

**An introduction to the
Orocos Toolchain v2
euRobotics Forum, 7 Apr 2011
Vasteras, Sweden**

Why Orocos?

- **Software which supports all advanced controllers for robotics and machine tools.**
 - **Focus: hard realtime & component based.**
 - **And: via interoperability with other projects!**
- **To provide industry-friendly licenses & support.**



Industry-friendly licensing & support

- **Orocos provides a framework**
 - the components contain the application builders' intellectual property.
 - and the Orocos licenses do not force any dissemination of that IP!
- **Support via**
 - ...classical mailing list/fora
 - ... Intermodalics (which employs key Orocos developers). . .
 - ... but also independent service providers have their hands completely free!

Real-time and Portable

- **Hard realtime is Orocos' competitive advantage:**
 - **Lock-free data ports favour highest priority component activity.**
 - **Realtime-aware memory management.**
 - **Does not prevent non-realtime use!**
- **... in an extremely flexible and extendible programming environment**

Real-time and Portable

- **Orocos Toolchain is supported on:**
 - **Linux 32/64bit (GNU,clang,Intel)**
 - **Real-Time Linux Extensions**
 - Xenomai
 - Real-Time Application Interface (RTAI)
 - **Mac OS-X (GNU)**
 - **Windows (XP->7) 32/64bit (MSVS2005-2010)**
 - **QNX (GNU) – beta**
 - **VxWorks - planned**

Orocos sub-projects

- **RTT: Real-Time Toolkit (“Run-Time Toolkit”!)**
 - This talk
- **KDL: Kinematics & Dynamics Library**
 - One of a kind in C++ land
- **BFL: Bayesian Filtering Library**
 - One of a kind in C++ land
- **OCL: Orocos Components Library**
 - Supports component development

Who's using Orocos. . . ?

- **Institutes:**

- K.U.Leuven (B), FU Berlin (D), Polytechnic University of Catalonia (ES), University of Florida (US), German research centre for Artificial Intelligence (DFKI, Bremen, D), University of New South Wales (Sydney, AU), University of Maryland (USA), Polytechnic University of Milan (I), University of Southern Denmark, Onera (F), Irisa (F), Cea (F)

- **Companies:**

- three FMTC member companies (with already one product several years on the market!); Willow Garage; and some other machine or robotics builder companies in Belgium, France, Spain, Canada, USA (NASA) and the Netherlands, with several products on the market

What is RTT ?

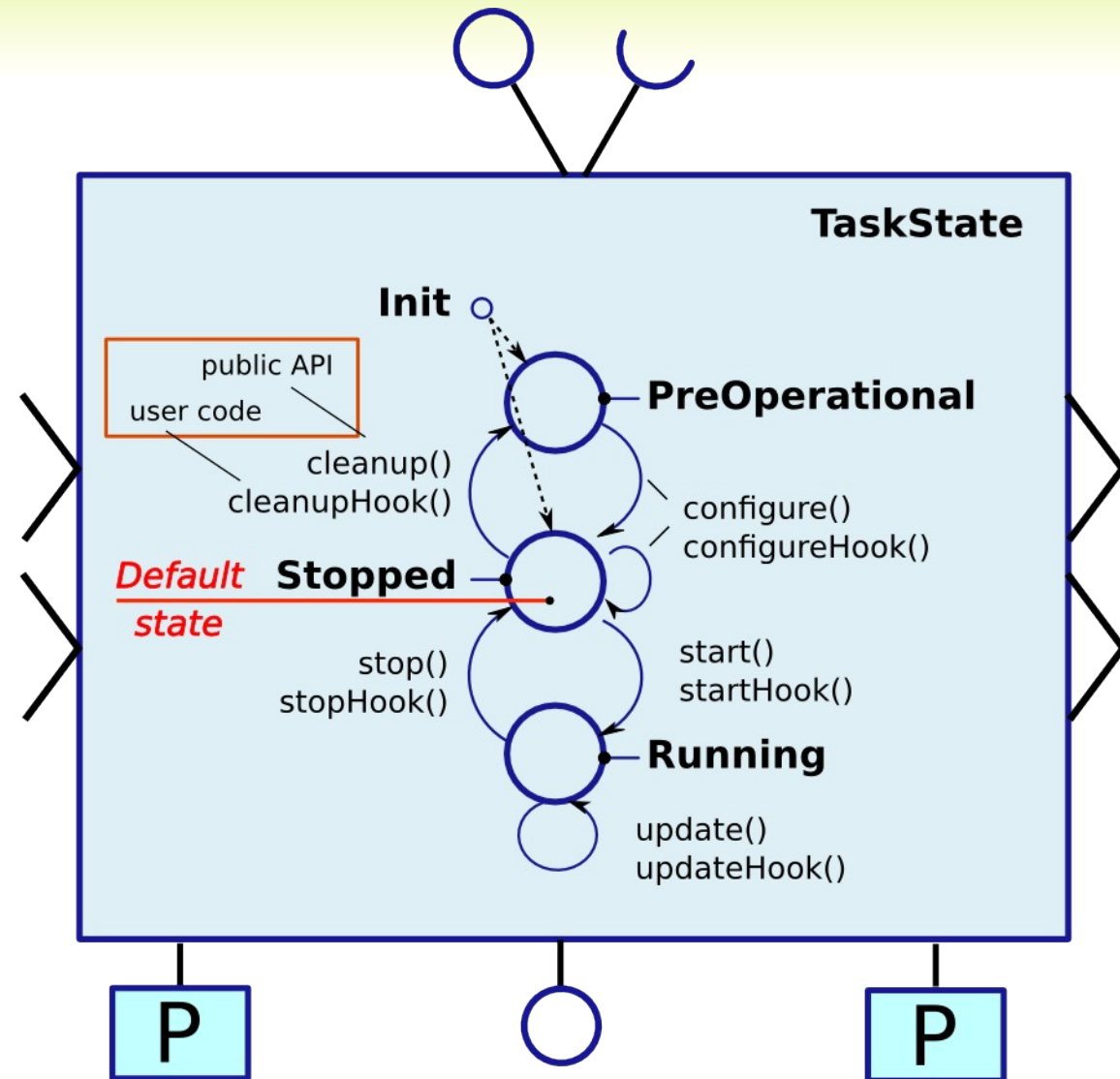
- **Acronym for Real-Time Toolkit**
- **A cross-platform, Component framework in C++ that allows:**
 - **creating dynamic loadable and distributable components**
 - **to guarantee real-time, thread-safe communication**
 - **real-time, event driven state machine scripts**
 - **run-time interface inspection and communication**
- **Without imposing any application architecture.**

