**Turtlebot Project**

**Framework Design Document**

Version 0.0.07

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 03-17-2017 | 0.0.01 | First High Level Framework Design | Aditya Mathur |
| 04-15-2017 | 0.0.02 | High Level Framework Design | Aditya Mathur |
| 06-15-2017 | 0.0.03 | Detailed Design | Aditya Mathur |
| 06-25-2017 | 0.0.04 | Updated Detailed Design – Using a service for nodes to enter/exit reconfiguration state | Aditya Mathur |
| 07-07-2017 | 0.0.05 | Updated Detailed Design | Aditya Mathur |
| 08-02-2017 | 0.0.06 | Update API provided by the framework and SCN structure parts | Yunpeng Xu |
| 08-03-2017 | 0.0.07 | Minor Updates | Aditya Mathur |

**Formal Approval**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Issues** | **Approved by** |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1. Introduction 4](#_Toc487971404)

[1.1 Purpose 4](#_Toc487971405)

[1.2 Definitions, Acronyms, Abbreviations 4](#_Toc487971406)

[2. Architecture 5](#_Toc487971407)

[2.1 System Control Library 5](#_Toc487971408)

[2.1.1 Application Programing Interface (API) offered by SCN Library 7](#_Toc487971409)

[2.2 System Configuration Node (SCN) 8](#_Toc487971410)

[2.2.1 SCN Structure 9](#_Toc487971411)

[2.2.2 SCN Operation 9](#_Toc487971412)

[2.3 Reconfiguration Activity 13](#_Toc487971413)

[2.4 Nodes in the system 15](#_Toc487971414)

[3. Appendix 15](#_Toc487971415)

[Reconfiguration State 15](#_Toc487971416)

# **Introduction**

The Turtle-bot project aims at building a framework to support configuration/reconfiguration and monitoring for mobile robots at run time.

## Purpose

The purpose of this document is to provide an overview of the design for a reconfiguration framework for mobile robots operating on ROS.

## Definitions, Acronyms, Abbreviations

|  |  |  |
| --- | --- | --- |
| **DAA** | **Term** | **Definition** |
| ROS | Robot Operating System |  |
| Bot | Robot |  |
| TBD | To Be Discussed |  |
| SCN | System Configuration Node |  |
| API | Application Programing Interface |  |

# **Architecture**

There are three key elements in the architecture of the System Configuration Framework-

1. **System Control Library** – This is the static library which provides interfaces to the application developers who need to use the framework.
2. **System Configuration Node** – This is the dynamic component of the system that shall be responsible for managing system states related to reconfiguration activities.
3. **Reconfiguration State** – This a new system state that shall be define for all the nodes that would like to use the framework.

All these elements have been described in detail below.

