Lab 2 – R-IDE Prototype Specification

Gavin St. Clair

Department of Computer Science Old Dominion University

CS 411 Professional Workforce Development Spring 2023

Professor J. Brunelle

March 20, 2023

First Version

Table of Contents

List of Figures

List of Tables

1. Introduction

Introducing the Prototype Specification, a solution designed to overcome the high barriers of entry for new developers in ROS development. The Prototype Specification addresses the common issues that developers face when working with ROS, such as managing multiple nodes, identifying when changes in one node affect a related node, organizing the interface, and debugging effectively. With the Prototype Specification, developers can streamline their workflow and focus on developing their robotics projects without the frustration of a difficult setup process or outdated documentation. R-IDE eliminates the hassle of managing numerous terminal windows and embraces a more efficient and effective approach to ROS development.

1.1 Purpose

The primary aim of R-IDE is to streamline and accelerate the ROS development lifecycle and learning process. As an extension for Visual Studio Code, R-IDE is designed for use by Dr. Belfore, his students, and other ROS developers. R-IDE provides numerous capabilities to facilitate common tasks and commands, as well as access to visualization tools, improved interface organization, and rapid environment building for any ROS project. In addition, R-IDE includes templates for various tasks, including node creation and launch file building, to simplify and expedite the ROS development process.

1.2 Scope

R-IDE will provide ROS developers with a range of benefits, including automated package management, graphical user interfaces for command-line functions, and pre-built templates for the rapid construction of functional ROS packages. The prototype will serve as a

real-world example, demonstrating the practical application of these features in a working environment. By leveraging intuitive interfaces to facilitate interaction between the user and ROS, the prototype will further ease the development process. To mitigate any potential risks, the development team will closely adhere to software guidelines and updates and continually seek feedback from users to ensure the Prototype of R-IDE remains at the forefront of ROS development solutions.

1.3 Definitions, Acronyms, and Abbreviations

Autonomous Machine: A machine capable of sensing its environment, carrying out computations to make decisions, and performing actions in the real world.

Command Palette: The command palette refers to the input box in VSCode where users can enter commands.

Filtering: Create new ROS bag files from older ROS bag files, filtering the new bag based on requested ROS topics from the old one.

Git: Git is a distributed version control system: tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development.

Graph View: A graphical representation of the communications between different nodes in a ROS system. It provides a visualization of the ROS topics being published and subscribed to by each node. Graphical View is a useful tool for understanding and debugging complex ROS systems.

GUI (Graphical User Interface): GUI is a type of user interface that has elements such as buttons, menus to allow the user to interact with the application.

IDE (Integrated Development Environment): An application software that provides tools and features for software development. They include features like debugger, source code editor, code completion and even version control in some cases.

IntelliSense: IntelliSense is a general term for various code editing features including code completion, parameter info, quick info, and member lists.

Monarch 1: A two-passenger autonomous vehicle design based on Polaris GEM e2, which is an electric vehicle that is intended to participate in the IGVC's "Self-Drive Challenge".

MongoDB: A free non-relational database that stores data in JSON format. The data is stored in collections of documents that can have nested structures for flexible data storage.

Mongoose: A library for Node.js that provides a higher level of abstraction on top of MongoDB that simplifies the process of interacting with MongoDB.

RQT: A QT-based framework for GUI development for ROS. It contains tools that support ROS topics, bags, node graphs, and many other tools for visualization and manipulation of ROS nodes.

ROS Bag: A file format in ROS for storing ROS message data. An important role in ROS, and a variety of tools have been written to allow the user to store, process, analyze, and visualize them. Bags are the primary mechanism in ROS for data logging, which means that they have a variety of offline uses.

ROS Master: The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer. The Master also provides the

Parameter Server and is most run using the roscore command, which loads the ROS Master along with other essential components.

ROS Messages: A message is a simple data structure, comprising typed fields. Nodes communicate with each other by publishing messages to topics. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays. As part of a ROS service call, a message can be exchanged between nodes in the form of request and response.

ROS Node: Ros Nodes are executable programs to perform tasks within the robotic system. Nodes are combined and communicate with one another using streaming topics, RPC services, and the Parameter Server. These nodes are meant to operate at a fine-grained scale; a robot control system will usually comprise many nodes. For example, one node controls a laser rangefinder, one Node controls the robot's wheel motors, one node performs localization, one node performs path planning, one node provides a graphical view of the system, and so on.

