The Usability Argument for ROS-based Robot Architectural Description Languages

Paulo Canelas

with Bradley Schmerl, Alcides Fonseca, and Christopher S. Timperley

Carnegie Mellon University

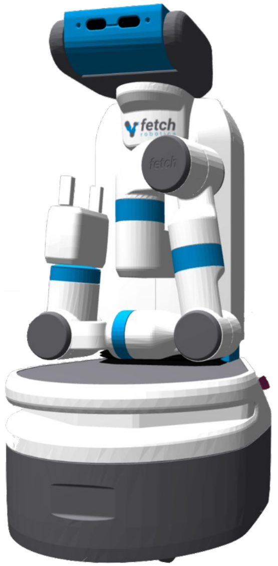
University of Lisbon

15th Annual Workshop on the intersection of HCI and PL (PLATEAU). 2025.

The Robot Operating System (ROS) allows developers to quickly compose and integrate components

> “We have designed ROS to support our **philosophy of modular**, tools-based software development”
> [Quigley et al, 2009]

Libraries and tools available for composing



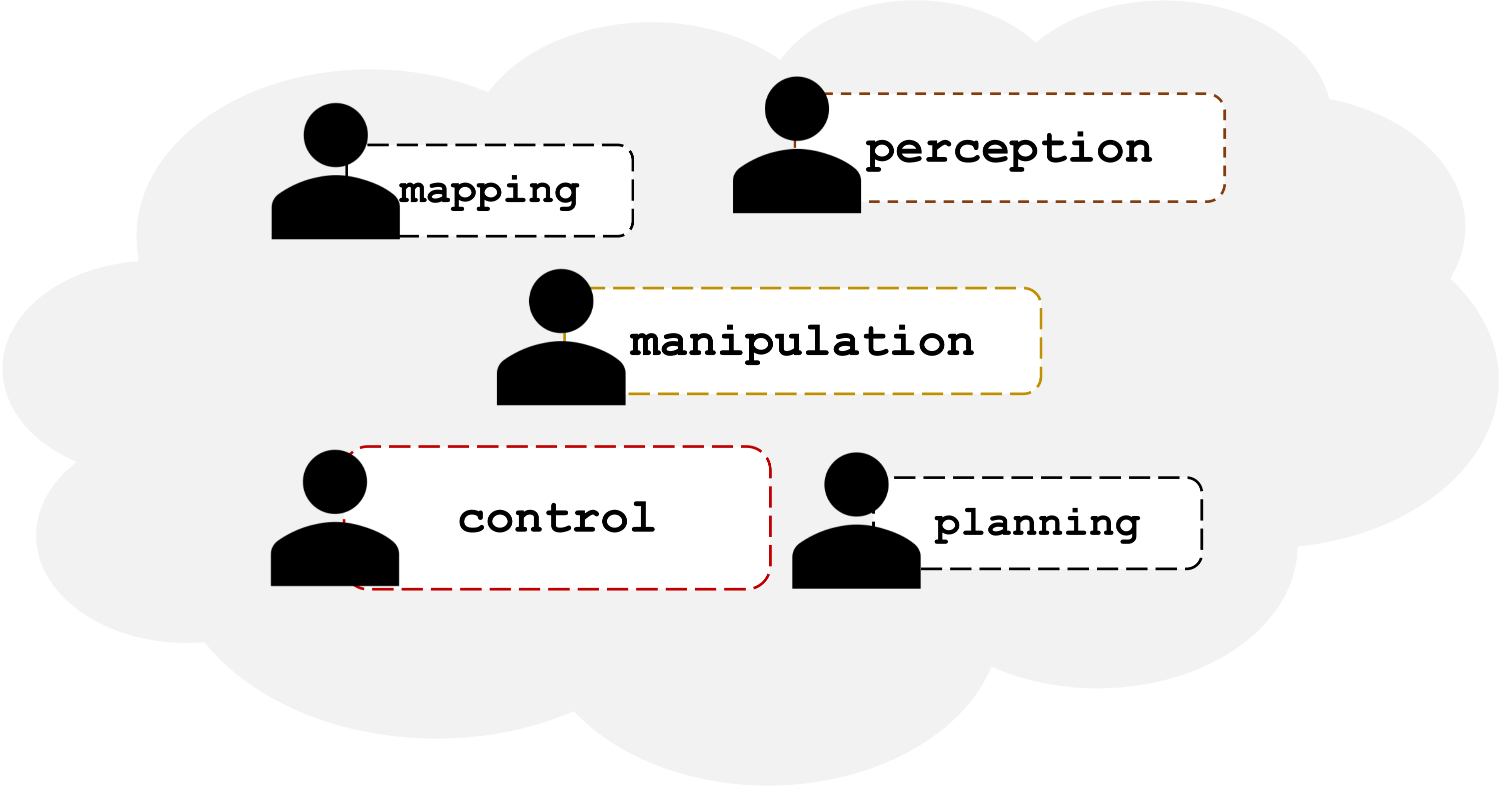
Quickly prototype the robotic system

BOSCH

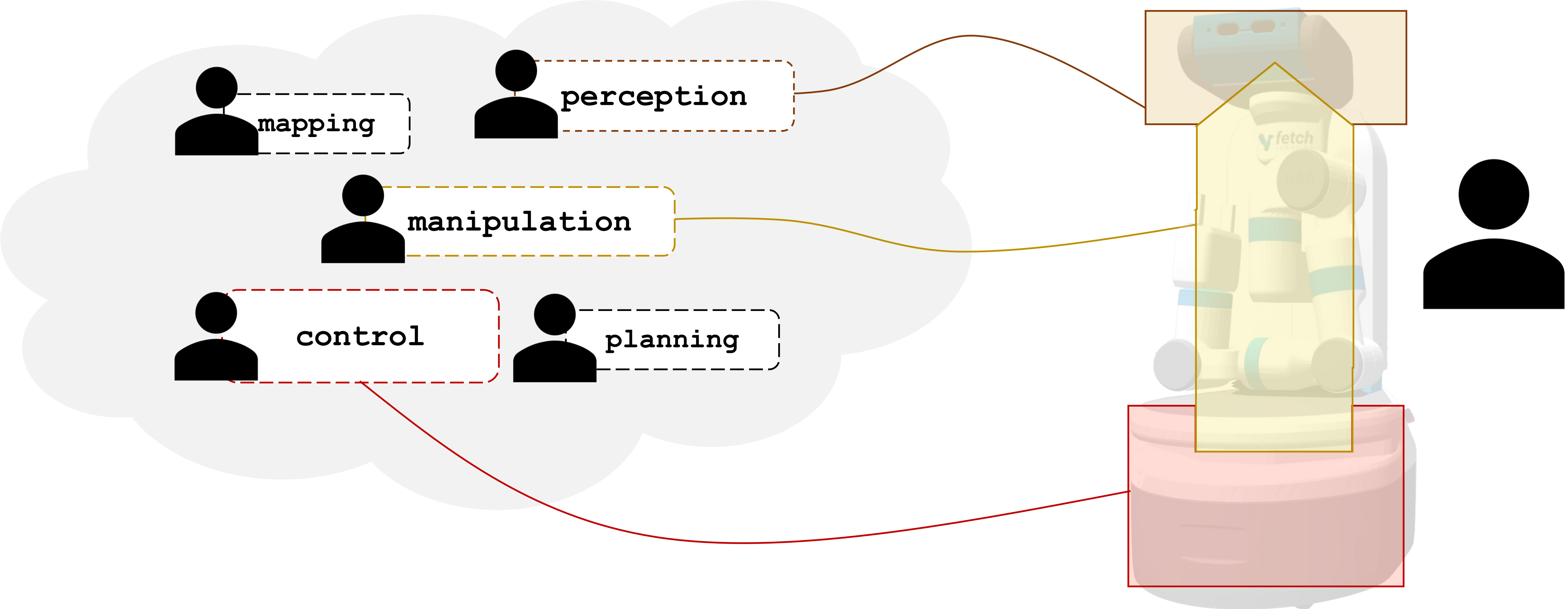
BOEING

Popular adoption in the industry

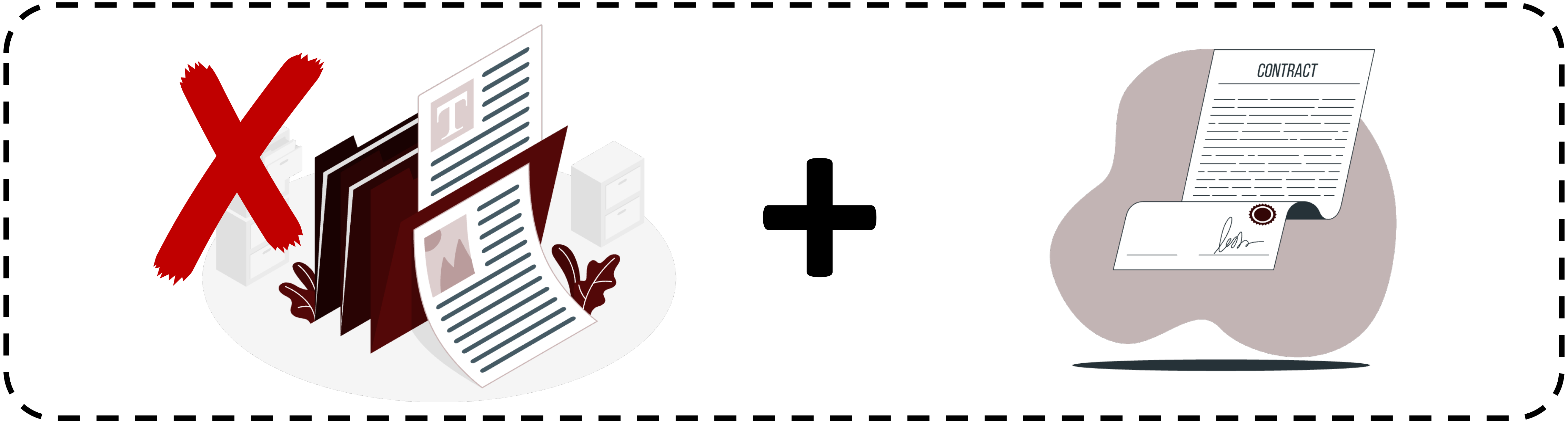
Component Writers create reusable components



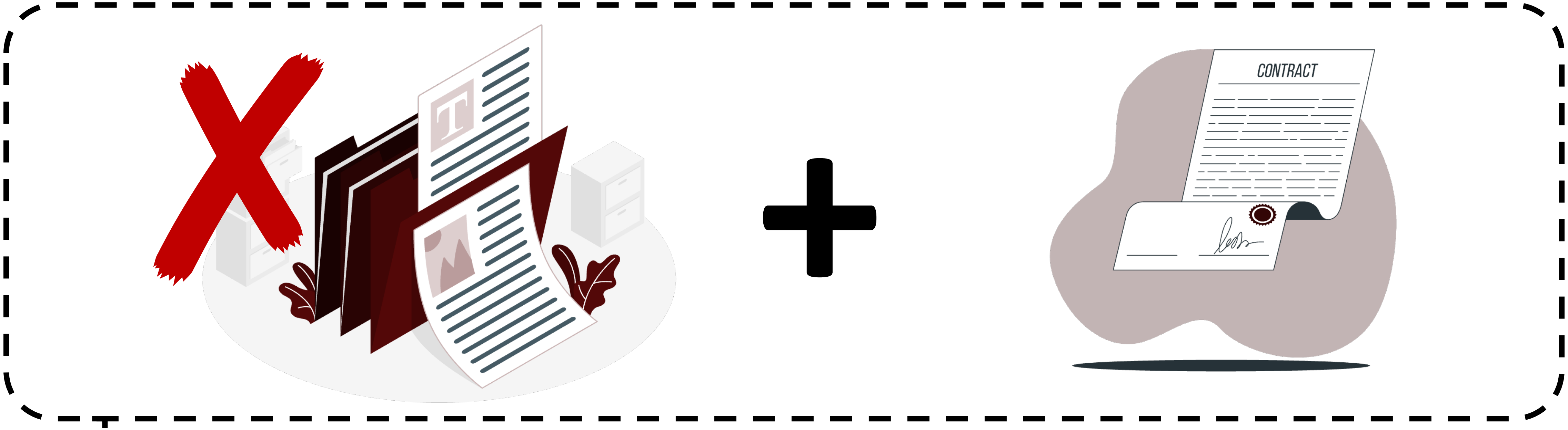
Component Writers create reusable components; System Integrators compose them into a system

