Cross-Control Conflict Convergence in Robotic Systems

A Public Conceptual Framework for Preventing Repeated Intervention Loops

Koji Mochizuki

Independent Researcher, Japan

Correspondence: koji.mochizuki@limflex.org

Version 1.0 - Public Conceptual Edition - 26 July 2026

DOI: 10.5281/zenodo.21580992

PUBLIC CONCEPTUAL EDITION - NON-PEER-REVIEWED PREPRINT.

This paper presents a public conceptual framework and does not claim experimental validation, formal proof, or safety certification.

Public disclosure boundary

This Public Conceptual Edition discloses the problem formulation, architecture, evaluation framework, scenarios, and limitations. It intentionally omits implementation-specific thresholds, proprietary compilation logic, internal structural-analogy mechanisms, dynamic boundary-change conditions, confidential control policies, source code, and customer-specific designs.

The related Japanese patent application was filed before this public release:

Japanese Patent Application No. 2026-178149, filed on 26 July 2026.

Abstract

Modern robotic systems combine heterogeneous control elements that operate at different time scales and pursue different local objectives, including hardware protection, servo control, reflex control, safety supervision, motion and task planning, learned policies, remote operation, and human intervention. Each element may be reasonable in isolation, yet the robot may still stop, replan, retry, or alternate authority without reaching an executable action. This paper treats that failure mode as a system-level convergence problem rather than only a compute-capacity or component-quality problem. It introduces a public conceptual framework in which cross-control conflict is represented as a time-ordered sequence: one control element proposes an output, another modifies, limits, rejects, interrupts, or supersedes it, and later elements respond to the resulting state. The framework separates deadline-aware arbitration on the real-time path from higher-computation background analysis. Current conflicts are resolved through bounded outcomes such as application, limited application, substitution, HOLD, additional evaluation, authority change, recovery-condition change, or blocking. The conflict sequence, judgment result, and observed execution result are then associated as lineage-bearing experience and made available for later control adaptation. Background intelligence may perform structural comparison, causal-candidate analysis, simulation, validation, audit, and reverse exploration, while the real-time path consumes only validated and versioned controller-specific artifacts. The objective is not to eliminate necessary safe stops or bypass certified safety functions. It is to provide a basis for reducing unnecessary cross-control stops, repeated intervention loops, and slow recovery caused by unresolved interactions among otherwise legitimate control elements. This is a non-peer-reviewed public conceptual paper; it presents an architecture and evaluation framework rather than experimental validation or implementation-specific logic.

Keywords: robotics; cross-control conflict; control convergence; runtime assurance; safe autonomy; real-time systems; control lineage; physical AI; industrial robotics; humanoid robotics

1. Introduction

Robotic intelligence is no longer concentrated in a single controller. A contemporary robot may include hardware protection, drive and servo loops, collision reflexes, safety-rated monitoring, motion generation, task planning, learned policies, foundation-model reasoning, remote supervision, and direct human intervention. These elements operate at different frequencies, observe different state abstractions, and are accountable for different objectives. A servo loop may preserve trajectory tracking. A reflex may react to unexpected contact. A safety controller may maintain a protective envelope. A planner may search for task completion. An AI component may interpret an ambiguous goal. A human operator may override the autonomous mode. None of these objectives is inherently wrong, yet their interaction can prevent the robot from settling on an action that is both executable and acceptable.

The resulting failure does not always look like a dramatic safety event. It can appear as hesitation: a robot starts and stops, repeats a near-identical plan, alternates between autonomous and supervised modes, or waits for a condition that another controller continuously invalidates. In industrial automation, the consequence may be degraded takt time and unexplained downtime. In mobile robotics, it may be repeated rerouting around the same local conflict. In humanoid manipulation, it may be a grasp plan that is locally valid for the hand but repeatedly invalidated by balance, contact, or whole-body constraints. In teleoperation, an operator command may be accepted, reduced, canceled, restored, and canceled again as communication and safety states change.

A common response is to add another deterministic gate, another exception rule, another retry, or more computing capacity. Each measure can be locally rational. A gate can prevent a hazardous output. An exception can cover a newly observed edge case. A retry can recover from a transient failure. Parallel hardware can reduce the latency of perception or planning. However, these additions do not necessarily resolve semantic or authority conflicts between controllers. They can increase the number of possible interventions, extend the path from proposal to actuation, and create new cycles in which one subsystem repeatedly undoes the work of another.

Robotics has a rich history of layered, hybrid, and behavior-based architectures. Brooks showed how competence could be built from asynchronous layers [1]. Three-layer and hybrid architectures separate deliberation, sequencing, and reactive control [2-4]. Behavior trees provide modular and reactive task

switching [5-7]. Supervisory control gives a formal basis for constraining discrete-event behavior [8], while Simplex and runtime-assurance architectures separate advanced control from verified or trusted fallback behavior [9-12]. Shielding and control-barrier methods constrain unsafe actions [13-15]. These approaches address essential parts of coordination, modularity, and safety. The present proposal does not replace them.

The narrower problem addressed here is what happens after multiple legitimate mechanisms begin to modify one another over time. A safety filter may correct a planner output, but the planner may repeatedly regenerate an equivalent output. A fallback controller may establish safety but fail to create a recoverable path to the task. A human override may change authority, while an autonomous recovery mechanism later restores the previous mode. The system needs not only a rule for the current transition but also a representation of the cross-control sequence, the judgment applied to that sequence, the action that was actually executed, and the observed result. Without that closed loop, the robot can encounter the same interaction as if it were new.

This paper proposes cross-control conflict convergence as a public conceptual framework. Its contributions are fourfold. First, it treats conflict as a sequence across heterogeneous control elements rather than an isolated rejection or fault. Second, it separates deadline-aware current arbitration from slower analysis and adaptation. Third, it associates conflict, judgment, execution, and result as lineage-bearing operational experience. Fourth, it defines an evaluation framework that distinguishes necessary safety stops from unnecessary cross-control stops and repeated intervention loops.

The framework is intentionally implementation-neutral. It does not disclose thresholds, proprietary compilation logic, internal structural-analogy mechanisms, dynamic boundary-change conditions, or customer-specific control policies. It does not claim experimental validation or safety certification. Its purpose is to establish a precise, testable design direction for industrial robots, humanoids, autonomous mobile robots, and other physical AI systems: the complexity of intelligence should not appear as hesitation in robot behavior.

