

AutowareNode base class for Lifecycle and Monitoring #3194

xmfcx started this conversation in **Design**



xmfcx on Jan 13, 2023

Maintainer


edited ▾

Problems being addressed

- It's hard to know what Autoware nodes are running currently
- It's hard to shut down a specific running node
- Not every node is being monitored automatically
 - For continuity (heartbeat/watchdog) (currently we have [topic_state_monitor](#))
 - For state (warning/error) (currently we have [system_error_monitor](#))
- Current monitoring system doesn't make use of DDS Liveliness like in https://github.com/ros-safety/software_watchdogs

Proposal

To address these issues, I've started working on this PR:

-  [feat: add autoware_node and autoware_control_center](#) [autoware.core#73](#)

I'd like to create a base class named `AutowareNode` in Core directory of Autoware.

And a node named `Autoware Control Center (ACC)` which will communicate with all the derived nodes from `AutowareNode`

We will have all new Autoware Core nodes to inherit from this new `AutowareNode`.

The `AutowareNode` will inherit from [rclcpp_lifecycle::LifecycleNode](#)

Interface

Between `AutowareNode`s and `ACC`

- Registering interface for new `AutowareNode`s to the `ACC`
- Reporting of current `AutowareNode` state to the `ACC`
- Control `AutowareNode` state from the `ACC`
 - shut down the node
 - toggle between active/inactive state (?)
- Low overhead watchdog monitoring

Category

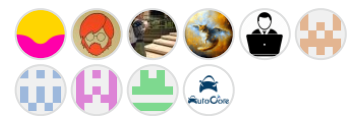


Design

Labels

version:autoware-c...

10 participants



Between AutowareNodes

- We can have monitored subscribers and publishers (See previous discussion in [Detect when nodes' incoming messages are skipped](#))

Similar Implementations

When I was looking for how I could allow Composable nodes and Lifecycle together, I came across the nav2 implementation.

Here are several links from nav2:

- <https://navigation.ros.org/concepts/index.html#lifecycle-nodes-and-bond>
- <https://navigation.ros.org/configuration/packages/configuring-lifecycle.html>
- https://github.com/ros-planning/navigation2/blob/main/nav2_util/include/nav2_util/lifecycle_node.hpp

They don't have all the functionalities I'm planning for our implementation but it's a solid example on how it can also be done.

They also make use of `bondcpp` for continuity monitoring of the nodes. But in the following discussion link, it's stated that it is there for legacy purposes and they are open to replacing it with modern liveliness DDS capability:

- https://answers.ros.org/question/380104/bond-vs-software_watchdog-ros2/

Related Links

- Autoware.Auto discussions:
<https://gitlab.com/autwarefoundation/autware.auto/AutowareAuto/-/issues/821>
- Discuss Implementing LifeCycleNode Support on Autoware.Auto Nodes -
<https://gitlab.com/autwarefoundation/autware.auto/AutowareAuto/-/issues/282>
- Autoware Diagnostics and Monitoring -
<https://gist.github.com/xmfcx/7eaae9750d6317a3a2aa23745ac99444>

Discussion

I'm working on the initial implementation on [autwarefoundation/autware.core#57](#) and I'm open to any design suggestions, criticisms and ideas. Please tell me how you feel about this proposal!

cc. [@isamu-takagi](#) [@yukkysaito](#) [@kenji-miyake](#) [@mitsudome-r](#) [@wep21](#) [@nabetetsu](#) [@takeshi-iwanari](#)



VRichardJP on Jan 13, 2023 Collaborator

I really would like a nice monitoring tool too.

I have few questions:

- How would we interact with ACC? Through some cli/GUI application?
- If ACC is used for heartbeat and watchdog monitoring, would we get rid of the current diagnostic aggregator system?
- What do you mean by monitoring subscribers/publishers. Besides basic info such as who subscribe/publish what, could you get data such as frequency, latency, drop rate, etc?