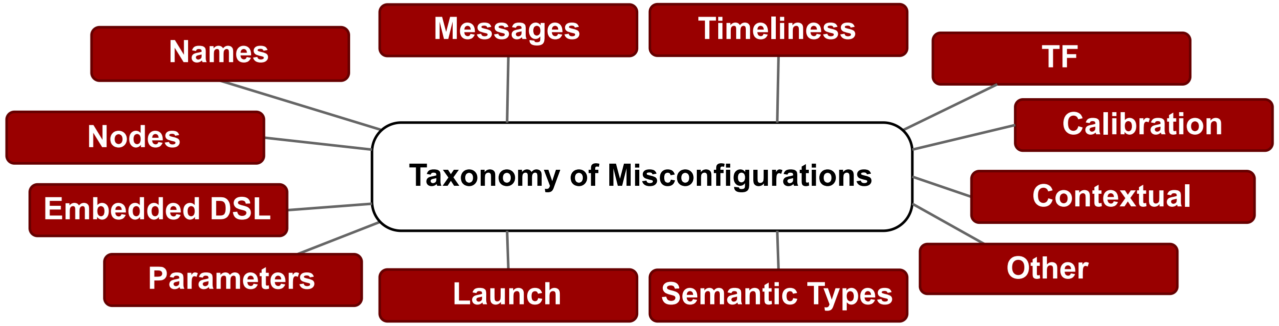


However, when trying to integrate components, misconfigurations arise due to lack of documentation or since it is not enforced



However, when trying to integrate components, misconfigurations arise due to lack of documentation or since it is not enforced



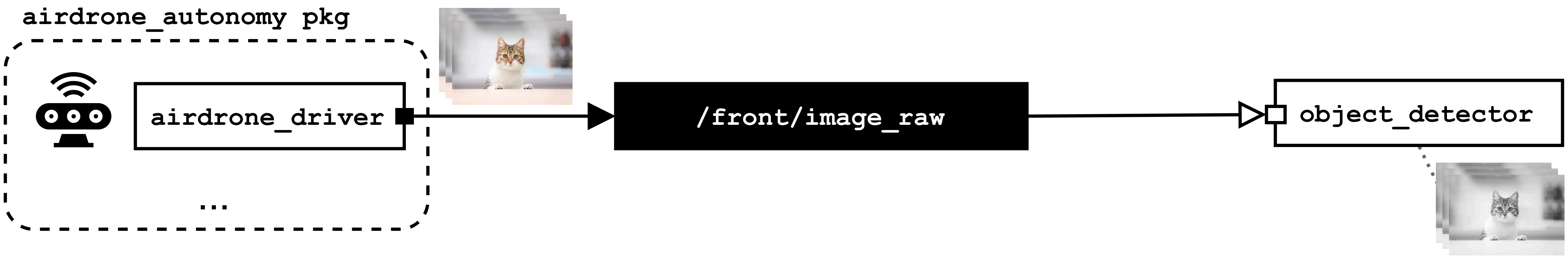


Taxonomy of Misconfigurations

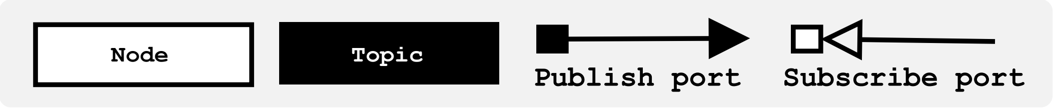
- Names
- Nodes
- Embedded DSL
- Parameters
- Messages
- Launch
- Timeliness
- Semantic Types
- TF
- Calibration
- Contextual
- Other

[Canelas et al, 2024]

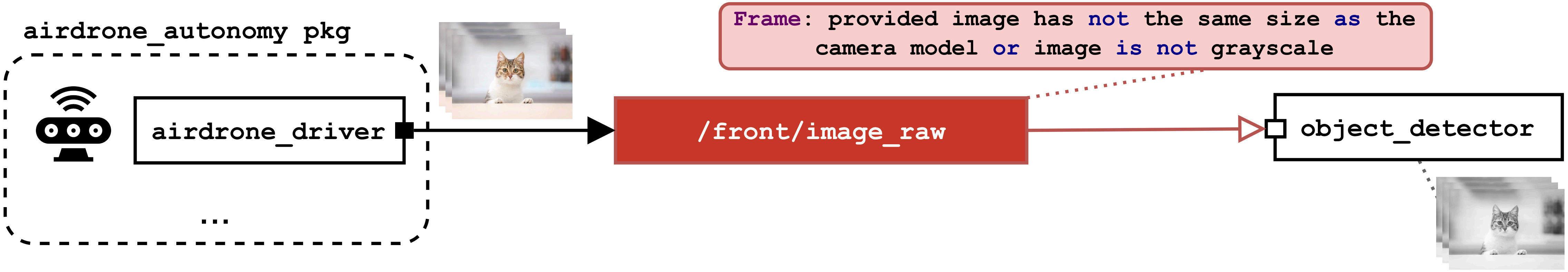
The airdrone_driver sends sensor data, while the object_detector receives and processes the information



Legend



Missing semantic information regarding image format leads to mismatches image color expectations



airdrone_autonomy pkg

airdrone_driver

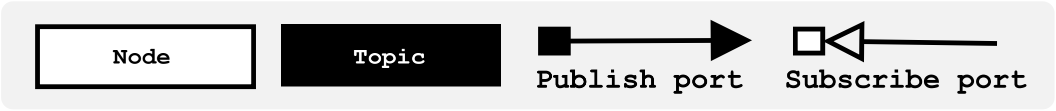
...

/front/image_raw

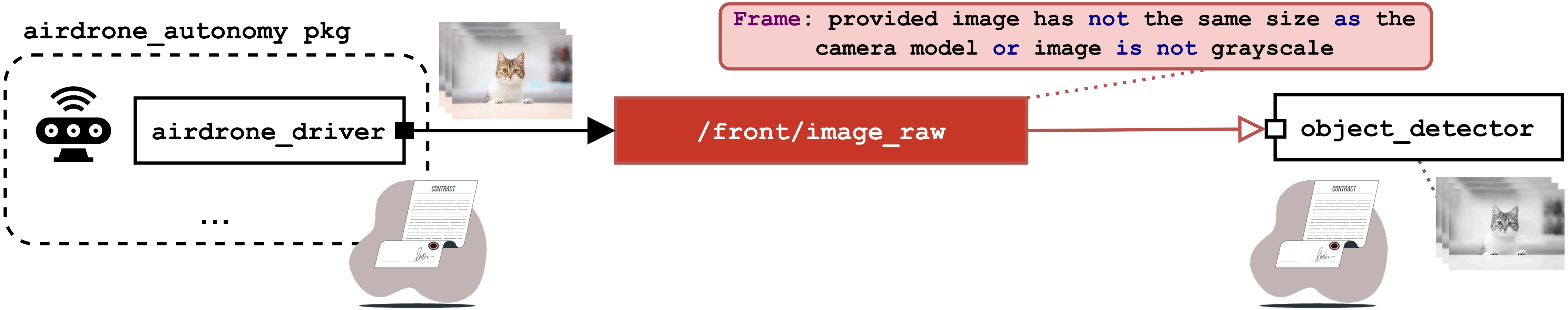
Frame: provided image has not the same size as the camera model or image is not grayscale

object_detector

Legend

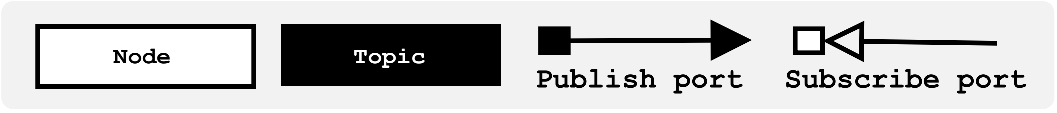


Missing semantic information regarding image format leads to mismatches image color expectations

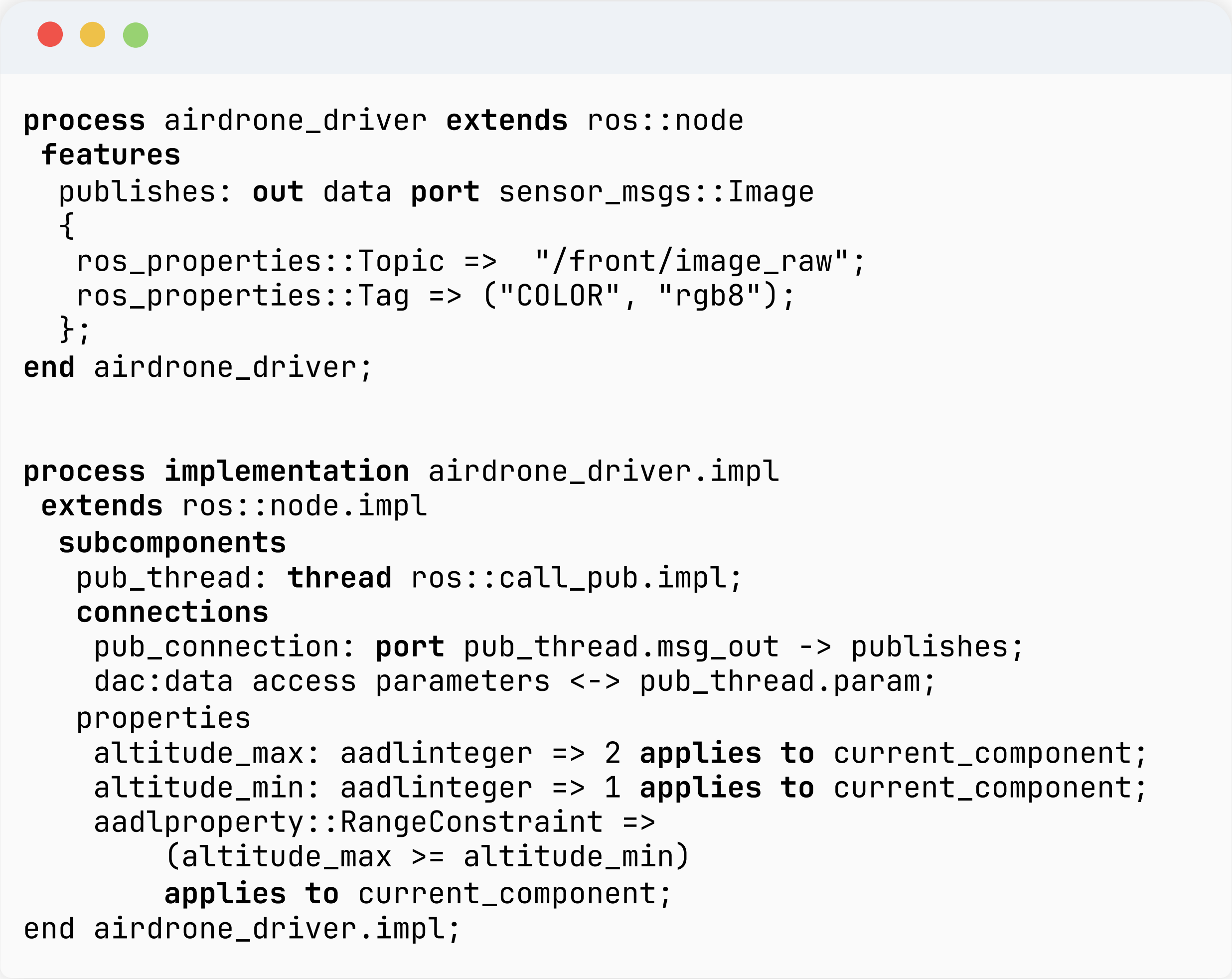


Frame: provided image has not the same size as the camera model or image is not grayscale