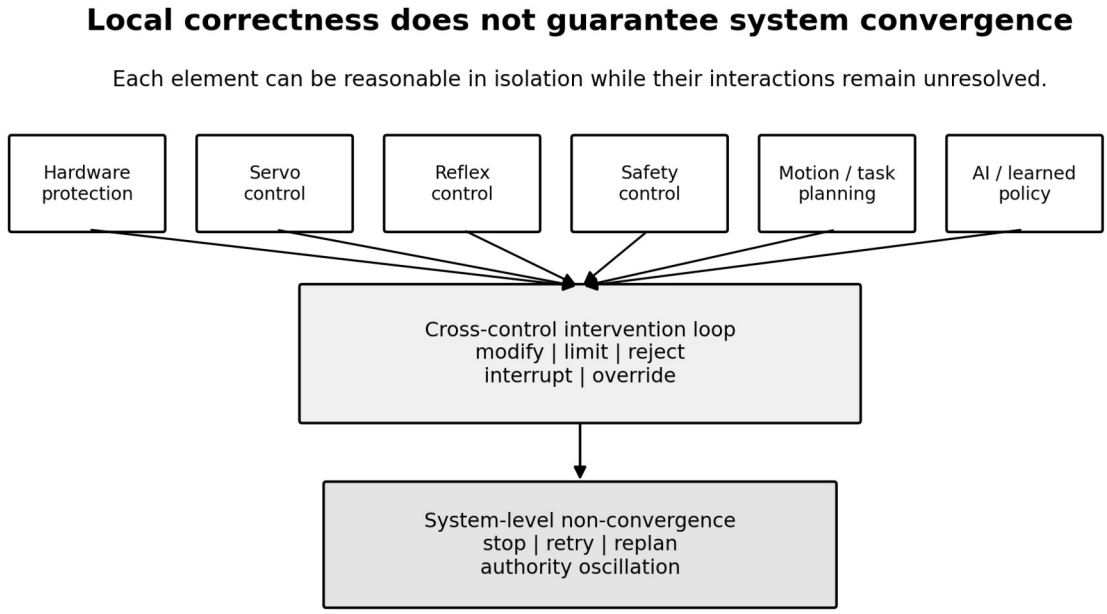

Figure 1. From local control correctness to system-level conflict. The conflict is an interaction among legitimate elements, not necessarily a defect in one component.

Table 1. Examples of heterogeneous control elements.

Control element	Primary objective	Typical cycle	Intervention type	Possible conflict
Hardware protection	Electrical and mechanical limits	microseconds to milliseconds	trip, clamp, inhibit	planner or servo requests beyond device limits
Servo control	tracking and stabilization	sub-millisecond to milliseconds	modify, saturate, track	trajectory demand conflicts with actuator or stability limits

Control element	Primary objective	Typical cycle	Intervention type	Possible conflict
Reflex control	rapid response to contact or disturbance	milliseconds	interrupt, substitute	reflex repeatedly cancels planned motion
Safety control	protective constraints and risk reduction	milliseconds to task cycle	limit, stop, authorize	valid task output repeatedly violates protective envelope
Motion / task planning	task completion and path selection	tens of milliseconds to seconds	propose, replan	replanning regenerates an equivalent rejected output
Learned / AI policy	interpretation, prediction, or candidate generation	variable	propose, rank, recommend	uncertain or stale candidate conflicts with bounded control
Remote operator	external command and exception handling	network-dependent	override, redirect	delayed command conflicts with current autonomous state
Human supervisor	accountability and non-routine judgment	seconds to minutes	approve, reject, change authority	human decision is later undone by automatic recovery

2. Problem Definition

2.1 Heterogeneous control elements

A control element is any software, hardware, human, or remotely operated component capable of proposing, modifying, limiting, rejecting, interrupting, substituting, or authorizing a robot output. Heterogeneity may arise from different control objectives, time scales, authority levels, state representations, implementation technologies, assurance levels, or organizational ownership. The term therefore includes low-level controllers and high-level decision systems, but it is not limited to a strict layer hierarchy.

Table 1 lists representative elements. Hardware protection may act in microseconds or milliseconds and is generally concerned with electrical or mechanical limits. Servo control closes high-frequency feedback around position, velocity, torque, or force. Reflex control reacts to contact or rapidly changing conditions. Safety control evaluates protective constraints and can reduce or block motion. Motion and task planning search over trajectories and action sequences. Learned or AI policies interpret observations, generate candidates, or select behaviors. Remote operators and human supervisors introduce intent, exception handling, and accountability that may not align with the robot's current autonomous state.

2.2 Control events and interventions

A control event is a compact record that a control element proposed, transformed, accepted, deferred, interrupted, or executed an output. Conceptually, an event may include the producing element, task or motion context, proposed output, intervention type, temporal or sequence relation, authority state, and a summary of robot state. This paper does not prescribe an event schema. The essential requirement is that events from different elements can be related sufficiently to reconstruct the interaction that preceded execution or non-execution.

An intervention occurs when one control element changes the practical effect of another element's output. The intervention may be beneficial and required. A safety controller limiting speed is not an error. A reflex interrupting motion after unexpected contact is not an error. Conflict arises when the interactions among such interventions prevent timely settlement on a valid next action, or when the same pattern repeatedly returns without meaningful state improvement.

2.3 Cross-control conflict and conflict sequence

A cross-control conflict occurs when the output of one control element is modified, limited, rejected, interrupted, superseded, or redirected by another control element in a way relevant to task progress or recovery. The conflict is a relationship, not a statement that either element is defective.

A conflict sequence is a time-ordered set of related control events spanning at least two heterogeneous elements. For example:

planner proposes motion -> safety controller limits motion -> planner regenerates an equivalent motion -> reflex controller interrupts -> recovery controller restarts -> planner returns to a similar proposal.

The sequence is more informative than any single rejection. It reveals that locally valid decisions form a cycle. Sequence representation also permits the system to distinguish a novel isolated intervention from a repeated pattern that has already consumed time, compute, or human attention.

2.4 Non-convergence

For this conceptual paper, non-convergence means that the robot fails to reach an executable and acceptable control state within the applicable operational deadline or recovery horizon. Observable indicators may include repeated equivalent interventions, authority oscillation, repeated replanning without state improvement, continued HOLD without a viable release condition, repeated restart, or a growing chain of mutually invalidating outputs. A formal convergence definition would depend on the robot, task, safety model, and timing assumptions and is left for future experimental and theoretical work.

2.5 Current judgment, execution result, and experience

Current judgment is the arbitration outcome applied to an active conflict sequence. Candidate outcomes include application, limited application, substitution, HOLD, additional evaluation, authority change, recovery-condition change, and blocking. Execution result is the subsequently observed physical or operational outcome, including task progress, safe stop, successful fallback, failure, renewed intervention, or a delayed effect.

Cross-control experience is the association of a conflict sequence, judgment result, execution result, and the conditions under which that association is applicable, non-applicable, or invalid. It is not merely a chronological log. Its intended function is to support later arbitration or control-condition changes while preserving the evidence needed to explain and, when necessary, reject the reuse.

3. Why More Gates and More Compute Are Not Enough

A deterministic gate can be indispensable. It may enforce access rules, geometric limits, collision constraints, force thresholds, or certified safety functions. The problem is not the existence of gates; it is the assumption that adding another gate always improves whole-system behavior. Every additional gate introduces another condition under which an output may be transformed, deferred, or blocked. When gates observe different state versions or operate at different rates, their combined behavior can be difficult to predict. The robot may remain safe and still fail to complete useful work.

Fixed priority is similarly valuable but incomplete. A priority order can resolve simultaneous requests: a safety-rated stop overrides a planner; a torque limit overrides a desired acceleration. Yet a fixed order does not necessarily define recovery. The highest-priority element can repeatedly dominate without producing a state from which lower-priority task control can resume. If the planner does not learn why its output was repeatedly modified, it can reproduce the same conflict. If a human override temporarily changes authority but the automatic mode later restores itself using stale assumptions, the conflict can return.

