Italian Institute of Technology (IIT)

Yarp Module Management toolkit

Ali Paikan

July 2011

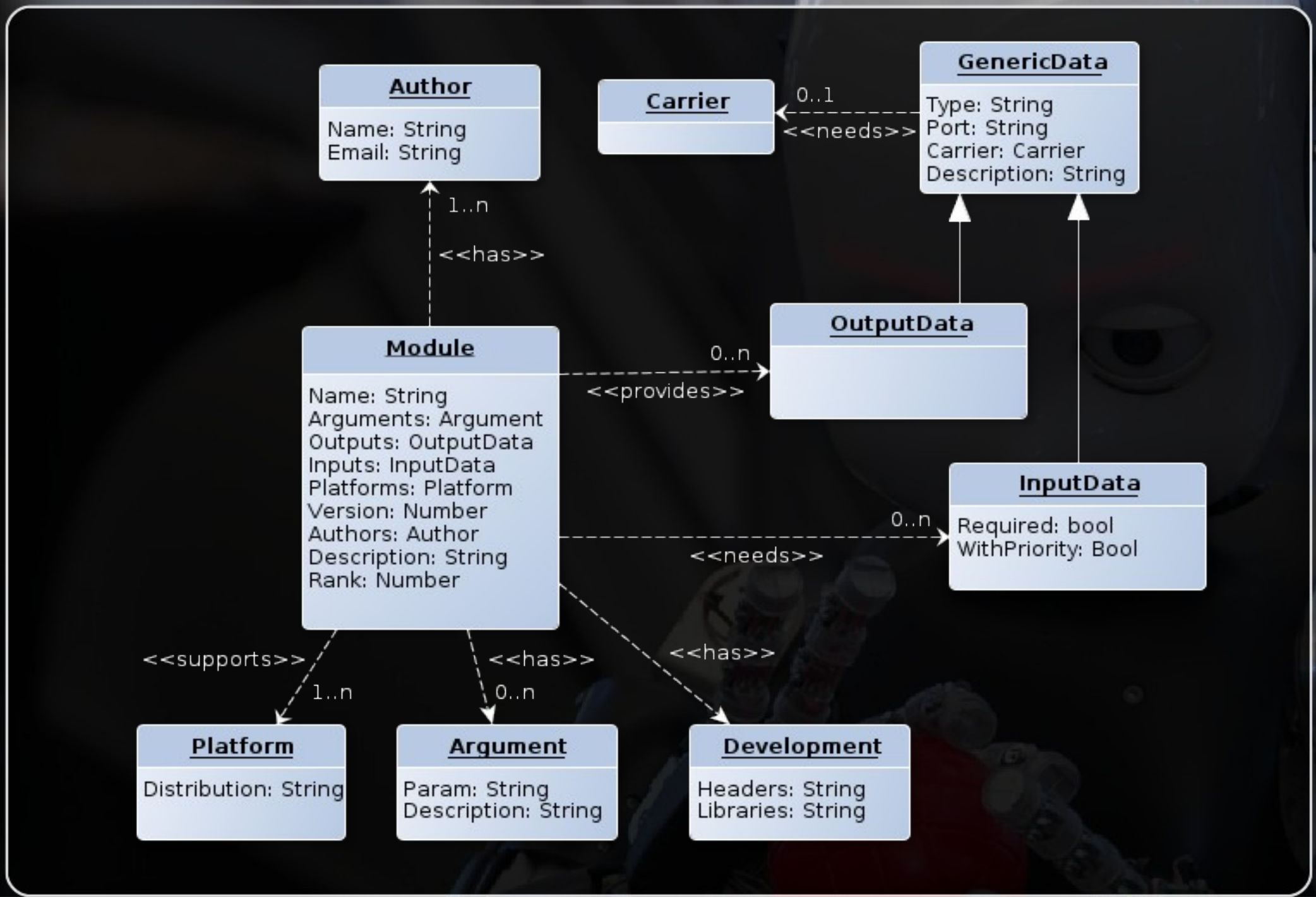
Problem Description

- Increasing in number of iCub software modules
- Needs for module description (input/output ports, dependencies, version, ...)
- Managing module connections in a complex application
- Needs for Module integrity (building module blocks over each other, nested application)
- Dependency resolving
- Application execution and failure recovery
- Writing, checking and modifying long XML files for implementing complex application

Available approaches

- *“Icubapp.py”*, “*Manager.py”* and “*yarprun”*
- Some of the available robotics middleware simply offers module execution via “SSH” (roslaunch, ...)
- Better approaches with failure recovery can be seen in Grid and Cloud computing (OLAN, PLUSH, …)
- Some alternative efficient approaches to “ssh” (focus on performance and usually need OS kernel modification)