Legend

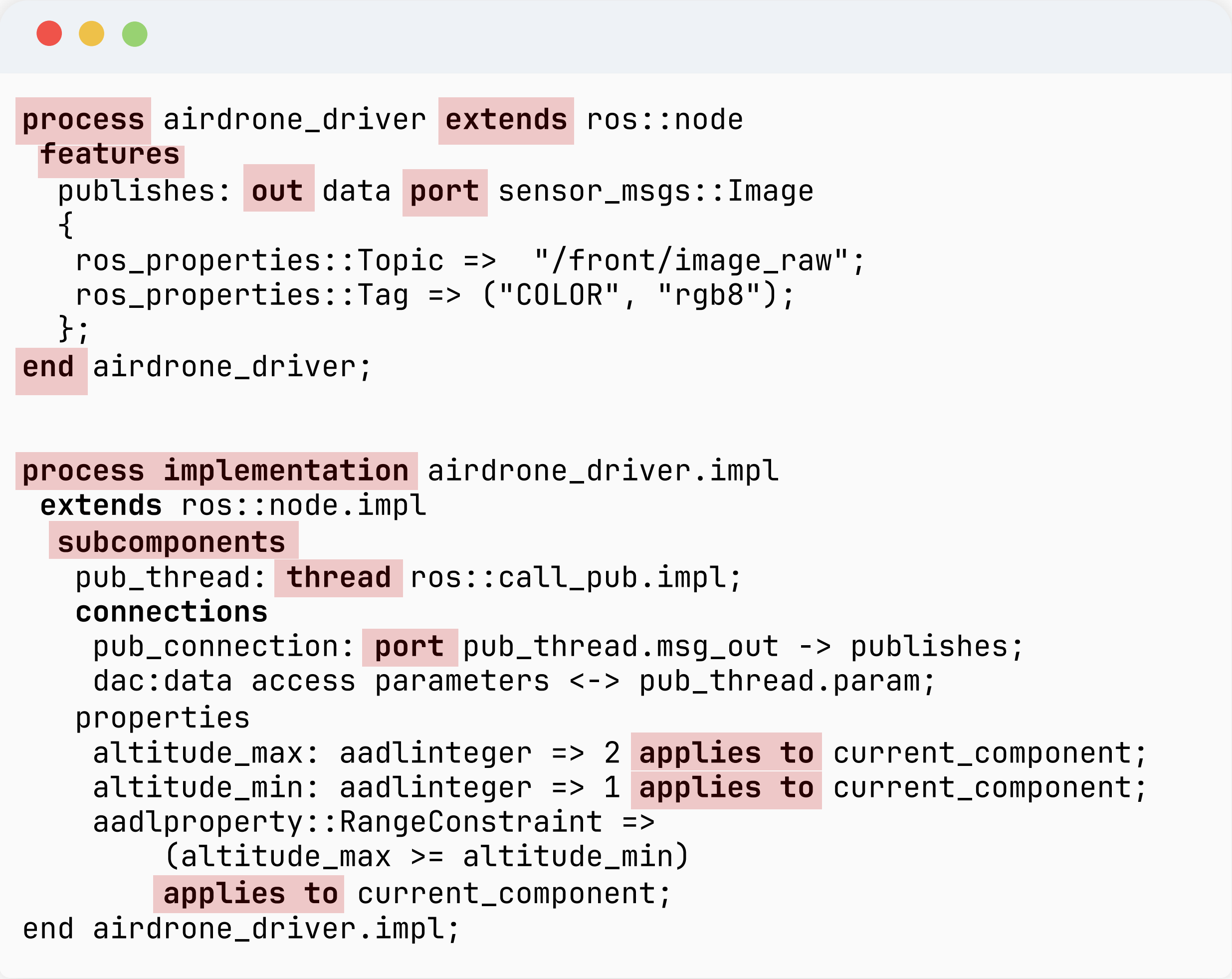


General Purpose Architectural Description Languages have been used specify components



```
process airdrone_driver extends ros::node
 features
  publishes: out data port sensor_msgs::Image
  {
   ros_properties::Topic =>  "/front/image_raw";
   ros_properties::Tag => ("COLOR", "rgb8");
  };
end airdrone_driver;


process implementation airdrone_driver.impl
 extends ros::node.impl
  subcomponents
   pub_thread: thread ros::call_pub.impl;
   connections
    pub_connection: port pub_thread.msg_out -> publishes;
    dac:data access parameters <-> pub_thread.param;
   properties
    altitude_max: aadlinteger => 2 applies to current_component;
    altitude_min: aadlinteger => 1 applies to current_component;
    aadlproperty::RangeConstraint =>
        (altitude_max >= altitude_min)
        applies to current_component;
end airdrone_driver.impl;
```

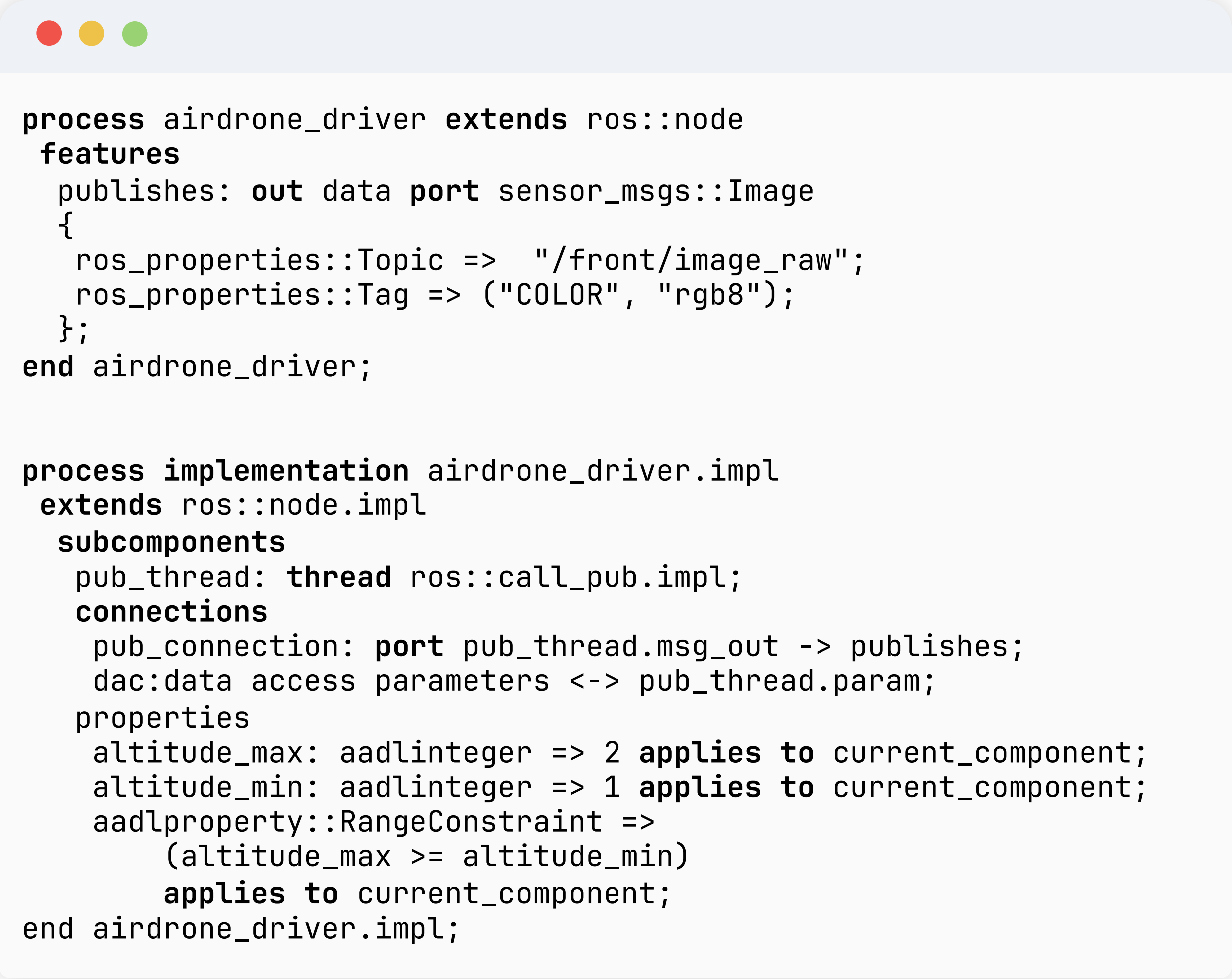
However, these require learning non-ROS concepts



```
process airdrone_driver extends ros::node
 features
  publishes: out data port sensor_msgs::Image
  {
   ros_properties::Topic =>  "/front/image_raw";
   ros_properties::Tag => ("COLOR", "rgb8");
  };
end airdrone_driver;


process implementation airdrone_driver.impl
 extends ros::node.impl
  subcomponents
   pub_thread: thread ros::call_pub.impl;
   connections
    pub_connection: port pub_thread.msg_out -> publishes;
    dac:data access parameters <-> pub_thread.param;
   properties
    altitude_max: aadlinteger => 2 applies to current_component;
    altitude_min: aadlinteger => 1 applies to current_component;
    aadlproperty::RangeConstraint =>
        (altitude_max >= altitude_min)
        applies to current_component;
end airdrone_driver.impl;
```

However, these require learning non-ROS concepts, do not distinguish between different stakeholders

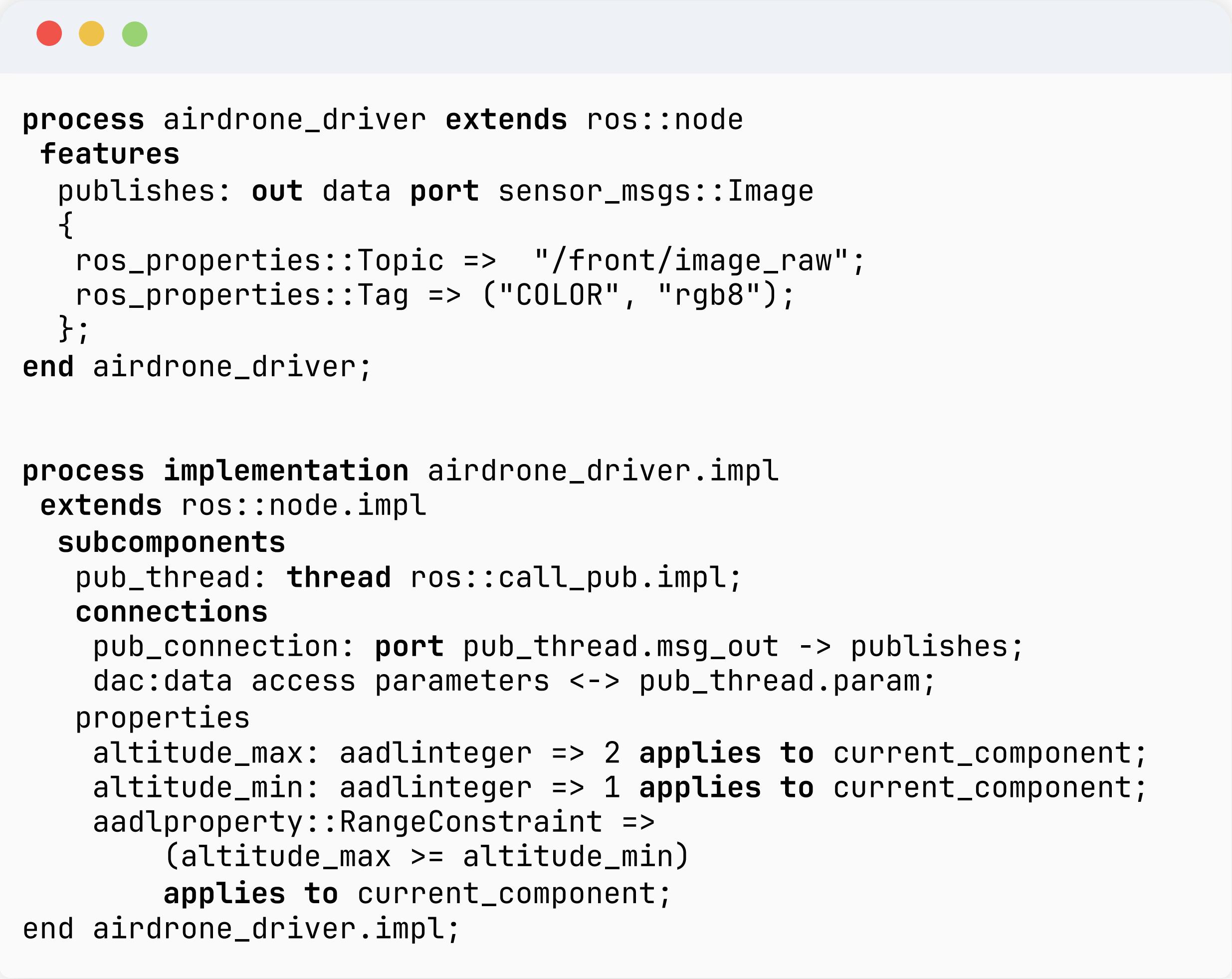
Component writer?



```
process airdrone_driver extends ros::node
 features
  publishes: out data port sensor_msgs::Image
  {
   ros_properties::Topic =>  "/front/image_raw";
   ros_properties::Tag => ("COLOR", "rgb8");
  };
end airdrone_driver;


process implementation airdrone_driver.impl
 extends ros::node.impl
  subcomponents
   pub_thread: thread ros::call_pub.impl;
   connections
    pub_connection: port pub_thread.msg_out -> publishes;
    dac:data access parameters <-> pub_thread.param;
   properties
    altitude_max: aadlinteger => 2 applies to current_component;
    altitude_min: aadlinteger => 1 applies to current_component;
    aadlproperty::RangeConstraint =>
        (altitude_max >= altitude_min)
        applies to current_component;
end airdrone_driver.impl;
```

System Integrator?

