ENGINEERING STANDARDS LIBRARY / SOFTWARE ENGINEERING

Polyglot Boundaries, Typed Contracts, and IPC Standard

Explicit language boundaries, schema-governed messages, compatibility policy, local IPC, remote RPC, and failure containment.

PERMANENT DOCUMENT ID

MYTHOS-SWE-0005

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Polyglot Boundaries, Typed Contracts, and IPC Standard

DOCUMENT ID
MYTHOS-SWE-0005

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

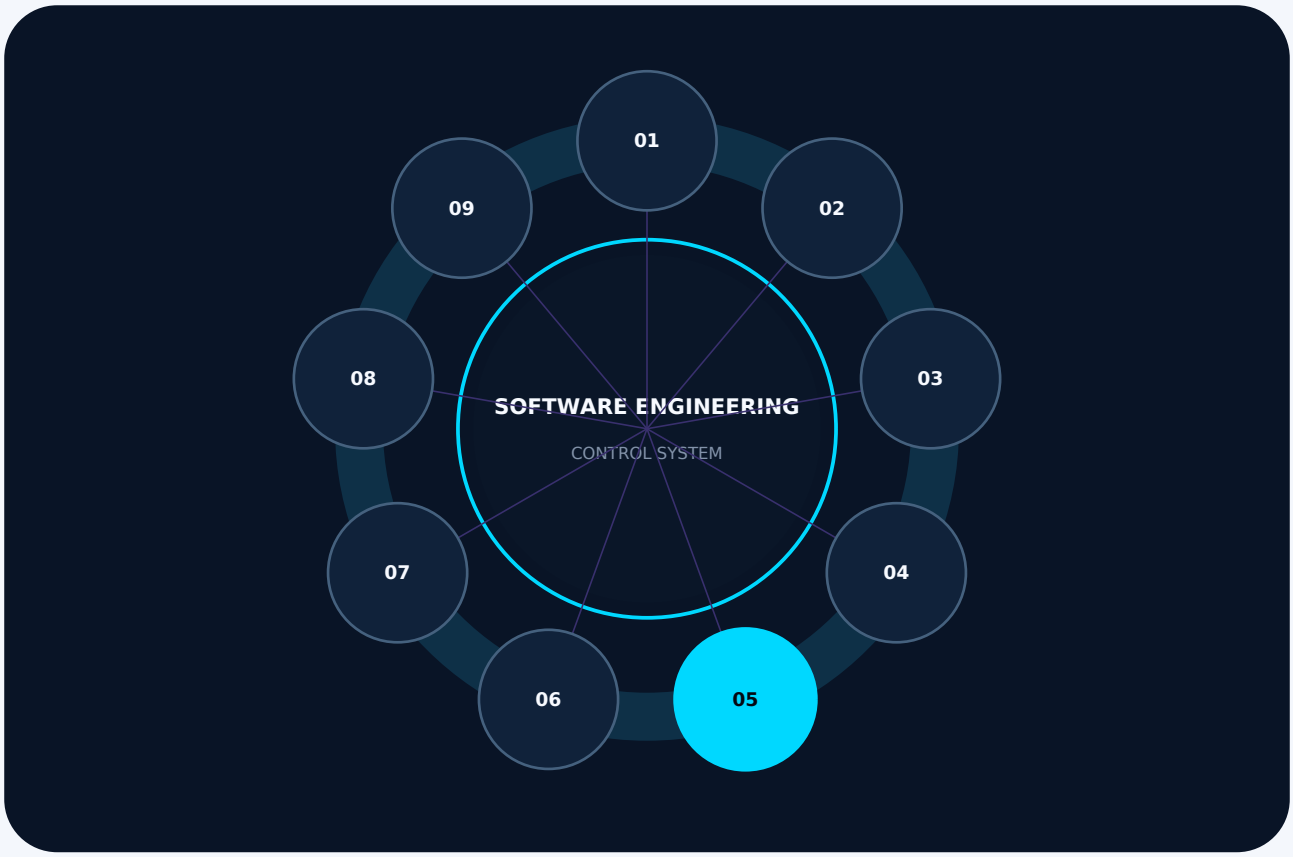
SCHEDULED REVIEW
2027-01-24

16	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable guidance or requirements for its defined scope. Interpretation must preserve the permanent document identity, recorded evidence, controlled exceptions, and formal revision history.

