ROSMOD Documentation

Release 0.3

Pranav Srinivas Kumar

May 22, 2015

1 Table of Contents **3**
- 1.1 Introduction . . . 3
- 1.2 Modeling Language . . . 5
- 1.3 Graphical User Interface . . . 7
- 1.4 Code Generators . . . 7
- 1.5 Deployment Infrastructure . . . 7
- 1.6 Logging Framework . . . 7
- 1.7 Tutorial . . . 7
- 1.8 Library Reference . . . 7

2 Indices and tables **13**

Index **15**

Welcome! This is the documentation for ROSMOD version 3.0

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Introduction

ROSMOD is a tool suite for rapid prototyping component-based software applications using the Robot Operating System (ROS). Using ROSMOD, an application developers can create and manage *projects* for distributed real-time embedded systems. Each ROSMOD Project consists of *models* that represent the structure and behavior of the system:

- Software Model : One or more ROS packages in the workspace.
- Hardware Model: One or more Hardware devices.
- Deployment Model: A Mapping between ROS nodes/processes & Hardware devices.

Using these models, ROSMOD can:

- Generate a skeleton ROS workspace, including build system files.
- Preserve already generated work-in-progress ROS workspaces using code-preservation markers.
- Generate deployment-specific XML files for ROS node lifecycle management.
- Generate timing analysis models from abstract business logic specification.
- Perform network analysis to admit/reject applications based on network profiles.

ROSMOD significantly improves the time taken to prototype ROS packages since much of the *boring* skeleton code e.g. port & timer initialization, callbacks for timers, servers & subscribers etc. are automatically generated from the Software Model. Once generated, a developer need only add the *"business logic"* of the generated callbacks to complete the package. A detailed tutorial for this workflow can be found *here*.

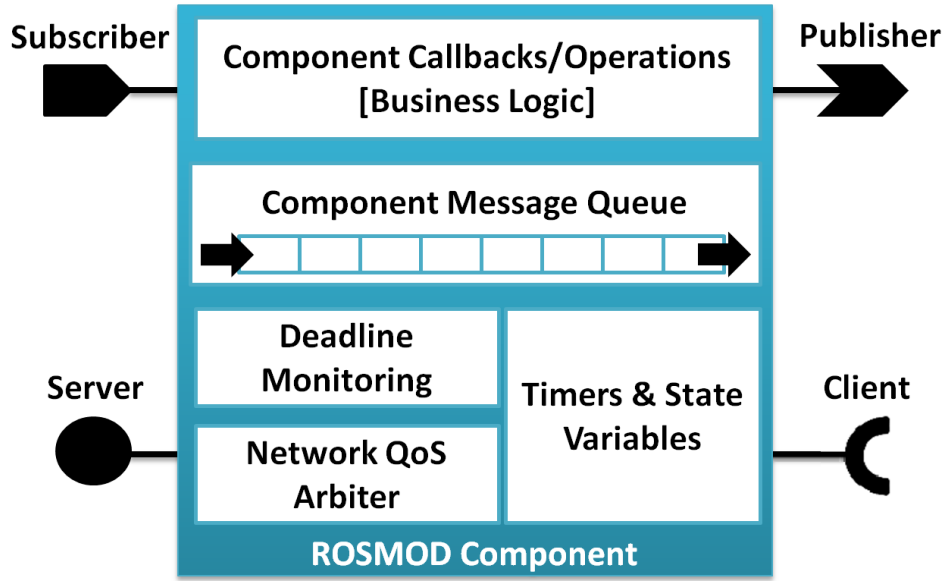
1.1.1 Component-based Software Development

Software development using ROSMOD is inspired by the principles of Component-based Software Engineering. Design and implementation of component-based software applications rests on the principle of assembly: *Complex systems are built by composing re-useable interacting components*. Components contain functional, business-logic code that implements operations/callbacks on state variables. Ports facilitate interactions between communicating components. A component-level message queue controls the scheduling of operations/callbacks.

