OROCOS Cheat Sheet

Major Concepts

Component

- exposes an algorithm to the rest of the software
- defines inputs, outputs and parameters
- is run by an activity
- is compiled into a library
- offers and uses services
- installs in lib/orocos

Package

- is a directory on your filesystem
- contains one or more component, plugin or typekit libraries
- contains a manifest.xml file
- can be installed or used in-place

Deployment

- description of a (part of) an application
- in an XML or script (ruby, rtt, Lua) file
- creates, connects, configures and starts components
- allocates threads and sets connection policies

Data Types

Orocos C++ types ...

- must be default constructible
- must be copy-able
- may be primitive types, structs, sequences (std::vector or []) or any combination

typegen can use C++ types ...

- that have all members as publics
- that are not templated
- that have no parent class

typekits are ...

- required for each data type to be usable
- generated by typegen if possible
- hand-written in other cases

Properties

- are structured name-value pairs
- are the run-time parameters
- can be serialized to XML

Flow Ports

- publish and receive data for algorithms
- are In or Out and of a given data type
- Outs are send-and-forget
- Ins can wake us up (triggering)

Operations

- are plain C/C++ functions
- are 'sent' or 'called'
- run in the caller's thread or the component's thread
- are grouped into service objects

Service

- a collection of flow ports, properties and operations
- is provided to and required by others
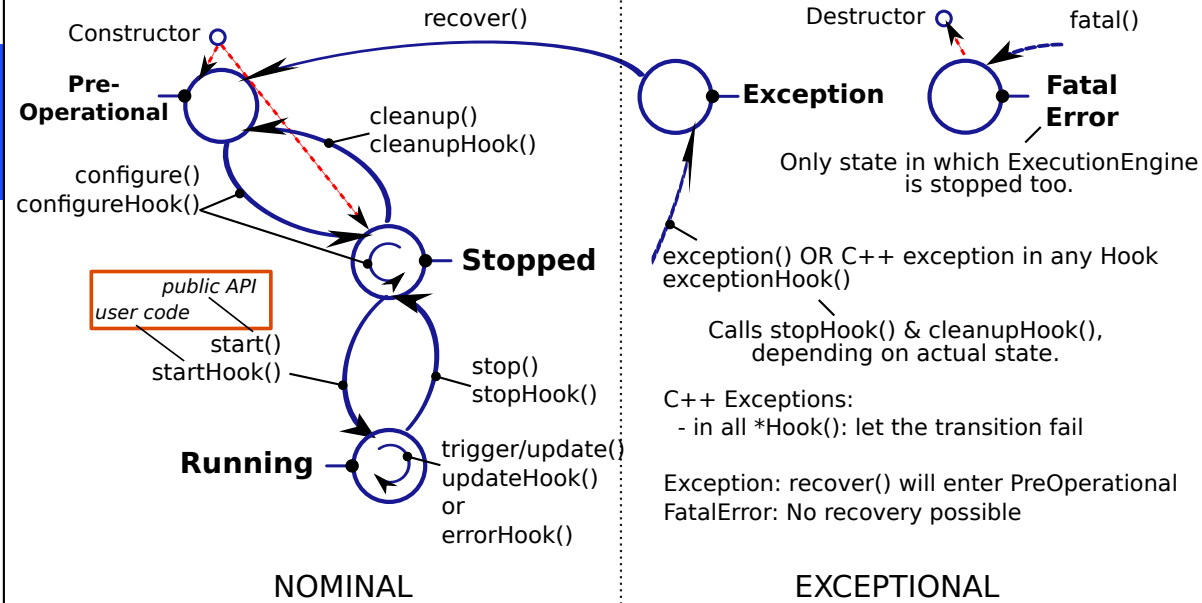- can be loaded at run-time in a component

Connection Policy

- defines the connection between an Input and Output port
- defines data buffering, locking mechanism, and initial state
- allows to specify a transport

Transports

- connect Orocos components to other robotics frameworks or protocols
- handle Orocos data types over a given protocol
- can support streaming, connection-oriented or service-oriented communication
- may or may not be hard real-time

Component Lifecycle StateMachine

exception() OR C++ exception in any Hook
exceptionHook()

Calls stopHook() & cleanupHook(), depending on actual state.

C++ Exceptions:
  - in all *Hook(): let the transition fail

Exception: recover() will enter PreOperational
FatalError: No recovery possible

Only state in which ExecutionEngine is stopped too.

NOMINAL   EXCEPTIONAL

Component Architecture

An **Activity** object executes the **ExecutionEngine**, which in turn processes incoming messages, plugin functions and finally updateHook() is called.