DOMAIN CONTROL SYSTEM

MYTHOS-SWE-0005 inside the software engineering standard constellation



Connected assurance	Architecture, implementation, operations, evidence, and lifecycle are evaluated as one controlled system.
Specialization rule	Companion standards may add precision, but they cannot silently weaken this publication's requirements.

EXECUTIVE POSITION

Polyglot Boundaries, Typed Contracts, and IPC Standard

The integration of polyglot systems has failed.

Evidence-bearing delivery path

Assurance maturity spectrum

CONTENTS

Publication navigation

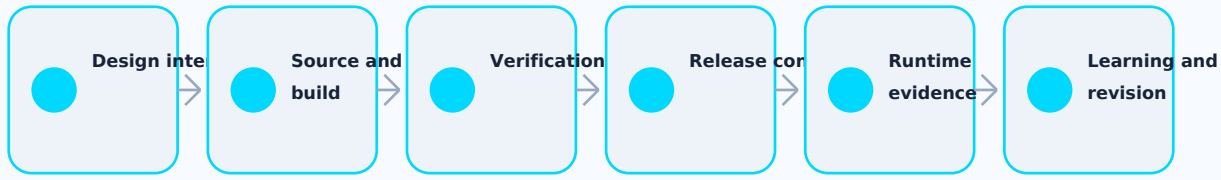
CONTROL TOPOLOGY

Assurance model and evaluation surface

The standard is assessed as an evidence-bearing system. Capability scores do not replace mandatory gates, implementation proof, or operational validation.

Capability family	Evaluation nodes
Typed Contracts and Schema-First Design (Weight: 25%)	2
Schema Evolution and Compatibility (Weight: 20%)	2
IPC Mechanisms and Performance (Weight: 20%)	2
Boundary Isolation and Shared DB Prohibition (Weight: 15%)	2
Contract Testing and Governance (Weight: 10%)	1
Polyglot Code Generation (Weight: 10%)	1
Suite Composition	2
Additional Evaluation Dimension	4

Lifecycle control loop



EVALUATION INDEX

Criteria and evidence map

#	Capability family	Criterion
1	Typed Contracts and Schema-First Design (Weight: 25%)	Schema Authority
2	Typed Contracts and Schema-First Design (Weight: 25%)	Type Strictness
3	Schema Evolution and Compatibility (Weight: 20%)	Breaking Change Prevention
4	Schema Evolution and Compatibility (Weight: 20%)	Versioning Strategy
5	IPC Mechanisms and Performance (Weight: 20%)	Internal Communication Protocol
6	IPC Mechanisms and Performance (Weight: 20%)	Latency and Overhead
7	Boundary Isolation and Shared DB Prohibition (Weight: 15%)	Shared Database Prohibition
8	Boundary Isolation and Shared DB Prohibition (Weight: 15%)	Anti-Corruption at Boundary
9	Contract Testing and Governance (Weight: 10%)	Consumer-Driven Contract Testing
10	Polyglot Code Generation (Weight: 10%)	Code Generation
11	Suite Composition	M-POLY-S-Contracts
12	Suite Composition	M-POLY-S-Isolation
13	Additional Evaluation Dimension	Boundary Dimensions
14	Additional Evaluation Dimension	The Boundary Doctrine
15	Additional Evaluation Dimension	Scoring Model
16	Additional Evaluation Dimension	Suite Composition

NORMATIVE PUBLICATION

Requirements, evaluation methodology, and implementation guidance

The following body preserves the substantive source standard while normalizing publication status, identity, navigation, and visual presentation.

Executive Mandate

The integration of polyglot systems has failed.

Organizations adopt multiple languages (Rust for performance, Go for control planes, Python for AI) to leverage their strengths, only to integrate them via the lowest common denominator: untyped JSON over HTTP REST. This results in runtime `KeyError` exceptions, implicit coupling via shared database tables, and versioning nightmares where adding a field to one service crashes three others.