1



1

3 replies



xmfxc on Jan 13, 2023 Maintainer Author

edited ▼

@VRichardJP Thanks for reviewing the proposal :)

GUI interaction

How would we interact with ACC? Through some cli/GUI application?

For this I see 2 options:

1. ACC provides its own GUI/cli
 - The most efficient way (no additional data transfer)
 - Harder to make it part of existing GUIs
2. ACC provides an interface for GUI tools.
 - We can create a GUI to showcase its capabilities.
 - Anyone can customize the reference GUI or make their own

For sake of flexibility, I will probably go with 2nd option. What do you think?

What happens to existing tools?

If ACC is used for heartbeat and watchdog monitoring, would we get rid of the current diagnostic aggregator system?

Yes, this new system would make some of the existing diagnostic tools obsolete.

But my plan is to implement the capabilities step by step.

Current tools that could be affected are:

- https://github.com/autowarefoundation/autoware.universe/tree/main/system/component_state_monitor
- https://github.com/autowarefoundation/autoware.universe/tree/main/system/system_error_monitor
- https://github.com/autowarefoundation/autoware.universe/tree/main/system/topic_state_monitor
- https://github.com/ros/diagnostics/tree/humble/diagnostic_aggregator

I didn't go through how each of them are affected by this proposal yet.

Monitored publishers, subscribers?

What do you mean by monitoring subscribers/publishers. Besides basic info such as who subscribe/publish what, could you get data such as frequency, latency, drop rate, etc?

For this I recommend to check

https://gitlab.com/autowarefoundation/autoware.auto/AutowareAuto/-/merge_requests/1205/diffs

Basic premise is, we don't want to monitor each topic with an outsider tool

- thus increasing network overhead for large messages
- like [topic_state_monitor](#)
Instead, we'd like each subscriber to monitor the topics they subscribe to.
With functionality like:
 - Allow subscribers to express what rate range they expect
 - What rates a publisher is expected to publish
 - If a mismatch occurs, they could report to `ACC`
 - If a critical subscriber doesn't have the topics it needs, it could report to `ACC`
 - If a topic misses its deadline, the subscriber could initiate a warning state and report to `ACC`
 - The deadlines could be set from subscribers maybe?

All of these functionalities can be possible because we can define custom subscribers and publishers and we have a common monitoring tool to report to.

Normally all these would be hard to implement in each node. But with the `AutowareNode` base class, most of the complexity would be hidden and be done automatically.

The users implementing new nodes only define the optional deadlines, expected rates etc.



VRichardJP on Jan 13, 2023 Collaborator

edited ▼

- For instance a `/acc/status` and `/acc/command` and `rqt` plugin (like `/diagnostics`)? or a dedicated non-ROS interface?

- I am not sure how the topic watchdog would work. Would you send a notification on each message published/received?
- If I understand the proposal from autoware auto, each node monitor its own publishers/subscribers and only reports errors when detected. In that case, I think it is not possible to detect when a callback is stuck, and thus the whole pipeline as well.
- Would this work in the case multiple nodes publish on the same topic? (I am not sure there is any in Autoware though.)
- Same question with synchronized subscriptions. Again, I am not sure it is used in Autoware a lot, I have seen it mostly in vehicle interface/driver code. But it is typically part of the pipeline that is worth monitoring.



xmfcx on Jan 15, 2023

Maintainer

Author

@VRichardJP

Interface

For instance a `/acc/status` and `/acc/command` and `rqt` plugin (like `/diagnostics`)? or a dedicated non-ROS interface?

For this I'm planning to use service interface of ROS2. For reporting warnings, registering and similar non-continuous calls. These will be handled by asynchronously by the ACC.

Watchdog

I am not sure how the topic watchdog would work. Would you send a notification on each message published/received?

I know how `bondcpp` works, by sending continuous messages and if something is missed, raise a callback.

But I haven't looked into https://github.com/ros-safety/software_watchdogs thoroughly.

