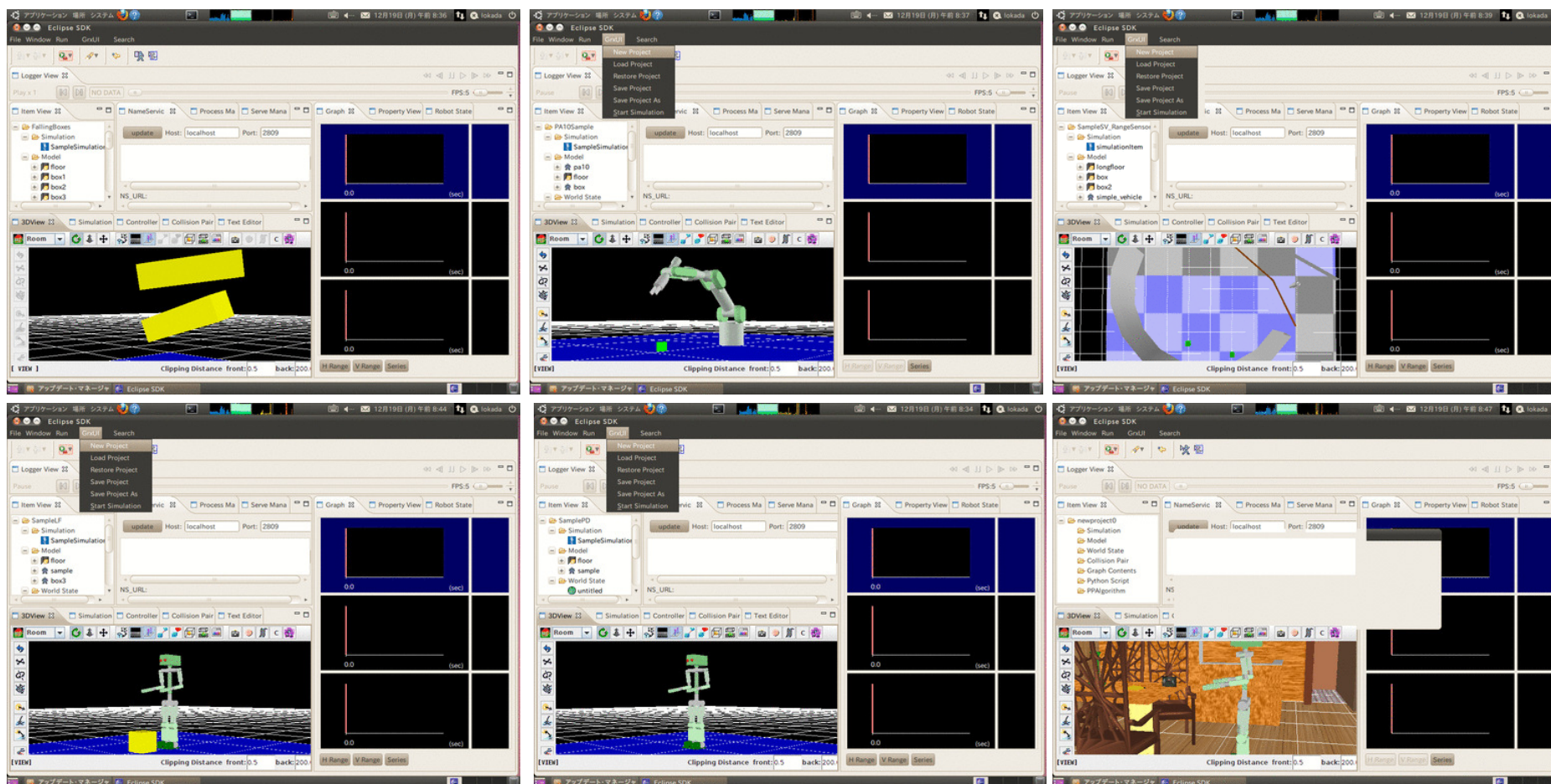
This launch file shows an example of ros bridge for open
hrp4 robot...

```
.. video:: build/images/HRP4C_PickUp
  :width: 600
]]></sphinxdoc>
  <test type="test-grxui.py" pkg="openhrrp3" ... />
</launch>
```