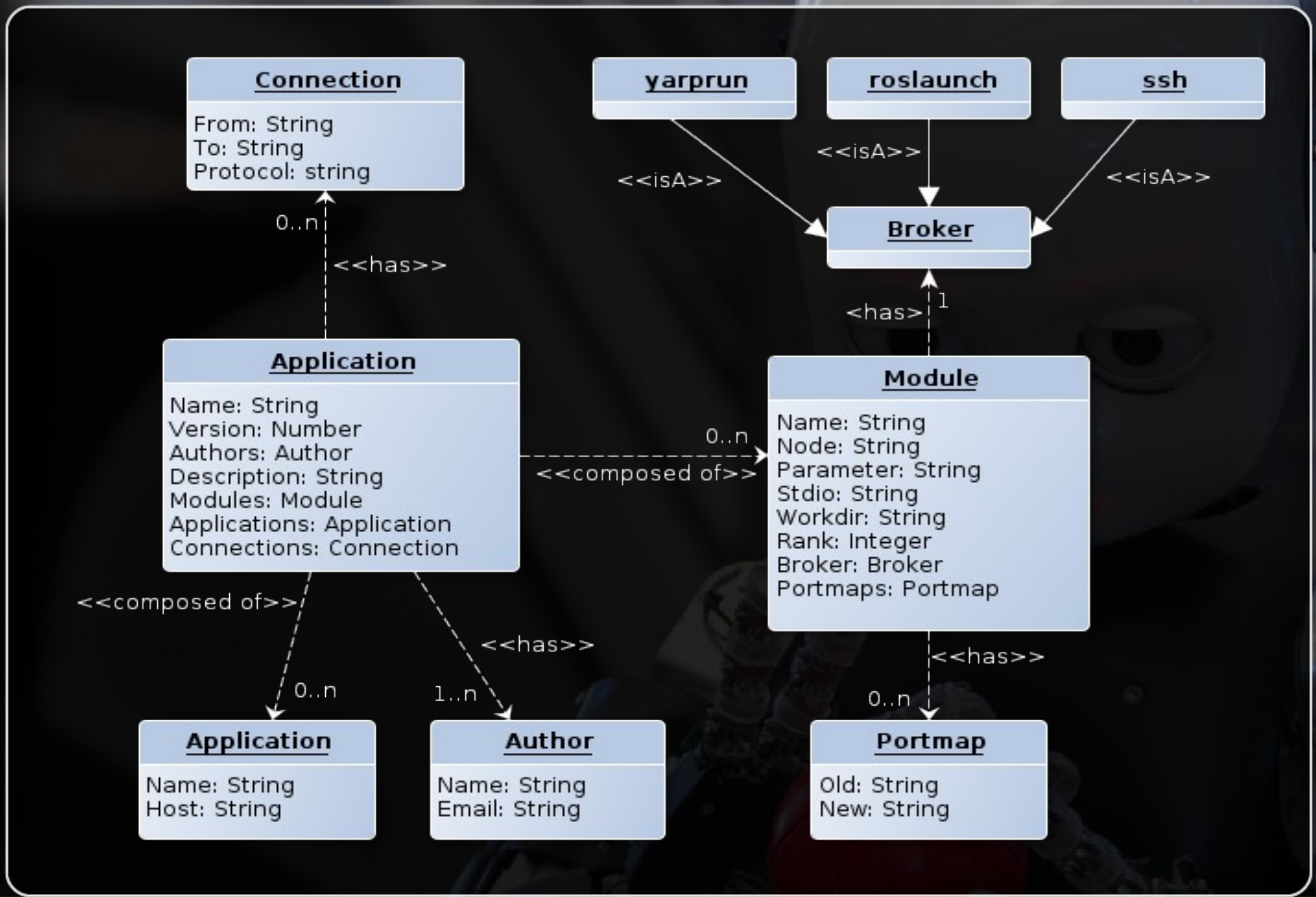
Module Description



Module Description (example)

```
<module>
    <name>camCalibConf</name>
    <description> Camera Calibration</description>
    <version>1.0</version>
    <rank>1</rank>

    <authors>
        <author email="jonas.ruesch@isr.ist.utl.pt"> Jonas Ruesch </author>
    </authors>

    <arguments>
        <param desc="configuration path"> context </param>
    </arguments>

    <input>
        <type>cartesian-image</type>
        <port carrier="UDP">/camCalibConf/image</port>
        <required>yes</required>
        <priority>yes</priority>
    </input>
    <output>
        <type>cartesian-image</type>
        <port carrier="UDP">/camCalibConf/image</port>
    </output>
</module>
```

