

KOAN Ontology

Core Specification v6.3.1

Il Ponte Operativo tra Business Umano e AI Generativa

Autore: Alessandro Bemporad

Revisore: Maddalena Grazzini

Versione: 6.3.1

Stato: Specifica Definitiva

Data: 06/07/2026

Authority: <https://koanontology.org>

Licenza: Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International — CC BY-NC-SA 4.0

Copyright e Licenza

KOAN Ontology è un'opera originale ideata, modellata e sviluppata da Alessandro Bemporad.

La presente specifica è distribuita secondo i termini della licenza **Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**. È consentito copiare, ridistribuire, adattare, modificare, estendere e costruire opere derivate a partire da questo testo, a condizione che:

- La paternità originale di KOAN Ontology sia attribuita chiaramente ad Alessandro Bemporad.
- Le opere derivate siano distribuite con la stessa licenza o con una licenza compatibile.
- Le versioni modificate, estese o specializzate non vengano presentate come specifica ufficiale di KOAN Ontology.
- L'uso della specifica o di opere derivate per scopi commerciali — inclusi prodotti software, piattaforme, servizi a pagamento — non è consentito senza autorizzazione esplicita e scritta di Alessandro Bemporad.

Riserva sui Namespace `koan_`. I namespace che iniziano con `koan_` sono riservati esclusivamente agli Entity Space registrati nel KOAN Registry ufficiale, amministrato da Alessandro Bemporad e dal team di KOAN Ontology. Le opere derivate distribuite sotto questa licenza possono estendere KOAN liberamente, ma devono adottare un namespace proprio che non inizi con `koan_`, salvo registrazione esplicita nel Registry ufficiale. La pubblicazione di Entity Space o Entity sotto namespace `koan_...` al di fuori del Registry ufficiale costituisce una violazione della presente licenza e dei diritti di paternità intellettuale dell'autore, indipendentemente dal rispetto delle altre condizioni CC BY-NC-SA.

La versione ufficiale di riferimento è identificata dall'authority: <https://koanontology.org>

Policy di utilizzo e licensing commerciale

1. Scopo della policy

La **KOAN Ontology Core Specification** è distribuita con licenza **CC BY-NC-SA 4.0**.

Questa policy chiarisce ulteriormente le autorizzazioni aggiuntive concesse da Alessandro Bemporad per l'uso pratico di KOAN Ontology e indica l'unico caso in cui è necessario concordare preventivamente una licenza commerciale.

Le autorizzazioni aggiuntive previste da questa policy si applicano, con effetto retroattivo, a tutte le versioni ufficiali di KOAN Ontology pubblicate a partire dalla versione 1.0.0.

2. Usi consentiti liberamente

KOAN Ontology può essere usata liberamente, anche in contesti professionali, per:

- uso, ricerca, sperimentazione e sviluppo interno a una persona o organizzazione;
- creazione e utilizzo di modelli KOAN;
- utilizzo con sistemi di AI;
- progettazione e sviluppo di software destinato all'uso interno;
- attività di consulenza;
- attività di formazione.

Non è richiesta alcuna autorizzazione preventiva per questi usi.

3. Corsi ufficiali e certificazioni ufficiali

Possono presentare corsi o programmi come **"ufficiali" KOAN Ontology**, e possono rilasciare **certificazioni ufficiali KOAN Ontology**, esclusivamente:

- Alessandro Bemporad;
- organizzazioni direttamente riconducibili ad Alessandro Bemporad;
- soggetti o organizzazioni accreditati personalmente e per iscritto da Alessandro Bemporad.

Chiunque può svolgere liberamente formazione e consulenza su KOAN Ontology, purché non presenti tali attività come ufficiali, certificate o approvate dal progetto KOAN senza autorizzazione scritta.

4. Quando serve una licenza commerciale

È necessario concordare preventivamente e per iscritto una licenza d'uso con Alessandro Bemporad **solo** quando ricorrono entrambe queste condizioni:

1. viene realizzato un sistema software che integra KOAN Ontology come componente funzionale;
2. tale sistema è destinato alla vendita, licenza, abbonamento, SaaS, white-label o altra forma di commercializzazione verso terzi.

Per integrazione si intende, ad esempio, software che genera, interpreta, valida, memorizza, interroga, trasforma o utilizza direttamente modelli KOAN come parte delle proprie funzionalità.

5. Attribuzione

L'attribuzione segue i termini della licenza **CC BY-NC-SA 4.0** e il normale buon senso rispetto al mezzo utilizzato.

Quando la specifica viene condivisa o KOAN viene presentata pubblicamente, l'origine dovrebbe essere indicata in modo chiaro e ragionevole, ad esempio:

KOAN Ontology è stata ideata e sviluppata da Alessandro Bemporad.

Quando opportuno, vanno aggiunti il riferimento alla specifica, al sito ufficiale e alla licenza Creative Commons applicabile.

6. Contatti e accordi

Per richiedere un accreditamento ufficiale o concordare una licenza commerciale per software destinato a terzi, è sufficiente contattare Alessandro Bemporad attraverso i riferimenti pubblicati su koanontology.org.

Ogni autorizzazione, accreditamento o licenza commerciale deve essere concordata **per iscritto**.

Changelog

v6.3.1 (06/07/2026)

E' stata meglio specificata la Policy di utilizzo e licensing commerciale di KOAN Ontology.

v6.3.0 (05/07/2026)

Versione evolutiva. Il campo `target` delle Characterization relazionali è rinominato in `target_as`, per rendere esplicita la sua funzione di contesto di destinazione e allineare semanticamente CHR e ASN.

- **`target_as` nelle CHR relazionali.** Una Characterization che applica una PTY `relation` dichiara ora `target_as` invece di `target`. Il campo indica l'ETT o il RLT specifico sotto cui la futura Entity di destinazione parteciperà alle Assertion conformi alla CHR.
- **Simmetria con le ASN.** Nelle Assertion, `target` continua a identificare la Entity concreta di destinazione, mentre `target_as` ne indica il contesto ETT/RLT. Il `target_as` della ASN deve coincidere con il `target_as` della CHR applicata.
- **Unicità delle CHR aggiornata.** L'identità di una CHR relazionale è ora determinata dalla terna `subject_as` + `property` + `target_as`. Per le CHR `attribute` e `measurement`, che non dichiarano `target_as`, resta la coppia `subject_as` + `property`.
- **Relazioni derivate aggiornate.** La coerenza tra CHR primaria e derivata usa ora `target_as`: il `target_as` della derivata coincide con il `subject_as` della primaria, e il `subject_as` della derivata coincide con il `target_as` della primaria.
- **Algoritmo di risoluzione aggiornato.** La Parte XIII usa `target_as` per selezionare il contesto della CHR derivata e per filtrare correttamente le ASN della relazione primaria.
- **Nota di migrazione.** Nei blocchi `[[characterization]]` che applicano PTY `relation`, rinominare `target = "..."` in `target_as = "..."`. Il campo `target` delle `[[assertion]]` non cambia.

v6.2.0 (05/07/2026)

Versione evolutiva retrocompatibile. Le Assertion possono ora dichiarare il campo opzionale `lang`, applicabile a qualunque archetipo di PTY, per qualificare linguisticamente l'asserzione nel suo complesso.

- **Campo `lang` opzionale nelle ASN.** Ogni blocco `[[assertion]]` può dichiarare `lang` come codice lingua BCP 47, per esempio `"it"`, `"en"`, `"fr"`, oppure come stringa vuota `""`.
- **Applicabilità generale.** `lang` può essere usato con ASN basate su PTY `attribute`, `relation` o `measurement`. La qualificazione linguistica riguarda l'asserzione nel suo complesso e non modifica la forma del valore richiesta dall'archetipo della PTY.
- **ASN linguisticamente neutre.** Se `lang` è omissso oppure vale `""`, l'ASN è considerata linguisticamente neutra: è valida indipendentemente dalla lingua, oppure la dimensione linguistica non è significativa per quell'asserzione. Le due situazioni non vengono distinte dal core.
- **Aggiornata la Definizione 27.** La Parte XII specifica sintassi, significato e vincoli del nuovo campo `lang`, con esempi su archetipi diversi.

v6.1.0 (03/07/2026)

Versione evolutiva retrocompatibile. Le PTY `relation` primarie possono ora dichiarare anche `target_kind = "MST"`, per modellare relazioni che scelgono o referenziano unità di misura senza registrare un valore numerico.

- **`target_kind = "MST"` ammesso nelle PTY `relation` primarie.** Una relazione può ora puntare a un Measurement Type (MST) quando il suo valore atteso è una Measurement Unit (MSU) appartenente a quella grandezza. Questo consente proprietà come unità preferita, unità di visualizzazione, unità ammessa o unità operativa predefinita.
- **Distinzione esplicita tra RLN verso MST e MSM.** Una PTY `measurement` registra una misura, cioè un valore numerico insieme a una unità. Una PTY `relation` con `target_kind = "MST"` registra invece soltanto un riferimento a una MSU, senza alcun valore numerico.
- **Aggiornate le Definizioni 23, 26 e 27.** La Definizione 23 chiarisce che `target_kind` guida il tipo di `target` ammesso nelle Characterization che applicano la PTY. Le regole di Characterization, le regole di validazione delle Assertion e gli esempi sono estesi da quattro a cinque tipi di destinazione: `"CLS"`, `"SCH"`, `"MST"`, `"RST"`, `"RLT"`.

v6.0.0 (03/07/2026)

Versione con modifiche non retrocompatibili (major bump SemVer). Il tipo di destinazione di una `relation` si sposta dalla CHR alla PTY.

- **`target_kind` dichiarato sulla PTY `relation`, non più sulla CHR.** Il tipo di destinazione di una relazione — `"CLS"`, `"SCH"`, `"RST"` o `"RLT"` — è una proprietà stabile del predicato: non cambia con il contesto d'uso. La stessa relazione punta sempre allo stesso tipo di bersaglio; se il tipo cambia, si tratta di un'altra relazione. Per questo `target_kind` è ora un campo della PTY `relation` (Definizione 23) e non più della Characterization. La CHR conserva soltanto ciò che è genuinamente contestuale: il `target` **specifico** (quale CLS, quale SCH, ecc.) e le cardinalità. Modifica non retrocompatibile.
- **Le relazioni derivate non dichiarano `target_kind`.** Una relazione derivata (con `inverse_of`) eredita il proprio tipo di destinazione dalla relazione primaria, letta al contrario: il `target_kind` della derivata è il tipo del *source* della primaria, ed è quindi implicito. Il campo **NON DEVE** essere dichiarato sulle PTY derivate.
- **Aggiornate le Definizioni 19 e 22.** La Parte IX (PTY) dichiara `target_kind` sulla `relation` primaria; la Parte XI (CHR) non lo richiede più, e la sua verifica di coerenza col tipo reale del `target` resta a carico del parser.
- **Campo `target` della PTY `measurement` rinominato in `mst`.** Per precisione e per evitare la collisione con il `target` di `relation` e delle Assertion, la PTY `measurement` dichiara ora la grandezza nel campo `mst` (in luogo di `target`). Modifica non retrocompatibile.

v5.0.0 (02/07/2026)

Versione con modifiche non retrocompatibili (major bump SemVer). Ridisegno degli archetipi di Property Type.

- **Archetipi PTY ridotti da cinque a tre.** Gli archetipi di Property Type sono ora `attribute`, `relation`, `measurement`. Gli archetipi `classification` e `resource` sono **rimossi** in quanto tali: erano casi particolari di `relation` (un riferimento a un'altra Entity), distinti solo dal tipo di destinazione. Una proprietà che in precedenza era una `classification` è ora una `relation` il cui `target`, dichiarato nella Characterization, è uno SCH; una proprietà che era una `resource` è una `relation` il cui `target` è un RST. La forma dei dati che un archetipo cattura è di tre specie soltanto — un valore scalare (`attribute`), un riferimento a un'Entity (`relation`), un valore numerico con unità (`measurement`) — e gli archetipi ora coincidono con queste tre forme, senza sovrapposizioni. Modifica non retrocompatibile.
- **Il `target` di una `relation` è dichiarato nella CHR, con la sua sigla esplicita (`target_kind`).** Una PTY `relation` continua a non dichiarare il proprio `target`: è la Characterization a fissarlo. La CHR di una `relation` **DEVE** ora dichiarare, oltre a `target`, anche `target_kind`: la sigla del tipo di destinazione, tra `"CLS"`, `"SCH"`, `"RST"`, `"RLT"`. La sigla è ridondante rispetto al tipo effettivo dell'Entity indicata in `target`, e proprio per questo esplicita in chiaro la natura del riferimento — categoria di uno schema (SCH), risorsa esterna (RST), oggetto di una classe (CLS), Entity in un ruolo (RLT) — rendendo l'intento leggibile senza risalire alla definizione del `target`. Un parser KOAN **DEVE** verificare che `target_kind` coincida con il tipo reale dell'Entity in `target`. Il `target` di una `relation` **NON DEVE**

essere un MST: le grandezze con unità sono coperte dall'archetipo `measurement`.