Extensions to RTT

- **Log4Cpp logging framework**
 - With real-time logging support
- **Lua scripting support**
 - With application deployment and supervision
- **OroGen / ROCK**
 - Model based code generation of components and applications
- **ROS integration**
 - Open source framework for service robotics
- **Networked component communication**
 - Message queues, CORBA, Yarp, ZeroMQ (planned)

A Component's Life Cycle

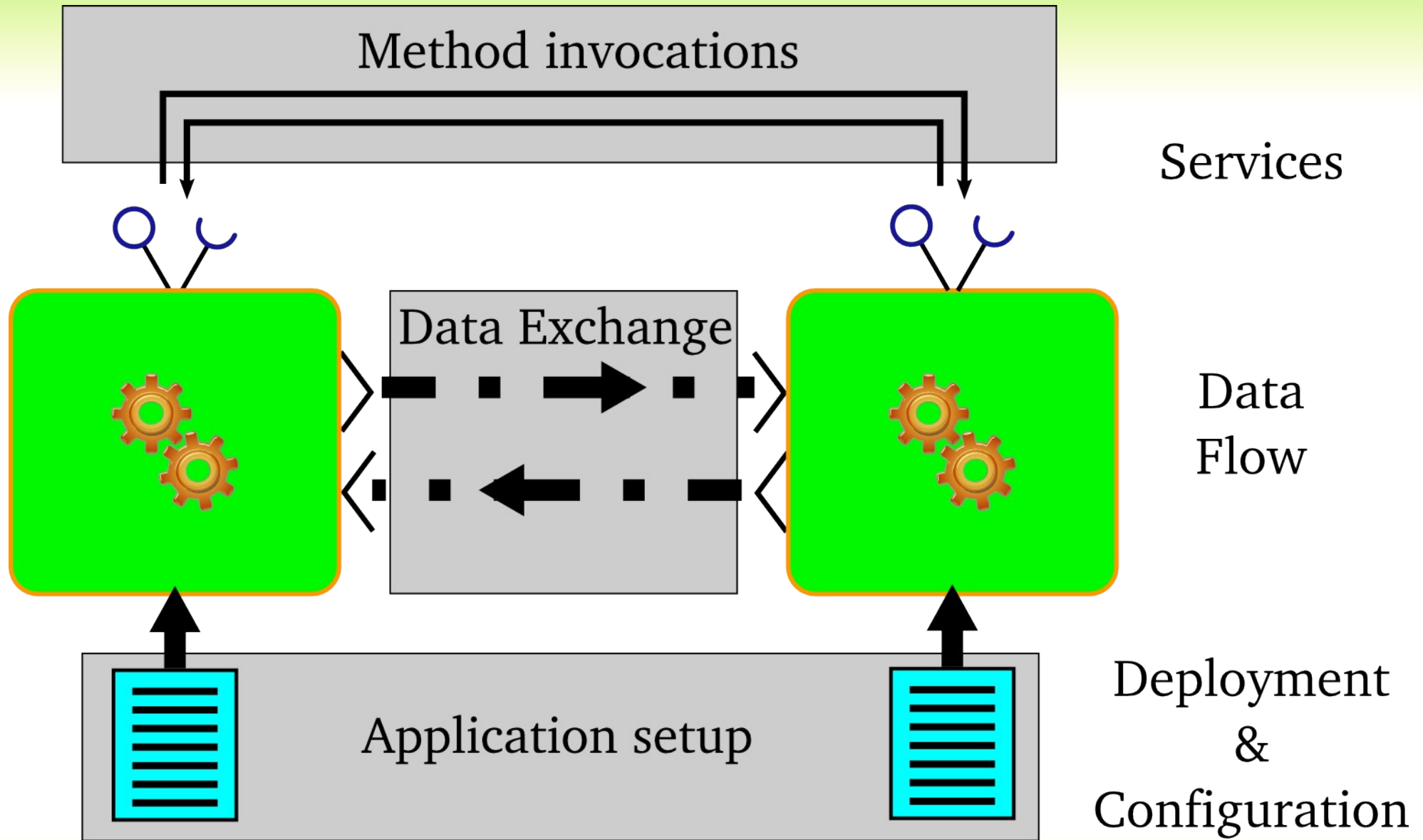
- Allows non real-time configuration and cleanup
- Starting is only allowed once configured correctly
- Extended with basic error states