However, these require learning non-ROS concepts, do not distinguish between different stakeholders, and are verbose, making it more challenging to write



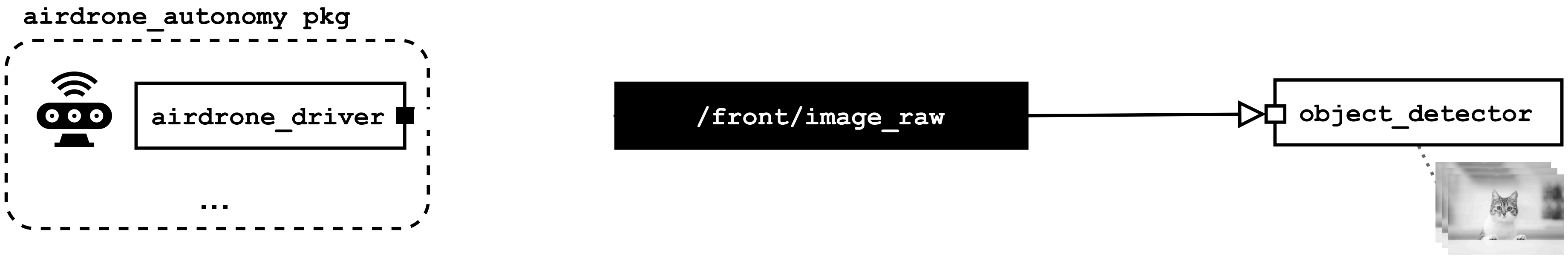
```
process airdrone_driver extends ros::node
 features
  publishes: out data port sensor_msgs::Image
  {
   ros_properties::Topic =>  "/front/image_raw";
   ros_properties::Tag => ("COLOR", "rgb8");
  };
end airdrone_driver;


process implementation airdrone_driver.impl
 extends ros::node.impl
  subcomponents
   pub_thread: thread ros::call_pub.impl;
   connections
    pub_connection: port pub_thread.msg_out -> publishes;
    dac:data access parameters <-> pub_thread.param;
   properties
    altitude_max: aadlinteger => 2 applies to current_component;
    altitude_min: aadlinteger => 1 applies to current_component;
    aadlproperty::RangeConstraint =>
        (altitude_max >= altitude_min)
        applies to current_component;
end airdrone_driver.impl;
```

By developing ROS-tailored Architectural Description Languages (ADL), we can improve documentation and detect misconfigurations prior to execution

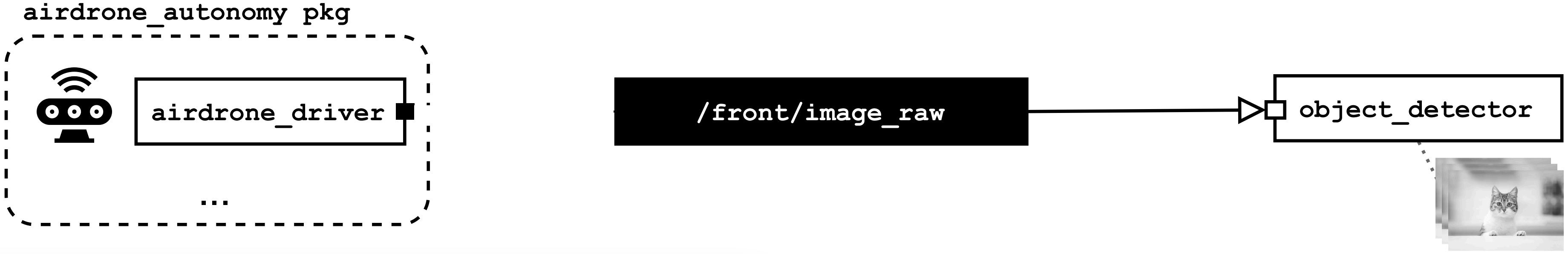
We introduce rospec, a architectural description language tailored to ROS to detect misconfigurations

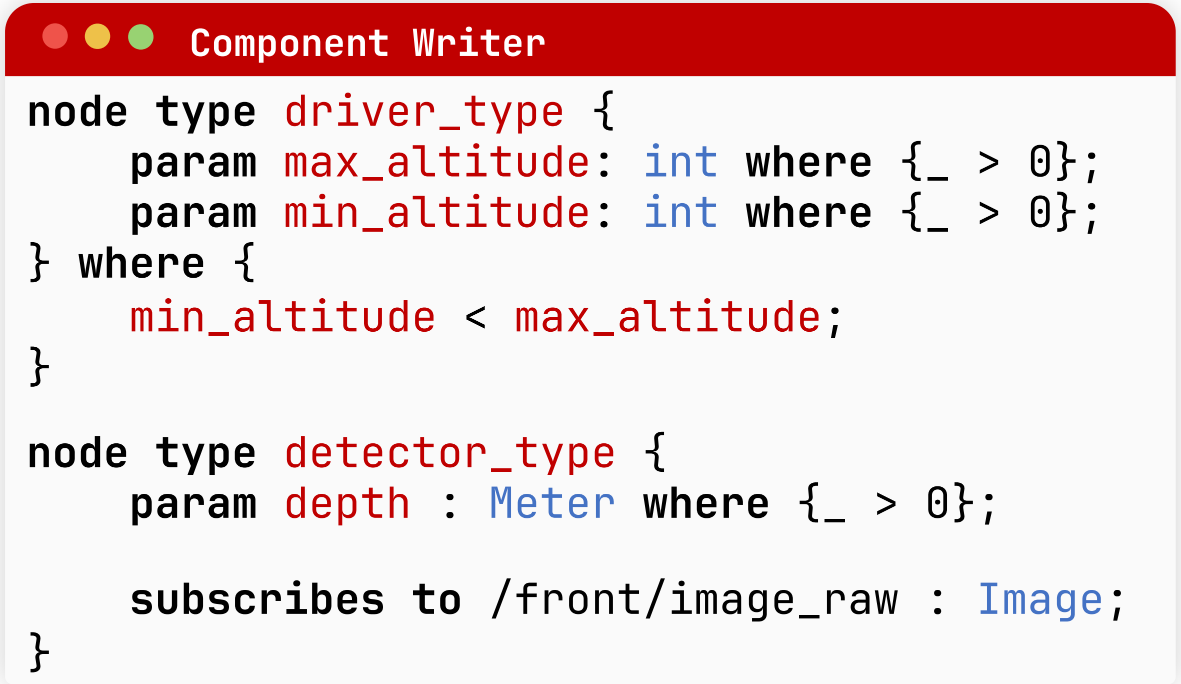
- Language specialization to stakeholders to provide separation of concerns and improve readability
- Embedding domain knowledge into language semantics improves its usability
- Uses prior knowledge in misconfigurations to enrich the type system and detect misconfigurations

Property: A System Integrator should be able to detect missing publishers to topics containing a subscriber relationship



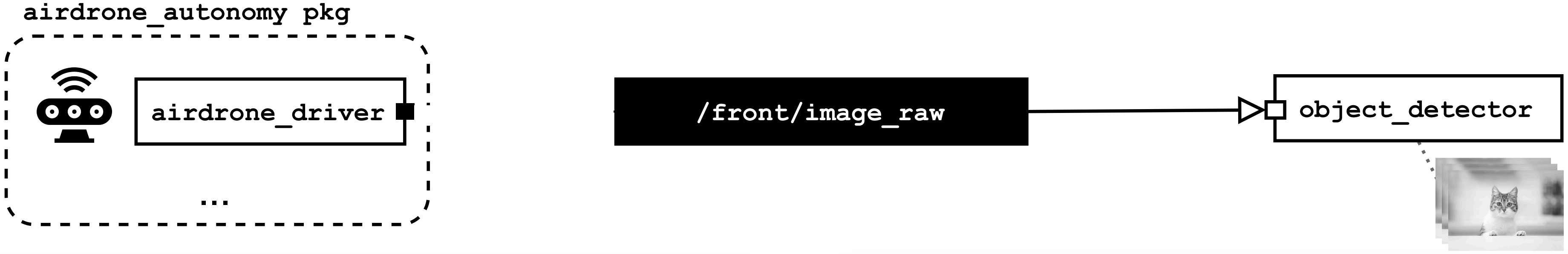
Property: A System Integrator should be able to detect missing publishers to topics containing a subscriber relationship