- **value_type e MST restano sulla PTY.** Coerentemente con il principio per cui ciò che è invariante rispetto al contesto appartiene al predicato e ciò che è contestuale appartiene alla Characterization: il `value_type` di un `attribute` e il MST di una `measurement` sono parte dell'identità del predicato e restano dichiarati sulla PTY. Solo il target di una `relation`, che è genuinamente contestuale, è dichiarato nella CHR.
- **Aggiornate le Definizioni 19, 22, 23.** La Parte IX (PTY), la Parte XI (CHR) e la Parte XII (ASN) sono riscritte per riflettere i tre archetipi. Nota di migrazione: le `classification` diventano `relation` con `target_kind = "SCH"`; le `resource` diventano `relation` con `target_kind = "RST"`. Le RES conservano `locator` e `media_type` sulla propria Entity: nulla si perde, sparisce solo l'archetipo PTY dedicato.

v4.0.0 (02/07/2026)

Versione con modifiche non retrocompatibili (major bump SemVer).

- **Campo `status` rimosso dal blocco `[[role_assignment]]`.** L'assunzione di ruolo non dichiara più un campo `status`, né il relativo insieme di valori (`planned`, `active`, `suspended`, `ended`, `revoked`), né alcun valore predefinito. Un'assunzione di ruolo è ora definita dai soli campi `entity` e `role` (obbligatori) e dai campi temporali `valid_from` e `valid_until` (opzionali). Lo `status` era un enumerato universale che pretendeva di valere per ogni dominio, mentre gli stati di un ruolo sono conoscenza di dominio; inoltre la sua sovrapposizione ai campi temporali generava combinazioni ambigue. Il core attribuisce all'assunzione di ruolo la sola dimensione temporale. Nota di migrazione: un dominio che necessita di stati di ruolo può modellarli con gli strumenti generali di KOAN, come una classificazione — una PTY con `archetype = "classification"` associata al RLT tramite una Characterization, i cui valori sono le categorie di uno SCH che raccoglie gli stati previsti dal dominio.

v3.0.0 (30/06/2026)

Versione con modifiche non retrocompatibili (major bump SemVer).

- **Campo `instance_of` rinominato in `instance_of`.** Il campo che esprime la relazione di istanziazione tra un Particular e il suo Entity Type si chiama ora `instance_of`. Il nuovo nome dichiara esplicitamente la relazione (l'Entity è *istanza di* un tipo) e si distingue nettamente da `kind` e `archetype`. La modifica riguarda tutti gli archetipi (OBJ→CLS, CTG→SCH, MSU→MST, RES→RST) e i riferimenti qualificati cross-namespace. Modifica non retrocompatibile.
- **Relazione di istanziazione ridefinita.** La relazione tra un Particular e il suo Entity Type è ridefinita: un Entity Type cattura una comunanza in un insieme di Entity, e le sue istanze sono le Entity che cadono nella sua estensione. Abbandonata la precedente etichetta "IS-A".
- **Divieto di istanziazione tra Entity Type.** Un Entity Type **NON DEVE** essere istanza di alcun Entity Type. Di conseguenza non esistono catene di istanziazione: ogni `instance_of` collega un Particular terminale al suo Entity Type, in un solo passaggio. La non-transitività diventa una proprietà strutturale del modello. Per predicare sui tipi si usano gli schemi di classificazione (SCH/CTG) o i ruoli (RLT). Modifica non retrocompatibile.
- **Comunanza per archetipo.** Esplicitato il modo in cui ciascun archetipo di Entity Type declina la comunanza catturata: CLS (natura comune di un insieme indefinito e non ordinato di oggetti), SCH (categorie di un solo schema, in numero definito, eventualmente ordinato o gerarchico), MST (unità di misura commensurabili di una stessa grandezza), RST (risorse digitali esterne dello stesso genere).

v2.2.3 (29/06/2026)

- **Terminologia: etichette neutre + per-lingua con merge e override.** Un blocco `lang = ""` definisce forme di base valide per ogni lingua; i blocchi con lingua esplicita vi si sovrappongono. Risoluzione per lingua: il `name` specifico fa override (il name base resta cercabile); `alias` e `hidden` si uniscono senza duplicati; il `sort_name` specifico prevale sul base. Al più un blocco `lang = ""` e uno per lingua, per Entity.
- **Terminologia: `name` obbligatorio a livello di Entity.** Il `name` non è più obbligatorio in ogni blocco, ma deve essere

risolvibile per ogni lingua dichiarata — fornito dal blocco base `lang = ""` o da ciascun blocco-lingua. Un blocco-lingua può così limitarsi ad aggiungere `alias`, `hidden` o `sort_name`.

v2.2.2 (29/06/2026)

- **Terminologia: `display_name` rimosso.** Il campo `display_name` è eliminato in quanto ridondante: nella stragrande maggioranza dei casi coincide con il `name` (preferred label), che è già di per sé la forma canonica di visualizzazione. Le convenzioni puramente visive (es. classi in maiuscolo) competono al livello di presentazione, non al modello.
- **Terminologia: `sort_name` introdotto.** Nuovo campo opzionale per la forma di ordinamento alfabetico nelle liste (es. "Da Vinci, Leonardo" per `name = "Leonardo da Vinci"`). Non è cercabile; può essere visualizzato nelle viste in ordine alfabetico. Se omesso, coincide con il `name`.
- **Terminologia: visibilità di `alias` e `hidden` chiarita.** `alias` è cercabile e visualizzato come etichetta secondaria (tipicamente dopo il `name`). `hidden` è cercabile ma **mai** visualizzato.

v2.2.1 (29/06/2026)

- **Entity Space standard `koan_core`.** Introdotto `koan_core` come Entity Space standard e fondante di KOAN Ontology (Definizione 5). Contiene primitive trasversali — unità di misura, relazioni standard e altri elementi comuni — su cui ogni modello può poggiare. È sempre disponibile e **PUÒ** essere referenziato senza essere importato tramite `[[namespace]]`. Le sue Entity vanno sempre referenziate in forma qualificata (`koan_core@...`); `koan_core` **NON DEVE** essere usato come `default_namespace`.
- **`default_namespace` condizionale.** Il campo `default_namespace` di `[koan]` è obbligatorio solo se il file contiene almeno un identificatore non qualificato; **PUÒ** essere omesso se tutti gli identificatori sono qualificati. Il blocco `[koan]` resta comunque obbligatorio.

v2.2 (29/06/2026)

- **Nuovo blocco `[entity_space]`.** Introdotto un blocco dedicato alla dichiarazione ufficiale di un Entity Space, con i campi `namespace`, `version`, `authority`, `uri`. Costituisce la fonte anagrafica dello spazio. Obbligatorio, uno per file.
- **`[koan]` semplificato.** Il blocco di intestazione contiene ora un solo campo, `default_namespace`, che indica il namespace predefinito per gli identificatori privi del simbolo `@`. Il campo `version` è stato rimosso da `[koan]` (la versione appartiene all'Entity Space, dichiarata in `[entity_space]`).
- **Materializzazione del namespace.** In fase di serializzazione verso altri sistemi, il namespace predefinito **DEVE** essere materializzato fisicamente davanti agli identificatori non qualificati.
- **`[[namespace]]` arricchito.** Il blocco di importazione di Entity Space esterni adotta i campi `namespace`, `version`, `authority`, `uri`, in simmetria con `[entity_space]`. Il campo `version` rende esplicita la versione dello spazio importato. Il campo precedentemente denominato `id` è ora `namespace`.
- **Vincolo sul `default_namespace`.** Deve riferirsi all'Entity Space dichiarato nel blocco `[entity_space]` dello stesso file, oppure a uno degli Entity Space importati tramite `[[namespace]]`.
- **Cardinalità `[entity_space]`.** Un file contiene al più un blocco `[entity_space]`. Un file che dichiara Entity proprie deve contenerlo; un file che opera solo su Entity importate tramite `[[namespace]]` può non contenerlo.

v2.1 (27/06/2026)

- Versione di riferimento precedente.

Una Nota Prima di Cominciare

L'Ontologia Applicata accompagna il mio lavoro da quasi vent'anni.

In questo periodo ho progettato numerosi modelli ontologici formali destinati a rappresentare domini complessi, successivamente implementati in piattaforme software e soluzioni applicative in ambiti molto diversi, dall'healthcare ai beni culturali, fino ai sistemi informativi per l'industria.

In tutti questi anni il problema è rimasto sostanzialmente lo stesso: trasformare la conoscenza di un dominio in una struttura esplicita, rigorosa e condivisibile. Un modello sufficientemente preciso da poter essere compreso da persone diverse, mantenuto nel tempo e tradotto in sistemi software complessi.

L'arrivo dell'Intelligenza Artificiale Generativa ha cambiato profondamente questo scenario.

Per la prima volta, un modello ontologico non rappresenta più soltanto un linguaggio condiviso tra analisti e sviluppatori, o la base di un sofisticato motore di persistenza. Può diventare anche il linguaggio condiviso tra esseri umani e Intelligenze Artificiali.

KOAN Ontology nasce proprio da questa convinzione.

KOAN rappresenta il punto di arrivo del percorso che mi ha portato fin qui e, spero, il punto di partenza di un percorso molto più ampio che altri potranno continuare. È stato progettato affinché un'AI Generativa possa comprenderlo, verificarlo e persino costruirlo direttamente a partire da requisiti espressi in linguaggio naturale o pseudo-naturale.

Un modello KOAN non è il software finale. È la descrizione formale della conoscenza su cui un sistema intelligente può ragionare e, a partire dalla quale, contribuire alla progettazione e alla generazione automatica del software stesso.

Per questo motivo ho scelto di rendere pubblica la specifica di KOAN Ontology e di distribuirla con una licenza aperta.

Credo che un linguaggio destinato a diventare il punto di incontro tra analisti, sviluppatori e Intelligenze Artificiali debba poter essere studiato, utilizzato, implementato e migliorato dalla comunità. Desidero che KOAN possa essere adottato nella ricerca, nella formazione, nelle organizzazioni e nello sviluppo di strumenti capaci di collaborare con l'AI nella progettazione di sistemi software complessi.

La specifica è quindi liberamente utilizzabile per lo studio, la ricerca, la formazione, l'addestramento di modelli di Intelligenza Artificiale e lo sviluppo di strumenti software per uso interno. Rimangono invece disciplinate da specifiche condizioni di licenza le implementazioni commerciali che integrano KOAN come engine ontologico all'interno di prodotti o servizi software.

Il mio auspicio è che KOAN contribuisca alla nascita di un linguaggio comune per la modellazione della conoscenza nell'era dell'Intelligenza Artificiale. Un linguaggio capace di ridurre la distanza tra requisiti espressi dagli esseri umani, modelli formali della conoscenza e sistemi software costruiti con il supporto dell'AI.

Alessandro Bemporad, Firenze, 27/06/2026

Fondamenti: Dominio, Modello, Conoscenza

Prima di descrivere KOAN, occorre definire con precisione i concetti su cui esso poggia.

Il Dominio

Chiamiamo **dominio** una porzione delimitata del mondo — reale o concettuale — che un sistema intende osservare, governare o riprodurre. Un dominio può essere un'azienda, un processo produttivo, un ecosistema biologico, una missione spaziale. Ciò che lo distingue dal mondo intero è il confine: il dominio è ciò che il sistema ha deciso di guardare.

Il confine del dominio non è una proprietà del mondo: è una scelta dell'osservatore. Definire un dominio è già un atto cognitivo e intenzionale.

Il Modello

Un **modello** è la rappresentazione formale di un dominio. Non è il dominio di per sé: è la mappa, non il territorio. Il modello seleziona, astrae e organizza gli elementi del dominio rilevanti per uno scopo preciso, escludendo tutto il resto.

Un buon modello non aspira alla completezza enciclopedica. Aspira alla coerenza interna e alla fedeltà rispetto agli scopi per cui è stato costruito.

