MODELLING ROBOT ARCHITECTURES WITH AADL

GIANLUCA BARDARO, MATTEO MATTEUCCI

RATIONALE – WHY EXPLORE MODEL-BASED DESIGN

Write less code

Robot software development is based on middleware

Many reusable components

A lot of boilerplate code

Write better code

Hardware configuration influences complex functionalities

Very long computational pipelines

Many dependencies between elements

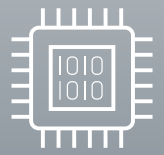
Better overview

Reusable components lead to a decentralised approach

No architectural view until runtime

No static check before deployment

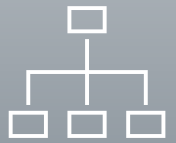
RATIONALE – WHY USE AADL



Hardware and software

Component-based design

Inheritance



Hierarchical structure

Architecture overview

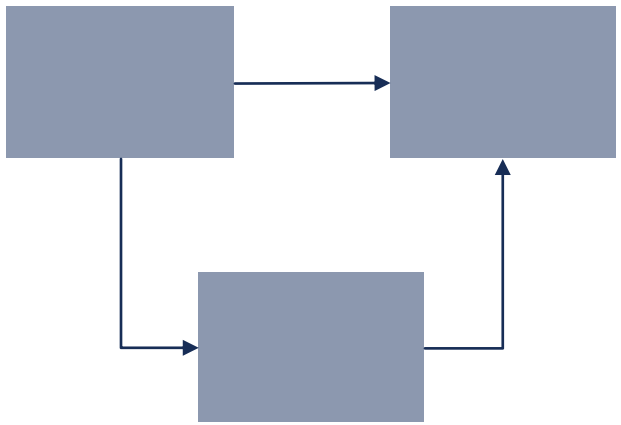
Static analysis

Graphical and textual syntax

Tool support

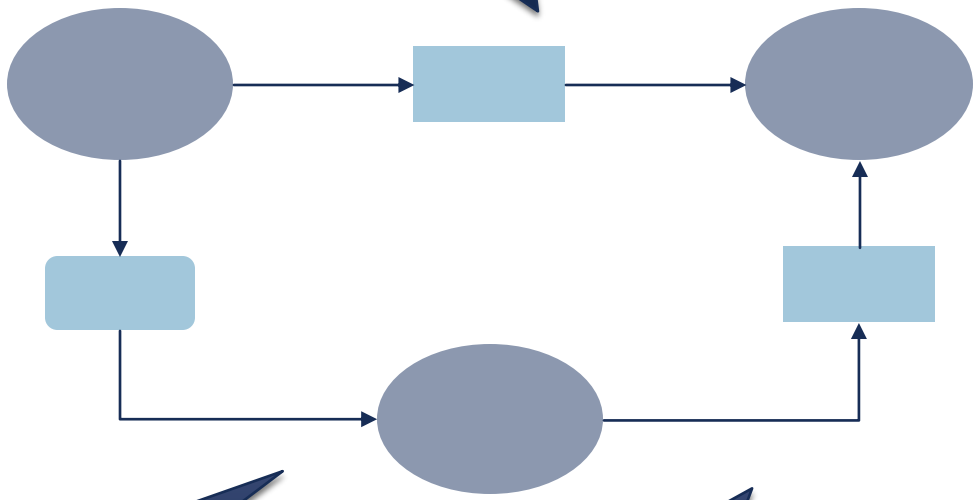
FROM COMPONENT-BASED TO COMPONENT-CONNECTOR

High-level abstraction



Strong emphasis on the component

Typed and named channels

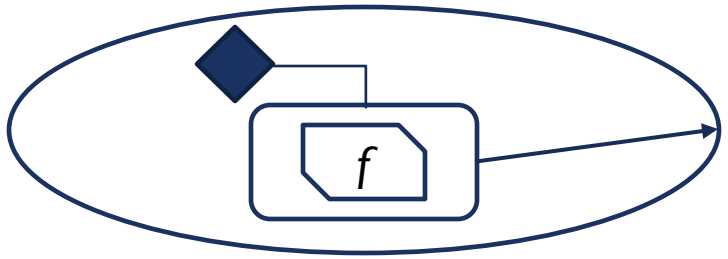


Components and connectors are first-class citizen

Well defined communication protocols

COMPONENT-CONNECTOR: A REFINEMENT

Source

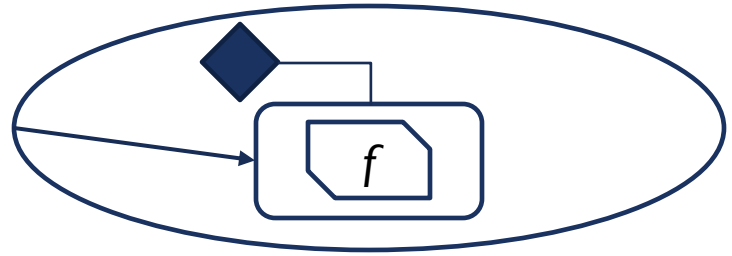


Components are too high-level

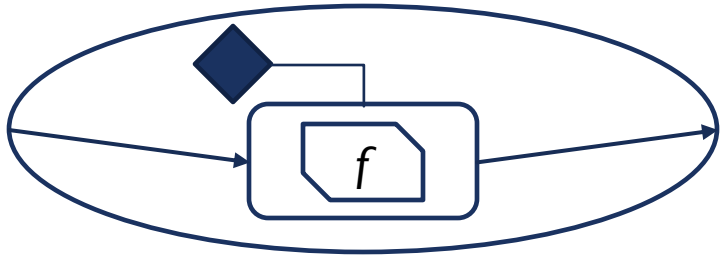
Identify composable component behaviors

Provide ready-to-use design building blocks

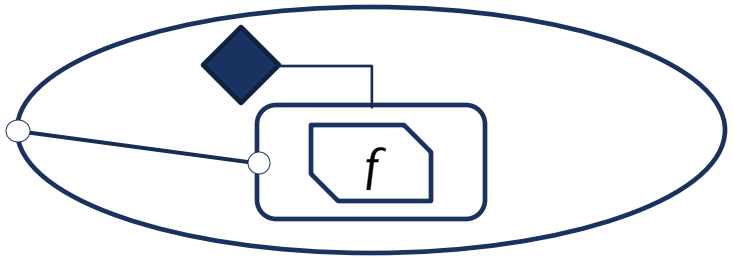
Sink



Filter



Reactive



FROM COMPONENT-CONNECTOR TO AADL


```
data internal_state end internal_state;

process component end component;
process implementation component.impl
  subcomponents
    internal_state: data internal_state;
end component.impl;

subprogram function
  features
    signal: out event port;
    ist: requires data access internal_state;
end function;
```

```
thread component_behaviour
  features
    ist: requires data access internal_state;
end component_behaviour;

thread implementation component_behaviour.impl
  subcomponents
    function: subprogram function;
  connections
    pc: data access function.ist -> ist;
end component_behaviour.impl;
```

