IJC International Journal of Computer
Peer-Reviewed International Journal
ISSN: 2307-4523
https://ijcjournal.org

Design of the Action Connector Layer in Enterprise Agentic Workflow Automation Platforms

Nitin Garg[a,*]
a: Senior Product Manager - Technical, Seattle, Washington, USA
* *Corresponding author: garg.ng@gmail.com*
Accepted: 28 September 2026 | Published: 24 October 2026

Abstract

An enterprise agent may produce a valid tool call while the connected system rejects the request, accepts it for later processing, or changes a record without returning confirmation. The action connector layer must preserve these distinctions. This article develops a design model for that layer by analyzing recent research on Application Programming Interface (API) use, stateful tool execution, and agent security. It specifies an action contract with typed inputs, preconditions, an authority scope, an expected effect, and defined outcomes. The model classifies integrations by credential ownership and the extent of control over the service interface. It then relates validation errors, authorization failures, provider rejections, and uncertain execution states to permissible agent responses. A versioned contract provides multiple agents with a common definition of an action, while the connector enforces permissions and interprets the external response. The proposed specifications support connector review, provider changes, and the diagnosis of failures across enterprise workflows.

Keywords: Action connectors; agentic workflow automation; enterprise integration; tool execution; authorization; action contracts; service failures

1. Introduction

An agent can prepare an update to a business record and submit a correctly formed tool call. The connected system still has to recognize the target record, accept the acting identity, permit the requested fields, and report what happened. If the connection closes after dispatch, the agent may receive an error even though the service completed the update. The component that handles this boundary determines whether the agent can make a defensible claim about the result.

An action connector links an agent-accessible operation to an external service operation. It checks the request before dispatch, applies the connection's authority, translates the request into the provider's format, and interprets the response. These responsibilities recur when several agents use the same action. They also persist when an external provider changes its interface.

This article defines the action connector layer as a distinct object of enterprise platform design. The analysis pursues three objectives: to specify a reusable contract for an agent-accessible action, to classify integration patterns according to their authority and interface requirements, and to relate external service failures to the responses an agent may provide or the actions it may take next. The working hypothesis is that a shared connector layer can preserve the meaning of an action across workflows when the platform defines its contract, permission check, and outcomes at the service boundary.

Basu et al. [1] examine Application Programming Interface (API) task detection and argument completion through API-BLEND, providing a basis for studying how a requested operation becomes an executable call. Patil et al. [2] develop the Berkeley Function Calling Leaderboard (BFCL) to evaluate function selection and argument validity across diverse calling settings. For connector design, these studies inform the definition of callable operations, typed parameters, required inputs. Establishing the resulting business effect requires additional evidence from the connected service.

Lu et al. [3] incorporate stateful execution and dependencies between tools into ToolSandbox, making the state preceding an action part of its evaluation. Yao et al. [4] evaluate task completion in τ-bench by comparing the final database state with an annotated goal state. Together, these approaches support an interpretation of action correctness that includes preconditions and observable effects. A connector applying this interpretation needs an explicit rule for deciding whether a service response establishes completion, acceptance for later processing, or an unresolved outcome.

Guo et al. [5] address the stability of tool evaluation under changing API conditions. This concern also affects connector maintenance, where revisions to provider fields or response behavior can alter the interpretation of a registered action. The present study translates these evaluation concerns into proposed requirements for contract versioning, provider mappings, outcome classification.

Earlier work on multi-agent coordination [6] and human participation [7] provides the workflow context for these requirements. The present study examines the execution boundary shared by such workflows. Its contribution is a conceptual design that connects action contracts, integration responsibilities, evidence-based outcome reporting. The analysis asks how these elements can be specified together so that callers receive a consistent interpretation of an external action.

2. Materials and Methods

2.1. Materials

The source corpus consists of recent studies of API tasks, tool-call evaluation, stateful execution, and agent security. Research on API task detection and argument completion informs the input contract [1]. Earlier publications on coordination and human participation define adjacent subjects [6], [7]. Studies of unstable APIs and task alignment inform the treatment of provider behavior and unauthorized calls [5], [8]. Stateful benchmarks examine dependencies between actions, function-call validity, final system state, and the security of tool access [2]–[4], [9], [10]. Sources were entered into the corpus when their methods or findings addressed a decision connector designers face. The analysis uses published research and a proposed enterprise action model; it contains no organization-specific implementation records or performance claims.