La Conoscenza del Dominio

Ogni dominio genera, nel tempo, una conoscenza stratificata: le regole che ne governano il funzionamento, i vincoli operativi, le relazioni tra i suoi elementi, le classificazioni che permettono di orientarsi. Questa conoscenza è il patrimonio più prezioso di qualsiasi organizzazione.

Il problema è che, nella grande maggioranza dei casi, questa conoscenza rimane implicita. Vive frammentata in diagrammi informali, nella testa degli esperti, sepolta dentro il codice software. Quando un nuovo membro del team deve capire il sistema, o quando un'Intelligenza Artificiale deve generare codice corretto, entrambi si trovano a dover ricostruire quella mappa da zero — con tutto il rischio di errori che ne consegue.

KOAN nasce per rendere questa conoscenza esplicita, formale e trasferibile.

Il Formato TOML

KOAN utilizza TOML come formato di serializzazione dei modelli. TOML è semplice da leggere e scrivere per un essere umano, ma sufficientemente strutturato da fornire indicazioni precise e non ambigue a una AI Generativa. Questa scelta non è estetica: è la dichiarazione di un'intenzione. KOAN rifiuta la complessità gratuita.

Il risultato è uno strumento simmetrico: l'analista umano enuncia il dominio in KOAN, o attraverso dei requisiti espressi in linguaggio naturale, e lo condivide con una AI. A sua volta l'AI può interpretare un modello KOAN, o può produrlo autonomamente sulla base dei requisiti ricevuti. Una grammatica condivisa che elimina il rumore tra l'intenzione e l'esecuzione, che serve come base per la produzione automatica di sistemi software complessi da parte di AI Generative.

Convenzioni Normative

All'interno di questa specifica, alcune parole chiave hanno un significato tecnico preciso:

- **DEVE / NON DEVE** — Vincolo logico assoluto. La sua violazione rende il modello non valido.
- **DOVREBBE / NON DOVREBBE** — Forte raccomandazione. La violazione è tollerata in fase di editing incrementale, ma segnala un'incoerenza potenziale.
- **PUÒ** — Facoltà opzionale, pienamente supportata dalla specifica.

Parte I — Entity Space e Identità

Definizione 1. Identificatore (`id`)

Un **identificatore** è una stringa stabile che fissa l'identità logica di un elemento del modello. Un identificatore non è un nome naturale, non appartiene a una lingua e non cambia quando cambiano le etichette usate dagli esseri umani.

Gli esempi della specifica possono usare prefissi convenzionali come `cls.`, `obj.`, `sch.`, `ctg.`, `mst.`, `msu.`, `rst.`, `res.`, `pty.` o `rol.` all'interno dell'identificatore. Tali prefissi non hanno valore normativo autonomo: servono soltanto a rendere gli identificatori più leggibili.

Definizione 2. Namespace

Un **namespace** è un'etichetta che identifica un Entity Space. La struttura sintattica di un identificatore completo è:

```
namespace@local_id
```

La parte prima del simbolo `@` indica l'Entity Space; la parte dopo identifica un singolo elemento al suo interno.

Quando un identificatore non contiene il simbolo `@`, esso viene interpretato nel namespace predefinito dichiarato dal campo `default_namespace` dell'intestazione `[koan]` (Definizione 6).

In fase di serializzazione verso altri sistemi, un sistema KOAN **DEVE** materializzare fisicamente il namespace predefinito davanti a ogni identificatore privo del simbolo `@`, in modo che l'identità resti inequivocabile anche fuori dal file di origine.

Esempio di identificatore qualificato (*un'Entity di un Entity Space locale fa riferimento, tramite il campo `instance_of`, a una classe definita in un Entity Space esterno importato come `parts_catalog`*):

```
[[entity]]
kind = "PAR"
archetype = "object"
id = "obj.press_pr500"
instance_of = "parts_catalog@cls.industrial_machine"
```

Definizione 3. Entity Space

Un **Entity Space** è lo spazio entro cui vengono dichiarate e interpretate le Entity di un modello KOAN. Un Entity Space contiene sia le Entity (dichiarate in blocchi `[[entity]]`) sia i blocchi strutturali che le descrivono: Characterization, Assertion, Role Assignment, Terminology. Solo le Entity hanno un `id` e un'identità propria; i blocchi strutturali esistono soltanto in relazione alle Entity che descrivono.

Ogni Entity Space è identificato da un namespace. Il namespace è l'identificativo dell'Entity Space; l'Entity Space è lo spazio reale di cui il namespace è l'identificativo.

Un file TOML non coincide con l'Entity Space: è una possibile serializzazione dell'Entity Space o di una sua parte. Uno stesso Entity Space **PUÒ** essere distribuito e rappresentato in più file TOML. Le condizioni che ciascun file deve rispettare sono dichiarate nelle Definizioni 5, 5-bis e 5-ter.

Definizione 4. KOAN Registry

Il **KOAN Registry** è il registro ufficiale degli Entity Space destinati allo scambio pubblico. I namespace che iniziano con `koan_` sono riservati al KOAN Registry ufficiale.

Un Entity Space con namespace nella forma `koan_xxx` ha validità globale solo se registrato presso `koanontology.org`, dove `xxx` è garantito univoco dal Registry ufficiale.

Un'organizzazione può creare Entity Space locali usando namespace propri, purché non inizino con `koan_`. In assenza di registrazione ufficiale, tali Entity Space hanno valore locale e non garantiscono unicità globale.

Procedura di standardizzazione

Un'organizzazione che abbia sviluppato un modello KOAN e intenda elevarlo a standard — affinché chiunque possa utilizzarlo liberamente — **PUÒ** sottoporlo a KOAN Ontology inviandolo all'indirizzo registry@koanontology.org.

Il team ufficiale di KOAN Ontology ha il diritto di esaminare il modello e di decidere, in completa autonomia, se accoglierlo nel Registry, rifiutarlo, oppure richiedere modifiche prima di concludere la valutazione.

In caso di esito positivo, KOAN Ontology:

- attribuisce al modello un namespace nella forma `koan_xxx`, garantito univoco;
- lo pubblica nel public registry su `koanontology.org`;
- rende disponibile, nel medesimo registry, un archivio ZIP contenente i file TOML del modello.

Definizione 5. Entity Space standard (`koan_core`)

L'Entity Space `koan_core` è lo spazio standard e fondante di KOAN Ontology. Contiene le primitive trasversali su cui qualunque modello può poggiare: unità di misura standard, relazioni di rilevanza generale e altri elementi comuni definiti e mantenuti ufficialmente da KOAN Ontology.

`koan_core` gode di uno statuto particolare rispetto a ogni altro Entity Space esterno:

- È **sempre disponibile**. **PUÒ** essere referenziato da qualunque modello senza essere importato mediante un blocco `[[namespace]]`. Costituisce l'unica eccezione alla regola per cui un Entity Space esterno deve essere importato prima di poter essere referenziato.
- Le sue Entity **DEVONO** essere referenziate sempre in forma qualificata, con il namespace per esteso: `koan_core@local_id`. `koan_core` non dispone di alcun comportamento predefinito.
- `koan_core` **NON DEVE** essere usato come valore di `default_namespace` nel blocco `[koan]`.

Esempio (un modello referencia l'unità di misura standard del chilogrammo, definita in `koan_core`, senza importarla):

```
[[assertion]]
subject = "obj.module_hydroponics"
subject_as = "cls.ship_module"
property = "pty.module_mass"
value = 18500.0
target = "koan_core@msu.kilogram"
target_as = "koan_core@mst.mass"
```

Definizione 6. Intestazione di serializzazione (`[koan]`)

Il blocco iniziale `[koan]` raccoglie i metadati strutturali della serializzazione KOAN. Ogni file TOML **DEVE** contenere il proprio blocco `[koan]`.

Il blocco `[koan]` **PUÒ** dichiarare:

- `default_namespace`: il namespace predefinito usato per interpretare tutti gli identificatori privi del simbolo `@` presenti nel file. In fase di serializzazione verso altri sistemi, è il namespace che viene aggiunto automaticamente davanti a ogni identificatore non qualificato.

Il campo `default_namespace` **DEVE** essere dichiarato se il file contiene almeno un identificatore non qualificato. **PUÒ** essere omissso se tutti gli identificatori presenti nel file sono qualificati — condizione verificabile da un parser KOAN. In ogni caso il blocco `[koan]` resta obbligatorio in quanto tale.

Quando dichiarato, il valore di `default_namespace` **DEVE** riferirsi a uno tra:

- il namespace dichiarato nel blocco `[entity_space]` dello stesso file (Definizione 7);
- il namespace di uno degli Entity Space importati tramite un blocco `[[namespace]]` (Definizione 8).

Il valore di `default_namespace` **NON DEVE** essere `koan_core` (Definizione 5).

Il blocco `[koan]` è predisposto per ospitare ulteriori metadati di serializzazione in versioni future della specifica.

Esempio:

```
[koan]
default_namespace = "plant"
```

Definizione 7. Dichiarazione dell'Entity Space (`[entity_space]`)

Il blocco `[entity_space]` **dichiara** l'Entity Space rappresentato dal file. È la fonte anagrafica ufficiale dello spazio: il punto in cui l'Entity Space viene definito e qualificato con le proprie coordinate.

Un file TOML **PUÒ** contenere al più un blocco `[entity_space]`. Un file che dichiara Entity proprie **DEVE** contenere il blocco `[entity_space]` dello spazio a cui tali Entity appartengono. Un file che non dichiara Entity proprie — ma opera esclusivamente su Entity dichiarate in Entity Space esterni, importati tramite `[[namespace]]` — **PUÒ** non contenere alcun blocco `[entity_space]`.

Il blocco `[entity_space]` **DEVE** dichiarare:

- `namespace`: il namespace che identifica l'Entity Space.
- `version`: la versione dell'Entity Space, in formato semantico.
- `authority`: l'entità (organizzazione, gruppo, persona) che amministra ufficialmente l'Entity Space.
- `uri`: l'URI ufficiale di riferimento dell'Entity Space.

Se un Entity Space è distribuito su più file TOML, **ogni** file che contiene Entity di quello spazio **DEVE** contenere un blocco `[entity_space]` identico in tutti i suoi campi.

Esempio:

```
[entity_space]
namespace = "plant"
version = "1.0.0"
authority = "Example Authority"
uri = "https://example.org/plant"
```

Definizione 8. Importazione di Entity Space esterni (`[[namespace]]`)

Il blocco `[[namespace]]` **importa** un Entity Space dichiarato altrove — tipicamente in altri file TOML, che dovrebbero essere collocati accanto al file corrente o comunque raggiungibili da un parser KOAN.

Mentre `[entity_space]` introduce un nuovo Entity Space, `[[namespace]]` rende disponibili al modello corrente le Entity di Entity Space esterni, permettendo agli identificatori qualificati di referenziarle.

Ogni blocco `[[namespace]]` **DEVE** dichiarare:

- `namespace`: il namespace dell'Entity Space importato.
- `version`: la versione dell'Entity Space che si sta importando.
- `authority`: l'entità che amministra ufficialmente l'Entity Space importato.
- `uri`: l'URI ufficiale di riferimento dell'Entity Space importato.

Esempio:

```
[[namespace]]
namespace = "parts_catalog"
version = "3.1.0"
authority = "Industrial Standards Group"
uri = "https://example.org/parts_catalog"
```

I due blocchi `[entity_space]` e `[[namespace]]` condividono la stessa struttura di campi. La differenza è direzionale: `[entity_space]` introduce un nuovo Entity Space (*io dichiaro me stesso*); `[[namespace]]` importa un Entity Space esistente (*io cito un altro*). Visto da un terzo modello, il blocco `[entity_space]` di uno spazio è esattamente l'informazione che quel terzo modello riporterà in un proprio `[[namespace]]` per importarlo.

Definizione 9. Esempio completo di intestazione

```
[koan]
default_namespace = "andromeda"

[entity_space]
namespace = "andromeda"
version = "1.0.0"
authority = "Andromeda Exploration Initiative"
uri = "https://example.org/andromeda"

[[namespace]]
namespace = "parts_catalog"
version = "3.1.0"
authority = "Industrial Standards Group"
uri = "https://example.org/parts_catalog"
```

In questo file: lo spazio `andromeda` è dichiarato come spazio proprio; lo spazio `parts_catalog` è importato; gli identificatori senza `@` appartengono ad `andromeda`; gli identificatori qualificati `parts_catalog@...` riferenziano lo spazio importato.

Parte II — Entity (ENT)

Definizione 10. Entity (ENT)

Una **Entity (ENT)** è l'unità elementare sussistente del modello: qualunque elemento che il modello riconosca come distinto è una Entity.

