

MoTiVML Guideline Document

This is a guideline document for the MoTiVML language. This guideline document assists end users in modelling variability in robotic systems as well as implementing the features they have defined in their models. As a supporting artifact to the language, this document serves as a form of verification for the MoTiVML language and its capabilities.

1 Prerequisites

- Ubuntu 20.04.3 LTS and above
- ROS installation. This application was developed and tested with ROS Neotic 1.15.9.
- Python3 installation
- C++ 11 or higher

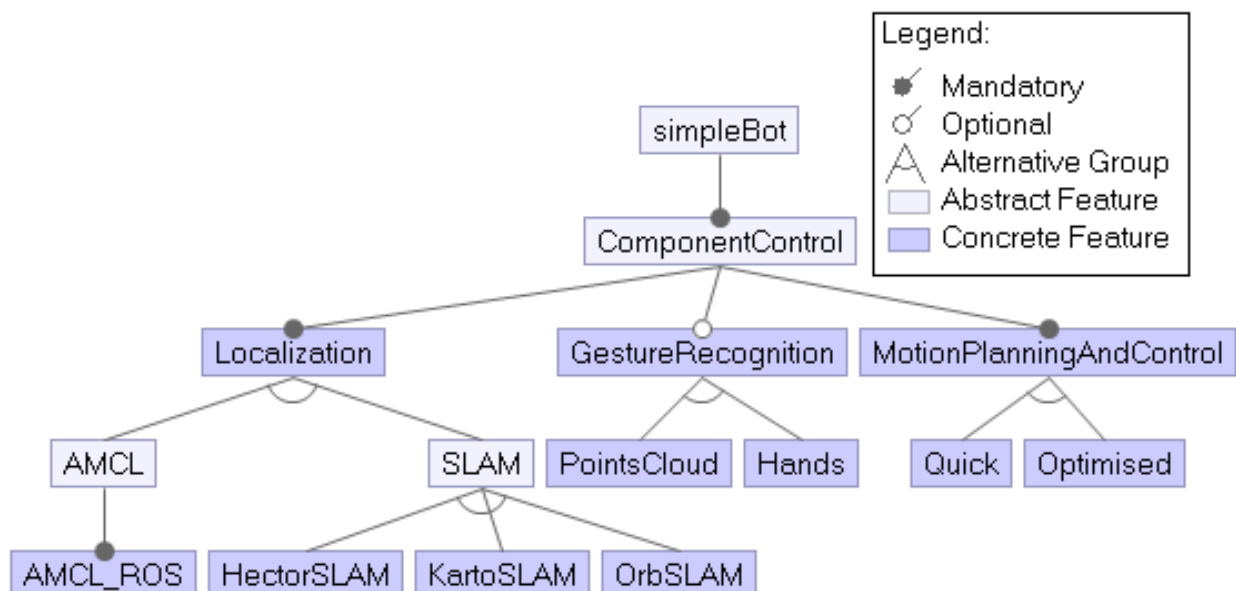
2 Setup and Installation of MoTiVML

- Clone MoTiVML from [here](#) into your catkin workspace.
- Go the MoTiVML workspace project directory and build the cloned project with the command `catkin_make` and press enter on your keyboard.

3 How To

This section gives a step-by-step walk-through of how MoTiVML can be used to instantiate a model or configuration, how it can be used to validate a model as well as the capability to link source code implementations with feature definitions. We also give a brief tutorial of our in-built MoTiVML console interface. Throughout our demonstrations in this guideline document, we use a sample model of a robotic system that is shown graphically in figure 1. This graphical representation of a simple robot model will also serve as our running example throughout this document.

Figure 1: Graphical Model of a Simple Robot



3.1 Instantiate a MoTiVML Model

- To create an instance of a model in MoTiVML, in the **src** folder of the cloned application, make a copy of the template folder provided and rename it to suit your preference.
- In your instance folder, there is a **model.json** with an existing root feature, wrapped in a object.

- The model definition can be extended by appending sub features to the root feature. You can add nested feature objects according to your preferences. Note that each feature specification must have attributes such as **id**, **name**, **constraints**, **group**, **isMandatory**. The constraints attribute must contain sub attributes **featureIncluded**, **featureExcluded**, **bindingTimeAllowed**, **bindingModeAllowed**. Listing 1 shows a demonstrated example of an extended MoTiVML model instance, for the feature model captured in Figure 1.
- The value of each feature attribute in a model must conform to strict type systems defined within MoTiVML. A list of all the attributes together with their expected values is provided below in Table 1.