This standard exists because:

1. Untyped JSON is a Liability: Passing `Dict[str, Any]` or `map[string]interface{}` across a network boundary is an architectural failure. If the contract is not machine-readable and strictly enforced, the integration will break at runtime.
2. A Shared Database is Not an API: Integrating services by reading another service's database tables couples the structural internals of both systems, making independent deployment impossible. Integration MUST occur via well-defined APIs, asynchronous events, or Anti-Corruption Layers.
3. REST is for Humans, gRPC is for Machines: RESTful JSON APIs are optimized for browser clients and human debugging. Internal service-to-service communication MUST use binary, strongly-typed, schema-first protocols (gRPC, Protobuf, Cap'n Proto) to ensure performance, determinism, and strict contracts.
4. Code-First is Anti-Contract: Generating an API specification from code annotations (e.g., Swagger/OpenAPI generated from Spring controllers) allows implementation details to leak into the contract. The Schema MUST be designed first; code is a derived artifact.

This document establishes the Mythos Polyglot Boundary Standard (MPBS), a comprehensive, evidence-based methodology for evaluating typed contracts, IPC mechanisms, and schema evolution in polyglot environments.

Scope and Applicability

In Scope

This standard covers the evaluation of:

1. Schema-First contract design (Protobuf, Smithy, OpenAPI)
1. Inter-Process Communication (IPC) protocols (gRPC, Cap'n Proto, Unix Domain Sockets)
1. Schema evolution and backward/forward compatibility

1. Polyglot code generation and type mapping
1. Shared database prohibition and API-only integration
1. Contract testing and compatibility verification

Out of Scope

1. General domain-driven design (covered in MYTHOS-SWE-0001)
1. Event-driven architecture and messaging (covered in MYTHOS-DAT-0004)
1. General network transport (covered in MYTHOS-NET-0006)

Audience

1. Principal engineers and software architects designing polyglot systems
1. Platform engineers building service meshes and API gateways
1. Security teams evaluating IPC attack surfaces
1. AI governance, risk, and compliance teams
1. Standards bodies seeking a reference polyglot methodology

Definitions and Taxonomy

Boundary Dimensions

Dimension	Definition
Schema-First	A design paradigm where the API contract (e.g., `.proto` file, `.smithy` model) is defined and reviewed before any implementation code is written.
Backward Compatibility	The ability of a service to process requests from older clients without breaking.
Forward Compatibility	The ability of an older client to process responses from a newer service without breaking (usually via unknown field ignorance).
Shared Database Anti-Pattern	The practice of integrating two services by allowing one to read/write directly to the other's database, bypassing the domain logic.
Contract Testing	Testing that verifies the contract (schema and behavior) between a consumer and a provider, ensuring no breaking changes are introduced.

The Boundary Doctrine

> Untyped JSON is a liability. A shared database is not an API. REST is for humans; gRPC is for machines. Code-first is anti-contract. If the schema is not the source of truth, the integration is a lie.

Evaluation Methodology

Scoring Model

Each capability is scored from 0 to 5.

Score	Meaning
0	Fails basic task; untyped JSON, shared DBs, no versioning
1	Basic OpenAPI but not enforced; REST only
2	Functional but lacks schema-first design or compatibility checks
3	Solid production performance; schema-first, gRPC, compatibility enforced
4	Strong performance; automated contract testing, zero shared DBs
5	Best-in-class; sets the standard for deterministic, typed polyglot systems

Weighting

Category	Weight	Rationale
Typed Contracts & Schema-First Design	25%	Untyped boundaries cause runtime crashes
Schema Evolution & Compatibility	20%	Breaking changes cause cascading outages
IPC Mechanisms & Performance	20%	JSON/REST is too slow and loose for internal comms
Boundary Isolation & Shared DB Prohibition	15%	Shared DBs destroy independent deployability
Contract Testing & Governance	10%	Unverified contracts drift
Polyglot Code Generation	10%	Manual serialization introduces bugs

Total: 100%

A system MUST score at least 3.0 in Typed Contracts and Boundary Isolation to be considered for production use.

Evaluation Categories

Typed Contracts and Schema-First Design (Weight: 25%)

Schema Authority

Is the API contract (schema) the source of truth, or is it derived from code?

Score	Criteria
0	No API specification; implementation is the contract
1	OpenAPI generated from code annotations (Code-First)
2	OpenAPI written manually but not enforced at runtime
3	Schema-First design (Protobuf, Smithy); code generated from schema
4	Schema-First + runtime enforcement (e.g., gRPC schema validation)
5	Perfect authority: schema is the immutable contract, formally verified, cryptographically signed, zero code-first generation

Type Strictness

Are all fields strictly typed (no `Any`, no `object`, no untyped JSON)?