This part is converted into HTML

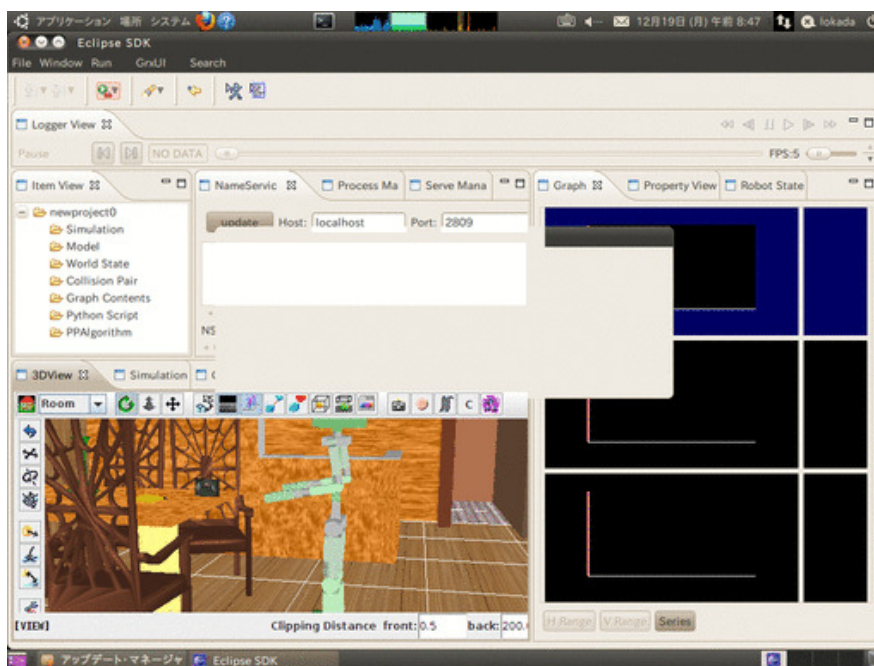
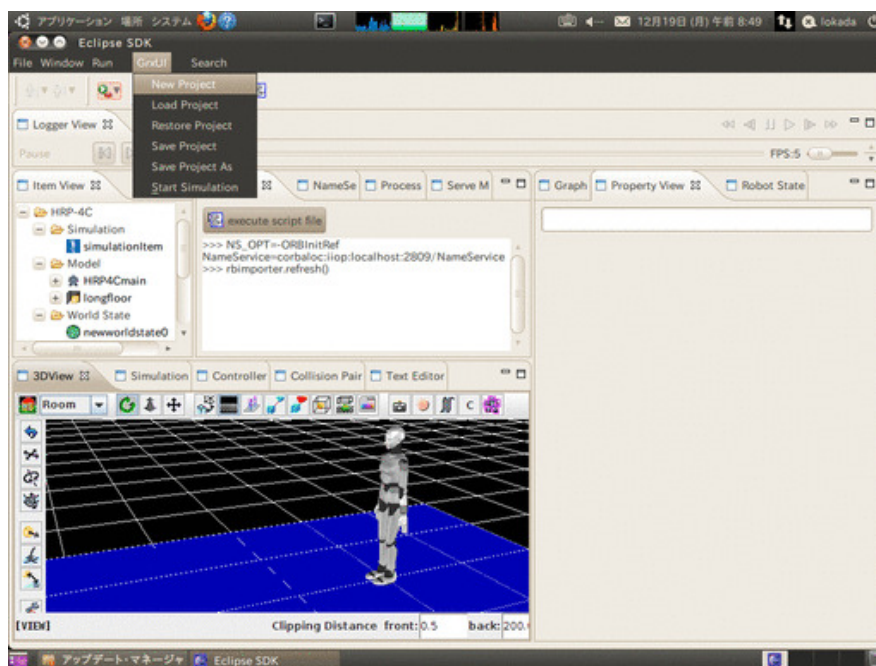
OpenHRP3 examples

