Advanced ROS2 Concepts

@arwo 2024

SCHANZER

RACING ELECTRIC

bit.ly/srd_arwo

bitbucket.org/schanzerracingelectric/arwo_advanced_ros2_concepts

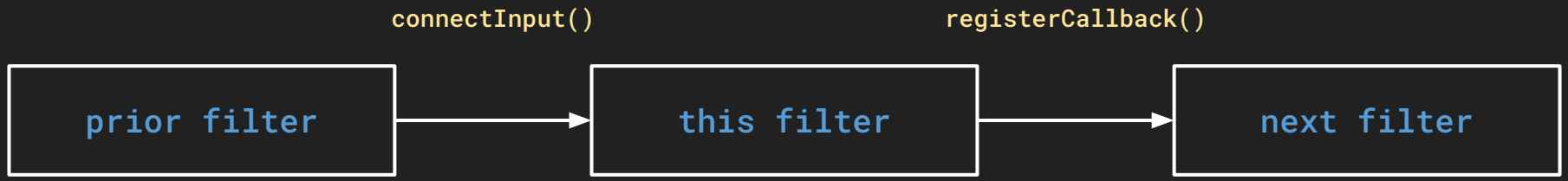
Agenda

1. Message Filters
2. Quality of Service
3. Lifecycle Management
4. Composition
5. Summary / Q&A

1. Message Filters

“A message filter is defined as something which a message arrives into and may or may not be spit back out of at a later point in time.”

1. Message Filters



1. Message Filters

- Utility library of commonly used message "filtering" algorithms.
- All message filters follow the same pattern for connecting inputs and outputs.
- Inputs are connected either through the filter's constructor or through the `connectInput()` method.
- Outputs are connected through the `registerCallback()` method.
- Note that the input and output types are defined per-filter, so not all filters are directly interconnectable.
- You can register multiple callbacks with the `registerCallbacks()` method. They will get called in the order they are registered.

1. Message Filters

Subscriber

Cache

Time Synchronizer

Policy-Based Synchronizer

Time Sequencer

Chain

1. Message Filters

Subscriber

- A ROS subscription wrapper
- Uses a ROS topic as its input

Time Synchronizer

- Synchronizes channels by timestamps
- C++ syncs up to 9 channels

Time Sequencer

- Ensures messages are processed in timestamp order
- Uses a specific delay before processing

Cache

- Stores message history
- Supports cache interval extraction
- Uses current ROS time if message lacks a timestamp

Policy-Based Synchronizer

- Synchronizes channels by timestamps
- C++ syncs up to 9 channels

  ExactTime Policy
  - Matches messages with the same timestamp

  ApproximateTime Policy
  - Matches messages using an adaptive algorithm
  - Uses current ROS time for missing timestamps

Chain

- Chains single-input/single-output filters dynamically
- Filters auto-connect in added order
- Useful for runtime filter determination

2. Quality of Service

“ROS 2 offers flexible QoS policies for node communication balancing the reliability of TCP or the best-effort nature of UDP with many possible states in between.”

2. Quality of Service

History

- Keep last
- Keep all

Depth

- Queue size

Reliability

- Best effort
- Reliable

Durability

- Transient local
- Volatile

Deadline

- Duration

Lifespan

- Duration

Liveliness

- Automatic
- Manual by topic

Lease Duration

- Duration

2. Quality of Service

History
- **Keep last:** Stores up to N samples. Configurable via queue depth.
- **Keep all:** Stores all samples, but within the limits of middleware resources.

Depth
- **Queue size:** Relevant only when history is set to "keep last".

Reliability
- **Best effort:** Tries to deliver samples but can lose them if the network is unstable.
- **Reliable:** Ensures sample delivery and may attempt multiple retries.

Durability
- **Transient local:** Publisher holds responsibility for saving samples for late-joining subscriptions.
- **Volatile:** No effort to save samples.

Deadline
- **Duration:** Specifies the maximum time between messages on a topic.

Lifespan
- **Duration:** Sets the maximum time between publishing and message reception, without considering the message stale.

