Enabling Runtime Reconfiguration in Multimodal Teleoperation Systems

Luuk B.L. Lenders[1,*], Thijs I. Bink[1], Douwe Dresscher[1], Kenan Niu[1], Jan B.F. van Erp[1,2] and Jan. F. Broenink[1]

[1]University of Twente, Drienerlolaan 5, 7522 NB Enschede, Netherlands
[2]TNO, Kampweg 55, 3769 DE Soesterberg, Netherlands

Abstract

Multimodal teleoperation systems integrate multiple sensing and control modalities, such as video, audio, and robot control. These systems are often configured only once during startup, making it difficult to adapt behaviour when operating conditions change. Adjustments to parameters or modality behaviour typically require restarting components, interrupting operation and reducing operator immersion.

This paper presents a software architecture that enables runtime reconfiguration of multimodal teleoperation systems. The design introduces a central configuration controller that coordinates parameter updates and lifecycle transitions across modality subsystems while the system remains operational. The architecture is implemented using lifecycle-managed components and integrated into an existing teleoperation platform.

Experimental evaluation shows that most configuration operations complete within tens of milliseconds, enabling reconfiguration without noticeable disruption to system interaction. The architecture provides a foundation for future work on automated runtime adaptation in multimodal teleoperation systems.

Keywords

Teleoperation, Runtime Reconfiguration, Robotics Software Architecture, ROS Lifecycle, Multimodal Systems

1. Introduction

Teleoperation systems enable human operators to interact with remote environments through a combination of parallel operating sensing and control modalities such as vision, haptics, and audio. These multimodal systems are inherently complex, as they must process multiple data streams while maintaining responsiveness and stability for the operator. They also integrate multiple research domains, including robotics, control, networking, human-robot interaction, and software engineering.

Despite this complexity, many teleoperation systems have been developed for a specific experiment or task and are configured with static system parameters. As a result, behaviour cannot easily be modified during operation. When conditions change, such as network quality, computational load, or task requirements, adaptation typically requires stopping and restarting components to reconfigure. This interruption breaks operator immersion and reduces the usability of the system, motivating mechanisms that allow safe runtime modification.

To address these challenges, teleoperation systems require mechanisms that allow system parameters and modality behaviour to be adapted during runtime without interrupting operation. This enables the system to maintain functionality under changing conditions while preserving operator immersion.

Several approaches allow adapting individual modalities, like adaptive video streaming techniques in teleoperation and remote vehicle operation to maintain visual feedback under fluctuating network conditions [1]. Similar adaptation strategies are widely used in teleconferencing systems, where video quality and bitrate are dynamically adjusted to maintain communication quality [2, 3]. However, these

ICRA'26: Workshop Robotics Software Engineering (RoSE), June 1 2026, Vienna, AU
*Corresponding author.
l.b.l.lenders@utwente.nl (L. B.L. Lenders)
https://www.ram.eemcs.utwente.nl (L. B.L. Lenders)
0009-0008-5352-4189 (L. B.L. Lenders); 0009-0005-0637-6919 (T. I. Bink); 0000-0002-5315-9029 (D. Dresscher); 0000-0002-5498-4311 (K. Niu); 0000-0002-6511-2850 (J. B.F. v. Erp); 0000-0003-3039-3012 (Jan. F. Broenink)

approaches focus on individual data streams rather than the overall behaviour of multiple sensing and control modalities in the teleoperation system.

Other work addresses resource optimisation from a networking perspective. These approaches typically optimise bandwidth allocation or communication behaviour based on observed traffic patterns[4, 5]. While such techniques can improve network utilisation, they operate primarily at the communication layer and do not consider how application-level components should adapt their behaviour.

Multimodal teleoperation systems introduce additional complexity, as different modalities impose distinct performance requirements. Visual feedback may prioritise bandwidth and resolution, while control signals require low latency and high reliability. Approaches that optimise individual streams, communication layers, or assume a single system design, therefore, do not capture the system-level trade-offs required to maintain operator experience.

These limitations highlight the need for a system-level approach that enables coordinated runtime reconfiguration of teleoperation system components.

In this paper, we present a software architecture for coordinated runtime reconfiguration of multimodal teleoperation systems, enabling safe adaptation of subsystem parameters and operational states during execution without interrupting the overall system. For validation, reconfiguration latency and practical usability in a real teleoperation system were investigated.