Retries can address transient faults. They become counterproductive when the rejected output is structurally equivalent to its predecessor. A planner that reruns with unchanged constraints may produce a nominally new trajectory that violates the same downstream condition. A task executive may interpret each interruption as a fresh failure and repeat the same recovery branch. Without cross-control sequence identity or similarity, the system cannot tell a productive retry from a loop.

Additional computing capacity reduces the duration of individual analyses but does not automatically resolve differences in meaning, authority, or timing. Two faster planners can disagree faster. A higher-throughput perception pipeline can deliver more observations whose time alignment remains incompatible with the state used by a controller. A large model can generate alternative actions, but the

alternatives may still collide with a fixed safety envelope. Parallelism addresses computational scarcity; it does not by itself create a common arbitration context.

These limitations are consistent with the scope of existing architectures. Layered and behavior-based structures organize control and improve modularity [1-7]. Supervisory control constrains event sequences with respect to formal specifications [8]. Simplex and runtime assurance support safe switching between advanced and trusted controllers [9-12]. Shielding and barrier functions modify actions to satisfy safety conditions [13-15]. Their primary objectives are not necessarily to retain a lineage-bearing record of repeated cross-control interactions and use observed execution results to modify later control conditions across heterogeneous elements.

Table 2 summarizes the distinction. The proposed framework does not argue against gates, priority, fallback, shielding, or replanning. It treats them as possible participants in a larger control interaction. The added question is: after these mechanisms intervene, does the system capture the sequence and use the physical result to reduce the chance of re-entering an equivalent conflict?

The hidden cost of omitting that question appears over long operation. Repeated interpretation consumes planning cycles, creates additional transitions, increases operational variance, and burdens human operators with problems that the robot has already encountered. The relevant optimization target is therefore not only component latency. It is time to an executable action, time to safe and useful recovery, number of repeated interventions, and the ability to explain why the system changed course.

Table 2. Conventional responses and their system-level limitations.

Response	Local benefit	System-level limitation
Additional deterministic gate	blocks a newly identified unsafe or invalid case	adds another intervention and may create cyclic rejection
Fixed priority	resolves simultaneous authority	may preserve safety without defining productive recovery
Retry or restart	recovers from transient faults	repeats structurally equivalent failure when constraints do not change
Planner-only replanning	searches for another candidate	may regenerate outputs that fail the same downstream condition
More parallel compute	reduces component latency	does not resolve semantic, authority, or state-version conflict
Human escalation	handles novel or high-consequence cases	can create workload and authority oscillation without retained experience

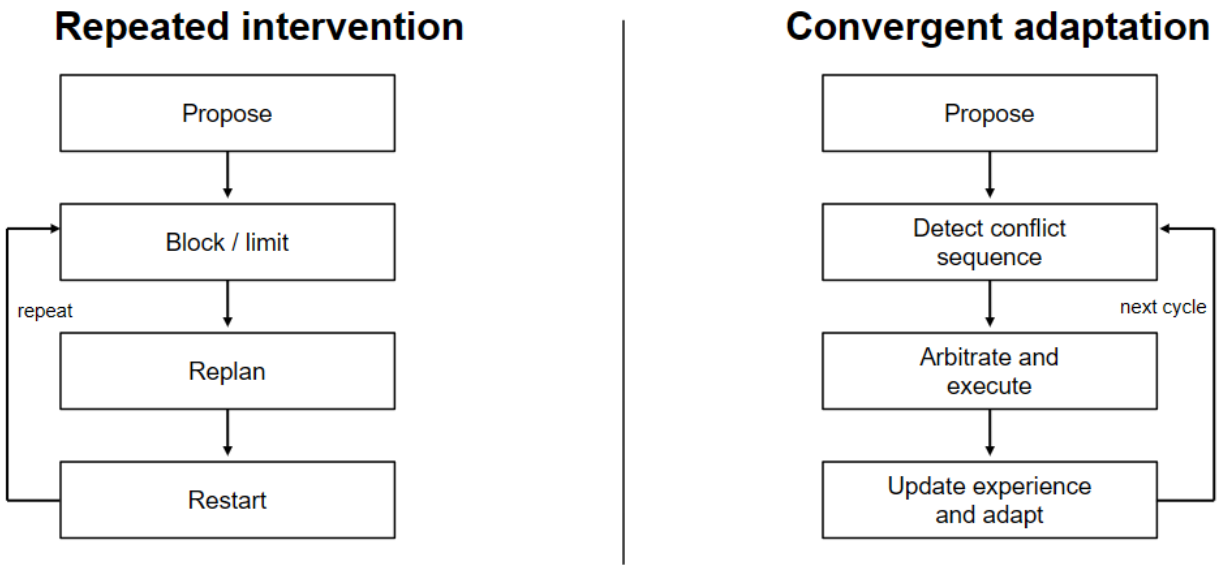

Figure 2. Repeated intervention versus convergent adaptation. Safety actions remain; the change is that their interaction becomes an experience-backed closed loop.

4. Proposed Conceptual Architecture

4.1 Architectural overview

The framework consists of nine conceptual functions: control-event acquisition, conflict-sequence detection, deadline-aware arbitration, a control interface, cross-control experience management, background intelligence, artifact management, audit and lineage, and abnormal-result reverse exploration. These functions may be centralized, distributed, embedded in existing middleware, or implemented through interfaces among current controllers. They are functional roles rather than mandatory software components.

Control-event acquisition observes proposals and interventions from heterogeneous control elements. Conflict-sequence detection relates events across time and control authority. Deadline-aware arbitration generates a bounded outcome for the present situation. The control interface applies the outcome to actuator commands or subsequent control conditions. Experience management associates the sequence, judgment, and result. Background intelligence performs analyses that do not belong on the immediate control deadline. Artifact management validates and versions outputs suitable for real-time consumption. Audit and lineage preserve relationships among inputs, transformations, versions, and results. Reverse exploration begins from an abnormal result and traces backward toward contributing events and assumptions.

4.2 Event acquisition and sequence formation

The framework requires enough observability to distinguish who proposed an output, who intervened, and what followed. An implementation may instrument a message bus, robot middleware, controller APIs, safety events, planner status, operator commands, or hardware fault signals. The public concept does not require full internal access to every controller. A minimal deployment can begin with externally observable transitions: proposal, intervention, execution, result, and recovery.

Sequence formation should preserve ordering and context without forcing every component onto a single clock or data model. Temporal windows, task identifiers, motion identifiers, command lineage, state summaries, or authority transitions may help establish that events belong to the same interaction. Importantly, the sequence can include both explicit rejection and implicit modification. A requested velocity may be clipped; a trajectory may be replaced; a task may remain pending while a safety controller holds motion. These are operationally different but can participate in the same conflict.