## System Control Library

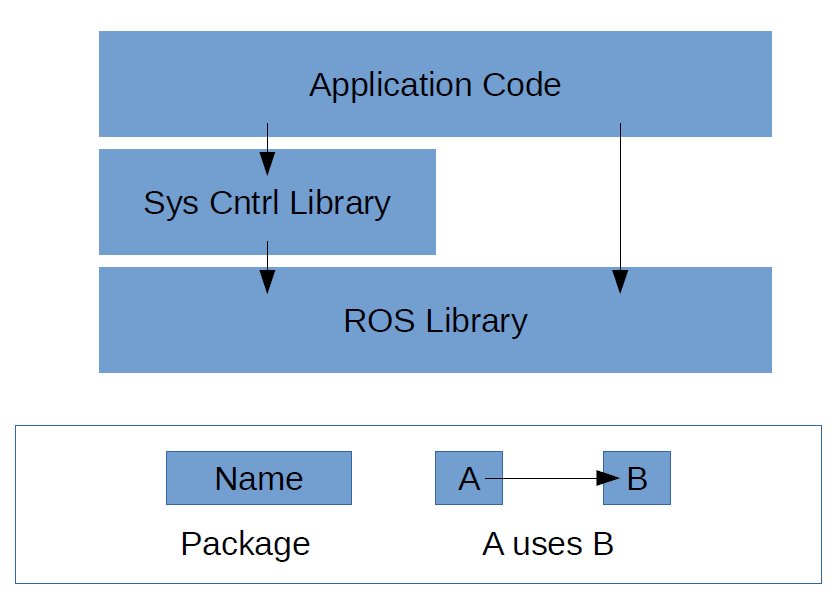


Figure 1: Static View of the System Configuration Node design (Using a framework library)

The System Control Library provides a set of APIs’ that wrap communication (topic, service, action) API’s provided by ROS. A list of APIs’ provided by this library can be found in the API documentation.

The aim of the library is to provide wrappers to the ROS communication API’s such that the framework can register the respective communication dependency with the SCN. The SCN library API’s register the dependencies with the SCN when the node uses any communication medium for the first time. On the other consecutive interactions, the API’s check the state of the system and then perform the respective actions.

The library provides under the hood functionality during the initialization, registration and reconfiguration of a node. Figure 2 describes the actions performed when a node initializes itself.

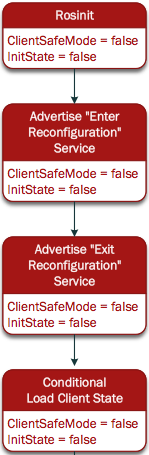


Figure 2: Sub-states within the “Init” state of a client/subscriber node

The initialization API performs the ROS init for the node, and then advertises services for the SCN to communicate with the node. Once advertised, it is checked if there is a need to load some state from a previously stored state on the disk.

In the initialization phase, the library initialization function checks for the existence of the SCN in the system. If it identifies that the SCN is not present at the point of its initialization, the library initialization function will spawn the SCN and will wait for SCN to start up and then move forward with other initialization activities.

The library also describes the behavior of a node when an enter/exit reconfiguration state message is received at the node. Figure 3 describes the state of a node when a node receives an enter reconfiguration state message from the SCN.

The framework calls a save state function that enables the user (application developer) to save the state of the node, if any, before entering the reconfiguration state. This is important because the framework enforces a stop on all communication (service calls, topic publishing, action requests) in the reconfiguration state. After saving the state of the node the framework enables the user to enter a user defined Reconfiguration state operation. The user (application developer) can perform any operation in this state, but there will be no communication allowed between the nodes in the framework.



Figure 3: Sub-states within the “Reconfiguration” state of a client/subscriber node

## Application Programing Interface (API) offered by SCN Library

1. **scn\_init():** Wraps init(). The scn\_init wrapper function registers three callback functions for the reconfiguration framework: saveStateRoutine, reconModeRoutine, and loadStateRoutine. saveStateRoutine will be triggered by the reconfigure node to save the state of the node. reconModeRoutine will be triggered by the reconfigure node to tell a node to enter/exit the reconfiguration mode. And loadStateRoutine is used to load the state of the node if some errors happen and the system needs to restore some states.
2. **scn\_advertise()**: Wraps advertise(). Communicates the identity of the node and the name of the topic that is being advertised, to the SCN. Calls the actual ros::topic::advertise() after communicating with the SCN. This is a blocking call, the call does not return until the registration with the SCN is successful.

It will keep trying to register the dependency until the SCN returns with a success message.

1. **scn\_subscribe():** Wraps the subscribe() call that tells ROS that the node wants to subscribe to the topic. Communicates the identity of the node and the name of the topic that the node is subscribing to, to the SCN. Calls the actual ros::topic::subscribe() after communicating with the SCN. This is a blocking call; the call does not return until the registration with the SCN is successful.
2. **scn\_advertiseService**(): Wraps advertiseService(). It communicates the name of the node and the service that is being advertised, to the SCN. On successful registration with the SCN, ROS advertiseService is called.