2.2. Methods

The study uses a qualitative conceptual design based on a purposively selected literature corpus. Each source is examined for its relevance to input specification, execution preconditions, authority enforcement, provider behavior, or evidence of task completion. These categories provide a common basis for comparing studies with different evaluation objectives.

The synthesis proceeds from the evidence available at successive execution stages. Input validation establishes whether a request satisfies the registered contract. Authorization determines whether the configured identity may perform the requested operation on the target resource. Provider responses and subsequent state checks supply evidence about acceptance or completion. Integration patterns are classified by credential ownership and responsibility for maintaining the service interface.

The resulting contract fields and outcome classes are proposed design requirements derived from this comparison. Their assessment in the present study concerns conceptual consistency and traceability to the problems addressed in the source corpus. Operational effectiveness remains a subject for implementation-based evaluation.

3. Results

A reusable action needs a definition that survives a change of calling agent. Its contract should identify the operation, input types, required fields, preconditions, target resource, authority scope, expected effect, and possible outcomes. The contract also needs to identify whether the action reads

information or changes the external state. Research on API task detection and slot filling addresses call formation [1], while function-call benchmarks assess tool selection and argument validity [2]. Those evaluations support input design, but an enterprise connector must also establish what occurred after dispatch.

Preconditions give the connector a place to express requirements beyond data type. An update request may contain a valid record identifier format while referring to a record that the acting identity cannot access. Another request may ask for a state transition that the provider disallows for the record's current state. The connector can reject the first issue it can establish before dispatch and return a specific validation result. The agent then has information for revising the request without interpreting that result as a provider outage.

A versioned contract fixes the meaning of an action for its callers. Connector engineers may change an endpoint mapping after a provider revises its API. If the action keeps its inputs and promised effect, existing callers can retain their contract version. A change in required inputs, authority, or the meaning of completion requires a version review. Research on tool evaluation has documented instability in live API conditions [5], which gives provider changes a direct place in the connector's maintenance model.

The integration pattern determines how the connector acquires authority. A platform-managed service connection may use an administrator-configured service identity. A third-party application may use credentials delegated by a user or a service account configured for a workflow. A custom interface may require its operator to specify the endpoint policy and define the meaning of each response. These patterns differ in who grants access, how engineers constrain its scope, and which party maintains the mapping between the registered action and the provider operation.

A workflow instruction cannot enlarge the permission attached to a connection. Research on indirect prompt injection examines how external content can steer a tool-using agent toward an attacker's requested action [9]. Work on task alignment asks whether a proposed call serves the authorized task [8], and broader security evaluation identifies tool use as a point where attackers can exploit an agent [10]. For a connector, the design consequence is a permission check against the acting identity, requested operation, and target resource immediately before dispatch. The check uses the configured authority, irrespective of the text that led the agent to propose the call.

Figure 1 maps validation, authority, dispatch, and outcome interpretation to a single proposed execution path. Its separate outcome branches draw on research that distinguishes call validity, state changes, and tool-access risks [1]–[5], [8]–[10].

