
INTENT REALIZATION FIDELITY: VERIFYING THAT SYSTEMS DO WHAT THEY CLAIM *

Michael J. Curnow II (Mike)
C6 Project
RTP NC, USA

ABSTRACT

Modern systems routinely declare intent to perform actions and subsequently report successful execution through logs, status indicators, or telemetry. In practice, such self-reported success is often accepted as evidence that the intended action was realized. Across physical, cyber-physical, and purely digital systems, this assumption frequently fails: internal execution and reported success may diverge from what actually occurs in the world.

This paper introduces *Intent Realization Fidelity (IRF)*, a system property describing the degree to which declared intent is faithfully realized at the lowest level where its effects materially manifest, as verified through independent evidence rather than self-reporting. IRF explicitly distinguishes between intent, internal execution, and external realization, and reframes verification around observable effects rather than internal claims of success.

We show that existing approaches, including logging, monitoring, runtime attestation, anomaly detection, and condition monitoring, address adjacent concerns but do not verify faithful realization of intent. Through illustrative examples spanning cyber-physical and non-physical systems, we demonstrate how reliance on self-reported execution obscures realization failures. We argue that treating Intent Realization Fidelity as a first-class design consideration is necessary for building systems that can meaningfully verify what they do, rather than merely what they claim to have done.

Keywords Intent realization, execution verification, system trust, independent evidence, cyber-physical systems, assurance

1 Introduction

Modern systems routinely declare intent to perform actions. These intents may originate from human operators, automated control logic, configuration changes, scheduled tasks, or higher-level orchestration systems. In response, systems commonly report successful execution through logs, status messages, acknowledgments, or telemetry. In practice, such self-reported success is frequently accepted as evidence that the intended action was carried out.

This assumption, that declared intent followed by no reported error implies successful realization, is deeply embedded in systems design. Yet it is rarely verified.

Across physical, cyber-physical, and purely digital systems, reality often diverges from reported execution. Mechanical components fail to actuate despite acknowledged commands. Automated jobs complete nominally while producing no meaningful effects. Distributed services report success while downstream actions never materialize. In each case, the system's internal narrative remains consistent, even as the external world contradicts it.

The gap between declared intent and realized outcome represents a fundamental trust assumption: execution is inferred from internal reporting rather than corroborated by independent evidence.

**Citation: Michael J. Curnow II. Intent Realization Fidelity: Verifying That Systems Do What They Claim. Public preprint, 2026. Available at <https://doi.org/10.5281/zenodo.18166827>*

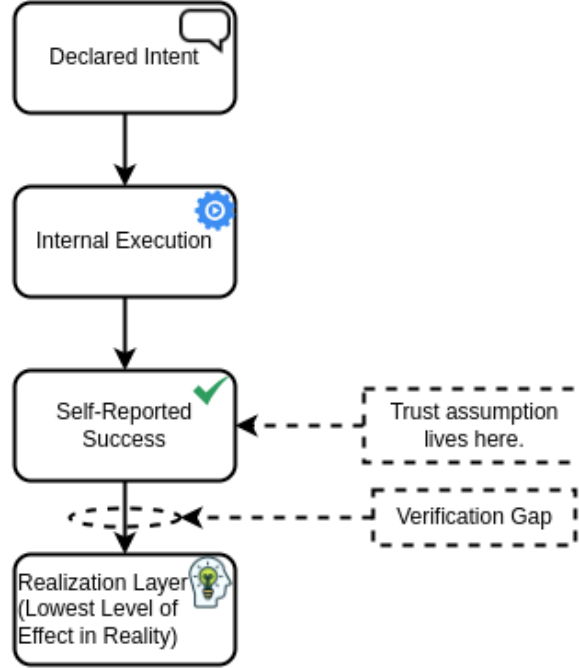


Figure 1: Declared intent and self-reported execution are not sufficient evidence of realized outcome.

Failures and adversarial actions routinely exploit this assumption. Faults, misconfigurations, degraded components, and malicious interference can all cause actions to be partially realized, incorrectly realized, or not realized at all, while upstream systems continue to report success. Because most architectures lack a principled way to verify that intent has been faithfully realized, such discrepancies often go undetected until their consequences become unavoidable.

Importantly, this problem is not confined to any single domain. In industrial control systems, the effects manifest physically through actuators and processes. In cyber-physical and robotic systems, they appear as mismatches between commanded and observed motion or behavior. In software and distributed systems, they arise when claimed execution does not correspond to observable side effects in the environment. Despite differences in implementation and context, these systems share a common architectural blind spot: they trust execution claims without independently verifying realization.

