Bridging ROS2 & RIX Middleware Systems

Darren Cleeman

INTRODUCTION

Modern robotic software systems frequently rely on middleware frameworks to manage communication between distributed components. ROS2 has become a widely adopted standard in both research and industry due to its flexibility, ecosystem support, and real-time capabilities. In parallel, the University of Michigan has developed RIX as an alternative middleware framework with distinct architectural and design goals. While both systems provide publish-subscribe communication, their internal implementations, message definitions, and runtime behaviors are incompatible, preventing direct interoperability between ROS2 and RIX-based systems.

This incompatibility presents a challenge for projects that wish to leverage components from both ecosystems or transition between them over time. Rather than rewriting existing nodes or duplicating functionality, a translation layer that enables message-level interoperability offers a more flexible and scalable solution. The goal of this research is to explore and implement such a translation layer in the form of a bidirectional ROS2 to RIX bridge.

The work presented in this paper documents the progress made during a single academic semester toward this goal. The focus is not on delivering a fully complete or production-ready system, but rather on designing, implementing, and validating the core architecture of the bridge, identifying practical constraints, and demonstrating partial end-to-end functionality. The resulting system establishes a foundation that can be extended and completed in future work.

DESIGN & IMPLEMENTATION

The ROS2-RIX bridge is implemented as a single Python process that simultaneously participates in both middleware environments. At a high level, the bridge consists of a ROS2 node that internally instantiates and manages a RIX node. This co-located design allows the bridge to subscribe to topics in one framework, translate the received messages, and republish them into the other framework without modifying any external ROS2 or RIX nodes.

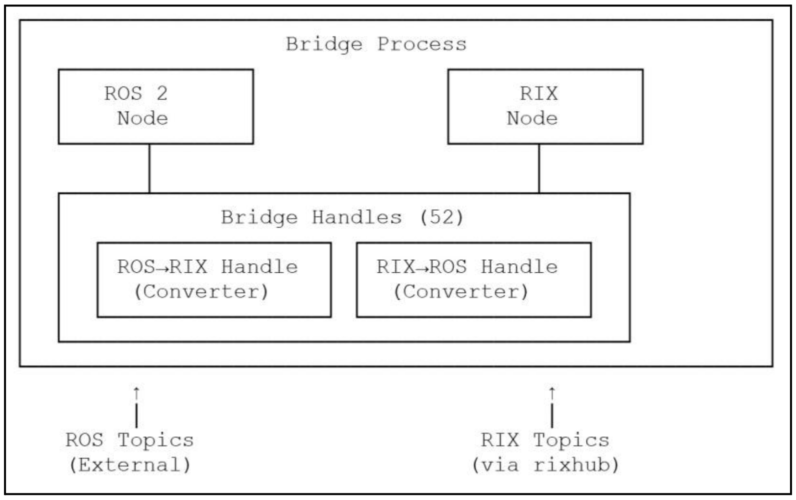


Figure 1: Bridge architecture showing the single process

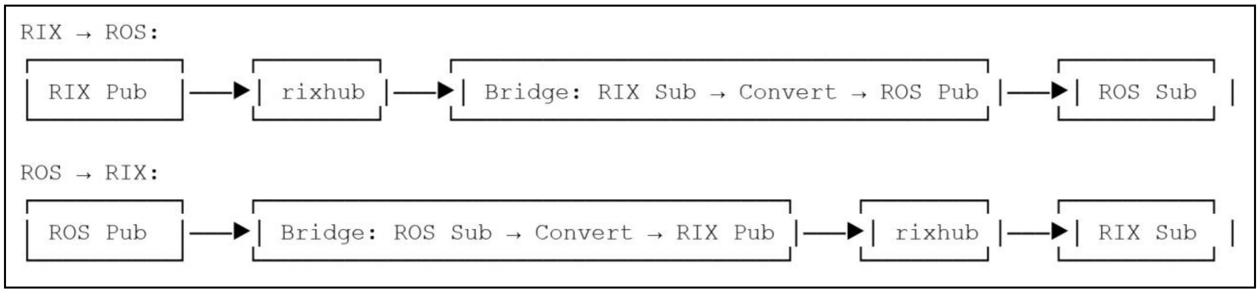
Message translation is handled by modular converter components, each responsible for translating a specific message type between ROS and RIX representations. Rather than embedding translation logic directly into the bridge core, each converter implements a consistent interface that supports bidirectional conversion. This separation allows new message types to be added incrementally without changing the overall system structure.