FROM COMPONENT-CONNECTOR TO AADL

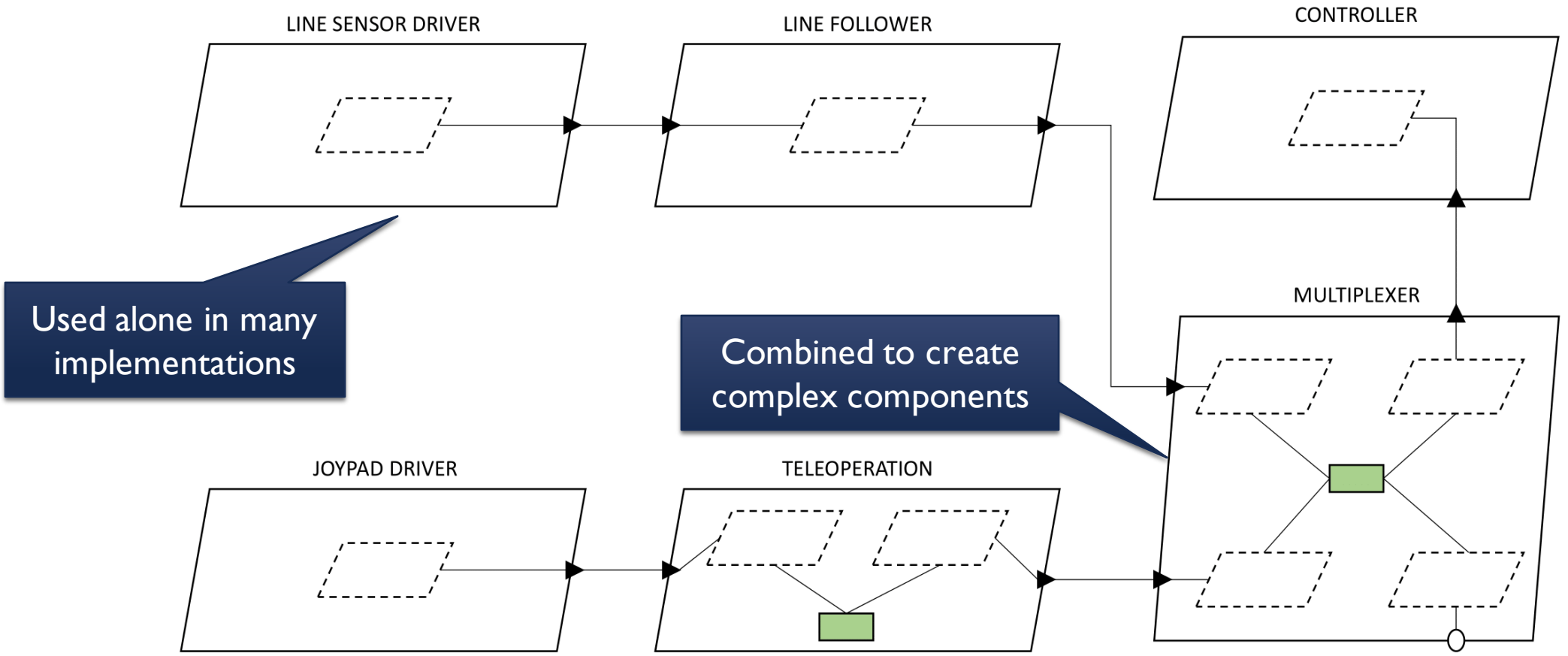

```
thread message_sink extends component_behaviour
  features
    msg: in event data port;
  properties
    Dispatch_Protocol => Aperiodic;
end message_sink;

thread message_source extends component_behaviour
  features
    msg: out data port;
  properties
    Dispatch_Protocol => Periodic;
end message_source;
```

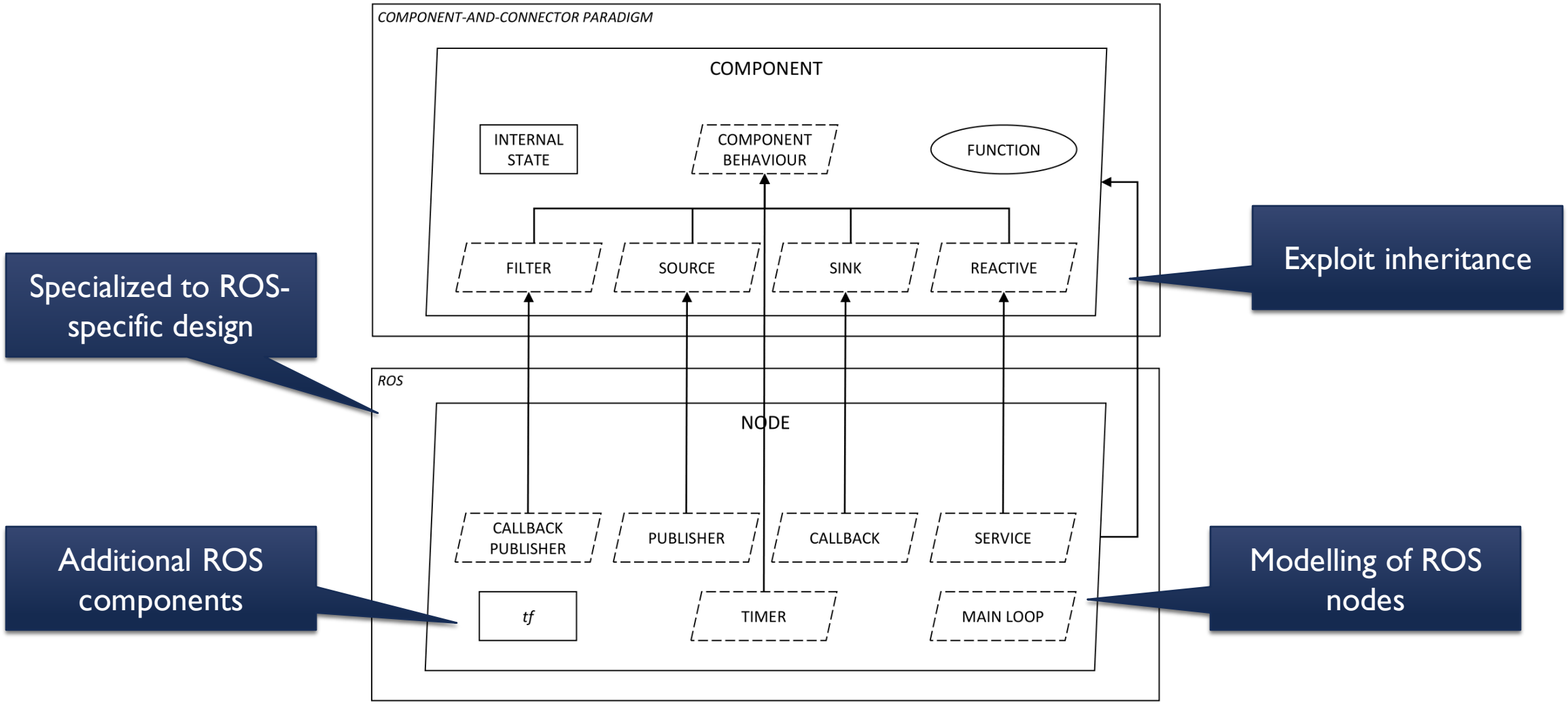
```
thread filter extends component_behaviour
  features
    msg_in: in event data port;
    msg_out: out data port;
  properties
    Dispatch_Protocol => Aperiodic;
end filter;

thread reactive extends component_behaviour
  features
    srv: provides subprogram access;
end reactive;
```