This paper argues that the absence of independent verification of realized outcomes is not merely an implementation oversight, but a missing system property. We introduce *Intent Realization Fidelity (IRF)*, defined as the degree to which declared intent is faithfully realized at the lowest level where its effects materially manifest, as verified through independent evidence rather than self-reporting. We show that common approaches, including logging, monitoring, runtime attestation, anomaly detection, and health diagnostics, do not provide IRF because they either rely on self-reported execution or observe behavior without explicitly verifying correspondence between intent and realized outcome. We argue that meaningful verification must extend to the lowest level at which reality enforces truth, whether physical, logical, or environmental.

By treating Intent Realization Fidelity as a first-class design concern, systems can move beyond trust-by-assumption toward evidence-based verification of what they actually do, rather than what they claim to have done.

2 Motivating Examples (Cross Domains)

The trust gap addressed by Intent Realization Fidelity appears across diverse system classes. While implementations differ, these systems share a common pattern: intent is declared, execution is reported, and realization is assumed. The following examples illustrate how this assumption fails in practice, even when logs and telemetry indicate success.

2.1 Industrial Control

An operator issues the command to open a valve in a control system. The control logic executes as expected, and the system reports successful actuation. Status indicators reflect the commanded state, and no fault is raised.

In reality, the valve never moves. A mechanical failure, miswired actuator, or obstructed linkage prevents physical motion. From the system's perspective, the command was executed successfully; from the process perspective, nothing changed.

Here, execution occurred at the logical level, but realization failed at the physical level. The discrepancy remains invisible until downstream effects are observed.

2.2 Automation and Scheduling

A scheduled automation task is configured to run periodically, such as a backup or data processing job. At the scheduled time, the task launches, completes, and records a successful status. Logs confirm execution, and monitoring systems observe no errors.

However, no meaningful side effects occur. No data is written, no files are produced, and no external systems are affected. The job ran, but its intended outcome was never realized due to misconfiguration, missing dependencies, or silent failure conditions.

In this case, intent was declared and execution occurred, yet realization was absent. System telemetry remains internally consistent while external reality contradicts it.

2.3 Human-in-the-Loop Systems

A human operator activates an emergency stop mechanism in response to unsafe conditions. The system acknowledges the input, records the event, and signals that the stop has been triggered.

Despite this acknowledgment, the machine continues moving. A failed relay, degraded contact, or bypassed safety circuit prevents the stop signal from reaching the actuators. From the interface's perspective, the intent was received and handled correctly; from the physical system's perspective, the intent was never realized.

This example highlights a critical failure mode in which acknowledgment of intent is mistaken for assurance of effect.

2.4 Distributed Systems

A management system issues a request to deploy a new resource, such as a virtual machine or service instance. The orchestration layer reports successful provisioning, and associated logs reflect completion. Upstream systems proceed under the assumption that the resource exists.

In reality, the resource was never instantiated. A failure in a downstream component, misreported status, or partial execution leaves the system in a state where the declared outcome does not exist. Subsequent operations fail in ways that obscure the original discrepancy.

Here, distributed execution masks realization failure behind layers of abstraction and self-reporting.

2.5 Common Pattern

Across these examples, the pattern is consistent:

- Intent is declared and processed.
- Execution is reported as successful.
- Logs and telemetry agree.
- Independent evidence of realization is absent or contradictory.

The systems involved do not lack observability or monitoring; they lack a principled way to verify that declared intent was faithfully realized at the point where its effects materially manifest. This gap motivates the need for a system property that explicitly addresses the correspondence between intent and reality.

3 Problem Statement

The examples in section 2 illustrate a recurring failure mode across system classes: declared intent and reported execution do not reliably imply that an intended action was realized. To reason about this gap, we first clarify what is meant by intent and realization, and then examine the trust assumption that links them in most system architectures.

3.1 Intent

In this work, intent refers to a declared expectation that a system perform a specific action. Intent may originate from multiple sources and take multiple forms, but in all cases it represents an assertion about what the system is expected to cause.

Intent may be:

- Human-originated, such as an operator command, manual input, or physical interaction.
- Automated, produced by control logic, orchestration, or decision-making systems.
- Scheduled, arising from time-based triggers such as cron jobs, timers, or policy driven execution.
- Configured, implied by system configuration, policy settings, or static declarations of desired state.

Intent may also be explicit or implicit. Explicit intent is directly expressed, for example through a command or an API call. Implicit intent arises from configuration state, where the system is expected to maintain or converge toward a specified condition without continuous direct commands.

Regardless of origin or form, intent expresses an expectation about future system behavior. Importantly, intent itself does not guarantee that any corresponding action has occurred or will occur.

3.2 Realization

Realization refers to the manifestation of intent in the world, the point at which an intended action produces observable effects beyond internal computation.

A critical distinction exists between:

- Internal execution, where software logic runs, control paths are taken, or internal state transitions occur.
- External manifestation, where the effects of an action become observable in physical, logical, or environmental reality.