2. Design

To guide the design, two aspects were considered: the requirements for the reconfiguration mechanism and the structure of the teleoperation system in which it is applied. The proposed architecture is designed to satisfy the following requirements: (1) The approach should be *broadly applicable* to a wide range of multimodal teleoperation systems; (2) Given available computational and network resources, the system should be able to *optimise behaviour* for the *operator's experience*; (3) System components must be able to *adapt* behaviour *during runtime* without requiring a system shutdown.

2.1. Configuration Controller

To enable runtime reconfiguration, we introduce a software component called the *configuration controller* that manages configuration changes across the teleoperation system by interacting with modality subsystems and modifying their parameters or operational state when required.

It consists of three core parts. (1) The *state manager* is responsible for tracking the lifecycle states of external subsystems and issuing commands that trigger state transitions when required. (2) The *configuration parameter manager* manages configuration updates by maintaining knowledge of configurable subsystem parameters and issuing modification requests to the respective components. (3) The controller exposes a centralised interface through which external tools, such as TUIs, GUIs, or adaptive controllers, can interact with the configuration controller. This separation of responsibilities simplifies runtime reconfiguration while maintaining a clear interface for external tools.

The design follows the 5C model [6], which structures robotic software into configuration, coordination, computation, communication, and composition blocks. Within this model, configuration defines system parameters, coordination determines when system behaviour changes occur, and computation performs the underlying processing tasks. In this implementation, runtime reconfiguration primarily affects resource usage through quality parameters, while preserving the functional behaviour of the component. This is in contrast to work looking at system variability and reconfiguration on a between-component level such as presented by [7]

The implementation uses the ROS lifecycle node concept [8], which naturally reflects this separation of concerns. Node parameters and initialisation correspond to configuration, while the lifecycle state machine provides a coordination mechanism that controls when components should transition their state. The periodic control loop performs the computational tasks, while ROS itself provides the communication and composition infrastructure between nodes. The mapping between the ROS lifecycle model and the 5C layers is illustrated in Figure 1.

The configuration controller operates at the configuration and coordination layers, managing parameters and lifecycle states while leaving subsystem computation unchanged. This allows existing components to operate independently of the reconfiguration mechanism.

By leveraging this lifecycle mechanism, the configuration controller can safely update parameters and trigger state transitions when required. This approach enables modifications to be applied during runtime while ensuring that system components remain in a valid operational state.

The question of when it is safe to apply reconfiguration is a key aspect of dynamic software systems, often formalised through concepts such as quiescence or tranquility [9, 10]. In our work, such conditions are not enforced explicitly. Most data streams are transient, and dropping a small amount of data is typically negligible for human operators, though it may affect algorithmic processing. If required, nodes can delay applying updates until a safe moment. This is particularly relevant for physical or actuated systems, where unsafe updates may affect stability.

Although the current implementation relies on ROS lifecycle nodes, the underlying design principles are not tied to a specific software platform. The architecture can be adapted to other software frameworks by replacing the communication and coordination mechanisms while keeping the overall control structure intact.

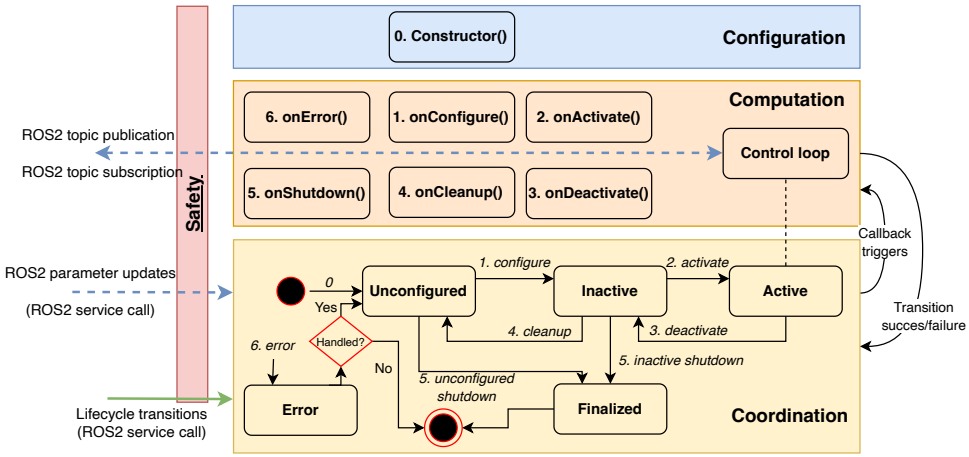

Figure 1: Mapping of the ROS lifecycle model onto the 5C software architecture model.