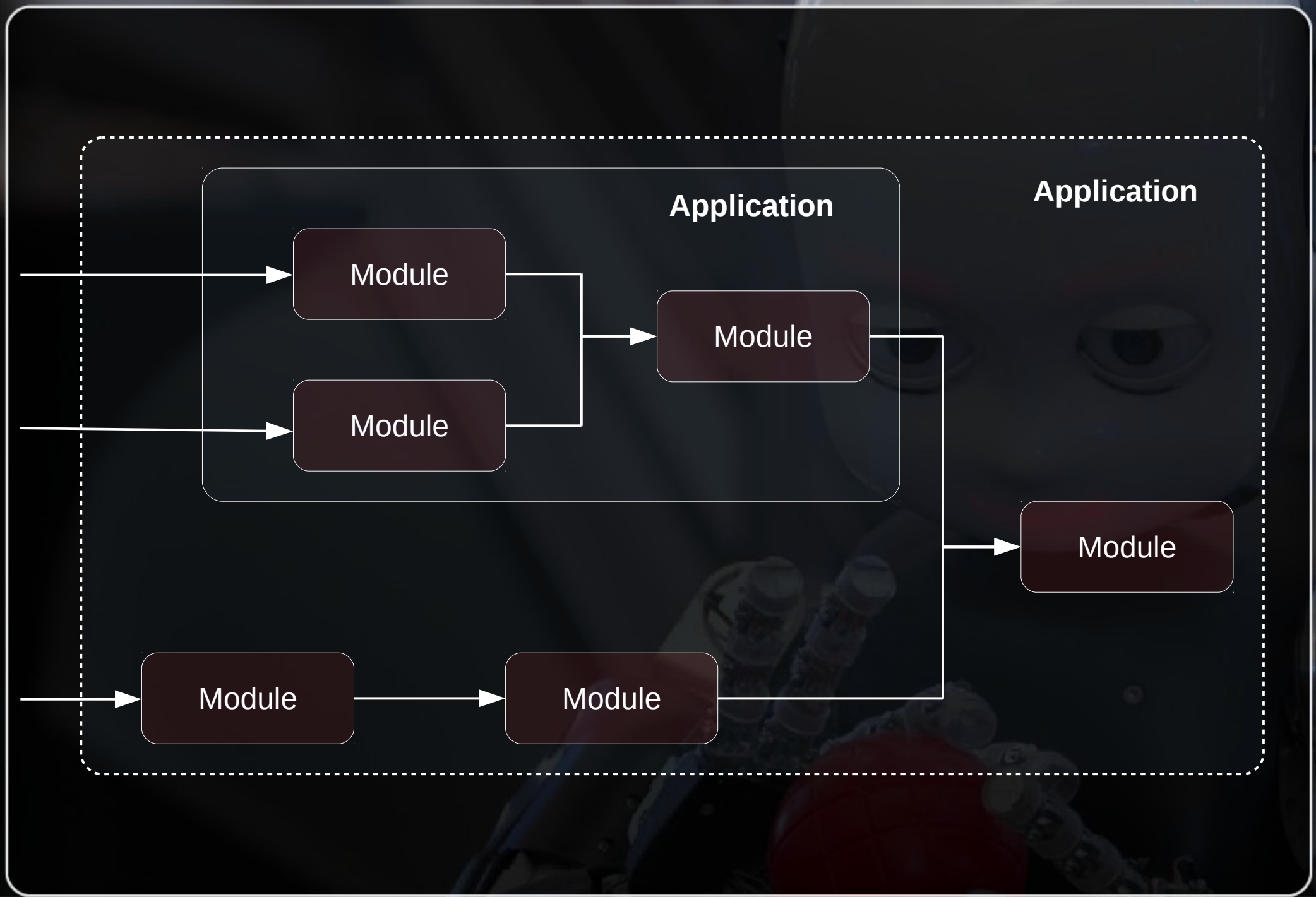
Application Description



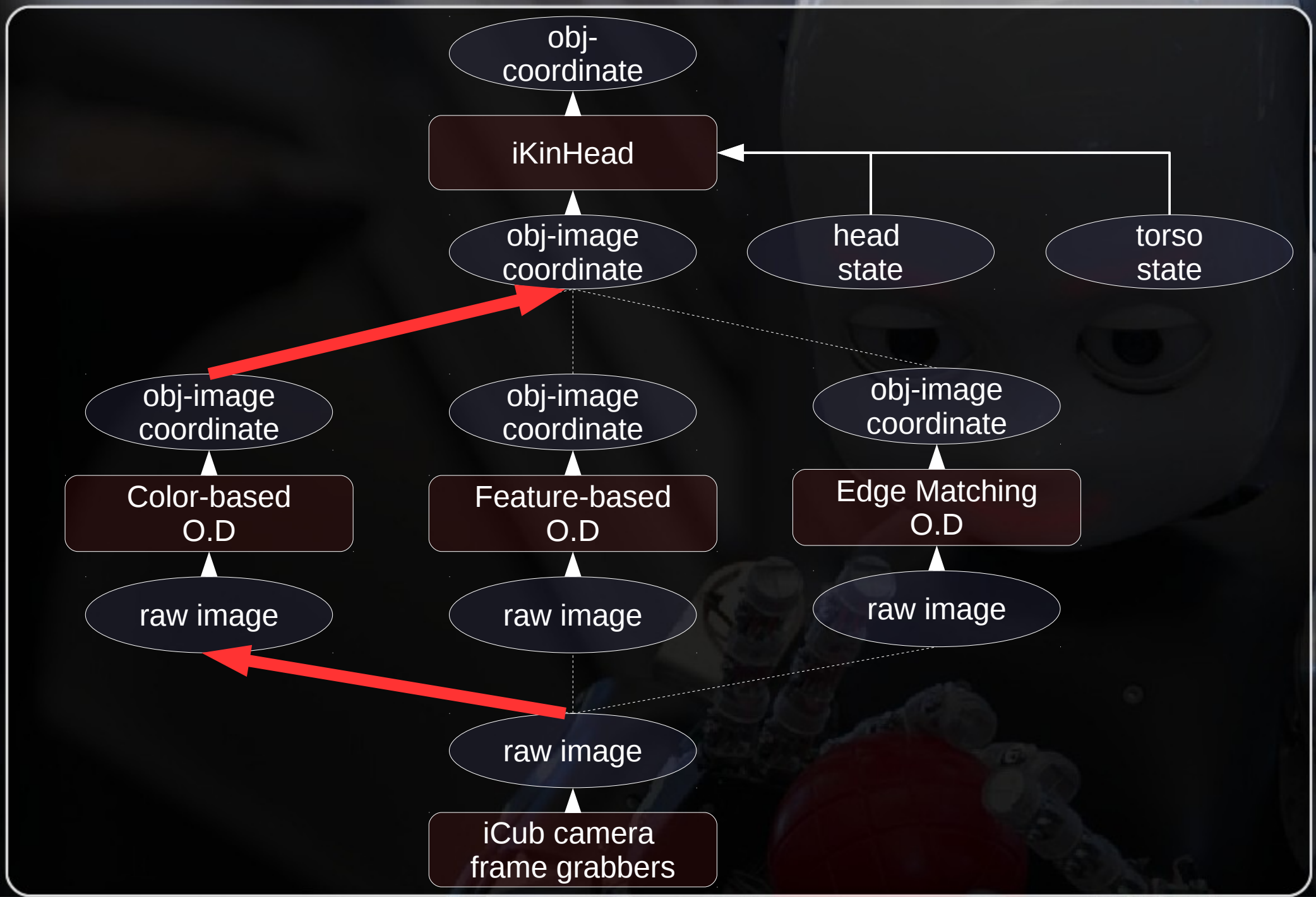
Application Description (example)

```
<application>
    <name>AllCameraCalibration</name>
    <description> ... </description>
    <version>1.0</version>

    <module>
        <name>camCalibConf</name>
        <parameters>--group CAMERA_CALIBRATION_CONFIGURATION_RIGHT</parameters>
        <node>icub1</node>
        <broker>yarprun</broker>
    </module>

    <module>
        <name>gnome-system-monitor</name>
        <node>localhost</node>
        <broker>ssh</broker>
    </module>

    <application>LeftCameraCalibration</application>

    <connection>
        <from>/icub/cam/right</type>
        <to>/camCalibConf/image</port>
    </connection>
</application>
```