FROM COMPONENT-CONNECTOR TO AADL



EXTENSION TO ROS



EXTENSION TO ROS

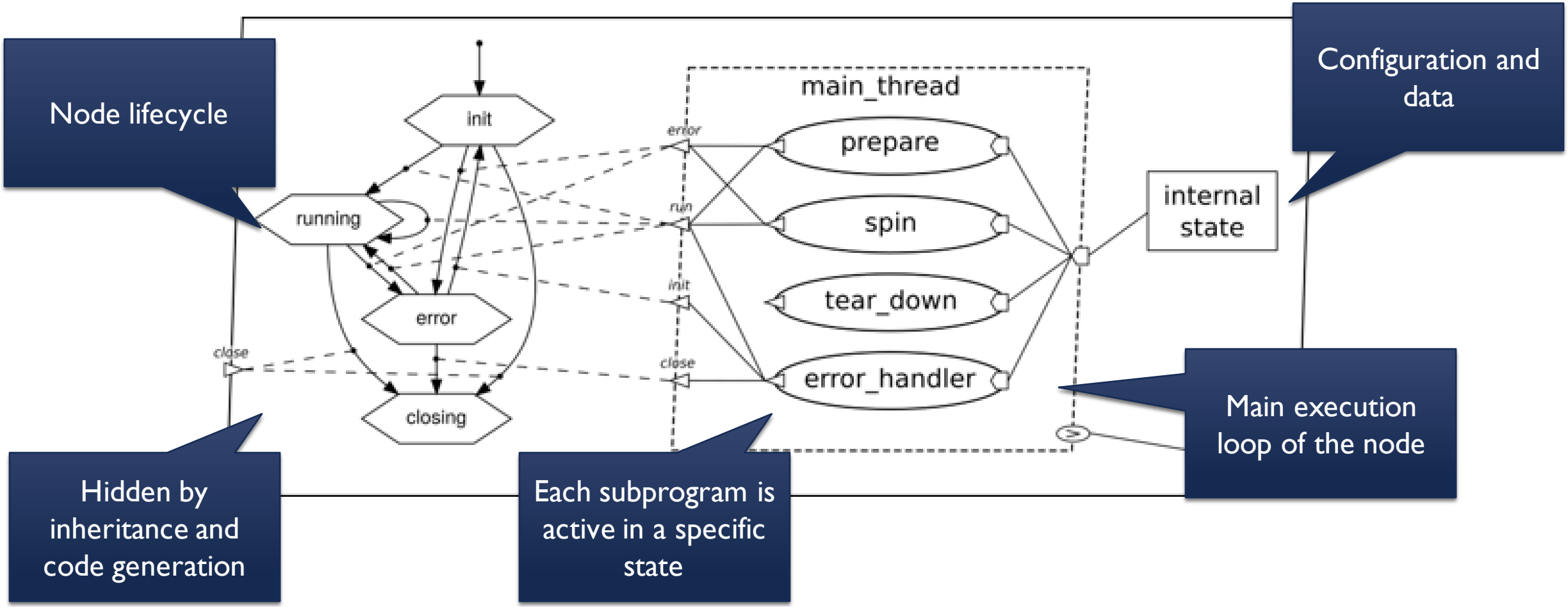

```
thread publisher extends cnc::message_source
  prototypes  message: data;
  features
    msg: refined to out data p
    tf: requires data access t
end publisher;
thread callback extends cnc::m
  prototypes  message: data;
  features
    msg: refined to in event data port message;
    tf: requires data access tf;
end callback;
```

Prototypes are used to accommodate the message type in topics/services

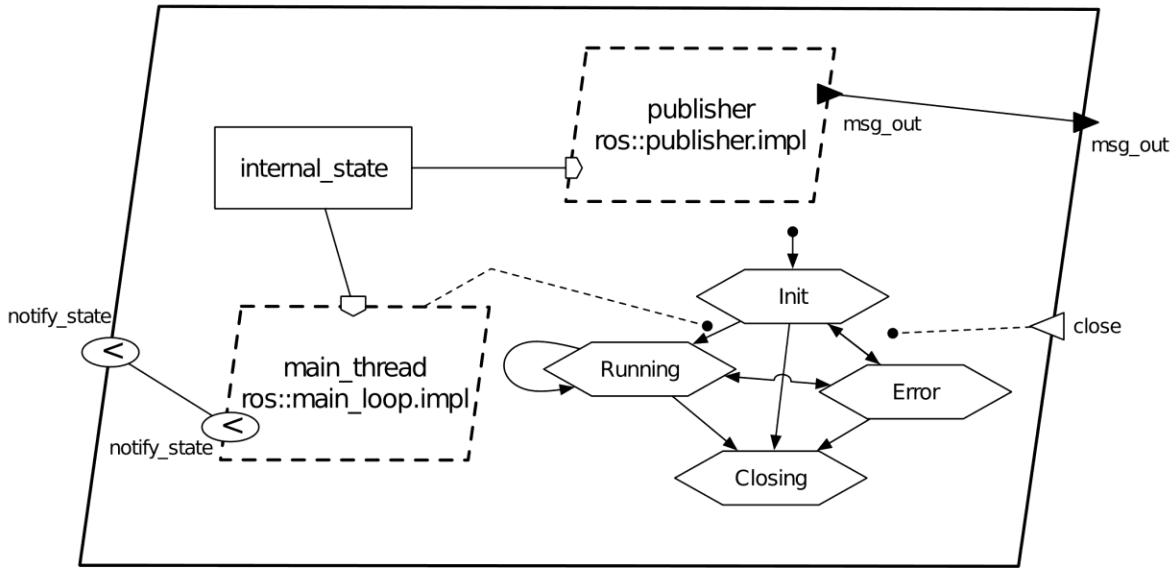
```
thread service_provider extends cnc::reactive
  prototypes  service: subprogram;
  features
                    to provides subprogram access

                    data access tf;
                    der;
                    nd cnc::component_behaviour
  features
    tf: requires data access tf;
  properties
    Dispatch_Protocol => Periodic;
end timer;
```