4.3 Deadline-aware arbitration

The present conflict must be handled within the robot's operational deadline. The arbitration function therefore uses bounded outcomes rather than open-ended reasoning. A conceptual outcome set includes:

- apply: execute the proposed output as accepted;
- apply with limitation: execute within constrained range, speed, force, authority, or duration;
- substitute: use an alternative output or verified fallback;
- HOLD: preserve the current candidate or state while preventing unsafe or premature execution;
- request additional evaluation: trigger a specified assessment outside or alongside the immediate path;
- change authority: transfer control among autonomous, supervisory, remote, or human elements under defined conditions;
- change recovery conditions: alter the conditions for restart, replanning, or release from HOLD;
- block: prevent the output from reaching the actuator or downstream system.

The framework does not prescribe how these outcomes are selected. Detailed priority calculations, thresholds, and policy compilation remain implementation-specific. Conceptually, the judgment uses the current conflict sequence, current robot state, applicable validated experience, artifact version, and timing constraint. Where no validated experience applies, the system should prefer conservative behavior rather than extrapolating an unverified rule.

4.4 Control interface and execution

The control interface translates the arbitration result into an action that existing controllers can consume. It may limit a command, select a fallback, hold a task, change an authority token, or update a recovery condition. Safety-rated functions remain authoritative within their certified scope. The conceptual layer is not intended to bypass a protective stop or convert an unsafe command into a superficially permitted command.

Execution is followed by observation. The observed result may be immediate, such as successful motion or renewed interruption, or delayed, such as degradation later in the task. The framework therefore allows result records to be updated as evidence becomes available. A judgment that appeared successful at the instant of execution may later be marked non-applicable or invalid for certain conditions.

4.5 Cross-control experience management

The core experience unit associates four kinds of information: the conflict sequence, the judgment result, the execution result, and applicability boundaries. The sequence tells what interacted. The judgment tells what the system decided. The result tells what happened in the physical or operational world. Applicability information prevents indiscriminate reuse.

Experience can support later control in several ways without exposing a general-purpose reasoning process to the deadline. It can change a retry condition, select a known substitute, constrain a planner, adjust a recovery sequence, alter an authority transition, or indicate that a superficially similar experience must not be reused. Experience may also encode that a particular conflict has no safe automated resolution and requires human review.

4.6 Background intelligence and artifact management

Background intelligence receives events, judgments, and results but is not required to complete inside the control cycle. It may compare structure across conflicts, generate causal candidates, search for counterevidence, simulate alternatives, test applicability, and propose controller-specific artifacts. The term intelligence is deliberately broad: the process may use rules, graph analysis, optimization, statistical learning, large models, simulation, human review, or combinations thereof.

An artifact is a bounded, validated, versioned output suitable for later execution or arbitration. Examples include a constrained parameter set, a lookup structure, a recovery transition, a policy fragment, a bounded rule set, a fallback configuration, or a condition for requesting a human gate. The background process does not place an unconstrained analysis directly into the real-time path. Artifact validation, versioning, and rollback are essential because background analysis can be wrong.

4.7 Audit, lineage, and reverse exploration

Lineage links each output to the events and transformations that produced it. Relations may include generated-from, modified-by, judged-by, compiled-from, validated-by, executed-as, resulted-in, and invalidated-by. These concepts are compatible with general provenance models such as W3C PROV-DM [16], but the domain-specific purpose here is operational control: reconstructing why a robot acted, stopped, retried, or changed authority.

Reverse exploration starts from an abnormal or undesirable result and walks backward through the executed output, applied artifact, arbitration judgment, conflict sequence, and originating events. This is not a claim of automatic root-cause proof. It is a way to produce testable causal candidates, locate stale or misapplied experience, and determine whether rollback or additional evidence is required.

4.8 Relationship to existing approaches

The architecture can coexist with behavior trees, supervisors, safety filters, barrier functions, runtime-assurance monitors, and learning controllers. A behavior tree may remain the task executive. A Simplex monitor may retain responsibility for switching to a trusted controller. A shield may continue correcting unsafe learned actions. The proposed layer observes their interactions and provides a closed loop from repeated conflict to later adaptation. Existing work addresses parts of the problem; the

proposed distinction lies in connecting current arbitration, lineage-bearing conflict experience, execution evidence, and subsequent control-condition change.

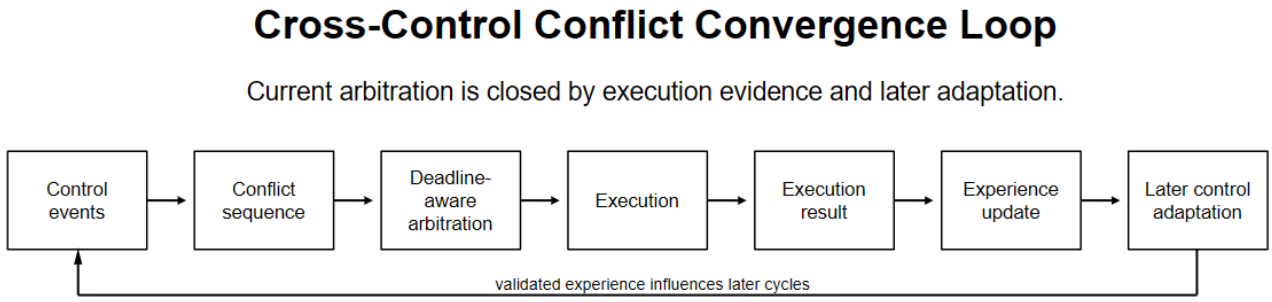

Figure 3. Cross-control conflict convergence loop. Execution evidence closes the loop between current arbitration and later adaptation.

Table 3. Public conceptual architecture components.

Component	Role	Time domain	Output
Control-event acquisition	observes proposals, interventions, authority, and results	real-time / interface	related event records
Conflict-sequence detection	groups cross-control events into an interaction episode	real-time or near-real-time	active sequence and recurrence signal
Deadline-aware arbitration	selects a bounded current outcome	real-time	apply, limit, substitute, HOLD, evaluate, authority change, recovery change, or block
Control interface	applies the arbitration outcome	real-time	actuator command or control-condition change
Experience management	links sequence, judgment, result, and applicability	background with real-time access	validated experience record
Background intelligence	compares structure, tests causes, simulates alternatives	background	candidate explanations and artifacts
Artifact management	validates, versions, publishes, and rolls back artifacts	background / deployment boundary	controller-specific versioned artifact
Audit and lineage	preserves derivation and execution relationships	both	forward and backward trace
Reverse exploration	starts from abnormal result and identifies contributing records	background	testable causal candidates and invalidations

5. Separation of Real-Time and Background Intelligence

Real-time control and high-computation reasoning have different failure tolerances. A real-time path must produce a bounded output before a deadline or execute a defined fallback. Background analysis may consume variable time, examine large histories, invoke simulation, or wait for human review. Combining them synchronously can make the robot's physical response depend on the completion time of an open-ended reasoning process.

The proposed separation is therefore architectural, not merely a deployment optimization. The real-time path acquires current state and events, checks whether a conflict or known pattern is active, verifies that the referenced artifact is valid for the current version and context, performs bounded arbitration, executes or falls back, and records the result. Its accepted computational budget is explicit. When the deadline cannot be met, the system does not wait indefinitely; it moves to a pre-defined safe or HOLD outcome.