Module integration

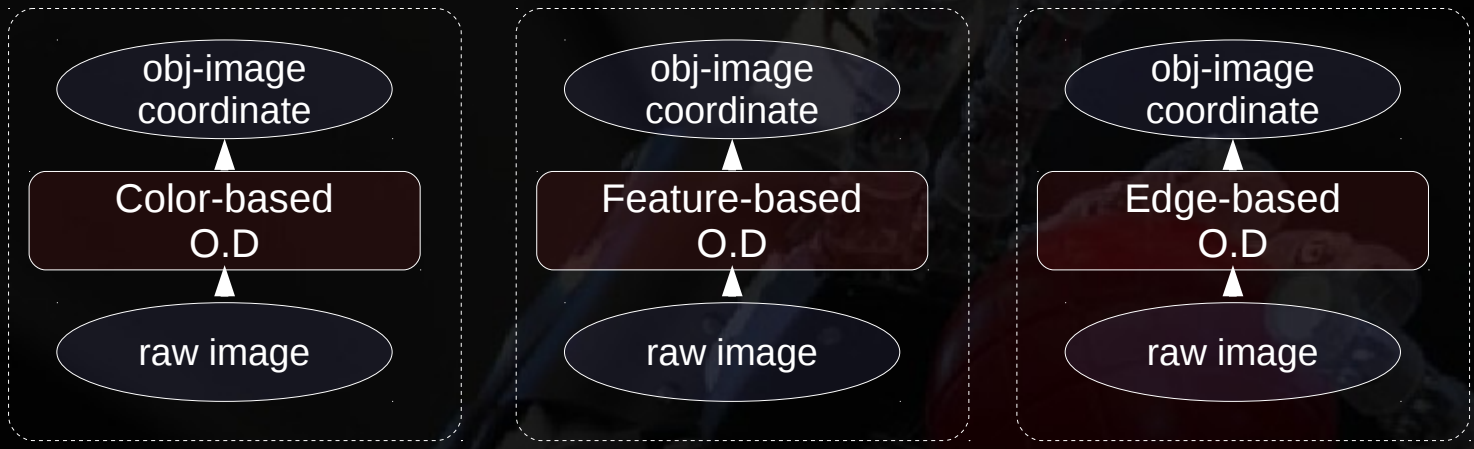


Automatic dependency resolver

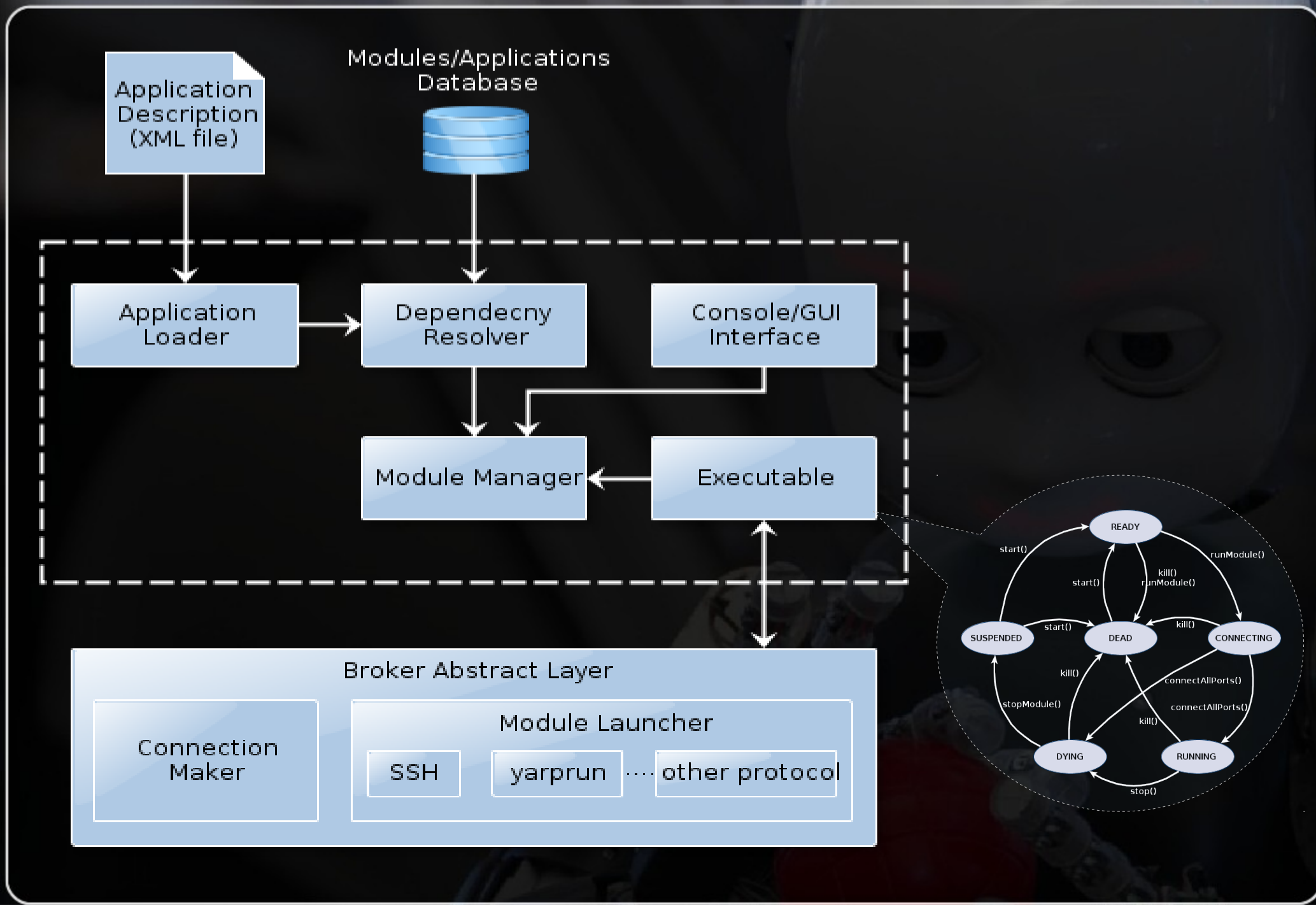


Selection policies

- Computational overload *(C)*
- Data rate *(P)*
- CPU utilization *(U = C/P)*
- Quality of data *(subjective)*
- Resource dependencies *(I/O, Memory, GPU, ...)*
- ...
- **User Rank** *(which can be a combination of all above criteria)*