Ogni Entity **DEVE** dichiarare un identificatore stabile mediante il campo `id`.

Ogni Entity **DEVE** dichiarare una famiglia fondamentale mediante il campo `kind`. Le famiglie fondamentali sono definite nelle parti successive della specifica.

Ogni Entity **DEVE** comparire all'interno di un blocco `[[entity]]`.

Esempio minimo:

```
[[entity]]
kind = "RLT"
id = "rol.operator"
```

Parte III — Terminologia (TRM)

Definizione 11. Struttura Terminologica

In KOAN l'identità di una Entity è separata dalle sue manifestazioni linguistiche. L'`id` è un identificatore logico stabile, privo di lingua. Non cambia se il modello viene tradotto e non cambia se il termine preferito evolve.

La terminologia è il riflesso linguistico e documentale di una Entity. Essa non si dichiara inline dentro l'Entity: è dichiarata in blocchi `[[terminology]]` dedicati.

Il blocco TRM **non è una Entity**. Non ha un `id`, non riceve ruoli, non riceve Characterization e non può essere soggetto di Assertion.

I blocchi `[[terminology]]` possono risiedere nello stesso file delle Entity, anche subito dopo l'Entity a cui si riferiscono, oppure in file dizionario separati.

Campi obbligatori

Ogni blocco `[[terminology]]` **DEVE** dichiarare:

- `entity`: id della Entity referenziata.
- `lang`: codice lingua secondo lo standard BCP 47, per esempio `"it"`, `"en"`, `"fr"`.
- `name`: etichetta preferita per quella lingua. In altri contesti è conosciuta spesso anche come *"preferred name"*.

Per etichette che non appartengono ad alcuna lingua naturale — come acronimi tecnici, codici o simboli universali — si usa la stringa vuota:

```
lang = ""
```

Etichette neutre ed etichette per lingua

Un blocco con `lang = ""` definisce le forme **di base**, valide per ogni lingua. Un blocco con un codice lingua esplicito (`it`, `en`, ...) definisce le forme **specifiche** per quella lingua, che si sovrappongono alle forme di base.

La terminologia effettiva per una lingua `xx` si ottiene combinando il blocco base (`lang = ""`, se presente) con il blocco specifico (`lang = "xx"`, se presente), secondo queste regole di risoluzione:

- `name`: se il blocco `xx` dichiara un `name`, esso è il preferred label per `xx`; il `name` del blocco base, in tal caso, resta valido come forma cercabile. Se il blocco `xx` non dichiara `name`, vale il `name` del blocco base.
- `alias`, `hidden`: unione delle forme del blocco base e del blocco `xx`, senza duplicati.
- `sort_name`: se il blocco `xx` lo dichiara, prevale; altrimenti vale quello del blocco base, se presente.

Per una stessa Entity **PUÒ** esistere al più un blocco `lang = ""` e al più un blocco per ciascuna lingua esplicita.

Obbligatorietà del `name`

Il `name` è obbligatorio **a livello di Entity**, non di singolo blocco: ogni Entity dotata di terminologia **DEVE** avere un `name` risolvibile per ogni lingua che dichiara. Questa condizione è soddisfatta se:

- esiste un blocco base `lang = ""` che dichiara un `name` — valido per tutte le lingue; **oppure**
- ogni blocco con lingua esplicita dichiara il proprio `name`.

Di conseguenza, un blocco con lingua esplicita **PUÒ** omettere il `name` quando un blocco base lo fornisce, limitandosi ad aggiungere `alias`, `hidden` o `sort_name`.

Campi delle etichette

Per ogni coppia `entity` + `lang`, il campo `name`, **quando presente, DEVE** essere unico. Esso rappresenta l'etichetta preferita della Entity in quella lingua ed è l'etichetta principale usata dai motori di ricerca basati su KOAN. L'obbligatorietà del `name` è definita a livello di Entity (vedi "Obbligatorietà del `name`"), non di singolo blocco.

Un blocco `[[terminology]]` **PUÒ** inoltre dichiarare:

- `alias`: array di etichette alternative accettabili, considerate equivalenti al `name` ai fini della ricerca. Gli alias sono **cercabili e visualizzati** come etichette secondarie, tipicamente accanto al `name` (per esempio tra parentesi).
- `hidden`: array di etichette nascoste, come varianti ortografiche, errori frequenti o termini deprecati. Gli hidden sono **cercabili ma mai visualizzati**.
- `sort_name`: forma dell'etichetta usata per l'ordinamento alfabetico nelle liste — per esempio `"Da Vinci,`

Leonardo" quando il `name` è "Leonardo da Vinci". Il `sort_name` **non** è usato per la ricerca. **PUÒ** essere visualizzato, tipicamente nelle viste in ordine alfabetico, dove sostituisce il `name` per riflettere l'ordinamento. Se omissso, coincide con il `name`.

La differenza tra `name`, `alias` e `hidden` riguarda due dimensioni indipendenti — ricerca e visualizzazione:

campo	cercabile	visualizzato
<code>name</code>	sì	sì (primario)
<code>alias</code>	sì	sì (secondario)
<code>hidden</code>	sì	no
<code>sort_name</code>	no	sì (solo in viste ordinate)

Il `sort_name`, quando presente, è soggetto a `lang` come il `name`.

Campi documentali

Un blocco `[[terminology]]` **PUÒ** dichiarare:

- `definition`: formulazione logica, formale e univoca del significato della Entity.
- `description`: spiegazione informale, testuale e narrativa della Entity.
- `example`: un esempio concreto di applicazione della Entity nel dominio.
- `note`: osservazioni aggiuntive a uso dell'analista o del sistema.

I campi documentali sono opzionali. Un blocco TRM contenente, oltre a `entity` e `lang`, il solo campo `name` è valido.

Esempio essenziale:

```
[[terminology]]
entity = "cls.machine"
lang = "it"
name = "Macchina"
definition = "Apparato industriale fisico impiegato nel processo produttivo."
```

Esempio con `sort_name` (l'entità ha come name la forma canonica "Leonardo da Vinci"; il `sort_name` fornisce la forma di ordinamento alfabetico per le liste, mentre gli `alias` sono forme alternative cercabili e visualizzabili):

```
[[terminology]]
entity = "obj.leonardo_da_vinci"
lang = "it"
name = "Leonardo da Vinci"
sort_name = "Da Vinci, Leonardo"
alias = ["Leonardo",
         "Leonardo di ser Piero da Vinci"]
hidden = ["Leonardo Da Vinci", "Lionardo",
         "leonardo da vinci"]
```

Esempio multilingua e completo (la stessa Entity riceve etichette in più lingue; il blocco italiano dichiara anche `alias` e varianti nascoste, oltre a tutti i campi documentali):

```
[[terminology]]
entity = "cls.machine"
lang = "it"
```

```

name = "Macchina"
alias = ["Apparato industriale", "Unità produttiva"]
hidden = ["macchinario", "machina"]
definition = "Apparato industriale fisico impiegato nel processo produttivo."
description = "Termine generico che identifica le macchine presenti negli impianti produttivi, indipendentemente dalla loro funzione specifica."
example = "Una pressa idraulica installata in una linea di stampaggio."
note = "Da non confondere con `cls.equipment`, che indica strumentazione ausiliaria non direttamente coinvolta nel processo."

[[terminology]]
entity = "cls.machine"
lang = "en"
name = "Machine"
alias = ["Industrial machine"]

[[terminology]]
entity = "cls.machine"
lang = "fr"
name = "Machine"

```

Esempio con etichetta priva di lingua naturale (*il codice ISO è un acronimo tecnico che non appartiene ad alcuna lingua*):

```

[[terminology]]
entity = "cls.iso_9001"
lang = ""
name = "ISO 9001"

```

Parte IV — Entity Type (ETT) e Particular (PAR)

Definizione 12. Entity Type (ETT)

Un **Entity Type (ETT)** cattura una comunanza in un insieme di Entity. Le Entity che cadono nella sua estensione sono dette **istanze** dell'ETT. L'ETT non è un elemento individuale del dominio: esprime ciò che le sue istanze hanno in comune, secondo cui esse possono essere riconosciute come dello stesso tipo.

Un ETT **NON DEVE** essere istanza di alcun ETT. L'istanziamento collega sempre una Entity definita come Particular (PAR) al suo ETT, mai un ETT a un altro ETT (Definizione 14).

Un ETT **DEVE** essere dichiarato come Entity con:

```
kind = "ETT"
```

Ogni ETT **DEVE** dichiarare un campo `archetype` scelto tra quattro valori. Ciascun archetipo declina in modo diverso la comunanza catturata:

archetype	Sigla	Comunanza catturata
"class"	CLS	La natura comune di un insieme indefinito e non ordinato di oggetti concreti o astratti del dominio.
"scheme"	SCH	L'essere categorie di un solo schema di classificazione, in numero definito, eventualmente ordinato o gerarchico.
"measurement_type"	MST	L'essere unità di misura, tra loro commensurabili, di una stessa grandezza.
"resource_type"	RST	La natura comune di un insieme di risorse digitali esterne al modello.

Definizione 13. Particular (PAR)

Un **Particular (PAR)** rappresenta un elemento individuale e terminale del dominio, istanza di un ETT. Un PAR non ha istanze proprie.

Un PAR **DEVE** essere dichiarato come Entity con:

```
kind = "PAR"
```

Ogni PAR **DEVE** dichiarare un campo `archetype` scelto tra quattro valori:

archetype	Sigla	Funzione
"object"	OBJ	Elemento individuale concreto o astratto del dominio, istanza di una ed una sola CLS.
"category"	CTG	Termine individuale appartenente ad uno schema di classificazione (SCH).
"measurement_unit"	MSU	Unità di Misura, appartenente ad un MST.
"resource"	RES	Artefatto o servizio digitale esterno referenziato dal modello, descritto da un RST.

Definizione 14. Istanziamento (instance_of)

Il campo `instance_of` esprime la relazione di istanziamento tra un PAR e il suo ETT: dichiara che il PAR è istanza dell'ETT indicato. Il campo è dichiarato sul PAR e punta all'ETT.

Un PAR è istanza di **esattamente un** ETT. Poiché un ETT non può a sua volta essere istanza di alcun ETT (Definizione 12), non esistono catene di istanziamento: ogni relazione `instance_of` collega un PAR al suo ETT in un solo passaggio. La relazione non è pertanto transitiva — non come vincolo dichiarato, ma come proprietà strutturale del modello.

KOAN **non supporta l'ereditarietà tra tipi**. Un PAR non eredita proprietà attraverso catene di tipi. L'estensione predicativa avviene tramite Characterization e Role Assignment.

I vincoli di istanziamento, per ciascun archetipo, sono:

PAR	ETT richiesto	Vincolo
OBJ	CLS	Un OBJ DEVE istanziare una e una sola CLS.
CTG	SCH	Una CTG DEVE istanziare uno e un solo SCH.
MSU	MST	Una MSU DEVE istanziare uno e un solo MST.
RES	RST	Una RES DEVE istanziare uno e un solo RST.

Esempio (*obj.press_pr500* è un oggetto istanza della classe *cls.machine*):

```
[[entity]]
kind = "ETT"
archetype = "class"
id = "cls.machine"

[[entity]]
kind = "PAR"
archetype = "object"
id = "obj.press_pr500"
instance_of = "cls.machine"
```

Parte V — Classi e Oggetti: CLS e OBJ

Definizione 15. Class (CLS)

Una **Class (CLS)** è un ETT con:

```
archetype = "class"
```

Una CLS definisce la natura strutturale di un insieme di oggetti concreti o astratti del dominio. Non rappresenta una entità individuale: rappresenta il tipo secondo cui uno o più oggetti possono essere riconosciuti come istanze della stessa classe.

Esempio (*cls.machine* è una classe di oggetti):

```
[[entity]]
kind = "ETT"
archetype = "class"
id = "cls.machine"
```

Definizione 16. Object (OBJ)

Un **Object (OBJ)** è un PAR con:

```
archetype = "object"
```

Un OBJ rappresenta un oggetto individuale — concreto o astratto — del dominio.

Esempio (*l'oggetto obj.press_pr500* è istanza della classe *cls.machine*):

```
[[entity]]
kind = "PAR"
archetype = "object"
id = "obj.press_pr500"
instance_of = "cls.machine"
```

Parte VI — Classificazioni: SCH e CTG

Definizione 17. Scheme (SCH)

Uno **Scheme (SCH)** è un ETT con:

```
archetype = "scheme"
```

Uno SCH istituisce uno schema di classificazione. Definisce il modo in cui un insieme di categorie può essere organizzato e usato come vocabolario controllato.