The background path operates on a different horizon. It can compare the current sequence with historical sequences, examine whether the apparent similarity is misleading, simulate alternative recovery conditions, evaluate counterevidence, and produce candidate artifacts. It can also incorporate fleet observations, provided that transfer conditions and confidentiality rules are respected. A candidate becomes available to the real-time path only after validation and version assignment.

This separation addresses two risks. The first is deadline exposure: high-computation reasoning should not be synchronously exposed inside the control deadline. The second is semantic exposure: a rich natural-language or model-generated recommendation should not be treated as an actuator-ready command. The output must be reduced to a bounded representation that a specific controller can validate and execute.

The architecture remains compatible with edge, on-premises, and cloud resources. Time-critical functions may run on a real-time CPU, PLC, MCU, DSP, or dedicated controller. Background analysis may run on a GPU, NPU, workstation, or external service. The key requirement is not a specific processor. It is that communication, versioning, and fallback behavior prevent the loss or delay of background services from destabilizing the immediate control path.

A strict separation does not imply no feedback between paths. Execution events and results flow from the real-time path to the background path. Validated artifacts and explicit invalidations flow back. The interface is intentionally narrower than the internal analytical process. This supports auditability and makes it possible to disable, roll back, or replace a background-generated artifact without rewriting the underlying safety controller.

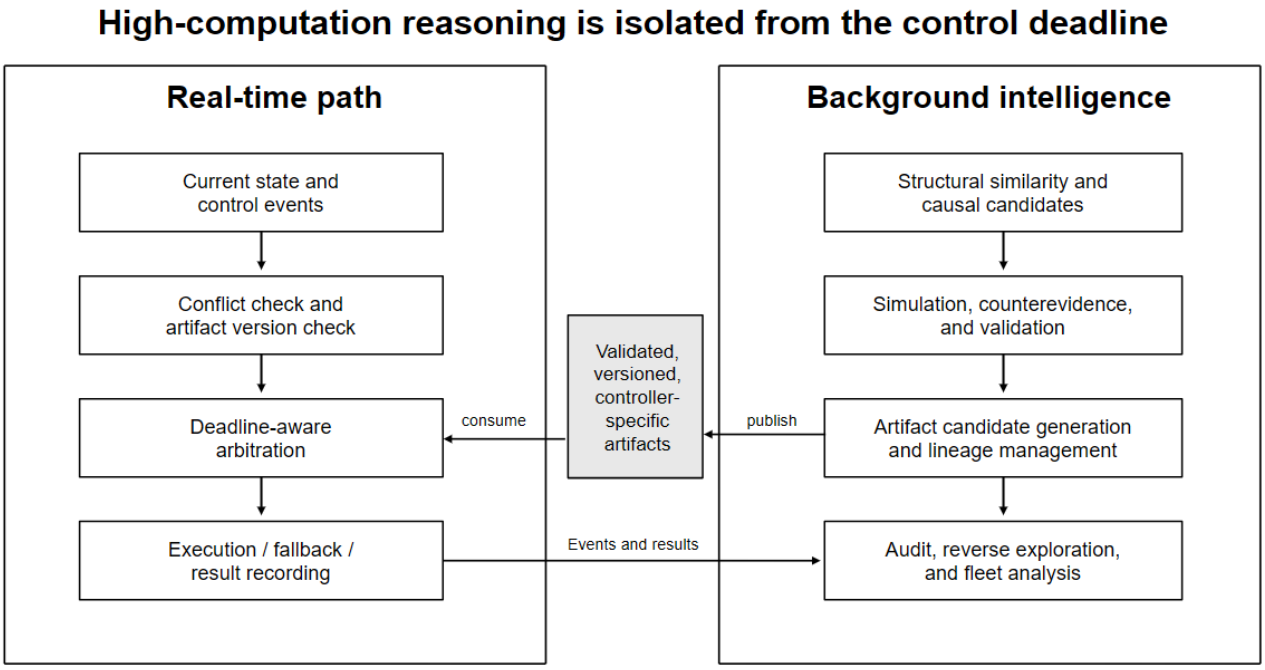

Figure 4. Separation of the real-time path from background intelligence. Only validated, versioned, controller-specific artifacts cross into the deadline path.

6. Lineage, Audit, and Reverse Exploration

Ordinary logs answer when a message was emitted or when a component changed state. They do not necessarily preserve why one control output replaced another, which assumptions were active, which version was applied, or how the physical result changed the later policy. For cross-control convergence, those relationships are part of the operational state.

A lineage-bearing experience graph can represent control events, conflict sequences, judgments, artifacts, executions, results, applicability conditions, and invalidations as related records. The graph need not use a particular database. Its value is the ability to follow both forward and backward relations. Forward tracing explains how a proposal became an executed output. Backward tracing starts from an anomaly and identifies the transformations and assumptions that deserve review.

The distinction from a generic provenance store is the intended control use. Provenance standards provide useful entity, activity, agent, and derivation concepts [16]. The proposed experience adds robotics-specific relationships and makes selected records available to later arbitration. A prior successful substitute may become a candidate for reuse; a result that later proved undesirable may invalidate an artifact; a conflict that required a human decision may set a condition for future escalation.

Auditability also constrains adaptation. A system that cannot identify which artifact changed a control outcome cannot safely learn from that outcome. Versioned lineage supports rollback, comparison, and independent review. It does not guarantee that the causal interpretation is correct, but it makes the interpretation inspectable and falsifiable.

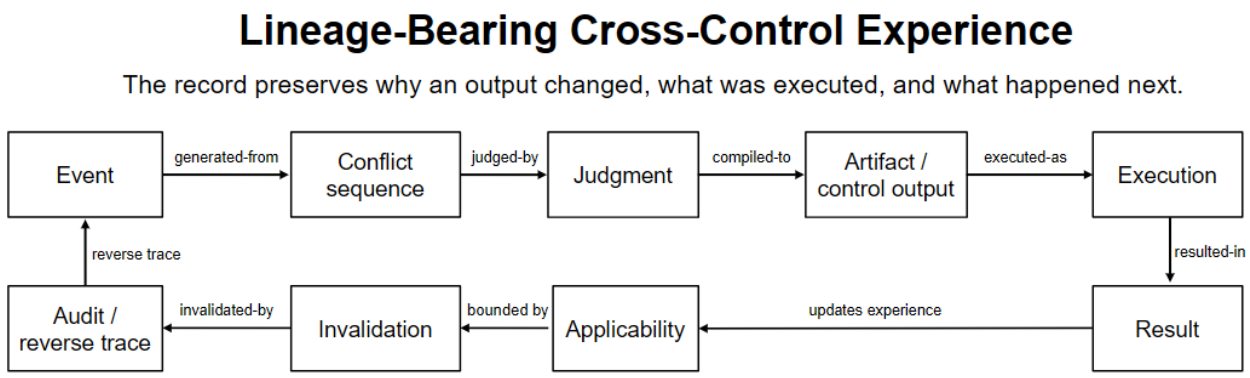

Figure 5. Lineage-bearing cross-control experience. Relationships support forward explanation, reverse exploration, invalidation, and rollback.

