The OROCOS Project

Open RObot COntrol Software

Copyright © 2002-2007 Herman Bruyninckx, Peter Soetens

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation, with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of this license can be found at http://www.fsf.org/copyleft/fdl.html.

Revision History

Revision 0.01	05 Dec 2002	hb
	Initial version	
Revision 0.19.2	11 March 2005	ps
	Added Application Stack figure, removed confusing objects and components section, adapted vision :-)	
Revision 0.22.0	7 March 2006	ps
	Added section 'Building Orocos Applications'	
Revision 1.0.2	4 Jan 2007	ps
	Made this a less technical document	

Abstract

This document gives an application oriented overview of Orocos [http://www.orocos.org], the *Open RObot COntrol Software* project.

Table of Contents

1. What is Orocos? 1
2. Target audience 2
3. Building Orocos Applications 4
    3.1. Application Templates 4
    3.2. Control Components 5
4. Related 'Orocos' Projects 6

1. What is Orocos?

"Orocos" is the acronym of the *Open Robot Control Software [http://www.orocos.org]* project. The project's aim is to develop a general-purpose, free software, and modular *framework* for *robot*and *machine control*. The Orocos project supports 4 C++ libraries: the Real-Time Toolkit, the Kinematics and Dynamics Library, the Bayesian Filtering Library and the Orocos Component Library.

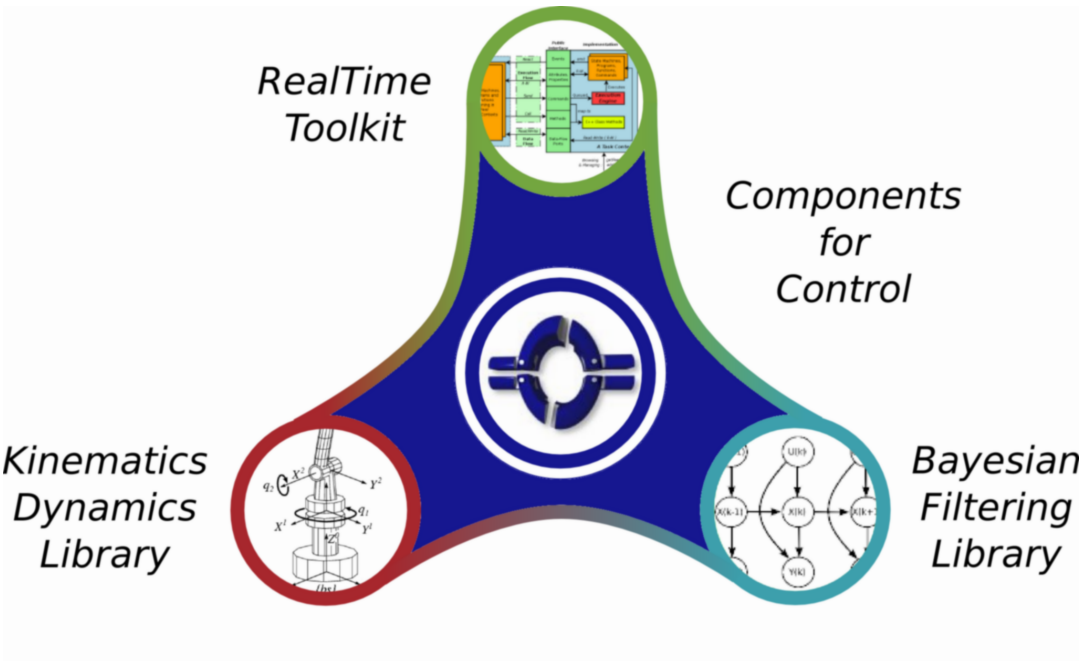


Figure 1. Orocos Libraries

- The Orocos Real-Time Toolkit (RTT) is not an application in itself, but it provides the infrastructure and the functionalities to build robotics applications in C++. The emphasis is on *real-time*, *on-line interactive* and *component based* applications.

- The Orocos Components Library (OCL) provides some ready to use control components. Both Component management and Components for control and hardware access are available.

- The Orocos Kinematics and Dynamics Library (KDL) is a C++ library which allows to calculate kinematic chains in real-time.