Many systems equate execution with realization. In practice, internal execution may complete successfully while external manifestation fails, is partial, or is materially different from what was intended.

We therefore define the lowest relevant level of effect as the deepest point in a system's interaction with reality at which the intended action produces consequences that cannot be abstracted away. This level depends on the system domain:

- For physical systems, it may be mechanical motion, force, pressure, or position.
- For cyber-physical systems, it may be sensor-observable behavior or environmental change.
- For purely digital systems, it may be a durable state change, external communication, or resource instantiation.

At this level, discrepancies between intent and outcome are no longer masked by internal abstraction layers. Reality enforces truth.

3.3 The Trust Assumption

Most system architectures implicitly rely on the assumption that declared intent combined with the absence of reported error implies successful realization. This assumption is rarely stated explicitly, yet it underlies common design practices across domains. Systems treat logs, acknowledgments, status flags, and telemetry as sufficient evidence that an intended action has occurred. Once intent is processed and no error is raised, realization is inferred. This assumption is false.

Execution paths may be complete without producing effects. Physical components may fail silently. Dependencies may be missing or degraded. Downstream subsystems may misreport status or fail independently. In adversarial contexts, systems may intentionally report success while suppressing or altering outcomes.

Crucially, many of these failures do not violate internal correctness conditions. Logs remain consistent. Control flow behaves as expected. Monitoring systems observe no anomalies. The system’s internal narrative agrees with itself even if it diverges from reality.

Because realization is inferred rather than verified, discrepancies between intent and outcome often persist undetected until secondary effects emerge. By that point, corrective action may be delayed, costly, or impossible.

This work argues that the absence of an explicit mechanism to reason about the correspondence between intent and realization represents a fundamental gap in system assurance. Addressing this gap requires treating realization as something that must be independently verified, rather than assumed.

4 Intent Realization Fidelity (IRF)

The problem described in Sections 2 and 3 arises from a fundamental omission in how systems reason about action and effect. While systems are designed to declare intent, execute logic, and report success, they generally lack a principled way to verify that an intended action was actually realized in the world. To address this gap, we introduce Intent Realization Fidelity (IRF) as a first-class system property.

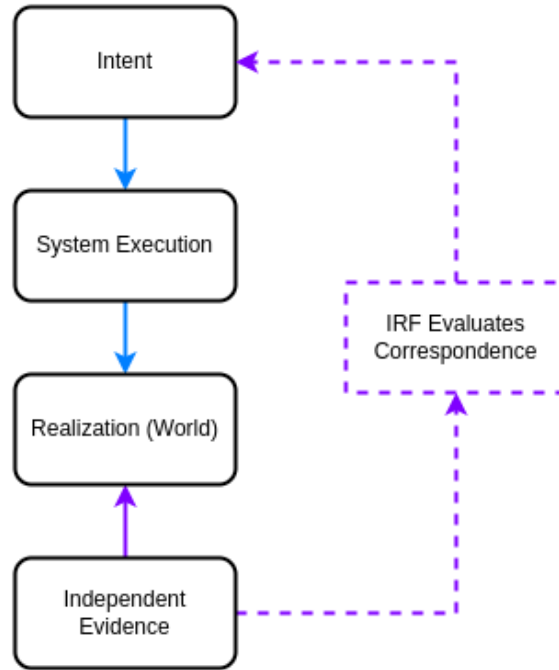


Figure 2: IRF evaluates correspondence between declared intent and realized effects using independent evidence, rather than self-reporting.

IRF formalizes the relationship between declared intent and realized outcome, shifting verification from internal self-reporting to externally grounded evidence.

4.1 Definition

Intent Realization Fidelity (IRF) is the degree to which a declared intent is faithfully realized at the lowest level where its effects materially manifest, as verified through independent evidence rather than self-reporting.

Several aspects of this definition are intentional.

First, fidelity is not binary. Realization may be complete, partial, delayed, degraded, or incorrect. A system may successfully execute internal logic while only partially realizing the intended effect, or realizing it in a manner that

deviates materially from intent. IRF captures these distinctions by describing degree, rather than treating realization as a Boolean condition.

Second, IRF is explicitly concerned with realization, not execution. Internal execution paths, control flow, and state transitions may complete without producing the intended external effects. IRF evaluates realization only at the point where those effects manifest in reality, beyond the reach of abstraction layers.

Finally, IRF requires independent evidence. Claims of success derived solely from the declaring or executing system are insufficient. Verification must rely on evidence that is causally downstream of intent and observable outside of the system's internal narrative.

Together, these constraints distinguish IRF from existing notions of correctness, health, or success.

4.2 Properties of IRF

IRF is characterized by four core properties that define what it means to verify realization meaningfully.

