OpenRDK: a modular framework for robotic software development

D. Calisi, A. Censi, L. Iocchi, D. Nardi
Software frameworks for robotics

- Software frameworks are useful
  - Promote standard design techniques
  - Aim at code reusability (components)
  - Provide ready-to-use architecture styles

- Software frameworks for robotics
  - OROCOS (EURON project)
  - CLARAty (NASA)
  - OpenRTM-aist (Japanese project)
  - Orca, Player/Stage, MARIE, MOAST, IPC, etc.
Software frameworks for robotics

- Concurrency model
  - Call-backs
  - Processes
  - Threads

- Information sharing model
  - Data ports
  - Blackboard

```c
// a callback is called by a scheduler
void callback() {
    // do your work quickly
    // and return the control
    // to the scheduler
}
```
OpenRDK main features and concepts

RAgent (Process)

Repository (Blackboard)

Module

Property1 (integer)
Property2 (string)
Property3 (image)
Property4 (map)
Property5 (user defined type)
...

session

Module

session

Repository (Blackboard)
An example

agent1

hwInterface

- laserScan
- odometry

repository

agent2

localizer

- laserScan
- odometry
- estimatedPose

mapper

- robotPose
- laserScan
- map

navigator

- robotPose
- map
- targetPose
- speed

repository

speed
Repository, properties and URLs

- Published property roles
  - Input/Output
  - Parameters
  - Information, etc.

- URL
  - Globally unique

- Types
  - Integers, strings, floating-point numbers, etc.
  - Images, maps
  - User defined types

Examples:
- rdk://agent1/hwInterface/speed
- rdk://agent1/hwInterface/odometryPose
- rdk://agent1/hwInterface/robotSerialPort
- rdk://agent1/hwInterface/currentSpeed
  ...
- rdk://agent2/localizer/laserScan
- rdk://agent2/localizer/estimatedPose
- rdk://agent2/mapper/map
- rdk://agent2/navigator/maxSpeed
- rdk://agent2/navigator/speed
  …
Queues as object dispatchers

- Addressed like other properties in the repositories
- FIFO, can contain any object
- Features
  - multiple clients (concurrently)
  - no object duplication
  - automatic garbage collection
  - filters
  - passive
- Examples
  - for localization (e.g., laser)
  - for logging
Configuration and object persistence

- Configuration for an agent
  - Is stored in a configuration file (XML)
  - Contains the list of modules to be instantiated
  - Contains properties for each module
  - Contains connections among modules

- Can be used to
  - Load parameters from configuration file
  - Save and load module states
  - Load static inputs (e.g., pre-built maps)
Property links

- **Policy**
  - A module reads from and writes to its own properties

- **Property links**
  - Connect inputs and outputs of different modules
  - Can refer to foreign repositories
  - Are stored in the configuration file

```
rdk://agent1/hwInterface/laserScan
rdk://agent1/hwInterface/odometry

rdk://agent2/localizer/odomentry
rdk://agent2/localizer/laserScan
rdk://agent2/localizer/estimatedPose

rdk://agent2/mapper/robotPose
rdk://agent2/mapper/laserScan
rdk://agent2/mapper/map

rdk://agent2/navigator/map
rdk://agent2/navigator/robotPose
rdk://agent2/navigator/targetPose
rdk://agent2/navigator/speed
```
Property sharing

- A module, through links, can access foreign properties

- Repository actions:
  - Requests properties
  - Publishes them locally

- Options:
  - When to send updates
  - Network protocol

- Data reconstruction layer
  - E.g. maps, images, etc.

```
rdk://agent1/hwInterface/laserScan
rdk://agent1/hwInterface/odometry
```

```
rdk://agent2/localizer/odometry
rdk://agent2/localizer/laserScan
rdk://agent2/localizer/estimatedPose
```
Tools and other utilities

- Modules for logging and replaying

- Profiling (work in progress)

- Modules to connect to simulators
  - Stage and Gazebo (through Player)
  - USARSim
  - Webots (work in progress)
Tools: RConsole

- Video
Case study: concurrent engineering

- We have to develop the localizer, mapper and navigator modules

- We have three students: we assign each student to the development of one module

- We have a set of pre-developed modules (e.g., hwInterface to connect to robot sensors and actuators)

- First step: interface design (inputs/outputs)

<table>
<thead>
<tr>
<th>hwInterface</th>
<th>localizer</th>
<th>mapper</th>
<th>navigator</th>
</tr>
</thead>
<tbody>
<tr>
<td>speed (in)</td>
<td>odometry (in, queue)</td>
<td>robotPose (in)</td>
<td>robotPose (in)</td>
</tr>
<tr>
<td>laserScan (out, queue)</td>
<td>laserScan (in, queue)</td>
<td>laserScan (in)</td>
<td>targetPose (in)</td>
</tr>
<tr>
<td>odometry (out, queue)</td>
<td>estimatedPose (out, queue)</td>
<td>map (out)</td>
<td>map (in)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case study: the localizer module

- Using logging/replaying during development
  - Save a log of a single run and then replay it
  - Use well-known logs databases (e.g., RAWSEEDS)
Case study: the mapper module

- Using odometry instead of estimated pose
- Using simulator (e.g., Stage, USARSim, etc.)
Case study: the navigator module

- As before
  - Use odometry, simulator clients

- Pre-built map from configuration file

- RConsole for target poses input

MAP
(configuration file)

usarsimClient/speed (in)
usarsimClient/laserScan (out, queue)
usarsimClient/odometry (out, queue)

localizer/odometry (in, queue)
localizer/laserScan (in, queue)
localizer/estimatedPose (out, queue)

mapper/robotPose (in, queue)
mapper/laserScan (in, queue)
mapper/map (out)

navigator/robotPose (in)
navigator/targetPose (in)
navigator/map (in)
navigator/speed (out)
OpenRDK current applications

- Rescue wheeled robots (real robots, USARSim)
- RoboCare project (assistive robots for the elders)
- Quadrotor, tarantula (real robots, USARSim)
- RoboCup Standard Platform League (“Nao league”) 
- HRI experiments (robot side)
Conclusions

- OpenRDK features
  - Full multi-thread support
  - Blackboard-style communication
  - Distributed systems
  - Modularity (code reuse without writing a line)
  - Tools (Logging/replaying, RConsole, etc.)
  - Concurrent engineering
  - Open source (GPL license)
On-going and future work

- Extend the property sharing mechanism
  - More network QoS (e.g., from DDS: latency budget)

- On-line fault detection system

- Configuration file editing and analysis tools
  - Detect possible deadlocks
  - Verify constraints on schedule
Questions

Questions?

We are on SourceForge

http://openrdk.sourceforge.net