MINIMAL ROS NODE

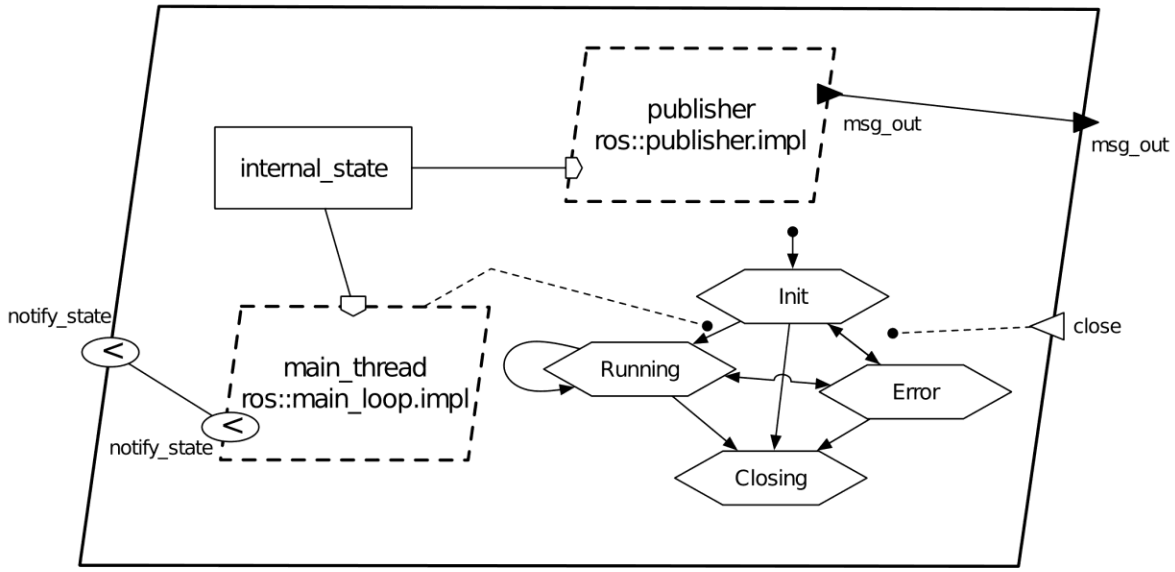


ROS NODES – EXAMPLES




```
process talker extends ros::node
  features
    msg_out : out data port chat_msgs::Chat;
end talker;
```

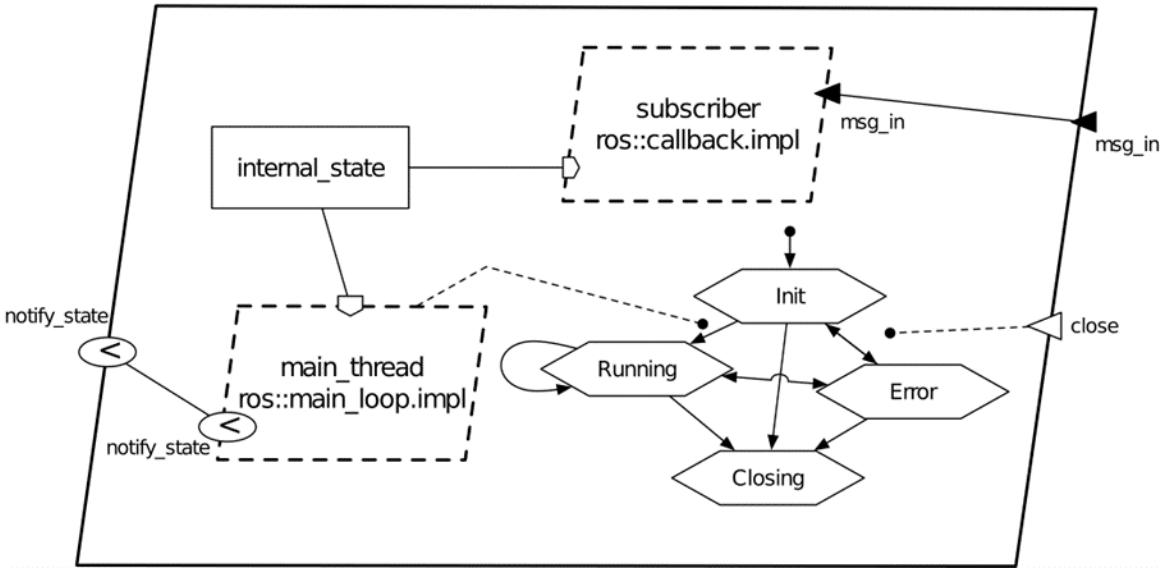
ROS NODES – EXAMPLES




```
process implementation talker.impl extends
ros::node.impl
  subcomponents
    publisher: thread ros::publisher.impl
(message => data chat_msgs::Chat);
  connections
    chatter : port publisher.msg_out -> msg_out;
end talker.impl;
```