This is a blocking call, it blocks till the registration with SCN is successful.

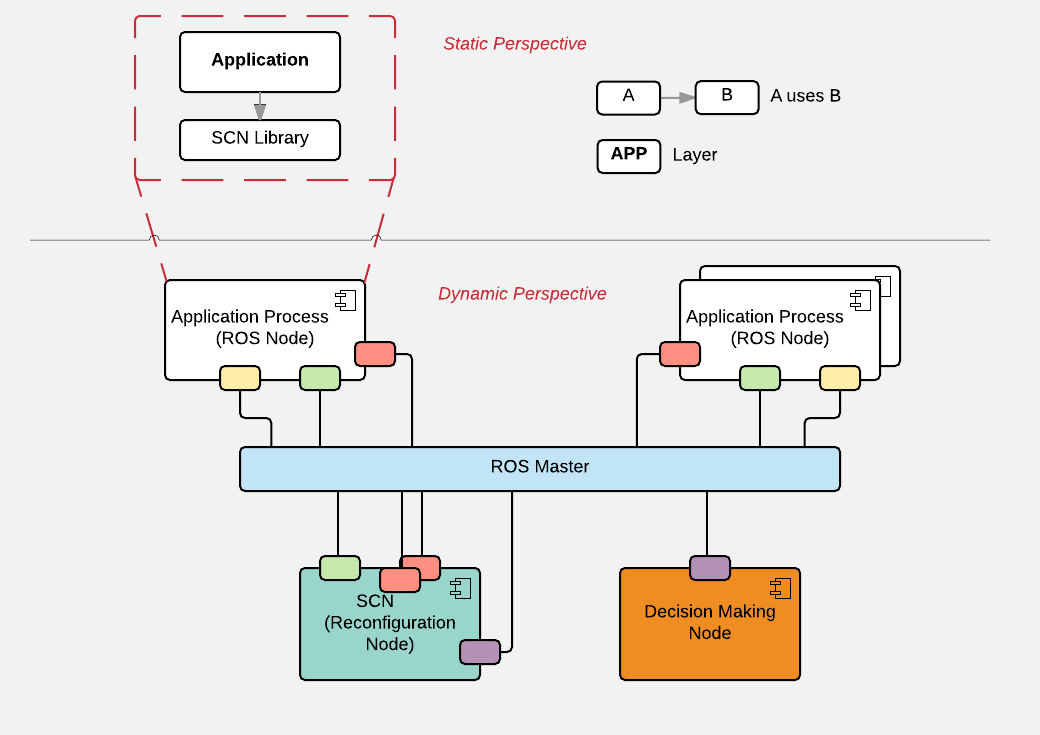
1. **scn\_serviceClient():** Wraps serviceClient(). It communicates the name of the node and the service that the node is going to use, to the SCN. This is a blocking call, it blocks till the registration with SCN is successful.

## System Configuration Node (SCN)

The “System Configuration Node” element is an artifact in the dynamic domain. The main idea in this design is to store the dependencies between various nodes in the system on the System Configuration Node such that the SCN can base the reconfiguration decisions during the reconfiguration activity in the system.

ROS provides a list of dependencies based on ROS topics available in the system. The real challenge in this project is identifying ROS nodes interacting among each other via mechanisms other than topics (services and actions). Thus, the intention behind having a System Configuration Node in this design is to maintain the dependencies between the various nodes in the SCN, which would include dependencies based on all the three communication mechanisms (topics, services and actions).

Based on the list of dependencies, the System Configuration Node would identify the nodes that will be effected by a particular reconfiguration activity. On the basis of the type of reconfiguration, the System Configuration Node will put the respective affected nodes in a “reconfiguration state” for the duration of the reconfiguration activity.