Early development focused on establishing a proof-of-concept implementation using a single message type. In this initial phase, the bridge supported only string messages, with one unidirectional path for ROS-to-RIX communication and another for RIX-to-ROS communication. Topic naming conventions were introduced to prevent message echoing and infinite feedback loops. This phase validated the core design decisions, including internal node management, message conversion, and event handling.

Following successful proof-of-concept testing, the bridge was generalized into a configurable system capable of supporting multiple message types. Rather than hard-coding topics and message mappings, the bridge loads its configuration from JSON files at startup. One configuration file specifies which topics should be bridged and in which direction, while a second mapping file defines how ROS message types correspond to RIX message types and which converter should be used. At runtime, the bridge dynamically instantiates a set of bridge handles based on this configuration.

To ensure predictable behavior and avoid concurrency-related issues, the bridge operates using a single-threaded execution model. Each iteration of the main loop alternates between processing ROS callbacks and processing RIX events. While a multithreaded ROS executor was briefly explored, this approach was abandoned due to stability concerns and potential thread-safety limitations on the RIX side. The resulting single-threaded design prioritizes reliability and clarity over raw throughput.

During operation, the bridge supports bidirectional message flow between the two middleware environments. Messages published in either framework are received by the corresponding internal node within the bridge, passed through the appropriate converter, and republished into the opposite framework. This design ensures that the same architectural components, internal nodes, bridge handles, and converters, are reused symmetrically for both directions of communication, providing a consistent and unified translation pipeline.


RIX → ROS:
RIX Pub
rixhub
Bridge: RIX Sub → Convert → ROS Pub
ROS Sub
ROS → RIX:
ROS Pub
Bridge: ROS Sub → Convert → RIX Pub
rixhub
RIX Sub
Figure 2: End-to-end message flow for messages through the bridge in both directions

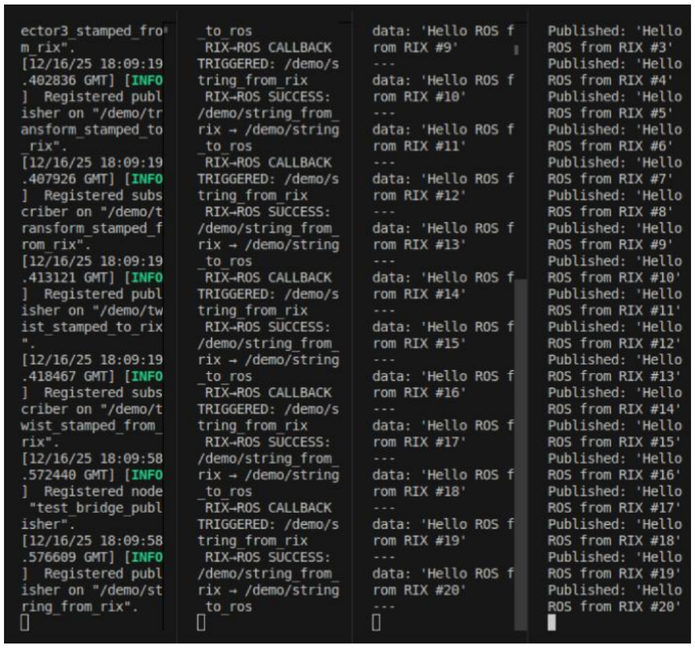
RESULTS

The work completed this semester resulted in a functional and extensible ROS2-RIX bridging system that demonstrates partial end-to-end interoperability and establishes a robust architectural foundation for continued development. Results are evaluated in terms of system stability, configuration correctness, message translation behavior, and end-to-end communication performance in both bridging directions.

The initial proof-of-concept implementation validated the feasibility of the approach using a single message type. Bidirectional string messages were successfully transferred between ROS2 and RIX, confirming that a single process could host both runtimes, subscribe to topics in one framework, convert messages, and republish them in the other without deadlock or runtime instability. This phase was essential in confirming that message translation and event handling could coexist within a unified execution loop.