- Auto-generated mp4/ogv files from test code



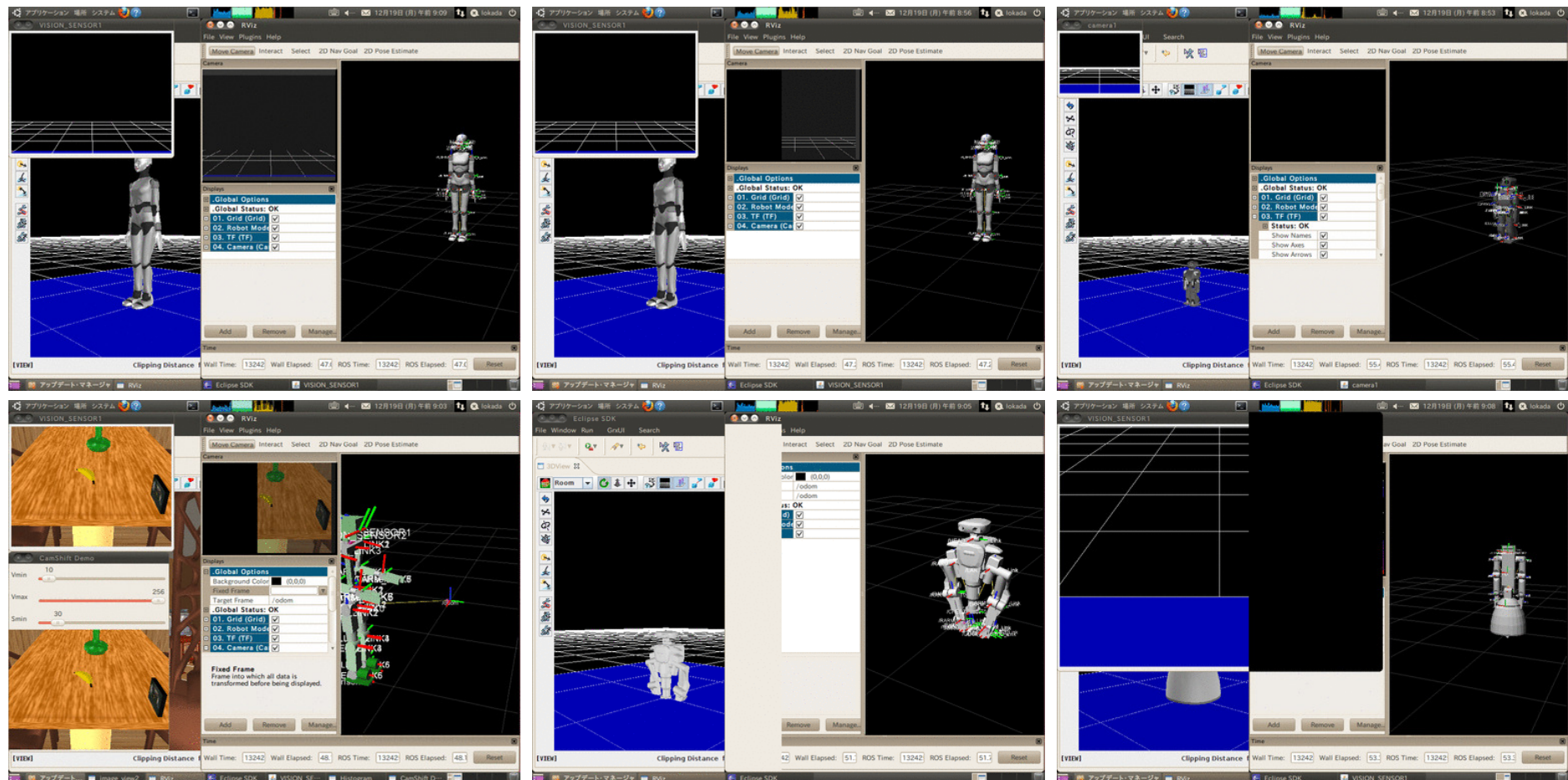
hrpsys examples

- Auto-generated mp4/ogv files from test code



hrpsys_ros_bridge examples

- Auto-generated mp4/ogv files from test code



hrp4c.launch

```
roslaunch hrpsys_ros_bridge hrp4c.launch
```

```
roslaunch hrpsys_ros_bridge hrp4c_startup.launch
```

```
roslaunch hrpsys_ros_bridge hrp4c_ros_bridge.launch
```

