
EKUMEN

EKUMEN

Markup Descriptions in ROS 2 Launch

Michel Hidalgo, Ivan Paunovic

October 2019

Background

- What is ROS 2 launch?

Background

- What is ROS 2 launch?
  - Launch description

```
LaunchDescription([
    # ...
])
```

Background

- What is ROS 2 launch?
  - Launch description
  - Actions

```
LaunchDescription([
    Executable(cmd=[
        # ...
    ])
])
```

Background

- What is ROS 2 launch?
  - Launch description
  - Actions
  - Substitutions

```
LaunchDescription([
    Executable(cmd=[
        FindPackagePrefix('my_package'),
        '/lib/my_executable'
    ])
])
```

Background

- What is ROS 2 launch?
  - Launch description
  - Actions
  - Substitutions

```
LaunchDescription([
    Executable(cmd=[
        FindPackagePrefix('my_package'),
        '/lib/my_executable'
    ])
])
```

Overview

- Write launch files without code!
  - XML and YAML support
- Designed to be
  - Easy to understand
  - Easy to maintain
  - Extensible

Motivations

- Simplicity and convenience
- Has potential for standardization
- Easy to port roslaunch XML to ROS 2
- Better suited for validation

Markup Descriptions in ROS 2 Launch

Reasoning

```
def generate_launch_description():
    return LaunchDescription([
        Node(package='demo_nodes_cpp', node_executable='talker'),
        Node(package='demo_nodes_cpp', node_executable='listener')
    ])
```

```
<launch>
    <node pkg='demo_nodes_cpp' exec='talker'/>
    <node pkg='demo_nodes_cpp' exec='listener'/>
</launch>
```

Markup Descriptions in ROS 2 Launch

Reasoning

```
def generate_launch_description():
    return LaunchDescription([
        Node(package='demo_nodes_cpp', node_executable='talker'),
        Node(package='demo_nodes_cpp', node_executable='listener')
    ])
```

```
<launch>
    <node pkg='demo_nodes_cpp' exec='talker'/>
    <node pkg='demo_nodes_cpp' exec='listener'/>
</launch>
```

Markup Descriptions in ROS 2 Launch

Reasoning

```
def generate_launch_description():
    return LaunchDescription([
        Node(package='demo_nodes_cpp', node_executable='talker'),
        Node(package='demo_nodes_cpp', node_executable='listener')
    ])
```

```
<launch>
    <node pkg='demo_nodes_cpp' exec='talker'/>
    <node pkg='demo_nodes_cpp' exec='listener'/>
</launch>
```

Markup Descriptions in ROS 2 Launch

Reasoning

```
def generate_launch_description():
    return LaunchDescription([
        Node(package='demo_nodes_cpp', node_executable='talker'),
        Node(package='demo_nodes_cpp', node_executable='listener')
    ])
```

```
<launch>
    <node pkg='demo_nodes_cpp' exec='talker'/>
    <node pkg='demo_nodes_cpp' exec='listener'/>
</launch>
```