Independence

Evidence used to establish IRF must not originate solely from the system that declares or executes intent. Logs, acknowledgments, return codes, and internal telemetry reflect the system's own account of execution and therefore cannot, by themselves, establish realization.

Independent evidence may still be instrumented, collected, or processed by system components, but it must reflect effects that cannot be trivially fabricated by the declaring logic alone.

Causality

Evidence must be causally downstream of the declared intent. That is, the observed evidence must arise as a consequence of the intended action, not merely coincide with it. Evidence that precedes intent, or that can occur independently of it, does not contribute to IRF.

Causality ensures that verification reflects realization rather than correlation.

Groundedness

Verification must extend to the lowest level where effects manifest. At this level, abstraction collapses and discrepancies between intent and outcome cannot be hidden by intermediate representations.

What constitutes this level depends on the system domain. It may be physical motion, environmental change, durable state mutation, or externally observable interaction. Regardless of domain, IRF requires verification to reach the point where reality enforces truth.

Corroboration

IRF is strengthened through multiple, independent sources of evidence. No single modality is assumed to be perfectly reliable. Corroboration across diverse evidence sources reduces ambiguity, mitigates noise, and increases confidence that realization has occurred as intended.

Importantly, corroboration does not require redundancy in mechanism; it requires diversity in failure modes. Evidence that shares a common dependency or failure domain contributes less to IRF than evidence derived from independent phenomena.

4.3 What IRF is Not

To avoid confusion, it is important to distinguish IRF from related but insufficient concepts.

IRF is not logging. Logs record declared actions and execution paths, but they do not independently verify that intended effects occurred.

IRF is not monitoring. Monitoring observes system behavior but does not by itself, establish correspondence between declared intent and realized outcome.

IRF is not anomaly detection. A system may behave normally and still fail to realize intent. Conversely, anomalous behavior does not necessarily imply failed realization.

IRF is not fault detection. Faults are one cause of realization failure, but IRF is concerned with verification of outcome regardless of cause.

IRF is not control or enforcement. It does not prevent failures or adversarial actions; it verifies whether intent was realized.

IRF is not safety certification. While it may support safety objectives, IRF is a verification property, not a guarantee of safe operation.

In essence, IRF addresses a different question than these approaches. Rather than asking whether a system is healthy, compliant, or behaving normally, IRF asks a simpler and more fundamental question:

Did the system actually do what it claimed to do?

5 Evidence and Verification

Intent Realization Fidelity (IRF) requires that claims of execution be evaluated against evidence of realization. This section describes the nature of such evidence and how it may be used to verify IRF across domains without constraining implementations to specific technologies, sensors, or architectures.

Rather than prescribing a single mechanism, IRF defines requirements on the quality and relationships of evidence to intent.

5.1 Independent Evidence

Independent evidence refers to observable effects that arise as a consequence of intent realization and are not derived solely from the internal state or self-reporting mechanisms of the system declaring intent.

Such evidence may include, but is not limited to:

- Physical signals, such as motion, force, current, vibration, or pressure.
- Environmental effects, including temperature changes, sound, light, or environmental state transitions.
- State changes, such as durable data modification, resource instantiation, or externally visible configuration updates.
- Observable side effects, including network communication, file creation, process execution, or interactions with external systems.

What qualifies as independent evidence depends on the system context, but the defining characteristic is that the evidence reflects effects that cannot be fully asserted by the declaring system alone. Evidence that merely echoes internal execution paths, logs, or acknowledgments does not satisfy this requirement.

Independence does not imply isolation from the system. Evidence may be collected by instrumentation, sensors, or observers that are integrated into the system architecture, provided that the observed effects are causally downstream of the intent and materially grounded in the system's interaction with reality.

5.2 Modalities

Evidence used to establish IRF may be drawn from multiple modalities, each corresponding to a different class of observable phenomena. Modalities reflect distinct physical, logical, or temporal domains in which realization may manifest.

Common modalities include:

- Electrical, such as voltage, current, signal transitions, or power consumption.
- Mechanical, includes motion, vibration, force, or structural response.
- Thermal, such as temperature change or heat dissipation.
- Kinematic, including position, velocity, orientation, or spatial displacement.
- Logical, encompassing durable state changes, data persistence, or externally observable software behavior.
- Temporal, such as timing relationships, duration of activity, or ordering of events.

No single modality is universally sufficient. Different systems expose realization through different phenomena, and some realizations may be more readily observable in one modality than another. IRF therefore does not privilege any specific modality; instead, it requires that at least one modality provide evidence that is both causally linked to intent and grounded at the lowest relevant level of effect.

5.3 Multi-Modal Corroboration

In practice, single-modality verification is rarely sufficient to establish high confidence in intent realization. Individual evidence sources may be noisy, ambiguous, delayed, or subject to failure or manipulation. Relying on a single modality risks mistaking coincidence, partial execution, or fabricated signals for faithful realization.