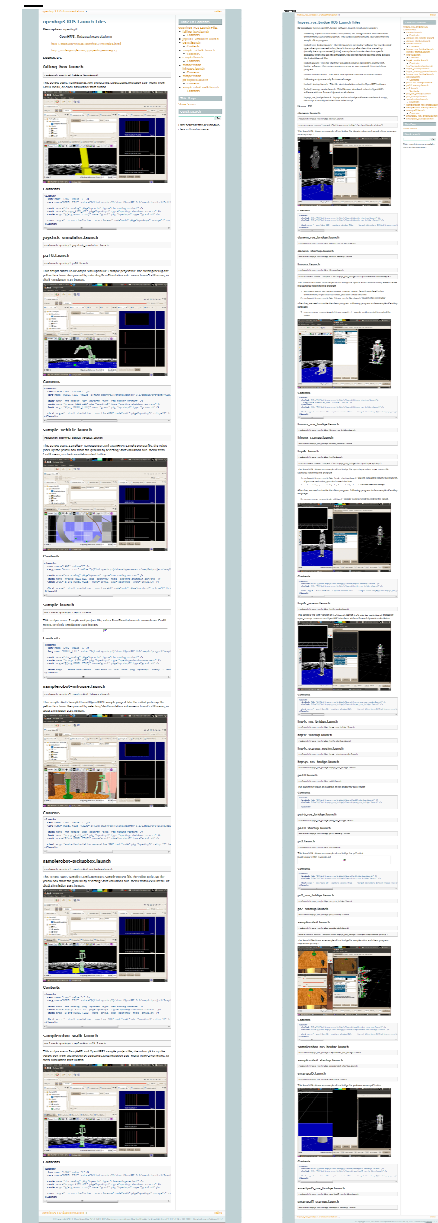
This launch file shows an example of ros bridge for open hrp4 robot, *hrp4c.launch* file launches following two program

1. `roslaunch hrpsys_ros_bridge hrp4c_startup.launch` : launch simulated robot environment, if you use real robot, you don't need this line
2. `roslaunch hrpsys_ros_bridge hrp4c_ros_bridge.launch` : launch rtm ros bridge

After that, we need to invoke the client program. Following program is the example of euslisp language

3. `roslaunch hrpsys_ros_bridge hrp4c_pickup.l` : sample euslisp script to control the robot.

The screenshot below shows the robot simulation environment (RViz) with a robot model and a 3D grid. The robot is positioned on a blue grid. The RViz interface includes a 'Global Options' panel with checkboxes for 'Global Status: OK', '01. Grid (Grid)', '02. Robot Mode', '03. TF (TF)', and '04. Camera (Ca)'. The status bar at the bottom shows 'Clipping Distance: 100', 'Wall Time: 13242', 'Wall Elapsed: 103', 'ROS Time: 13242', and 'ROS Elapsed: 103'.



Example of laundoc

<http://jenkins.jsk.imi.i.u-tokyo.ac.jp:8080/job/agentsystem-test/lastStableBuild/artifact/openhrp3-example/index.html>

<http://jenkins.jsk.imi.i.u-tokyo.ac.jp:8080/job/agentsystem-test/lastStableBuild/artifact/hrpsys-example/index.html>

http://jenkins.jsk.imi.i.u-tokyo.ac.jp:8080/job/agentsystem-test/lastStableBuild/artifact/hrpsys_ros_bridge-example/index.html

Conclusion

- Many robots, many software frameworks
- Building interoperability between OpenRTM and ROS
 - Using ROS tools to build and deploy
 - COLLADA support for robot model compatibility
 - Generating ROS node from OpenRTM component
 - Statically analyzes component (IDL) to generate service message and ros node source file
 - Generating OpenRTM component from ROS node
 - Dynamically analyzes ros node input/output message and generate OpenRTM component

svn: <http://rtm-ros-robotics.googlecode.com/svn/trunk/>

Issue: <http://code.google.com/p/rtm-ros-robotics/issues/list>

ML: <http://groups.google.com/group/rtm-ros-robotics>