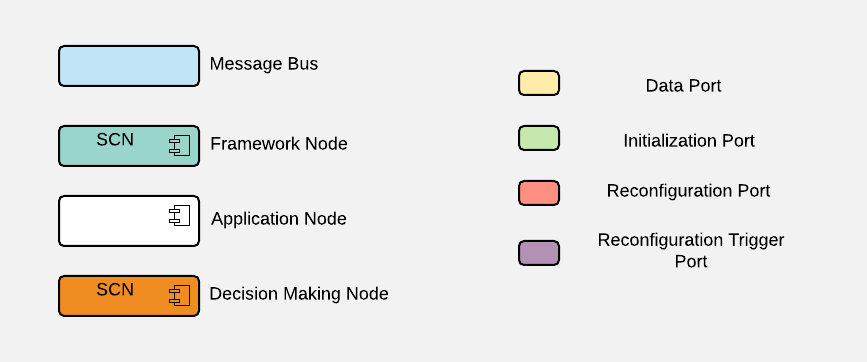


Figure 4: Dynamic View of the System Configuration Node design

## SCN Structure

1. SCN is the first node (within the scope of the reconfiguration framework) to start in the system
2. SCN provides the following communication mechanisms:
   1. **SystemControlRegisterService**: A service offered by SCN via which each node in the scope of the framework will communicate its respective dependencies to the SCN (see Section for dependency registration mechanism).
   2. **FrameworkInfoService**: A service offered by SCN via which node developer can query the information of the reconfiguration framework, including nodes that’re registered into the framework and the current dependency tree of the framework.
   3. **UserInterfaceService**: A service offered by SCN to the adaptation layer (rainbow) to communicate the reconfiguration and its details to the SCN for the SCN to initiate and perform the same.
   4. **scnDebug**: SCN will publish debug/information on this topic. Only for user interaction purpose.

SCN maintains a data structure of dependencies between all the nodes registered to the framework. Each node communicates its dependencies as and when it starts using the respective dependency i.e. when it initializes the dependency (advertise a service, advertise an action, register a topic etc.).

## SCN Operation

* + - 1. **SCN States**



Figure 5: States of the System Configuration Node

SCN will typically be in either of the six states listed as follows at any given point of time:

1. **Does Not Exist**: The SCN is not present in the system. SCN is the heart of the framework. There will be no notion of the framework in the system (robot environment) in this state.
2. **Initialization State**: The state where in the SCN will perform initialization activity for itself. Essentially, SCN will register itself with ROS core and advertise the services as mentioned in [Section 2.2.1](#_SCN_Structure).
3. **Normal Idle State**: The SCN and the system is in normal operation and all nodes are operating as per their respective operational logic. Any node can register its dependencies with the SCN and thus the framework, while the SCN is in this state. SCN will accept reconfiguration requests from the adaptation layer only when it is in this state.
4. **Reconfiguration State**: The SCN will be in this state when there is a reconfiguration activity under process in the system. All requests to SCN to store dependencies i.e. to register with the framework will be denied (NACK). **An exception to the denial of registration in this state is – the case when a new node is spawned by the SCN as a part of the reconfiguration activity.**
5. **Processing State:** The SCN will be in this state when it receives a request from any other node in the system to register their dependency with the SCN. The SCN will only register one dependency at a time.
6. **Save State:** This state is for the SCN to make a copy of the dependencies it has dynamically accumulated, based on the registrations that SCN has received. SCN will make and store a copy on the disk when in this state.

Once successfully initialized, the state of SCN defaults to Normal Operation State when there is no reconfiguration activity under process.

* + - 1. **SCN Functional Logic**

For using the reconfiguration framework it is required that the SCN be started before any node that wishes to be a part of this framework. Figure 4 shows the initialization sequence of the SCN. The SCN initializes itself with ROS and checks if there is a need for loading previously saved dependency data from the disk. SCN loads data from the disk in cases where SCN starts after a crash. The SCN then advertises the services that it offers (systemControlRegisterService, userInterfaceService) and then moves into Normal Idle state.