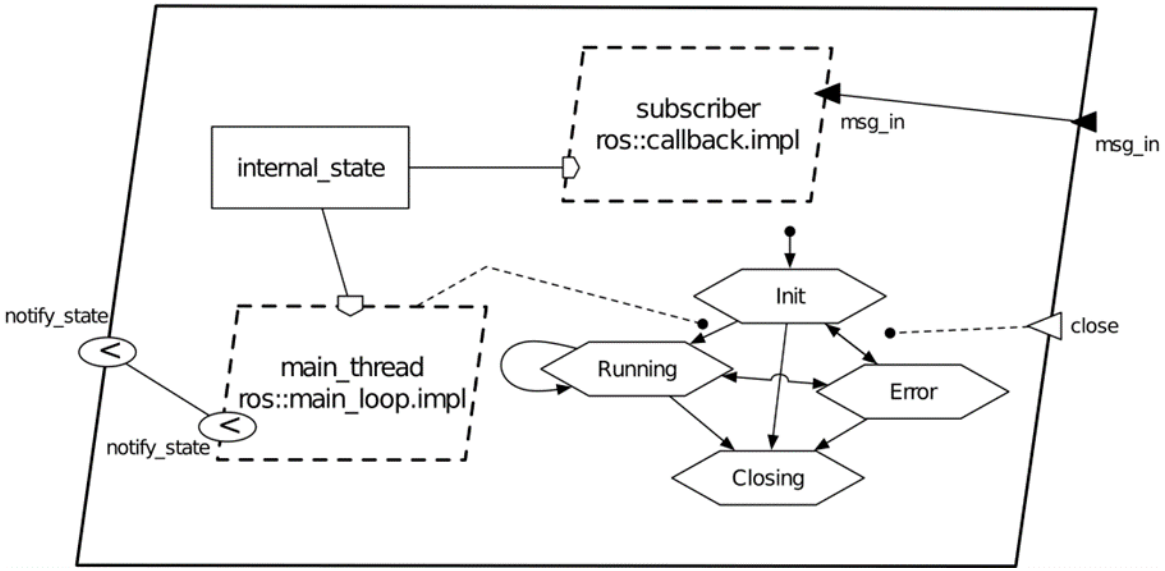
```
properties
   Period => 10 ms applies to publisher;
   topic_properties::Default_Name => "/out_chat" applies to msg_out;
   Source_Text => ("talker.schema.json") applies to internal_state;
   Source_Text => ("talker.h") applies to publisher.function;
   Source_Name => "talk" applies to publisher.function;
```

ROS NODES – EXAMPLES




```
process listener extends ros::node
  features
    msg_in : out data port chat_msgs::Chat;
end talker;
```

ROS NODES – EXAMPLES




```
process implementation listener.impl extends
ros::node.impl
  subcomponents
    subscriber: thread ros::callback.impl
(message => data chat_msgs::Chat);
  connections
    chatter : port msg_ing -> subscriber.msg_out;
end listener.impl;
```

```
properties
   Queue_Size => 1 applies to subscriber.msg;
   topic_properties::Default_Name => "/in_chat" applies to msg_in;
   Source_Text => ("listen.schema.json") applies to internal_state;
   Source_Text => ("listener.h") applies to subscriber.function;
   Source_Name => "listen" applies to subscriber.function;
```

ROS NODES – EXAMPLES


```
system implementation talking.ros
  subcomponents
    talker: process talker.impl;
    listener: process listener.impl;
  connections
    chatter: port talker.msg_out -> listener.msg_in;
  properties
    topic_properties::Name => "/chatter" applies to chatter;
    Source_Text => ("talker.json") applies to talker;
    Source_Text => ("listener.json") applies to listener;
end talking.ros;
```

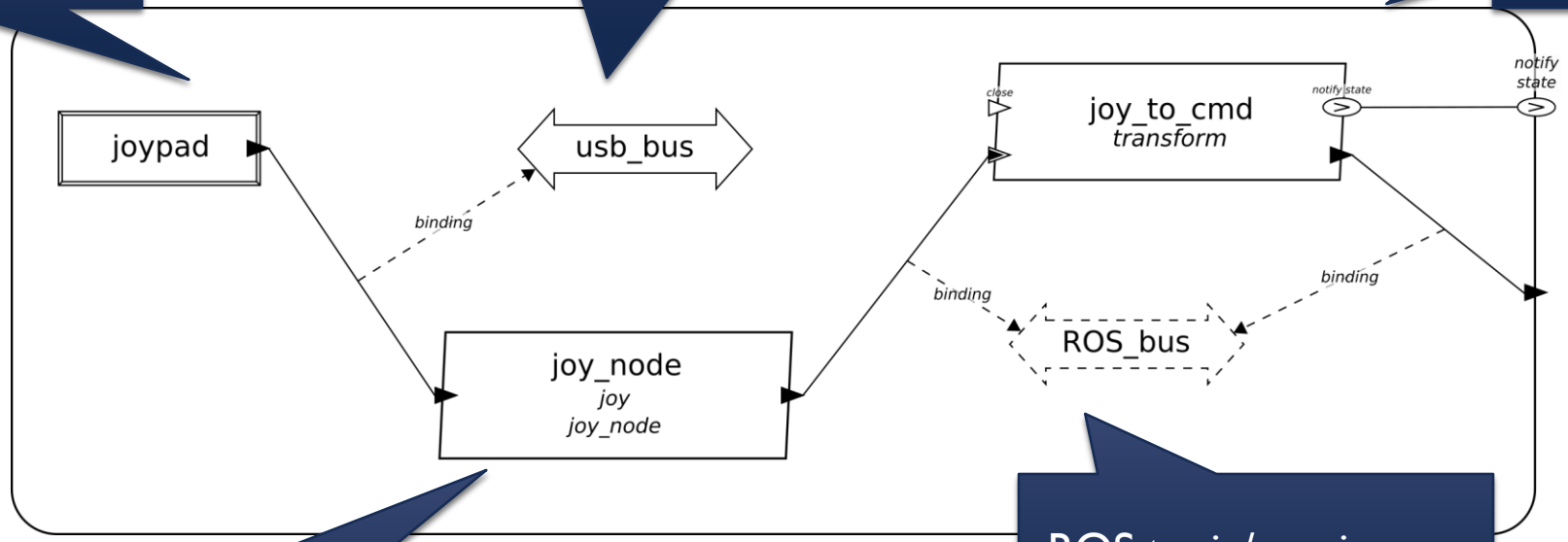
```
data Chat
  properties
    Source_Text => ("Chat.schema.json");
end Chat;
```

