

6. Knowledge Graph e Ontologie

Dispensa ICon

versione: 17/11/2024, 19:27

Knowledge Graph · Classi e Proprietà · Ontologie e Condivisione della Conoscenza

1 Knowledge Graph

1.1 Triple

Dato un *linguaggio* di rappresentazione logico e un *mondo* sul quale ragionare, i progettisti devono scegliere *individui* e *relazioni*, ossia come decomporre il dominio d'interesse per formalizzarne un suo modello. Il livello di dettaglio dipende dal compito da svolgere.

Ad esempio, la proprietà di essere di colore rosso, **red**, ascrivibile a diversi oggetti del mondo reale, può essere rappresentata in base allo *spettro di frequenze* della luce assorbite o riflesse. La proprietà potrebbe indicare un certo insieme o intervallo di frequenze con una mappatura su termini come, ad esempio **pink**, **scarlet**, **ruby** e **crimson**. In alternativa, lo spettro può anche essere diviso in regioni che, pur non corrispondendo a termini precisi del linguaggio comune, risultano più utili a distinguere diverse categorie di individui.

Le *relazioni* andranno definite seguendo alcune *linee guida* ingegneristiche:

Esempio — **red** può essere una categoria di classificazione appropriata per gli individui rappresentata come *relazione unaria*:

- “*l’oggetto **a** è rosso*” si traduce con **red(a)**;
- una query possibile sarebbe: “*Cosa si conosce di colore rosso?*” traducibile in: **ask red(X)**
 - i valori di **X** restituiti rappresentano individui di colore rosso;
- con questa rappresentazione, però, è difficile chiedere “*Di che colore è l’oggetto **a**?*” Nel linguaggio delle clausole definite, non si può chiedere **ask X(a)**. I nomi dei predicati non possono essere variabili. Ciò diventa possibile in logiche di ordine superiore per qualsiasi proprietà di **a**.

Si può pensare a una forma di **reificazione**, ossia alla riduzione a oggetto di un concetto astratto, ad esempio la relazione fra un individuo e il suo colore, quindi il colore stesso può essere modellato come *individuo*:

- usando la costante **red** per denotare il colore rosso che diventa un individuo referenziabile come gli altri (ad esempio un plico);
- definendo un predicato **color(Ind, Val)**, dove **Val** è il colore dell’individuo **Ind**, si ha una relazione binaria tra individui fisici e i loro colori: ad esempio “*l’oggetto **a** è rosso*” si può scrivere **color(a, red)**.
- si potrà allora chiedere: “*Cosa è di colore rosso?*” con la query **ask color(X, red)**
e anche “*Di che colore è l’oggetto **a**?*” con **ask color(a, C)**.

Esempio — Ulteriori estensioni

color come predicato binario non consente domande come “*Quale proprietà di questo oggetto ha valore red?*”, con risposta *color*.

Un'ulteriore trasformazione può prevedere la modellazione della proprietà *color* come *individuo* e con la nuova relazione *prop*: ad esempio, per rappresentare l'enunciato “*l'individuo a ha per la proprietà color il valore red*” si può scrivere *prop(a, color, red)*.

Sono anche possibili tutte le query viste prima, riscrivendo tutte le relazioni in termini di *prop*.

Il predicato ternario *prop* avrà la forma:

$$prop(Ind, Prop, Val)$$

con l'individuo *Ind* ha per la proprietà *Prop* il valore *Val*.

In alternativa si può considerare la struttura di **tripla** $\langle s, p, o \rangle$ con:

- *s* **soggetto**;
- *p* **predicato verbale**;
- *o* **oggetto**.

che può essere scritta anche come *enunciato*:

$$\text{soggetto} \quad \text{verbo} \quad \text{oggetto}.$$

equivalente all'atomo *prop(soggetto, verbo, oggetto)* che, in notazione *funzionale*, si può scrivere anche *verbo(soggetto, oggetto)*.

In una tripla, il *verbo* è una **proprietà** *p* con un **dominio**, l'insieme di individui che possono essere *soggetti* in triple con verbo *p* e un **codominio**, l'insieme di valori che possono essere *oggetti* in triple con verbo *p*.

Un **attributo** è una coppia *proprietà-valore*, ad esempio un certo plico ha *colore rosso*.

