Paper 031 / Application 008

A Mission-Grade Effects Runtime for Local-First Autonomous Systems

Making failure, authority, telemetry, and recovery explicit in JMP0X1B programs

JMP0X1B Research Group

Working paper draft / June 13, 2026

Abstract

The JMP0X1B language direction emphasizes purity by default, actor awareness, explicit effects, and distributed mission-grade systems. This paper proposes an effects runtime for local-first autonomy. Effects such as telemetry, high-voltage control, actuator command, human override, cryptographic signing, degraded-network operation, and recovery are declared, supervised, and audited.

Status. Draft manuscript for review and revision. This paper is not an empirical claim of new physics or field-ready technology. It is a structured proposal, theory note, protocol, or application architecture intended to be auditable, falsifiable, and publishable with source.

Contents

1 Problem

Autonomous systems become fragile when side effects are hidden. A command that moves an actuator, opens a high-voltage gate, sends a radio packet, updates a model, or accepts a human override should not look like ordinary computation. The problem is to make effects explicit enough for safety, replay, and certification without making programs unusable.

2 Model

The runtime separates pure decision logic from effect handlers. Actors own stateful interfaces. Supervisors control restart, degradation, and authority. Every effect has a declared failure mode, retry policy, telemetry signature, and audit behavior.

$$\text{Program} = \text{PureCore} \; + \; \sum_i \text{EffectHandler}_i \; + \; \text{Supervisor}.$$

$$\text{Risk}(e) = P(\text{fail} \mid e) \cdot C(\text{fail}, e),$$

which determines required gate strength and telemetry.

3 Claims

1. **Claim 1.** Explicit effects are a safety feature for autonomous infrastructure, not only a programming-language aesthetic.
2. **Claim 2.** Local-first systems require degraded-mode effects with bounded authority and clear synchronization semantics.
3. **Claim 3.** Replayability depends on recording effect boundaries and external observations, not pretending the world is deterministic.
4. **Claim 4.** A shared effects runtime can serve laboratories, swarms, planetary intelligence, and spacecraft subsystems.

4 Evidence Plan

A prototype should reimplement small slices of ProjectionLab, SwarmAssure, and Planetary Intelligence on a common effects runtime. Benchmarks measure failure injection, replay fidelity, recovery time, and operator comprehension of effect traces.

5 JMP0X1B Implementation Surface

This is a core JMP0X1B technology paper. The language can start with typed contracts and later add compiler/runtime enforcement. Libraries should declare effect capabilities and test degraded behavior.

Illustrative JMP0X1B-style contract

```
effect HighVoltage { arm: HVProfile -> Gate, disarm: Unit -> Receipt }
effect DegradedNetwork { send: Packet -> DeliveryStatus, sync: DigestSet -> SyncReport }
fn decide(frame: WorldFrame) -> Command effects{Policy} // pure except declared policy effect
```

6 Release And Review Plan

Release effect taxonomy, runtime interface, failure-injection examples, and conformance tests for packages that claim mission-grade behavior.

7 Open Questions

- Which effects deserve language-level syntax and which are library contracts?
- How should effect authority be delegated and revoked?
- Can replay remain useful when sensors are nondeterministic?
- What minimal runtime can run on constrained edge devices?

8 Conclusion

A mission-grade effects runtime turns JMP0X1B principles into deployable infrastructure. It is the software foundation beneath the application papers.

References

References

[1] JMP0X1B. *Technology for Earth, designed with space as the operating condition.* `https://jmp0x1b.com/`

[2] JMP0X1B. *jmp0x1b Documentation Portal.* `https://lang.jmp0x1b.com/latest/site/index.html`

[3] JMP0X1B. *jmp0x1b_lang_spec package.* `https://libs.jmp0x1b.com/packages/jmp0x1b_lang_spec`

[4] C. Hewitt, P. Bishop, and R. Steiger. A universal modular actor formalism for artificial intelligence. *Proceedings of IJCAI*, 1973.

[5] L. Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers.* Addison-Wesley, 2002.

[6] JMP0X1B. *jmp0x1b_provenance package.* `https://libs.jmp0x1b.com/packages/jmp0x1b_provenance`