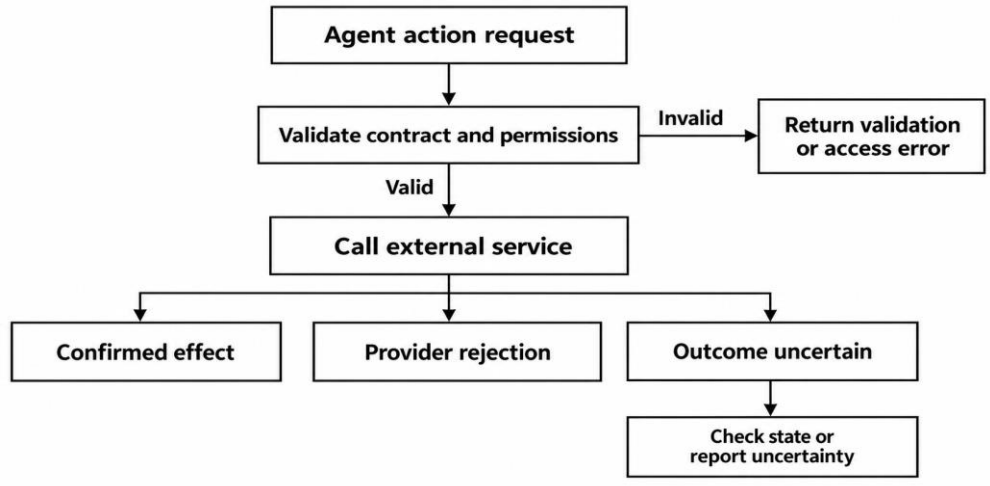


Figure 1. Proposed action connector execution path, synthesized from [1]–[5], [8]–[10]

Call formation, service acknowledgment, and external effect require separate interpretations. Function-call benchmarks assess whether a model selects a suitable tool and supplies valid arguments [2]. ToolSandbox adds state dependencies and checks intermediate outcomes [3]. Research on tool-agent interaction compares the final database state with the requested goal state [4]. A valid call can therefore precede a provider rejection, while an acknowledgment can precede completion. The connector's outcome vocabulary should retain all three stages.

Failure classes follow the evidence available at the boundary. A validation error means the connector found an invalid input or unmet precondition before dispatch. An authorization error indicates that the configured identity lacks permission to perform the operation or access the resource. A provider rejection means the service received the request and declined it. The agent can respond to these classes by correcting the request, seeking the required access through the established process, or addressing the provider's stated objection.

A transport failure needs a more exact description. A failure before dispatch gives the connector a basis to report that it sent no request. A lost response after dispatch leaves the external effect uncertain. Repeating a state-changing request can create another record or apply an update twice. The connector may retry under a defined rule when the provider supports request deduplication or when the connector has established that the first attempt had no effect. Otherwise, it should check the target state through an available read operation and return an unknown execution state if the check yields no conclusive answer.

Some services acknowledge receipt and finish the operation later. The connector should report acceptance as pending until a provider response or a state check establishes completion. This distinction matters for any caller that might otherwise tell a user that a record has changed. The proposed contract, integration classification, and failure policy meet at this point. Each action defines what completion means, the provider mapping identifies the available evidence, and the connector returns only the outcome that evidence supports.

4. Discussion

The proposed model assigns a specific evidential requirement to each reported outcome. Input validity establishes that a request can proceed to execution checks. Service acceptance establishes that the provider has received or admitted the operation. Completion requires evidence matching the effect defined in the action contract. This interpretation makes the completion criterion part of the connector specification and constrains the claims a calling agent can make about an external change.

The relationship between the contract and external state develops the evaluation concerns identified in ToolSandbox [3] and τ-bench [4]. State dependencies inform the specification of preconditions, while goal-state evaluation informs the definition of an expected effect. Their use in the proposed architecture is a design inference. A benchmark can inspect a controlled environment against a known target state. An enterprise connector depends on the response fields, status endpoints, resource visibility available from its provider. The contract therefore needs to identify the evidence through which the expected effect can be established.

An uncertain response exposes the practical consequence of this requirement. Consider a state-changing request whose response is lost after dispatch. A timeout establishes a communication failure, leaving the external effect unresolved. The proposed policy links any subsequent retry to documented deduplication support or evidence that the initial attempt produced no effect. A status lookup can resolve the outcome when the provider exposes a sufficiently informative execution record. Otherwise, the unknown state remains an explicit result requiring reconciliation. This example illustrates the proposed decision rule and its dependence on provider capabilities.

Authority enforcement addresses a separate requirement within the same execution path. A permission check establishes whether the acting identity has access to the operation and resource. Task-alignment controls examine whether the proposed action serves the authorized user request [8]. These checks address different properties of a call, and their joint application requires a surrounding authorization and task-governance process. The connector model specifies the enforcement point for configured permissions, with the wider platform responsible for determining the legitimate task context.