Uno SCH **PUÒ** dichiarare:

- `scale`: "nominal" o "ordinal". Il valore predefinito è "nominal".
- `organization`: "flat" o "hierarchical". Il valore predefinito è "flat".

Uno schema `nominal` raccoglie categorie tra loro indipendenti, senza alcun ordine intrinseco — come i colori o i reparti di un'azienda. Uno schema `ordinal` raccoglie categorie disposte in una sequenza dove la posizione conta — come livelli di criticità o gradi di un giudizio. Sul piano organizzativo, uno schema `flat` espone le sue categorie su un unico livello, mentre uno schema `hierarchical` le organizza in una struttura ad albero (tipicamente una tassonomia), dove ciascuna categoria può avere una categoria genitore e una o più categorie figlie. Le due dimensioni — scala e organizzazione — sono indipendenti, con un'unica eccezione: `ordinal` e `hierarchical` non possono coesistere, perché una sequenza lineare e una struttura ad albero esprimono ordini incompatibili.

I valori `scale = "ordinal"` e `organization = "hierarchical"` **NON DEVONO** coesistere nel medesimo SCH.

Esempio essenziale (*`sch.operational_status` è uno schema flat e nominale; non richiede alcun parametro opzionale*):

```
[[entity]]
kind = "ETT"
archetype = "scheme"
id = "sch.operational_status"
```

Esempio (*`sch.machine_taxonomy` è uno schema di classificazione nominale e gerarchico*):

```
[[entity]]
kind = "ETT"
archetype = "scheme"
id = "sch.machine_taxonomy"
organization = "hierarchical"
```

Definizione 18. Category (CTG)

Una **Category (CTG)** è un PAR con:

```
archetype = "category"
```

Esempio (*`ctg.hydraulic_press` è una categoria appartenente allo schema di classificazione `sch.machine_taxonomy`*):

```
[[entity]]
kind = "PAR"
archetype = "category"
id = "ctg.hydraulic_press"
instance_of = "sch.machine_taxonomy"
```

Categorie in schemi gerarchici

Quando lo SCH di riferimento dichiara `organization = "hierarchical"`, le relative CTG definiscono una struttura ad albero (foresta) attraverso tre chiavi: `root_node`, `parent` e `leaf_node`.

Gli indicatori booleani `root_node` e `leaf_node` compaiono esclusivamente quando assumono valore affermativo. Non è ammessa la loro esplicitazione con valore negativo, per esempio `root_node = false`.

- `root_node = true`: identifica la categoria come nodo radice dell'albero, privo di antenati. Una categoria radice **NON DEVE** dichiarare il campo `parent`.
- `parent`: ospita l'id della categoria genitore diretta. Ogni categoria che non sia un nodo radice **DEVE** dichiarare questo campo. I legami gerarchici espressi tramite `parent` **NON DEVONO** generare cicli o ricorsioni e devono appartenere rigorosamente al medesimo SCH. Una CTG **PUO'** indicare al più un solo `parent`.
- `leaf_node = true`: identifica la categoria come nodo foglia, privo di ulteriori ramificazioni. Una categoria foglia **DEVE** dichiarare un `parent`, e allo stesso tempo nessun'altra categoria del modello può indicare tale foglia come proprio `parent`.

Esempi (la categoria `ctg.forming_equipment` è il root node dello schema `sch.machine_taxonomy`; `ctg.presses` è una categoria intermedia, con genitore `ctg.forming_equipment`; `ctg.hydraulic_presses` è un leaf node, con genitore `ctg.presses`):

```
[[entity]]
kind = "PAR"
archetype = "category"
id = "ctg.forming_equipment"
instance_of = "sch.machine_taxonomy"
root_node = true

[[entity]]
kind = "PAR"
archetype = "category"
id = "ctg.presses"
instance_of = "sch.machine_taxonomy"
parent = "ctg.forming_equipment"

[[entity]]
kind = "PAR"
archetype = "category"
id = "ctg.hydraulic_presses"
instance_of = "sch.machine_taxonomy"
parent = "ctg.presses"
leaf_node = true
```

Categorie in schemi ordinali

Quando lo SCH di riferimento dichiara `scale = "ordinal"`, ogni sua CTG **DEVE** dichiarare un intero positivo univoco nel campo `ordinal_index`, esprimendo la posizione obbligatoria della CTG nella sequenza lineare.

Esempio (Schema di classificazione `sch.criticality_level`, dichiarato come ordinale e non gerarchico):

```
[[entity]]
kind = "ETT"
archetype = "scheme"
id = "sch.criticality_level"
scale = "ordinal"
organization = "flat"
```

Esempio (categorie ordinali `ctg.low_criticality` e `ctg.high_criticality` dello schema `sch.criticality_level`, ordinate secondo l'ordinal index):

```
[[entity]]
kind = "PAR"
archetype = "category"
id = "ctg.low_criticality"
instance_of = "sch.criticality_level"
ordinal_index = 1

[[entity]]
kind = "PAR"
archetype = "category"
id = "ctg.high_criticality"
instance_of = "sch.criticality_level"
ordinal_index = 2
```

Parte VII — Misure: MST e MSU

Definizione 19. Measurement Type (MST)

Un **Measurement Type (MST)** è un ETT che definisce la natura comune di un insieme di unità di misura. Un MST viene definito da:

```
archetype = "measurement_type"
```

Un MST **DEVE** dichiarare:

- `base_unit`: id della MSU (cioè di una delle sue Unità di Misura) designata come unità fondamentale.
- `value_type`: tipo numerico ammesso per i valori delle unità di misura di questo MST, scelto tra `"integer"` e `"float"`.

Un MST **PUÒ** dichiarare:

- `dimensional_symbol`: stringa di notazione dimensionale, per esempio `"M·L-1·T-2"`.

Esempio (`mst.mass` è un tipo di unità di misura, nel quale l'unità di misura fondamentale è `msu.kilogram` e il simbolo dimensionale è `M`. I valori numerici delle unità di misura di questo tipo sono espresse da numeri in virgola mobile):

```
[[entity]]
kind = "ETT"
archetype = "measurement_type"
id = "mst.mass"
base_unit = "msu.kilogram"
value_type = "float"
dimensional_symbol = "M"
```

Esempio con `value_type = "integer"` (`mst.cycle_count` rappresenta un conteggio di cicli operativi; i suoi valori sono per natura interi e non frazionari):

```
[[entity]]
kind = "ETT"
archetype = "measurement_type"
id = "mst.cycle_count"
base_unit = "msu.cycle"
value_type = "integer"
```

Definizione 20. Measurement Unit (MSU)

Una **Measurement Unit (MSU)** rappresenta una specifica unità di misura, ed è un particular (PAR) con:

```
archetype = "measurement_unit"
```

La MSU che è stata specificata come `base_unit` nel proprio MST non richiede parametri numerici. Ogni altra MSU dello stesso MST **DEVE** invece dichiarare:

- `conversion_factor`: il fattore di conversione verso la base unit; **NON DEVE** essere `0.0`.
- `conversion_offset`: l'offset di conversione verso la base unit.

La normalizzazione avviene secondo la formula:

```
base_value = (value × conversion_factor) + conversion_offset
```

Esempio (*msu.kilogram* è una unità di misura che appartiene al tipo *mst.mass*. *msu.gram* è una unità di misura che appartiene allo stesso MST, ma indica un fattore di conversione - da grammi a kilogrammi):

```
[[entity]]
kind = "PAR"
archetype = "measurement_unit"
id = "msu.kilogram"
instance_of = "mst.mass"

[[entity]]
kind = "PAR"
archetype = "measurement_unit"
id = "msu.gram"
instance_of = "mst.mass"
conversion_factor = 0.001
conversion_offset = 0.0
```

Esempio con `conversion_offset` non nullo (*mst.temperature* ha come base unit *msu.kelvin*; *msu.celsius* richiede sia un fattore di conversione unitario sia un offset di 273.15, secondo la formula `kelvin = (celsius × 1.0) + 273.15`):

```
[[entity]]
kind = "ETT"
archetype = "measurement_type"
id = "mst.temperature"
base_unit = "msu.kelvin"
value_type = "float"
dimensional_symbol = "Θ"

[[entity]]
kind = "PAR"
archetype = "measurement_unit"
id = "msu.kelvin"
```

```
instance_of = "mst.temperature"

[[entity]]
kind = "PAR"
archetype = "measurement_unit"
id = "msu.celsius"
instance_of = "mst.temperature"
conversion_factor = 1.0
conversion_offset = 273.15
```

Parte VIII — Risorse Esterne: RST e RES

Definizione 21. Resource Type (RST)

Un **Resource Type (RST)** definisce uno schema predicativo che descrive la natura di un insieme di risorse digitali esterne al modello. Un RST è un ETT con:

```
archetype = "resource_type"
```

Esempio (*rst.technical_documentation* è un resource type):

```
[[entity]]
kind = "ETT"
archetype = "resource_type"
id = "rst.technical_documentation"
```

Definizione 22. Resource (RES)

Una **Resource (RES)** è un particular (PAR) con:

```
archetype = "resource"
```

Una RES descrive un artefatto digitale o un servizio esterno (come ad esempio un web service), e **DEVE** dichiarare nativamente:

- `locator`: URI o URL della risorsa.
- `media_type`: formato IANA della risorsa (ex *mime-type*).

Una RES **PUÒ** dichiarare:

- `file_name`: nome della risorsa, quando rappresenta un file digitale. Se specificato, il nome compare anche nell'ultima parte del `locator`, ma viene dichiarato separatamente per renderlo disponibile al modello in forma esplicita.
- `lang`: codice lingua della risorsa, quando applicabile.

Esempio (viene definita una risorsa digitale *res.manual_pr500_it*, di nome *manual_it.pdf*, appartenente al tipo di risorse *rst.technical_documentation*. Il suo `locator` è https://docs.plant.it/manual_it.pdf, ed il suo `media type` è PDF. La lingua della risorsa digitale è l'italiano):

```
[[entity]]
kind = "PAR"
archetype = "resource"
id = "res.manual_pr500_it"
instance_of = "rst.technical_documentation"
file_name = "manual_it.pdf"
locator = "https://docs.plant.it/manual_it.pdf"
media_type = "application/pdf"
lang = "it"
```

Parte IX — Property Type (PTY)

Definizione 23. Property Type (PTY)

Una **Property Type (PTY)** è una Entity che definisce in astratto un tipo di proprietà: ne stabilisce la natura — se sia un valore semplice (una stringa di testo, un numero, una data, un valore booleano), un riferimento a un'altra Entity, o una misura — e i vincoli formali a cui sottostà. Una PTY non descrive un singolo fatto e non appartiene ad alcuna Entity: esiste indipendentemente da chi la userà.

Una PTY non viene "posseduta" dalle Entity. Il suo impiego nel modello segue tre livelli distinti:

- la **PTY** definisce la proprietà in astratto;
- una **Characterization (CHR)** associa la PTY a un ETT o a un RLT — e soltanto a questi — dichiarando che le Entity considerate in quel contesto tipologico o funzionale possono ricevere quella proprietà, e con quali vincoli. L'insieme delle PTY così associate a un ETT o a un RLT ne costituisce il Property Set;
- una **Assertion (ASN)** registra il valore effettivo della proprietà per una singola Entity, considerata sotto il profilo di un ETT o di un RLT. L'Entità asserita non "possiede" la PTY: ne esprime un valore specifico, in conformità a una CHR.

Nessun altro tipo di Entity, e in particolare nessun Particular in quanto tale, può avere PTY associate al di fuori di questo meccanismo.

Una PTY **DEVE** essere dichiarata come Entity con:

```
kind = "PTY"
```

Ogni PTY **DEVE** dichiarare un campo `archetype` scelto tra tre valori. I tre archetipi corrispondono alle tre forme che un fatto può assumere: un valore semplice, un riferimento a un'altra Entity, un valore numerico con unità.

<code>archetype</code>	Sigla	Forma catturata
<code>"attribute"</code>	ATR	Un valore semplice e primitivo.
<code>"relation"</code>	RLN	Un riferimento a un'altra Entity.
<code>"measurement"</code>	MSM	Un valore numerico con un'unità di misura appartenente a un MST.

