Component-based Robotics Middleware

Software Development and Integration in Robotics (SDIR V)
Tutorial on Component-based Robotics Engineering

2010 IEEE International Conference on Robotics and Automation (ICRA)

A. Shakhimardanov and N. Hochgeschwender

Bonn-Rhine-Sieg University of Applied Sciences, Germany
Department of Computer Science
nico.hochgeschwender@h-brs.de
azamat.shakhimardanov@h-brs.de

May 8, 2010
Overview

1. Motivation
2. Use Case
3. Component-based Robotics Middleware
4. Conclusion and Discussion
Motivation

Challenges in building robotic applications:

- Heterogeneity
- Distribution
- Integration of legacy code (e.g. vendor-specific drivers and algorithms)
- Quality of Service (e.g. real-time requirements)
- ...

Other domains (e.g. avionics) are also faced with this challenges.

But:

- Robotics is an experimental science,
- we do not (always) have write-compile-restart cycles
- interaction with the environment is unique in robotics

⇒ Technology is needed to tackle this challenges
Motivation

Paradigm shift (composition of components):

- **Integration** over from scratch developments

⇒ (Again) Technology is **needed** to tackle this challenges
Motivation

The marketplace of robotic software systems:

The robotics community developed software systems (frameworks) to deal with this challenges.

Examples:

- ROS
- YARP
- Orocos
- Orca2
- Player
- OpenRTM-aist
- ...

\[http://wiki.robot-standards.org/index.php/Main_PAGE\]
The marketplace of robotic software systems and their users:

- Diverse robotics software systems and their users (e.g. RoboCup@Home)

<table>
<thead>
<tr>
<th>b-it-bots</th>
<th>NimbRo</th>
<th>homer</th>
<th>RadicalDudes</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICE/ROS</td>
<td>ROS/self-made</td>
<td>self-made</td>
<td>YARP</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Why** have you chosen framework A or B?
- **What** are the **features** do you need?
Motivation

Features provided by robotic software systems:

Categorization of features according to three classes:

1. Robotic software framework (e.g. tools and robot-specific functionality)
2. Drivers and algorithms (DA)
3. Communication middleware (CM)

Note: RSF ≠ CM

---

Supplementary Material:


Component-based robotics middleware (CRM):

A robotic software systems following the component-based paradigm (please refer to part 1).
Objectives

Objectives of part 2:
Acquire an overview
Understand the terminology
Understand the fundamental concepts
Estimate drawbacks and advantages

Approach:
A use case will serve as a subject of explanation and assessment
Use Case

Objective:
Component-driven development of a robot which does self-localization.

Components:
- **Sonar**: provides distance data on request
- **Laser**: provides distance data on request
- **Localizer**: requires distance data and provides position data within a frame of 1 Hz
- **Visualizer**: requires the position data to visualize robot’s path

Tasks for the developer:
- **Task 1** *Reuse* component(s)
- **Task 2** *Implement* component(s)
- **Task 3** *Compose* component(s)
Use Case

Composed system:

Requirements for component-based robotics middleware:

Component level versus system requirements

1. Communication (one-way and two-way semantics)
2. Directionality
3. Component execution (periodic and aperiodic)
4. Location transparency, and more
Requirements

Categorization of the requirements according to four concerns:

- Computation
- Communication
- Configuration
- Coordination

Design goal(s) for component-based robotics middleware:

- Separation of Concerns (SoC)!
- Availability of **means** for the application developer to deal with this concerns
Requirements

Computation:

Computation is concerned with the data processing algorithms required by an application\(^a\).


Example(s):

1. The Localizer component must provide every second position data.
2. The periodicity of the Localizer must be defined at design-time.
Requirements

Communication:

Communication deals with the exchange of data\(^a\).

\(^a\)Radestock, M. and Eisenbach, S.: Coordination in evolving systems

Example(s):

1. The Sonar component must provide distance data on request.
2. The Localizer component needs to 'know' the Sonar component in order to request data.
Configuration:

Configuration determines which system components should exist, and how they are inter-connected\(^a\).


Example(s):

1. The apex angle of the Laser component must be configured before laser-scans are retrievable.

2. The topology of the application (Sonar, Laser, Localizer,...) must be defined.
Requirements

Coordination:

Coordination is concerned with the interaction of the various system components\(^a\).


Example(s):

1. The Laser component must exist and run before the Localizer component is able to request distance data.
Realization of the Use Case

Realization:

- We ‘implement’ the use case with different component-based robotics middleware
- Focus on the communication middleware
Realization of the Use Case: Computation

Computation:

Computation is concerned with the data processing algorithms required by an application

Spotlight: ROS

Developed by WillowGarage (see ros.org or go upstairs ;-)).

<table>
<thead>
<tr>
<th>Language(s)</th>
<th>License(s)</th>
<th>OS(s)</th>
<th>CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>BSD</td>
<td>Linux</td>
<td>XML-RPC</td>
</tr>
<tr>
<td>Python</td>
<td></td>
<td>Mac OS X</td>
<td>self-made</td>
</tr>
<tr>
<td>(Java)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Realization of the Use Case: Computation

Example(s):
The Localizer component must provide every second position data.