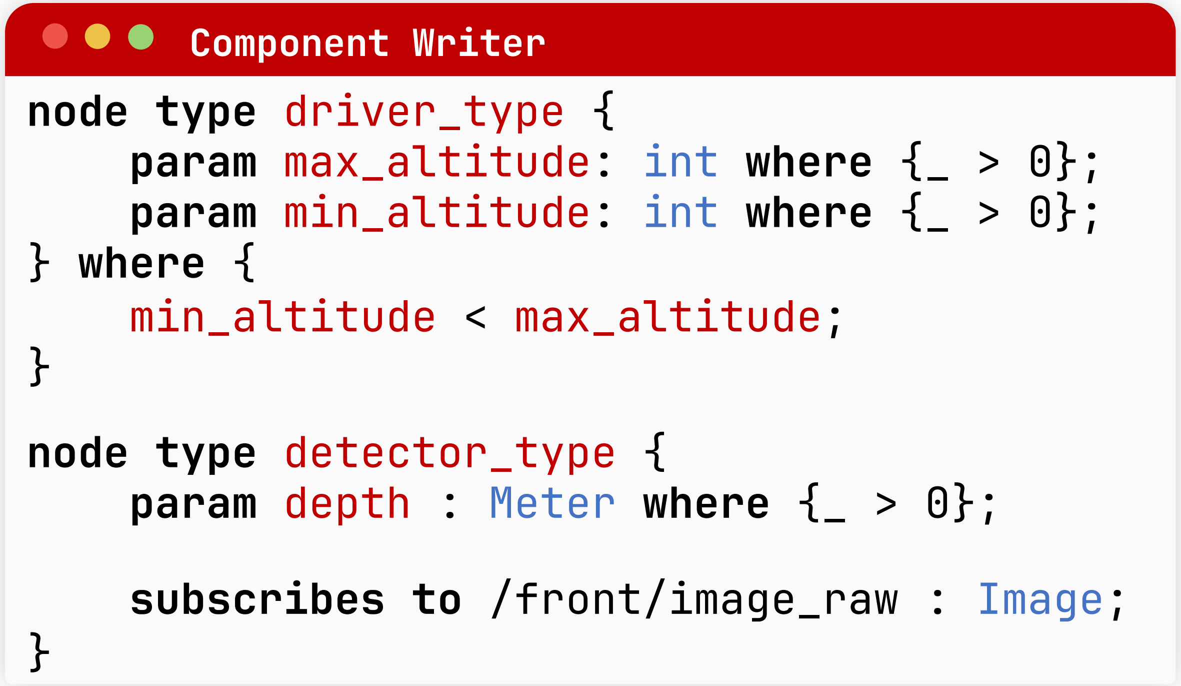




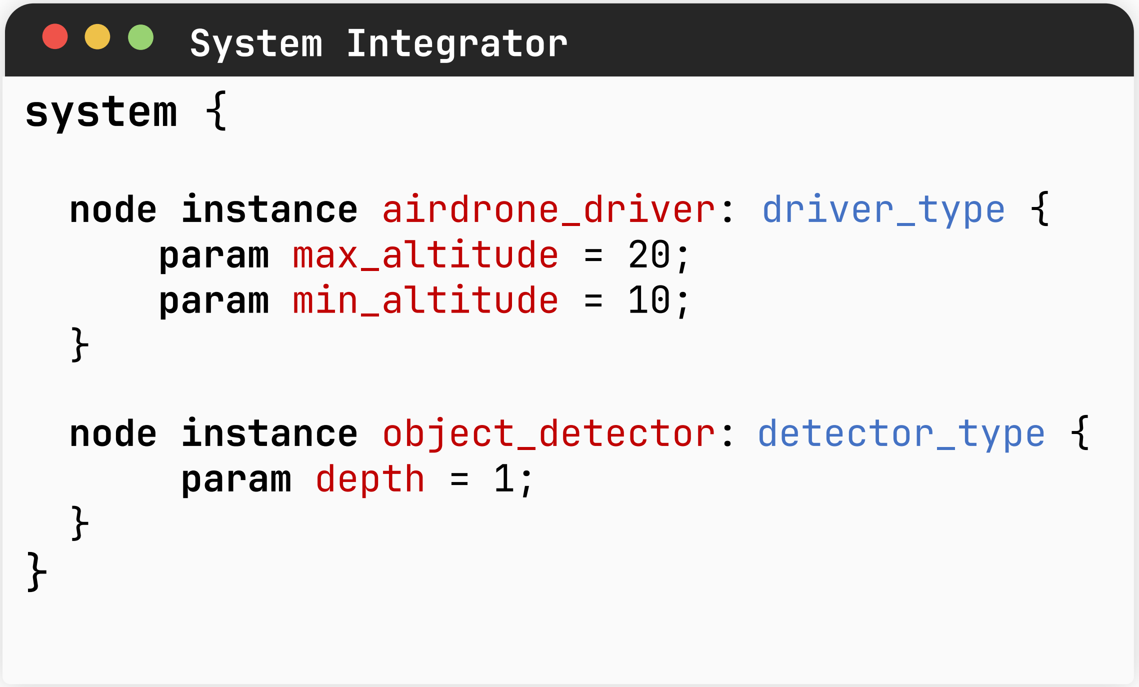
```
node type driver_type {
    param max_altitude: int where {_ > 0};
    param min_altitude: int where {_ > 0};
} where {
    min_altitude < max_altitude;
}

node type detector_type {
    param depth : Meter where {_ > 0};

    subscribes to /front/image_raw : Image;
}
```

Property: A System Integrator should be able to detect missing publishers to topics containing a subscriber relationship



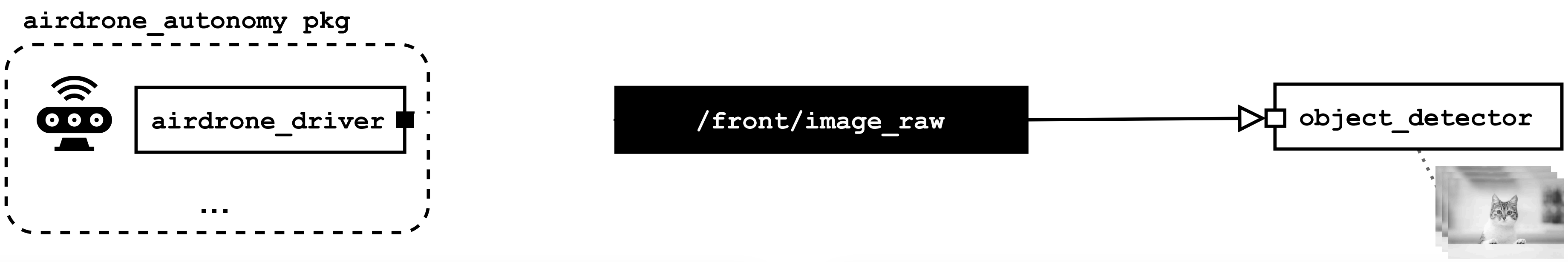


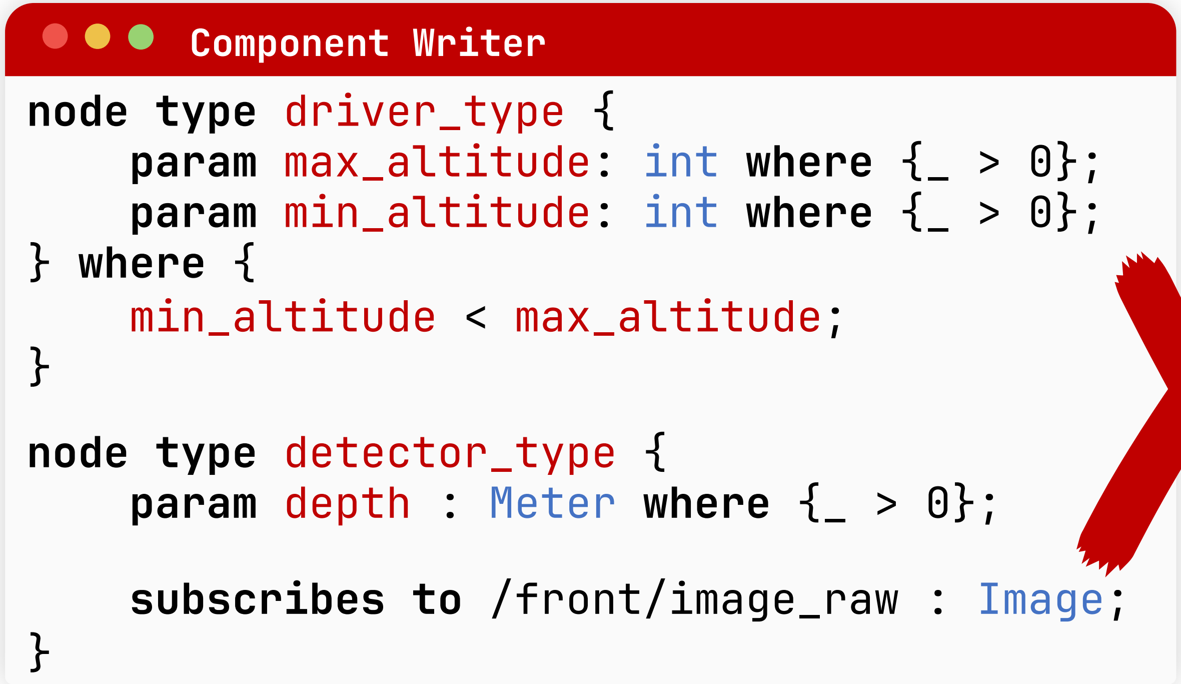
```
node type driver_type {
    param max_altitude: int where {_ > 0};
    param min_altitude: int where {_ > 0};
} where {
    min_altitude < max_altitude;
}

node type detector_type {
    param depth : Meter where {_ > 0};

    subscribes to /front/image_raw : Image;
}
```



```
system {

  node instance airdrone_driver: driver_type {
      param max_altitude = 20;
      param min_altitude = 10;
  }

  node instance object_detector: detector_type {
       param depth = 1;
  }
}
```

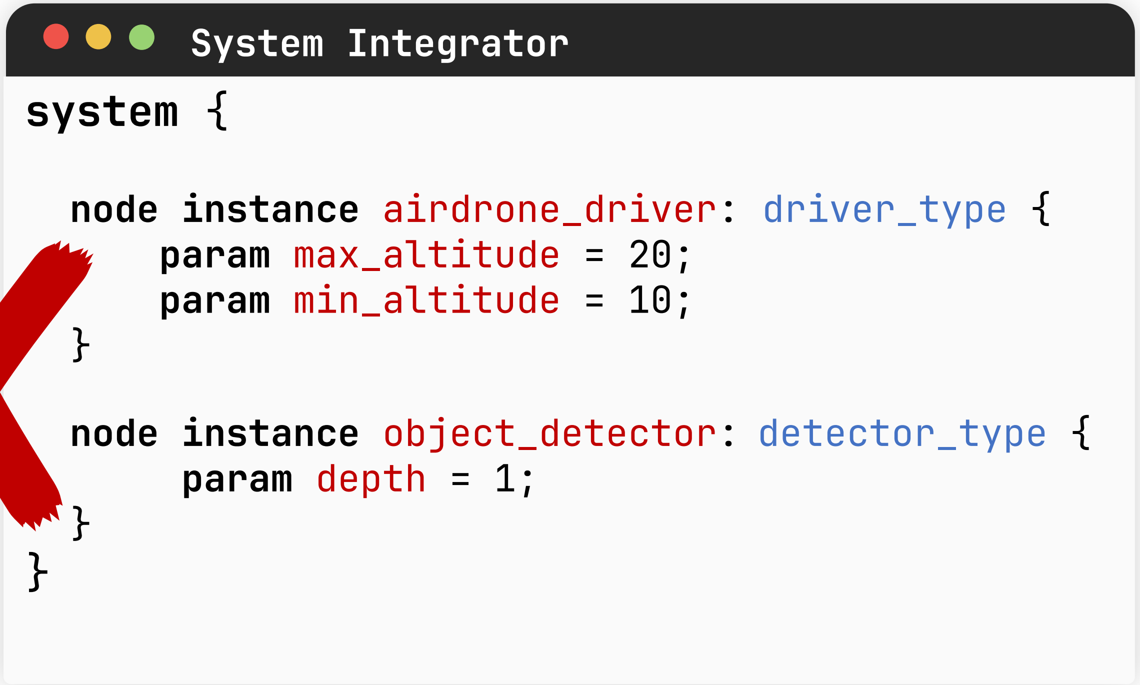
Property: A System Integrator should be able to detect missing publishers to topics containing a subscriber relationship





Component Writer

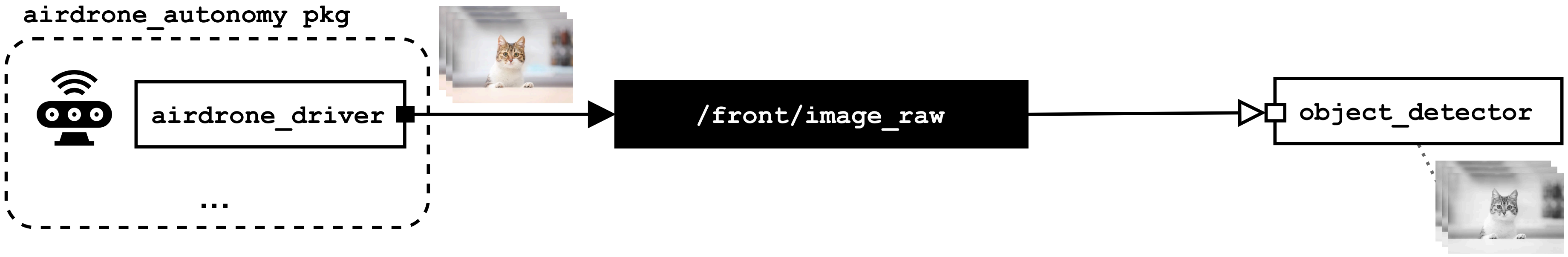
```
node type driver_type {
    param max_altitude: int where {_ > 0};
    param min_altitude: int where {_ > 0};
} where {
    min_altitude < max_altitude;
}

node type detector_type {
    param depth : Meter where {_ > 0};

    subscribes to /front/image_raw : Image;
}
```