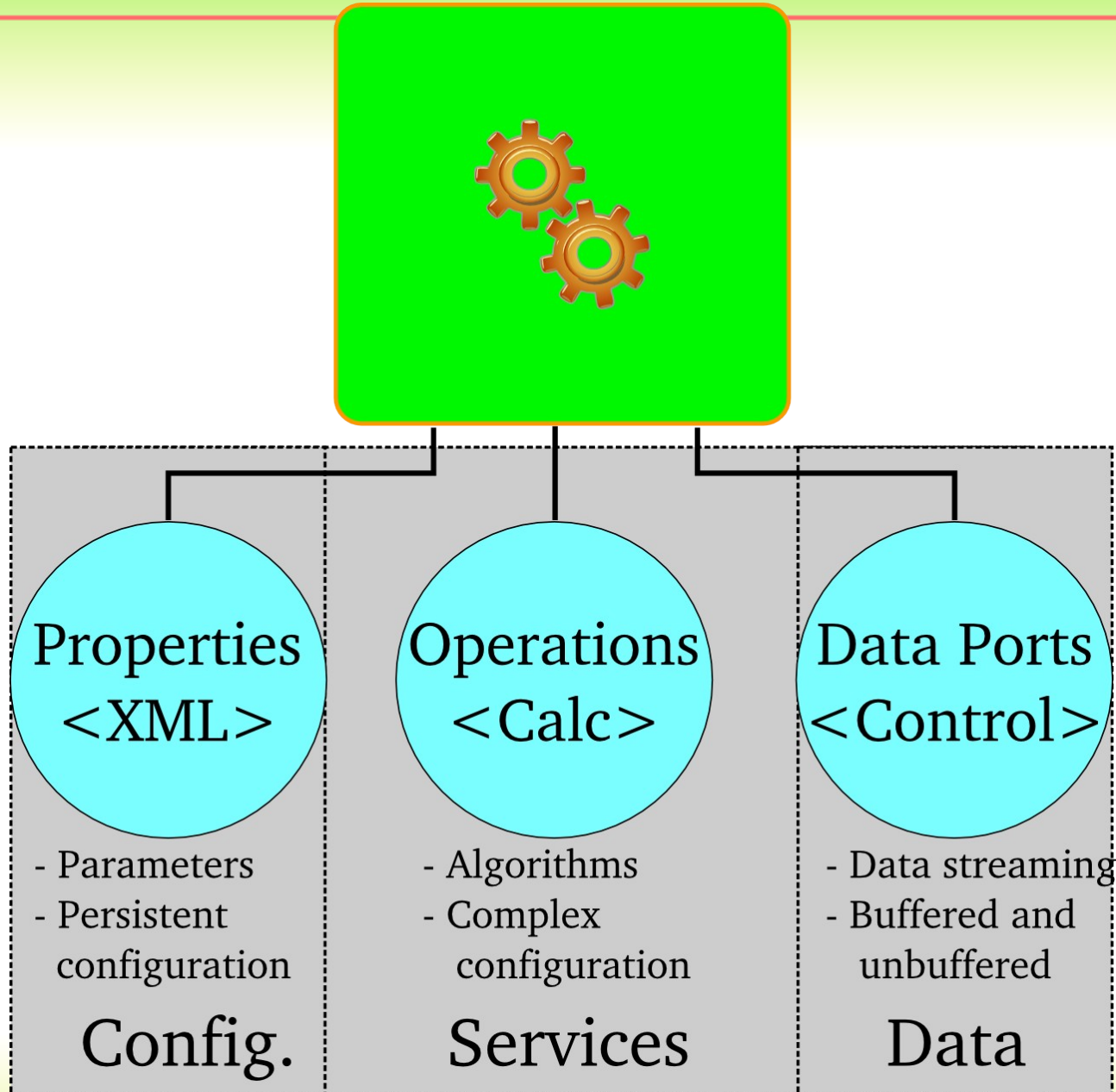
A Component's Basic Communication

- **In which ways can components communicate?**
 - **Configuration of parameters**
 - **Exchange (streaming) data**
 - **Cooperate to achieve a task**

