A Conformance Checking System for Interaction Testing in Virtual Reality

1st Vijay Aravynthan S.R.
Software Engineering Research Center
International Institute of Information Technology
Hyderabad, India
0009-0006-0099-5839

2nd Y. Raghu Reddy
Software Engineering Research Center
International Institute of Information Technology
Hyderabad, India
0000-0003-2280-5400

***Abstract*—The rapid growth of Virtual Reality (VR) across critical sectors like healthcare, education, and gaming necessitates robust methods for quality assurance. As VR applications increase in complexity, ensuring correctness of interactions becomes challenging. This paper presents our ongoing work on a novel, cross-platform conformance checking system designed to verify that VR interactions are as specified. Our system features three key parts: a JSON-based format for defining ideal interaction specifications, an intuitive visual editor for designing the interaction flows, and a rules engine that automatically compares runtime behavior against the predefined sequences. This system is designed to streamline the testing process and ensure behavioral consistency across diverse VR platforms.**

***Index Terms*—Virtual Reality, Conformance Testing, Process Mining, Interaction Logging, Quality Assurance, Visual Editor**

I. Introduction

Virtual Reality (VR) has expanded beyond gaming to areas such as medical training, industrial design, and remote collaboration. As applications in these areas become more complex, ensuring their correctness becomes increasingly challenging due to challenges inherent in VR, such as tracking multiple inputs (gaze, gesture, voice), spatial interactions, multiple users and real-time performance.

In addition to these complexities, a key aspect of VR quality assurance is ensuring applications guide users through correct workflows and respond predictably. This is especially important in training and simulation, where following procedures accurately is essential. However, verifying these interaction sequences manually can be slow, difficult, and prone to errors. To address this, we propose a conformance checking system that enables developers and quality assurance (QA) engineers to specify and verify interaction sequences in VR. Building on our prior research in building a framework for logging various aspects of VR applications [1], this paper introduces a system that allows for the creation of an ideal specification for application behavior and a mechanism to check runtime behavior against it. The key contributions of this paper are:

- A JSON-based format for defining interaction sequences referred to as *Ideal specification*.
- A user-friendly *Visual Editor* that allows non-programmers to design and visualize test cases as flowcharts.
- A platform-agnostic *Rules Engine* that validates runtime logs against the ideal specifications.

II. The Case for VR Conformance Testing

VR interaction logging is an emerging field, but its full potential for quality assurance is unlocked only when logs are actively verified against expected behavior. This form of verification, known as conformance checking, is a cornerstone of process mining and is used across industries to validate whether the actual execution of a process aligns with a predefined model [2], [3].

A conformance checking framework elevates logging from a passive data collection activity into a powerful verification mechanism, turning logs into a verifiable asset for ensuring system reliability [2]. This approach transforms the abstract goal of ”the application should work correctly” into a concrete, machine-readable set of rules. This allows for an objective, scalable, and repeatable method of verification, replacing what is often a manual and subjective testing process.

A. Applications of Conformance Checking

- **Ensuring Procedural Integrity and Compliance:** In regulated or high-stakes industries, VR training simulations must adhere to strict procedural standards. Our system can generate auditable, objective evidence that a simulation correctly implements and enforces these standards. This is especially critical in domains such as manufacturing, life sciences, aerospace, and healthcare, where precise adherence to procedures is essential. For example, in manufacturing, verifying that a technician follows the exact sequence for installing a critical component ensures safety and quality; in life sciences and healthcare, training personnel on complex lab protocols is vital, as deviations could compromise safety or accuracy.
- **Optimizing user interfaces:** A conformance specification defines the ”ideal” path for a task. By comparing user logs against this baseline, developers can perform *deviation analysis* to identify points of confusion in the User Interface or gaps in the training curriculum. The insights gathered can be used for

2151-0849/25/$31.00 ©2025 IEEE
DOI 10.1109/ASEW67777.2025.00070

data-driven design improvements, leading to more intuitive user interfaces.