Building upon this foundation, the generalized bridge successfully supports a total of 26 message types through its configuration-driven architecture, consisting of 14 standard message types and 12 geometry-related message types. At runtime, the bridge correctly parses its configuration files, instantiates bridge handles for each specified topic and direction, and associates each handle with the appropriate message converter. Across repeated launches, the system consistently initializes all configured bridges and remains stable during execution, even when managing a large number of active topic connections simultaneously.

The most complete and thoroughly validated functionality achieved this semester is the RIX-to-ROS communication path. In this direction, messages published by external RIX nodes are reliably routed through the RIX runtime to the bridge's internal RIX subscriptions. These messages are then converted into their corresponding ROS message formats and published on ROS topics, where they are successfully received by external ROS subscribers. This behavior was observed consistently across multiple message categories, including both standard and geometry-related message types, demonstrating that the conversion logic preserves message structure and data integrity.



```
ector3_stamped_fro
m_rix".
[12/16/25 18:09:19
.402836 GMT] [INFO
]  Registered publ
isher on "/demo/tr
ansform_stamped_to
_rix".
[12/16/25 18:09:19
.407926 GMT] [INFO
]  Registered subs
criber on "/demo/t
ransform_stamped_f
rom_rix".
[12/16/25 18:09:19
.413121 GMT] [INFO
]  Registered publ
isher on "/demo/tw
ist_stamped_to_rix
".
[12/16/25 18:09:19
.418467 GMT] [INFO
]  Registered subs
criber on "/demo/t
wist_stamped_from_
rix".
[12/16/25 18:09:58
.572440 GMT] [INFO
]  Registered node
 "test_bridge_publ
isher".
[12/16/25 18:09:58
.576609 GMT] [INFO
]  Registered publ
isher on "/demo/st
ring_from_rix".
▯
```

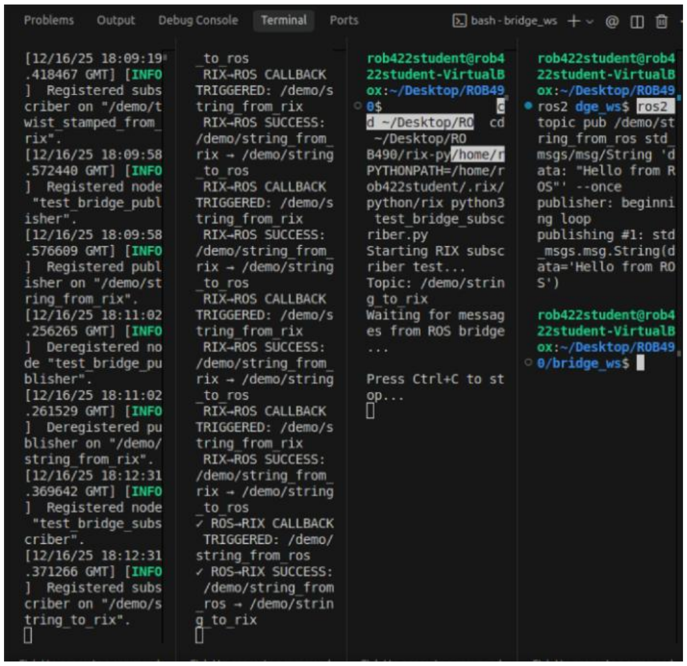
```
 _to_ros
  RIX→ROS CALLBACK
 TRIGGERED: /demo/s
 tring_from_rix
  RIX→ROS SUCCESS:
 /demo/string_from_
 rix → /demo/string
 _to_ros
  RIX→ROS CALLBACK
 TRIGGERED: /demo/s
 tring_from_rix
  RIX→ROS SUCCESS:
 /demo/string_from_
 rix → /demo/string
 _to_ros
  RIX→ROS CALLBACK
 TRIGGERED: /demo/s
 tring_from_rix
  RIX→ROS SUCCESS:
 /demo/string_from_
 rix → /demo/string
 _to_ros
  RIX→ROS CALLBACK
 TRIGGERED: /demo/s
 tring_from_rix
  RIX→ROS SUCCESS:
 /demo/string_from_
 rix → /demo/string
 _to_ros
  RIX→ROS CALLBACK
 TRIGGERED: /demo/s
 tring_from_rix
  RIX→ROS SUCCESS:
 /demo/string_from_
 rix → /demo/string
 _to_ros
▯
```

```
data: 'Hello ROS f
rom RIX #9'
---
data: 'Hello ROS f
rom RIX #10'
---
data: 'Hello ROS f
rom RIX #11'
---
data: 'Hello ROS f
rom RIX #12'
---
data: 'Hello ROS f
rom RIX #13'
---
data: 'Hello ROS f
rom RIX #14'
---
data: 'Hello ROS f
rom RIX #15'
---
data: 'Hello ROS f
rom RIX #16'
---
data: 'Hello ROS f
rom RIX #17'
---
data: 'Hello ROS f
rom RIX #18'
---
data: 'Hello ROS f
rom RIX #19'
---
data: 'Hello ROS f
rom RIX #20'
---
▯
```

```
Published: 'Hello
ROS from RIX #3'
Published: 'Hello
ROS from RIX #4'
Published: 'Hello
ROS from RIX #5'
Published: 'Hello
ROS from RIX #6'
Published: 'Hello
ROS from RIX #7'
Published: 'Hello
ROS from RIX #8'
Published: 'Hello
ROS from RIX #9'
Published: 'Hello
ROS from RIX #10'
Published: 'Hello
ROS from RIX #11'
Published: 'Hello
ROS from RIX #12'
Published: 'Hello
ROS from RIX #13'
Published: 'Hello
ROS from RIX #14'
Published: 'Hello
ROS from RIX #15'
Published: 'Hello
ROS from RIX #16'
Published: 'Hello
ROS from RIX #17'
Published: 'Hello
ROS from RIX #18'
Published: 'Hello
ROS from RIX #19'
Published: 'Hello
ROS from RIX #20'
█
```