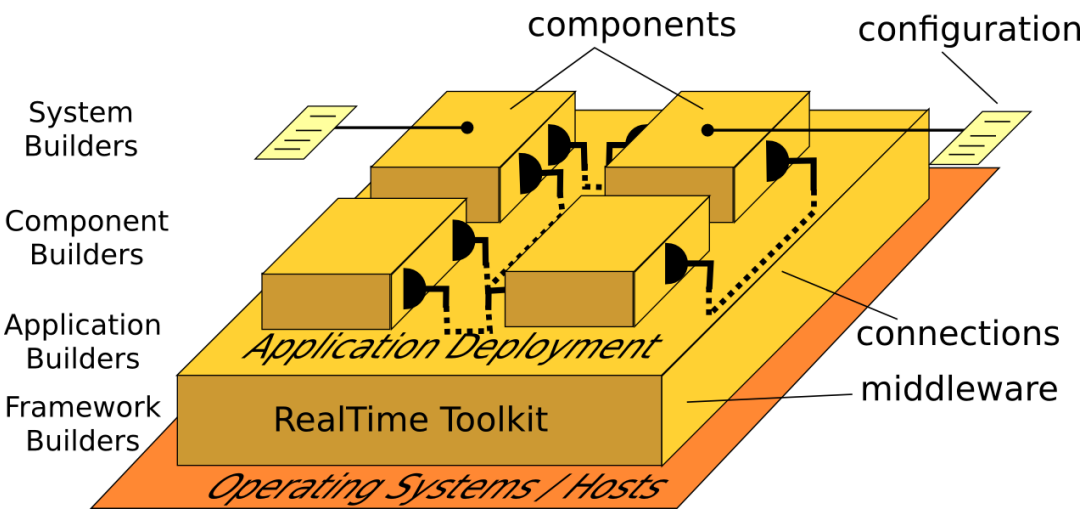
- The Orocos Bayesian Filtering Library (BFL) provides an application independent framework for inference in Dynamic Bayesian Networks, i.e., recursive information processing and estimation algorithms based on Bayes' rule, such as (Extended) Kalman Filters, Particle Filters (Sequential Monte methods), etc.

Orocos is a free software project, hence its code and documentation are released under Free Software licenses.

Your feedback and suggestions are greatly appreciated. Please, use the project's mailing list [http://lists.mech.kuleuven.be/mailman/listinfo/orocos] for this purpose.

2. Target audience

Robotics or machine control in general is a very broad field, and many roboticists are pursuing quite different goals, dealing with different levels of complexity, real-time control constraints, application areas, user interaction, etc. So, because the robotics community is not homogeneous, Orocos targets four different categories of “Users” (or, in the first place, “Developers”):



1. *Framework Builders.*

   These developers do not work on any specific application, but they provide the infrastructure code to support applications. This level of supporting code is most often neglected in robot software projects, because in the (rather limited) scope of each individual project, putting a lot of effort in a generic support platform is often considered to be “overkill”, or even not taken into consideration at all. However, because of the large scope of the Orocos project, the supporting code (the “Framework”) gets a lot of attention. The hope is, of course, that this work will pay of by facilitating the developments for the other “Builders”. The RTT, KDL and BFL are created by Framework builders

2. *Component Builders.*

   Components provide a “service” within an application. Using the infrastructure of the framework, a Component Builder describes the interface of a service and provides one or more implementations. For example a Kinematics Component may be designed as such that it can “serve” different kinematic architectures. Other examples are Components to hardware devices, Components for diagnostics, safety or simulation. The OCL is created by Component Builders.

3. *Application Builders.*

   These developers use the Orocos' Framework and Components, and integrate them into one particular application. That means that they create a specific, application-dependent *architecture*: Components are connected and configured as such that they form an application.

4. *End Users.*

   These people use the products of the Application Builders to program and run their particular tasks.

End Users do not directly belong to the target audience of the Orocos project, because Orocos concentrates on the common *framework*, independent of any application architecture. Serving the needs of the End Users is left to (commercial and non-commercial) Application Builders.

3. Building Orocos Applications

Orocos applications are composed of software components, which form an application specific network. When using Orocos, you can choose to use predefined components, contributed by the community, or build your own component, using the Orocos Real-Time Toolkit. This section introduces both ways of building applications.