B. Challenges in Conformance Checking

Verifying conformance of interactions in VR applications presents unique challenges. VR environments involve multi-modal inputs such as gaze, gesture, voice, and haptic feedback, which often occur simultaneously or in rapid succession, making it difficult to precisely record and analyze each interaction. Some other challenges are:

- **Temporal Variance:** Users do not perform actions with machine precision. The system must account for reasonable variations in timing. For instance, a user might be expected to press a button within 5 seconds of a prompt appearing. The system must validate this time window, not an exact millisecond.
- **Spatial Ambiguity:** Actions like "touch button" involve spatial proximity, not exact coordinates. A 'touch' event should register if the user's virtual hand is within a small threshold distance of a button, rather than requiring an exact intersection of 3D coordinates.
- **Non-Deterministic Behavior:** Users may perform many extraneous, but valid, actions. The checker must be robust enough to ignore irrelevant events and focus only on the sequence under test.

III. Related Work and Research Gaps

The testing and verification of interactions in VR applications is an active area of research, motivated by the growing importance of comprehensive quality assurance (QA) methodologies. A systematic review by Speicher et al. [4] underscores the increasing need for advanced QA practices in virtual reality. While early research primarily focused on performance metrics such as frame rate and latency [2], the focus has recently shifted toward the challenges of interaction testing.

In addressing the complexities of VR interactions, Stauffert et al. [6] proposed a taxonomy for bugs in VR games, emphasizing issues related to user interaction. Attempts to automate testing using AI agents to navigate virtual environments are effective in finding crashes but do not inherently verify behavior against formal specifications [7], [8]. Our approach differs by being specification-based, aiming to verify correctness in accordance with a predefined model.

A crucial element for any conformance checking is the availability of reliable runtime data. Several researchers have underscored the necessity for structured logging in immersive environments [4], [9]. Our framework capitalizes on this need by utilizing standardized logs as the primary input.

Despite progress in VR testing, recent surveys of the field highlight a significant gap in the formal specification and automated verification of complex interaction sequences. As noted by Speicher et al. [4], much of the existing research focuses on performance metrics and bug reporting, with less emphasis on the automated testing of functional correctness from a user-interaction perspective [5]. Current approaches often lack a standardized, human-readable, and machine-parseable format for defining correct interaction flows, as well as an intuitive method for non-programmers, like instructional designers and QA testers, to create and interpret these test cases. Furthermore, there is an absence of a platform-agnostic engine to validate runtime logs against these ideal specifications.

Our work directly addresses these gaps by offering an end-to-end solution that includes visual test case design and automated runtime verification. This framework advances the foundation laid by predecessors such as Bierbaum et al. [8], by introducing a decoupled, human-readable specification and a visual authoring tool. This makes test creation accessible to non-programmers, broadening the practical application of our approach.

IV. System overview

The system consists of three major parts: Ideal specification (in JSON format), a Visual Editor, and the Rules Engine. These parts are supported by a lightweight logging framework called VRSLOG [1], which in turn is based on a role-based meta-model for VR software to address platform fragmentation [10]. The meta-model serves as a structured foundation for building VR applications by using JSON-based specifications [11]. The logging framework provides the necessary interaction data.

A. Ideal specification

The Ideal specification is the backbone of the system. It is a declarative, human-readable data represented using JSON format that defines a valid sequence of events. The schema's design balances expressiveness with simplicity through a hierarchical structure and the use of optional fields for complex scenarios. Key sections include *settings for global tolerances, objects to define trackable entities, and a timeline array that specifies the expected event sequence*. Each event in the timeline has a type, can be associated with an object and target object, and can include temporal constraints like delay and cutoff.

A Quality Assurance engineer can potentially create the ideal specification using 3 methods:

- Automated creation using parsing of a golden run log from the VR application
- Manual creation of the specification using the visual editor interface
- Manual creation of the specification using JSON (Not preferred)

Flexibility for handling optional steps or alternative paths is achieved through several mechanisms. The prerequisites array allows for the creation of non-linear dependency chains, while the optional script field enables arbitrary validation logic for complex state-based conditions that go beyond simple event matching. Listing 1 illustrates a simple example of an ideal specification in which the user is able to see a cube and then grab it.

