Aspects of Integrating Diverse Software into Robotic Systems

Tim Niemueller¹, Gerhard Lakemeyer¹, Alexander Ferrein²

¹RWTH Aachen University, Aachen, Germany
²FH Aachen University of Applied Sciences, Aachen, Germany

{niemueller,gerhard}@kbsg.rwth-aachen.de
ferrein@fh-aachen.de

06.05.2013
Problem Statement

▶ Ready-to-run systems come with their own framework making it hard to integrate and adopt own software components

▶ A framework always makes the user adhere the framework’s philosophy (component design, communication, logging, . . .)

▶ Integration of components into other frameworks is not easy as the framework is deeply intertwined with the code

▶ Possible solution: implement functionality in thin client libraries and provide wrappers for different frameworks
Example 1: Ready-to-run Systems
Nao and Robotino

Pros:
- Works “out-of-the-box”
- Lots of software components come with the system

Cons:
- Software architecture is often closed;
- Tailored to the software framework that comes with the robot
- Re-use of own components is tough
Example 2: Youbot and ROS

ROS:

- developing into de-facto standard robot framework
- supported by many research institutions world-wide
- loads of modules for a wide variety of robotics tasks available

- The Youbot platform is prepared to run ROS, but using it with any other framework is painful
- It is, however, easier with ROS than on with other frameworks as ROS provides efficient remote access and thin client libraries
- Examples for thin client libs in ROS: Octomap, PCL
Framework Module vs Library

Library:
- Robot library offers an API to solve a particular task
- Rather focuses on algorithmic details than on how to run things on a real robot

Framework module:
- Embedded into the framework’s software architecture
- Deploys framework’s functions w.r.t. real-time aspects
- Inverse flow of control

Thin client library:
- Enjoys features of a library
- Can be plugged easily into different frameworks
Own Experiences and Pitfalls

Until 2008:

- 2008–today: Herb 2.0, Festo’s Robotino
- Mix of legacy modules with new-developed stuff
- Integration into our latest framework FAWKES tedious
- Problem of adapting modules to changing framework infrastructure
- Problem lies mainly in
  - man-power
  - time
  - missing design patterns
Our approach FAWKES

- Component-based architecture with aspects for communication, configuration, or logging
- Each component (thread) can hook into a timed main loop
- Blackboard communication infrastructure for exchanging data between components
- Message passing mechanism for direct control flow
- Remote access to distribute components
Summary and Open Questions

- Problems exists with porting modules from one framework to another
- Functionalities should be programmed in a way that makes it easy to integrate it into different frameworks with different communication and module (infra)structure
- Possible solution: thin client libs

What are general software design patterns for thin client libs for robotic software systems?