Score	Criteria
0	`map` or `Dict[str, Any]` in contracts
1	Basic types but frequent use of `oneof` with untyped fallback
2	All fields typed; enums used for fixed values
3	Custom scalar types (e.g., `UUID`, `IPv4Address`, `Timestamp`)
4	Strict type mapping across all polyglot targets (Rust, Go, Python)
5	Perfect strictness: zero untyped fields, type-state modeling in schemas, formally verified type compatibility

Schema Evolution and Compatibility (Weight: 20%)

Breaking Change Prevention

Does the system prevent breaking changes from reaching production?

Score	Criteria
0	No compatibility checking; breaking changes deployed manually
1	Manual review of schema diffs
2	Automated diff tool (e.g., `buf breaking`) run locally
3	Automated compatibility check in CI; breaking changes block merge
4	Backward and forward compatibility guaranteed; unknown fields preserved
5	Perfect prevention: formally verified compatibility, automated migration tools, zero breaking changes possible

Versioning Strategy

How are API versions managed?

Score	Criteria
0	No versioning; URL path versioning (e.g., `/v1/`, `/v2/`)
1	Header-based versioning
2	Semantic versioning of the schema package
3	Continuous evolution without version bumps (Protobuf/Smithy style: add fields, never remove)
4	Schema registry with lifecycle management (deprecated -> sunset -> removed)
5	Perfect versioning: continuous evolution, formally verified sunset periods, automated client migration tracking

IPC Mechanisms and Performance (Weight: 20%)

Internal Communication Protocol

What protocol is used for service-to-service communication?

Score	Criteria
0	JSON over HTTP REST (internal)
1	JSON over HTTP with HTTP/2

Score	Criteria
2	gRPC (Protobuf over HTTP/2) for some services
3	gRPC mandatory for all internal service-to-service communication
4	gRPC + Unix Domain Sockets for sidecar/co-located communication
5	Perfect protocol: gRPC for distributed, Cap'n Proto/uDS for co-located, formally verified payload sizes, zero internal REST

Latency and Overhead

Is the IPC mechanism optimized for low latency and minimal serialization overhead?

Score	Criteria
0	High latency (JSON serialization/deserialization dominates)
1	gRPC with standard Protobuf serialization
2	Protobuf with zero-copy parsing (e.g., `bytes` fields)
3	gRPC with HTTP/2 multiplexing and connection pooling
4	Cap'n Proto or FlatBuffers for zero-copy inter-process communication
5	Perfect performance: zero-copy IPC, shared memory for co-located services, formally verified latency bounds

Boundary Isolation and Shared DB Prohibition (Weight: 15%)

Shared Database Prohibition

Are services prevented from accessing another service's database?

Score	Criteria
0	Services share databases freely
1	Read-only access to other services' databases
2	Logical separation but same physical database instance
3	Separate database instances; integration via API only
4	Network-level segregation; database credentials scoped to single service
5	Perfect isolation: cryptographic boundary enforcement, zero cross-service database access, formally verified data ownership

Anti-Corruption at Boundary

Are external models prevented from leaking into the domain?

Score	Criteria
0	External contract types (e.g., Protobuf messages) used in domain logic
1	Manual mapping between external and domain types
2	Automated mapping but in domain layer
3	Anti-Corruption Layer (ACL) at the boundary; domain uses its own types
4	ACL with generated mappers and contract tests

Score	Criteria
5	Perfect isolation: WASM-sandboxed ACL, formally verified translation, zero external schema leakage

Contract Testing and Governance (Weight: 10%)

Consumer-Driven Contract Testing

Are consumer expectations verified against the provider?

Score	Criteria
0	No contract testing; integration tested in staging only
1	Provider tests its own API
2	Consumer-driven contract tests (Pact) written but not in CI
3	Contract tests in CI for every PR
4	Contract tests + schema compatibility checks + backward compatibility verification
5	Perfect governance: formally verified consumer contracts, automated provider verification, zero untested integrations

Polyglot Code Generation (Weight: 10%)

Code Generation

Is client/server code generated from the schema, or written manually?

Score	Criteria
0	Manual HTTP clients and JSON parsing
1	Generated clients but manually maintained
2	Generated clients from schema (e.g., `buf generate`, `protoc`)
3	Generated clients in CI; schema update triggers client regeneration
4	Generated clients with strict type mapping and custom scalar support
5	Perfect generation: deterministic, reproducible, formally verified type mapping across Rust/Go/Python/TS, zero manual serialization

The Mythos Polyglot Suite (M-POLY-S)

Traditional benchmarks measure endpoint latency. The M-POLY-S evaluates contract strictness, compatibility, and boundary isolation.