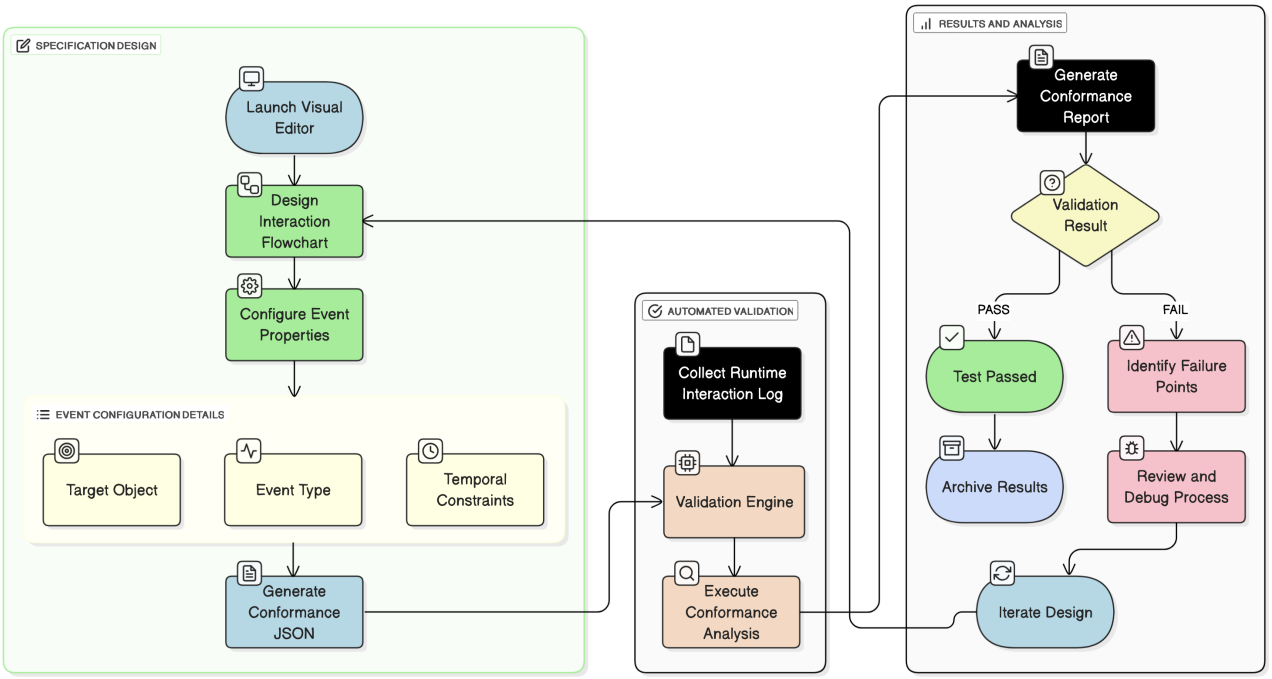

Fig. 1. Overview of VR Conformance Approach consisting of Visual Editor, Ideal Specification, and Rules Engine

```
{
"testSuite": "Basic VR Grab Test",
"version": "1.0",
"settings": {
"timeTolerance": { "value": 100, "unit": "ms" }
},
"objects": [
{ "id": "RedCube", "type": "grabbable" }
],
"timeline": [
{
"event": "Player sees red cube",
"type": "object_seen",
"object": "RedCube"
},
{
"event": "Player grabs red cube",
"type": "object_grabbed",
"object": "RedCube",
"delay": 2000,
"prerequisites": ["object_seen"]
}
]
}
```

Listing 1. An example ideal specification

B. Visual Editor

Designers/Quality assurance team members might find writing ideal specifications using JSON a bit cumbersome. To simplify the process of creating ideal specifications, we developed a web-based visual authoringt tool. This allows users to design interaction sequences as flowcharts by dragging, dropping, and connecting nodes. Each node corresponds to an event in the ideal specification, and its properties can be edited through simple UI (as shown in Figure 2). The editor provides an intuitive abstraction over the raw JSON format file, enabling instructional designers and QA testers to create complex test cases without writing code. Visual modeling is a well-established practice in software engineering for clarifying complex systems. In our context, flow visualization makes it easy to understand and debug test sequences at a glance, which is particularly beneficial for non-technical stakeholders [12].

C. Rules Engine

The rules engine is the component that performs the validation. It takes two inputs: a Conformance JSON file and a VR interaction log file. The engine's core logic is designed to be robust against the inherent variability of user behavior.

Timing Variations: The engine handles temporal variance through in two ways. A global timeTolerance can be set in the specification's settings, which applies to all events. This can be overridden on a per-event basis using the delay and cutoff fields, allowing for fine-grained control over timing constraints.