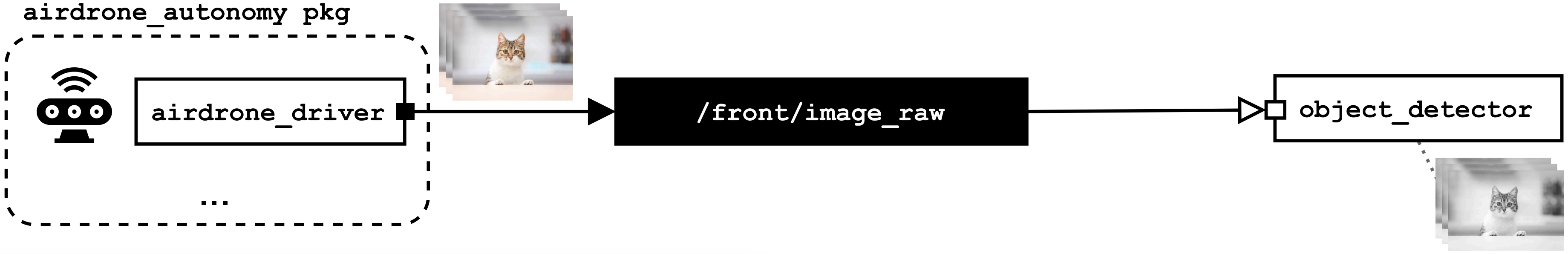
System Integrator

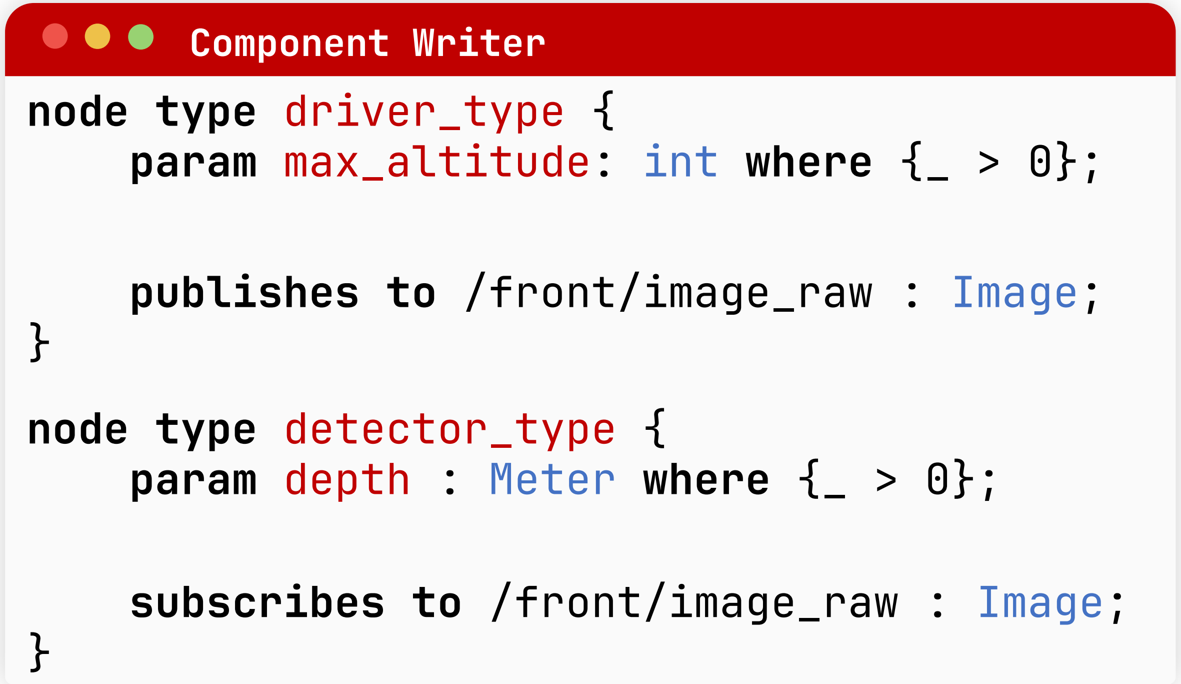
```
system {

  node instance airdrone_driver: driver_type {
      param max_altitude = 20;
      param min_altitude = 10;
  }

  node instance object_detector: detector_type {
       param depth = 1;
  }
}
```

Property: A System Integrator should be able to detect inconsistencies in the color format of images from sensors



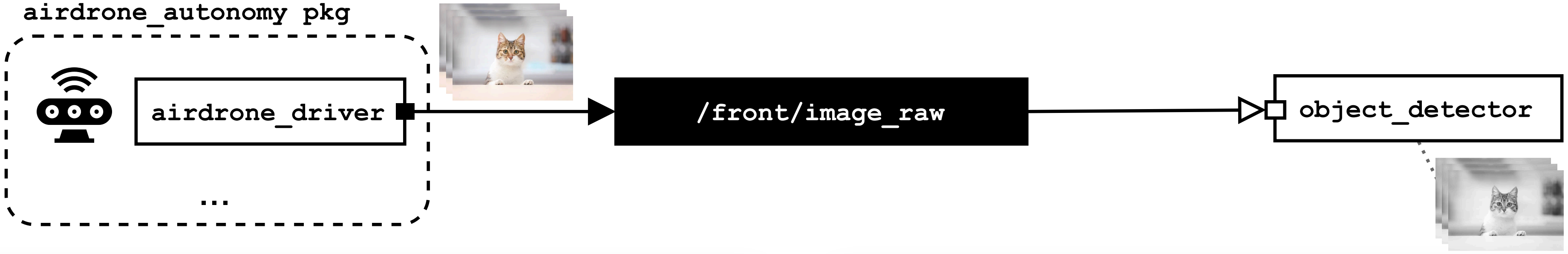
Property: A System Integrator should be able to detect inconsistencies in the color format of images from sensors

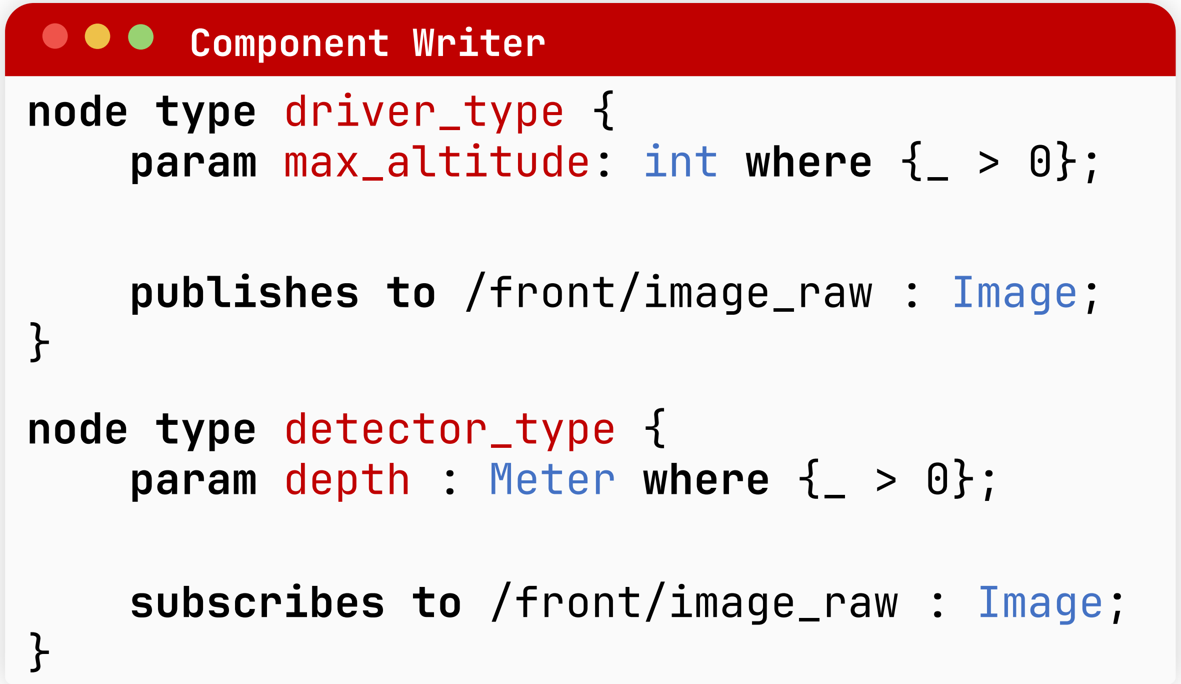




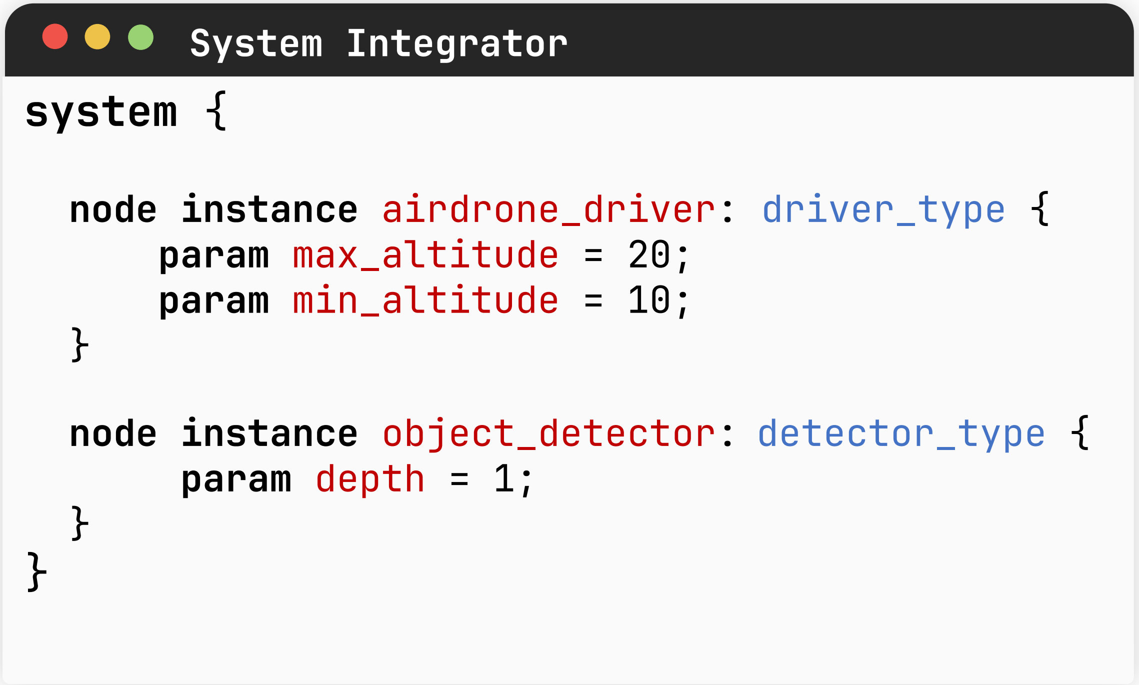
```
node type driver_type {
    param max_altitude: int where {_ > 0};


    publishes to /front/image_raw : Image;
}

node type detector_type {
    param depth : Meter where {_ > 0};


    subscribes to /front/image_raw : Image;
}
```

Property: A System Integrator should be able to detect inconsistencies in the color format of images from sensors