Figure 6: Sub-states within the “Init” state of SCN node

The adaptation layer will communicate the reconfiguration activity to the SCN over the [**UserInterfaceService**](#_SCN_Structure_1)**.** On reception of a reconfiguration request (Figure 5), SCN switches its state to “reconfiguration state” and identifies the nodes that will be involved in the respective reconfiguration activity. SCN requests all the identified nodes in the order of their dependencies to enter their respective “Reconfiguration States” by making individual service calls to each of the identified nodes. The order of identifying the hierarchy followed by SCN is -

1. **Services**: Identify the service offered by the node that is subjected to reconfiguration. For each identified service, identify the client nodes and request those client nodes to enter reconfiguration state. Once all client nodes have entered reconfiguration state, the node providing the respective service is requested to enter reconfiguration state.
2. **Actions**: Identify the actions serviced by the node that is subjected to reconfiguration. For each identified action, identify the client nodes and request those client nodes to enter reconfiguration state. Once all client nodes have entered reconfiguration state, the node providing the respective action service is requested to enter reconfiguration state
3. **Topics**: Identify the topics to which the node that is subjected to reconfiguration publishes. For each identified topic, identify the subscriber nodes and request those subscriber nodes to enter reconfiguration state.

In case of a chained dependency, SCN will regard the chain of dependencies for enforcing a “reconfiguration state” on the nodes.



Figure 7: Sub-states within the “Reconfiguration” state of SCN node

Once all nodes enter reconfiguration state (get an acknowledgement from all the nodes), SCN invokes the reconfiguration activity. (see section 2.3)

Once the reconfiguration activity successfully completes, SCN starts the process of requesting all involved nodes to switch back to “normal state”. The hierarchy for this action is the opposite of the one which was identified for putting nodes into “reconfiguration state”. Once all the nodes switch to “normal state”, the SCN will switch to “normal state” and will broadcast normal state on the degug/information topic.

SCN discards all registration requests (NACK) while it is in “reconfiguration state”.

(\*An exception to the denial of registration in this state is – the case when a new node is spawned by the SCN as a part of the reconfiguration activity.)

* + - 1. **SCN Error Scenarios**

The possible error scenarios that the SCN may encounter are as follows:

1. SCN requests a node to enter Reconfiguration mode and the node does not return the service call.
2. SCN requests a node to exit Reconfiguration mode and the node does not return the service call.

For both the scenarios listed above, the SCN employees a timeout mechanism. Before each call, SCN starts a timer of 60 seconds and expects a response from the pier node within that period of time, if the pier node fails to respond within the 60 seconds, SCN terminates that respective service call and performs a retry for the same node. If the node fails to reply the second time (each request has a 60 second timeout), SCN performs a roll back for the system and reports this incident to the adaptation layer.

## Reconfiguration Activity

**Swapping nodes**

Figure 8 depicts the sequence of events that occur when a Node level reconfiguration is triggered by the user (user of the framework). Typically, the user communicates the reconfiguration details to the SCN. The SCN interprets the reconfiguration details and uses its dependency tree to identify the nodes that will be effected by this reconfiguration activity. The SCN uses the reconfiguration port to request all the nodes that will be effected to enter a reconfiguration state. This reconfiguration message is received on the respective nodes and these nodes trigger the reconfiguration callback, that was installed at the time of initialization, so as to put the node in to reconfiguration state. Once the node switches into the reconfiguration state, it responds back to the SCN with an acknowledgement message (response to the service).

After the SCN receives a positive acknowledgement from all the nodes (all the nodes that are effected by the reconfiguration), SCN kills the node to be replaced (Node B) and then spawns the new node (Node X) that is to replace the existing node (Node B). Node X follows the same initialization process as explained earlier.