Also this will be a watchdog for the node, not each particular topic.

Monitored subscriber callback getting stuck

If I understand the proposal from autoware auto, each node monitor its own publishers/subscribers and only reports errors when detected.

True.

In that case, I think it is not possible to detect when a callback is stuck, and thus the whole pipeline as well.

In the current implementation, yes, it will be. I haven't put too much thought into this but this is one of the challenges in making it work as intended.

One of the solutions that comes to my mind is to have an async watcher get created for each subscriber. And every time the subscriber receives a message, it notifies the watcher thread. And watcher notifies ACC if the subscriber it's watching gets stuck or doesn't receive anything for a while.

Multiple publishers on same topic

Would this work in the case multiple nodes publish on the same topic? (I am not sure there is any in Autoware though.)

It would ensure a topic is being published in a required rate, but wouldn't be able to confirm if each publishing source is contributing enough. But I think this is not so important for autoware.

Monitored synchronized subscribers

Same question with synchronized subscriptions. Again, I am not sure it is used in Autoware a lot, I have seen it mostly in vehicle interface/driver code. But it is typically part of the pipeline that is worth monitoring.

I think monitoring the callback would work the same. If we apply the solution I mentioned above. But I will take a note as an implementation task, thanks for bringing it up!



ralwing on Jan 13, 2023

This idea sounds really great! I have some suggestions to be considered:

- The `ACC` should allow creating multiple simultaneously working instances. I think that there might be a few instances of it, especially when there'll be a lot of working Nodes. And there also might be a one general `ACC`, which observes the local `ACC`s.
- The `AutowareNode` `state` `UnConfigured` and `InActive` may require getting statuses of other nodes. E.g. a group of fully configured nodes would instantly change the state to `Active`. This logic requires some kind of Nodes grouping facility, looks a bit like the `Saga Pattern`



1



1

1 reply



xmfcx on Jan 13, 2023 Maintainer Author

The `ACC` should allow creating multiple simultaneously working instances. I think that there might be a few instances of it, especially when there'll be a lot of working Nodes. And there also might be a one general `ACC`, which observes the local `ACC` s.

Why do you think so? What would be the benefit of having multiple `ACC` s?

And for this to work, `ACC` would need to derive from `AutowareNode` too and relaying warning messages higher up could get complicated.

The `AutowareNode` [state](#) `UnConfigured` and `InActive` may require getting statuses of other nodes. E.g. a group of fully configured nodes would instantly change the state to `Active`. This logic requires some kind of Nodes grouping facility, looks a bit like the `Saga Pattern`

Yes, if implemented correctly, we could add activation-dependencies for `AutowareNode` s.

When everything is launched, the `AutowareNode` s without activation-dependencies gets activated normally.

And as they get activated, they trigger other `AutowareNode` s to activate too.

And if they are not activated, they report to `ACC` for their requirements not met.



yukkysaito on Jan 13, 2023 Maintainer

Sounds good.

Please let me check to make sure.

Does this mean that `ACC` has the following functions?

- Check that a node has not died using heartbeats.
- Send shutdown command and change state command

Also, what is the use case for this functionality?

I would like to know the future road map after implement this function.



1



1

5 replies



xmfcx on Jan 13, 2023 Maintainer Author

Does this mean that `ACC` has the following functions?

Yes.

Check that a node has not died using heartbeats.

So that when something goes wrong and a node dies, we will know which node has died.

Send shutdown command and change state command

In ROS1 we could call `rostop kill /node_name` to kill the node that we like. This could be for debugging purposes generally.

But in ROS2 the alternative is to use the Lifecycle Nodes. This new architecture will enable us to do the same.



xmfcx on Jan 13, 2023 Maintainer Author

For the roadmap, once we discuss the details, I'd like to generate tasks and a timeline.

For now I'm implementing the registration stage in the PR [autowarefoundation/autoware.core#57](https://github.com/autowarefoundation/autoware.core/pull/57)



yukkysaito on Feb 19 Maintainer

[@xmfcx](#) Thank you for your reply!

