AI-Native APIs

Designing Platforms for Agentic Autonomy

Mike Jackson
Co-Founder & CTO
Varindor Oy

https://www.varindor.com

AI-native APIs: Semantic Middleware, not REST

Executable operating surfaces that agents can discover, use safely, and recover from

Agents need concrete callable actions, not implied workflows

Human API contract

Docs + examples + institutional context + judgment

→

Agent Ontology contract

Runtime-discoverable actions, constraints, states, and recovery paths

FAILURE MODE

Agentic autonomy breaks when the API depends on human interpretation

Every implicit API assumption becomes a place where an agent guesses

Humans resolve these questions from external context, agents need answers as data

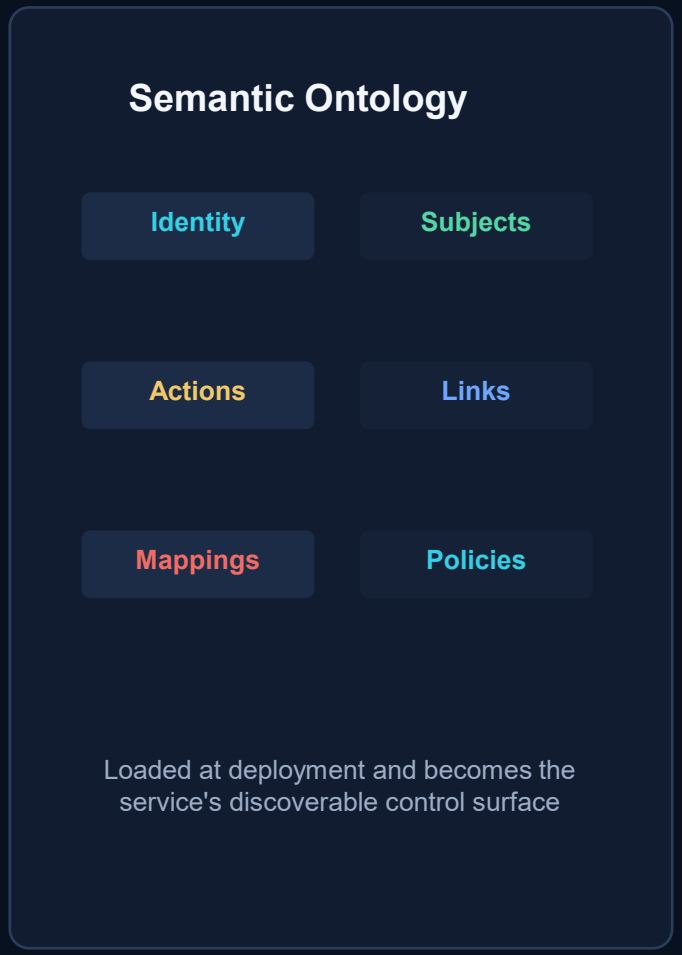
Move from REST endpoint catalog to Semantic Ontology

The ontology should expose the world the agent acts in, not just methods it can call

988	40	54	17	563
registry rows	subject types	actions	link types	properties

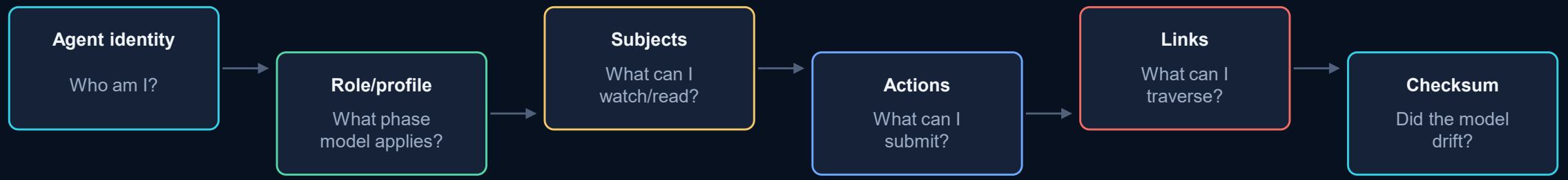
Not documentation. Not SDK folklore. Runtime data.

Agent code consumes definitions, not assumptions



Bootstrap the Agents: Runtime Behaviour as Data

An agent should discover identity, role, readable subjects, submittable actions, links, and version drift before it acts



Same binary

Different identity + role produces a different operating surface

No hardcoded domain

The role data names the watched subjects and submittable actions

Hot model refresh

Checksum drift triggers re-bootstrap and atomic model swap

OODA+ as an Ontology-driven interaction pattern

Observe, Orient, Decide, Act, and Learn are all explicit Ontology actions and data subjects

The loop stays generic

The ontology supplies the domain model, the agent supplies the cycle

One action per tick

Bounded actuation beats unconstrained generation

An action is a governed command, not a REST endpoint

Schema, policy, prechecks, effects, transport, audit, and lifecycle travel together

ActionDefinition

schema	input/output types	**policy**	HUMAUTH + classification
prechecks	required bindings	**effects**	what state/artifact changes
runtime	sync or async transport	**audit**	fields + lineage

54 typed actions	**46** async commands	**8** sync commands
44 with prechecks	**54** function bindings	**52** with effects

The reasoner can propose. The platform still adjudicates.

Denial as data

A useful refusal tells the agent what failed, where it failed, and what evidence the platform used

failed_stage | error_kind | denial_id | reason | evidence | lineage

A useful refusal is a machine-readable branch, not a dead end

Agents need distributed-systems discipline

Statelessness, idempotency, async lifecycle, drift detection, and topology abstraction matter more than prompt tricks

Stateless replicas

No session affinity required

Idempotency

Retries collapse to one decision

Async lifecycle

Dispatch/result pairs

Drift checks

Checksum gates re-bootstrap

Federation

Storage topology hidden

Audit lineage

Every action leaves a trace

Agent safety is mostly ordinary platform engineering, made explicit

TAKEAWAY

Expose Ontology Actions and Subjects

Agents need semantically named capabilities, not opaque endpoints

1

DISCOVERABLE

Enumerate the world

2

TYPED

Validate every payload

3

SCOPED

Identity + role bound

4

GOVERNED

Policy outside prompt

5

REPLAY-SAFE

Idempotent retry

6

EXPLAINABLE

Denial with evidence

If an agent has to infer what an API means from documentation, the platform has outsourced control to the model

Thank You!

https://www.varindor.com