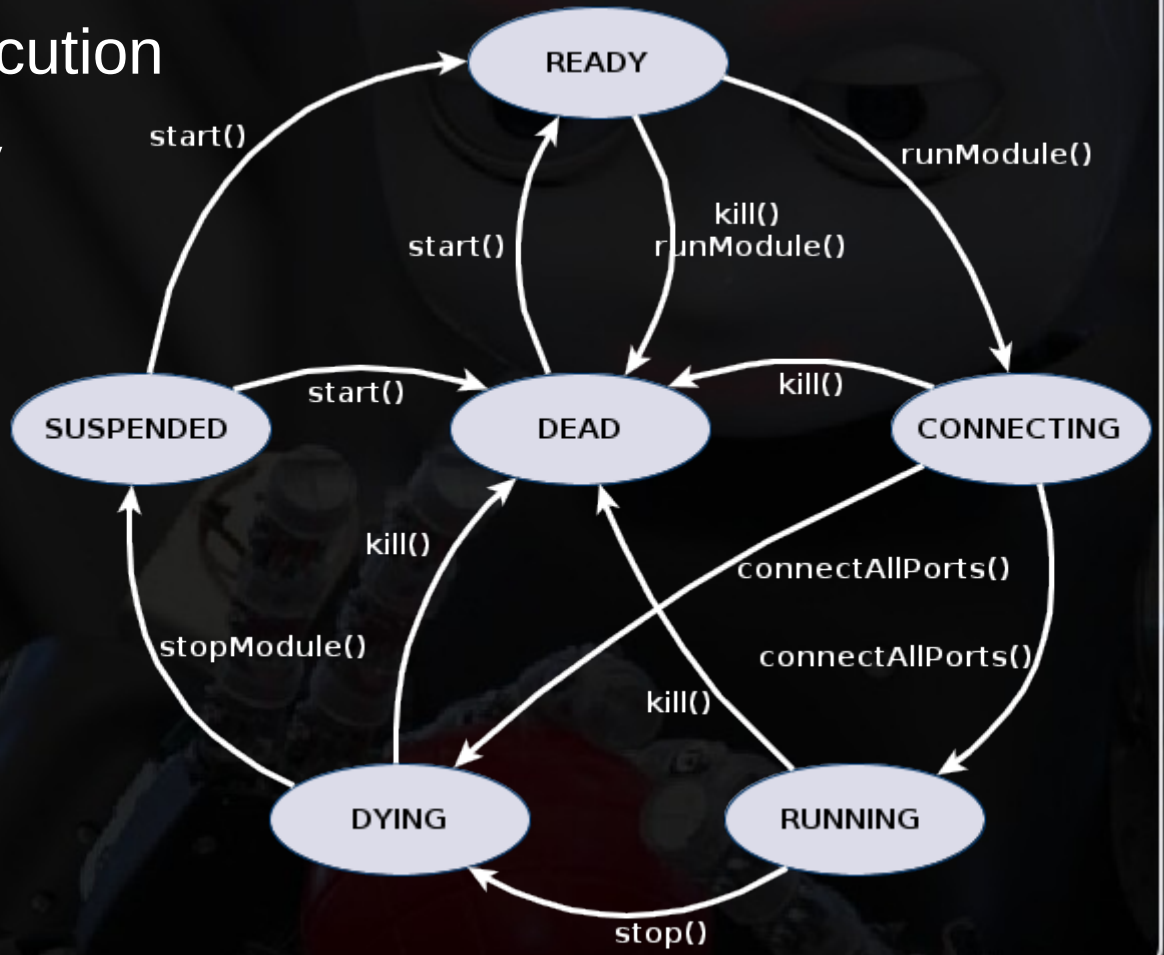
The system


Application Description (XML file)
Modules/Applications Database
Application Loader
Dependecny Resolver
Console/GUI Interface
Module Manager
Executable
Broker Abstract Layer
Connection Maker
Module Launcher
SSH
yarprun
other protocol
READY
start()
runModule()
kill()
runModule()
start()
SUSPENDED
start()
DEAD
kill()
CONNECTING
kill()
connectAllPorts()
stopModule()
connectAllPorts()
kill()
DYING
RUNNING
stop()

