Middleware

Introduction from the robotics point of view

Václav Hlaváč

Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Cybernetics

Center for Machine Perception

121 35 Praha 2, Karlovo náměstí 13

Czech Republic

hlavac@fel.cvut.cz, http://cmp.felk.cvut.cz/~hlavac

Middleware in computer science

- Middleware is software providing services to applications beyond those available from the operating system.
- Middleware makes it easier for software developers to perform communication and input/output.
- Most commonly used in the context of distributed applications.
- More specifically: dash in “client-server”.
- Also used in a sense of: a software driver, an abstraction layer hiding details of hardware and software from an application.

Middleware taxonomy

1. Message oriented middleware: asynchronous store and forward application messaging.
2. Object middleware: object request brokers, manages communication between objects.
3. RPC middleware: synchronous interaction, usually within an application.
4. Database middleware: direct access to data structures allowing interaction with DB directly.
5. Transaction middleware: transaction processing as well as web application servers.
6. Portals: enterprise portal servers allowing access from user’s desktop to back end systems and services.

CORBA, my first middleware

- Common Object Request Broker Architecture (CORBA) is a standard enabling software components written in multiple computer languages and running on multiple computers to work together.
- Used in our ActIPret project (2001-2004) to control a robot with various vision sensors.
- It was to heavy and slow.
- There was a need to write a lightweight middleware allowing real-time interaction.

Middleware in robotics

- Glue software to connect software and hardware components together.
- Often, communication between components is considered to be middleware.
- The look is from the software developer perspective.
- In addition, all application-independent helping composition of subsystems into larger systems are often included too.
- Middleware should be invisible.

Four minimal primitive concepts

- Communication: components must exchange information (data, events, commands,…), and how this exchange is done is an important property of the composite system.
- Computation: each component performs certain computations that are necessary to provide the functionality that is expected from the system.
- Configuration: components should be usable in more than one possible configuration (i.e., concrete settings for each of their variable parameters).
- Coordination: at the system level. Involves: decision making, scheduling, (de)activating subsystems and/or their interconnections, etc.

Robotic middleware examples

- OpenRDK http://openrdk.sourceforge.net/
- Urbi – for complex organization of components, French company GOSTAI
- MIRO, based on CORBA,
- http://miro-middleware.berlios.de/
- OpenNI – middleware for 3D sensing, http://www.openni.org/

Middleware V. Hlaváč has some experience with

- RSB - U of Bielefeld
- ROS

Three issues to be tackled

when developing robot software:

1. Sequential programming ill-suited to asynchronous world.
2. Must manage significant complexity.
3. Details of a specific robot hardware have to be abstracted.

Ad 1. Avoiding seq. programming

Callback:

- Function that’s called whenever data is available for processing.
- Asynchronous: callback can happen anytime.

Examples:

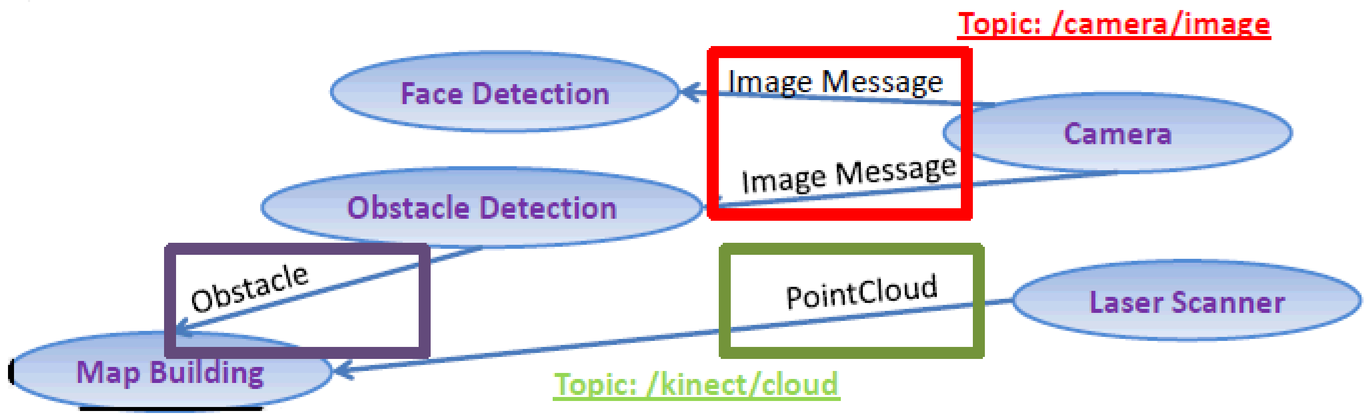
- An image is read from the camera.
- A bumper tells that the robot hit something.