ROS Package: Software in ROS is organized by packages, a collection of ROS nodes, a ROS library, a dataset, or anything else that requires a useful module for basic functionality.

ROS Parameter Server: A parameter server is a shared, multivariate dictionary that is accessible via network APIs. Nodes use this server to store and retrieve parameters at runtime. As the server is not designed for high-performance, it is best used for static, non-binary data such as configuration parameters. The server is meant to be globally viewable so that tools can easily inspect the configuration state of the system and modify it if necessary.

ROS Services: ROS Services provide a way for nodes to communicate with each other in a synchronous, request-response fashion. The request / reply is done via a Service, which is defined by a pair of messages: one for the request and one for the reply. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply. Client libraries usually present this interaction to the programmer as if it were a remote procedure call.

ROS Subscriber: A node that receives data from a topic published by another node. Nodes use ROS topics to communicate, and a subscriber subscribes to a topic to receive messages that are published on that topic.

ROS Topics: Named buses over which nodes exchange messages. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption. In general, nodes are not aware of who they are communicating with. Instead, nodes that are interested in data subscribe to the relevant topic; nodes that generate data publish to the relevant topic. There can be multiple publishers and subscribers to a topic. Topics are intended for unidirectional, streaming communication.

Robot Operating System (ROS): A set of software libraries that helps to build robot applications. Ranging from drivers to algorithms and powerful developer tools, ROS is the preferred tool for robotics projects.

Rviz: (Ros Visualization) A 3D visualizer for displaying sensor data and state information from ROS.

Snippets: A coding tool to automatically generate repeating code when developing.

Template: An editable text or code snippet that can be filled with given values from another tool like a wizard.

Topic Monitor: Displays debug information about ROS topics including publishers, subscribers, publishing rate, and ROS messages.

Tutorial: A method of transferring knowledge that teach via example and supplies the information to complete a given task.

Trimming the ROS bag time range: This reduces the duration in time of a requested ROS bag.

VSCode Extension: A tool designed to add additional features and capabilities to VSCode. It can create new dialogues, add functions, or change the appearance of VSCode.

Webview: A type of web browser that can be embedded within applications allowing it to display web content.

Wizard: A user interface that presents dialog to lead a user through a sequence of steps. Often used to configure a service for the first time or to simplify a complex or unfamiliar process.

1.4 References

Ackerman, E. (2021, June 24). Microsoft announces experimental release of ROS for Windows 10. Retrieved November 3, 2022, from https://spectrum.ieee.org/microsoft-announces-experimental-release-of-ros-for-windows-10

Belfore, L. (jul 8, 2022). *Software Design Report for the ODU Monarch I* (Vol. 1.3, p. 3, Tech.).

Cardona, M., & Manzanares, J. (2020). *COVID-19 Pandemic Impact on Mobile Robotics Market* (pp. 1-4) (A. Palma, Ed.). IEEE. doi:10.1109/ANDESCON50619.2020.9272052

Fujita, T. (2018). Tomoya Fujita R&D center Sony Corporation - roscon.ros.org. Retrieved November 3, 2022, from https://roscon.ros.org/2018/presentations/ROSCon2018_Aibo.pdf

Gerkey, B. (2014, September 1). Ros running on ISS. Retrieved November 3, 2022, from https://www.ros.org/news/2014/09/ros-running-on-iss.html

Global Robot Operating System Market Research Report 2022(status and outlook). (2022, June 29). Retrieved November 3, 2022, from https://www.marketreportsworld.com/global-robot-operating-system-market-21185690

Guerry, M., Müller, C., Kraus, W., & Bieller, S. (2021, October 28). IFR International Federation of Robotics. Retrieved November 3, 2022, from https://ifr.org/downloads/press2018/2021_10_28_WR_PK_Presentation_long_version.pdf

Lunar Rover. (2021). Retrieved November 3, 2022, from https://www.openrobotics.org/customer-stories/lunar-rover

Robot Operating System Market. (2022, August). Retrieved November 3, 2022, from https://www.futuremarketinsights.com/reports/robot-operating-system-market

Robot operating system. (n.d.). Retrieved November 3, 2022, from https://www.ros.org/

Wessling, B. (2022, February 11). Open robotics developing space ROS with Blue Origin, NASA. Retrieved November 3, 2022, from https://www.therobotreport.com/open-robotics-developing-space-ros/

1.5 Overview

The remainder of this prototype description will provide a more detailed overview of R-IDE, covering its software configuration, external interfaces, and key features. It will also include an in-depth exploration of the architecture of R-IDE, outlining its underlying design principles and technical capabilities. Additionally, the description will provide a comprehensive overview of the features and capabilities of R-IDE, highlighting its unique selling points and demonstrating how it can streamline the ROS development process. Throughout this document, the focus will be on presenting a detailed and comprehensive overview of R-IDE, highlighting its potential benefits, and demonstrating how it can support ROS developers in creating innovative robotics projects.