Table 1: Feature Attributes and Types

Attribute	Expected Value
id	Alphanumeric characters. No spaces allowed.
name	Alphanumeric characters. No spaces allowed.
constraints	Object containing attributes featureIncluded , featureExcluded , bindingTimeAllowed , bindingModeAllowed
featureIncluded	Array of existing feature "ids"
featureExcluded	Array of existing feature "ids"
bindingTimeAllowed	Early / Late / Any
bindingModeAllowed	Static / Dynamic / Any
group	XOR / OR
isMandatory	True or False

- To add a sub-feature to a defined feature, add the **sub** attribute to the feature and assign an array of sub-feature objects to it. However, the sub attribute is optional. Features that do not have sub-features can exist without a **sub** attribute.

Listing 1: SimpleRobot Model Instantiation in MoTiVML

```

1  {
2
3      "id": "root_feature",
4      "name": "SimpleBot",
5      "group": "",
6      "isMandatory": true,
7      "isSelected": true,
8      "sub": [
9
10         {
11             "id": "compcontrol",
12             "name": "ComponentControl",
13             "constraints": {
14                 "featuresIncluded": [],
15                 "featuresExcluded": [],
16                 "bindingTimeAllowed": "Early",
17                 "bindingModeAllowed": "Static"
18             },
19             "group": "OR",
20             "isMandatory": true,
21             "sub": [
22
23                 {
24                     "id": "localztn",
25                     "name": "Localisation",
26                     "constraints": {
27                         "featuresIncluded": [],
28                         "featuresExcluded": [],
29                         "bindingTimeAllowed": "Early",
30                         "bindingModeAllowed": "Static"
31                     },
32                     "group": "OR",
33                     "isMandatory": true,
34                     "sub": [
35
36                         {
37                             "id": "amcl",
38                             "name": "AMCL",
39                             "constraints": {
40                                 "featuresIncluded": [],
41                                 "featuresExcluded": [],
42                                 "bindingTimeAllowed": "Early",
43                                 "bindingModeAllowed": "Static"
44                             },
45                             "group": "XOR",
46                             "isMandatory": false,

```

```

44         "sub": [
45             {
46                 "id": "amclros",
47                 "name": "AmclRos",
48                 "constraints": {
49                     "featuresIncluded": [],
50                     "featuresExcluded": [],
51                     "bindingTimeAllowed": "Early",
52                     "bindingModeAllowed": "Static"
53                 },
54                 "group": "OR",
55                 "isMandatory": true
56             }
57         ],
58     },
59     {
60         "id": "slam",
61         "name": "SLAM",
62         "constraints": {
63             "featuresIncluded": [],
64             "featuresExcluded": [],
65             "bindingTimeAllowed": "Early",
66             "bindingModeAllowed": "Static"
67         },
68         "group": "XOR",
69         "isMandatory": false,
70         "sub": [
71             {
72                 "id": "hectorslam",
73                 "name": "HectorSLAM",
74                 "constraints": {
75                     "featuresIncluded": [],
76                     "featuresExcluded": ["kartoslam", "orbslam"],
77                     "bindingTimeAllowed": "Late",
78                     "bindingModeAllowed": "Dynamic"
79                 },
80                 "group": "XOR",
81                 "isMandatory": false
82             },
83             {
84                 "id": "kartoslam",
85                 "name": "KartoSLAM",
86                 "constraints": {
87                     "featuresIncluded": [],
88                     "featuresExcluded": ["orbslam", "hectorslam"],
89                     "bindingTimeAllowed": "Late",
90                     "bindingModeAllowed": "Dynamic"
91                 },
92                 "group": "XOR",
93                 "isMandatory": false
94             },
95             {
96                 "id": "orbslam",
97                 "name": "OrbSLAM",
98                 "constraints": {
99                     "featuresIncluded": [],
100                    "featuresExcluded": ["kartoslam", "hectorslam"],
101                    "bindingTimeAllowed": "Late",
102                    "bindingModeAllowed": "Dynamic"
103                },
104                "group": "XOR",
105                "isMandatory": false
106            }
107        ]
108    },
109 ],
110 },
111 {
112     "id": "gestrec",
113     "name": "GestureRecognition",
114     "constraints": {
115         "featuresIncluded": [],
116         "featuresExcluded": [],
117         "bindingTimeAllowed": "Early",
118         "bindingModeAllowed": "Static"