Ad 2. Tackling complexity

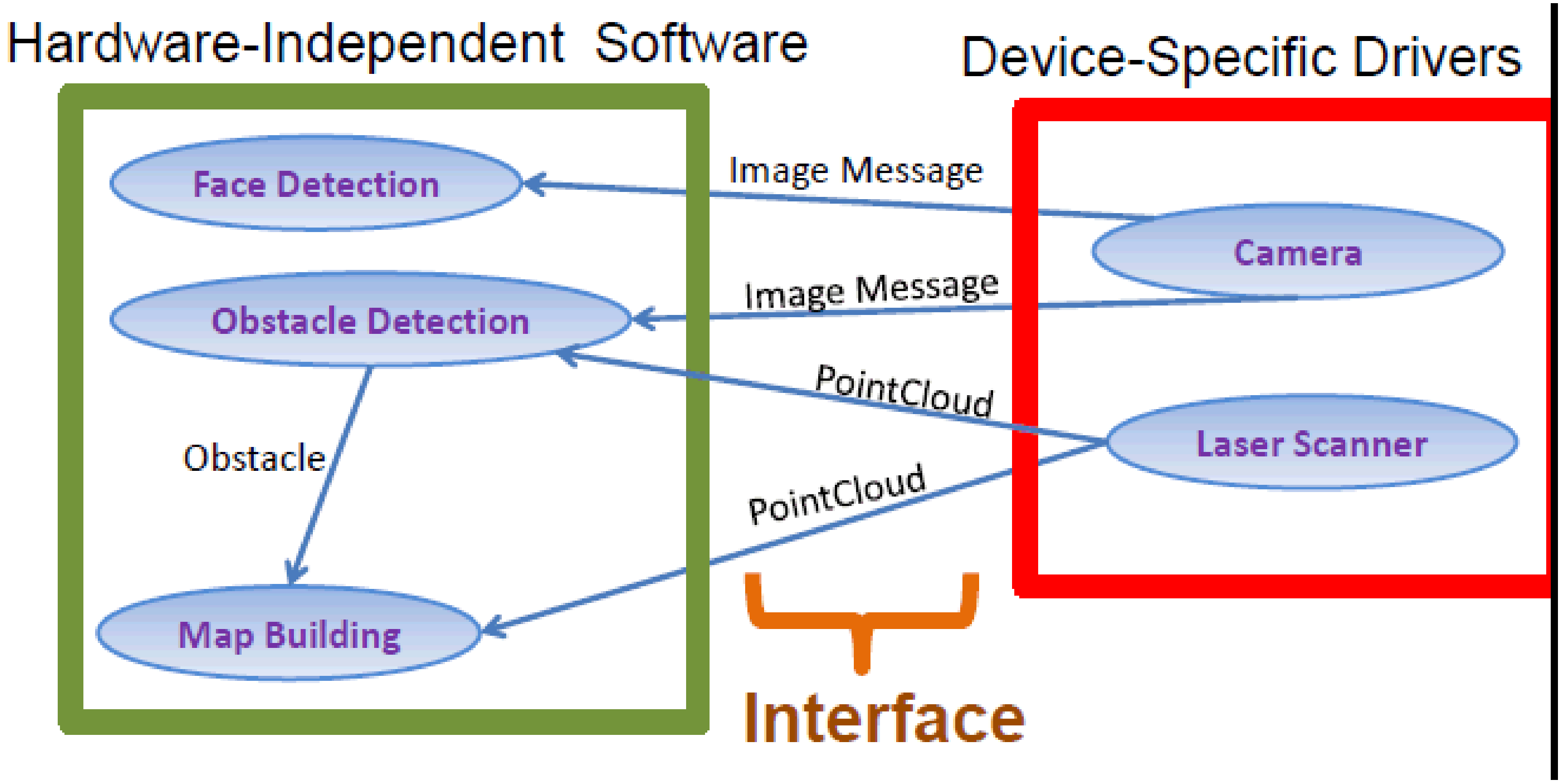
Code organization

- Separate processes: cameras, odometry, map creation …
  Can be separated out and interact through an interface.

- Interfaces: SW processes; in ROS “nodes” communicate about shared “topics”.
- Publish/subscribe: each piece of sw receives only messages it requests.



Ad 3. Hardware abstraction



ROS Robot Operating System

- Solves all three above discussed issues (callbacks, interface, hardware).
  - Initiated by Willow Garage.
  - Message passing.
  - Debugging tools.
  - Visualization tools.
  - Software Management (compiling, packaging).
  - Libraries.

Goals:

- Hardware agnostic.
- Peer-to-Peer.
- Tools-based.
- Multiple programming languages.
- Lightweight.
- Free + open source.
- Suitable for large scale research.