Executable state machine

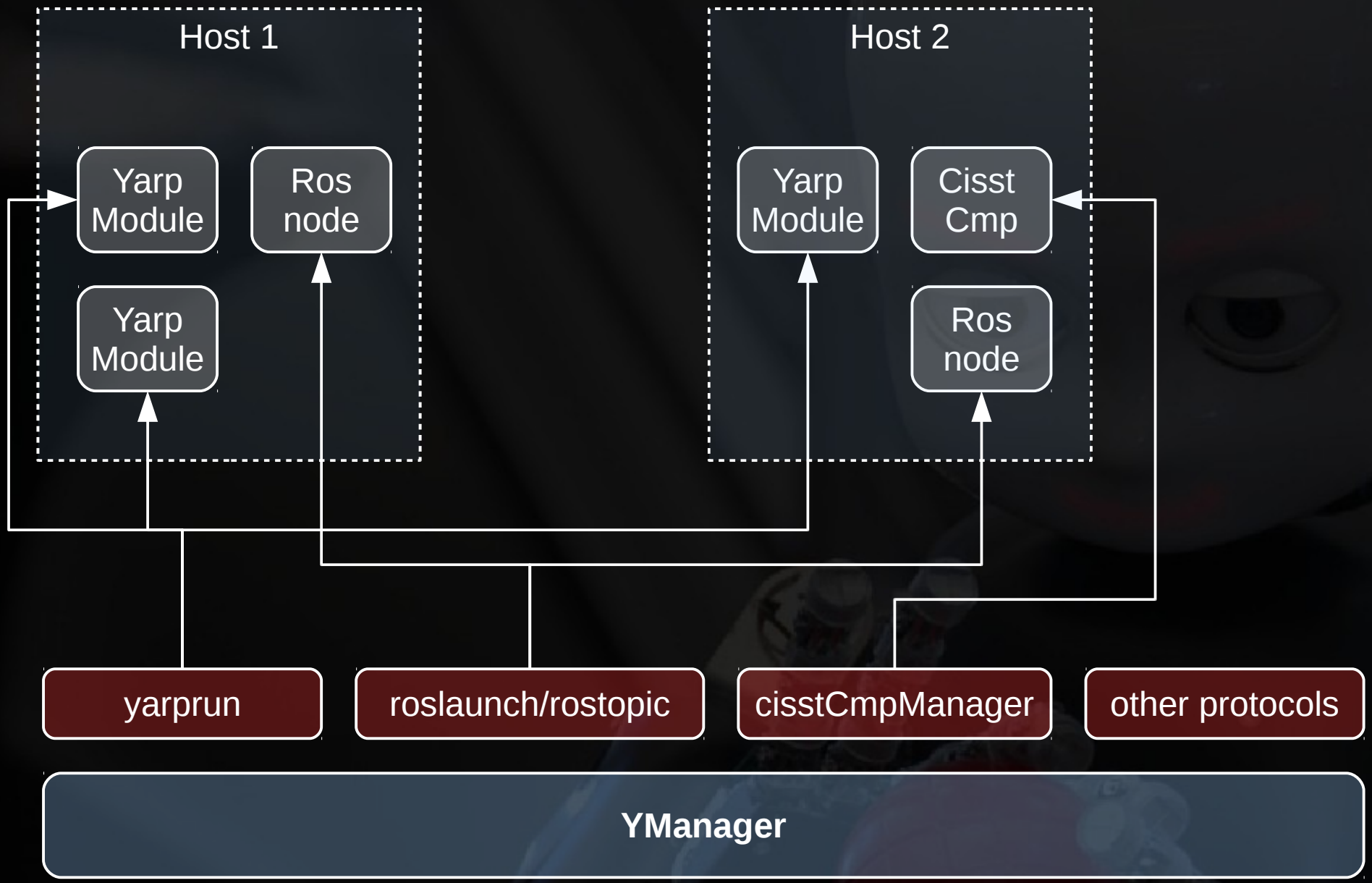
Advantages:

- Synchronous execution *(launching modules with priority)*
- Real-time module status checking *(context awareness)*
- Avoiding pooling/wait execution
- Real-time failure recovery

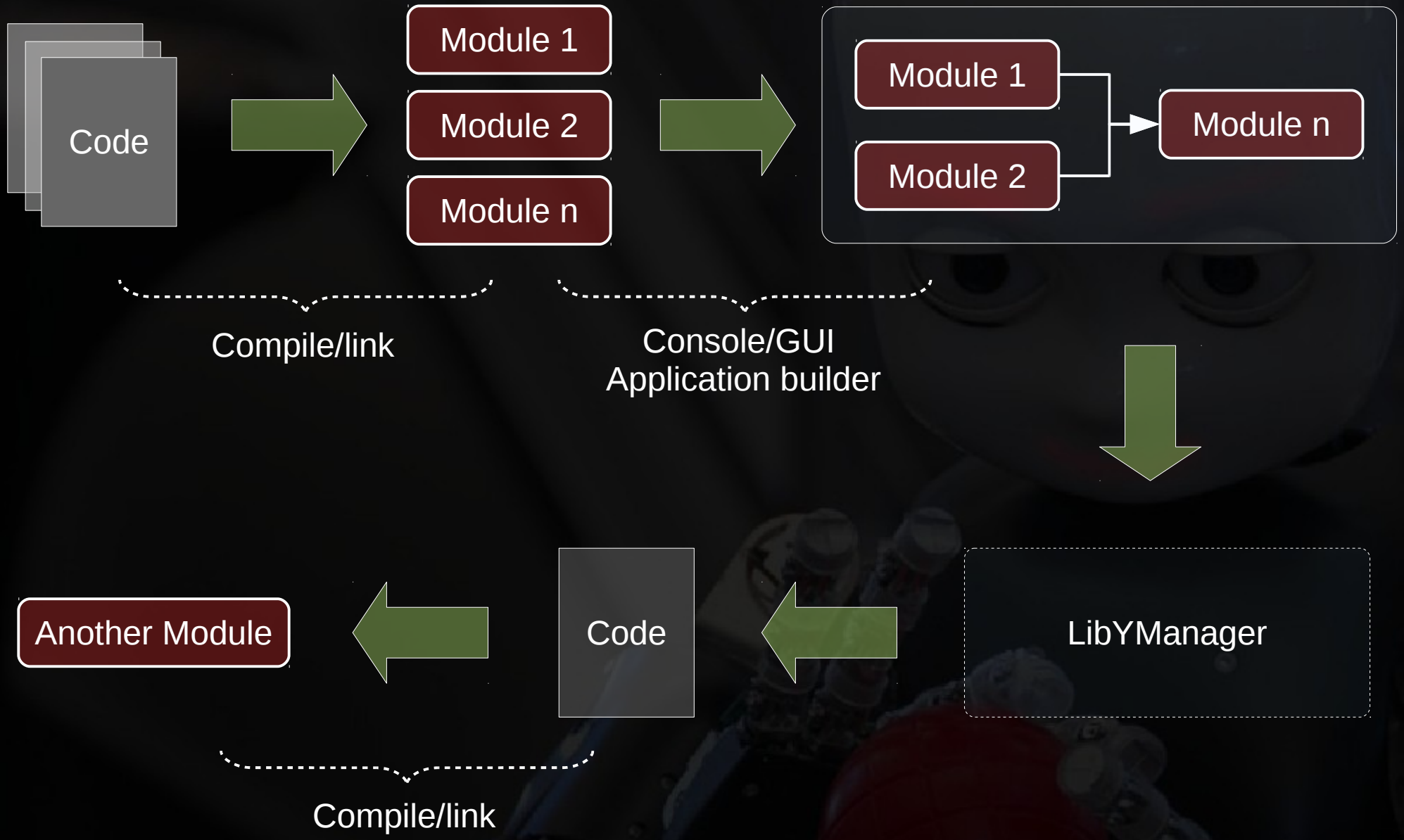


Most of the available robotics middlewares neglect these points!

