Component-based Robotics Engineering

Davide Brugali, Luca Gherardi
Università degli Studi di Bergamo
Anchorage, AK, USA
May 7, 2010
IEEE Tutorial

- Component-based Robotics Engineering
  - Part I: Reusable building block
    - IEEE RAM, December 2009
  - Part II: Systems and Models
    - IEEE RAM, March 2010
Introduction

- TODAY: A lot of robotics software are available but often not reusable
  - They are tight to specific robot

- GOAL: build robotic software through a composition of reusable building blocks
  - called component

- Solution: Component Based Software engineering
What is reuse?

Software reuse is:

- the practice of developing software,
- from a stock of building blocks,
- so that similarities in requirements and/or architecture between applications can be exploited,
- to achieve substantial benefits in productivity, quality and business performance.
What makes a component reusable

- Three aspects are equally important

  - **Quality.** The component has to be usable (reliability, performance, efficiency,...),

  - **Functionality.** It should be desirable to reuse the component (it offer a function that is needed),

  - **Technique.** It should be possible to reuse the component (portability, interoperability, modularity).
Types of reuse

- **Horizontal component**: provide functionality that can be used in totally different use cases.
- **Vertical component**: provide functionality that can be used in a specific domain.

### Diagram

**Application components**
- HMI
- Science

**Vertical components**
- Map Builder
- Navigator
- Locomotor
- Pose Estimator

**Horizontal components**
- Device drivers
- Math library
- Middleware services
- GUI library
- Simulators

- Visual Servoing
- Remote Control

- Path Planner
- Kinematics
- Collision Checker
- Dynamics

- 15%
- 65%
- 20%
Component: a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party.
- Separation of specification and implementation
Component Specification

- Interface design concept
  - Provided/Required
  - Service/Data
  - Strongly-typed/Loosely-typed
  - Stateful/Stateless
  - Minimal/Complete

- Contract
Component framework

- A skeleton of a component implementation that can be specialized by a component developer to produce custom component.
- Stable point
- Variation point
- Variant
- Life span
Implementing variability

- Classification according to binding time
  - Compile time, Link time, Run time

- Technologies
  - Inheritance and extension
  - Aggregation and delegation
  - Parameterization
  - Conditional compilation
  - Dynamic Link Libraries
  - Reflection
A technique that aims to restructure a set of existing software libraries without affecting their external behavior in order to harmonize their architecture, data structures, and APIs.
Path planner component-framework

Davide Brugali, Luca Gherardi
Anchorage, AK, USA - May 7, 2010
Motion planning
Component-based system
Separation of functional and extra-functional requirements

Component Extra-Functional Specification

Component Functional Specification

Computation

Configuration

Container

Component Implementation

Provided Interface

Required Interface

Contract

Object

Class

Port

Resource

Service

instance
Computation

- Is concerned with the data processing algorithms
  - Data transformation -> data flow design
  - Control transformation -> control flow design

- Different levels of concurrency's granularity
  - Fine grain
  - Medium grain
  - Large grain
Granularity of control transformation

Sequential Components
- Path Planner
- Configuration Space
- Collision Checker
- Cartesian Space

Service Components
- Map Updater
- Motion Controller

Container Component
- Resource Manager
- Scheduler

Infrastructure Middleware
Components Assembly

\[ n \text{ sequential Components} + m \text{ service Components} + 1 \text{ container Component} = \text{Component Assembly} \]
Separation of functional and extra-functional requirements

Component Extra-Functional Specification

Computation

Configuration

Component Functional Specification

Provided Interface

Required Interface

Contract

Component Implementation

Container

Port

Resource

Service

Object

Class

Resource Service Port

instance
Configuration

- Determines which system components should exist, and how they are inter-connected

- A configuration is described in term of:
  - Components
  - Connection between component
  - Connectors: architectural building blocks used to model interactions among components and rules that govern those interactions
Quality of Service

- QoS depends on implementation, availability of resources and environment
Communication

- Deal with the exchange of data

- Two types of communication
  - Imperative: caller/provider mechanism
  - Reactive: broadcaster/listener mechanism

- Visibility implies dependencies and influence reusability of components.

- Three dimensions of decoupling
  - Space / Time / Synchronization decoupling
Decoupling degree between component

- Three dimensions of decoupling
  - Space decoupling
  - Time decoupling
  - Synchronization decoupling

- Communication paradigm influence the decoupling degree
Communication paradigm influence the decoupling degree

- Remote method invocation
- Publish/Subscribe
Coordination

- Is concerned with the interaction of the various system components

- Coordination language & models
  - Data-driven
  - Control-driven
Interaction between components