ROSMOD Components

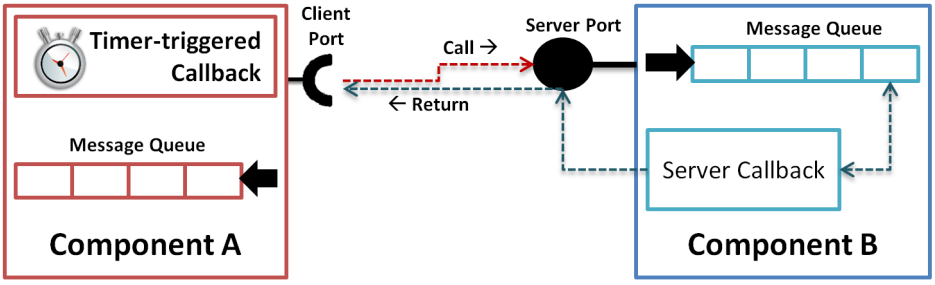
A ROSMOD Component is a re-useable unit/piece of software in an application. Components can be thought of as LEGO pieces - Each piece has a well-defined shape; Multiple such pieces are connected together to build large structures (applications). Component-based software divides application concerns into smaller, manageable blocks that can be easily composed. In ROSMOD, each component can contain one or more of the following:

- *Publishers*: A publisher port publishes (without blocking) on a message/topic (msg in ROS)
- *Subscribers*: A subscriber port subscribes to a message/topic (msg in ROS)
- *Servers*: A server port provides an "operation" (srv in ROS) to the external world
- *Clients*: A client port requires/uses an operation (srv in ROS) provided by a server (potentially on another component)
- *Timers*: A timer is used to trigger the component. Timer callbacks are invoked when a timer expires



Each component has a single thread called the "Component Executor Thread". This thread handles all requests from external entities (other components) and infrastructural triggers (timer expiry). This thread is therefore responsible for executing all triggered callbacks e.g. subscriber callbacks, server callbacks & timer callbacks. To facilitate interactions with other components, each component also has a "Component Message Queue". This queue *holds* requests received from other interacting entities.

The following figure shows a simple Client-Server component interaction. Component A is periodically triggered by a timer. At each timer expiry, Component A makes a blocking remote procedure call to Component B using its client port. This service request, on reaching Component B, is enqueued onto Component B's message queue. When this request reaches the front of the queue, the corresponding server-side callback is executed by the Component B executor thread and the response is returned back to Component A. This message queue-based interaction is also true for timers; when the timer in Component A expires, a timer callback request is enqueued onto its message queue and eventually handled.



Design Notes:

- In each component, the message queue is processed by a single executor thread. Multiple components can run concurrently but each component execution is single-threaded.
- The component message queue supports several scheduling schemes including FIFO (first-in first-out), PFIFO (priority first-in first-out) and EDF (earliest deadline first).
- Requests in the message queue are processed using a non-preemptive scheduling scheme. This means that each callback/operation run by the executor thread is run to completion before the next one (in the message queue) is processed.
- These rules are strictly applied to all ROSMOD components.

ROSMOD Nodes

ROSMOD Nodes are deployable processes that run the application/package. These Nodes contain one or more *instances* of ROSMOD Components. Therefore, each ROSMOD node contains one or more component threads, each of which (1) uses component ports to push out queries, (2) uses its message queue to receive external requests, and (3) executes triggered callbacks. ROSMOD Nodes are similar to ROS Nodes but follow a strict structure and behavior.

1.2 Modeling Language

The ROSMOD Modeling Language is both textual and graphical. The tool suite provides a Graphical User Interface to build these models but the state and configuration properties of the ROSMOD Project are saved in a set of text files (*models*) that follow a strict set of grammatical rules. The grammatical rules were written using Antlr 4 and the models are loaded using the generated parser, lexer and listener classes. This section briefly describes the ROSMOD Textual Modeling Language, along with sample model files.

1.2.1 Software Model

The Software model completely describes a ROS Workspace.