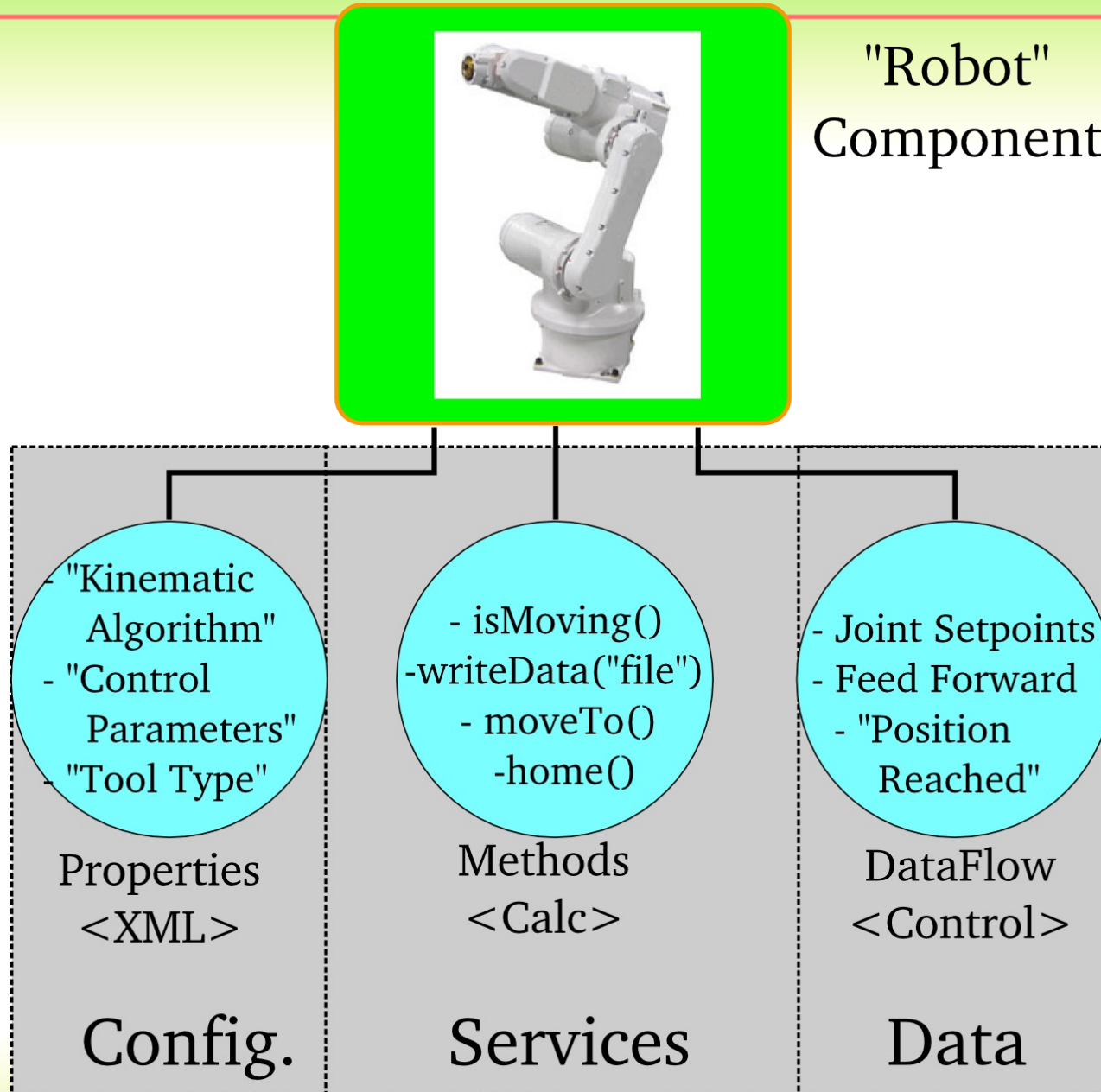
A Component's Basic Communication



A Component's Interface: Robot Example