La riduzione a queste tre forme è deliberata. Un riferimento a un'altra Entity è sempre una `relation`, indipendentemente dal tipo dell'Entity raggiunta: un oggetto di una classe (OBJ), una categoria di uno schema (CTG), una unità di misura (MSU), una risorsa esterna (RES) o un'Entity in un ruolo sono tutti destinazioni possibili di una `relation`, distinte dal tipo del target dichiarato dalla PTY nel campo `target_kind` (Definizione 23). Fa eccezione il caso della misura in senso stretto: quando il riferimento a un'unità è accompagnato da un valore numerico, e la grandezza misurata fa parte dell'identità del predicato, si usa l'archetipo `measurement`.

PTY **attribute** (ATR)

Una PTY con:

```
archetype = "attribute"
```

esprime un valore semplice e primitivo.

Essa **DEVE** dichiarare il campo `value_type`, scelto tra i tipi nativi TOML ammessi:

```
string, integer, float, boolean, date, time, datetime, datetime-local
```

Il `value_type` è dichiarato sulla PTY, e non sulla CHR, perché è invariante rispetto al contesto d'uso: il tipo del valore fa parte dell'identità del predicato e non cambia da un impiego all'altro.

Esempio (*pty.serial_number* è un attributo i cui valori sono espressi come stringa):

```
[[entity]]
kind = "PTY"
archetype = "attribute"
id = "pty.serial_number"
value_type = "string"
```

Altri esempi con `value_type` diversi:

```
[[entity]]
kind = "PTY"
archetype = "attribute"
id = "pty.production_year"
value_type = "integer"

[[entity]]
kind = "PTY"
archetype = "attribute"
id = "pty.is_active"
value_type = "boolean"

[[entity]]
kind = "PTY"
archetype = "attribute"
id = "pty.installation_date"
value_type = "date"
```

PTY **relation** (RLN)

Una PTY con:

```
archetype = "relation"
```

esprime un riferimento a un'altra Entity: il valore di una `relation` è costituito dall'id di un'Entity.

Una PTY `relation` primaria **DEVE** dichiarare:

- `target_kind`: la sigla del tipo di Entity a cui la relazione punta, scelta tra `"CLS"`, `"SCH"`, `"MST"`, `"RST"`, `"RLT"`.

Il tipo di destinazione è una proprietà **stabile** del predicato, non del contesto d'uso: una relazione punta sempre allo stesso tipo di bersaglio, e se il tipo cambia si tratta di un'altra relazione. Per questo `target_kind` è dichiarato qui, sulla PTY, e non nella Characterization.

Una relazione con `target_kind = "MST"` non è una misura. Essa non registra un valore numerico: registra un riferimento a una MSU appartenente al MST scelto dalla CHR. Se invece la proprietà deve registrare un valore numerico insieme a una unità, si usa l'archetipo `measurement`.

Ciascun valore di `target_kind` determina il tipo di Entity che potrà essere indicato come `target_as` nelle Characterization che applicano questa PTY:

- `target_kind = "CLS"`: la CHR dovrà indicare come `target_as` una CLS.
- `target_kind = "SCH"`: la CHR dovrà indicare come `target_as` uno SCH.
- `target_kind = "MST"`: la CHR dovrà indicare come `target_as` un MST.
- `target_kind = "RST"`: la CHR dovrà indicare come `target_as` un RST.
- `target_kind = "RLT"`: la CHR dovrà indicare come `target_as` un RLT.

Una PTY `relation` **NON DEVE** dichiarare né `target` né `target_as`, né il tipo di Entity di partenza. Il `target_kind` fissa soltanto il tipo di contesto di destinazione ammesso nelle Characterization che useranno la PTY. La Characterization (CHR) fisserà invece il `target_as` **specifico** — quale CLS, quale SCH, quale MST, quale RST o quale RLT — e le cardinalità, insieme al contesto di partenza (`subject_as`). In altre parole: il tipo generale di destinazione appartiene al predicato; il contesto specifico di destinazione appartiene all'uso della proprietà.

Relazioni inverse

Una PTY `relation` **PUÒ** dichiarare il campo opzionale:

- `inverse_of`: id di un'altra PTY con `archetype = "relation"`, di cui la PTY corrente costituisce l'inversa.

L'inversione è una proprietà del predicato astratto, non del contesto applicativo. Per questo viene dichiarata a livello di PTY e non di CHR. Una PTY senza `inverse_of` è una **relazione primaria**: le Assertion che la usano vengono registrate esplicitamente nel modello. Una PTY con `inverse_of` è una **relazione derivata**: i suoi valori non vengono inseriti manualmente come Assertion, ma sono ricavati interrogando le Assertion della relazione primaria indicata da `inverse_of`, lette nel verso opposto.

Il campo `inverse_of` è **opzionale**, e la sua presenza è essa stessa una scelta di modellazione. Dichiarare l'inversa significa voler rendere navigabile il legame anche nel verso opposto; non dichiararla significa che quel verso di lettura non interessa al dominio. Una relazione può quindi esistere nel solo verso diretto: il verso inverso esiste nel modello unicamente se una PTY derivata lo dichiara e una CHR lo assegna all'Entity di arrivo.

Il valore di `inverse_of` **DEVE** puntare a una PTY con `archetype = "relation"` priva a sua volta di `inverse_of`. Non è ammesso dichiarare una catena di inverse: la relazione di inversione è binaria e asimmetrica, e una sola delle due PTY ne porta la dichiarazione.

Una relazione derivata **NON DEVE** dichiarare `target_kind`. Il suo tipo di destinazione è ereditato dalla relazione primaria, letta al contrario: è il tipo dell'Entity di partenza della primaria, e resta quindi implicito. Solo le relazioni primarie dichiarano `target_kind`.

Esempio senza inversa (*pty.operates* è una relazione primaria verso una classe; si sceglie di non rendere navigabile il verso opposto. Chi consulta un'Entity operata non risalirà automaticamente a chi la opera):

```
[[entity]]
kind = "PTY"
archetype = "relation"
id = "pty.operates"
target_kind = "CLS"
```

Esempio con inversa (si aggiunge *pty.is_operated_by* come inversa derivata di *pty.operates*: ora il legame è navigabile anche nel verso opposto. La derivata non dichiara *target_kind*, che eredita dalla primaria):

```
[[entity]]
kind = "PTY"
archetype = "relation"
id = "pty.operates"
target_kind = "CLS"

[[entity]]
kind = "PTY"
archetype = "relation"
id = "pty.is_operated_by"
inverse_of = "pty.operates"
```

PTY **measurement** (MSM)

Una PTY con:

```
archetype = "measurement"
```

esprime una proprietà composta da un valore numerico e da un'unità di misura.

Una PTY **measurement** **DEVE** dichiarare:

- **mst**: id del MST da cui sono ammesse le MSU usate nelle Assertion.

La **measurement** dichiara quindi il proprio MST direttamente sulla PTY, nel campo **mst**. La ragione è che il MST **è la grandezza stessa** che la proprietà misura: una proprietà che misura una lunghezza misura lunghezze in ogni contesto, e la grandezza fa parte dell'identità del predicato, non del suo uso specifico.

Il tipo numerico ammesso per la misura non viene dichiarato nella PTY. È ereditato dal **value_type** del MST indicato nel campo **mst**.

Esempio (*pty.weight* è una proprietà misurabile, le cui unità di misura devono appartenere al tipo *mst.mass*):

```
[[entity]]
kind = "PTY"
archetype = "measurement"
id = "pty.weight"
mst = "mst.mass"
```

Parte X — Role Type (RLT) e Role Assignment (RAS)

Definizione 24. Role Type (RLT)

Un **Role Type (RLT)** è una Entity che definisce una funzione, uno stato operativo o un contesto funzionale che qualunque Entity può assumere o dismettere mediante Role Assignment (Definizione 25).

L'assunzione di un RLT non è limitata ai Particular. Un RLT può essere assunto da qualunque famiglia di Entity: **PAR**, **ETT**, **PTY** o anche un altro **RLT**. In questo modo KOAN può assegnare funzioni operative non solo agli oggetti individuali, ma anche trasversalmente ai tipi, ai predicati e agli stessi ruoli.

Un RLT **DEVE** essere dichiarato come Entity con:

```
kind = "RLT"
```

Un RLT non è il ruolo assunto da una specifica Entity. È un tipo di ruolo: la forma funzionale che può essere assegnata a una o più Entity tramite Role Assignment.

Esempio (*rol.operator* è un role type):

```
[[entity]]
kind = "RLT"
id = "rol.operator"
```

Definizione 25. Role Assignment (RAS)

Un **Role Assignment (RAS)** è il blocco strutturale che dichiara l'assunzione di un RLT da parte di una Entity.

Il RAS **non è una Entity**. Non ha un `id`, non riceve terminologia, non riceve Characterization e non può essere soggetto di Assertion.

Il RAS viene dichiarato mediante il blocco:

```
[[role_assignment]]
```

Ogni `[[role_assignment]]` **DEVE** dichiarare:

- `entity`: id della Entity che assume il ruolo.
- `role`: id di una Entity con `kind = "RLT"`.

Ogni `[[role_assignment]]` **PUÒ** dichiarare:

- `valid_from`: data o datetime di inizio validità.
- `valid_until`: data o datetime di fine validità.

La finestra temporale è l'unica dimensione che l'assunzione di ruolo dichiara oltre a `entity` e `role`. Entrambi i campi temporali sono opzionali: un'assunzione priva di entrambi vale senza delimitazione temporale; un'assunzione con il solo `valid_from` è in corso a partire da quella data; un'assunzione con entrambi delimita un intervallo chiuso.

Esempio (*obj.ferrari_luca* assume il ruolo di *rol.operator* a partire dal 1 gennaio 2024; l'assunzione è in corso):

```
[[role_assignment]]
entity = "obj.ferrari_luca"
role = "rol.operator"
valid_from = 2024-01-01
```

Esempio con periodo chiuso (*obj.bianchi_anna* ha ricoperto il ruolo di *rol.operator* dal 1 marzo 2022 al 31 dicembre 2023):

```
[[role_assignment]]
entity = "obj.bianchi_anna"
role = "rol.operator"
valid_from = 2022-03-01
valid_until = 2023-12-31
```

Esempio con assunzione futura (*obj.rossi_marco* assumerà il ruolo di *rol.shift_supervisor* dal 1 gennaio 2027):

```
[[role_assignment]]
entity = "obj.rossi_marco"
role = "rol.shift_supervisor"
valid_from = 2027-01-01
```

Tutte le assunzioni di ruolo **DEVONO** essere espresse mediante `[[role_assignment]]`.

Parte XI — Characterization (CHR)

Definizione 26. Characterization (CHR) e Property Set (PST)

Una **Characterization (CHR)** dichiara che una PTY è applicabile a una determinata categoria di soggetti — un ETT o un RLT — e con quali vincoli.

Il **Property Set (PST)** di un ETT o di un RLT è l'insieme delle CHR che hanno quell'ETT o RLT come `subject_as`. Esso definisce quali PTY possono essere usate per descrivere le Entity considerate in quel contesto tipologico, e con quali vincoli.

La CHR viene dichiarata mediante il blocco:

```
[[characterization]]
```

La CHR **non è una Entity**. Non ha un `id`, non riceve terminologia, non riceve ruoli e non può essere soggetto di Assertion.

Ogni CHR **DEVE** dichiarare:

- `subject_as`: id dell'ETT o del RLT a cui la PTY viene associata.
- `property`: id della PTY applicata.

Ogni CHR **PUÒ** dichiarare:

- `min_count`: cardinalità minima delle Assertion conformi alla CHR. Il valore predefinito è `0`.
- `max_count`: cardinalità massima delle Assertion conformi alla CHR. Se omissa, la cardinalità è illimitata.
- `display_order`: intero che governa l'ordinamento visivo della proprietà nelle interfacce utente visuali.

Unicità della CHR

Una CHR relazionale è identificata dalla combinazione dei campi `subject_as`, `property` e `target_as`. Tale combinazione **DEVE** essere unica nel modello.

Una stessa PTY `relation` **PUÒ** quindi comparire in più CHR aventi lo stesso `subject_as`, purché ciascuna dichiari un `target_as` diverso. Questo accade tipicamente quando una stessa proprietà classificatoria viene applicata mediante schemi di classificazione differenti, oppure quando una stessa relazione collega lo stesso `subject_as` a contesti di destinazione differenti.

Per le CHR che applicano una PTY con `archetype = "attribute"` o `archetype = "measurement"`, il campo `target_as` non è dichiarato. In tali casi l'identità della CHR si riduce alla coppia `subject_as` + `property`, e ogni ETT o RLT **PUÒ** dichiarare una sola CHR per ciascuna PTY `attribute` o `measurement`.