2.2. Integration into an Existing Teleoperation System

Integrating the configuration controller requires subsystems to expose mechanisms for configuration updates. First, the subsystem structure must separate coordination and computation functionality into lifecycle-managed states, enabling safe transitions. Second, the subsystem must provide mechanisms for applying configuration updates requested by the configuration controller. The node provides this knowledge to the controller, which maintains a central registry.

For example, camera resolution changes may require restarting the capture pipeline, while haptic systems may restrict update-rate changes due to stability concerns. The controller coordinates these actions through parameter updates and lifecycle transitions, minimising disruption.

3. Evaluation and Results

To evaluate the proposed architecture, we conducted two experiments: (1) Measuring the latency introduced by runtime configuration changes and lifecycle transitions; and (2) Evaluating the practical usability of the architecture by letting users modify system parameters during runtime.

Hardware and Software Components The experiments were conducted on a multimodal teleoperation platform that was initially implemented in earlier work [11]. In the current implementation,

configuration decisions are triggered manually by either the system operator or the experimenter[12].[1], This allows the architecture to be evaluated independently of a specific optimisation strategy.

The configuration controller interacts with all subsystems of the teleoperation system (Figure 2). Five out of the eight subsystems are implemented using lifecycle nodes. The FR3 controller is implemented using the `ros2_control` framework and therefore supports parameter updates but does not expose lifecycle transitions. This is due to manufacturer limitations and incompatibility with the lifecycle concept. Similarly, the C# ROS implementation does not support lifecycle nodes.

All nodes interact with the configuration controller. In the experiments below, the configuration controller was controlled from a TUI, and a GUI written for Unity, respectively. For neither experiment quiescence or tranquility where used as a requirement for parameter updates.

3.1. Experiment 1: Parameter and Lifecycle Latency

For each subsystem, all supported lifecycle transitions, parameter updates, or combined operations were executed multiple times system, for a total of approximately 2200 requests. Different restart policies were attached to parameter updates. Three policies exist: (1) No restart; only the parameter is updated; (2) Partial restart; deactivate and reactivate the node; or (3) Full restart; complete lifecycle reset.

All components remained active during the experiment to simulate realistic computational and network load. Latency was measured as the time between issuing a command and receiving a successful completion response. For multi-step commands, the independent steps were also measured separately. The resulting latency values were then aggregated to obtain the median latency for each operation.

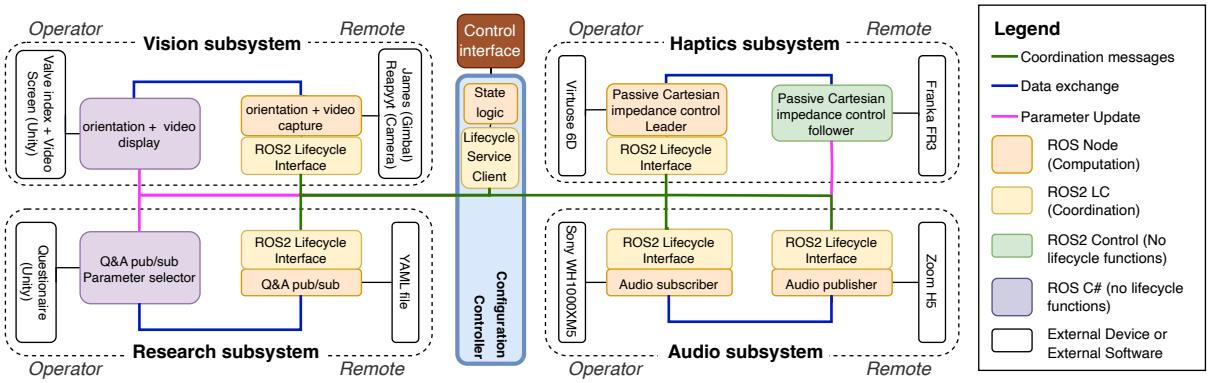

Figure 2: The Configuration Controller implemented in an existing teleoperation system. Colours match with functionality from Figure 1. Experiment 1 uses the Vision and Audio subsystems; Experiment 2 uses the Full system as shown.

Results The measured transaction latencies varied significantly across nodes and command types. Figure 3 and Table 1 presents the results.

Parameter updates (denoted by the 'set' command) without lifecycle restarts exhibit the lowest latency. They complete within a few computation cycles of the remote node. E.g. the FR3 and Camera Quality (Q) Set commands. Parameter updates which require some form of I/O handling have significantly higher delays. E.g., Camera Res/FPS (R) Set commands.