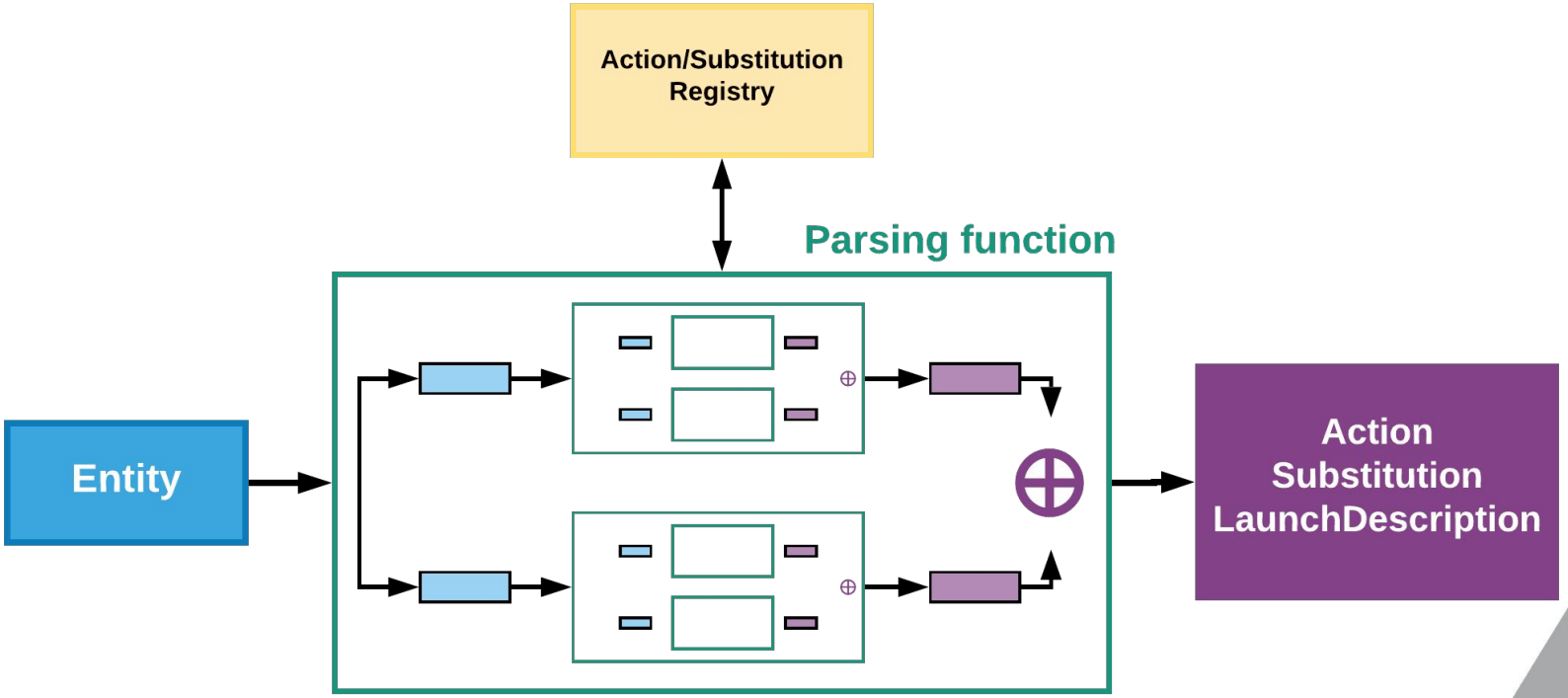
Markup Descriptions in ROS 2 Launch

Reasoning

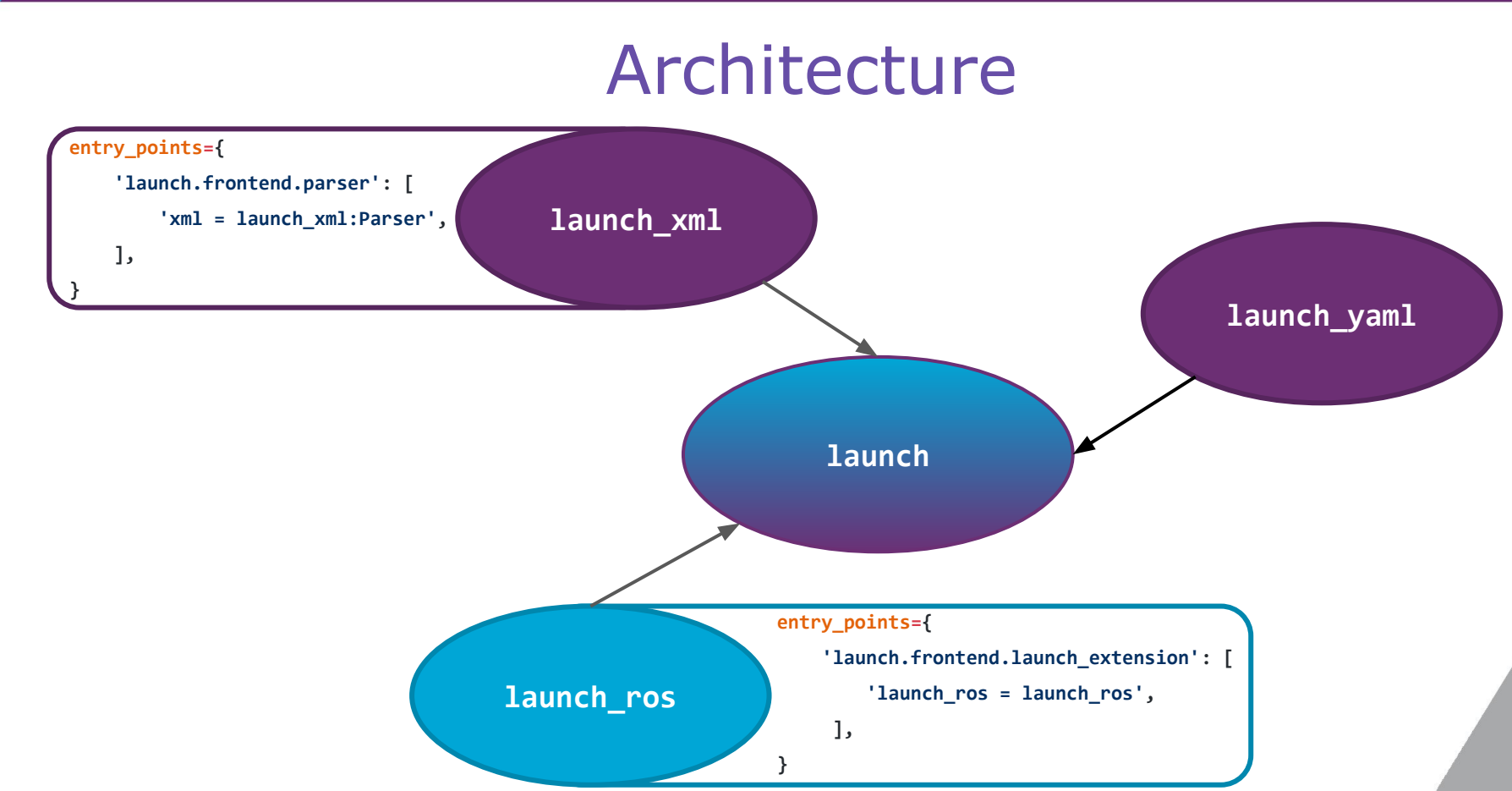
```
def generate_launch_description():
    return LaunchDescription([
        Node(package='demo_nodes_cpp', node_executable='talker'),
        Node(package='demo_nodes_cpp', node_executable='listener')
    ])
```

```
Entity (type: 'launch')
 children:
          Entity (type: 'node')
            package: 'demo_nodes_cpp'
            node_executable: 'listener'
```