Esempio (la classe `cls.opera_pittorica` può ricevere classificazioni di soggetto iconografico secondo schemi differenti, ciascuno con la propria CHR):

```
[[characterization]]
subject_as = "cls.opera_pittorica"
property = "pty.soggetto_iconografico"
target_as = "sch.iconografia_cristiana"
```

```
[[characterization]]
subject_as = "cls.opera_pittorica"
property = "pty.soggetto_iconografico"
target_as = "sch.mitologia_classica"
```

Characterization di una PTY Attribute (ATR)

Se una CHR applica una PTY con `archetype = "attribute"`, non richiede informazioni aggiuntive.

Il tipo del valore ammesso è già dichiarato nella PTY mediante il campo `value_type`.

Esempio (*cls.machine* ha la proprietà *pty.serial_number*, con uno ed un solo valore ammesso; il campo `display_order` ne fissa la posizione di visualizzazione nelle interfacce):

```
[[characterization]]
subject_as = "cls.machine"
property = "pty.serial_number"
min_count = 1
max_count = 1
display_order = 10
```

Characterization di una PTY Relation (RLN)

Se una CHR applica una PTY con `archetype = "relation"`, la CHR **DEVE** dichiarare:

- `target_as`: id dell'ETT o del RLT **specifico** sotto cui la futura Entity di destinazione parteciperà alle Assertion conformi a questa CHR. Il suo tipo **DEVE** coincidere con il `target_kind` dichiarato dalla PTY (Definizione 23): una CLS se `target_kind = "CLS"`, uno SCH se `"SCH"`, un MST se `"MST"`, un RST se `"RST"`, un RLT se `"RLT"`. Un parser KOAN **DEVE** verificare questa coerenza; una discordanza rende il modello non valido.

La CHR non ridichiara `target_kind`: il tipo di destinazione è già fissato dalla PTY. La CHR sceglie soltanto *quale contesto di destinazione* di quel tipo — quale classe, quale schema, quale measurement type, quale tipo di risorsa, quale ruolo — e con quali cardinalità. Non identifica ancora una Entity concreta di destinazione: quella comparirà nel campo `target` della ASN.

Una PTY relazionale può essere **primaria** (priva del campo `inverse_of`) o **derivata** (con `inverse_of` dichiarato). La CHR si comporta in modo diverso nei due casi.

CHR di una relazione primaria

Se la PTY indicata in `property` è una relazione primaria, le Assertion conformi alla CHR vengono inserite esplicitamente nel modello.

Una CHR di relazione primaria **PUÒ** dichiarare `min_count` e `max_count` con il significato consueto: cardinalità minima e massima delle Assertion conformi alla CHR.

Esempio verso una CLS (*pty.operates* è una relazione primaria con `target_kind = "CLS"` sulla PTY; qui il Role Type *rol.operator* la applica scegliendo come `target_as` la classe *cls.machine*. Le Assertion verranno inserite manualmente):

```
[[characterization]]
subject_as = "rol.operator"
property = "pty.operates"
target_as = "cls.machine"
```

Esempio verso un RLT (*pty.reports_to* ha `target_kind = "RLT"` sulla PTY; il Role Type *rol.system_analyst* la applica scegliendo come `target_as` il ruolo *rol.supervisor*):

```
[[characterization]]
subject_as = "rol.system_analyst"
property = "pty.reports_to"
target_as = "rol.supervisor"
max_count = 1
```

Esempio verso uno SCH (*pty.machine_type* ha `target_kind = "SCH"` sulla PTY; la classe *cls.machine* la applica scegliendo come `target_as` lo schema *sch.machine_taxonomy*. Questo è il caso che nelle versioni precedenti era l'archetipo classification):

```
[[characterization]]
subject_as = "cls.machine"
property = "pty.machine_type"
target_as = "sch.machine_taxonomy"
min_count = 1
max_count = 1
```

Esempio verso un MST (*pty.preferred_display_unit* ha `target_kind = "MST"` sulla PTY; la classe *cls.distance_indicator* la applica scegliendo come `target_as` il measurement type *koan_core@mst.length*. Le Assertion useranno una MSU appartenente a quel MST, senza registrare un valore numerico):

```
[[characterization]]
subject_as = "cls.distance_indicator"
property = "pty.preferred_display_unit"
target_as = "koan_core@mst.length"
max_count = 1
```

Esempio verso un RST (*pty.has_manual* ha `target_kind = "RST"` sulla PTY; la classe *cls.machine* la applica scegliendo come `target_as` il tipo di risorsa *rst.technical_documentation*. Questo è il caso che nelle versioni precedenti era l'archetipo resource):

```
[[characterization]]
subject_as = "cls.machine"
property = "pty.has_manual"
target_as = "rst.technical_documentation"
min_count = 0
max_count = 1
```

CHR di una relazione derivata

Se la PTY indicata in `property` è una relazione derivata, la CHR definisce un percorso di navigazione: i valori esposti non sono ASN proprie, ma vengono ricavati interrogando le ASN della relazione primaria indicata dal campo `inverse_of` della PTY.

Una CHR di relazione derivata **DEVE** essere coerente con la CHR della relazione primaria: il `target_as` della CHR derivata **DEVE** coincidere con il `subject_as` della CHR primaria, e il `subject_as` della CHR derivata **DEVE** coincidere con il `target_as` della CHR primaria. La CHR derivata dichiara `target_as`, ma non `target_kind`: la PTY derivata non lo possiede (Definizione 23), e il tipo della destinazione è quello del `subject_as` della primaria.

Una CHR di relazione derivata:

- **PUÒ** dichiarare `max_count`. Esprime il numero massimo di risultati ammessi nella navigazione inversa: vincola quante ASN della relazione primaria possono puntare, come target, a una stessa Entity nel contesto `subject_as` della CHR derivata.

- **NON DEVE** dichiarare `min_count`. Una cardinalità minima imporrebbe l'esistenza di un numero minimo di ASN della relazione primaria che puntino, come target, alla Entity considerata. Tale vincolo non riguarda però le ASN governate dalla CHR derivata — che non esistono — bensì le ASN della relazione primaria. Esprimerlo sulla CHR derivata costituirebbe un'attribuzione impropria del vincolo: il `min_count` apparterrebbe semanticamente a un soggetto diverso da quello dichiarato in `subject_as`.

Una CHR di relazione derivata **PUÒ** inoltre dichiarare:

- `auto_materialization`: booleano che indica se la relazione deve essere estratta automaticamente quando si consulta una Entity nel contesto indicato da `subject_as`.

Se `auto_materialization` è omissso, il valore predefinito è `false`.

Il campo `auto_materialization` non modifica il significato della relazione. Governa soltanto il comportamento di estrazione in fase di query.

Il campo `auto_materialization` **NON DEVE** essere dichiarato nelle CHR di relazioni primarie: in tali CHR le ASN sono già registrate nel modello e non richiedono alcuna navigazione di materializzazione.

Comportamento di `auto_materialization`

Il campo `auto_materialization` governa il comportamento di estrazione della relazione derivata in fase di consultazione. Per relazioni con pochi valori conviene estrarli automaticamente; per relazioni potenzialmente molto popolate è preferibile lasciarne il caricamento alla richiesta esplicita.

Una bicicletta ha poche parti: la sua relazione derivata `has_part` può essere materializzata automaticamente. Una categoria molto generale può raccogliere migliaia di elementi collegati: in tal caso la materializzazione automatica è sconsigliata.

Esempio di CHR di relazione derivata (*la classe `cls.machine` ha una proprietà `pty.is_operated_by`, i cui valori sono entità nel ruolo di `rol.operator`. La proprietà `pty.is_operated_by` è dichiarata come inversa di `pty.operates`. La materializzazione di questa proprietà non è automatica, ed avverrà solo su richiesta esplicita*):

```
[[entity]]
kind = "PTY"
archetype = "relation"
id = "pty.is_operated_by"
inverse_of = "pty.operates"

[[characterization]]
subject_as = "cls.machine"
property = "pty.is_operated_by"
target_as = "rol.operator"
max_count = 1
auto_materialization = false
```

Characterization di una PTY Measurement (MSM)

Se una CHR specifica una PTY con `archetype = "measurement"`, non deve dichiarare `target_as`.

Il MST di riferimento è già dichiarato nella PTY `measurement`, mediante il suo campo `mst`.

La CHR stabilisce soltanto in quale contesto la misura può essere specificata, e con quali cardinalità.

Il campo `value_type` **NON DEVE** essere dichiarato. Il tipo numerico ammesso viene ereditato dal MST indicato nella PTY.

Esempio (*la classe `cls.machine` ha una proprietà `pty.machine_weight`, configurata come *measurement property* sul tipo `mst.mass`*):

```
[[entity]]
kind = "PTY"
archetype = "measurement"
id = "pty.machine_weight"
mst = "mst.mass"

[[characterization]]
subject_as = "cls.machine"
property = "pty.machine_weight"
min_count = 0
max_count = 1
```

Parte XII — Assertion (ASN)

Definizione 27. Assertion (ASN)

Una **Assertion (ASN)** registra un'asserzione atomica concernente una Entity.

L'ASN viene dichiarata mediante il blocco:

```
[[assertion]]
```

L'ASN **non è una Entity**. Non ha un `id`, non riceve terminologia, non riceve ruoli e non può essere soggetto di altre Assertion.

Ogni ASN **DEVE** corrispondere a una CHR predefinita.

Ogni ASN **DEVE** dichiarare:

- `subject`: id della Entity di cui si predica l'asserzione.
- `subject_as`: id dell'ETT/RLT con il quale subject compare nell'asserzione.
- `property`: id della PTY oggetto dell'asserzione. Deve esistere una CHR che dichiari questa PTY per l'ETT/RLT indicato in `subject_as`.

Ogni ASN **DEVE** dichiarare i campi di valore richiesti dall'archetipo della PTY applicata. I campi disponibili sono:

- `value`: valore semplice e primitivo.
- `target`: id della Entity di destinazione dell'asserzione.
- `target_as`: id dell'ETT/RLT secondo cui il `target` partecipa all'asserzione.

Ogni ASN **PUÒ** inoltre dichiarare:

- `lang`: qualificazione linguistica dell'asserzione nel suo complesso. Il valore **DEVE** essere un codice lingua BCP 47, per esempio `"it"`, `"en"`, `"fr"`, oppure la stringa vuota `""`. Il campo è ammesso con qualunque archetipo di PTY. Se omesso oppure dichiarato come `lang = ""`, l'ASN è linguisticamente neutra: è valida indipendentemente dalla lingua, oppure la lingua non è significativa per l'asserzione corrente. Le due situazioni non vengono distinte dal core.

Il campo `lang` qualifica l'ASN nel suo complesso. Non modifica la forma del valore richiesta dall'archetipo della PTY, non sostituisce `value`, `target` o `target_as` e non altera i vincoli di compatibilità definiti nelle sezioni successive.

Il campo `target`, quando presente, contiene sempre l'id di una Entity. Il tipo di Entity atteso dipende dall'archetipo della PTY e, quando necessario, dalla CHR corrispondente.

Ogni ASN che dichiara un `target` **DEVE** dichiarare anche un corrispondente `target_as`. Se `target_as` punta a un ETT, il `target` **DEVE** essere un PAR il cui `instance_of` coincida con quell'ETT. Se `target_as` punta a un RLT, deve esistere un Role Assignment compatibile tra il `target` e quel RLT.

I vincoli di cardinalità dichiarati nella CHR si applicano a tutte le ASN conformi a quella CHR. Per le PTY `relation`, la CHR corrispondente è individuata dalla terna `subject_as` + `property` + `target_as`; per le PTY `attribute` e `measurement`, dalla coppia `subject_as` + `property`.

Vincolo del soggetto

Se `subject_as` punta a un ETT, il `subject` **DEVE** essere un PAR il cui campo `instance_of` coincida con quell'ETT.

Se `subject_as` punta a un RLT, deve esistere un Role Assignment compatibile tra il `subject` e quel RLT.

La verifica temporale della compatibilità tra ASN e Role Assignment è applicata solo quando sia l'ASN sia il Role Assignment dichiarano informazioni temporali confrontabili.

Validazione del valore

La forma del valore di una ASN dipende dall'archetipo della PTY dichiarata nel campo `property` e, quando necessario, dalla CHR corrispondente.

ASN con PTY `attribute`

- **DEVE** dichiarare: `value`
- **NON DEVE** dichiarare: `target`, `target_as`
- Vincolo: `value` **DEVE** essere compatibile con il `value_type` dichiarato dalla PTY.