Esempio — Per trasformare in forma di tripla l'atomo *parcel(a)*:

1. *parcel* può essere rappresentato come *concetto* (reificato come valore):

$$prop(a, type, parcel).$$

- l'individuo *a* è nella classe *parcel*;
- *type* è una proprietà speciale che collega individui a classi, spesso indicata con *is_a* (ossia relazione di appartenenza \in della matematica);
- la costante *parcel* denota una *classe* di individui: l'insieme di tutte le cose, reali o potenziali, che sono plichi;

2. *parcel* può essere rappresentato come *proprietà booleana*, con valore *true* per soggetti che sono plichi:

$$prop(a, parcel, true).$$

- una *proprietà booleana* ha codominio $\{true, false\}$, dove *true* e *false* sono costanti.

Le relazioni con specifiche proprietà possono essere rappresentate con parole, numeri, e.g. #prenotazione, #volo, #ordine.

Esempio — L'azione del verbo “dare” coinvolge tre attori, detti *agente*, *ricevente* e *paziente*. Data la frase “Alex ha dato un libro a Chris”, una sua rappresentazione relazionale possibile è:

- *gave(alex, chris, book)*

Introducendo una costante come *ga3545* per un'azione specifica (*giving_act*), si può scrivere in forma di triple:

- *prop(ga3545, type, giving_act)*.
- *prop(ga3545, agent, alex)*.
- *prop(ga3545, patient, b342)*.
- *prop(ga3545, recipient, chris)*.
- *prop(b342, type, book)*.

dove agente/ricevente/paziente sono *relazioni tematiche* ossia **ruoli semantici**.

La reificazione consente di aggiungere altre proprietà, come ad esempio la data dell'evento, il prezzo del libro, ...

1.2 Individui e Identificatori

Gli individui vanno denotati usando identificatori univoci. Ad esempio, nel contesto universitario si usa la matricola per riferirsi univocamente a uno studente, essendo il solo nominativo meno adeguato a evitare casi di omonimia (o problemi insidiosi in caso uno studente cambi nome).

Nel contesto del web, si può usare uno **Uniform Resource Identifier (URI)**, o anche un **Internationalized Resource Identifier (IRI)**, come nome univoco per identificare qualsiasi **risorsa**, ossia qualunque cosa debba essere denominata. La dorma tipica è simile a quella dello *uniform resource locator* (URL) con prefisso <http://> o <https://>. Nel caso dell'URI, però, si denota un'entità, non un sito web: s'intende l'individuo denotato da esso. Il suo significato è dato dal suo uso (coerente), non da una definizione formale.

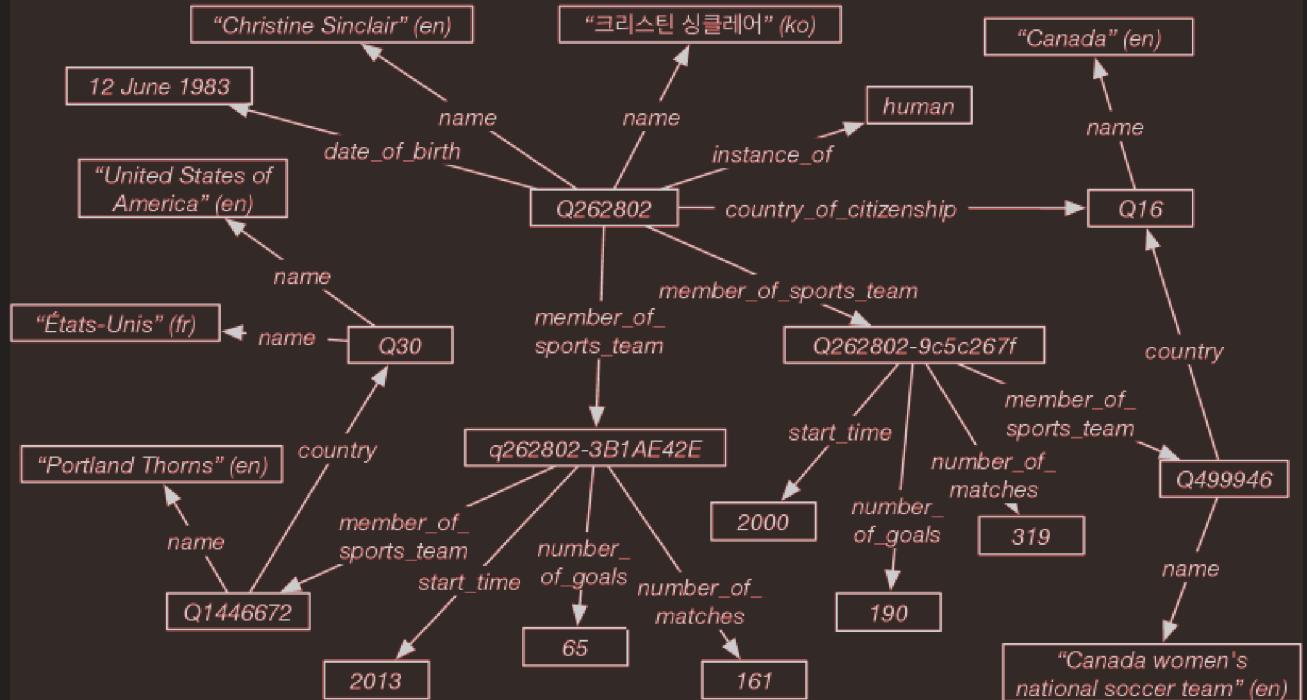
Esempio — Si consideri Wikidata, un knowledge graph aperto e collaborativo con miliardi di triple che descrivono 130M entità (statistiche del 2024).