Suite Composition

M-POLY-S-Contracts

1. Breaking Change: 100+ tests applying known breaking schema changes, verifying CI blocks them.
1. Compatibility: 50+ tests verifying old clients can communicate with new servers (forward compatibility).

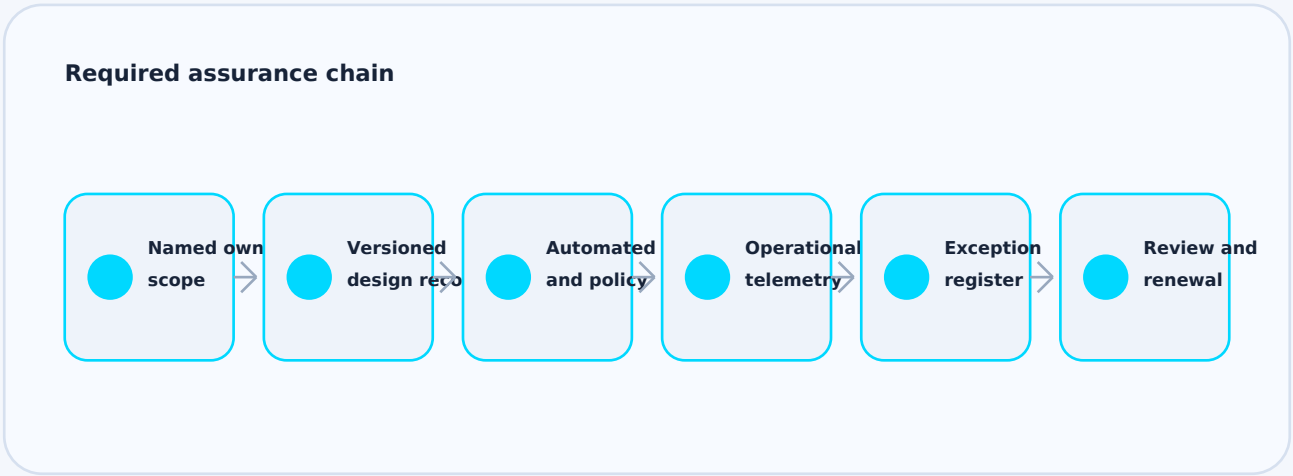
M-POLY-S-Isolation

1. Shared DB Scan: 50+ tests scanning for database credentials used by multiple services.
1. Untyped Field Scan: 100+ tests scanning for `Any`, `object`, or `map` in schemas.

Execution Protocol

ASSURANCE AND ADOPTION

Implementation evidence and review gates



A passing score is not sufficient without current evidence. High-assurance adoption requires traceable design decisions, reproducible builds, independently reviewable tests, controlled secrets, runtime observability, failure exercises, and a dated exception register.

Current authority register

Authority	Version / snapshot	Applicability	Review by
Rust Project: The Rust Programming Language and Rust Reference	Rolling official documentation; book assumes Rust 1.90.0 or later and Rust 2024 Edition	High	2026-10-22
Go Project: The Go Programming Language Specification and Effective Go	Rolling official documentation snapshot	High	2026-10-22
Python Software Foundation: Python Language Reference, Packaging, Typing, and PEP Index	Rolling official documentation snapshot	High	2026-10-22
Tauri Programme: Tauri 2 Documentation	Tauri 2 rolling documentation snapshot	High	2026-10-22
Svelte Project: Svelte Documentation	Svelte 5 rolling documentation snapshot	High	2026-10-22
TLA+ Foundation: TLA+ Language and Tools Documentation	Rolling documentation snapshot	High	2027-01-24
OpenAPI Initiative: OpenAPI Specification	3.2.0	High	2027-01-24
OpenTelemetry Project: OpenTelemetry Documentation and Specification	Rolling specification snapshot; profiles public alpha and declarative configuration stable in 2026	High	2026-10-22

Research limitation. Official documentation establishes published interfaces and declared behavior. It does not prove the performance, security, interoperability, cost, or operational suitability of a specific implementation. Those claims require controlled testing and retained evidence.