Architecture



Markup Descriptions in ROS 2 Launch



Architecture

```
entry_points={
    'launch.frontend.parser': [
        'xml = launch_xml:Parser',
    ],
}
```

launch_xml

launch_yaml

launch

launch_ros

```
entry_points={
    'launch.frontend.launch_extension': [
        'launch_ros = launch_ros',
    ],
}
```

Features

- Consistency
  - One set of rules to deal with a markup
  - One way to parse an action
  - One way to parse a substitution

Markup Descriptions in ROS 2 Launch

Example: Consistent markup

```
<launch>
    <arg name="foo" default="foo"/>
    <arg name="bar" default="bar"/>
    <group if="$(var condition)">
        <node pkg="pkg" exec="exec">
            <remap from="foo" to="$(var foo)"/>
            <remap from="bar" to="$(var bar)"/>
            <param name="a_str" value="asd"/>
            <param name="an_int_list" value="1, 2, 3" value-sep=", "/>
        </node>
        <executable cmd="my_cmd">
            <env name="MY_ENV" value="MY_VALUE"/>
            <env name="MY_ENV2" value="MY_VALUE2"/>
        </executable>
    </group>
    <executable cmd="my_other_cmd" output="screen"/>
</launch>
```

Markup Descriptions in ROS 2 Launch

Example: Consistent markup

```
<launch>
    <arg name="foo" default="foo"/>
    <arg name="bar" default="bar"/>
    <group if="$(var condition)">
        <node pkg="pkg" exec="exec">
            <remap from="foo" to="$(var foo)"/>
            <remap from="bar" to="$(var bar)"/>
            <param name="a_str" value="asd"/>
            <param name="an_int_list" value="1, 2, 3" value-sep=", "/>
        </node>
        <executable cmd="my_cmd">
            <env name="MY_ENV" value="MY_VALUE"/>
            <env name="MY_ENV2" value="MY_VALUE2"/>
        </executable>
    </group>
    <executable cmd="my_other_cmd" output="screen"/>
</launch>
```

Takes list of actions

Takes list of actions

Markup Descriptions in ROS 2 Launch

Example: Consistent markup

```
<launch>
    <arg name="foo" default="foo"/>
    <arg name="bar" default="bar"/>
    <group if="$(var condition)">
        <node pkg="pkg" exec="exec">
            <remap from="foo" to="$(var foo)"/>
            <remap from="bar" to="$(var bar)"/>
            <param name="a_str" value="asd"/>
            <param name="an_int_list" value="1, 2, 3" value-sep=", "/>
        </node>
        <executable cmd="my_cmd">
            <env name="MY_ENV" value="MY_VALUE"/>
            <env name="MY_ENV2" value="MY_VALUE2"/>
        </executable>
    </group>
    <executable cmd="my_other_cmd" output="screen"/>
</launch>
```