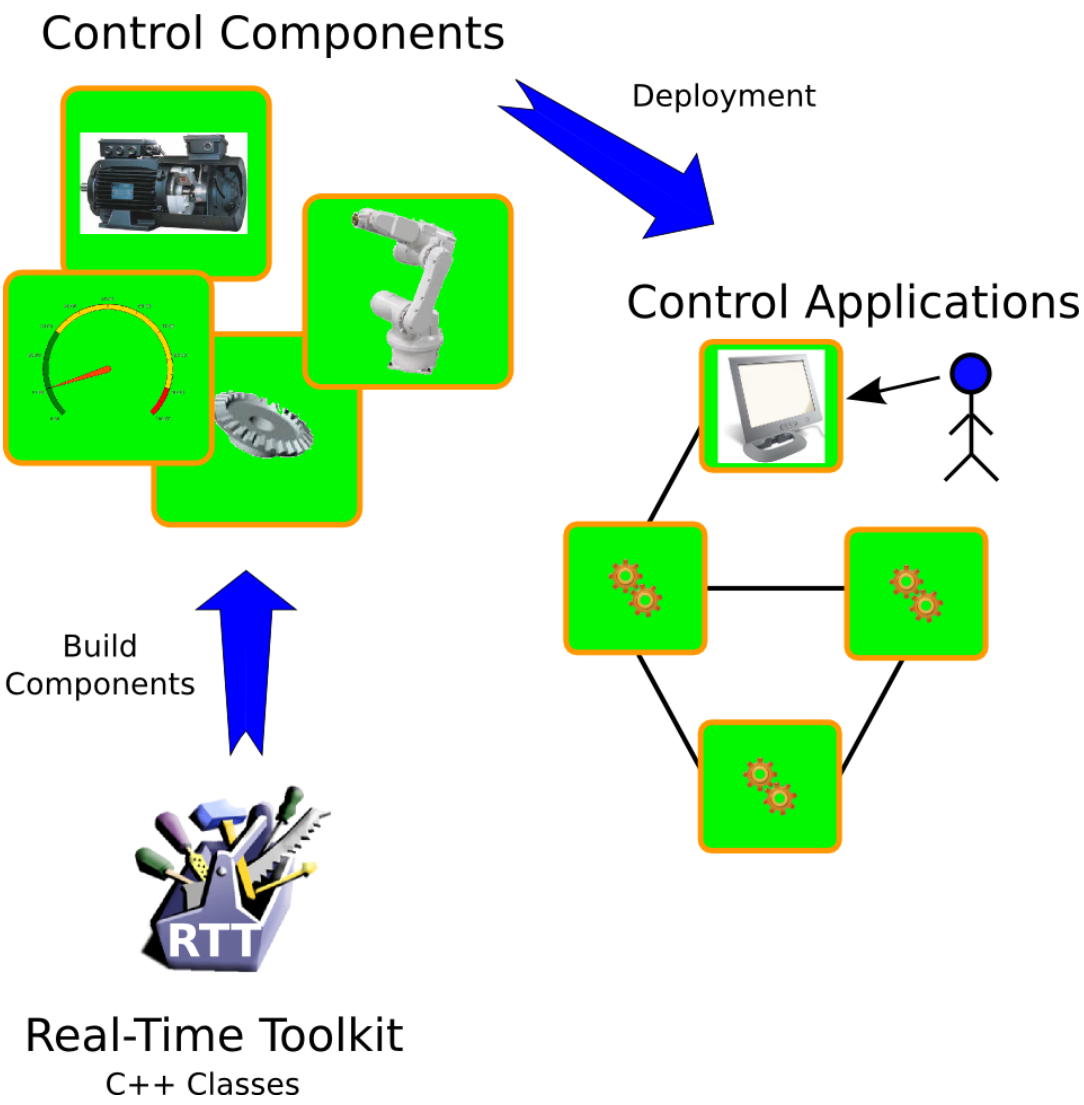


Figure 2. Orocos Real-Time Toolkit

3.1. Application Templates

An "Application Template" is a set of components that work well together. For example, the application template for motion control contains components for path planning, position control, hardware access and data reporting. The components are chosen as such that their interfaces are compatible.

An application template should be so simple that any Orocos user can pick one and modify it, hence it is the first thing a new user will encounter. An application template should be explainable on one page with one figure explaining the architecture.

Note

An application template has no relation to 'C++' templates.