7. Illustrative Robotic Scenarios

7.1 Industrial robot: speed request versus protective constraints

Context. A robot cell is asked to increase throughput. The task planner generates a faster trajectory. Servo and drive controllers can track it under nominal conditions, but a safety or collision-monitoring function repeatedly limits speed near a shared workspace boundary.

Conflict sequence. The planner proposes a trajectory; the safety element clips velocity; the task misses its timing target; the planner regenerates a similar trajectory; the same limitation occurs; a recovery branch restarts the task. Every local decision is reasonable, but the cell enters a repeated intervention loop.

Conventional response. Add another timing exception, increase planning frequency, or reduce the global speed limit. These measures may preserve safety but can either leave the loop unresolved or sacrifice performance across all conditions.

Conceptual response. Capture the repeated planner-safety sequence, arbitrate a constrained trajectory or alternative task ordering, execute it, and associate the result with the operating conditions. Later planning can avoid regenerating the same infeasible timing pattern. Candidate metrics are effective takt time, repeated intervention count, unnecessary restart count, and time to a stable executable trajectory.

7.2 Humanoid manipulation: local grasp success versus whole-body feasibility

Context. A humanoid reaches for an object. The hand planner identifies a stable grasp, while whole-body balance, foot contact, joint limits, and collision reflexes constrain the arm and torso.

Conflict sequence. The hand planner selects a grasp pose; whole-body control shifts the torso; balance protection limits the reach; the hand replans locally; contact reflex interrupts; the robot returns to a similar pre-grasp state. The hand can repeatedly solve its local problem without producing a whole-body executable action.

Conventional response. Increase the complexity of the grasp planner or impose a fixed priority in favor of balance. The former may increase latency, and the latter may preserve stability while producing indefinite task failure.

Conceptual response. Treat the hand, arm, balance, contact, and task events as one conflict sequence. Use deadline-aware arbitration to HOLD, substitute a validated reach strategy, or change the recovery condition. Record the whole-body result so that later attempts in structurally similar poses can select a feasible artifact or escalate early. Metrics include time to executable grasp, repeated pre-grasp cycles, number of authority changes, and task completion under stable contact.

7.3 Autonomous mobile robot: local obstacle response versus fleet objectives

Context. An autonomous mobile robot operates in a warehouse with route planning, local obstacle avoidance, fleet priority, and remote supervision.

Conflict sequence. The fleet manager assigns a narrow corridor; local avoidance repeatedly stops for temporary traffic; the route planner selects the same corridor because it remains globally shortest; remote supervision commands continuation; the safety controller blocks motion; authority returns to autonomy and the sequence repeats.

Conventional response. Raise the obstacle timeout, permanently blacklist the corridor, or ask a human to resolve each occurrence. Each response can be appropriate in some contexts but can create new delays or unnecessarily reduce route options.

Conceptual response. Preserve the sequence across fleet, planner, safety, and operator elements. The current arbitration may HOLD, select a verified detour, or retain human authority until a release condition is met. The execution result updates later route and authority conditions. Metrics include recovery time, repeated corridor assignments, remote interventions, and throughput across the fleet.

7.4 Teleoperation and autonomy: delayed command and authority oscillation

Context. A robot is remotely operated in an environment with variable communication delay. Autonomous stabilization and safety functions remain active.

Conflict sequence. The operator sends a motion command based on an earlier view; local autonomy modifies it; the communication state changes; safety interrupts; the operator repeats the command; the autonomous mode resumes and restores a prior objective. Authority moves between participants without a stable recovery state.

Conventional response. Always prioritize the operator, always prioritize autonomy, or stop whenever latency exceeds a threshold. These policies can be necessary in specific systems but may not address repeated transitions around the threshold or stale commands that remain in the queue.

Conceptual response. Include command age, authority state, intervention, and execution result in the conflict sequence. Current arbitration can reject stale commands, preserve a safe HOLD, assign authority for a bounded interval, or require additional confirmation. The outcome is stored with the subsequent physical result. Metrics include authority transitions, stale-command executions prevented, time to safe recovery, and human workload.

Table 4 consolidates these scenarios. They are illustrative hypotheses, not reported experiments. Their purpose is to define testable points at which system-level convergence can be compared with component-centric recovery.

Table 4. Illustrative application scenarios.

Domain	Typical conflict	Conceptual response	Candidate metrics
Industrial robot	planner - safety - restart loop	stable constrained trajectory or task ordering	takt time; intervention and restart count
Humanoid manipulation	hand - whole body - balance - contact loop	whole-body feasible substitute or HOLD	time to executable grasp; repeated pre-grasp cycles
Autonomous mobile robot	fleet - route - avoidance - operator loop	detour, bounded authority, or release condition	recovery time; remote interventions; throughput
Teleoperation	stale command - autonomy - safety - authority loop	reject stale command, HOLD, or bounded authority	authority transitions; stale-command prevention; workload

8. Evaluation Framework

A future evaluation should compare the proposed architecture with common baselines under controlled conflict patterns. Suitable baselines include fixed-priority arbitration, accumulated deterministic gates, retry-based recovery, planner-only replanning, direct human escalation, and a system that records logs but does not reuse cross-control experience.

The experimental unit should be a conflict episode rather than only a completed task. An episode begins when an output is first modified, limited, rejected, interrupted, or superseded by another control element. It ends when the robot reaches an executable stable action, enters a necessary safe stop with a defined resolution path, or exceeds a declared recovery horizon. This makes repeated interventions visible even when the final task eventually succeeds.

Primary metrics should include time to conflict resolution, time to executable action, repeated intervention count, replanning count, unnecessary stop count, recovery time, human intervention count, deadline-miss rate, effective takt time, and task-completion rate. Secondary metrics should include real-time computational load, background-analysis cost, experience-reuse rate, invalid-experience rejection rate, artifact-rollback time, and audit-reconstruction time.

Safety evaluation must distinguish four outcomes: a necessary safe stop, an unnecessary cross-control stop, a repeated intervention loop, and a non-convergent recovery. A reduction in total stops is not sufficient evidence of improvement; it could indicate unsafe suppression of legitimate protection. The architecture should be judged on whether it preserves required safety actions while reducing repeated or avoidable interactions around them.

Experiments should also test negative transfer. A prior experience may look similar but be invalid because the payload, software version, robot morphology, environment, safety configuration, or authority state differs. The evaluation should deliberately introduce such cases and measure whether the system rejects non-applicable experience.

A staged program is practical. Stage 1 can replay logs or simulate controller interactions without commanding hardware. Stage 2 can use a digital twin or hardware-in-the-loop system. Stage 3 can apply bounded artifacts in a non-safety-critical task while existing safety functions remain unchanged. Stage 4 can assess long-duration operation, where the benefit of avoiding repeated interpretation is most likely to appear.

No empirical results are claimed in this paper. The metrics in Table 5 define a falsifiable research and proof-of-concept agenda. A useful outcome may be that the framework helps only certain classes of conflict, or that the instrumentation cost outweighs the benefit in tightly integrated legacy controllers. Such results would refine the applicability boundary rather than invalidate the underlying distinction between local correctness and system convergence.

Table 5. Candidate evaluation metrics.