Figure 3: Terminal output showing successful RIX-to-ROS, Successfully published message in terminal 2, and ROS message received in third terminal

The ROS-to-RIX direction exhibits partial functionality. The bridge reliably receives ROS messages through its ROS subscriptions and correctly applies the appropriate conversion logic. Diagnostic logging confirms that converted messages are published through the RIX interface within the bridge process. However, external RIX subscribers running in separate processes do not consistently receive these messages. This indicates that the bridge's internal logic for ROS-to-RIX translation is functioning as intended, while the remaining limitation likely resides in RIX inter-process message routing, runtime configuration, or discovery behavior rather than in the bridge itself.



```
[12/16/25 18:09:19
.418467 GMT] [INFO
]  Registered subs
criber on "/demo/t
wist_stamped_from_
rix".
[12/16/25 18:09:58
.572440 GMT] [INFO
]  Registered node
 "test_bridge_publ
isher".
[12/16/25 18:09:58
.576609 GMT] [INFO
]  Registered publ
isher on "/demo/st
ring_from_rix".
[12/16/25 18:11:02
.256265 GMT] [INFO
]  Deregistered no
de "test_bridge_pu
blisher".
[12/16/25 18:11:02
.261529 GMT] [INFO
]  Deregistered pu
blisher on "/demo/
string_from_rix".
[12/16/25 18:12:31
.369642 GMT] [INFO
]  Registered node
 "test_bridge_subs
criber".
[12/16/25 18:12:31
.371266 GMT] [INFO
]  Registered subs
criber on "/demo/s
tring_to_rix".
```

```
 _to_ros
  RIX→ROS CALLBACK
TRIGGERED: /demo/s
tring_from_rix
  RIX→ROS SUCCESS:
/demo/string_from_
rix → /demo/string
 _to_ros
  RIX→ROS CALLBACK
TRIGGERED: /demo/s
tring_from_rix
  RIX→ROS SUCCESS:
/demo/string_from_
rix → /demo/string
 _to_ros
  RIX→ROS CALLBACK
TRIGGERED: /demo/s
tring_from_rix
  RIX→ROS SUCCESS:
/demo/string_from_
rix → /demo/string
 _to_ros
  RIX→ROS CALLBACK
TRIGGERED: /demo/s
tring_from_rix
  RIX→ROS SUCCESS:
/demo/string_from_
rix → /demo/string
 _to_ros
✓ ROS→RIX CALLBACK
  TRIGGERED: /demo/
string_from_ros
✓ ROS→RIX SUCCESS:
  /demo/string_from
 _ros → /demo/strin
g_to_rix
```

```
rob422student@rob4
22student-VirtualB
ox:~/Desktop/ROB49
0$                c
d ~/Desktop/RO  cd
 ~/Desktop/RO
B490/rix-py/home/r
PYTHONPATH=/home/r
ob422student/.rix/
python/rix python3
 test_bridge_subsc
riber.py
Starting RIX subsc
riber test...
Topic: /demo/strin
g_to_rix
Waiting for messag
es from ROS bridge
...

Press Ctrl+C to st
op...
```

```
rob422student@rob4
22student-VirtualB
ox:~/Desktop/ROB49
ros2 dge_ws$ ros2
topic pub /demo/st
ring_from_ros std_
msgs/msg/String 'd
ata: "Hello from R
OS"' --once
publisher: beginni
ng loop
publishing #1: std
_msgs.msg.String(d
ata='Hello from RO
S')

rob422student@rob4
22student-VirtualB
ox:~/Desktop/ROB49
0/bridge_ws$
```