3.2. Control Components

Applications are constructed using the Orocos "Control Component". A distributable entity which has a control oriented interface.

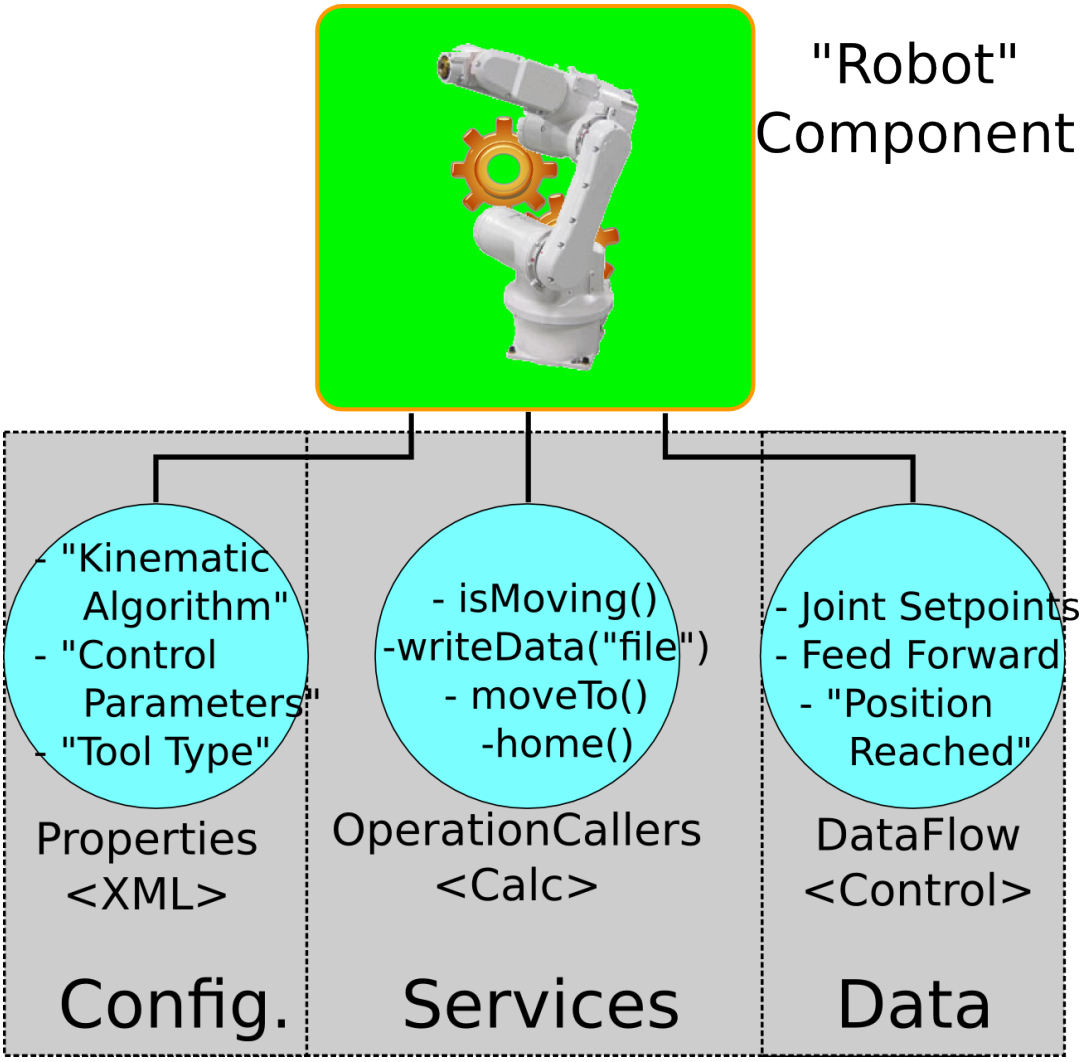


Figure 3. Orocos Control Component Interface

A single component may be well capable of controlling a whole machine, or is just a 'small' part in a whole network of components, for example an interpolator or kinematic component. The components are built with the "Real-Time Toolkit" and optionally make use of any other library (like a vision or kinematics toolkit). Most users will interface components through their (XML) properties or command/method interface in order to configure their applications.