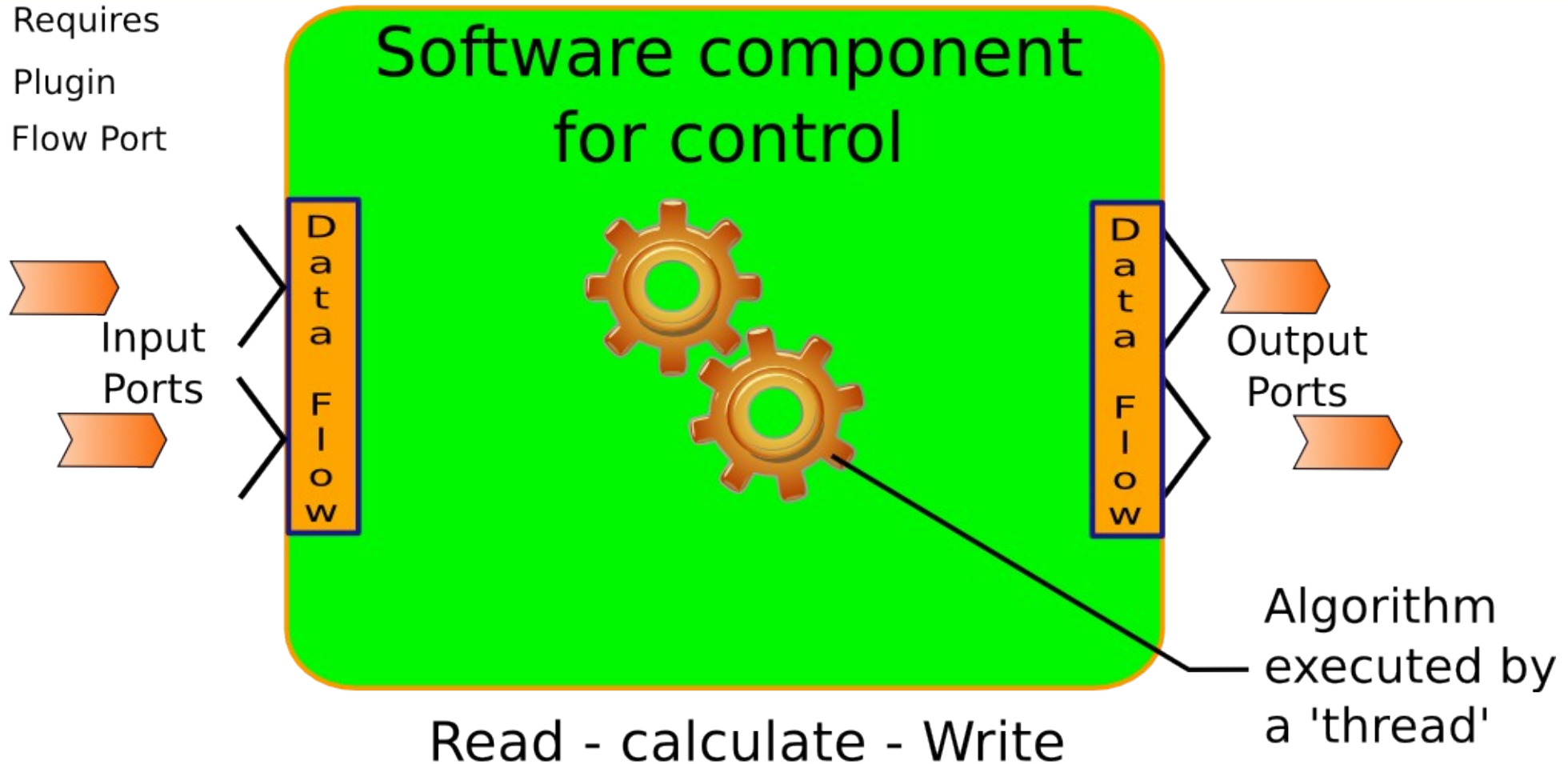


A Component's Interface: Robot Example

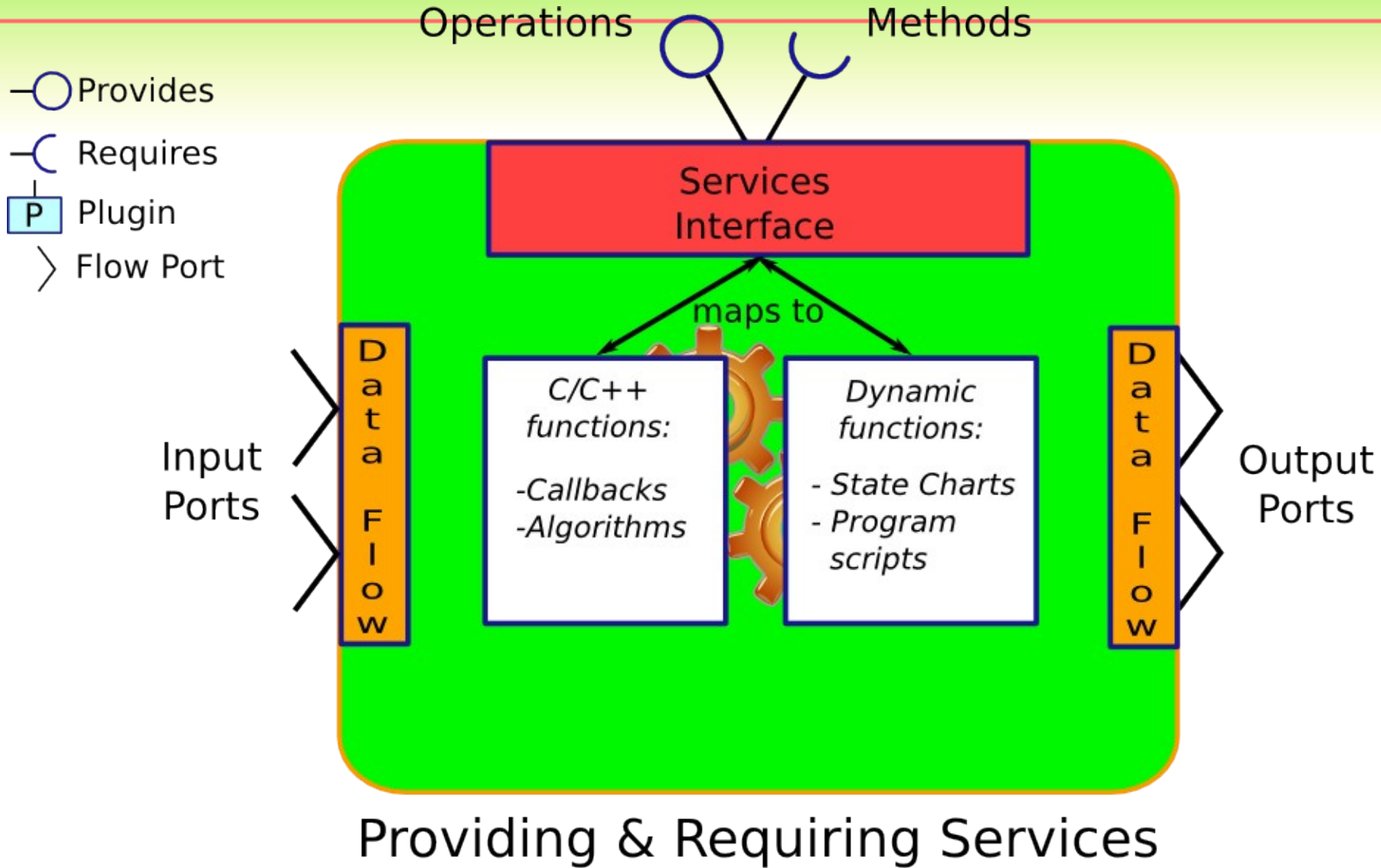