Connector owners can publish an action only after specifying its inputs, preconditions, authority requirements, expected effect, and outcome types. Engineers then map that contract to a provider operation. The review should include examples of an accepted request, a request the connector rejects before dispatch, and a provider response the connector classifies. These examples make the registered action usable by multiple agents under a single definition.

The integration patterns in Table 1 assign authority and interface maintenance to different parties. A team selecting a pattern needs to know who controls the credentials and who can confirm the external effect before exposing the action to callers.

Table 1. Integration patterns and connector responsibilities

Integration pattern	Authority configuration	Interface maintenance	Evidence of effect
Platform-managed service	The administrator configures a service identity and resource permissions	Connector team maps registered actions to supported operations	Service response or available resource-state check
Third-party application	User delegates access, or the administrator configures a service identity	Connector team tracks provider fields, scopes, and API changes	Provider response, returned record identifier, or status lookup
Custom service interface	The operator defines the identity and endpoint access policy	The operator and connector team define the request and response mapping	Documented acknowledgment rule or target-state check

The same action catalog can serve all three patterns, although each places a different maintenance obligation on connector owners. For a third-party connection, an expired credential may prevent dispatch. For a custom interface, the principal difficulty may be establishing whether an acknowledgment confirms a completed change. The contract should state the outcome the connector can return under either condition.

Failure behavior belongs in the action specification. Owners should identify whether repeated calls can duplicate an effect, whether the provider accepts a deduplication key, and whether a later read can establish the result. Table 2 connects those properties to the information an agent receives.

Table 2. External-service outcomes and agent responses

Outcome class	Connector's established information	Agent response
Invalid request	Input or precondition fails before dispatch	Revise the identified input or condition
Unauthorized request	Acting identity lacks the required access	Stop the call and report the access requirement
Provider rejection	Provider receives and declines the request	Report the stated reason
Safe transient failure	No effect occurred, or provider-supported deduplication protects a repeat	Retry under the action's defined rule
Unknown execution state	The provider may have received the request; the effect remains uncertain	Check the status if available; report uncertainty
Accepted, pending effect	Provider acknowledges receipt without confirming completion	Retain pending status and check for completion
Confirmed completion	Response or state check establishes the specified effect	Report completion and the relevant record identifier

An unknown execution state requires a durable request record. The connector owner should retain the action version, target, acting identity reference, dispatch time, and provider request identifier

where available. Those fields support a later status check. They also prevent a generic timeout from automatically triggering a repeat of a state-changing call.

Connector monitoring can group records by validation error, access denial, provider rejection, pending effect, and unknown outcome. A rise in validation errors for one contract version directs engineers to its input definition or its callers—repeated unknown outcomes for one provider call for examination of its response and status-check facilities. The categories locate the work required to repair a connection without implying an unmeasured improvement in platform performance.

The study develops a conceptual architecture from published research and a proposed action model. Its findings establish design requirements and relationships between execution stages. Claims concerning reduced failure rates, improved task completion, lower latency, or stronger security require empirical evaluation using an implemented connector layer.

The literature corpus was selected purposively for its relevance to connector design. The selection procedure supports a focused synthesis, with coverage bounded by the included publications. The resulting categories remain open to revision through a broader review of enterprise integration research and implementation evidence.

Applicability depends on the provider's execution semantics and observability. Deduplication requires documented provider support or an equivalent implementation guarantee. Completion checks require an accessible status mechanism or a resource-state observation that can be linked to the submitted action. Concurrent updates and delayed state visibility can weaken that link, requiring action-specific reconciliation rules. Providers with limited execution evidence may leave a larger proportion of requests in an unknown state.

The model addresses individual action execution at the connector boundary. Cross-service transaction coordination, compensation across workflow steps, tenant isolation, credential lifecycle management require additional architectural specifications. The permission check also depends on the correctness of the configured identity and access policy. Its protective effect against maliciously induced actions requires evaluation alongside task-alignment controls and the wider platform security design.