OTHER ARCHITECTURAL ELEMENTS

Sensors and actuators are modeled using devices

Physical connection are bound to buses

Systems represent launch files



Existing nodes are modeled using auto-generated interfaces

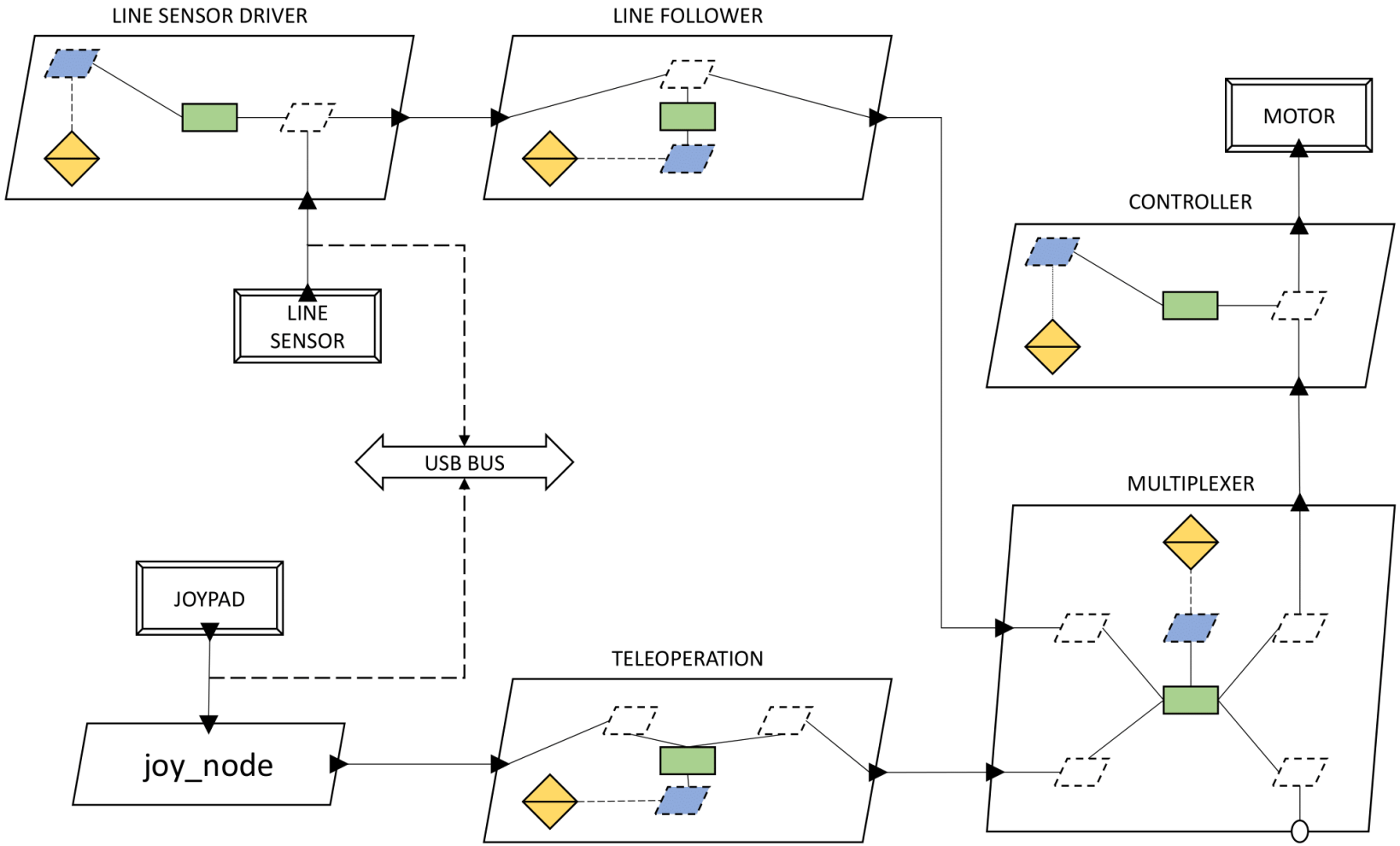
ROS topic/service are bound to virtual buses

EXISTING ROS NODE

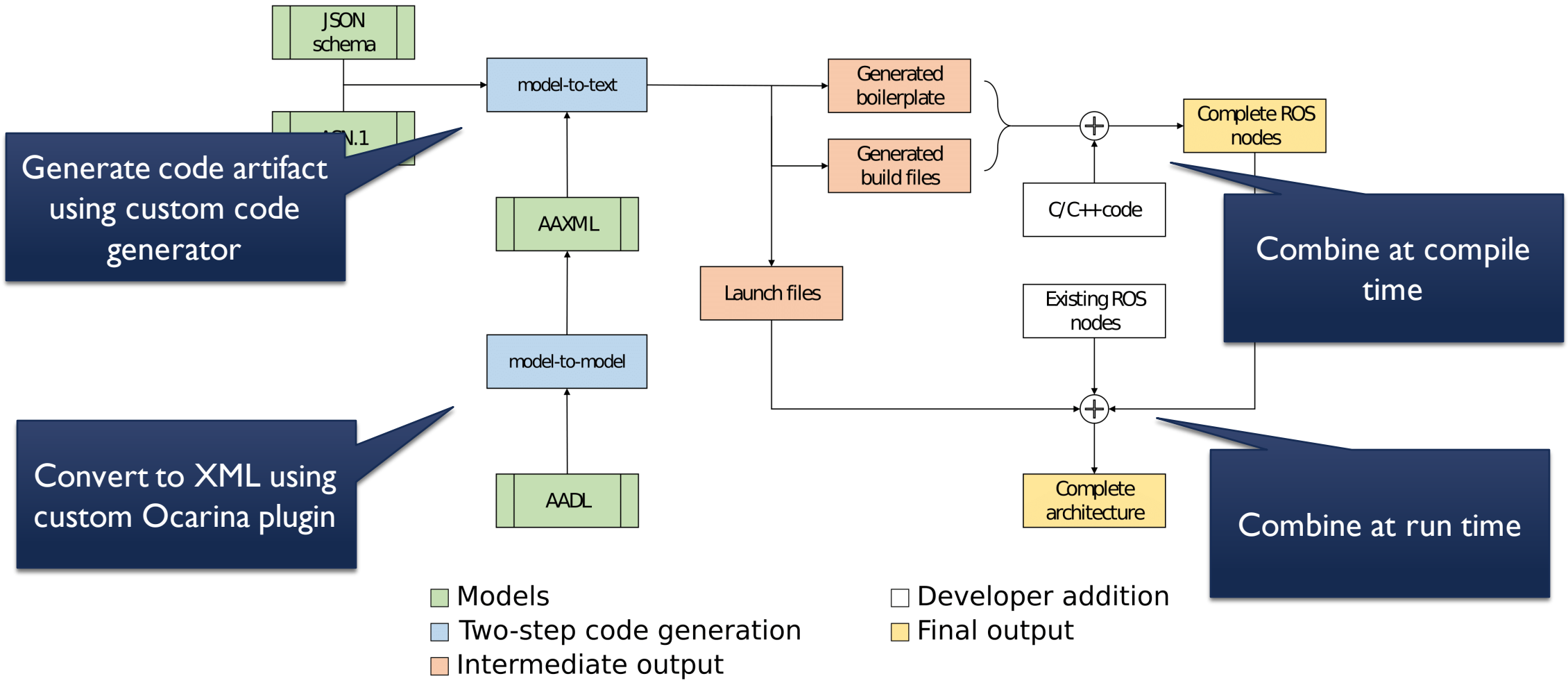

```
process amcl
  features
    amcl_pose: out data port geometry_msgs::PoseWithCovarianceStamped;
    particlecloud: out data port geometry_msgs::PoseArray;
    scan: in event data port sensor_msgs::LaserScan;
    map: in event data port nav_msgs::OccupancyGrid;
    initialpose: in event data port geometry_msgs::PoseWithCovarianceStamped;
  properties
    topic_properties::Default_Name => "/amcl_pose" applies to amcl_pose;
    topic_properties::Default_Name => "/particlecloud" applies to particlecloud;
    topic_properties::Default_Name => "/scan" applies to scan;
    topic_properties::Default_Name => "/map" applies to map;
    topic_properties::Default_Name => "/initialpose" applies to initialpose;
end amcl;
```

```
with topic_properties,
geometry_msgs,
sensor_msgs, nav_msgs;
```