```

```

119         },
120         "group": "OR",
121         "isMandatory": false,
122         "sub": [
123         {
124             "id": "hands",
125             "name": "Hands",
126             "constraints": {
127                 "featuresIncluded": [],
128                 "featuresExcluded": ["pointscld"],
129                 "bindingTimeAllowed": "Late",
130                 "bindingModeAllowed": "Dynamic"
131             },
132             "group": "XOR",
133             "isMandatory": false
134         },
135         {
136             "id": "pointscld",
137             "name": "PointsCloud",
138             "constraints": {
139                 "featuresIncluded": [],
140                 "featuresExcluded": ["hands"],
141                 "bindingTimeAllowed": "Late",
142                 "bindingModeAllowed": "Dynamic"
143             },
144             "group": "XOR",
145             "isMandatory": false
146         }
147     ]
148 },
149 {
150     "id": "motplannctrl",
151     "name": "MotionPlanningAndControl",
152     "constraints": {
153         "featuresIncluded": [],
154         "featuresExcluded": [],
155         "bindingTimeAllowed": "Early",
156         "bindingModeAllowed": "Static"
157     },
158     "group": "OR",
159     "isMandatory": true,
160     "sub": [
161     {
162         "id": "quick",
163         "name": "Quick",
164         "constraints": {
165             "featuresIncluded": [],
166             "featuresExcluded": ["optimised"],
167             "bindingTimeAllowed": "Late",
168             "bindingModeAllowed": "Dynamic"
169         },
170         "group": "XOR",
171         "isMandatory": false
172     },
173     {
174         "id": "optimised",
175         "name": "Optimised",
176         "constraints": {
177             "featuresIncluded": [],
178             "featuresExcluded": ["quick"],
179             "bindingTimeAllowed": "Late",
180             "bindingModeAllowed": "Dynamic"
181         },
182         "group": "XOR",
183         "isMandatory": false
184     }
185     ]
186 }
187 ]
188 }
189 ]
190 }
191

```

3.2 Instantiate a MoTiVML Configuration

- Again in your instance folder, there is a **config.json** with an empty **properties** array value.
- In the config.json file of your project, for every feature added in your model.json file, a corresponding configuration object must be added. For each feature configuration object, there must be an **id** attribute which references the **id** of a feature in your model.json file. The **props** attribute must contain **time** and **mode** attributes. Listing 2 shows a MoTiVML configuration translation of the features present in Figure 1.
- The values of each configuration attribute for your model must conform to strict type systems defined within MoTiVML. A list of all the configuration attributes together with their expected values is provided below in Table 2.

Table 2: Configuration Attributes and Types

Attribute	Expected Value
id	Alphanumeric characters. No spaces allowed.
props	Object containing attributes time and mode
time	Early / Late
mode	Static / Dynamic

Listing 2: SimpleRobot Model Configuration Instantiation in MoTiVML

```
1 {
2
3   "properties": [
4     {
5       "id": "compcontrol",
6       "props": {
7         "time": "Early",
8         "mode": "Static"
9       }
10    },
11    {
12      "id": "localztn",
13      "props": {
14        "time": "Early",
15        "mode": "Static"
16      }
17    },
18    {
19      "id": "gestrec",
20      "props": {
21        "time": "Early",
22        "mode": "Static"
23      }
24    },
25    {
26      "id": "motplannctrl",
27      "props": {
28        "time": "Early",
29        "mode": "Static"
30      }
31    },
32    {
33      "id": "amcl",
34      "props": {
35        "time": "Early",
36        "mode": "Static"
37      }
38    },
39    {
40      "id": "amclros",
41      "props": {
42        "time": "Early",
43        "mode": "Static"
44      }
45    },
46    {
47      "id": "slam",
48      "props": {
49        "time": "Early",
50        "mode": "Static"
51      }
52    }
53  ]
54 }
```

```

52     },
53     {
54         "id": "hectorslam",
55         "props": {
56             "time": "Late",
57             "mode": "Dynamic"
58         }
59     },
60     {
61         "id": "kartoslam",
62         "props": {
63             "time": "Late",
64             "mode": "Dynamic"
65         }
66     },
67     {
68         "id": "orbslam",
69         "props": {
70             "time": "Late",
71             "mode": "Dynamic"
72         }
73     },
74     {
75         "id": "hands",
76         "props": {
77             "time": "late",
78             "mode": "Dynamic"
79         }
80     },
81     {
82         "id": "pointscld",
83         "props": {
84             "time": "Late",
85             "mode": "Dynamic"
86         }
87     },
88     {
89         "id": "quick",
90         "props": {
91             "time": "Late",
92             "mode": "Dynamic"
93         }
94     },
95     {
96         "id": "optimised",
97         "props": {
98             "time": "Late",
99             "mode": "Dynamic"
100        }
101    }
102 ]
103 }