remap: a list of a complex type

param: a list of a complex type

env: a list of a complex type

Markup Descriptions in ROS 2 Launch

Example: Consistent markup

```
<launch>
    <arg name="foo" default="foo"/>
    <arg name="bar" default="bar"/>
    <group if="$(var condition)">
        <node pkg="pkg" exec="exec">
            <remap from="foo" to="$(var foo)"/>
            <remap from="bar" to="$(var bar)"/>
            <param name="a_str" value="asd"/>
            <param name="an_int_list" value="1, 2, 3" value-sep=", "/>
        </node>
        <executable cmd="my_cmd">
            <env name="MY_ENV" value="MY_VALUE"/>
            <env name="MY_ENV2" value="MY_VALUE2"/>
        </executable>
    </group>
    <executable cmd="my_other_cmd" output="screen"/>
</launch>
```

value: a list of scalar types

Markup Descriptions in ROS 2 Launch

Example: Consistency across markups

```yaml
launch:
- group:
  - push_ros_namespace:
      namespace: 'my_ns'
  - node:
      pkg: my_pkg
      exec: my_node
      param:
      - name: a_str
        value: asd
      - name: an_int_list
        value: [1, 2, 3]
  - node:
      pkg: my_pkg
      exec: another_node
```

```xml
<launch>
  <group>
    <push_ros_namespace namespace="my_ns"/>
    <node pkg="my_pkg" exec="my_node">
      <param name="a_str" value="asd"/>
      <param name="an_int_list"
        value="1, 2, 3"
        value-sep=", "/>
    </node>
    <node pkg="my_pkg" exec="another_node"/>
  </group>
</launch>
```

Markup Descriptions in ROS 2 Launch

Example: Consistency across markups

```yaml
launch:
- group:
  - push_ros_namespace:
      namespace: 'my_ns'
  - node:
      pkg: my_pkg
      exec: my_node
      param:
      - name: a_str
        value: asd
      - name: an_int_list
        value: [1, 2, 3]
  - node:
      pkg: my_pkg
      exec: another_node
```

```xml
<launch>
  <group>
    <push_ros_namespace namespace="my_ns"/>
    <node pkg="my_pkg" exec="my_node">
      <param name="a_str" value="asd"/>
      <param name="an_int_list"
                  value="1, 2, 3"
                  value-sep=", "/>
    </node>
    <node pkg="my_pkg" exec="another_node"/>
  </group>
</launch>
```

Features

- Consistency
  - One set of rules to deal with a markup
  - One way to parse an action
  - One way to parse a substitution

- Extensibility
  - One parsing function per type
  - One Entity class per markup

Example: Parsing a custom action

```
@launch.frontend.expose_action('in_order_group')
class InOrderGroup(Action):

    def __init__(self,
                 actions: Iterable[Action],
                 continue_after_fail: Union[bool, SomeSubstitutionsType]):
        self.__actions = actions
        self.__continue_after_fail = continue_after_fail

    @classmethod
    def parse(cls, entity: launch.frontend.Entity, parser: launch.frontend.Parser):
        ...
```

Example: Parsing a custom action

```
@launch.frontend.expose_action('in_order_group')
class InOrderGroup(Action):

    def __init__(self,
                 actions: Iterable[Action],
                 continue_after_fail: Union[bool, SomeSubstitutionsType]):
        self.__actions = actions
        self.__continue_after_fail = continue_after_fail

    @classmethod
    def parse(cls, entity: launch.frontend.Entity, parser: launch.frontend.Parser):
        ...
```

Markup Descriptions in ROS 2 Launch

Example: Parsing a custom action

```
@launch.frontend.expose_action('in_order_group')
class InOrderGroup(Action):

    def __init__(self,
                 actions: Iterable[Action],
                 continue_after_fail: Union[bool, SomeSubstitutionsType]):
        self.__actions = actions
        self.__continue_after_fail = continue_after_fail

    @classmethod
    def parse(cls, entity: launch.frontend.Entity, parser: launch.frontend.Parser):
        ...
```

Decorator exposing the class to the parser.

Class method taking an entity and a parser.

Example: Parsing a custom action

```
@classmethod
def parse(cls, entity: launch.frontend.Entity, parser: launch.frontend.Parser):
    _, kwargs = super().parse(entity, parser)
    continue_after_fail = entity.get_attr(
        'continue_after_fail', data_type=Union[bool, str],
        optional=True)
    if isinstance(continue_after_fail, str):
        continue_after_fail = parser.parse_substitution(continue_after_fail)
    if continue_after_fail is not None:
        kwargs['continue_after_fail'] = continue_after_fail
    kwargs['actions'] = [parser.parse_action(e) for e in entity.children]
    return cls, kwargs
```

Reuse parsing method from the parent class.