Empirical validation should examine actions with synchronous completion, deferred completion, lost responses after dispatch. Relevant measures include outcome-classification accuracy, duplicate state changes following retries, time to reconcile unknown outcomes, authorization enforcement under adversarial inputs. Comparisons across providers would help establish which requirements transfer between integration environments and which require provider-specific adaptation.

5. Conclusion

The proposed action connector model combines a versioned action contract, an allocation of integration responsibilities, an execution-outcome policy. The contract defines the requested operation and its completion criterion. The integration pattern assigns responsibility for credentials and provider mappings. Outcome classification links the agent's response to the evidence available before dispatch, after acknowledgment, during subsequent verification.

The architectural contribution consists of specifying these decisions at a shared execution boundary. Application requires documented provider semantics, enforceable permissions, action-specific rules for uncertain outcomes. The study provides a conceptual basis for implementation and evaluation, with operational benefits remaining subject to empirical validation.

Acknowledgements

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

References

[1] K. Basu, I. Abdelaziz, S. Chaudhury, S. Dan, M. Crouse, A. Munawar, V. Austel, S. Kumaravel, V. Muthusamy, P. Kapanipathi, and L. Lastras, “API-BLEND: A comprehensive corpora for training and benchmarking API LLMs,” in Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, vol. 1, 2024, pp. 12859–12870, doi: 10.18653/v1/2024.acl-long.694.

[2] S. G. Patil, H. Mao, F. Yan, C. C.-J. Ji, V. Suresh, I. Stoica, and J. E. Gonzalez, “The Berkeley Function Calling Leaderboard (BFCL): From tool use to agentic evaluation of large language models,” in Proceedings of the 42nd International Conference on Machine Learning, vol. 267, 2025, pp. 48371–48392. [Online]. Available: https://proceedings.mlr.press/v267/patil25a.html

[3] J. Lu, T. Holleis, Y. Zhang, B. Aumayer, F. Nan, F. Bai, S. Ma, S. Ma, M. Li, G. Yin, Z. Wang, and R. Pang, “ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities,” arXiv, 2024, doi: 10.48550/arXiv.2408.04682.

[4] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan, “τ-bench: A benchmark for tool-agent-user interaction in real-world domains,” arXiv, 2024, doi: 10.48550/arXiv.2406.12045.

[5] Z. Guo, S. Cheng, H. Wang, S. Liang, Y. Qin, P. Li, Z. Liu, M. Sun, and Y. Liu, “StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models,” in Findings of the Association for Computational Linguistics: ACL 2024, 2024, doi: 10.18653/v1/2024.findings-acl.664.

[6] N. Garg, “Agentic AI for enterprise workflow automation: Impact and architectural principles for multi-agent orchestration,” International Journal of Research in Commerce and Management Studies, vol. 8, no. 3, pp. 122–136, 2026, doi: 10.38193/IJRCMS.2026.8309.

[7] N. Garg, “The role of humans-in-the-loop in hybrid automation systems based on generative artificial intelligence,” Australian Journal of Science and Technology, vol. 10, no. 2, 2026.

[8] F. Jia, T. Wu, X. Qin, and A. Squicciarini, “The task shield: Enforcing task alignment to defend against indirect prompt injection in LLM agents,” in Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics, vol. 1, 2025, pp. 29680–29697, doi: 10.18653/v1/2025.acl-long.1435.

[9] Q. Zhan, Z. Liang, Z. Ying, and D. Kang, “InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents,” in Findings of the Association for Computational Linguistics: ACL 2024, 2024, pp. 10471–10506, doi: 10.18653/v1/2024.findings-acl.624.

[10] H. Zhang, J. Huang, K. Mei, Y. Yao, Z. Wang, C. Zhan, H. Wang, and Y. Zhang, “Agent Security Bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents,” in Proceedings of the International Conference on Learning Representations, 2025. [Online]. Available: https://proceedings.iclr.cc/paper_files/paper/2025/hash/5750f91d8fb9d5c02bd8ad2c3b44456b-Abstract-Conference.html