2. General Description

R-IDE provides users with a creation wizard that can create new functional ROS nodes, service files, and message files. Additionally, R-IDE includes the ability to play previously recorded ROS bags from Dr. Belfore, providing users with the ability to test and refine their projects with real-world data.

Figure 1

R-IDE ROS Bag Manager

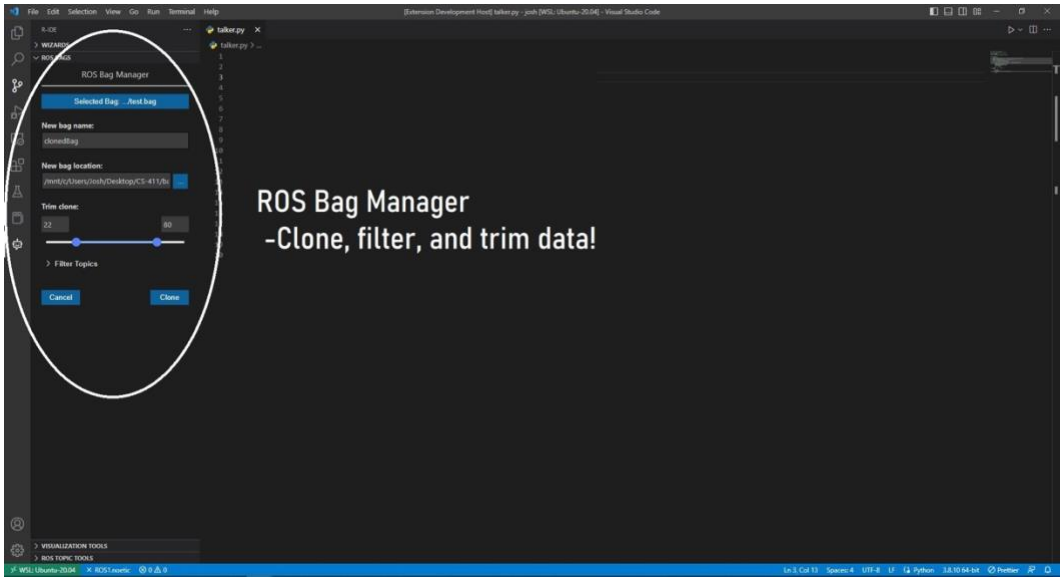
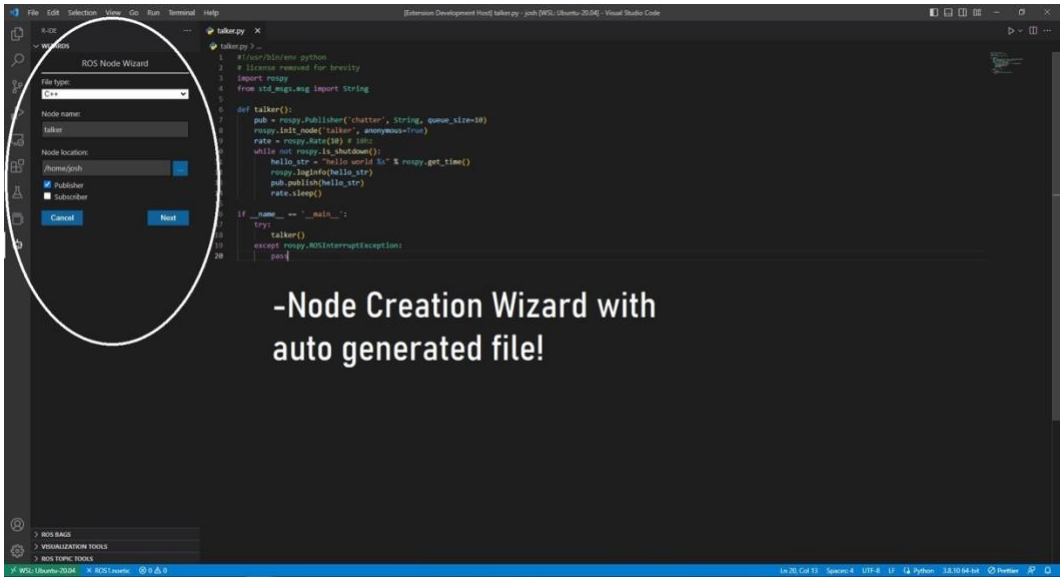