```

3.3 Standalone Model Validation

To validate your model and its configuration, navigate to the `/src/motivml` directory and run the following command in the ROS console:

Listing 3: MoTiVML Validation Command

```

1
2 python validator.py <project_directory_name>
3

```

All models created in the MoTiVML language are validated on two distinct levels. i.e. syntactically and semantically. Syntactical validation has to do with feature and configuration schemas being evaluated for errors, while semantic validation has to do with modelling and binding constraints evaluation.

To demonstrate this we evaluate our constructed model and configuration shown in Listing 1 and 2 and discuss the results accordingly. A careful observation of the console output generated from the validation indicates the presence of zero errors.

When our in-built schema and constraint checker identify an error in either a model or a configuration, an error message is outputted to the console indicating where the error is located and why the language has flagged it as an error.

Figure 2: Simplebot Validation Output

```
C:\catkin_ws\src\motivml_ros\src\motivml>python motivml.py simplebot
Compiling simplebot
Generating Early Bindings
-----
Reading configuration: C:\catkin_ws\src\motivml_ros\src\simplebot\config.json
-----Parsing Model-----
[SUCCESS] Model Schema Has Zero Syntax Errors
-----Model Parsing Ended-----

-----Parsing Binding-----
[SUCCESS] Model Binding Schema Has Zero Syntax Errors
----- Binding Parsing Ended-----

-----Constraint Checker Validation-----
[SUCCESS] Constraint Checker Found Zero Constraint Errors
```

3.4 Implement a Statically Bound Feature

3.4.1 Static Early

- Create a static class in the namespace **static_integration** which inherits from the class **public static_base::StaticInterface**. NB: The feature id in your model must match your class name. For Example:

Listing 4: Sample Static Early Class

```
1 namespace static_integration{
2
3
4     class MyStaticEarlyClass: public static_base::StaticInterface{
5         std::string feature_id = "my_unique_feature_id";
6     public:
7         MyStaticEarlyClass(){};
8
9         void executeFeature(){
10             std::cout << "Hello My Feature!!" << std::endl;
11         };
12
13     };
14
15 };
16
```

- Set the class variable **feature_id** to the same value as the **id** attribute in the MoTiVML model.
- Add the following preprocessor include above your class to include the static base interface class your class will be inheriting from:

Listing 5: Class Inclusion

```
1
2 #include "../include/motivml_ros/static_base.h"
3
```

- Define a static early feature constant at the top of **plugin_feature_loader.cpp**. For Example

Listing 6: Constant Definition

```
1
2 #define UNKOWNFEATURE
3
```

- In **static_feature_loader.h**, if the feature constant you created in the previous step is true, include your static class. See Listing 7 as an example

Listing 7: Including Static Early

```

1
2  #ifdef UNKNOWNFEATURE
3  #include "../featx/MyStaticClass.h"
4  #endif
5

```

- Extend the static early conditional block of the **callback_load_plugin_features** method in **plugin_feature_loader.cpp** to run a method in your static early class.
- Run the **catkin_make** command to build the project.

3.4.2 Static Late

- Create a static class in the namespace **static_integration** which inherits from the class **public static_base::StaticInterface**. NB: The feature id in your model must match your class name. For Example:

Listing 8: Sample Static Late Class

```

1
2 namespace static_integration{
3
4     class MyStaticEarlyClass: public static_base::StaticInterface{
5         std::string feature_id = "my_unique_feature_id";
6         public:
7         MyStaticEarlyClass(){};
8
9         void executeFeature(){
10             std::cout << "Hello My Feature!!" << std::endl;
11         };
12
13     };
14
15 };
16

```

- Encapsulate Feature As Plugin
- Register/Export Plugin

3.5 Implement a Dynamically Bound Feature

3.5.1 Dynamic Early

- Encapsulate Feature As Plugin
- Register/Export Plugin

3.5.2 Dynamic Late

- Encapsulate Feature As Plugin
- Register/Export Plugin

3.6 MoTiVML Console Interface

To add some interactivity to our variability modelling language, we have provided an in-built console interface for interacting with models and features through MoTiVML specific commands.