I have an additional question. In ROS1, there was a feature similar to Autoware Control Center called rosmaster. However, the ROS master is well-known to be a single point of failure. As such, my understanding is that ROS2 has moved towards a masterless approach. Will this type of issue not occur with the current implementation?

- <https://medium.com/@oelmofty/ros2-how-is-it-better-than-ros1-881632e1979a>
- <https://roscon.ros.org/2017/presentations/ROSCon%202017%20D%20MTCP.pdf>



xmfcx on Feb 19 Maintainer Author

edited ▼

[@yukkysaito](#)

Autoware Control Center (ACC) is not like rosmaster.

ACC is more like the [system_error_monitor](#) and the [topic_state_monitor](#).

ACC is an efficient, lightweight and simple way to monitor nodes and raise emergencies when necessary.

It's a tool for debugging, understanding the cause of problems for developers or users.

We are still using the ROS2 infrastructure, and even utilizing liveliness, deadline features from DDS for lightweight monitoring.

I don't understand your exact concerns. ACC doesn't even have to be mandatory for everything else to work. It's just a useful helper tool.

Most of the functionality is embedded to the nodes themselves with the AutowareNode base class implementation (which inherits from the `LifeCycleNode`).



xmfcx on Feb 19 Maintainer Author

The monitoring nodes in systems will always be "single point of failure" and the way to mitigate risks is to make the implementation of these nodes simple and well tested.

JWhitleyWork on Jan 13, 2023

Collaborator

@xmfcx What types of functionality do you propose adding to `AutowareNode` that couldn't be achieved by utilizing `rclcpp_lifecycle::LifecycleNode` and enabling Topic Statistics without having to inherit from a custom class? I'm not saying that a custom class is a bad thing (it allows for the flexibility to update most ROS-based functionality from a single class) but it is non-standard and obfuscates the existing ROS2 functionality, thus making it less transparent and harder to enter the ecosystem for newer developers.

↑ 1

❤ 1

2 replies



xmfcx on Jan 13, 2023

Maintainer

Author

edited ▼

@JWhitleyWork thanks for the comment!

Here are the general expected features:

- Registering interface for new `AutowareNode`s to the `ACC`
- Reporting of current `AutowareNode` state to the `ACC`
- Control `AutowareNode` state from the `ACC`
 - shut down the node
 - toggle between active/inactive state (?)
- Low overhead watchdog monitoring

The `ACC` will be similar to [nav2_lifecycle_manager](#) in some ways.

The most important feature will be the luxury of seeing the list of `AutowareNode`s from a central place.

- Their states (error/warning/dead/inactive)
- Also watchdog monitoring will come automatically for each node.
 - The developer won't need to deal with setting it up.
 - If a node just inherits from the `AutowareNode`, this will be done automatically.

I know that controlling states of the nodes are pure `rclcpp_lifecycle::LifecycleNode` features.

But having these nodes registered to a central place will enable us to control which of these to shutdown in a more managed way.

Also paving way for creating guis for shutting down specific nodes. Similar to a task manager. Right now it is like looking for a needle in a haystack in `htop` if you need to kill a specific node.

Topic Statistics

From what I see in

- <https://docs.ros.org/en/rolling/Concepts/About-Topic-Statistics.html>
- <https://docs.ros.org/en/rolling/Tutorials/Advanced/Topic-Statistics-Tutorial/Topic-Statistics-Tutorial.html>

The topic statistics is very similar to `ros2 topic hz /topic_name` with a bit more time related statistics.

This is not what I need from a `monitored_publisher` or `monitored_subscriber`.

I've listed these features in

<https://github.com/orgs/autowarefoundation/discussions/3194#discussioncomment-4677947> under title **Monitored publishers, subscribers?**

Basic premise is, we don't want to monitor each topic with an outsider tool

- thus increasing network overhead for large messages
- like [topic_state_monitor](#)

Instead, we'd like each subscriber to monitor the topics they subscribe to.