Event Matching and Filtering: The engine validates the log stream against the timeline by matching log entries to event criteria. A log entry is considered a match if its type, primary object, and target object align with the current event in the specification. Crucially, the engine is designed to handle irrelevant events. By filtering the log stream and only processing entries that match the criteria of an expected event, the system can ignore extraneous user actions that are not

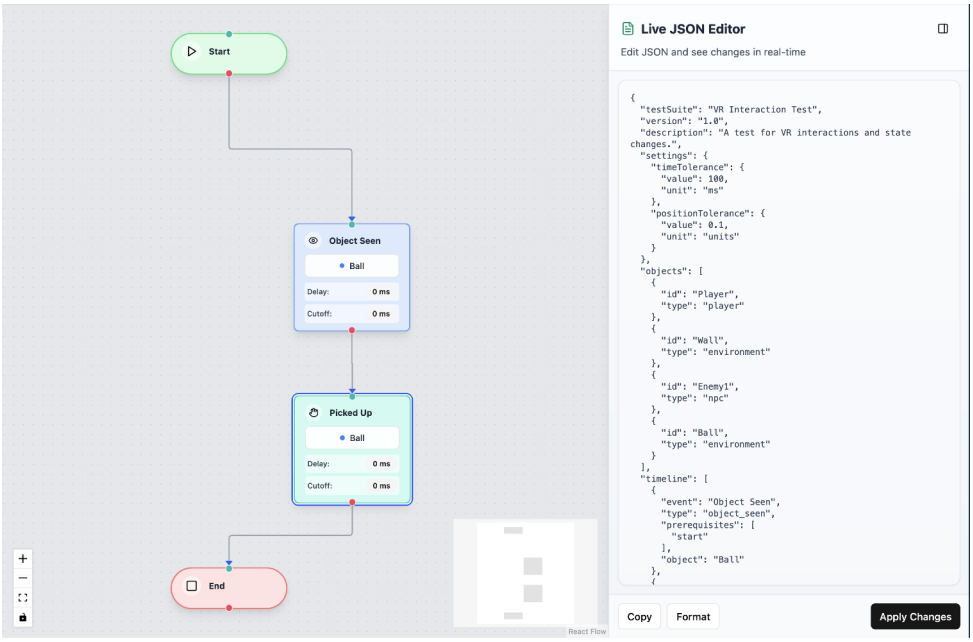

Fig. 2. Example of a sequence in the Visual Editor (left) and its x specification in JSON format (right).

part of the sequence under test. This allows VR applications to log extensively while the conformance checking system focuses only on the specified behaviors, making the approach practical for real-world scenarios.

The engine operates as a state machine, advancing to the next expected event only after its prerequisites and timing constraints are satisfied by the incoming log stream. Each timeline event represents a state transition that must be validated before proceeding. The validation algorithm operates in O(n×m) time complexity, where n represents the number of log entries, and m represents the specification events, making it suitable for real-time validation of VR interaction sequences.

D. Logging Framework

The conformance checking system relies on a stream of structured log data from the VR application. The logging framework is designed to capture key interaction details in a simple, extensible format [9]. Each log entry includes a timestamp, a human-readable event description, the primary object. The framework is platform-agnostic with current implementation for Unity and planned support for Unreal Engine.

E. Implementation

The entirety of our framework, including the visual editor, is implemented as a web application using Next.js and React. We understand the potential performance trade-off of a web-based solution compared to a native application. However, this approach offers significant advantages in terms of *cross-platform accessibility* and rapid development. The use of the React Flow library (https://reactflow.dev/) for the visual editor allowed us to create a robust and feature-rich interface. The rules engine is currently implemented in JavaScript for tight integration but is designed to be portable to other languages.

V. Demonstration Scenarios

To demonstrate the system's capability, we illustrate two example scenarios implemented in Unity for the Meta Quest platform.

Sequence 1: A non-interactive sequence where the system must display a series of instructions in a specific order and time. This demo showcases the framework's ability to validate timed, system-driven events.

This demo validates that the object follows the expected trajectory, moving in a straight line and impacting both the ground and Unity statue in sequence. The developer workflow for this specific sequence is illustrated in Figures 3 to 8.

Figure 7 illustrates that the system processed 50 log entries in just 24 milliseconds, demonstrating a high level of efficiency. In contrast, performing this analysis manually would likely require at least 1 to 2 minutes, highlighting a substantial time savings. Furthermore, the system successfully detected 100% of timing violations across corrupted test runs, showcasing its capability in identifying errors. While these exact figures may vary in different scenarios, it is reasonable to infer that automating this process offers significant advantages in speed and accuracy. Additionally, during the evaluation, memory consumption remained under 50 MB, indicating that the system is also resource-efficient.