Figure 4: Terminal output showing ROS-to-RIX issue. Successfully published message in terminal 2, but no message received in terminal 3

In addition to functional results, several important system-level observations were made during testing and debugging. Execution environment consistency was found to be critical for reliable RIX communication; running RIX publishers and subscribers within the correct virtual environment and as standalone scripts significantly improved behavior compared to inline execution. Furthermore, attempts to introduce multithreading revealed stability issues consistent with thread-safety limitations, leading to the adoption of a single-threaded execution model. While this choice limits potential throughput, it provides predictable and stable behavior appropriate for a research prototype.

Collectively, these results demonstrate that the core bridge architecture, configuration system, and conversion framework are sound. The system achieves reliable interoperability in one direction and provides strong evidence that full bidirectional communication is attainable with further investigation into RIX runtime behavior.

FUTURE WORK

The immediate priority for future work is resolving the remaining ROS-to-RIX message delivery issue. Although the bridge successfully receives ROS messages, converts them, and publishes them through the RIX interface, external RIX subscribers running in separate processes do not consistently receive these messages. Addressing this limitation is essential for achieving full bidirectional interoperability.

Initial efforts will focus on investigating how rixhub handles messages published by the bridge's internal RIX node. In particular, it will be necessary to determine whether messages originating from the bridge are treated differently than those from native RIX publishers. To isolate the source of the issue, testing will be performed using simplified, minimal RIX-only scenarios that remove the bridge entirely, allowing baseline multi-process communication behavior to be established. These results will then be compared against bridge-mediated publishing. In parallel, relevant RIX documentation and examples will be consulted to ensure that the bridge follows recommended patterns for multi-process communication and publisher registration.

Once reliable ROS-to-RIX delivery is achieved, development will proceed to a third phase focused on architectural and performance improvements. One major area of future work involves revisiting multi-threaded execution. While the current single-threaded model prioritizes stability, future versions of the bridge will aim to re-enable multi-threaded execution with appropriate synchronization mechanisms on the RIX side. This includes introducing thread-safe queues for message passing across threads and carefully managing access to shared RIX resources. Successfully reintroducing multithreading would improve throughput and better support high-volume message scenarios.

Future work will also expand the range of supported message types. Planned additions include commonly used message categories for sensing and state estimation, such as scan data, image data, point cloud representations, odometry, and path descriptions (e.g., LaserScan, Image, PointCloud2, Odometry, and Path). Expanding support for these message types would increase the bridge's applicability to perception, navigation, and mapping pipelines, with a target of supporting more than forty message types in total.

Finally, support for custom message types is a longer-term goal. This includes allowing users to define their own ROS-RIX message mappings, provide custom converter functions, and potentially enable dynamic message type discovery at runtime. Together, these extensions would make the bridge more flexible and adaptable to project-specific needs while preserving the configuration-driven design established during this semester.