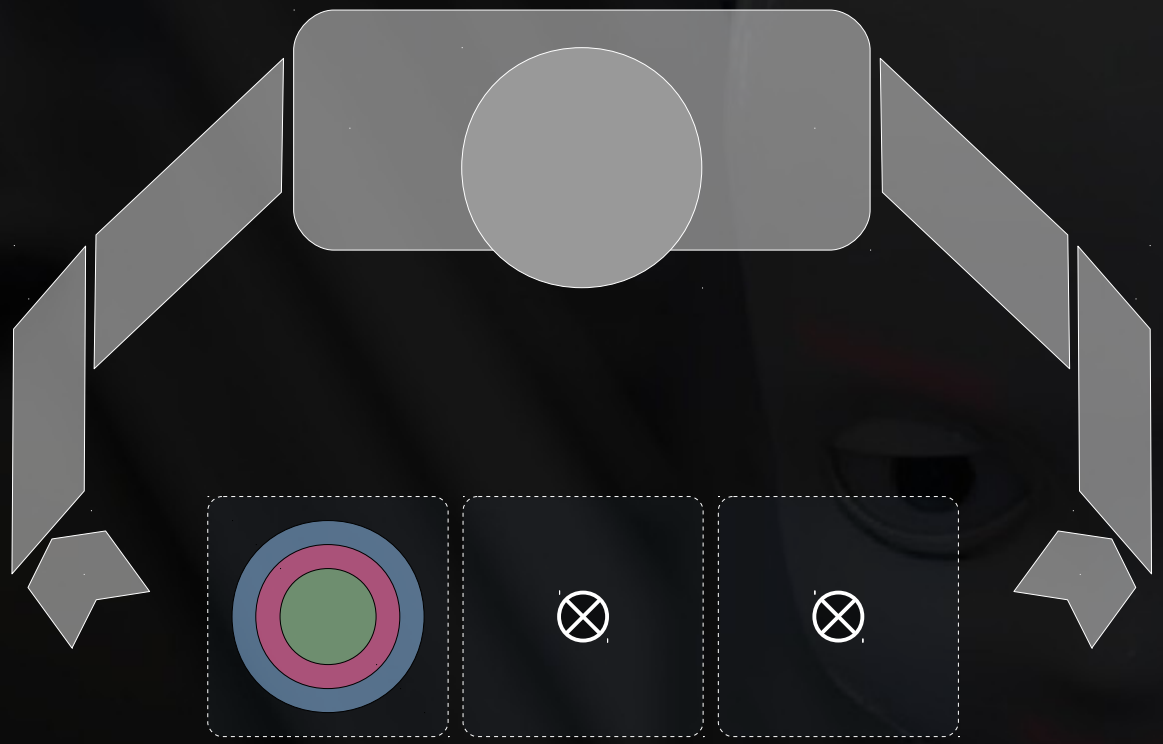
Cross-middleware management



Code/Application integration



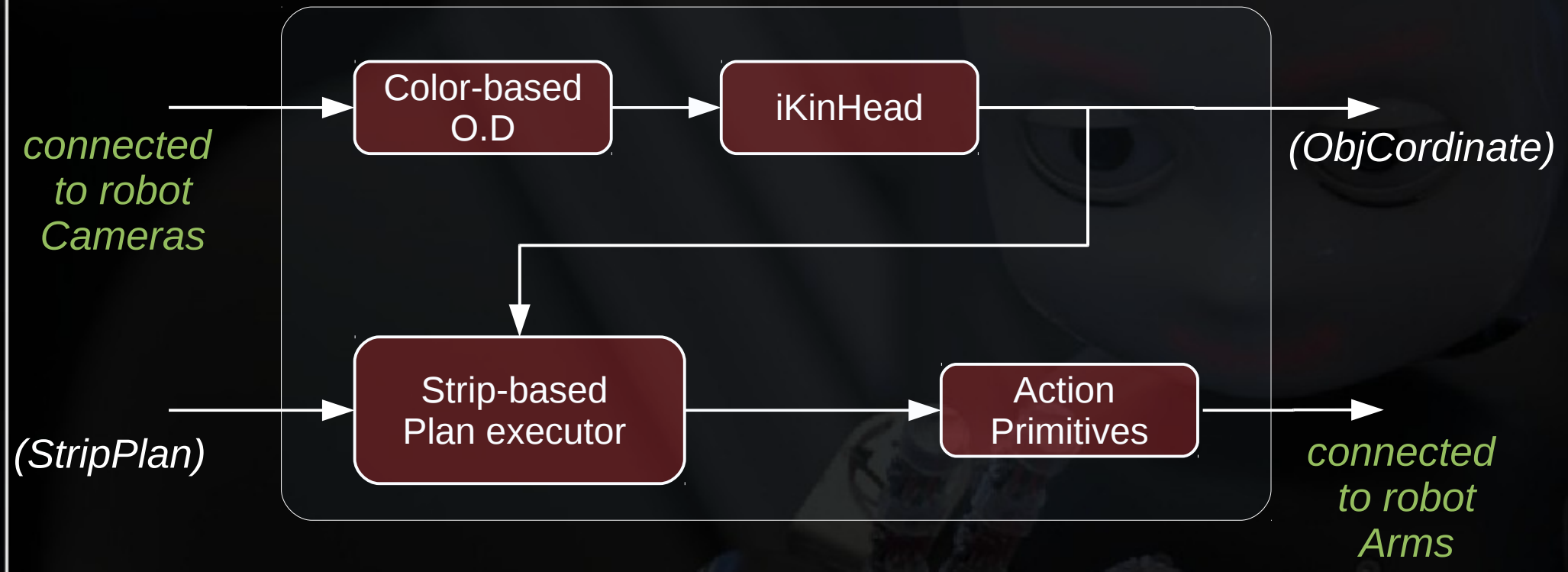
Code/Application example



From a Planer:

“PICK” GREEN A1 RIGHT
“PICK” RED A1 LEFT
“PUT” GREEN A3 RIGHT
...