To better understand the sources of latency in a restart, Figure 4 presents a breakdown of the median latency per lifecycle step across nodes. Lifecycle commands themselves generally complete quickly. Often between 10 ms and 60 ms. These results indicate that most runtime reconfiguration operations occur well below typical human interaction latency thresholds of approximately 100 ms [13, 14].

When a step takes a larger amount of time, this is often due to hardware handling or stopping time-triggered callback functions safely. E.g. the Virtuose activate and the Virtuose or Camera deactivate, respectively. The Gimbal Configuration is the slowest due to the recalibration of its relative sensors.

[1]Version used for this paper can be found at RAM Gitlab under the tag *ROSE26*

Both partial and full resets incur an extra latency cost to parameter updates, of which the cost is approximately the sum of the parameter update and associated lifecycle steps, plus an average overhead of approximately 25 ms.

Table 1
Median latencies [ms] per node and function.

Function	Camera	Gimbal	FR3	Virtuose
Configure	20.56	14820.58	-	1.83
Activate	2.36	5.03	-	52.33
Deactivate	55.67	8.41	-	17.65
Cleanup	11.60	4.70	-	2.08
Set	(R) 510.43 / 102.37 (Q)	-	4.72	3.68

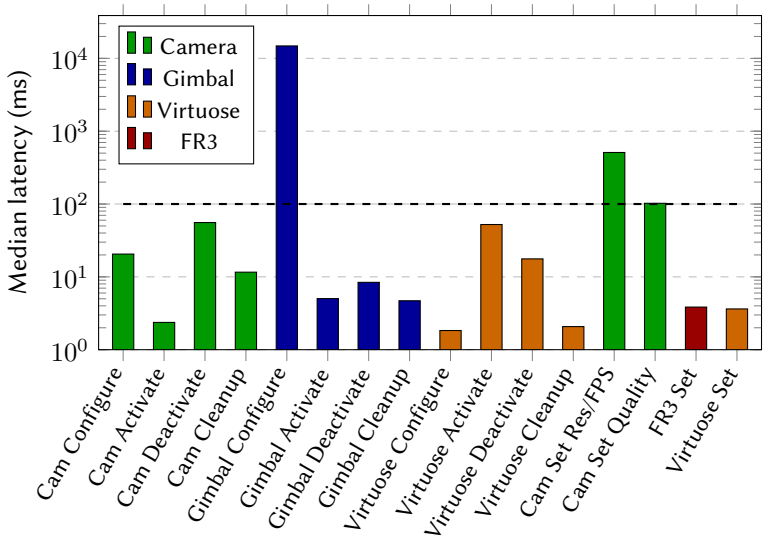

Figure 3: Median latency per node and command (log scale). Typical human interaction latency indicated as dashed line (100ms).

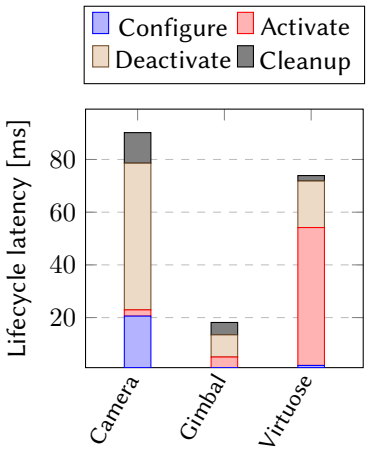

Figure 4: Lifecycle transition latency per node (stacked). Gimbal Configuration latency is too high to show.

3.2. Experiment 2: User-driven Runtime Reconfiguration

In the second experiment, the system was used by a group of 25 non-expert participants. Participants performed a teleoperation task while modifying modality parameters through the configuration interface. Parameter options were presented without descriptive labels and randomised between trials. This prevented participants from relying on prior knowledge of the parameter layout.

System behaviour and stability were monitored to detect failures or subsystem interruptions.

Results As the participants randomly changed settings, the system was exposed to a randomised set of instructions. The experiment ran for an average of 2 hours per participant without experimenter intervention. In total, approximately 50 hours with constant user input and use of the full teleoperation system were recorded. No failure of systems was experienced during these long test sessions. Furthermore, participants did not speak up about reconfiguration latency.

4. Conclusion

The results demonstrate that the overhead introduced by the configuration controller and lifecycle transitions is generally small compared to the internal processing performed by individual subsystems.