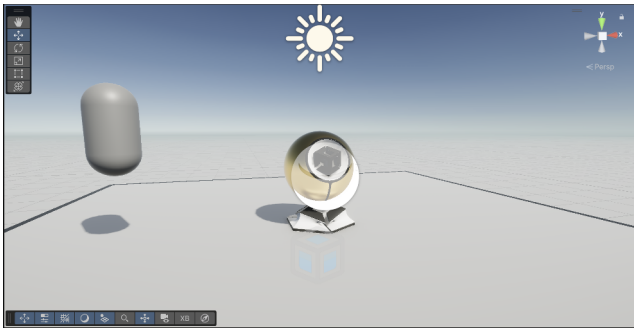

Fig. 3. VR scene in Unity showing the stationary object

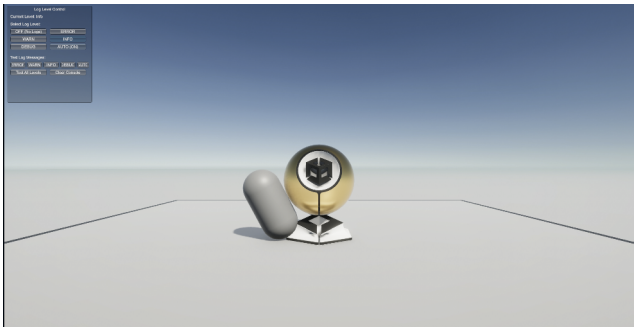

Fig. 4. VR scene in Unity showing the moving object

Sequence 2 (User-Input Driven Sequence): A training task where the user must pick up multiple objects in a set sequence and place them in designated locations before pressing a final confirmation button. This scenario is designed to test the system's ability to validate complex interactions that depend on correct user input. The ideal specification defines the required order of pickup and place events, including rules that forbid pressing the final button until all prerequisite object manipulations are logged correctly. While a full visual breakdown is not shown here for brevity and lack of space, the testing process follows the same developer workflow as illustrated in Fig. 5-8.

VI. Limitations and Future Work

Our system offers a strong foundation for VR conformance checking, but existing limitations open clear avenues for future enhancements. Addressing these will further strengthen its capabilities and broaden its impact across diverse VR testing needs.

A. Current Limitations

Evaluation Model: The system provides only binary (pass/fail) evaluation without supporting granular outcomes like "partially correct" or state-based conditions (e.g., object properties).

Limited Interaction Coverage: The current event vocabulary lacks support for rotational actions, complex gestures, and precise spatial validation beyond simple thresholds for trajectory compliance.

Extensibility Constraints: Custom scripting capabilities are underdeveloped, and the system is primarily designed for VR rather than broader XR environments.

Performance Scalability: The JavaScript implementation requires rigorous testing for large-scale VR environments and extended session durations.

B. Future work

Rigorous testing: To rigorously assess the system's effectiveness, we plan a multi-faceted evaluation targeting performance, accuracy, and usability.

- **Performance and Scalability Analysis:** We plan to conduct a series of experiments using log files of varying lengths and specifications of varying complexity. Measuring the execution time and peak memory consumption for each combination to analyze the system's scalability and identify potential performance bottlenecks is in the pipeline.
- **Accuracy Validation:** We plan to create a benchmark dataset of interaction logs from several VR scenarios. This dataset will include both "golden" (conforming) logs and logs with a variety of seeded errors, such as incorrect event sequences, missing events, and timing violations. We will process these logs with our system and compare its output against a ground truth established by manual analysis. The system's effectiveness will be measured using standard metrics.
- **Usability Study:** A user study with participants representative of our target users (e.g., QA engineers, instructional designers) is being planned. Participants shall be given a set of realistic tasks and their perceived usability shall be assessed through a standardized questionnaire.

Enhanced Paths: We plan on implementing action-based adaptive routes in the implementation, enabling complex branching paths while balancing expressiveness with non-programmer usability.

Expanded Interaction Support: We plan on building a comprehensive interaction library covering rotation, gestures, and dynamic tolerance zones for precise spatial validation in procedural training scenarios.

Improved Authoring Experience: We plan to develop a visual debugging interface with graphical deviation highlighting and side-by-side expected-vs-actual comparisons, plus a library of common interaction templates for streamlined test case creation.