Code/Application example



“SimpleDiskManipulation”

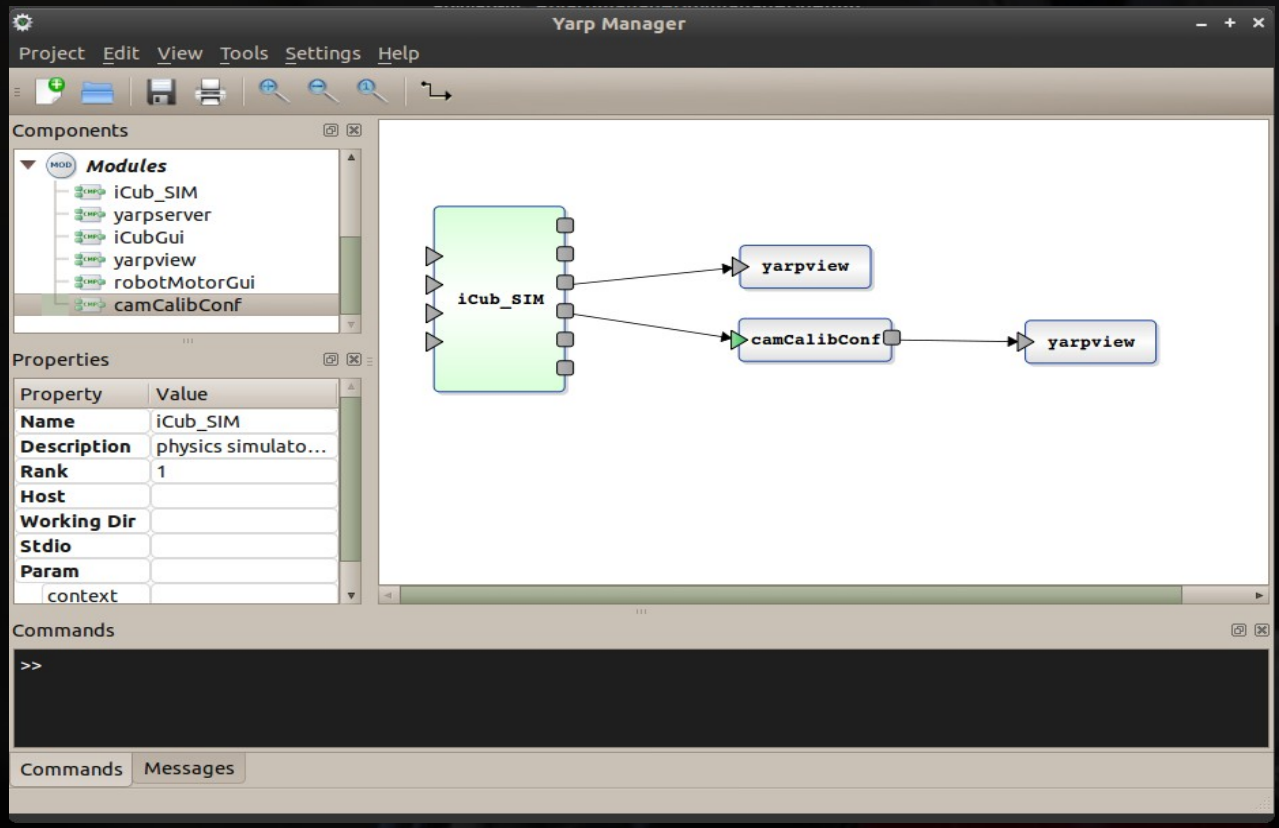
Code/Application example

I want to test my own planner on the robot. I have an application to recognize and manipulate simple disks. Then all I need is to use this application inside my test code...

```
LibYManager::loadApplication(“SimpleDiskManipulation”)
strObjPort = LibYManager::getDataPort(“ObjCoordinate”)
strPlanPort = LibYManager::getDataPort(“StripPlan”)
LibYManager::connect(myPlanPort.getName(), strPlanPort)
LibYManager::connect(myObjPort.getName(), strObjPort)
LibYManager::run()
...
myPlanner.updatePlannerDomain(myObjPort.read())
myPlanner.plan()
myPlanner.writePlan(MyPlanPort);
…
LibYManager::stop()
```

YManager toolkit

- A library for application integration/management (*libYManager*)
- Command line module manager (*ymanager*)
- Interactive GUI (*qymanager*)



Problems

Automatically connected
by YManager

YManager reasons on module dependencies and find a set of connections to established based on module description XML file.

```
<module>
    <name>Module1</name>
    <node>localhost</node>

    <parameters>--name myModule</parameters>

    <portmap>
        <old>/Module1/out</old>
        <new>/myModule/out</new>
    </portmap>

</module>
```

Summary

- Management of many software modules in a complex robotics application requires easier and more flexible tools
- Standard modules can be interconnected and integrated via proper tools for rapid application prototyping
- Automatic dependency resolving, modules execution and failure recovery can improve reliability of robotics application
- Code/Application integration and cross middleware management can help users to benefit from software modules written in other robotic frameworks
- A nice GUI is always welcome for users

Question and discussion

Thank you

and

please help me to

justify my ideas by your

valuable feedback.