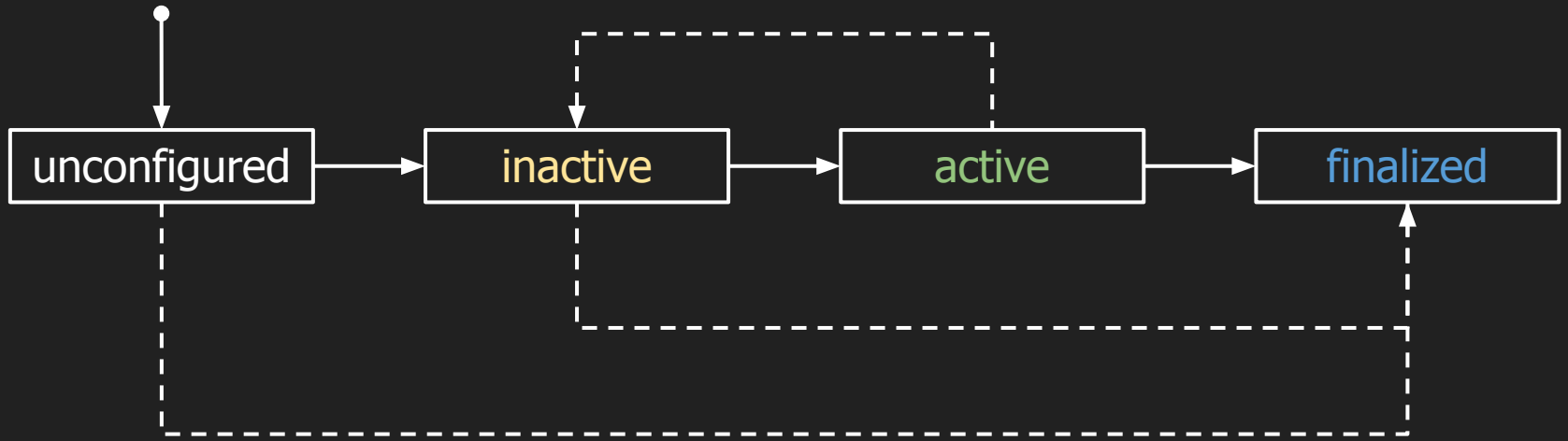
Liveliness
- **Automatic:** Nodes remain alive when any publisher sends a message.
- **Manual by topic:** Node stays alive if manually asserted.

Lease Duration
- **Duration:** Sets the time a publisher has to indicate its liveliness.

3. Lifecycle Management

“A managed life cycle for nodes enables greater control over the state, ensures that components have been instantiated correctly before execution and allows nodes to be restarted or replaced on-line.”