Broader Applicability: A two-pronged approach will (1) develop a robust scripting API with SCORM/xAPI-compliant reporting for educational integration, and (2) extend support to AR environments, handling real-world sensor data and computer vision systems for mixed-reality validation.

Performance Optimization: Migration to a Rust-based implementation can enhance processing speed and log handling capacity for production environments.

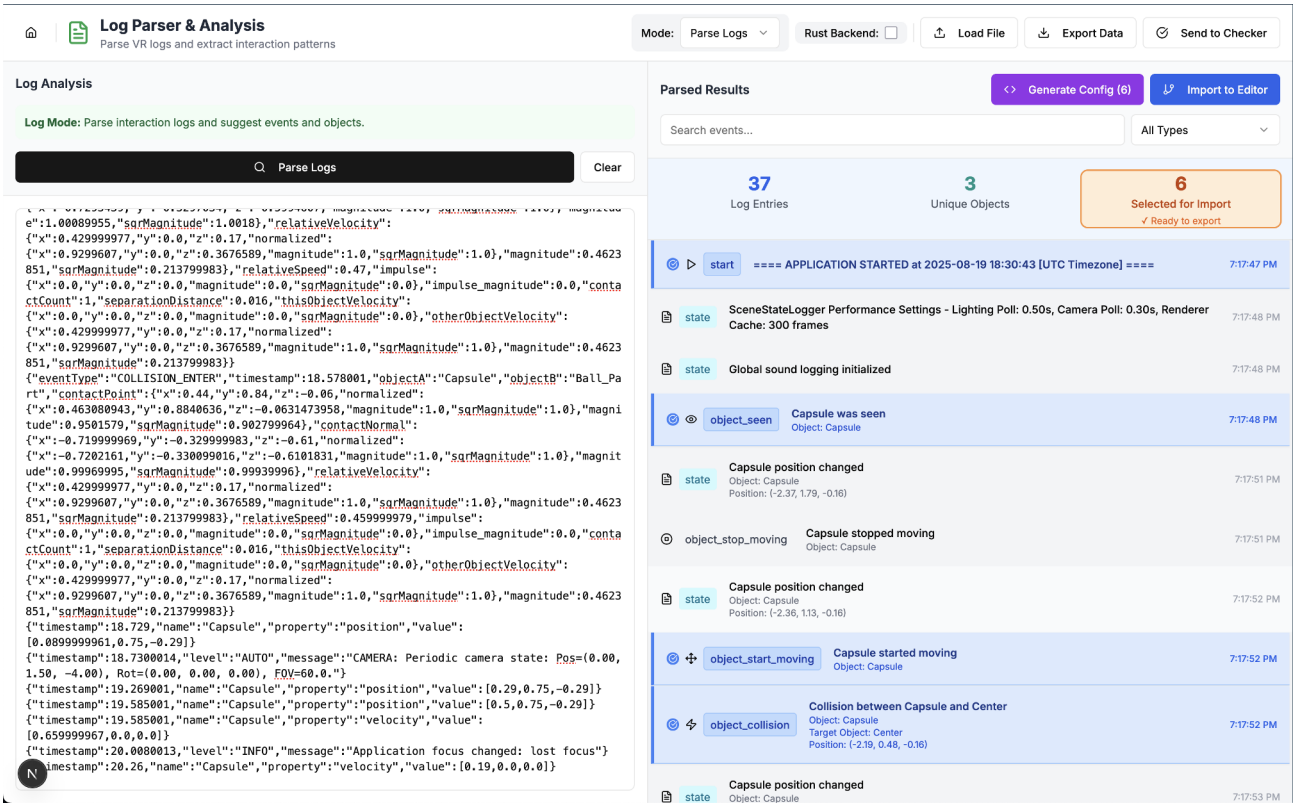

Fig. 5. Log parser: Creating a ideal specification from a golden run

VII. Conclusion

This paper presented a system for the conformance checking of interaction sequences in Virtual Reality. By combining a ideal (JSON based) specification, an intuitive visual editor, and a rules engine, our system provides a practical solution that can automate conformance checking. It empowers stakeholders to create robust, objective, and machine-verifiable test cases. While we acknowledge the current limitations, there is a clear path forward. Our future work shall focus on a more user-centric design with visual debugging and template-based authoring. Crucially, the planned integration with standards like **xAPI** will transform our system from a standalone QA tool into a vital component of the digital learning ecosystem, enabling VR training platforms to generate detailed, standards-compliant analytics on user performance.