As shown in the above class diagram, every ROS workspace contains one or more ROS packages. Each ROS package contains one or more (1) messages, (2) services and (3) components. In ROSMOD, such packages are built using the Software Modeling Language.

Messages

Messages are constructed identical to the textual language used in .msg files by ROS. Here is a sample msg file:

```
int64 sat_id
float64 inclination
float64 longitudeAN
float64 argumentOfPeriapsis
float64 eccentricity
float64 semimajorAxis
float64 meanAnomaly
```

Services

Services are also constructed identical to the textual langugage used in .srv files by ROS. Here is a sample srv file:

```
int64 thruster_id
float64 amount
float64 duration
---
int64 return_value
```

Components

```
/*
 * ROSMOD Software Model
 */

// ROSMOD Package - three_component_example
package three_component_example
{
  // ROSMOD Component - Component_1
  component Component_1 : Base
  {
    // ROSMOD Publisher - Name_Publisher
    publisher <three_component_example/ComponentName> Name_Publisher;
    // ROSMOD Subscriber - Name_Subscriber
    subscriber <three_component_example/ComponentName> Name_Subscriber
    {
      priority = 50;
      deadline = 0.300;
    }
    // ROSMOD Timer - Timer_1
    timer Timer_1
    {
       period = 0.5;
       priority = 50;
       deadline = 0.200;
    }
  }

  // ROSMOD Component - Component_2
  component Component_2 : Base
  {
    // ROSMOD Server - Service_Server
    server <three_component_example/ComponentService> Service_Server
    {
      priority = 50;
      deadline = 0.500;
    }
    // ROSMOD Publisher - Name_Publisher
    publisher <three_component_example/ComponentName> Name_Publisher;
    // ROSMOD Timer - Timer_2
    timer Timer_2
    {
       period = 1.0;
       priority = 50;
       deadline = 0.200;
    }
```

```
  }

  // ROSMOD Component - Component_3
  component Component_3 : Base
  {
    // ROSMOD Client - Service_Client
    client <three_component_example/ComponentService> Service_Client;
    // ROSMOD Timer - Timer_3
    timer Timer_3
    {
       period = 2.0;
       priority = 50;
       deadline = 0.100;
    }
  }
}
```

1.2.2 Hardware Model

1.2.3 Deployment Model

1.3 Graphical User Interface

1.4 Code Generators

1.5 Deployment Infrastructure

1.6 Logging Framework

1.7 Tutorial

1.8 Library Reference

1.8.1 Grammar Metaclass

Grammar_MetaClass

class **Grammar_MetaClass** (*type*)

Metaclass for generating on-the-fly ANTLR 4 grammar-specific listener functions to parse model files (*.rml | *.rhw | *.rdp)

__new__ (*name*, *bases*, *attrs*)

Intercept ROSMOD Object creation and add attributes to the objects based on model parsing

Parameters

- **name** – Name of the to-be-created class
- **bases** – List of all base classes of the to-be-created class

- **attrs** – Dictionary of all attributes in the to-be-created class

1.8.2 Project Classes

ROSMOD Project

class `ROSMOD_Project` (*Drawable_Object*)
The high-level interface to a ROSMOD Project

`__init__` (***kwargs*)
Initialize an empty ROSMOD Project.

`new` (*project_name*, *project_path*, *workspace_name*, *hardware_name*, *deployment_name*[, *q=None*])
Create a new ROSMOD Project.

Parameters

- **project_name** (*str*) – The name of the ROSMOD Project
- **project_path** (*str*) – The absolute path to the root directory of the ROSMOD Project
- **workspace_name** (*str*) – The name of the ROSMOD Software Model and the generated ROS Workspace
- **hardware_name** (*str*) – The name of the ROSMOD Hardware Model
- **deployment_name** (*str*) – The name of the ROSMOD Deployment Model

`open` (*project_path*[, *progressQ = None*])
Open an existing ROSMOD Project

Parameters **project_Path** (*str*) – The absolute path to the root directory of the ROSMOD Project