ASN con PTY `relation`

Una ASN che usa una PTY `relation` è ammessa solo se la PTY indicata in `property` è una **relazione primaria**, cioè priva del campo `inverse_of`.

- **DEVE** dichiarare: `target`, `target_as`
- **NON DEVE** dichiarare: `value`
- `target_as`: **DEVE** coincidere con il `target_as` della CHR utilizzata (la CLS, lo SCH, il MST, il RST o il RLT ammesso).
- Vincoli, secondo il `target_kind` dichiarato dalla PTY `relation`:
 - `target_kind = "CLS"`: il `target` della ASN **DEVE** essere un OBJ il cui `instance_of` coincida con quella CLS;
 - `target_kind = "SCH"`: il `target` della ASN **DEVE** essere una CTG il cui `instance_of` coincida con quello SCH;
 - `target_kind = "MST"`: il `target` della ASN **DEVE** essere una MSU il cui `instance_of` coincida con quel MST;
 - `target_kind = "RST"`: il `target` della ASN **DEVE** essere una RES il cui `instance_of` coincida con quel RST;
 - `target_kind = "RLT"`: il `target` della ASN **DEVE** essere una Entity per cui esista un Role Assignment compatibile verso quel RLT.

Una PTY relazionale **derivata** (con `inverse_of` dichiarato) **NON AMMETTE** ASN proprie. I suoi valori sono ricavati interrogando le ASN della relazione primaria indicata da `inverse_of`.

ASN con PTY `measurement`

- **DEVE** dichiarare: `value`, `target`, `target_as`
- Campi:

- `value`: valore numerico della misura;
- `target`: id della MSU usata come unità di misura;
- `target_as`: id del MST a cui la MSU appartiene.
- Vincoli:
 - il `target` **DEVE** puntare a una MSU il cui `instance_of` coincida con il MST dichiarato nel campo `mst` della PTY `measurement`;
 - `value` **DEVE** essere compatibile con il `value_type` del MST dichiarato nel campo `mst` della PTY.

Qualificazione linguistica (`lang`)

Il campo opzionale `lang` qualifica linguisticamente l'asserzione nel suo complesso. Può essere usato con ASN basate su qualunque archetipo di PTY: `attribute`, `relation` o `measurement`.

Quando `lang` contiene un codice BCP 47 non vuoto, l'ASN è dichiarata nel relativo contesto linguistico. Esempi validi sono:

```
lang = "it"
```

```
lang = "en-US"
```

Quando il campo è omissso, oppure quando è dichiarato come:

```
lang = ""
```

l'ASN è linguisticamente neutra. Ciò significa che l'asserzione è valida indipendentemente dalla lingua, oppure che la dimensione linguistica non ha significato per l'asserzione corrente. Il core non distingue formalmente queste due situazioni.

`lang` non è limitato alle PTY `attribute` con `value_type = "string"`: può qualificare anche relazioni e misure quando la validità, la pertinenza o l'interpretazione dell'asserzione dipendono da un contesto linguistico. La presenza di `lang` non modifica comunque i campi di valore richiesti dall'archetipo della PTY.

Esempio con PTY `attribute` (*un'osservazione di laboratorio in italiano sul frammento*):

```
[[assertion]]
subject = "obj.fragment_k47"
subject_as = "cls.space_fragment"
property = "pty.surface_observation"
value = "Superficie metallica con tracce di fusione"
lang = "it"
```

Esempio con PTY `relation` (*una relazione dichiarata nel contesto linguistico italiano; la forma relazionale resta invariata*):

```
[[assertion]]
subject = "obj.module_hydroponics"
subject_as = "cls.ship_module"
property = "pty.has_technical_manual"
target = "res.hydroponics_manual_it"
target_as = "rst.technical_manual"
lang = "it"
```

Esempio linguisticamente neutro (*la massa non richiede alcuna qualificazione linguistica*):

```
[[assertion]]
subject = "obj.fragment_k47"
subject_as = "cls.space_fragment"
property = "pty.mass"
value = 18.6
target = "koan_core@msu.gram"
target_as = "koan_core@mst.mass"
lang = ""
```

Polarity, Confidence e Asserter

I tre campi `polarity`, `confidence` e `asserter` qualificano l'asserzione: ne specificano il segno, il grado di certezza e l'origine.

Polarity. Indica se l'asserzione afferma o nega il fatto rilevato. Dire che una macchina è operativa non è la stessa cosa che dire che non lo è.

Confidence. Esprime il grado di fiducia attribuito all'asserzione. Un fatto può essere certo, probabile, importato da una fonte incompleta o prodotto da un'analisi automatica.

Asserter. Identifica l'Entity che ha registrato o prodotto l'asserzione, rendendo esplicita la responsabilità epistemica del fatto. Per questo può essere utile censire nella propria ontologia Entity che operano come asserter — persone, sistemi software, procedure, organizzazioni o agenti AI — ed eventualmente assegnare loro un RLT dedicato, per esempio `rol.asserter`.

Ogni ASN **PUÒ** includere:

- `polarity`: "positive" o "negative". Se omissso, il valore predefinito è "positive".
- `confidence`: intero da 0 a 100, espresso come percentuale. Se omissso, il valore predefinito è 100.
- `asserter`: id della Entity che ha registrato o prodotto l'asserzione. Se omissso, l'origine dell'asserzione non è specificata; in particolare non è specificato chi abbia selezionato la coppia `polarity` / `confidence`.

Esempi di Assertion

Gli esempi seguenti mostrano la forma completa delle ASN per i tre archetipi di PTY, inclusi i cinque tipi di destinazione di una `relation` e l'uso opzionale della qualificazione linguistica `lang`. Essi presuppongono che le Entity, le PTY, le CHR e gli eventuali Role Assignment citati siano già stati dichiarati nel modello.

Esempio con PTY `attribute`

Una PTY `attribute` usa `value` e non usa `target` e `target_as`.

```
[[assertion]]
subject = "obj.module_hydroponics"
subject_as = "cls.ship_module"
property = "pty.module_code"
value = "HYD-03"
asserter = "obj.aurora_ai"
```

Esempio con `lang` su una PTY `attribute`

Il campo `lang` qualifica l'asserzione nel suo complesso. In questo esempio l'osservazione di laboratorio è valida nel contesto linguistico italiano.

```
[[assertion]]
subject = "obj.fragment_k47"
subject_as = "cls.space_fragment"
property = "pty.surface_observation"
value = "Superficie metallica con tracce di fusione"
lang = "it"
asserter = "obj.aurora_ai"
```

Esempio con PTY **relation** verso uno SCH (ex classification)

Una **relation** con target di tipo SCH usa **target** per indicare una CTG appartenente allo SCH ammesso dalla CHR; **target_as** riporta lo SCH. È il caso che nelle versioni precedenti era l'archetipo **classification**.

```
[[assertion]]
subject = "obj.module_hydroponics"
subject_as = "cls.ship_module"
property = "pty.operational_status"
target = "ctg.operational"
target_as = "sch.operational_status"
confidence = 100
asserter = "obj.aurora_ai"
```

Esempio con PTY **measurement**

Una PTY **measurement** usa sia **value** sia **target** che **target_as**: **value** contiene il valore numerico, mentre **target** indica la MSU usata come unità di misura, e **target_as** riporta il MST a cui l'MSU appartiene.

```
[[assertion]]
subject = "obj.module_hydroponics"
subject_as = "cls.ship_module"
property = "pty.module_mass"
value = 18500.0
target = "msu.kilogram"
target_as = "mst.mass"
confidence = 98
asserter = "obj.sensor_array_07"
```

Esempio con PTY **relation** verso un MST

Una **relation** con target di tipo MST usa **target** per indicare una MSU appartenente al MST ammesso dalla CHR; **target_as** riporta lo MST. Non registra un valore numerico e non sostituisce una PTY **measurement**.

```
[[assertion]]
subject = "obj.cockpit_rangefinder"
subject_as = "cls.distance_indicator"
property = "pty.preferred_display_unit"
target = "koan_core@msu.metre"
target_as = "koan_core@mst.length"
asserter = "obj.navigation_system"
```

Esempio con PTY **relation** verso un RST (ex resource)

Una **relation** con target di tipo RST usa **target** per indicare una RES il cui **instance_of** coincide con il RST ammesso dalla CHR; **target_as** riporta il RST. È il caso che nelle versioni precedenti era l'archetipo **resource**.

```
[[assertion]]
subject = "obj.module_hydroponics"
subject_as = "cls.ship_module"
property = "pty.has_technical_manual"
target = "res.hydroponics_manual_it"
target_as = "rst.technical_manual"
asserter = "obj.archive_system"
```

Esempio con PTY **relation** verso una CLS

Una PTY **relation** primaria usa **target** per indicare una Entity compatibile con il **target_as** dichiarato nella CHR. In questo esempio il **target_as** della CHR è una CLS; il **target** della ASN è un OBJ di quella classe e **target_as** riporta l'id della CLS.

```
[[assertion]]
subject = "obj.ishikawa_mina"
subject_as = "rol.system_analyst"
property = "pty.inspects_module"
target = "obj.module_hydroponics"
target_as = "cls.ship_module"
confidence = 95
asserter = "obj.mission_log_system"
```

Esempio con PTY **relation** verso un RLT

Quando il **target_as** della CHR è un RLT, la Entity indicata nel **target** della ASN deve avere un Role Assignment compatibile con quel RLT. L'RLT viene riportato nel campo **target_as**:

```
[[assertion]]
subject = "obj.ishikawa_mina"
subject_as = "rol.system_analyst"
property = "pty.reports_to"
target = "obj.vera_tanaka"
target_as = "rol.supervisor"
confidence = 100
asserter = "obj.command_registry"
```

In questo caso **obj.vera_tanaka** deve avere un Role Assignment compatibile con il RLT previsto dalla CHR, per esempio **rol.mission_commander**.

Esempio con **polarity = "negative"**

La **polarity** negativa permette di registrare esplicitamente che una certa asserzione viene negata dal modello.

```
[[assertion]]
subject = "obj.module_hydroponics"
subject_as = "cls.ship_module"
property = "pty.operational_status"
target = "ctg.decommissioned"
target_as = "sch.operational_status"
polarity = "negative"
confidence = 100
asserter = "obj.aurora_ai"
```

Questa ASN non afferma che il modulo è dismesso. Afferma, al contrario, che il modello assume come falso il fatto che il modulo appartenga alla categoria `ctg.decommissioned`.

Parte XIII — Risoluzione delle Relazioni Derivate

Questa parte descrive l'algoritmo di risoluzione delle relazioni derivate, già definite nella Parte IX (PTY) e nella Parte XI (CHR).

Per risolvere una relazione derivata su una Entity `x`, il motore di query:

1. legge la CHR corrente `c`;
2. verifica che `c.property` punti a una PTY `p` con `archetype = "relation"`;
3. verifica che `p` dichiari il campo `inverse_of`;
4. legge `p.inverse_of` e ottiene la PTY relazionale primaria `p_primaria`;
5. cerca le ASN tali che `property = p_primaria`, `target = x` e `subject_as = c.target_as`;
6. restituisce i `subject` di tali ASN, trattandoli nel contesto tipologico o funzionale indicato da `c.target_as`.

Il filtro su `subject_as` è obbligatorio. Senza questo filtro, la materializzazione potrebbe raccogliere soggetti appartenenti a contesti diversi solo perché usano la stessa relazione primaria.

Il campo `auto_materialization` governa soltanto il comportamento di estrazione automatica durante la consultazione. Non modifica il significato della relazione e non influisce sulla validità delle ASN della relazione primaria.

Navigazione attraverso più CHR derivate

La stessa PTY relazionale derivata **PUÒ** comparire in più CHR aventi lo stesso `subject_as` ma `target_as` differenti. Ciò accade quando una stessa relazione collega l'Entity considerata a contesti tipologici differenti — per esempio una bicicletta che ha come parti sia ruote sia telai, ciascuno rappresentato da una CLS distinta.

In questi casi, una richiesta di navigazione che non specifichi il `target_as` applica l'algoritmo a ciascuna CHR derivata avente la coppia (`subject_as`, `property`) corrispondente, e restituisce l'unione dei risultati. Ogni elemento del risultato resta associato alla CHR che lo ha prodotto, e quindi al contesto tipologico indicato dal `target_as` di quella CHR.

Una richiesta di navigazione che specifichi esplicitamente il `target_as` seleziona invece una sola CHR derivata, e restituisce un risultato omogeneo rispetto a quel contesto tipologico.