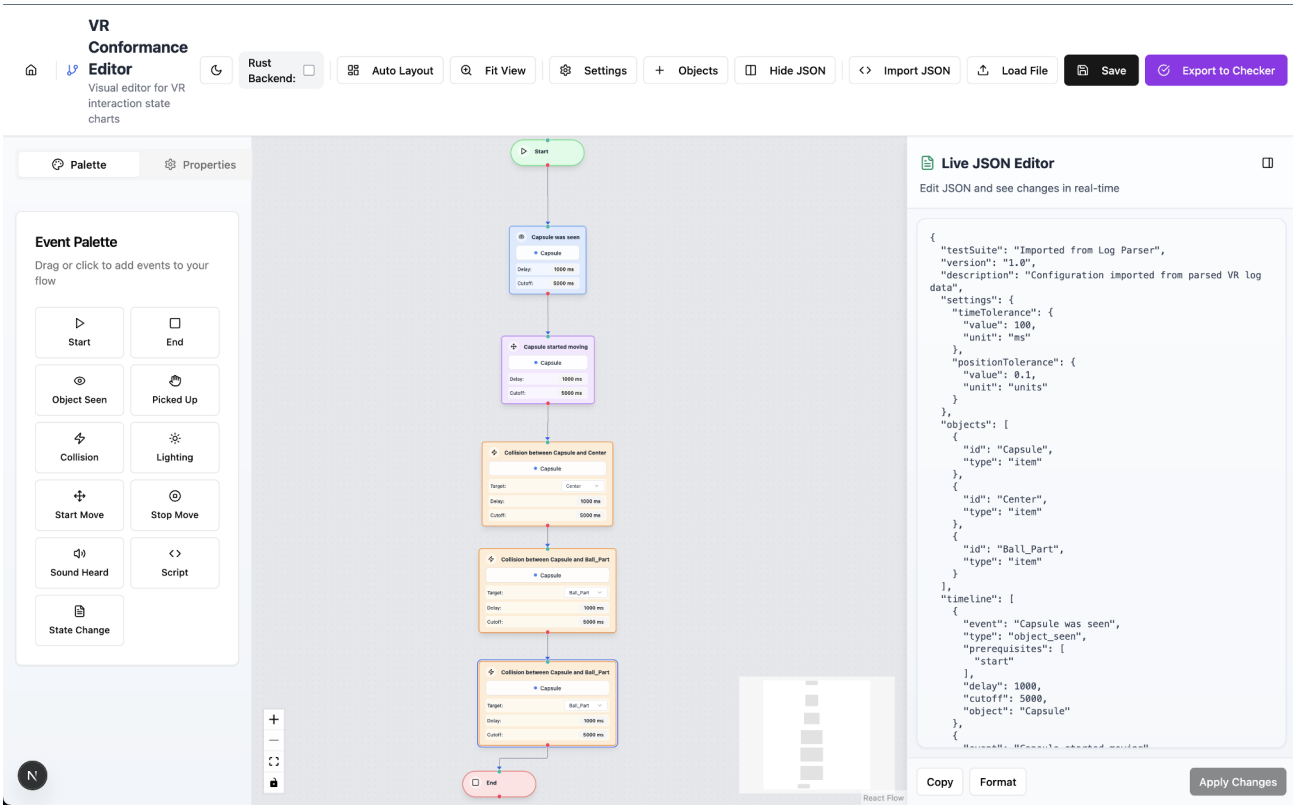

Fig. 6. Visual editor: Adjusting the flow and parameters of the specification

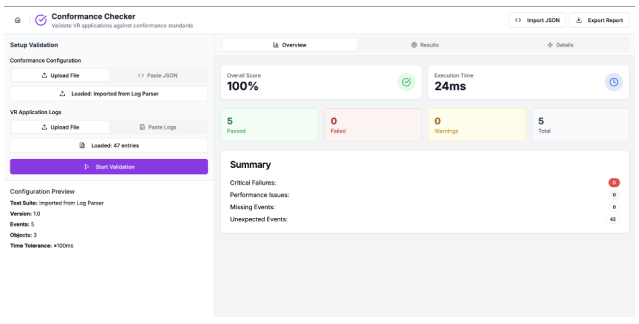

Fig. 7. Conformance tester: Passed Run

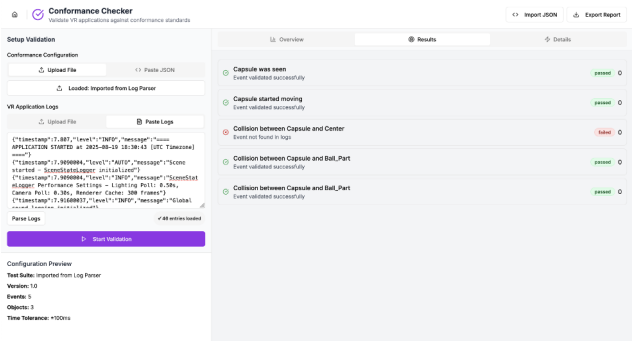

Fig. 8. Conformance tester: Failed Run

REFERENCES

[1] D. Divij, Y. R. Reddy, B. Radha, and S. Karre, "VRSLOG: An approach to log immersive experiences in virtual reality systems," in *Proc. 20th Int. Conf. Evaluation of Novel Approaches to Software Engineering (ENASE)*, 2025, pp. 229–239.

[2] W. M. P. van der Aalst, *Process Mining: Data Science in Action*, 2nd ed. Berlin, Germany: Springer-Verlag, 2016.

[3] P. C. Jorgensen, *Software Testing: A Craftsman's Approach*, 4th ed. Boca Raton, FL, USA: CRC Press, 2014.

[4] M. Speicher, F. D. M. de R. da Silva, A. T. T. Nguyen, B. T. Dvornik, and G. A. D. S. Saraiva, "A systematic review of quality assurance in virtual reality," *IEEE Trans. Vis. Comput. Graph.*, vol. 28, no. 11, pp. 4049–4065, Nov. 2022.

[5] M. Speicher, B. L. William, G. Mostajeran, and F. Mueller, "A systematic review of the state-of-the-art in automated testing for virtual, augmented, and mixed reality applications," in *Proc. IEEE Int. Symp. Mixed Augmented Reality (ISMAR)*, Singapore, 2022, pp. 638–647.

[6] P. Stauffert, F. Niebling, and M. E. Latoschik, "A taxonomy of bugs in virtual reality games," in *Proc. IEEE Int. Conf. Artif. Intell. Virtual Reality (AIVR)*, 2020, pp. 69–76.