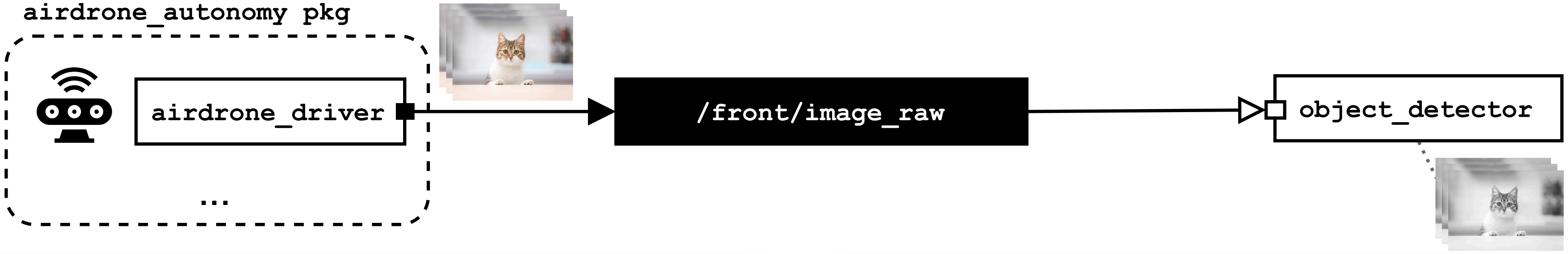


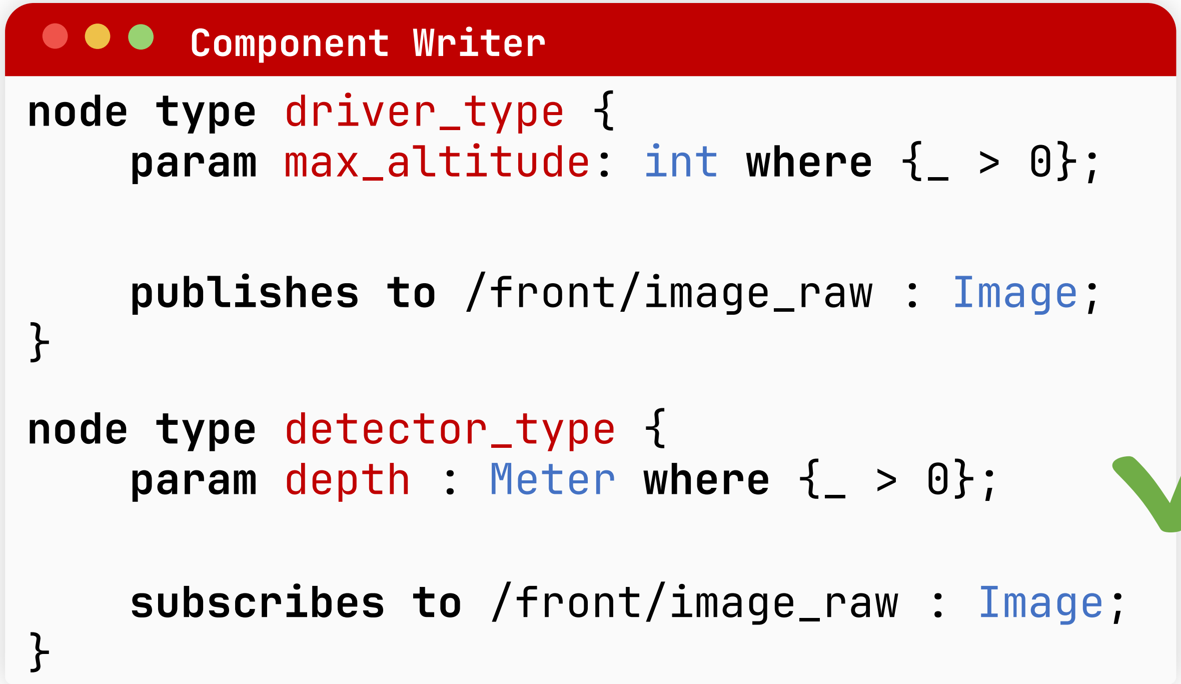
```
node type driver_type {
    param max_altitude: int where {_ > 0};


    publishes to /front/image_raw : Image;
}

node type detector_type {
    param depth : Meter where {_ > 0};


    subscribes to /front/image_raw : Image;
}
```



```
system {

  node instance airdrone_driver: driver_type {
      param max_altitude = 20;
      param min_altitude = 10;
  }

  node instance object_detector: detector_type {
       param depth = 1;
  }
}
```

Property: A System Integrator should be able to detect inconsistencies in the color format of images from sensors

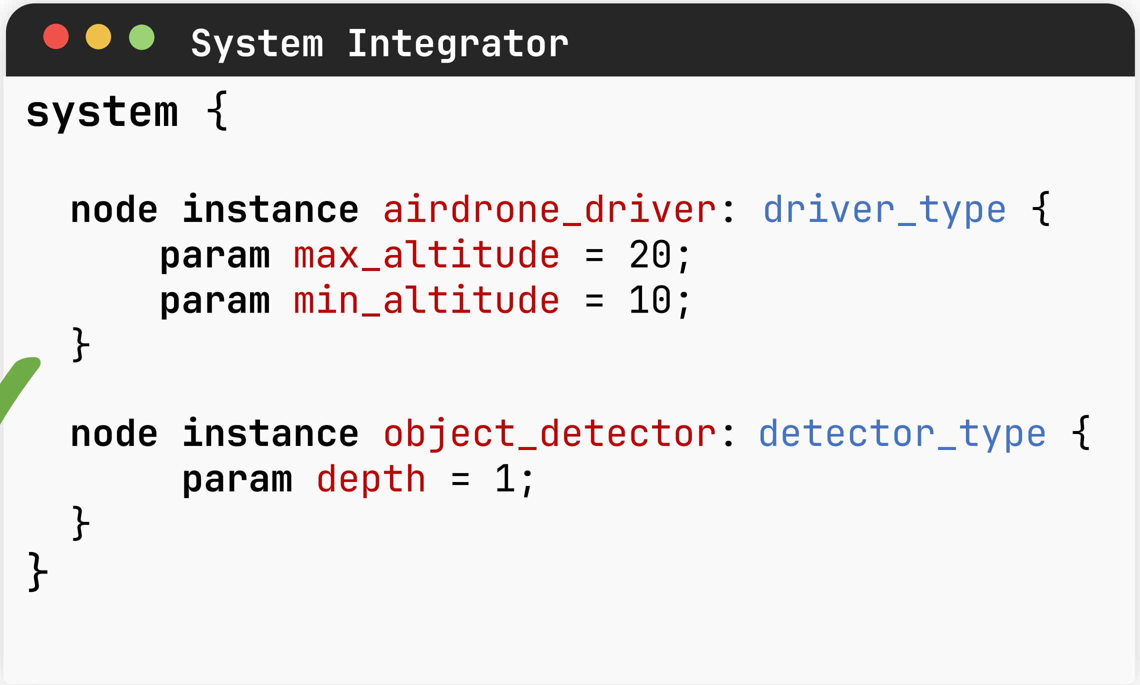




Component Writer

```
node type driver_type {
    param max_altitude: int where {_ > 0};

    publishes to /front/image_raw : Image;
}

node type detector_type {
    param depth : Meter where {_ > 0};

    subscribes to /front/image_raw : Image;
}
```

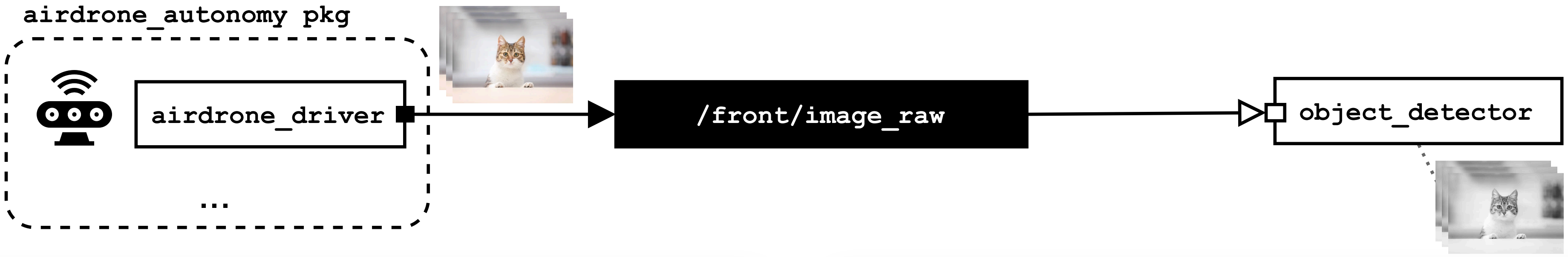
✓

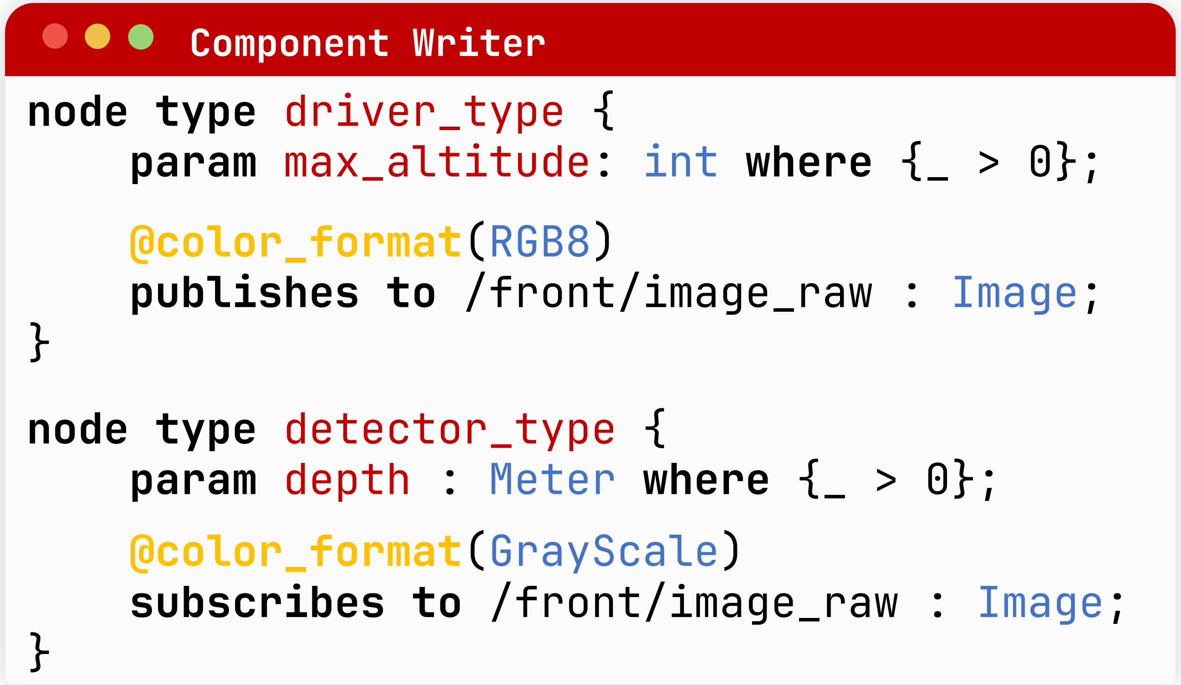


System Integrator

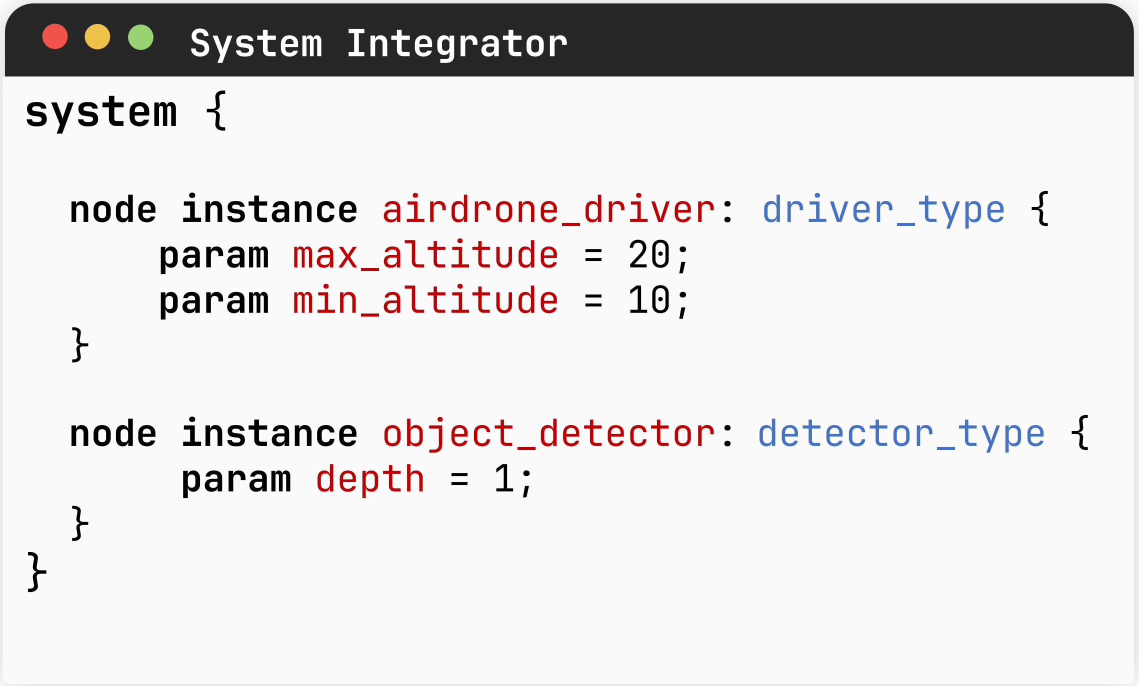
```
system {

  node instance airdrone_driver: driver_type {
      param max_altitude = 20;
      param min_altitude = 10;
  }

  node instance object_detector: detector_type {
       param depth = 1;
  }
}
```

Property: A System Integrator should be able to detect inconsistencies in the color format of images from sensors