For most components, lifecycle transitions complete within tens of milliseconds. These values indicate that coordinated runtime reconfiguration can be performed without significantly disrupting system responsiveness. However, the results also highlight that subsystem-specific implementation details can dominate reconfiguration latency. In particular, hardware initialisation routines or calibration procedures can significantly increase configuration time, as observed for the camera gimbal calibration.

In the user test, no note was made of reconfiguration latencies by the participants, although this might be due to the fairly passive nature of the tasks, which did not require constant input.

To conclude, experimental results show that most reconfiguration operations complete within tens of milliseconds, indicating that runtime adaptation can be achieved without significantly disrupting system responsiveness. This suggests that the proposed architecture introduces only minimal overhead, given that internal coordination steps are implemented efficiently. The presented architecture forms a foundation for future work on automated runtime adaptation strategies in multimodal teleoperation systems. Furthermore, future work will focus on automating configuration decisions based on observed system conditions. By analysing user configuration choices under varying resource constraints, this may enable the system to automatically select configurations that maintain operator experience under degraded conditions. Finally, future research will also investigate when a actuated system is stable enough to trigger parameter updates.

In addition, future research will investigate the use of external signals such as operator physiological measurements or task context to further improve adaptive system behaviour.

Declaration on Generative AI No AI was used for the writing of this paper

References

[1] M. Hofbauer, Adaptive Live Video Streaming for Teleoperated Driving, Ph.D. thesis, TUM, 2022. URL: https://mediatum.ub.tum.de/doc/1651626/1651626.pdf.

[2] L. De Cicco, S. Mascolo, V. Palmisano, Skype Video congestion control: An experimental investigation, Computer Networks 55 (2011) 558–571. doi:10.1016/j.comnet.2010.09.010.

[3] L. Liu, J. Li, H. Xu, K. Xue, J. C. Xue, Efficient Real-time Video Conferencing with Adaptive Frame Delivery, Computer Networks 234 (2023) 109918. doi:10.1016/j.comnet.2023.109918.

[4] V. Gokhale, J. Nair, S. Chaudhuri, S. Kakade, Network-aware adaptive sampling for low bitrate telehaptic communication, in: Haptics: Science, Technology, and Applications, Springer International Publishing, Cham, 2018, pp. 660–672. doi:10.1007/978-3-319-93399-3_56.

[5] B. Cizmeci, X. Xu, R. Chaudhari, C. Bachhuber, N. Alt, E. Steinbach, A multiplexing scheme for multimodal teleoperation, ACM Transactions on Multimedia Computing, Communications and Applications 13 (2017). doi:10.1145/3063594.

[6] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, D. Brugali, The brics component model: a model-based development paradigm for complex robotics software systems, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Association for Computing Machinery, New York, NY, USA, 2013, p. 1758–1764. doi:10.1145/2480362.2480693.

[7] S. Peldszus, D. Brugali, D. Strüber, P. Pelliccione, T. Berger, Software reconfiguration in robotics, Empirical Software Engineering 30 (2025) 94. doi:10.1007/s10664-024-10596-9. arXiv:2310.01039.

[8] G. Biggs, T. Foote, Managed Nodes, 2021. URL: https://design.ros2.org/articles/node_lifecycle.html.

[9] J. Kramer, J. Magee, Dynamic Change Management: Quiescence Revisited, IEEE Transactions on Software Engineering 51 (2025) 746–750. doi:10.1109/TSE.2024.3521298.

[10] Y. Vandewoude, P. Ebraert, Y. Berbers, T. D'Hondt, Tranquility: A low disruptive alternative to quiescence for ensuring safe dynamic updates, IEEE Transactions on Software Engineering 33 (2007) 856–868. doi:10.1109/TSE.2007.70733.

[11] T. I. Bink, Quantifying the impact of Quality of Service on the Quality of Experience in multimodal teleoperation, Msc. thesis, University of Twente, 2025.

[12] L. Lenders, T. I. Bink, D. Dresscher, K. Niu, J. v. Erp, J. F. Broenink, Dataset on Zenodo, 2026. doi:10.5281/zenodo.19933980.

[13] Y. Xu, L. Huang, T. Zhao, Y. Fang, L. Lin, A Timestamp-Independent Haptic–Visual Synchronization Method for Haptic-Based Interaction System, Sensors 22 (2022) 1–14. doi:10.3390/s22155502.

[14] D. Brahimaj, M. Ouari, A. Kaci, F. Giraud, C. Giraud-Audine, B. Semail, Temporal Detection Threshold of Audio-Tactile Delays With Virtual Button, IEEE Transactions on Haptics 16 (2023) 491–496. doi:10.1109/TOH.2023.3268842.