- *Christine Sinclair* è una famosa calciatrice canadese, il cui URI in Wikidata è <http://www.wikidata.org/entity/Q262802> (abbreviabile con [Q262802](#))
- la proprietà *paese di cittadinanza*, contraddistinta dall'URI <http://www.wikidata.org/prop/direct/P27>, ha come soggetto una persona e come oggetto il paese di cui ha la cittadinanza; ad esempio la frase “*Christine Sinclair è cittadina canadese*” si traduce nella tripla:
 - <http://www.wikidata.org/entity/Q262802>
 - <http://www.wikidata.org/prop/direct/P27>
 - <http://www.wikidata.org/entity/Q16>.
- analogamente si può usare la proprietà definita in [schema.org](#) per dare un *nome* al soggetto è <http://schema.org/name>: il valore nella tripla (oggetto) è una coppia stringa+lingua, ad esempio: (“*Christine Sinclair*”, en).

1.3 Rappresentazioni Grafiche

La relazione *prop* può essere interpretata in termini di *grafo orientato*: l'atomo *prop(Ind, Prop, Val)* può essere raffigurato con i *nodi* *Ind* e *Val* congiunti da un *arco* orientato etichettato con *Prop*. Il grafo risultante dalla rappresentazione di tali relazioni sarà un **knowledge graph** [H+21] o **rete semantica**, con una mappatura diretta su una base di conoscenza tramite *prop*.

Esempio — Parte del KG che riguarda *Christine Sinclair* (Q262802) in Wikidata (agosto'22):



- fra le 3400+ triple si può notare, ad esempio, il nome descritto in 47 lingue.

Diversi sono i vantaggi della notazione grafica:

- la facilità di visualizzazione delle relazioni senza essere costretti a imparare la sintassi di un dato linguaggio logico: ciò è d'ausilio ai progettisti nell'organizzazione della conoscenza essendo disponibili strumenti specifici per costruire *knowledge / mind map*;
- la possibilità di ignorare le etichette con nomi privi di significato (e.g. Q262802-9c5c267f in figura o ga3545 in un esempio precedente): sono ammessi i cosiddetti nodi **blank**, ossia privi di etichetta, che possono servire a dare nomi arbitrari solo in caso di mappatura su un formalismo logico;
- l'efficienza della memorizzazione in un **triple store**: mentre un RDB ha bisogno della progettazione delle chiavi usate per gli indici, un triple store utilizza esattamente 8 indici per ritrovare le triple nelle diverse posizioni; una query basta a determinare se una particolare tripla sia in memoria, mediante semplice enumerazione, e sono possibili casi di query con soggetto, verbo, oggetto eventualmente non specificati.

2 Classi e Proprietà

Tipicamente, di un dominio si conoscono e formalizzano il database di *fatti* e le *regole* generali tramite le quali è possibile derivare altri fatti. La modalità con cui farlo costituisce una scelta progettuale.

Si distingue la **conoscenza primitiva** specificata esplicitamente e la **conoscenza derivata**, inferita da quella primitiva e/o altra conoscenza derivata.

Le regole permettono una rappresentazione più *compatta* della conoscenza: dato che non tutto è osservabile, attraverso relazioni derivate si possono trarre conclusioni a partire da osservazioni. La conoscenza su un dato dominio sarà in gran parte *inferita*, oltre che da fatti osservati e da conoscenza più generale disponibile o anche appresa. Grazie alle regole si potranno raggruppare gli individui in classi ed anche associare *proprietà* generali alle classi, in modo che i loro individui le *ereditino*. In tal modo si hanno rappresentazioni più *concise*: le istanze di una classe condividono attributi comuni, senza necessità di specificarli caso per caso, come si discuterà in seguito (cfr. classificatori probabilistici e apprendimento non supervisionato).