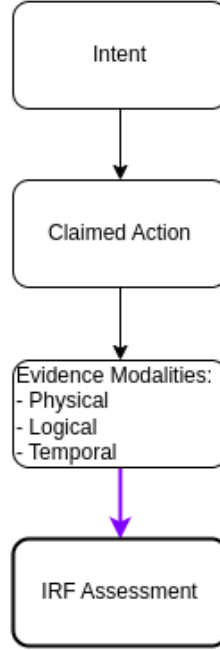


Figure 3: Multi-modal corroboration improves confidence by combining evidence sources with diverse failure domains and time scales.

IRF is strengthened through multi-modal corroboration, where evidence from multiple, independent modalities is evaluated collectively. Corroboration offers several advantages:

- **Complementary time scales:** Some modalities reflect realization immediately, while others manifest over longer durations. Combining fast and slow signals enables detection of both instantaneous failures and gradual degradation.
- **Diverse failure modes:** Modalities governed by different physical or logical principles are unlikely to fail or be manipulated in the same way simultaneously.
- **Reduced spoofability:** An adversary or fault capable of influencing one modality may not be able to convincingly influence others, particularly when they originate from distinct domains.

Crucially, corroboration is not a matter of redundancy alone. Multiple evidence sources that share a common dependency or failure domain do not meaningfully increase confidence. Effective corroboration depends on diversity of evidence, not quantity.

IRF does not require exhaustive evidence collection. Instead, it emphasizes sufficiency: enough independent, causally linked evidence to establish that realization occurred in a manner consistent with declared intent.

6 Instantiations

Intent Realization Fidelity is intentionally defined independent of any specific system architecture or domain. Nevertheless, its utility becomes clearest when examined through concrete instantiations. This section presents illustrative examples spanning cyber-physical and non-physical systems to demonstrate how IRF manifests in practice, without prescribing implementations or exhaustively cataloging use cases.

The purpose of these instantiations is not to enumerate all possible applications, but to show that the same verification principle applies across fundamentally different system classes.

6.1 Cyber-Physical Systems (Illustrative)

In cyber-physical systems, intent is often expressed through commands or control logic, while realization manifests as physical effects in the environment. These systems expose a clear separation between internal execution and external manifestation, making them a natural illustration of IRF.

Consider a system in which an intent is declared to actuate a physical component. The control logic executes, and internal indicators reflect the commanded state. From the system's internal perspective, execution has succeeded.

Realization, however, occurs only when the physical system responds. The lowest level of realization may be a mechanical motion, applied force, fluid flow, or positional change. At this level, discrepancies between intent and outcome cannot be abstracted away. A component that fails to move, moves partially, or moves incorrectly represents a failure of realization, regardless of internal execution status.

Independent physical evidence, such as motion, pressure, vibration, or environmental change, provides a basis for verifying realization. When such evidence contradicts the declared outcome, a mismatch between intent and realization is revealed. Importantly, this mismatch may arise from benign faults, degradation, misconfiguration, or adversarial interference. IRF is agnostic to cause and focused solely on verification.

In this context, IRF exposes failures that remain invisible to systems that rely solely on internal state, control flow, or reported success.

6.2 Non-Physical Systems

IRF is equally applicable to systems in which realization does not involve physical motion, but instead manifests as externally observable logical or environmental effects.

Software Execution In software systems, intent may take the form of a command to execute a task, apply a configuration change, or perform a computation. Execution is typically confirmed through return values, logs, or status codes.

Realization, however, occurs only when the intended effects materialize, for example when data is durably written, resources are created, or observable outputs are produced. A task may run and report success while producing no meaningful side effects. In such cases, internal execution occurred, but realization did not.

IRF distinguishes between these outcomes by requiring verification against evidence that reflects actual state change beyond the execution context.

Distributed Systems Distributed systems further complicate the relationship between intent and realization. Intent is often declared at a high level, while realization depends on multiple downstream components operating correctly. Partial execution, inconsistent state propagation, or misreported status can result in systems claiming success even when intended outcomes never materialize.

Here, IRF emphasizes verification at the lowest level where the intended effect should be externally observable, such as the presence of a deployed resource, the availability of a service endpoint, or durable interaction with external systems. Evidence refined to orchestration layers or control planes insufficient to establish realization.

Automation Pipelines Automation systems commonly express intent through scheduled or policy-driven execution. Tasks may launch, complete, and record success without producing their intended results due to missing dependencies, environmental drift, or silent failures.

In these cases, IRF requires verification through observable consequences of execution, rather than acceptance of nominal completion. A pipeline that runs but produces no output, effects, or downstream interactions exhibits low realization fidelity despite reporting success.