With functionality like:

- Allow subscribers to express what rate range they expect
- What rates a publisher is expected to publish
- If a mismatch occurs, they could report to `ACC`
- If a critical subscriber doesn't have the topics it needs, it could report to `ACC`
- If a topic misses its deadline, the subscriber could initiate a warning state and report to `ACC`
- The deadlines could be set from subscribers maybe?

All of these functionalities can be possible because we can define custom subscribers and publishers and we have a common monitoring tool to report to.

Normally all these would be hard to implement in each node. But with the `AutowareNode` base class, most of the complexity would be hidden and be done automatically.

The users implementing new nodes only define the optional deadlines, expected rates etc.

Topic statistics let you publish a `/statistics` topic for each topic and monitor them as it's needed.

If we use it for each topic, it will generate a lot of network traffic.

Instead, if we use `monitored_publisher` and `monitored_subscriber` wrappers, they can monitor their statuses themselves within the `AutowareNode` and they can report only if an anomaly occurs to the `ACC`.

I understand your concerns on how it might make things more complicated for new or even existing developers to develop for Autoware.

But I will try to find a balance on how it will benefit and how it will be easy and intuitive to use.



xmfcx on Jan 13, 2023 Maintainer Author

Also from the [nav2 page](#)

Within Nav2, we use a wrapper of LifecycleNodes, nav2_util LifecycleNode. This wrapper wraps much of the complexities of LifecycleNodes for typical applications.

It also includes a bond connection for the lifecycle manager to ensure that after a server transitions up, it also remains active. If a server crashes, it lets the lifecycle manager know and transition down the system to prevent a critical failure.

So I've also assumed it is ok to do it as long as it's justified.



isamu-takagi on Jan 13, 2023 Maintainer

edited ▼

it's a nice feature. The requirements I think are:

- ACC supports register and unregister function for restarting AutowareNode
- AutowareNode supports reconnection protocol for restarting ACC
- AutowareNode should work with or without ACC so that ACC is not a single point failure
- ACC should be redundant because it cannot detect its own errors
- AutowareNode can also be applied to class modules such as behavior_velocity_planner's scene module, so make it possible to create from existing AutowareNode or rclcpp::Node



1



1

3 replies



xmfcx on Jan 15, 2023 Maintainer Author

edited ▼

[@isamu-takagi](#) thanks for reviewing the proposal!

De-registering

- ACC supports register and unregister function for restarting AutowareNode
- AutowareNode supports reconnection protocol for restarting ACC

Yes, I've thought of the potential run orders like following:

- node starts after acc

- acc starts after node
- another node with same name tries to register
- node registers, then dies
- acc dies after a node registered

AutowareNode requiring ACC

AutowareNode should work with or without ACC so that ACC is not a single point failure

My thoughts on this is to go with a flag for AutowareNode named `require_acc` which is by default `false`.

For production, it can be enabled to be used.

But for debugging nodes, it will be disabled, so, it doesn't get in the way.

ACC Redundancy

ACC should be redundant because it cannot detect its own errors

I'm not sure if this is practical. My plan was to make it thoroughly tested and if it crashes, the car should stop / give control to the driver.

What would be your recommendations to make this happen?

Allow some of AutowareNode functionality to be used by non-lifecycle nodes

AutowareNode can also be applied to class modules such as `behavior_velocity_planner`'s scene module, so make it possible to create from existing AutowareNode or `rclcpp::Node`

This is something I'd like to avoid. I don't want non-lifecycle nodes to be in the Autoware Core.

This single interface will make it easier to maintain for both development and runtime.

Do you have any plans/ideas on why you'd like such a feature?



isamu-takagi on Jan 16, 2023

Maintainer

AutowareNode requiring ACC

My thoughts on this is to go with a flag for AutowareNode named `require_acc` which is by default `false`.