`parse_msg` (*dirname*)
Parse all msg (message) files in the Software Model

Parameters **dirname** (*str*) – The path to the msg directory in the ROSMOD Software Model

`parse_srv` (*dirname*)
Parse all srv (service) files in the Software Model

Parameters **dirname** (*str*) – The path to the srv directory in the ROSMOD Software Model

`parse_abl` (*dirname*)
Parse all abl (abstract business logic) files in the Software Model

Parameters **dirname** (*str*) – The path to the abl directory in the ROSMOD Software Model

`parse_pnp` (*dirname*)
Parse all pnp (port network profiles) files in the Software Model

Parameters **dirname** (*str*) – The path to the pnp directory in the ROSMOD Software Model

`parse_snp` (*dirname*)
Parse all snp (system network profiles) files in the Hardware Model

Parameters **dirname** (*str*) – The path to the snp directory in the ROSMOD Hardware Model

`parse_rml` (*filename*)
Parse the provided rml file (ROSMOD Software Model)

Parameters **filename** (*str*) – The name of the .rml file

parse_rhw (*filename*)
Parse the provided rhw file (ROSMOD Hardware Model)

Parameters **filename** (*str*) – The name of the .rhw file

parse_rdp (*filename*)
Parse the provided rdp file (ROSMOD Deployment Model)

Parameters **filename** (*str*) – The name of the .rdp file

parse_models ([*progressQ=None*])
Parse all Software, Hardware and Deployment models in the ROSMOD Project

check_workspace ()
Check for an existing ROS Workspace in the ROSMOD Project in order to preserve any code between code-preservation markers

generate_workspace ()
Generate a ROS Workspace from the Software Model. The workspace is generated at PROJECT_ROOT/01-Software/<workspace_name>

generate_xml ()
Generate Deployment XML files for all ROS nodes in the Deployment Model

generate_cpn ()
Generate a Colored Petri Net timing analysis model for the ROSMOD Project

resolve_references ([*progressQ = None*])
Resolve all missing object references after parsing models

save_rml ([*path=""*])
Save a modified workspace (.rml) file

Parameters **path** (*str*) – Optional path where the .rml file will be saved

save_rhw ([*path=""*])
Save a modified workspace (.rhw) file

Parameters **path** (*str*) – Optional path where the .rhw file will be saved

save_rdp ([*path=""*])
Save a modified workspace (.rdp) file

Parameters **path** (*str*) – Optional path where the .rdp file will be saved

save_msg (*msg_object*)
Save a modified msg object to file

Parameters **msg_object** – Message Object to be saved

save_srv (*srv_object*)
Save a modified srv object to file

Parameters **srv_object** – Service Object to be saved

save_abl (*port_object*)
Save a modified abstract business logic to file

Parameters **port_object** – Port Object that contains an abstract business logic

save_pnp (*port_object*)
Save a modified port network profile to file

Parameters **port_object** – Port Object that contains a port network profile

save_snp (*port_object*)
Save a modified system network profile to file

Parameters **port_object** – Port Object that contains a system network profile

save ([*project_name=""*][, *project_path=""*])
Used to "Save" or "Save-As" an entire ROSMOD Project

Parameters

- **project_name** (*str*) – Name of ROSMOD Project used for Save-As
- **project_path** (*str*) – Absolute path to ROSMOD Project used for Save-As

ROSMOD_Generator

class ROSMOD_Generator
Primary Generator class for ROSMOD

generate_workspace (*workspace*, *path*)
Generate the ROS Workspace and build system from the Software Model

Parameters

- **workspace** – The ROS_Workspace object to traverse
- **path** (*str*) – The absolute path to the to-be-generated workspace directory

generate_xml (*deployments*, *deployment_path*)
Generate Deployment-specific ROS Node XML files from the Deployment Model

Parameters

- **deployments** – The list of all ROS_Deployment objects in 03-Deployment directory
- **deployment_path** – The absolute path to the Deployment directory

generate_cpn (*workspace*, *deployments*, *deployment_path*)
Generate a Colored Petri Net-based Timing Analysis Model for the Project