6.3 Summary

Across these instantiations, the pattern is consistent:

- Intent is declared and processed.
- Execution may occur internally.
- Realization may succeed, partially succeed, or fail entirely.

- Self-reporting alone is insufficient to distinguish these outcomes.

IRF provides a unifying lens for reasoning about these discrepancies by shifting verification to the point where effects manifest and reality enforces truth. Whether the system is physical or purely logical, IRF applies wherever intent is assumed to imply outcome without independent verification.

7 Relationship to Prior Work

A wide range of prior work addresses correctness, integrity, monitoring, and fault detection in complex systems. These approaches provide important guarantees and capabilities, but they largely focus on execution correctness or system behavior, rather than on verifying that declared intent was faithfully realized at the point where its effects manifest. This section situates Intent Realization Fidelity within that landscape and clarifies how it differs from existing approaches.

7.1 Runtime Attestation

Runtime attestation mechanisms aim to verify that a system executes authorized code paths or maintains expected internal state. Techniques such as control-flow integrity, trusted execution environments, and runtime measurement seek to ensure that execution has not been tampered with.

While these approaches strengthen confidence in how a system executes, they do not establish whether execution produced the intended external effects. A system may faithfully execute approved logic while failing to realize the declared intent due to misconfiguration, missing dependencies, downstream failures, or interference beyond the execution boundary.

Runtime attestation verifies internal execution integrity; IRF verifies external realization of intent. These guarantees are complementary but not equivalent.

7.2 Anomaly Detection

Anomaly detection techniques identify deviations from expected behavior by modeling normal system operation and flagging outliers. These methods are widely used across cyber-physical, industrial, and software systems to detect faults, misbehavior, or attacks.

However, anomaly detection does not explicitly reason about declared intent. A system may behave normally according to its learned or modeled baseline while still failing to realize a specific intended action. Conversely, a system may realize intent correctly while exhibiting behavior deemed anomalous.

IRF does not seek to characterize normal or abnormal behavior. Instead, it evaluates whether a specific intent was realized, regardless of whether the resulting behavior is typical.

7.3 Process Monitoring

Process monitoring approaches observe system or environmental state to infer correctness, safety, or efficiency. In cyber-physical contexts, such monitoring often relies on physical measurements or process invariants to detect inconsistencies.

While process monitoring leverages physical reality, it typically focuses on system state consistency, and not on correspondence between declared intent and realized outcome. Monitoring may reveal that a process is in an unexpected state without establishing whether a particular intent was fulfilled, partially fulfilled, or never realized.

IRF differs by anchoring verification to declared intent and evaluating realization explicitly, rather than inferring correctness solely from observed state.

7.4 Logging and Auditing

Logs and audit trails record system activity, execution paths, and decision points. They are foundational to observability, debugging, and post-incident analysis.

However, logs primarily reflect the system's internal account of events. Even when logs are tamper-resistant, or cryptographically protected, they remain a form of self-reporting. A system can log that an action was taken without that action being realized in the world.

IRF explicitly rejects logs and audit records as sufficient evidence of realization. While such records may contribute contextual information, they cannot independently establish that intent was faithfully realized.

7.5 Condition Monitoring

Condition monitoring techniques assess the health of operation status of components by analyzing signals such as vibration, temperature, or performative metrics. These approaches are commonly used for diagnostics and predictive maintenance.

Condition monitoring identifies how components behave, not why they behave that way. It does not, by itself, establish whether a particular intended action occurred. A components may appear healthy while failing to realize intent, or may realize intent despite degraded condition.

IRF may leverage similar evidence sources, but it repurposes them to answer a different question: whether declared intent resulted in the intended effect.

7.6 Summary

Across these approaches, a common pattern emerges. Prior work provides mechanisms to verify execution integrity, detect abnormal behavior, monitor system state, or assess component health. These capabilities are essential, but they share an implicit assumption: that declared intent and reported execution are sufficient proxies for realization.

Intent Realization Fidelity addresses a different problem. Rather than verifying how a system executes or how it behaves, IRF verifies whether a declared intent was faithfully realized at the lowest level where effects manifest, using independent evidence rather than self-reporting.

By making this distinction explicit, IRF complements existing approaches while addressing a gap they do not fill.

To our knowledge, no prior work explicitly defines or treats the correspondence between declared intent and realized outcome as a first-class verification property independent of execution integrity or behavioral correctness.

8 Limitations and Open Questions

Intent Realization Fidelity addresses a fundamental gap in how systems reason about action and effect. However, like any verification property, IRF is subject to practical and theoretical limitations. Recognizing these limitations is essential for understanding both the scope of IRF and the challenges involved in applying it to real systems.

8.1 Partial Observability

In many systems, the lowest level at which realization occurs may not be fully observable. Physical effects may be obscured, inaccessible, or only indirectly measurable. In purely digital systems, relevant side effects may be transient, distributed, or hidden behind additional layers of abstraction.

