

AI-BOK TOOLKIT v1.2

ArchiMate Modelling Conventions

Companion to the AI Body of Knowledge v1.2

Jan Willem van Veen · ArchiXL · ai-bok.nl

ArchiMate 3.2 Modelling Conventions

ArchiMate 3.2 purist mapping for the AI-BOK constructs - cognition plane, Agent, Mandate, Authority Register, Failure Modes - without custom stereotypes. Enables adoption in organisations with strict EA-governance house rules.

1. Purpose

This note documents the **modelling-correct mapping** from AI-BOK constructs to ArchiMate 3.2 standard elements, without introducing custom stereotypes or profile abuse. It also documents the **practitioner variants** that the reference model uses for readability, so that adopters can choose between strict-purist and practitioner renderings depending on their EA-governance posture.

The release ships the ArchiMate reference model in **two variants**:

- `AI-BOK-Reference-Architecture-EN-v1.2-clean.archimate` - strict ArchiMate 3.2 only, this note's mappings applied throughout.
- `AI-BOK-Reference-Architecture-EN-v1.2-practitioner.archimate` - same model, with the additional practitioner conveniences (Pattern C RACI overlay, cognition-plane swimlane decorations, FM-icon overlays).

Element IDs are stable between the two variants. Tools that import either variant get the same logical model; only the visual rendering differs.

2. Cognition plane

AI-BOK construct	Strict ArchiMate 3.2 mapping	Practitioner variant
Cognition plane (as a layer)	A <code>Grouping</code> element labelled "Cognition Plane" containing the cognition-plane-resident elements. ArchiMate has no notion of a <i>layer</i> beyond Business / Application / Technology / Physical; the plane is a logical grouping, not a layer.	Same <code>Grouping</code> , with custom colour (teal in v1.1) and a layered backdrop in the view. The colour is decorative; the underlying construct is <code>Grouping</code> .
Data plane	<code>Grouping</code> labelled "Data Plane" containing relevant Application-Data and Technology elements.	Same.
Control plane	<code>Grouping</code> labelled "Control Plane" containing the identity / authorisation / contract / audit	Same.

AI-BOK construct	Strict ArchiMate 3.2 mapping	Practitioner variant
	elements.	

Rationale: ArchiMate's three named layers are concrete (Business / Application / Technology). The cognition plane is *cross-layer* - it has business-level constructs (mandates, policies), application-level constructs (policy-as-code services, authority-register endpoints) and technology-level constructs (NHI key material). A `Grouping` therefore models it correctly; introducing a custom Layer would break ArchiMate compliance.

3. Agent

AI-BOK construct	Strict ArchiMate 3.2 mapping	Practitioner variant
Agent (as an entity in the metamodel)	<code>ApplicationComponent</code> with a Specialization "Agent". Rationale: an agent is a software unit that delivers a behaviour and consumes/produces data - that is the definition of <code>ApplicationComponent</code> .	Same, plus an "Agent" icon.
Agent identity (NHI)	<code>Contract</code> (Business layer) labelled "NHI: " linked to the <code>ApplicationComponent</code> via <code>AssignmentRelationship</code> . Rationale: a <code>Contract</code> in ArchiMate is "a formal or informal specification of an agreement between a provider and a consumer that specifies the rights and obligations". An NHI is exactly that - an identity-as-credential with a specified scope.	Same.
Agent fleet	<code>Grouping</code> of <code>ApplicationComponent</code> instances; no fleet-as-first-class element.	Same; practitioner variant adds a "Fleet" decoration.

The model used a custom "Agent" element type. demotes this to `ApplicationComponent` + Specialization in the clean variant; the practitioner variant retains the icon.

4. Mandate, Authority Register, Policy-as-Code

AI-BOK construct	Strict ArchiMate 3.2 mapping	Practitioner variant
Mandate	<code>Contract</code> (Business layer) with attributes <code>scope</code> , <code>exclusivity</code> , <code>freshness_budget</code> , <code>operating_envelope</code> . Linked to the Agent	Same, plus a "Mandate" icon.

AI-BOK construct	Strict ArchiMate 3.2 mapping	Practitioner variant
	(<code>ApplicationComponent</code>) via <code>AssignmentRelationship</code> , and to a <code>BusinessObject</code> "Authority Register" via <code>AccessRelationship</code> .	
Authority Register	<code>BusinessObject</code> "Authority Register" - the persisted state. Implemented by an <code>ApplicationService</code> "Authority Register Service" (<code>RealizationRelationship</code>).	Same.
Policy-Consultation Protocol	<code>BusinessProcess</code> "Policy Consultation" between Agent and Authority Register. Produces a <code>BusinessObject</code> "Decision Artefact" (signed) via <code>AccessRelationship</code> .	Same.
Policy-as-Code	<code>ApplicationService</code> "Policy Engine" with <code>ApplicationComponent</code> "Policy Rules".	Same.
Escalation threshold	<code>Driver</code> (Motivation layer) "Escalation threshold" with <code>InfluenceRelationship</code> to the relevant <code>Goal</code> .	Same.