Figure 2

R-IDE Creation Wizard



2.1 Prototype Architecture Description

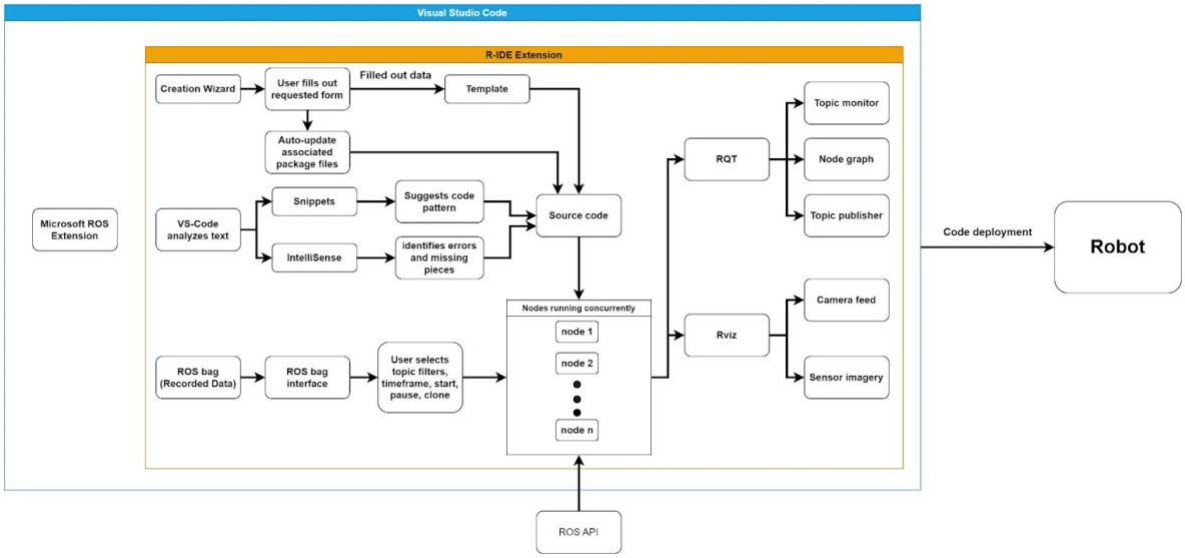
The R-IDE architecture is designed as a full extension for Visual Studio Code that exists alongside complementary extensions such as the Microsoft ROS extension. R-IDE includes a creation wizard that requests users to fill out a form to generate code templates for common ROS components. The creation wizard simplifies the process of creating new ROS nodes, service files, and message files by streamlining the initial setup process.

R-IDE leverages Visual Studio Code's IntelliSense and customized snippets to provide suggestions and fixes for code, making the development process more efficient and less prone to errors. Additionally, R-IDE includes a ROS bag interface that enables users to manipulate and manage ROS bags with ease. R-IDE provides users with various visualization tools embedded within VS-code, making it easier to analyze and interpret data output. The code generated by R-IDE is deployed to a robot through other tools and systems, and R-IDE does not handle the deployment process. However, R-IDE provides users with a comprehensive development environment that simplifies the process of developing and testing ROS applications.

The major functional components of the R-IDE architecture include a creation wizard, customized snippets and IntelliSense, a ROS bag interface, embedded visualization tools, and the usage analytics component using MongoDB via Mongoose. These components work together to simplify the process of developing ROS applications and provide valuable feedback to the development team for continuous improvement of R-IDE.

Figure 3

R-IDE Major Functional Components Diagram



2.2 Prototype Functional Description

R-IDE comes with a range of features that make it easy for developers to create, test, and debug ROS applications. R-IDE includes a wizard that allows users to quickly create message files, service files, and create nodes with templated publishers or subscribers. It also features an auto-update function that helps users to manage cmake files and package.xml. R-IDE includes snippets that enable developers to autocomplete code patterns for service files, message files, and C++ and Python packages. Additionally, it has a ROS bag interface that allows users to start and stop ROS bag recording, playback ROS bags, and filter topics from ROS bags. R-IDE also comes with powerful visuals such as rviz and node graph for visualizing and debugging ROS applications.