Partial observability limits the ability to establish IRF conclusively. Absence of evidence does not necessarily imply absence of realization, particularly when sensing or observation is incomplete. IRF therefore operates under bounded knowledge: it evaluates realization based on available evidence rather than processing omniscience.

Determining which levels of effect are observable, and to what degree, remains an open challenge that is highly context-dependent.

8.2 Evidence Trustworthiness

IRF relies on independent evidence, but independence does not imply inherent trustworthiness. Evidence sources themselves may be faulty, miscalibrated, degraded, or compromised. Sensors may drift, logs of external systems may be incorrect, and observers may be subject to interference.

As a result, IRF cannot eliminate the need to reason about the trustworthiness of evidence. Instead, it shifts the trust boundary outward, from internal self-reporting to externally observable effects. Establishing confidence in evidence sources, and reasoning about their failure modes, remains a necessary component of any IRF instantiation.

8.3 Cost and Complexity

Verifying realization through independent evidence introduces additional cost and complexity. Instrumentation, sensing, data collection, and analysis all require resources. In some systems, the overhead of acquiring sufficient evidence may be prohibitive or may introduce latency incompatible with operational requirements.

IRF does not prescribe universal verification of all intents. Practical deployments must balance verification depth, coverage, and cost. Determining which intents warrant verification, at what granularity, and under what conditions remains an open design decision.

8.4 Adversarial Evidence Manipulation

While IRF reduces reliance on self-reporting, it does not make systems immune to adversarial manipulation. An adversary may attempt to fabricate, suppress, or distort evidence to create the appearance of realization where none occurred, or to mask partial or incorrect realization.

Multi-modal corroboration increases the difficulty of such manipulation, but does not guarantee resistance. Evidence sources may share dependencies or be subject to coordinated attack. Understanding adversarial models, identifying shared failure domains, and designing evidence diversity are critical challenges for robust IRF implementations.

8.5 Defining "Sufficient Fidelity"

IRF characterizes realization as a matter of degree rather than a binary condition. This raises the question of what constitutes sufficient fidelity for a given intent. Partial realization, delayed effects, or degraded outcomes may be acceptable in some contexts and unacceptable in others.

Determining appropriate thresholds for fidelity is inherently application-specific and may depend on safety, security, performance, or economic considerations. IRF provides a framework for reasoning about realization, but it does not dictate normative judgments about adequacy.

Formalizing fidelity thresholds and decision criteria remains an open area for future work, and should be considered domain and context dependent.

8.6 Summary

These limitations do not undermine the value of Intent Realization Fidelity. Rather, they clarify its role. IRF does not promise perfect verification, nor does it eliminate uncertainty. Instead, it provides a principled way to reason about the correspondence between intent and outcome using evidence grounded in reality, rather than assumption.

By making the verification problem explicit, IRF enables systems to confront uncertainty directly instead of silently trusting execution claims.

9 Implications

Intent Realization Fidelity has implications that extend beyond any single domain or implementation. By making explicit the distinction between declared intent, internal execution, and realized outcome, IRF challenges long-standing assumptions in how systems are designed, secured, and assured. This section discusses the broader consequences of treating IRF as a first-class property.

9.1 Systems Design

Most systems are designed around the assumption that successful execution implies successful outcome. As a result, architectures prioritize correctness of logic, reliability of components, and observability of internal state, while leaving realization largely implicit.

IRF reframes this approach. Systems designed with IRF in mind must explicitly consider where and how intended effects manifest, and how those effects can be verified independently. This encourages designs that expose meaningful points of verification, reduce reliance on self-reporting, and acknowledge that abstraction boundaries do not eliminate the need for evidence.

Incorporating IRF into system design shifts emphasis from doing things to demonstrating that things were actually done.

9.2 Security Architecture

From a security perspective, IRF highlights a class of failures that traditional controls often miss. Many security mechanisms focus on preventing unauthorized actions or ensuring correct execution paths. Fewer mechanisms verify that authorized actions were actually realized as intended.

Attackers and adversarial conditions frequently exploit this gap by allowing systems to appear correct while suppressing, altering, or degrading outcomes. IRF provides a lens for detecting such discrepancies by anchoring verification in externally observable effects rather than internal claims.

Security architectures that incorporate IRF move from assumption-based trust toward evidence-based assurance. Rather than asking only whether an action was permitted or executed, they also ask whether it actually occurred.

9.3 Safety Engineering

In safety-critical systems, the consequences of assuming realization can be severe. Safety mechanisms often depend on the belief that commands, interlocks, or emergency actions have taken effect once they are acknowledged or logged.

IRF challenges this assumption by making explicit the need to verify that safety-related intents are realized at the point where they matter most. Acknowledgment without realization provides a false sense of security, particularly in systems where delayed or partial realization can have catastrophic consequences.