Spotlight: ROS
- Algorithms are embedded in nodes (stand-alone processes)
- No (explicit) execution logic
- **But**: ROS provides means in terms of an API for several execution models, e.g. periodicity
  ```
  ros::Rate r(1);
  while (condition){
    publishPositionData();
    ros::spinOnce();
    r.sleep();
  }
  ```
Realization of the Use Case: Computation

Spotlight: Orocos

- Developed at KU Leuven (see orocos.org).
- Orocos includes three main libraries:
  - Real-time toolkit (RTT)
  - Kinematics and dynamics (KDL)
  - Bayesian filtering library (BFL)

<table>
<thead>
<tr>
<th>Language(s)</th>
<th>License(s)</th>
<th>OS(s)</th>
<th>CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>GPL/LGPL</td>
<td>Linux</td>
<td>CORBA</td>
</tr>
</tbody>
</table>
Realization of the Use Case: Computation

Example(s):

The Localizer component must provide every second position data.

Spotlight: Orocos

- RTT provides a C++ class framework to develop components
- The TaskContext class defines the environment in which an algorithm is embedded
- Execution explicitly defined
- `mytask.setActivity( new PeriodicActivity( ... ) );`
Realization of the Use Case: Computation

<table>
<thead>
<tr>
<th>CRM</th>
<th>Component</th>
<th>Execution logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROS</td>
<td>stand-alone process</td>
<td>not explicit</td>
</tr>
<tr>
<td>Orocos</td>
<td>shared-object</td>
<td>explicit</td>
</tr>
<tr>
<td></td>
<td>executed as a thread</td>
<td></td>
</tr>
</tbody>
</table>
Realization of the Use Case: Communication

Communication

Communication deals with the exchange of data

Example(s):

The Sonar component must provide distance data on request.

Spotlight: Orocos

Commands: Are used to instruct components (synchronous).

Methods: Are used to provide a specific functionality (synchronous).

Events: Are used to inform components when a particular change occurs (asynchronous).

Data Ports: Are used to communicate data (asynchronous).
Realization of the Use Case: Communication

**Communication**

*Communication deals with the exchange of data*

**Example(s):**

The Sonar component must provide distance data on request.

**Spotlight: ROS**

*Topics:* Are used to name messages which are published (asynchronous).

*Services:* Are used to provide a specific functionality through a request/reply message (synchronous).
Realization of the Use Case: Communication

Communication in summary:

<table>
<thead>
<tr>
<th>CRM</th>
<th>two-way</th>
<th>one-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROS</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Orocos</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Orca2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Coarse-grained versus fine-grained means of communication
- Be *aware* of the framework-specific paradigm
Realization of the Use Case: Configuration

Configuration

Configuration determines which system components should exist, and how they are inter-connected.\(^a\)

\(^a\)And more!. See example.

Example(s):

The apex angle of the Laser component must be configured before laser-scans are retrievable.

Spotlight: ROS

- Parameter Server provides a 'globally' viewable dictionary of key-value pairs
Realization of the Use Case: Configuration

**Configuration**

Configuration determines which system components should exist, and how they are inter-connected.\(^a\)

\(^a\)And more! See example.

**Example(s):**

The apex angle of the Laser component must be configured before laser-scans are retrievable.

**Spotlight: ROS**

- Parameter Server provides a 'globally' viewable dictionary of key-value pairs
Realization of the Use Case: Configuration

**Configuration**

Configuration determines which system components should exist, and how they are inter-connected.

**Example(s):**

The topology of the application must be defined.

**Spotlight: ROS**

- XML configuration file describes which node runs on which host
Example(s):
The apex angle of the Laser component must be configured before laser-scans are retrievable.

Spotlight: Orocos
Set, get, and load configuration value(s) through:
- Attributes (at run-time)
- Properties (persistent)
Realization of the Use Case: Coordination

Coordination

Coordination is concerned with the interaction of the various system components

Example(s):

The Laser component must exist and run before the Localizer component is able to request distance data.

Spotlight: Orocos

- Scripting environment to start, stop, load, and deploy components
**Communication middleware in robotic software systems:**

<table>
<thead>
<tr>
<th>Robotic Software System</th>
<th>OS(s)</th>
<th>CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROS</td>
<td>Linux, OS X</td>
<td>XML-RPC, self-made</td>
</tr>
<tr>
<td>Orca2</td>
<td>Linux</td>
<td>ICE</td>
</tr>
<tr>
<td>Orocos</td>
<td>Linux</td>
<td>CORBA</td>
</tr>
<tr>
<td>OPRoS</td>
<td>Linux</td>
<td>CORBA</td>
</tr>
<tr>
<td>OpenRTM-aist</td>
<td>Linux, Windows, OS X</td>
<td>CORBA</td>
</tr>
</tbody>
</table>
Conclusion and Discussion

Some observations:

- Mainly object-oriented communication middleware is used.
- Any alternatives? E.g. message-oriented middleware:
  - Data Distribution Service (DDS)
  - Advanced Message Queue Protocol (AMQP)
- Some services (e.g. persistence) provided by communication middleware are not available in CRM. Why?
- How to avoid a communication middleware vendor-lockin?
- Is it possible to abstract the communication middleware?
- What is about interoperability between CRMs?
- We need common data-types and mappings.
Conclusion and Discussion

Slides
The slides are available under:
http://www2.inf.h-brs.de/~nhochg2m/
Best Practice in Robotics (BRICS)

The research leading to these results has received funding from the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement no. FP7-ICT-231940-BRICS (Best Practice in Robotics).

www.best-of-robotics.org
Thank you! Questions?
Bibliography


Bibliography