Component break-down

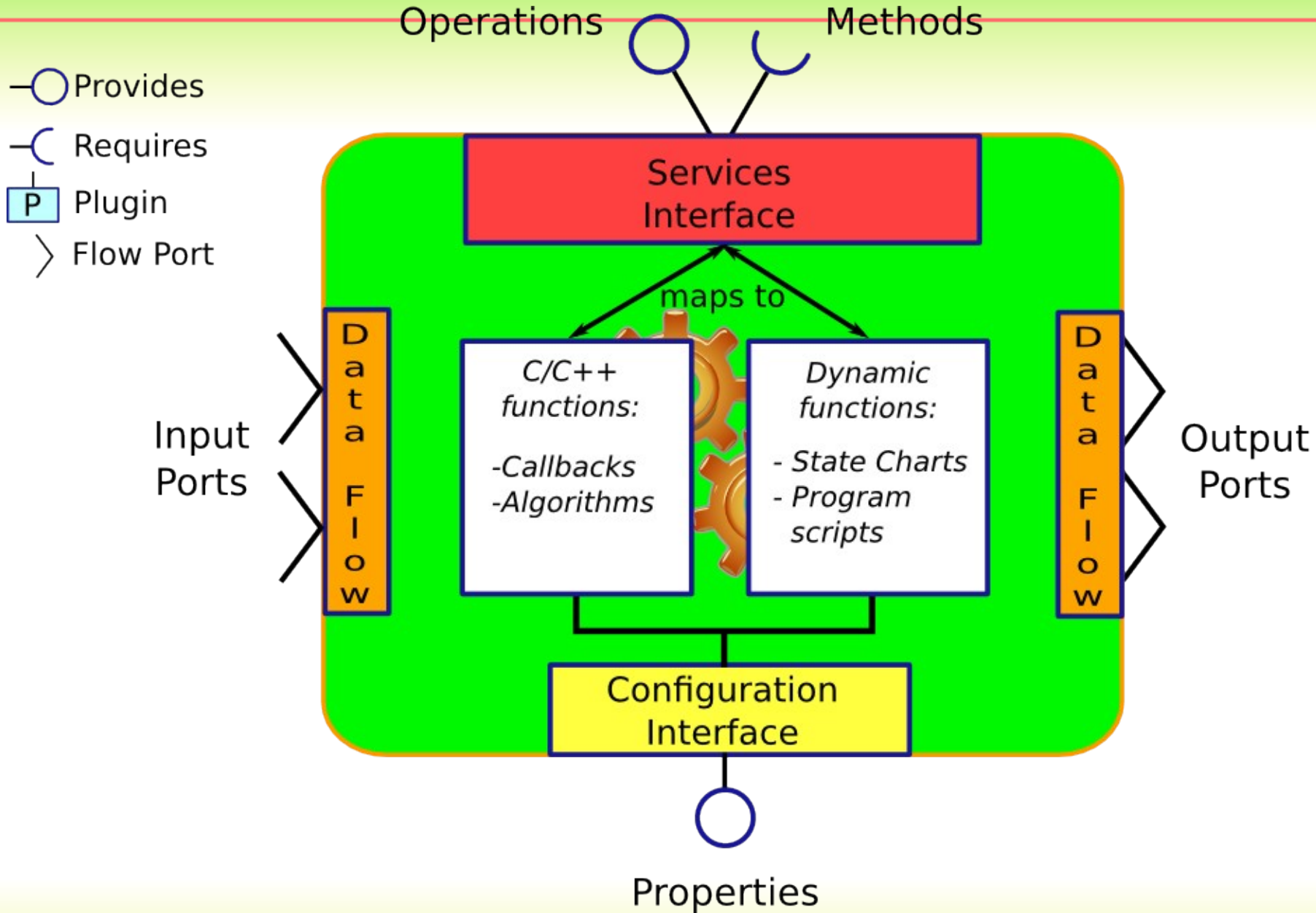
- Provides
- ⊂ Requires
- P Plugin
- > Flow Port