R-IDE includes ROS topic functions that allow developers to monitor ROS topics, publish ROS topics, and view media on ROS topics. It also includes data management features that provide usage analysis. Test management is also possible with the ability to create mock data and automated tests.

Table 1

R-IDE Prototype Features

	Feature	RWP	Prototype
Wizard	Create node w/ templated publisher or subscriber	Full	Full
	Create msg	Full	Full
	Create srv	Full	Full
Auto update	cmake file	Full	Full
	package.xml	Full	Full
Snippets	Autocomplete Code Patterns for Srv	Full	Full
	Autocomplete Code Patterns for Msg	Full	Full
	Autocomplete Code Patterns for C++ Packages	Full	Full
	Autocomplete Code Patterns for Python Packages	Full	Full
ROS bag	Start Rosbag recording	Full	Full
	Stop Rosbag recording	Full	Full
	Play back Rosbag	Full	Full
	Filter topics from Rosbag	Full	Full
Visuals	rviz	Full	Partial: Depending on performance of embedded features
	Node Graph	Full	Full
ROS Topic	ROS topic monitor	Full	Full
	ROS topic publisher	Full	Full
	ROS topic media viewer	Full	Full
Data Management	Usage Analysis	Full	Full
Test Management	Create Mock Data	Eliminated	Full
	Automated Tests	Eliminated	Full

2.3 External Interfaces

R-IDE will utilize specific software and user interfaces to enable users to interact with all components of the application. The primary user interface for R-IDE is Virtual Studio Code. In addition, R-IDE will interface with ROS 1 and ROS 2, two popular open-source robotics middleware frameworks. This allows developers to seamlessly integrate R-IDE into their existing ROS workflows, making it easy to transition from traditional ROS development to the streamlined environment provided by R-IDE. By leveraging these external interfaces, R-IDE

provides developers with a familiar and intuitive environment for developing advanced robotics applications.

2.3.1 Software Interfaces

R-IDE uses a range of software interfaces to provide a seamless development experience for ROS developers. These interfaces include the VSCode API extension, which enables R-IDE to integrate fully with Visual Studio Code. The R-IDE also uses MongoDB, through Mongoose, to collect and store usage data, which is essential for improving the tool over time. ROS Bridge and ROSLib are also key software interfaces used by R-IDE to interact with ROS 1 and ROS 2, respectively. RVIZ is used for visualization purposes, allowing developers to see the output data and make changes as necessary. Finally, R-IDE relies heavily on ROS itself, utilizing its functionalities to simplify and streamline the development process. By utilizing these software interfaces, R-IDE provides developers with a comprehensive and intuitive development environment that simplifies the process of developing ROS applications.

2.3.2 User Interfaces

R-IDE uses the Svelte framework for its graphical user interface to create several features, including the creation wizard, topic monitor, and ROS bag view. R-IDE's creation wizard is designed to enable users to expediently create message files, service files, and nodes with templated publishers or subscribers. This feature is implemented using Svelte's component-based architecture, which allows for the creation of reusable and easily manageable components. With Svelte's components, it is possible to create a wizard with a highly intuitive user interface.

Similarly, the topic monitor and ROS bag view features are also built using Svelte's component-based architecture. The topic monitor is designed to monitor ROS topics in real-time, and the ROS bag view allows users to record and play back ROS bags, as well as filter topics

from them. The use of Svelte in creating these components allows for dynamic and reactive interfaces, making it easy to display and manipulate data in real-time.

In addition to its component-based architecture, Svelte's state management library is used to manage the state of the GUI. This enables R-IDE to handle user inputs and application events in a simple and organized way. Furthermore, Svelte's reactive nature enables R-IDE to update the GUI in real-time, providing users with a seamless and intuitive experience.

2.3.3 Communications Protocols and Interfaces

R-IDE leverages MongoDB via Mongoose to enable efficient database communication. MongoDB is a NoSQL database that uses a document-oriented data model, while Mongoose is a Node.js-based object modeling tool that provides a simple and flexible API for working with MongoDB. The usage analytics component of the R-IDE uses MongoDB via Mongoose to insert event data into the database. This involves creating and validating schemas for the data and using Mongoose's API to insert the data into MongoDB. Mongoose also provides error handling capabilities, allowing the R-IDE to detect and handle errors that may occur during database operations. The flexible data model of MongoDB allows R-IDE to store data in a way that is optimal for developers use case, while Mongoose's powerful API simplifies the database communication process, minimizing the potential for errors and improving the overall development experience.