To launch the MoTiVML console interface, navigate to the **/src/motivml** directory and run the following command in Listing 9:

Listing 9: MoTiVML Console Launch Command

```

1
2 python mmconsole.py <project_directory_name>
3

```

When the above command is run, the model and configuration defined in the **project_directory_name** is compiled and validated, prior to the console interface being launched. If there are no errors detected, the console interface is then launched successfully.

Figure 3: Simplebot Console

```
C:\catkin_ws\src\motivml_ros\src\motivml>python mmconsole.py simplebot
Compiling simplebot
Generating Early Bindings
-----
Reading configuration: C:\catkin_ws\src\motivml_ros\src\simplebot\config.json
-----Parsing Model-----
[SUCCESS] Model Schema Has Zero Syntax Errors
-----Model Parsing Ended-----

-----Parsing Binding-----
[SUCCESS] Model Binding Schema Has Zero Syntax Errors
----- Binding Parsing Ended-----

-----Constraint Checker Validation-----
[SUCCESS] Constraint Checker Found Zero Constraint Errors
MoTiVML:[simplebot]>> █
```

In Figure 3 we can observe the launched MoTiVML console. As highlighted in the figure above, the console prompt always bears the name of the project which was launched with the console. For this reason, all MoTiVML console commands that are run will be affiliated with the model highlighted in the console prompt.

3.7 MoTiVML Console Commands

- **show** < *parameter* >: The show command is used to display graphically a user constructed model. This command prints the hierarchical structure of a selected model while displaying details such as its mandatory status, group and binding mode. The show command takes a single parameter. This parameter can either be **all** or **config**.

The **show all** command only shows all selected features in the model tree. By default all features are selected when instantiated. Thus, this command shows every single feature in the model tree.

Figure 4: show all Command

```
MoTiVML:[simplebot]>> show all

|---- SimpleBot--{id: root_feature}
  |m--s--o ComponentControl--{id: compcontrol}
    |m--s--o Localisation--{id: localztn}
      |!m--s--x AMCL--{id: amcl}
        |m--s--o AmclRos--{id: amclros}
          |!m--s--x SLAM--{id: slam}
            |!m--D--x HectorSLAM--{id: hectorslam}
              |!m--D--x KartoSLAM--{id: kartoslam}
                |!m--D--x OrbSLAM--{id: orbslam}
          |!m--s--o GestureRecognition--{id: gestrec}
            |!m--D--x Hands--{id: hands}
              |!m--D--x PointsCloud--{id: pointscld}
        |m--s--o MotionPlanningAndControl--{id: motplannctrl}
          |!m--D--x Quick--{id: quick}
            |!m--D--x Optimised--{id: optimised}

Notation:[ m = Mandatory, !m = Optional, o = OR, x = XOR, s = Static, D = Dynamic]
```

The **show config** command only shows currently selected or bound features. That is, unselected or unbound features in the model are excluded from the output tree.

Figure 5: show config Command

```
MoTiVML:[simplebot]>> show config

|m--s--o ComponentControl--{id: compcontrol}
  |m--s--o Localisation--{id: localztn}
    |!m--s--x AMCL--{id: amcl}
      |m--s--o AmclRos--{id: amclros}
        |!m--s--x SLAM--{id: slam}
          |!m--s--o GestureRecognition--{id: gestrec}
            |m--s--o MotionPlanningAndControl--{id: motplannctrl}

Notation:[ m = Mandatory, !m = Optional, o = OR, x = XOR, s = Static, D = Dynamic]
```

- **exit:** The exit command shuts down the console interface and returns back to the original ROS console. The exit command also asks for a yes or no confirmation before proceeding. Figure 6 shows an exit command execution and output.

Figure 6: Exit Command

```
MoTiVML:[simplebot]>> exit
MoTiVML:[simplebot]>> Confirm exit (y/n): y
```

References

- [1] "ROS Pluginlib Documentation," [Online] Available: <http://wiki.ros.org/pluginlib>.