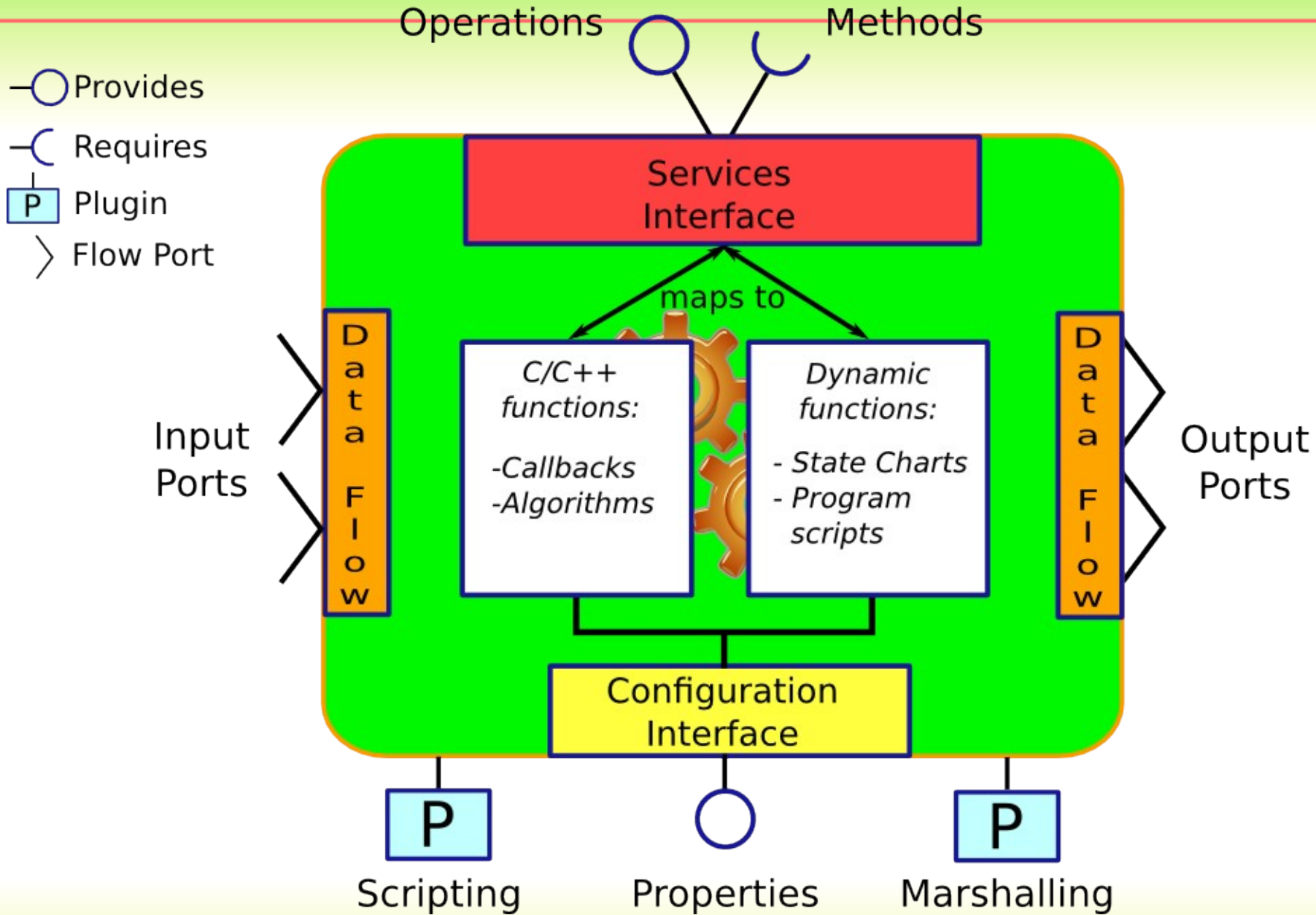
Component break-down



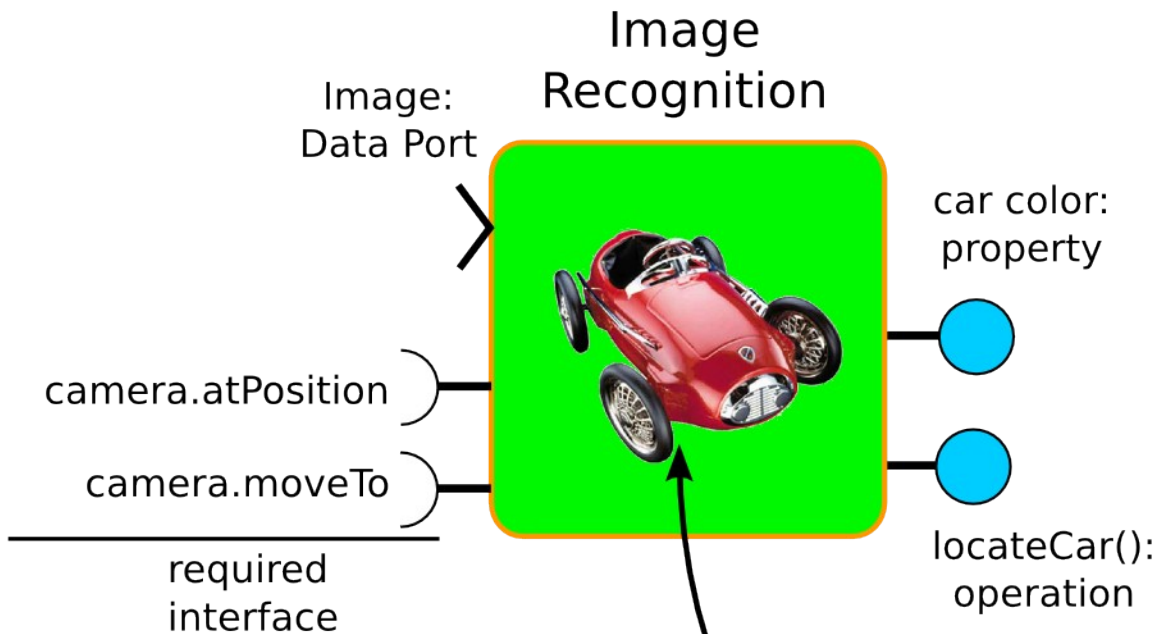
Component break-down



Component break-down



Example of image recognition



file: statemachine.osd

```
StateMachine ExampleSM
{
  initial state wait_for_image {
    // on imageReady event, make
    // transition to other state:
    transition Image( new_image )
      select image_captured;
  }

  state image_captured {
    // program executed when this state
    // is entered:
    entry {
      set p = this.locateCar();
      camera.moveTo( p );
    }
    // when entry is done, go back:
    if camera.atPosition( p )
      select wait_for_image;
  }

  final state end {}
}
RootMachine ExampleSM sm;
```

Code break-down

```
#include <rtt/TaskContext.hpp>
#include <ocl/ComponentDeployment.hpp>
```

```
/**
 * Note: we're defining a component as a class in a .cpp file
 * not in a header file !
 */
```

MyComponent.cpp

```
class MyComponent
: public RTT::TaskContext
```

RTT::TaskContext is our base

```
{
public:
```

```
MyComponent(string name)
: RTT::TaskContext(name)
{
addOperation("set_param",&MyComponent::set_param,this)
.doc("Set parameter X")
.arg("value", "The argument of this method.");
}
```

Interface building

```
bool configureHook() {
// further setup which could not be done in the
// constructor...
}
```

User code is placed in hooks...

```
void set_param(double x) {
// ...
}
```

... or in custom functions

```
};

/**
 * Make MyComponent dynamically loadable.
 */
```

```
ORO_CREATE_COMPONENT( MyComponent )
```

Makes this library a loadable Orocos component

That's all folks !

Let's start for real after the break !