A SIMPLE ROS ARCHITECTURE

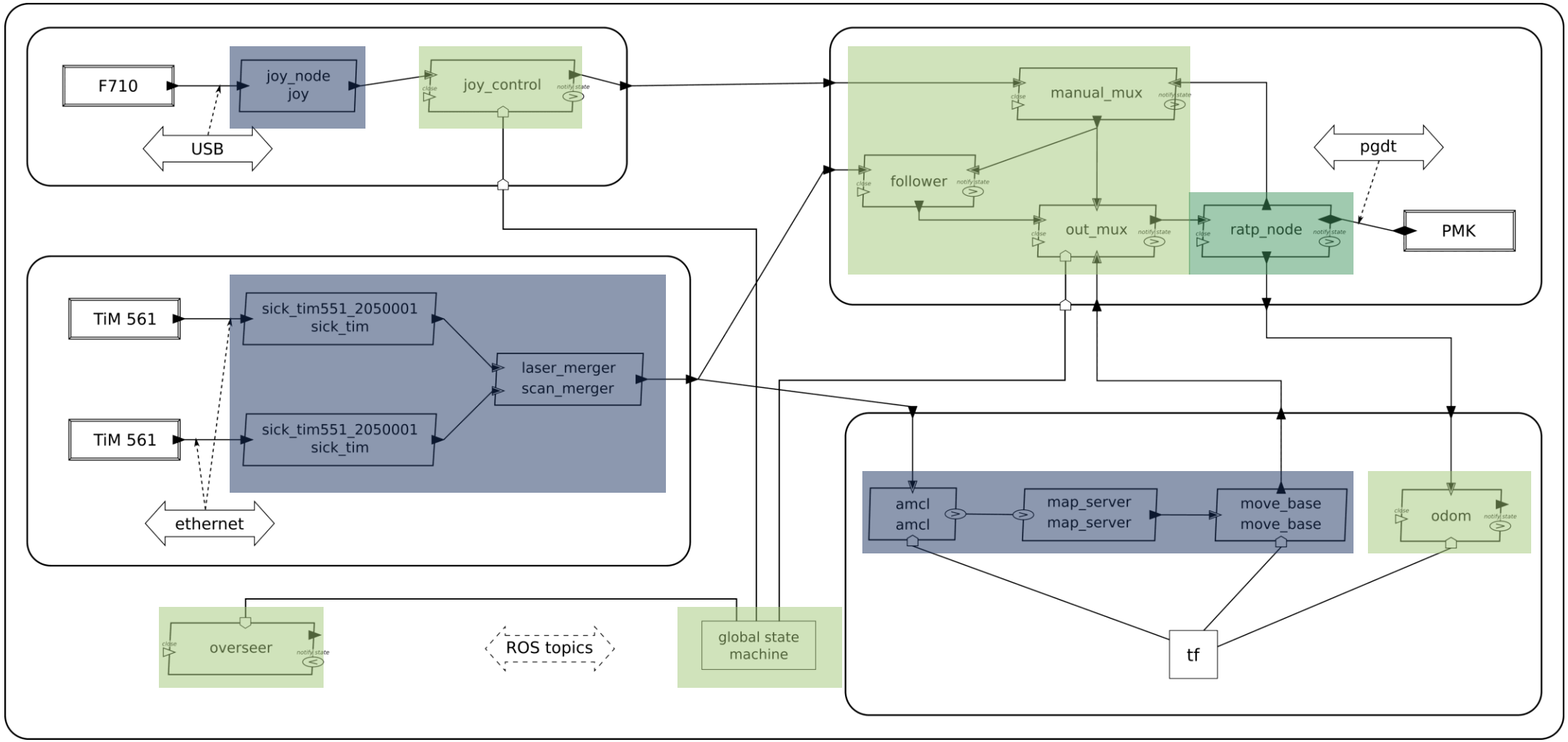


CODE GENERATION

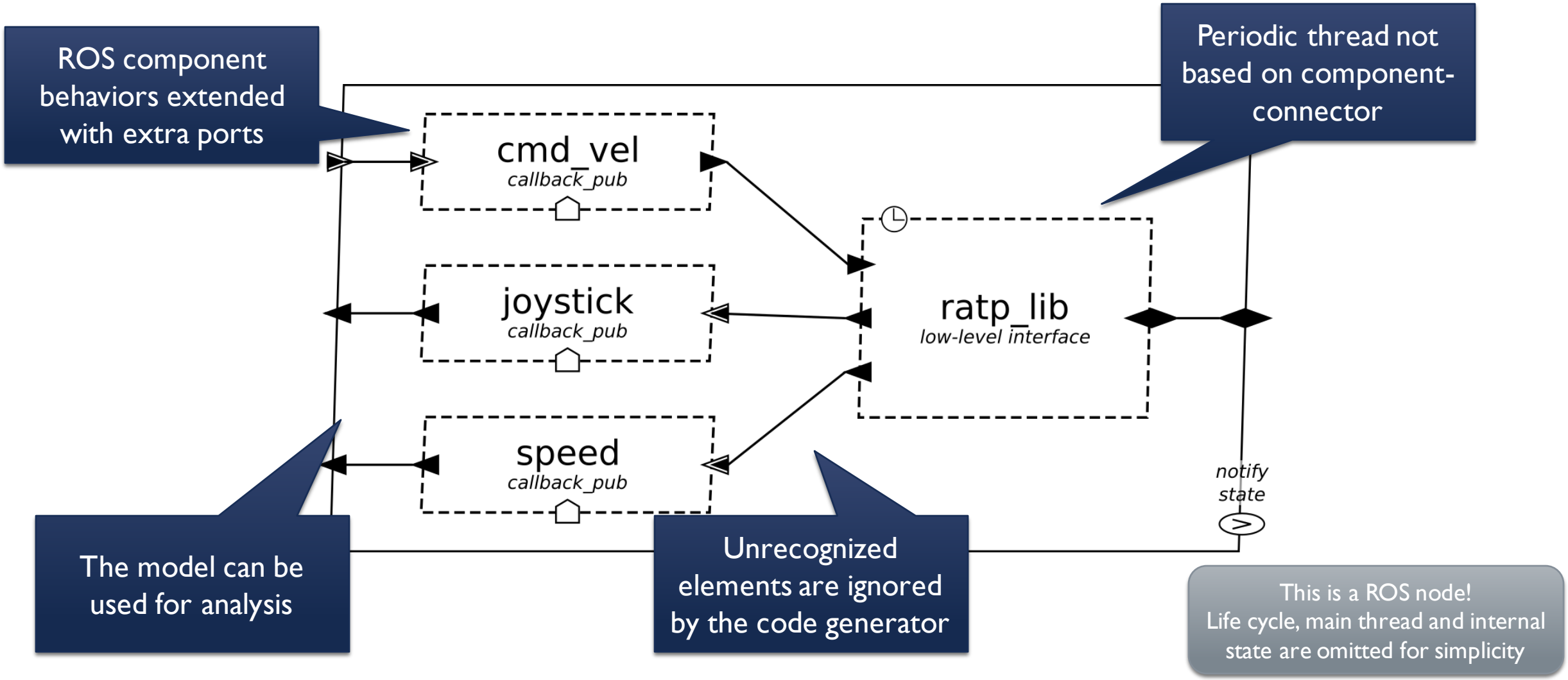


REAL USE CASE: PERSONAL MOBILITY KIT

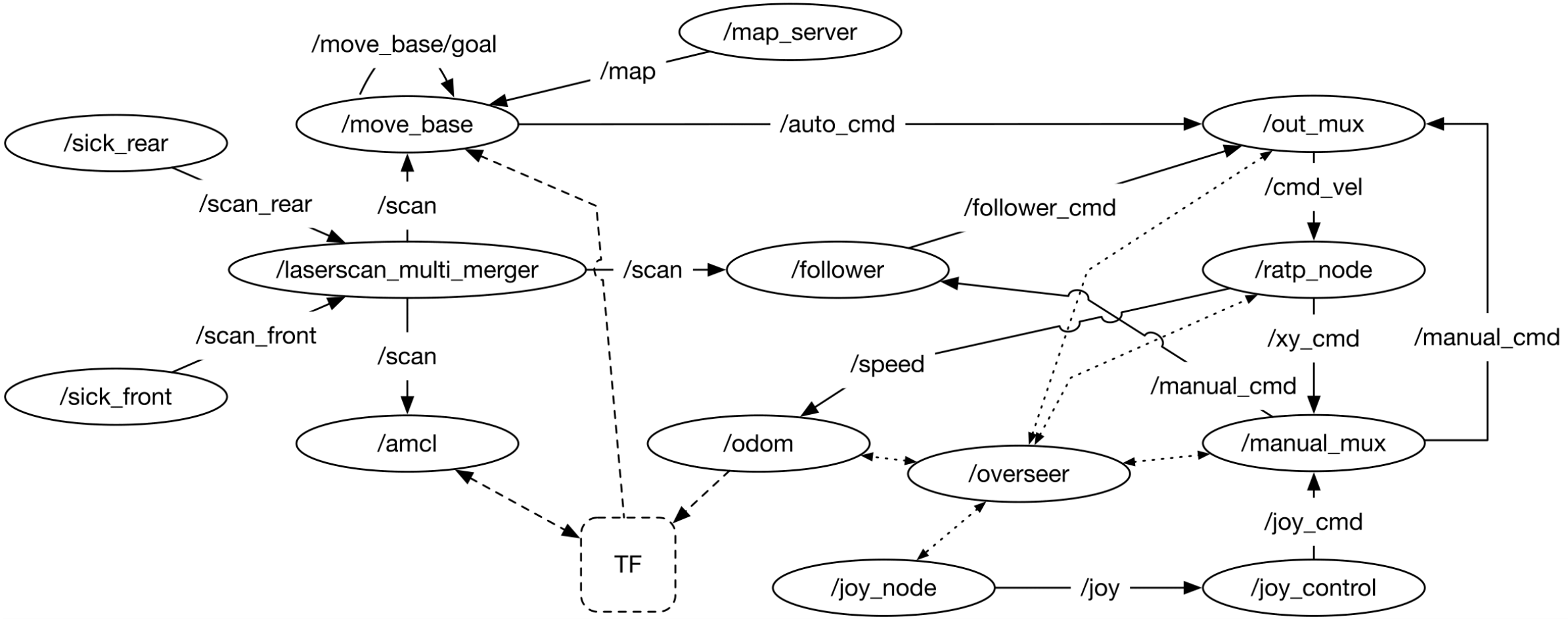
AADL MODEL


F710
joy_node
joy
joy_control
USB
manual_mux
follower
out_mux
ratp_node
pgdt
PMK
TiM 561
sick_tim551_2050001
sick_tim
TiM 561
sick_tim551_2050001
sick_tim
laser_merger
scan_merger
ethernet
amcl
amcl
map_server
map_server
move_base
move_base
odom
tf
overseer
ROS topics
global state
machine

THE RAPT NODE



ROS GRAPH


/move_base/goal
/map_server
/map
/move_base
/auto_cmd
/out_mux
/sick_rear
/scan_rear
/scan
/follower_cmd
/cmd_vel
/laserscan_multi_merger
/scan
/follower
/ratp_node
/scan_front
/scan
/speed
/xy_cmd
/manual_cmd
/sick_front
/manual_cmd
/amcl
/odom
/manual_mux
/overseer
/joy_cmd
TF
/joy_node
/joy
/joy_control

CONCLUSIONS

Thank you for you attention – more details here:

Bardaro G., Semprebon A., Chiatti A., and Matteucci M.

From models to software through automatic transformations:An AADL to ROS end-to-end toolchain.

3rd IEEE International Conference on Robotic Computing (IRC). 2019.

Bardaro G., Semprebon A., and Matteucci M.

A use case in model-based robot development using AADL and ROS.

1st International Workshop on Robotics Software Engineering. 2018.

Bardaro G., and Matteucci M.

Using AADL to model and develop ROS-based robotic application.

1st IEEE International Conference on Robotic Computing (IRC). IEEE, 2017.