By treating realization as something that must be verified rather than inferred, IRF complements existing safety practices and supports more robust assurance of safety functions.

9.4 Assurance and Verification Culture

Perhaps the most significant implication of IRF is cultural. Many assurance practices implicitly equate internal correctness with external truth. Logs, metrics, and status indicators become substitutes for verification, reinforcing a culture of trust in system self-reporting.

IRF challenges this culture by insisting on evidence grounded in reality. It encourages practitioners to ask not only "Did the system say it succeeded?" but "What evidence exists that the intended effect actually occurred?"

Adopting IRF as a design and assurance principle promotes skepticism of unverified success and fosters a mindset in which trust is earned through observation rather than assumed through reporting.

9.5 Key Takeaway

At its core, Intent Realization Fidelity exposes a simple but often overlooked distinction: execution is not outcome, and acknowledgment is not proof.

Systems that cannot demonstrate Intent Realization Fidelity are operating on trust, not evidence.

Recognizing this distinction is the first step toward building systems that can meaningfully verify what they do, rather than merely reporting what they intended to do.

10 Conclusion

Modern systems are built on an implicit trust assumption: that declared intent followed by successful execution implies realized outcome. As this paper has shown, that assumption does not reliably hold. Across physical, cyber-physical, and purely digital systems, internal execution and self-reported success frequently diverge from what actually occurs in the world.

This gap between intent and realization is not an implementation defect or an edge case; it is a structural blind spot. Existing approaches emphasize execution integrity, behavioral monitoring, and fault detection, yet leave realization largely inferred rather than verified. As a result, systems often operate with confidence that is unsupported by independent evidence.

To address this gap, Intent Realization Fidelity (IRF) was introduced as a first-class system property. IRF characterized the degree to which declared intent is faithfully realized at the lowest level where its effects materially manifest, based on independent evidence rather than self-reporting. By explicitly separating intent, execution, and realization, IRF provides a framework for reasoning about what systems actually do, not merely what they claim to have done.

IRF is not a replacement for existing assurance mechanisms, nor does it prescribe specific implementations. Instead, it complements current practices by exposing a missing dimension of verification. Treating IRF as a design consideration encourages systems to surface meaningful points of evidence, reduce reliance on assumption-based trust, and acknowledge the limits of abstraction when verifying outcomes.

As systems grow more complex, automated, and interconnected, the cost of unverified realization continues to rise. Designing systems that can demonstrate Intent Realization Fidelity is therefore not an academic exercise, but a practical necessity. Future systems that aspire to be trustworthy must be able to answer a simple question with evidence grounded in reality:

Did the system actually do what it intended to do?

Acknowledgments

The author thanks Joe Weiss for his work in raising awareness of the “Level 0 Problem” in industrial and cyber-physical systems, which influenced the author’s earlier thinking on field-level verification and informed subsequent work.

References

1. Jerome H. Saltzer, David P. Reed, and David D. Clark. *End-to-End Arguments in System Design*. ACM Transactions on Computer Systems, vol. 2, no. 4, pp. 277–288, 1984.
2. Butler W. Lampson. *Hints for Computer System Design*. IEEE Software, vol. 1, no. 1, pp. 11–28, 1984.
3. Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. *SWATT: Software-Based Attestation for Embedded Devices*. Proceedings of the IEEE Symposium on Security and Privacy, pp. 272–282, 2004.
4. Jairo Giraldo, Ehsan Sarkar, Alvaro A. Cárdenas, Michail Maniatakis, and Murat Kantarcioglu. *A Survey of Physics-Based Attack Detection in Cyber-Physical Systems*. ACM Computing Surveys, vol. 51, no. 4, pp. 1–36, 2018.
5. Fabio Pasqualetti, Florian Dörfler, and Francesco Bullo. *Attack Detection and Identification in Cyber-Physical Systems*. IEEE Transactions on Automatic Control, vol. 58, no. 11, pp. 2715–2729, 2013.
6. Varun Chandola, Arindam Banerjee, and Vipin Kumar. *Anomaly Detection: A Survey*. ACM Computing Surveys, vol. 41, no. 3, pp. 1–58, 2009.
7. Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. *Detecting Large-Scale System Problems by Mining Console Logs*. Proceedings of the 22nd ACM Symposium on Operating Systems Principles, pp. 117–132, 2009.
8. Robert Bond Randall. *Vibration-Based Condition Monitoring: Industrial, Automotive and Aerospace Applications*. John Wiley & Sons, 2021.
9. International Electrotechnical Commission. *IEC 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*. IEC Standard, Parts 1–7, 2010.
10. International Organization for Standardization. *ISO 26262: Road Vehicles – Functional Safety*. Second edition, ISO Standard, 2018.