Markup Descriptions in ROS 2 Launch

Example: Parsing a custom action

```
@classmethod
def parse(cls, entity: launch.frontend.Entity,                    arser):
    _, kwargs = super().parse(entity, parser)
    continue_after_fail = entity.get_attr(
        'continue_after_fail', data_type=Union[bool, str],
        optional=True)
    if isinstance(continue_after_fail, str):
        ail = parser.parse_substitution(continue_after_fail)
    if    il is not None:
        kwargs['continue_after_fail'] = continue_after_fail
    kwargs['actions'] = [parser.parse_action(e) for e in entity.children]
    return cls, kwargs
```

Using get_attr method, specifying the allowed types.

Attribute may not be specified by the user.

Markup Descriptions in ROS 2 Launch

Example: Parsing a custom action

```
@classmethod
def parse(cls, entity: launch.frontend.Entity, parser: launch.frontend.Parser):
    _, kwargs = super().parse(entity, parser)
    continue_after_fail = entity.get_attr(
        'continue_after_fail', data_type=
        optional=True)
    if isinstance(continue_after_fail, str):
        continue_after_fail = parser.parse_substitution(continue_after_fail)
    if continue_after_fail is not None:
        kwargs['continue_after_fail'] = continue_after_fail
    kwargs['actions'] = [parser.parse_action(e) for e in entity.children]
    return cls, kwargs
```

Use parse_substitution to interpolate $(...) substitutions.

Example: Parsing a custom action

```
@classmethod
def parse(cls, entity: launch.frontend.Entity, parser: launch.frontend.Parser):
    _, kwargs = super().parse(entity, parser)
    continue_after_fail = entity.get_attr(
        'continue_after_fail', data_type=Union[bool, str],
        optional=True)
    if isinstance(continue_after_fail, str):
        continue_after_fail = parser.parse_substitution(continue_after_fail)
    if continue_after_fail is not None:
        kwargs['continue_after_fail'] = continue_after_fail
    kwargs['actions'] = [parser.parse_action(e) for e in entity.children]
    return cls, kwargs
```

Example: Parsing a custom action

```
@classmethod
def parse(cls, entity: launch.frontend.Entity, parser: launch.frontend.Parser):
    _, kwargs = super().parse(entity, parser)
    continue_after_fail = entity.get_attr(
        'continue_after_fail', data_type=Union[bool, str],
        optional=True)
    if isinstance(continue_after_fail, str):
        continue_after_fail = parser.parse_substitution(continue_after_fail)
    if continue_after_fail is not None:
        kwargs['continue_after_fail'] = continue_after_fail
    kwargs['actions'] = [parser.parse_action(e) for e in entity.children]
    return cls, kwargs
```

Markup Descriptions in ROS 2 Launch

Example: Parsing a custom action

```
<launch>
    <in_order_group continue_after_fail="true">
        <in_order_group>
            <executable cmd="mkdir ~/my_new_folder"/>
            <executable cmd="touch ~/my_new_folder/my_file"/>
            <executable cmd="ls ~/my_new_folder/my_file"/>
        </in_order_group>
        <executable cmd="rm -fr ~/my_new_folder"/>
    </in_order_group>
</launch>
```

Example: Parsing a custom action

```
launch:
- in_order_group:
    continue_after_fail: True
    children:
    - in_order_group:
      - executable:
        cmd: mkdir ~/my_new_folder
      - executable:
        cmd: touch ~/my_new_folder/my_file"
      - executable:
        cmd: ls ~/my_new_folder/my_file
    - executable:
       cmd: mkdir ~/my_new_folder
```

Summary

- ROS 2 launch files in XML and YAML
- Easy to pick up
  - Consistent writing
- Easy to extend from external packages
  - Expose actions
  - Expose substitutions
  - Support markups

Future work

- Expose launch event system to the markup descriptions
  - How to emit events?
  - How to handle events?
  - How to target actions?

- Add namespaces for actions and substitutions
  - To avoid tag name collisions

Future work

- Normalize substitution value type inference
- Add more parsing functions!
  - For actions like TimerAction
  - For substitutions

Resources

- Migration guide from ROS 1 to ROS 2
- ROS 2 Launch design document
- ROS 2 Launch Python architecture document
- ROS 2 launch XML format

Questions?

ivanpauno@ekumenlabs.com
michel@ekumenlabs.com