Metric	Definition	Expected direction
Time to conflict resolution	elapsed time from first intervention to stable current judgment	lower
Time to executable action	elapsed time from conflict start to accepted execution	lower
Repeated intervention count	number of related modifications, rejections, or interruptions	lower
Replanning count	number of planning cycles within the episode	lower
Unnecessary stop count	stops not required by the reference safety condition	lower without reducing necessary stops
Recovery time	time from stop or HOLD to valid operation	lower
Human intervention count	human actions required per conflict episode	lower where automation is appropriate
Deadline-miss rate	fraction of arbitration cycles exceeding the declared deadline	lower
Effective takt time	task-cycle time including conflict and recovery overhead	lower
Experience-reuse rate	fraction of episodes using a validated prior experience	context-dependent, with quality controls
Invalid-experience rejection rate	fraction of deliberately non-applicable experiences correctly rejected	higher
Audit reconstruction time	time to reconstruct proposal, intervention, judgment, execution, and result	lower

9. Expected Benefits

If implemented with adequate observability, validation, and version control, the framework may reduce unnecessary stops, repeated replanning, repeated authority changes, and repeated human escalation. It can support faster recovery by allowing a known applicable response to be selected without rerunning every high-cost analysis. It can also reduce load on the real-time path because validated artifacts, rather than open-ended reasoning, are consumed under the deadline.

Lineage-bearing experience may improve operational explanation. Engineers can examine which control element intervened, which judgment was applied, what was executed, and what happened next. This provides a basis for targeted rollback and for distinguishing a bad artifact from a correct safety intervention responding to an invalid upstream proposal.

At fleet scale, a validated experience may eventually be shared with sufficiently similar robots or tasks, subject to applicability checks, confidentiality, and local validation. That possibility could turn one well-understood failure into an operational improvement across multiple systems. The paper does not define the transfer algorithm, and indiscriminate distribution would be unsafe.

The principal expected benefit is therefore not a promise of maximum speed. It is improved effective performance: less time spent in unresolved control interaction, more predictable recovery, and better use of evidence from prior execution.

10. Limitations

The framework cannot eliminate all conflict. Some tasks are infeasible, some environments remain uncertain, and some safety stops must persist until a human changes the physical situation. Convergence should never be forced by suppressing legitimate protection.

The approach depends on observability. If a proprietary controller exposes neither its intervention nor a meaningful result, the sequence may be incomplete. Instrumentation can also introduce latency, storage cost, synchronization error, and integration risk. A legacy system may require adapters or partial deployment, which can limit the accuracy of sequence reconstruction.

Experience reuse creates a negative-transfer risk. Structural similarity does not prove equivalent dynamics, safety conditions, payload, software version, or authority. Every reusable artifact therefore needs applicability and invalidation conditions, and high-consequence changes may require simulation, formal checking, or human approval. The paper intentionally does not specify an automatic similarity threshold.

Background analysis and artifact validation have computational and organizational cost. Simulation models can be inaccurate. Human review can be slow. Fleet data can be confidential or incompatible. Version control and rollback require disciplined operations. A poor governance process could turn an adaptive layer into another source of conflict.

The proposed architecture does not itself establish functional-safety compliance or replace product-specific hazard analysis. Industrial robot safety remains governed by applicable standards, regulations, risk assessment, and certified functions, including the ISO 10218 series for industrial robots and applications [17,18] and other relevant functional-safety standards [19]. The conceptual layer must be integrated without invalidating those protections.

Finally, this paper offers no experimental result, formal convergence proof, source code, or reference implementation. It describes a public conceptual framework and an evaluation agenda. Claims about performance, reliability, or return on investment require robot-specific testing.

11. Discussion

11.1 Intelligence versus hesitation

As robots gain more perception, planning, learning, and supervisory capability, internal disagreement becomes more likely. The appropriate response is not necessarily to simplify intelligence. Complex

environments can require multiple models and safeguards. The design challenge is to prevent internal complexity from appearing externally as repeated hesitation, unexplained stopping, or unstable authority.

11.2 Component optimization versus system convergence

Engineering organizations often optimize components independently. Perception is measured by accuracy and latency; planning by path quality; servo control by tracking error; safety by coverage and reaction; operator support by usability. These measures remain important, but none directly captures whether interactions among components settle efficiently. Cross-control convergence adds a system metric without denying component metrics.

11.3 Experience as executable operational knowledge

Many systems store logs for diagnosis and datasets for later learning. The proposed experience sits between those categories. It retains enough lineage to explain a judgment and enough operational structure to influence later control. It is therefore closer to bounded executable knowledge than to a raw history. That power also requires stronger validation and invalidation than ordinary logging.

11.4 Relationship to runtime assurance and safety filters

Runtime-assurance architectures distinguish advanced and trusted control and can switch to a safe controller when required [9-12]. Safety shields and barrier functions constrain actions to meet safety conditions [13-15]. Cross-control convergence can observe the repeated interaction among the advanced controller, monitor, fallback, planner, and recovery logic. It does not weaken the safety layer. Its added concern is whether the system learns how to return from repeated intervention to a valid operating state.

11.5 Relationship to behavior architectures

Behavior trees, hybrid architectures, and supervisory control organize execution and switching [1-8]. They can encode retries, fallbacks, and priorities. The proposed framework can be implemented within or around these structures. Its distinction is not a new syntax for task control. It is the explicit lifecycle from heterogeneous intervention sequence to current judgment, execution evidence, lineage-bearing experience, and later condition change.

11.6 Physical AI and governance

Physical AI converts uncertain model outputs into real-world action. This makes timing, authority, version, state, reversibility, and evidence operational rather than administrative concerns. General AI risk-management guidance emphasizes governance and operational risk management across the system lifecycle [20]. The proposed framework contributes a robotics-focused view: governance is not only whether an action is permitted, but whether multiple legitimate controllers can converge on an executable action while preserving an auditable path.

11.7 Fleet implications

A fleet can magnify both value and error. One validated conflict experience could prevent the same repeated loop across many similar robots. One misapplied artifact could also propagate failure. Fleet use therefore requires staged validation, robot and software compatibility checks, local authority, rapid revocation, and preserved lineage. The concept supports such a direction but leaves the transfer mechanism for future work.

12. Future Work

Future work should formalize conflict-sequence identity and convergence under explicit timing and safety assumptions. A simulation benchmark should include repeated planner-safety interaction, authority oscillation, stale commands, failed recovery, and negative-transfer cases. Industrial robot, autonomous mobile robot, and humanoid manipulation studies would test whether the same conceptual structure applies across different control rates and physical risks.

Additional work is required on controller-specific artifact validation, rollback, and formal compatibility with runtime-assurance architectures. Human-robot authority transitions deserve separate study because human intervention can be both a safety mechanism and a source of stale or conflicting intent. Long-duration experiments are important: the value of experience reuse may be small in a single episode but substantial across thousands of cycles.

A later peer-reviewed paper should report reproducible experiments, baseline definitions, statistical analysis, failure cases, and the conditions under which the framework does not improve performance.