3. Lifecycle Management



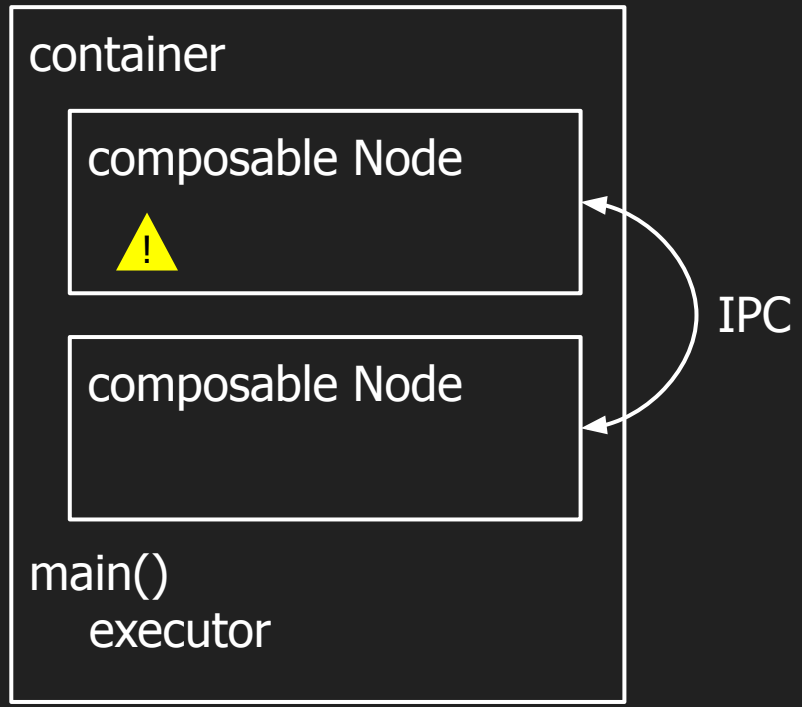
3. Lifecycle Management

- ROS 2 Humble introduced lifecycle nodes, also known as managed nodes, transitioning through four states: `unconfigured`, `inactive`, `active`, and `finalized`.
- Lifecycle nodes offer finer control over robotics systems, allowing reconfiguration and reactivation at runtime.
- Lifecycle nodes allow reconfiguration without affecting running behavior and provide greater activation control.
- Each state transition is triggered by a specific action, ensuring deterministic behavior.
- Each transition is logged, offering visibility into the node's current state.
- Lifecycle nodes enhance robotics development by ensuring correct configuration and activation, improving predictability and reliability.

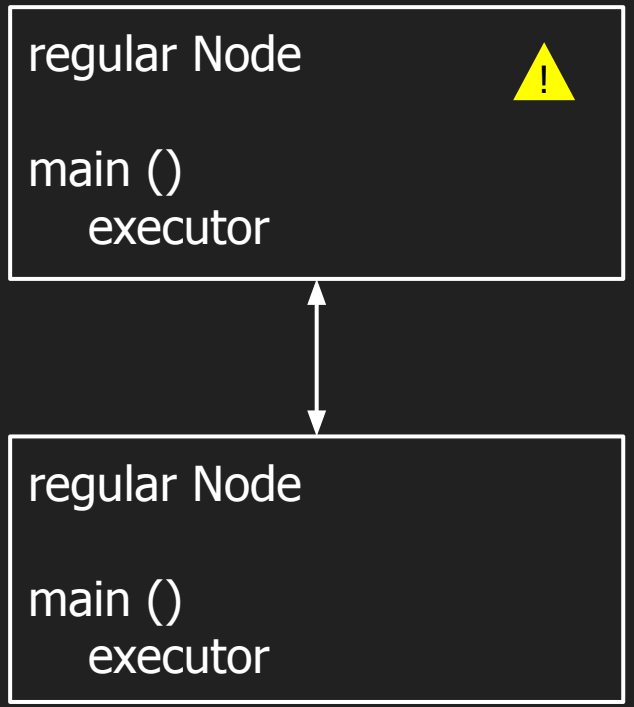
4. Composition

“ROS2 components inherit nodelet concepts from ROS1, serving as lightweight variants of nodes, while using the same API as ROS2 nodes.”

4. Composition



IPC



4. Composition

link time

dlopen

compile time

runtime

4. Composition

`linktime`

composition with CMakeLists linking

pros:

- lead to faster startup times and improved runtime performance
- simplifies deployment by reducing number of files
- easy debugging and dependencies

cons:

- statically linked components requires recompilation
- executables are larger in size compare to dynamically linked libs

`compile time`

composition inside cpp source

pros:

- there is no runtime overhead associated with dynamic loading or linking
- allows statistical analysis of the software system
- ensures deterministic behavior of the system

cons:

- changes requires recompilation
- resolving dependencies and configurations at compile time can increase build times
- may lead to code duplication or bloat, resulting in larger executables or libraries

`dlopen`

composition at runtime with command line arguments

pros:

- modularity
- reduced compilation time for large projects
- allows adding functionalities through dynamically loaded plugins without modifying core executable

cons:

- dependencies management challenges
- potential runtime errors, memory leaks, segmentation fault etc.
- performance overhead

`runtime`

composition in launch file or with services at runtime

pros:

- only way to change components after launch
- high flexibility

cons:

- see dlopen

Summary / Q&A

Thank you for listening!

source code

bit.ly/srd_arwo

feedback

leonce.mollerus@schanzer-racing.de
robert.kalmar@schanzer-racing.de

SCHANZER

RACING ELECTRIC