This mapping uses only ArchiMate 3.2 vocabulary. Anyone who imports the clean model into a strict-validation tool (BiZZdesign, Sparx EA, Archi with validation) will not see custom-type warnings.

5. Knowledge Source

AI-BOK construct	Strict ArchiMate 3.2 mapping	Practitioner variant
Knowledge Source (trusted source: ontology, knowledge graph, RAG index, signed document corpus, live sensor stream)	<code>BusinessObject</code> "Knowledge Source" with attributes <code>provenance</code> , <code>trust_grade</code> , <code>freshness</code> . Realised by an <code>ApplicationService</code> "Retrieval Service" (<code>RealizationRelationship</code>).	Same, with a "Knowledge" icon.
Sensor-Trust Grade	Property on the <code>BusinessObject</code> (runtime value).	Same.

The GEMMA-style pattern (cf. the VNG/BZK/GEMMA cross-walk §4) maps cleanly onto this - a local GEMMA register holds a `BusinessObject` per AI-System with a `RealizationRelationship` to an "AI-BOK reference object" `BusinessObject` defined in the AI-BOK reference model. No custom types required.

6. Failure modes (FM01-FM22)

AI-BOK construct	Strict ArchiMate 3.2 mapping	Practitioner variant
Failure mode (e.g. FM01 Prompt Injection)	<code>Assessment</code> (Motivation layer) in a folder "Failure Modes". Linked to affected KAs (Capabilities) via <code>AssociationRelationship</code> .	Same, with a per-FM icon.
Cognition-plane binding (which mandate / authority register / NHI / policy-as-code / escalation / KA12 binding carries the FM)	<code>InfluenceRelationship</code> from the <code>Assessment</code> (FM) to the relevant Mandate / Authority Register / Policy-Engine / Escalation Driver.	Same.
Mitigation	<code>Goal</code> (Motivation layer) realised by the cognition-plane elements that carry it (<code>RealizationRelationship</code>).	Same.

The model placed FM01-FM16 as Assessments correctly; FM17-FM22 were stubs. completes them and adds the cognition-plane-binding `InfluenceRelationship`s consistently for all 22.

7. Lifecycle phases and roles

AI-BOK construct	Strict ArchiMate 3.2 mapping
Lifecycle phase	<code>BusinessProcess</code> in the lifecycle group.
Role	<code>BusinessRole</code> .
RACI assignment	<code>AssignmentRelationship</code> between <code>BusinessRole</code> and <code>Capability</code> , carrying the R/A/C/I value as a Specialization (preferred) or a Property (acceptable).
Governance body (AI Board, AI Review Committee, AI Ethics Committee, AI Risk Committee)	<code>BusinessActor</code> (collective).

8. Metamodel entities

AI-BOK construct	Strict ArchiMate 3.2 mapping
AI System	<code>ApplicationComponent</code> with <code>deployment-tier</code> Property (device / edge / centre).
AI Model	<code>DataObject</code> or <code>ApplicationComponent</code> depending on whether it is treated as artefact (<code>DataObject</code>) or service (<code>ApplicationComponent</code>). standardises on

AI-BOK construct	Strict ArchiMate 3.2 mapping
	<code>ApplicationComponent</code> realised by a <code>DataObject</code> "Model Artefact".
Dataset	<code>DataObject</code> .
Prompt	<code>DataObject</code> .
Interaction	<code>BusinessProcess</code> (channel of human-AI interaction).
Decision	<code>BusinessObject</code> (Decision Type).
Knowledge Source	<code>BusinessObject</code> (see §5).
Agent	<code>ApplicationComponent</code> (see §3).
Governance Framework	<code>BusinessActor</code> (collective: AI Board etc., see §7) plus <code>BusinessObject</code> for the Charter document.
Risk Profile	<code>Assessment</code> (Motivation layer).

9. Things deliberately does *not* introduce as custom types

- "Layer" for cognition plane (use `Grouping`).
- "Agent" as a new element kind (use `ApplicationComponent` + Specialization).
- "Mandate" as a new element kind (use `Contract`).
- "NHI" as a new element kind (use `Contract`).
- "Failure Mode" as a new element kind (use `Assessment`).
- "RACI assignment" as a new relationship kind (use `AssignmentRelationship` with Specialization).

This means an organisation whose EA-governance policy disallows custom ArchiMate stereotypes can adopt the AI-BOK reference model unchanged. That is the outcome.

10. Two variants - choosing between them

If your context is...	Use
Strict ArchiMate 3.2 EA governance; tool validates against the spec; multiple downstream tools	<code>...-v1.2-clean.archimate</code>
Single-tool (Archi); readability and visual differentiation matter; you accept a documented practitioner stretch	<code>...-v1.2-practitioner.archimate</code>

If your context is...	Use
You want both	Use clean as canonical, generate practitioner as derived view

Both variants carry identical logical content. Switching is one tool-import away.

11. Acknowledgement

The substance of this note responds to the question of how to render the model in a strictly correct way. The Pattern C overlay is preserved in the practitioner variant; the clean variant gives strict-purist organisations the same model without it. The intent is that no organisation has to compromise its EA-governance posture to adopt the AI-BOK.