Una **classe** è un insieme di individui che sono sue istanze, sia *effettivi* sia *potenziali*. Essa viene definita in una delle due forme:

- *intensionale*, ossia tramite una **funzione caratteristica** che avrà valore **1/true** per i membri dell'insieme e **0/false** per gli altri individui;
- *estensionale*, ossia come elenco delle sue istanze.

Ad esempio la classe **Chair** sarà l'insieme di tutte le cose che possono essere *sedie*, non limitato a quelle già osservate, per non escludere sedie ancora da inventare / produrre.

Pertanto l'*equivalenza* tra classi **non** dipende dalle sole istanze conosciute. Ad esempio la classe degli unicorni verdi e quella delle sedie alte 120m, pur contenendo entrambe nessun elemento eppure esse rimangono diverse (le sedie non sono unicorni).

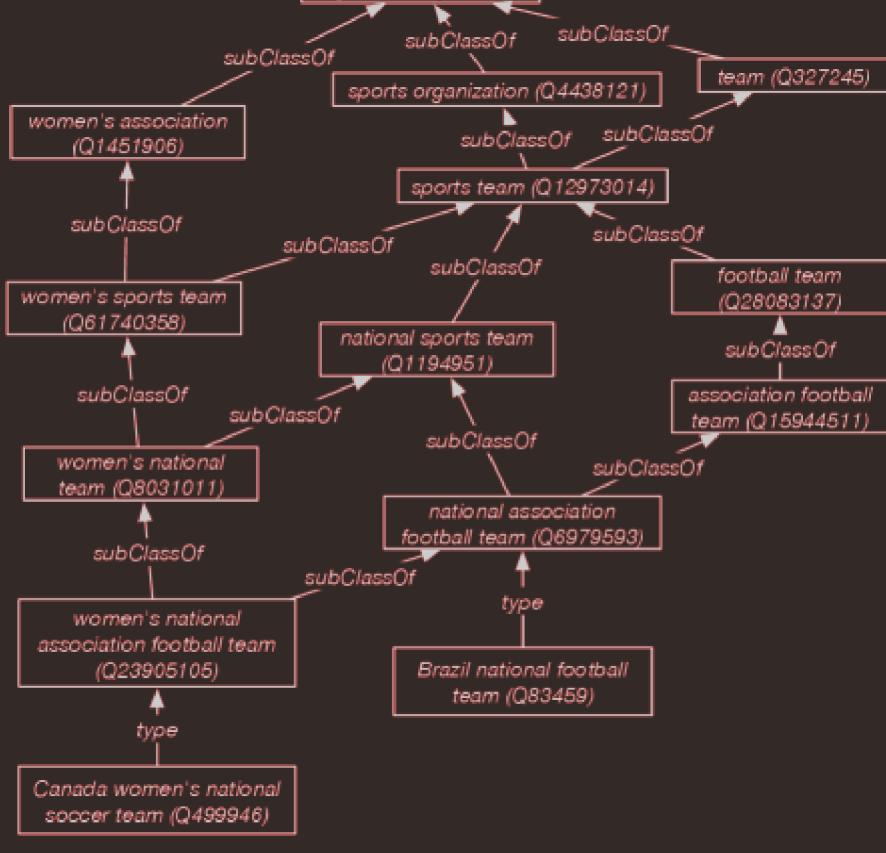
La definizione data consente di descrivere *qualsiasi* insieme come classe: ad esempio una classe che comprende il numero **18**, il **Dipartimento_di_Informatica_di_Bari**, e il **primo_goal_della_SerieA_23_24**. In tal caso, però, bisogna chiedersi se tale classe sia effettivamente *utile*.

Occorrerà cercare di definire una classe come **tipo naturale**, ossia dovrebbe ammettere una descrizione dell'insieme più concisa. Ad esempio la classe **mammifero** è un tipo naturale: descrive attributi comuni ai mammiferi e rende più compatta la KB, evitando ripetizioni per ogni singolo individuo.

2.1 Gerarchie di Classi e Proprietà

S è una **sottoclasse** di **C** se fra di esse intercorre una relazione di *sottoinsieme*: ogni individuo di tipo **S** è anche di tipo **C** (anche quelli futuri!)