There are five distinct ways in which an Orocos component can be interfaced: through its properties, events, methods, commands and data flow ports (Figure 3, “ Orocos Control Component Interface ”). These are all optional interfaces. The purpose and use of these interface 'types' is documented in the Orocos Component Builder's Manual. Each component docu-

ments its interface as well. To get a grip on what these interfaces mean, here are some fictitious component interfaces for a 'Robot' Component:

- *Data-Flow Ports:* Are a thread-safe data transport mechanism to communicate buffered or un-buffered data between components. For example: "JointSetpoints", "EndEffectorFrame", "FeedForward",...
- *Properties:* Are run-time modifiable parameters, stored in XML files. For example: "Kinematic Algorithm", "Control Parameters", "Homing Position", "ToolType",...
- *OperationCallers:* Are callable by other components to 'calculate' a result immediately, just like a 'C' function. For example: "getTrackingError()", "openGripper()", "writeData("filename")", "isMoving()", ...
- *Commands:* Are 'sent' by other components to instruct the receiver to 'reach a goal' For example: "moveTo(pos, velocity)", "home()",... A command cannot, in general, be completely executed instantaneously, so the caller should not block and wait for its completion. But the Command object offers all functionalities to let the caller know about the progress in the execution of the command.
- *Events:* Allows functions to be executed when a change in the system occurs. For example: "Position Reached", "Emergency Stop", "Object Grasped",...

Besides defining the above component communication mechanisms, Orocos allows the Component or Application Builder to write hierarchical state machines which use these primitives. This is the Orocos way of defining your application specific logic. State machines can be (un-)loaded at run-time in any component.

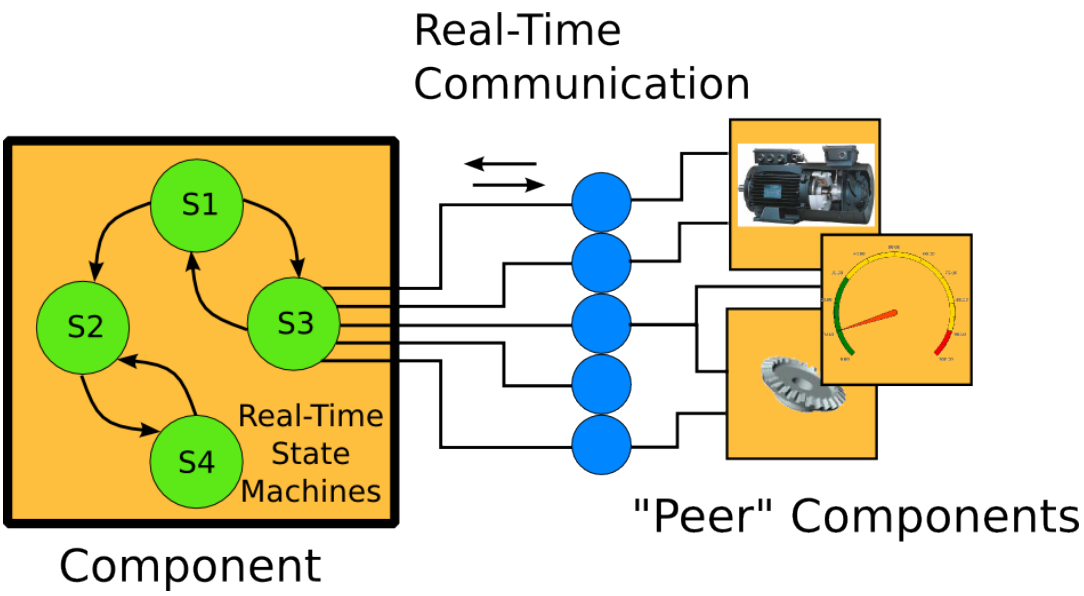


Figure 4. Orocos Control Component State Machines.

4. Related 'Orocos' Projects

The Orocos project spawned a couple of largely independent software projects. The documentation you are reading is about the Real-Time Control Software located on the Orocos.org web page. The other *not real-time* projects are :

- At KTH Stockholm, several releases have been made for component-based robotic systems, and the project has been renamed to Orca [http://orca-robotics.sourceforge.net/].
- Although not a project funded partner, the FH Ulm maintains Free CORBA communication patterns for modular robotics : Orocos::SmartSoft [http://www.rz.fh-ulm.de/~cschlege/orocos/].

This documentation is targeted at industrial robotics and real-time control.