Python for Control Systems, **Big** and *Small*

Thomas Kluyver, European XFEL GmbH

Holzkoppel 4, 22869 Schenefeld, Germany

www.xfel.eu

The big: European XFEL's 3.4 km tunnels under Hamburg. An XFEL (X-ray free electron laser) accelerates electrons, then converts their energy into X-rays which are focused onto a sample to investigate its molecular structure. The control system at European XFEL is called Karabo.

The small: the *Black Python*, a one metre autonomous sailing boat built at the University of Southampton. The boat is controlled by ROS (the Robot Operating System) running on a Raspberry Pi.

Karabo

in.xfel.eu/readthedocs/docs/karabo/en/latest/

ROS

www.ros.org

The Language Stack

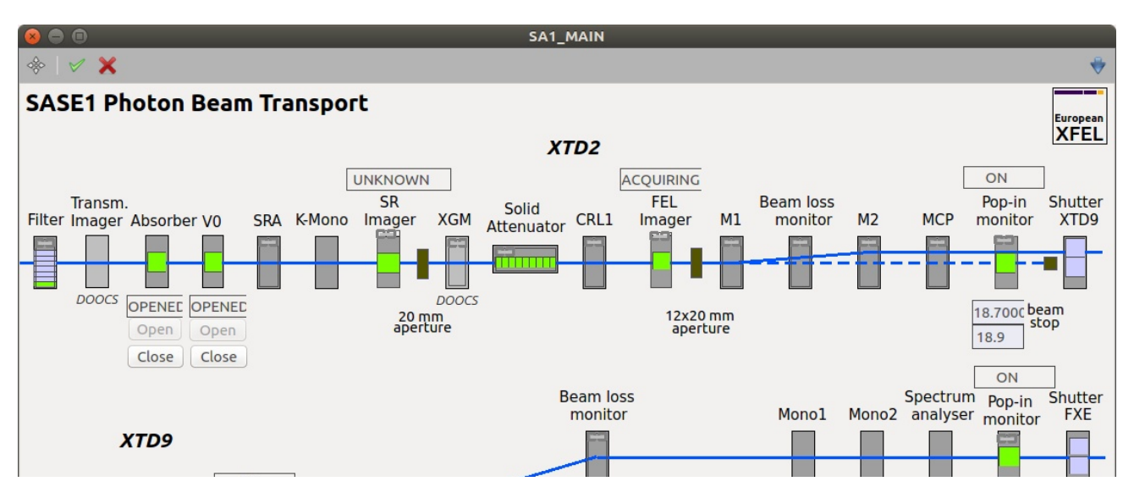
Both systems have low level layers written in C++, and Python interfaces for higher levels of control and coordination, as do other control systems like Tango. The tried-and-tested compiled language is popular where performance is important, and for direct control of crucial hardware.

There are two ways to interface Python with low-level code. **Bindings** allow Python to call lower-level functions in the same process, which can be more efficient. Alternatively, the interface can be built atop a **communication layer**, typically using network sockets. ROS uses the latter approach, which also allows interfaces from other languages, such as Lisp and Java. Karabo has both a bound Python API and the 'middlelayer' API using the messaging layer.

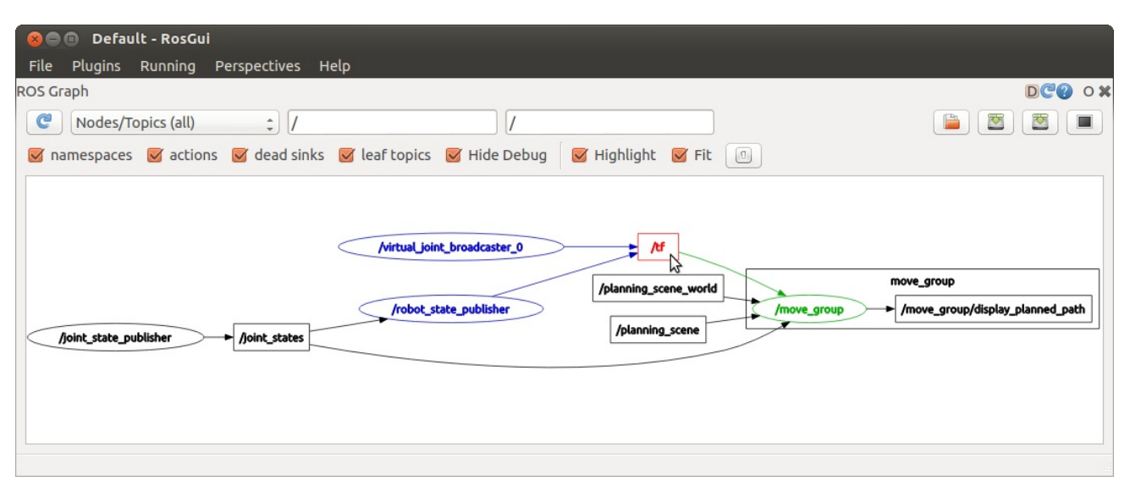
User Interfaces

Python is used to construct graphical user interfaces (GUIs) and visualisations, such as the Karabo GUI, many *rqt* plugins for ROS, and in other systems, such as *Sardana* for Tango. These examples all use Qt, an open-source, cross-platform GUI toolkit, along with plotting libraries such as matplotlib and PyQwt.

Control systems often expose a command line interface (CLI) where a user can directly enter Python code. *IKarabo* and *ITango* are both based on the popular IPython CLI, customised for their respective control systems.



A scene in the Karabo GUI



The rqt_graph plugin for ROS

Recording Data

ROS' *rosbag* tool can record all the messages sent around a robot system to a 'bag' file, which can be replayed later for debugging. For the sailing project, bag flles could easily be stored on a Micro SD card.

For European XFEL, the scale of the data produced (tens of GB per second) makes recording more of a challenge. A number of ‘data acquisition’ devices running in parallel receive data from selected sources and write it to HDF5 files on a parallel filesystem.

Messages

Karabo and ROS both use a publish/subscribe messaging pattern, allowing components in different processes and different computers to communicate. But the way they work is quite different.

ROS nodes communicate directly, with a central ‘master’ which is used only to match up publishers and subscribers. As used for the *Black Python*, ROS messages are repeated at regular intervals, even if their data hasn't changed. This is a simple way to improve reliability: if a subscriber node crashes and is restarted, it will soon get the current data.

In Karabo, control data is sent through a central broker in a distributed signals & slots system. Larger data, such as camera images, can be sent directly between devices using input and output channels.

Bridging Out

Rosbridge provides a simpler JSON-based protocol for ROS. Web pages can talk to Rosbridge from the browser using websockets, so users can build dashboards in HTML and Javascript.

Karabo bridge allows data analysis code to receive data from Karabo without having to be built against the framework. It uses ZeroMQ and msgpack to send data efficiently. In the future, this may be extended to send data back to Karabo as well.

The existence of these ‘bridge’ protocols may seem strange. Both Karabo and ROS are built around parts communicating over network sockets, but it's not practical to have everything use the core protocol. Could simpler or more standardised protocols avoid the need for a bridge? Perhaps the range of systems that need to communicate is too broad for a single protocol to work for all of them.

References

B. Heisen, et al. *Karabo: An integrated software framework combining control, data management, and scientific computing tasks*, in 14th ICALEPCS, San Francisco, USA, 2013

M. Quigley, et al. *ROS: an open-source Robot Operating System*, in ICRA Workshop on Open Source Software, Vol. 3 No. 3.2, 2009.