For production, it can be enabled to be used.

Can AutowareNode continue processing if ACC dies? No problem then.

ACC Redundancy

I'm not sure if this is practical. My plan was to make it thoroughly tested and if it crashes, the car should stop / give control to the driver.

What would be your recommendations to make this happen?

In addition to crashes, errors that detect abnormalities but are judged to be normal are also possible. I mentioned redundancy, but if the implementation is simple, there is also the approach of doing enough testing as you said. This depends on the safety analysis.

Allow some of AutowareNode functionality to be used by non-lifecycle nodes

I made a mistake. It was `rcldcpp::LifecycleNode`, not `rcldcpp::Node`. My idea is to group monitored interfaces as following.

```
MyNode::MyNode() : AutowareNode()
{
    auto module1 = AutowareNode(this, "module1")
    module1.create_subscription(...) // monitoring results
    can be filtered by "module1"
}
```



1



xmfxc on Jan 16, 2023

Maintainer

Author

I made a mistake. It was `rcldcpp::LifecycleNode`, not `rcldcpp::Node`. My idea is to group monitored interfaces as following.

Oh sure, this should be doable with minimal effort! Thanks!



esteve on Feb 3, 2023

Maintainer

[@xmfxc](#) one feature we'd like to have at TierIV is the possibility of enabling/disabling nodes. Originally we wanted to do it from `ros2 launch`, but its design doesn't really allow such things. Instead, one option would be so that nodes that inherit from `AutowareNode` have a parameter so that when launched via `ros2 launch`, it can be toggled to enable/disable the node. The Node would still be constructed, but its run logic wouldn't be triggered.



1

2 replies



xmfxc on Feb 3, 2023

Maintainer

Author

Yes, that's what the `toggle between active/inactive state` is for from the first post, under **Interface -> Between AutowareNodes and ACC -> Control AutowareNode state from the ACC**

It'd be possible to run everything at an inactive state and toggle them at will from the ACC.



esteve on Feb 3, 2023 Maintainer

Sorry, I missed that part. That sounds good to me.



mitsudome-r on May 9, 2023 Maintainer

@autowarefoundation/autoware-developers-autocore Could you read this proposal and give feedback on this?



1 reply



liuXinGangChina on May 11, 2023 Collaborator

This is a pretty big proposal let's discuss this on next week's ASWG meeting



Tom-Li-Lee on Jun 25, 2023 Collaborator

It is a very nice feature that provides a flexible monitoring mechanism based on LifecycleNode.

I have a few questions. Please correct me if my understanding is wrong.

1. The monitor procedure is done automatically. Based on LifecycleNode's state callback, AutowareNode can report state to ACC if some transition is triggered. User doesn't need report state directly.
2. How to deploy ACC? It is a universally unique node, and should be started before any other AutowareNodes, right?
3. ACC is not a static node and can be customized. If a node becomes error/warning/dead/inactive and known by ACC, the behavior to do can be customized.
4. Again, Single point failure should be taken into consideration.



1 reply



xmfcx on Jul 17, 2023 Maintainer Author

Hi @Tom-Li-Lee, thanks for your interest in the package.
Sorry for this very late reply 🙏, I thought I've sent the reply earlier but I had connectivity issues.

The monitor procedure is done automatically. Based on LifecycleNode's state callback, AutowareNode can report state to ACC if some transition is triggered. User doesn't need report state directly.

Correct.

How to deploy ACC? It is a universally unique node, and should be started before any other AutowareNodes, right?

I think it should support all the possibilities on run or crash orderings. Explained more in detail [in here above](#).

ACC is not a static node and can be customized. If a node becomes error/warning/dead/inactive and known by ACC, the behavior to do can be customized.

Once we add the behaviors, I think we can add customization features later too.

Again, Single point failure should be taken into consideration.

The aim is to make ACC simple and indestructible by vigorous testing. And if ACC fails, the vehicle should perform emergency braking or hand over to the driver.