apilabs.ai: An AI-Native Executable Contract System for Stateful API Testing and Workflow Synthesis

apilabs.ai Research

https://apilabs.ai — June 2026

Abstract

Modern REST APIs govern complex, distributed systems and are inherently stateful. However, the industry-standard contracts describing them—such as the OpenAPI Specification [1]—remain largely static, structural, and endpoint-centric. This structural bias forces engineering teams to bridge the gap using fragmented tools, manual scripting, and disjointed Continuous Integration (CI) pipelines to validate multi-step workflows. This paper introduces an executable behavioral contract system driven by a specialized Domain Specific Language (DSL), engineered for apilabs.ai — the AI-Native platform for APIs and MCPs. By unifying schema definitions, state transitions, authentication dependencies, and business invariants into a single executable artifact, we establish a comprehensive API System Model. Coupled with a persistent Execution Context, this architecture allows AI agents via Model Context Protocol (MCP) [2] orchestration to autonomously synthesize stateful tests, dynamically navigate workflows, and detect semantic drift. This effectively transforms API contracts from passive documentation into an active operating system for AI-driven development.

1. Introduction

The proliferation of API-first architectures has made interface contracts the backbone of modern software. Tools like OpenAPI successfully standardize the description of syntax: available endpoints, HTTP methods, and payload schemas. However, they fail to capture *behavior*. A typical REST implementation involves interconnected resources where the validity of one action (e.g., `refund_order`) is strictly dependent on the prior successful execution of another (e.g., `pay_order`).

Because static schemas lack semantic awareness of workflows and state transitions, validation efforts are relegated to imperative test scripts (e.g., Postman collections) or custom CI automation. We argue that APIs should no longer be modeled merely as disjointed endpoints. Instead, they must be represented as *executable behavioral systems*. This paper presents an AI-native approach that shifts the paradigm from structural validation to stateful workflow synthesis, utilizing a unified DSL to drive autonomous testing agents.

2. The API System Model and Execution Context

To enable agentic reasoning over APIs, we decouple the theoretical structure of the API from its runtime memory. This requires a bipartite architecture hosted within the apilabs.ai platform [3]:

2.1. The API System Model

The System Model serves as the durable source of truth. It extends beyond endpoint documentation to capture the full semantic topology of the service. Formally, we define the API System Model as a tuple **M = < R, E, W, S, I >**:

- **R**: The set of interconnected business resources (e.g., Users, Invoices).
- **E**: The environment and authentication constraints.

- **W**: The set of valid operational workflows connecting endpoints.
- **S**: The state space of the system over time.
- **I**: The set of business invariants that must hold true across any transition.

2.2. The API Execution Context

While the System Model defines what is *possible*, the Execution Context provides the stateful runtime memory required by AI agents. This context tracks active authentication bindings, recent execution traces, temporary tokens, and historical contract drift, allowing agents to replay failures and synthesize contextual tests dynamically within developer environments.

3. The Executable Behavioral DSL

At the core of this system is a specialized YAML-based DSL designed to capture execution logic rather than just interface shape. Unlike an OpenAPI specification that answers *"What does the payload look like?"*, the apilabs.ai DSL answers *"What sequences are valid?"*

```
workflow:
  onboarding_sequence:
    - action: create_account
      endpoint: POST /users
      extract:
        user_id: response.body.id
    - action: assign_role
      endpoint: PATCH /users/{user_id}/role
      payload: { "role": "admin" }
    - action: validate_invariant
      assert:
        user.status == "active"
        user.role == "admin"
```

By encoding the extraction of variables (`user_id`) and explicitly defining assertions within the workflow graph, the DSL acts simultaneously as documentation, an execution graph, and a validation engine.

4. Agentic Operations, MCPs, and System Implementation

With a machine-readable, stateful contract in place, testing shifts from a human-authored imperative task to an AI-synthesized autonomous operation [4]. Utilizing the Model Context Protocol (MCP) [2], AI agents can interface directly with the Execution Context to act as continuous API operators.

To validate this thesis, operational prototypes of this architecture are built and executed entirely within the apilabs.ai API platform. Through this platform, agents perform the following:

Stateful Test Synthesis: Rather than generating isolated endpoint fuzzing, the AI agent explores the state graph, inferring required authentication dependencies and generating valid, multi-step workflows.

Semantic Drift Detection: By continuously monitoring live traffic or CI execution traces against the apilabs.ai DSL, the system identifies undocumented state transitions, broken invariants, or authentication regressions—flagging discrepancies where the implementation drifts from the behavioral contract.

5. Related Work

The limitations of stateless API testing are well documented. Schemathesis [5] has successfully popularized property-based testing and schema-aware fuzzing, generating edge-case inputs directly from OpenAPI specs. However, its stateful capabilities remain constrained by the semantic limits of the underlying OpenAPI links. Microsoft's RESTler [6] advanced the field by introducing stateful fuzzing, analyzing producer-consumer dependencies to infer valid request sequences. Similarly, Pact [7] champions consumer-driven contracts to ensure integration compatibility. Our approach builds upon these foundations but diverges by introducing a first-class behavioral DSL, elevating workflows to the same architectural tier as endpoints and optimizing specifically for AI-agent orchestration via MCPs.

6. Conclusion

The next evolution of API engineering requires moving beyond static interface descriptions. By introducing an executable contract DSL that natively understands state, dependencies, and workflows, we provide the necessary foundation for autonomous agents to reason about, test, and operate APIs. Prototyped within apilabs.ai, this architecture not only bridges the gap between documentation and execution but paves the way for self-healing, agent-operated API ecosystems.

References

[1] OpenAPI Initiative. "OpenAPI Specification." https://swagger.io/specification/ (accessed Jun. 2026).

[2] Anthropic. "Model Context Protocol (MCP) Specification." https://modelcontextprotocol.io/ (accessed Jun. 2026).

[3] apilabs.ai. "apilabs.ai — The AI-Native platform for APIs and MCPs." https://apilabs.ai (accessed Jun. 2026).

[4] S. G. Patil et al., "Gorilla: Large Language Model Connected with Massive APIs," arXiv preprint arXiv:2305.15334, 2023.

[5] Z. Hatfield-Dodds et al., "Schemathesis: Property-based testing for API schemas." https://github.com/schemathesis/schemathesis (accessed Jun. 2026).

[6] V. Atlidakis, P. Godefroid, and M. Marina, "RESTler: Stateful REST API Fuzzing," in Proc. 41st Int. Conf. on Software Engineering (ICSE), 2019, pp. 748–758.

[7] Pact Foundation. "Pact: Consumer-Driven Contract Testing." https://pact.io/ (accessed Jun. 2026).