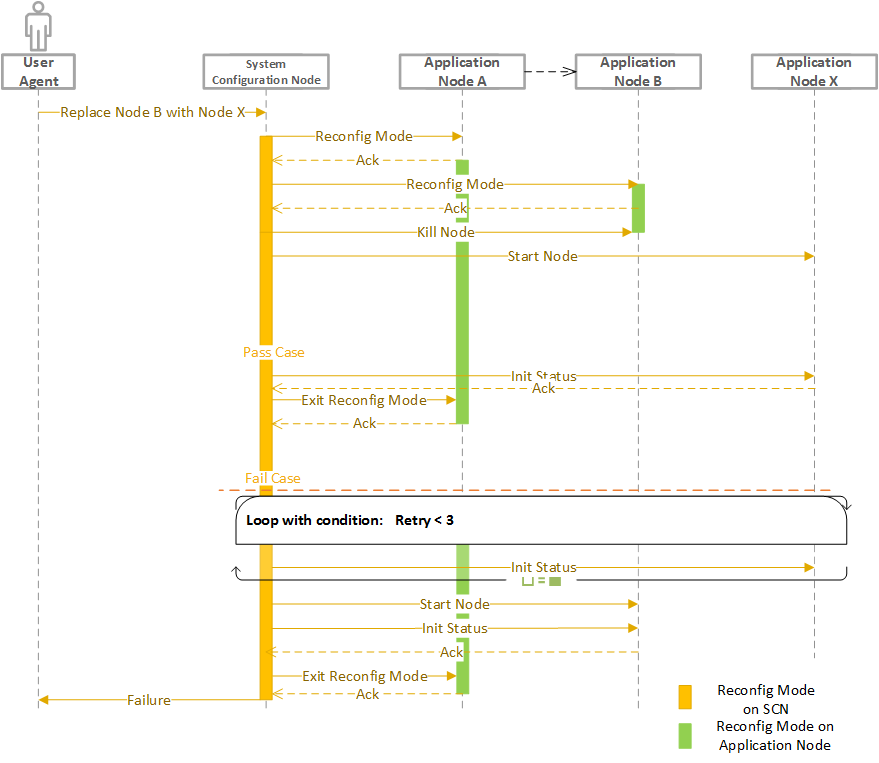


Figure 8. Reconfiguration Sequence

Once the new node has successfully started the SCN responds with a “success” message to the new node (Node X) and orders all other to revoke their reconfiguration state and switch to normal operation state.

In the case when the new node does not successfully start up, SCN kill the new node (Node X) and restarts the old node (Node B) with a parameter that specifies that Node B has to load state from the disk (state was saved when Node B received a reconfiguration message).

**Parameter Reconfiguration**

The sequence for a parameter based reconfiguration is the same as that of the “Swapping Nodes”, just that in this case, no node is killed and no new node is started.

## Nodes in the system

All nodes that are a part of the framework will be in either of the following states:

1. **Normal Operation State**: The node will be operating in their normal operation. All communication (topic, service and action) will go through the SCN APIs’ without any blocking.  
   \*Both SCN and the respective node are in normal operation state
2. **Reconfiguration State**: The node will be in a reconfiguration state. The behavior of the nodes in the reconfiguration state is defined by the node application developers with respect to the nature of the respective node. All communication through the APIs’ in this state will go through successfully as demanded by the application node.

\*Both SCN and the respective node are in reconfiguration state.

# **Appendix**

## Reconfiguration State

The framework introduces a concept of “reconfiguration state” for all the nodes on the system that are registered with the SCN. The reconfiguration state is a state that every node puts itself in, when there is some reconfiguration activity being performed by the SCN. A node in the reconfiguration state saves its current state and important data internally and does not produce critical output for other nodes. The activities each node performs in the reconfiguration state are to be decided by the node itself (developer of the node) since the framework is generic for mobile robots and it is hard to maintain states for all the nodes in the system in a centralized format. The SCN is only responsible for instructing the nodes to enter and exit reconfiguration state.