[7] A. T. T. Nguyen, B. T. T. Nguyen, and S. H. Kim, "Automated testing for VR applications using deep reinforcement learning," in *Proc. 34th ACM/SIGAPP Symp. Appl. Comput.*, 2019, pp. 1045–1052.

[8] A. Bierbaum, P. Hartling, and C. Cruz-Neira, "Automated testing of virtual reality application interfaces," in *Proc. Eurographics Workshop Virtual Environments*, 2003, pp. 1–2.

[9] J. Flotynski and P. Sobocinski, "Logging interactions in explorable immersive VR/AR applications," in *Proc. ACM Int. Conf. Interactive Surf. Spaces*, 2019, pp. 277–280.

[10] S. A. Karre, V. Pareek, R. Mittal, and Y. R. Reddy, "A role based model template for specifying virtual reality software," in *Proc. Int. Workshop Virtual Augmented Reality Software Eng. (VARS)*, Oakland Center, MI, USA, Oct. 2022.

[11] S. A. Karre, A. A. Halhalli, and Y. R. Reddy, "VReqST: A requirement specification tool for virtual reality software products," in *Proc. Int. Conf. Software Eng.: Companion (ICSE)*, 2025, pp. 9–12.

[12] N. Mathur, S. A. Karre, and Y. R. Reddy, "Usability evaluation framework for mobile apps using code analysis," in *Proc. 22nd Int. Conf. Evaluation Assessment Software Eng. (EASE)*, New York, NY, USA, Jun. 2018, pp. 187–192.

[13] M. Waseem, P. Liang, and M. A. Shahin, "A systematic mapping study on low-code/no-code software development," *J. Syst. Software*, vol. 192, p. 111426, Oct. 2022.

[14] A. D. Johnson, A. G. Whitaker, and V. J. Shute, "xAPI for monitoring and assessing collaborative problem solving: A proof-of-concept study," *Comput. Hum. Behav.*, vol. 96, pp. 273–284, 2019.