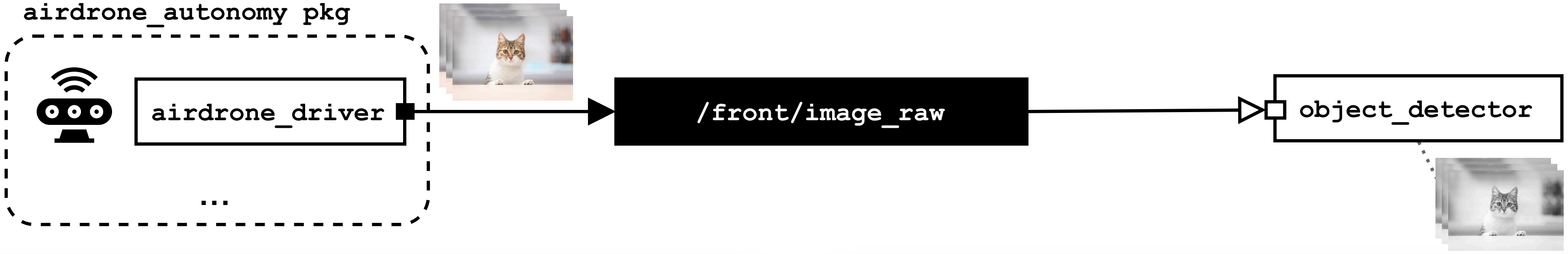


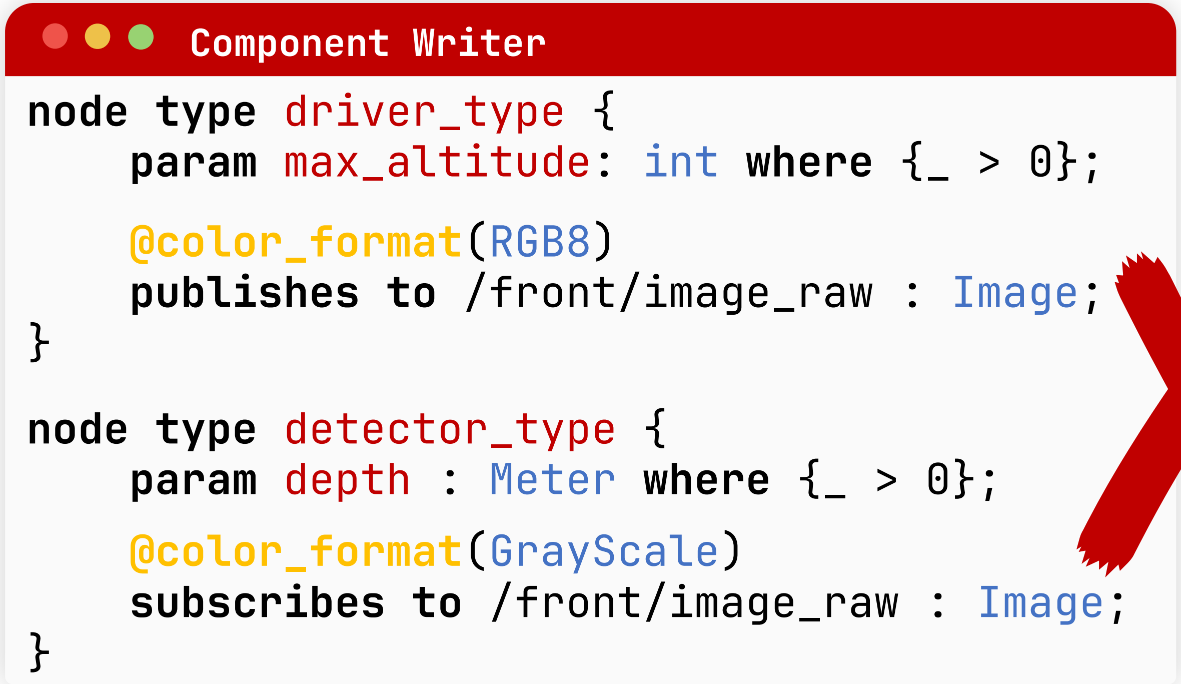
```
node type driver_type {
    param max_altitude: int where {_ > 0};

    @color_format(RGB8)
    publishes to /front/image_raw : Image;
}

node type detector_type {
    param depth : Meter where {_ > 0};

    @color_format(GrayScale)
    subscribes to /front/image_raw : Image;
}
```



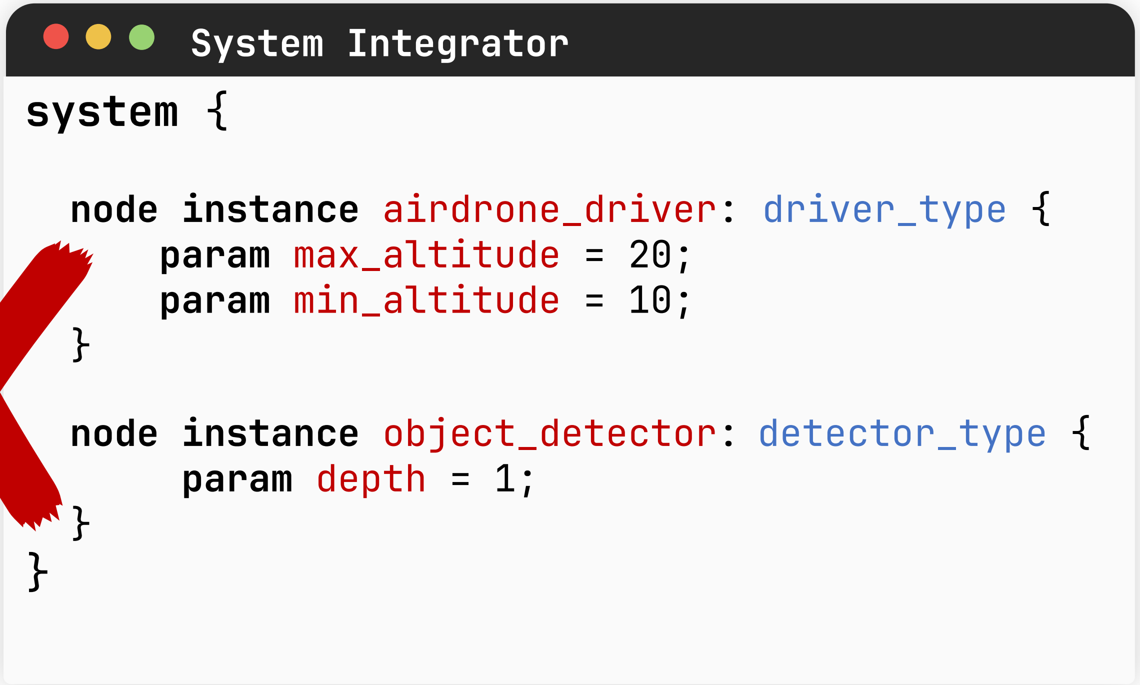
```
system {

  node instance airdrone_driver: driver_type {
      param max_altitude = 20;
      param min_altitude = 10;
  }

  node instance object_detector: detector_type {
       param depth = 1;
  }
}
```

Property: A System Integrator should be able to detect inconsistencies in the color format of images from sensors





```
node type driver_type {
    param max_altitude: int where {_ > 0};

    @color_format(RGB8)
    publishes to /front/image_raw : Image;
}

node type detector_type {
    param depth : Meter where {_ > 0};

    @color_format(GrayScale)
    subscribes to /front/image_raw : Image;
}
```



```
system {

  node instance airdrone_driver: driver_type {
      param max_altitude = 20;
      param min_altitude = 10;
  }

  node instance object_detector: detector_type {
       param depth = 1;
  }
}
```

On-going Work & Following Steps

D-NodeType

$$\frac{\begin{array}{c}\Gamma \vdash S_{p_1},\ \dots,\ S_{p_m} \dashv y_1 : T_1,\ \dots,\ y_m : T_m \quad \Gamma \vdash S_{c_1},\ \dots,\ S_{c_n} \dashv S'_{c_1},\ \dots,\ S'_{c_n} \\ \vdash U = \textbf{struct}\{\ y_1 : T_1,\ \dots,\ y_m : T_m\ \} : \text{NodeType}\end{array}}{\Gamma \vdash \textbf{node type}\ \ x\ \{\ \overline{S_p};\ \overline{S_c}\ \} \dashv \Gamma,\ x\ :\ U,\ x\ \mapsto [S'_{c_1},\ \dots,\ S'_{c_n}]}$$

D-NodeTypeWhere

$$\frac{\Gamma \vdash e : \texttt{bool} \qquad \Gamma \vdash \textbf{node type}\ \ x\ \{\ \overline{S_p};\ \overline{S_c}\ \} \qquad \Gamma \vdash x : T \quad \Gamma \vdash x \mapsto \overline{S'_c}}{\Gamma \vdash \textbf{node type}\ \ x\ \{\ \overline{S_p};\ \overline{S_c}\ \}\ \textbf{where}\ \{\ e\ \} \dashv \Gamma, x : T\ \textbf{where}\ \{\ e\ \},\ x\ \mapsto \overline{S'_c}}$$

D-NodeInstance

$$\frac{\begin{array}{c}\Gamma \vdash x_2 : T \quad \Gamma \vdash x_2 \mapsto \overline{S_{c_2}} \\ \Gamma \vdash \overline{S_p} \dashv \overline{y_p : T_p} \\ \text{We need to verify that T (which may contain dependent types) holds} \\ \Gamma \vdash \textbf{struct}\{\ \overline{y_p : T_p}\ \} <: T\end{array}}{\Gamma \vdash \textbf{node instance}\ \ x_1 : x_2\ \{\ \overline{S_p}\ \} \dashv \Gamma, x_2 : \textbf{struct}\{\ \overline{y_p : T_p}\ \},\ x_2 \mapsto \overline{S_{c_2}}}$$

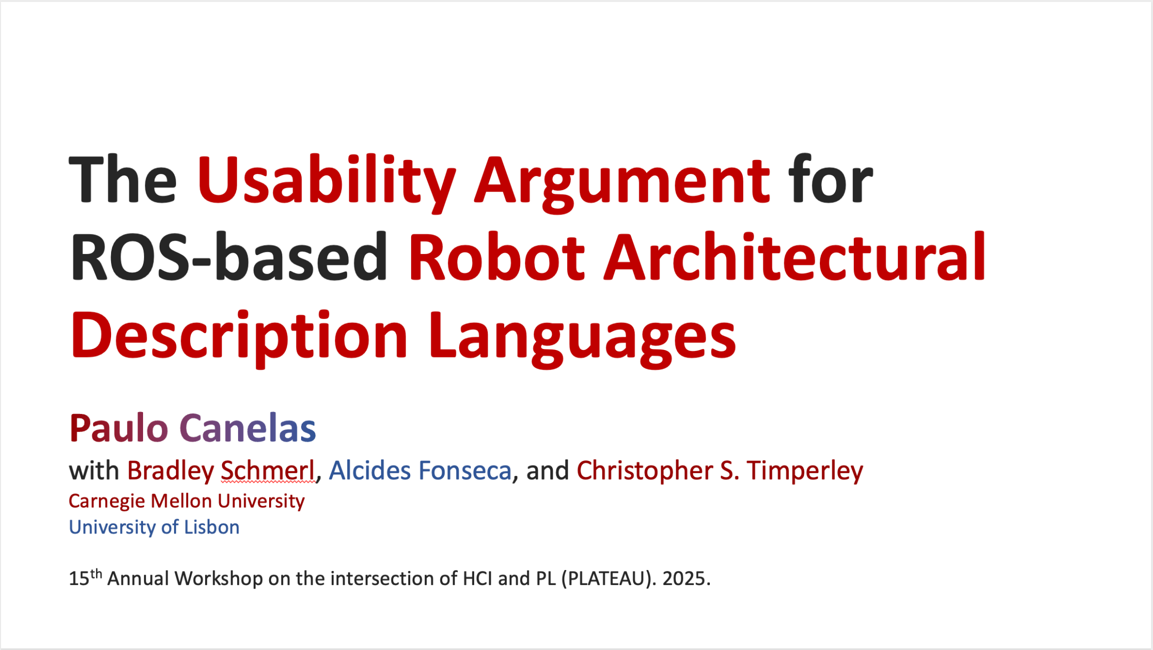
D-System

$$\frac{\Gamma \vdash \overline{D} \dashv \Gamma_2 \quad \text{Check the pub/subs}}{\Gamma \vdash \textbf{system}\ \{\ \overline{D}\ \}}$$

Formalization of the language

Case Study of a warehouse robotic system

Study the usability of ROS-based ADL



The Usability Argument for ROS-based Robot Architectural Description Languages

Paulo Canelas
with Bradley Schmerl, Alcides Fonseca, and Christopher S. Timperley
Carnegie Mellon University
University of Lisbon

15th Annual Workshop on the intersection of HCI and PL (PLATEAU). 2025.

https://pcanelas.com

pasantos@andrew.cmu.edu

*By **understanding misconfigurations** in Component-based Robot Software we can develop **usable domain-specific languages** to **specify components** and **detect misconfigurations** prior to the systems execution*