Parameters

- **workspace** – The ROS_Workspace object to traverse
- **deployments** – The list of all ROS_Deployment objects in 03-Deployment directory
- **deployment_path** – The absolute path to the Deployment directory

ROSMOD_Loader

class ROSMOD_Loader
Loads an existing ROS Workspace for Code Preservation

load (*workspace*, *path*)
Load an existing ROS workspace and preserve code blocks between code-preservation markers so that regeneration does not completely overwrite existing work

Parameters

- **workspace** – The ROS_Workspace object to traverse
- **path** (*str*) – The absolute path to the ROS workspace directory

ROSMOD_Software_Builder

class `ROSMOD_Software_Builder` (*ROSMOD_SoftwareListener*)
Builds a Software Object Tree from parsing the .rml file in 01-Software/*

Variables **__metaclass__** – *Grammar_MetaClass*

`__init__` (*project*)
Initialize the root of the Software Object Tree, a "ROS_RML" class object

Parameters **project** – The ROSMOD_Project object that is the parent of the Software model

ROSMOD_Hardware_Builder

class `ROSMOD_Hardware_Builder`
Builds a Hardware Object Tree from parsing the .rhw files in 02-Hardware/*

Variables **__metaclass__** – *Grammar_MetaClass*

`__init__` (*project*)
Initialize the root of the Hardware Object Tree, a "ROS_RHW" class object

Parameters **project** – The ROSMOD_Project object that is the parent of the Software model

ROSMOD_Deployment_Builder

class `ROSMOD_Deployment_Builder`
Builds a Deployment Object Tree from parsing the .rdp files in 03-Deployment/

Variables **__metaclass__** – *Grammar_MetaClass*

`__init__` (*project*)
Initialize the root of the Deployment Object Tree, a "ROS_RDP" class object

Parameters **project** – The ROSMOD_Project object that is the parent of the Software model

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

__init__() (ROSMOD_Deployment_Builder method), 11
__init__() (ROSMOD_Hardware_Builder method), 11
__init__() (ROSMOD_Project method), 8
__init__() (ROSMOD_Software_Builder method), 11
__new__() (Grammar_MetaClass method), 7

check_workspace() (ROSMOD_Project method), 9

generate_cpn() (ROSMOD_Generator method), 10
generate_cpn() (ROSMOD_Project method), 9
generate_workspace() (ROSMOD_Generator method), 10
generate_workspace() (ROSMOD_Project method), 9
generate_xml() (ROSMOD_Generator method), 10
generate_xml() (ROSMOD_Project method), 9
Grammar_MetaClass (built-in class), 7

load() (ROSMOD_Loader method), 10

new() (ROSMOD_Project method), 8

open() (ROSMOD_Project method), 8

parse_abl() (ROSMOD_Project method), 8
parse_models() (ROSMOD_Project method), 9
parse_msg() (ROSMOD_Project method), 8
parse_pnp() (ROSMOD_Project method), 8
parse_rdp() (ROSMOD_Project method), 9
parse_rhw() (ROSMOD_Project method), 8
parse_rml() (ROSMOD_Project method), 8
parse_snp() (ROSMOD_Project method), 8
parse_srv() (ROSMOD_Project method), 8

resolve_references() (ROSMOD_Project method), 9
ROSMOD_Deployment_Builder (built-in class), 11
ROSMOD_Generator (built-in class), 10
ROSMOD_Hardware_Builder (built-in class), 11
ROSMOD_Loader (built-in class), 10
ROSMOD_Project (built-in class), 8
ROSMOD_Software_Builder (built-in class), 11

save() (ROSMOD_Project method), 10
save_abl() (ROSMOD_Project method), 9
save_msg() (ROSMOD_Project method), 9
save_pnp() (ROSMOD_Project method), 9
save_rdp() (ROSMOD_Project method), 9
save_rhw() (ROSMOD_Project method), 9
save_rml() (ROSMOD_Project method), 9
save_snp() (ROSMOD_Project method), 9
save_srv() (ROSMOD_Project method), 9