13. Conclusion

Robotic systems can fail to progress even when their individual control elements are locally reasonable. The failure may emerge from a sequence in which planners, safety functions, reflexes, low-level controllers, AI components, and humans repeatedly modify one another without reaching a stable executable action. More gates, more retries, and more compute can improve individual functions while leaving this system-level convergence problem unresolved.

This paper proposed a public conceptual framework for cross-control conflict convergence. It treats conflict as a sequence, resolves the current situation through deadline-aware bounded arbitration, observes the execution result, and associates sequence, judgment, and result as reusable experience. High-computation analysis is separated from the real-time path, and only validated, versioned, controller-specific artifacts are returned for later use. Lineage and reverse exploration support audit, invalidation, and rollback.

The framework does not eliminate necessary safe stops, replace certified safety functions, or claim experimental proof. It establishes a testable architecture and evaluation agenda for reducing unnecessary cross-control stops, repeated intervention loops, and slow recovery. The central design goal is not to make intelligence simpler, but to prevent its internal complexity from appearing as hesitation, repeated intervention, or non-convergent behavior in the robot.

Declarations

Funding: No external funding was received for this conceptual paper.

Related patent application: The conceptual architecture described in this paper is related to Japanese Patent Application No. 2026-178149, filed on 26 July 2026. This Public Conceptual Edition does not reproduce the full patent specification and intentionally excludes implementation-specific thresholds, internal logic, and confidential deployment details.

Competing interests: The author is the inventor and applicant of the related patent application and may have a future commercial interest in licensing or adoption of the described architecture. No other competing interests are declared. No patent rights are granted by the copyright licence applied to this manuscript.

Author contributions: Koji Mochizuki conceived the framework, developed the conceptual architecture, and wrote the manuscript.

Data availability: No dataset was generated or analyzed for this conceptual paper.

Code availability: No source code or reference implementation is released with this conceptual paper.

Ethics statement: Not applicable. The paper reports no study involving human participants, animals, or personal data.

Consent to publish: The sole author approves publication of this Public Conceptual Edition.

Acknowledgments: The author acknowledges the robotics, control, runtime-assurance, provenance, and human-robot interaction research communities whose work provides foundations for this conceptual synthesis. OpenAI ChatGPT was used for language drafting, structural editing, and document preparation. The author reviewed and revised the full manuscript and accepts responsibility for its accuracy, originality, and integrity.

References

[1] R. A. Brooks, "A Robust Layered Control System for a Mobile Robot," IEEE Journal on Robotics and Automation, vol. 2, no. 1, pp. 14-23, 1986. doi: 10.1109/JRA.1986.1087032.

[2] E. Gat, "On Three-Layer Architectures," in Artificial Intelligence and Mobile Robots, D. Kortenkamp, R. P. Bonasso, and R. Murphy, Eds. Cambridge, MA: AAAI Press/MIT Press, 1998.

[3] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, "An Architecture for Autonomy," The International Journal of Robotics Research, vol. 17, no. 4, pp. 315-337, 1998. doi: 10.1177/027836499801700402.

[4] F. Ingrand and M. Ghallab, "Deliberation for Autonomous Robots: A Survey," Artificial Intelligence, vol. 247, pp. 10-44, 2017. doi: 10.1016/j.artint.2014.11.003.

[5] M. Colledanchise and P. Ogren, Behavior Trees in Robotics and AI: An Introduction. Boca Raton, FL: CRC Press, 2018. doi: 10.1201/9780429489105.

[6] M. Iovino, E. Scukins, J. Styrud, P. Ogren, and C. Smith, "A Survey of Behavior Trees in Robotics and AI," arXiv:2005.05842, 2020. doi: 10.48550/arXiv.2005.05842.

[7] M. Colledanchise and L. Natale, "On the Implementation of Behavior Trees in Robotics," IEEE Robotics and Automation Letters, vol. 6, no. 3, pp. 5929-5936, 2021. doi: 10.1109/LRA.2021.3087442.

[8] P. J. Ramadge and W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes," SIAM Journal on Control and Optimization, vol. 25, no. 1, pp. 206-230, 1987. doi: 10.1137/0325013.

[9] D. Seto, B. H. Krogh, L. Sha, and A. Chutinan, "The Simplex Architecture for Safe Online Control System Upgrades," in Proceedings of the American Control Conference, 1998, pp. 3504-3508. doi: 10.1109/ACC.1998.703255.

[10] L. Sha, R. Rajkumar, and M. Gagliardi, "Evolving Dependable Real-Time Systems," in Proceedings of the IEEE Aerospace Applications Conference, 1996, pp. 335-346. doi: 10.1109/AERO.1996.495894.

[11] A. Desai, S. Ghosh, S. A. Seshia, N. Shankar, and A. Tiwari, "SOTER: A Runtime Assurance Framework for Programming Safe Robotics Systems," in 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2019, pp. 138-150. doi: 10.1109/DSN.2019.00027.

[12] S. Shivakumar, H. Torfah, A. Desai, and S. A. Seshia, "SOTER on ROS: A Run-Time Assurance Framework on the Robot Operating System," in Runtime Verification, LNCS 12399, 2020, pp. 184-194. doi: 10.1007/978-3-030-60508-7_10.

[13] M. Alshiekh, R. Bloem, R. Ehlers, B. Konighofer, S. Niekum, and U. Topcu, "Safe Reinforcement Learning via Shielding," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1, 2018. doi: 10.1609/aaai.v32i1.11797.

[14] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control Barrier Function Based Quadratic Programs for Safety Critical Systems," IEEE Transactions on Automatic Control, vol. 62, no. 8, pp. 3861-3876, 2017. doi: 10.1109/TAC.2016.2638961.

[15] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin, "FaSTrack: A Modular Framework for Fast and Guaranteed Safe Motion Planning," in IEEE Conference on Decision and Control, 2017, pp. 1517-1522. doi: 10.1109/CDC.2017.8263867.

[16] L. Moreau and P. Missier, Eds., "PROV-DM: The PROV Data Model," W3C Recommendation, 30 April 2013. https://www.w3.org/TR/prov-dm/.

[17] ISO 10218-1:2025, Robotics - Safety Requirements - Part 1: Industrial Robots. International Organization for Standardization, 2025.

[18] ISO 10218-2:2025, Robotics - Safety Requirements - Part 2: Industrial Robot Applications and Robot Cells. International Organization for Standardization, 2025.

[19] IEC 61508-1:2010, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems - Part 1: General Requirements. International Electrotechnical Commission, 2010.

[20] E. Tabassi, Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1. Gaithersburg, MD: National Institute of Standards and Technology, 2023. doi: 10.6028/NIST.AI.100-1.

[21] P. F. Orzechowski, C. Burger, and M. Lauer, "Decision-Making for Automated Vehicles Using a Hierarchical Behavior-Based Arbitration Scheme," in IEEE Intelligent Vehicles Symposium, 2020, pp. 767-774. doi: 10.1109/IV47402.2020.9304723.

[22] A. Bensalem, L. de Silva, F. Ingrand, and R. Yan, "A Verifiable and Correct-by-Construction Controller for Robot Functional Levels," arXiv:1309.0442, 2013. doi: 10.48550/arXiv.1309.0442.