Esempio — Struttura classi per le nazionali femminili brasiliana e canadese:



- la nazionale brasiliana è istanza di (ha come tipo) **Q6979593**, classe denominata "national association football team" che, a sua volta, è sottoclasse di "national football team" e "association football team".

La relazione tra tipi e sottoclassi è definibile come *clausola definita* nel modo seguente:

$$prop(X, type, C) \leftarrow prop(S, subClassOf, C) \wedge prop(X, type, S).$$

o, passando alla *notazione funzionale*, usando le proprietà speciali *type* e *subClassOf*:

$$\text{se } subClassOf(S, C) \text{ e } type(E, S) \text{ allora } type(E, C).$$

Per definire l'**ereditarietà di proprietà**, il valore di una proprietà, specificato a livello di classe, sarà ereditato da tutte le sue istanze. Per specificare che tutti gli individui di classe *c* hanno il valore *v* per la proprietà *p*, in **DATALOG** si può scrivere:

$$prop(Ind, p, v) \leftarrow prop(Ind, type, c).$$

che, insieme alla regola precedente che collega tipi e sottoclassi, serve a far ereditare (valori di) proprietà.

Le **proprietà** rappresentano relazioni *binarie* *p* con:

- **dominio**, la classe *C* dei *soggetti* delle triple con *p* come verbo, per cui se $p(x, y)$ allora $x \in C$, quindi se *p* ha dominio *C₁* e dominio *C₂*, allora ogni soggetto apparterrà sia a *C₁* sia a *C₂*;
- **range (codominio)**: la classe *C* per gli *oggetti* delle triple con *p*, per cui se $p(x, y)$ allora $y \in C$;
- *p* è **funzionale** se c'è al più un oggetto associato a ogni soggetto: se $p(x, y_1)$ e $p(x, y_2)$ allora $y_1 = y_2$; in matematica si dice che *p* è una *funzione (parziale)*.

Esempio — Possibili proprietà da aggiungere a Wikidata:

- dominio di **P54** (“member of sports team”): **Q5** (“human”);
- range di **P54** (“member of sports team”): **Q12973014** (“sports team”);
- **P54** non è funzionale, si appartenere a più squadre contemporaneamente, come ad esempio Christine Sinclair (squadra di club e nazionale);
- **P569** (“date of birth”) è funzionale: ogni persona può avere una sola data di nascita.

Una gerarchia di ereditarietà è possibile anche fra proprietà. Si dirà che *p₁* è **sotto-proprietà** di *p₂* se ogni coppia in *p₁* è anche in *p₂*. In notazione funzionale si può scrivere $p_1(x, y)$ implica $p_2(x, y)$, o anche, usando le triple, (x, p_1, y) implica (x, p_2, y) .

Esempio — sotto-proprietà in Wikidata:

- **P54** (“member of sports team”) sotto-proprietà di **P463** (“member of”)
- **P463** sotto-proprietà di **P1416** (“affiliation”)
- **P1416** sotto-proprietà di **P361** (“part of”)
- **P361** sotto-proprietà di **P1382** (“partially coincident with”)
- **P1382** sotto-proprietà di **P1889** (“different from”)

2.2 Progettazione delle Classi

La categorizzazione degli oggetti è alla base delle moderne ontologie. Storicamente già Aristotele aveva proposto un modo di definire una nuova classe *C* in termini di:

- **genus**, una super-classe di *C*;
- **differentia**, gli attributi che distinguono le istanze di *C* dal resto delle istanze della super-classe di *C*.

Ad esempio, nelle ontologie, **animale** e **conoscenza** possono essere definite come *classi* e la proprietà **essere_bipede** può avere come dominio **animale**, mentre non ha senso per un'istanza di **conoscenza** avere un valore per tale proprietà.

Per costruire un'ontologia:

- per ogni classe da definire, occorre determinare una super-classe rilevante e poi selezionare gli attributi che la distinguono dalle altre sottoclassi. Per ogni attributo si definirà una proprietà e un valore;
- per ogni proprietà, si definiscono come dominio e range le classi più generali che abbia senso considerare, anche definendole tramite semplice enumerazione dei valori.

Esempio — *nazionale* = “*squadra che rappresenta una nazione in uno sport*”, con genus “*squadra*” e differentia “*rappresenta una nazione in uno sport*”

- una *squadra di calcio* è una *squadra sportiva che gioca a calcio*;
- una *squadra nazionale di calcio* è una *squadra che rappresenta una nazione nel calcio*, per cui è sottoclasse di entrambe;
- una *squadra* è “*un gruppo con scopi comuni*”.

Nella gerarchia vista in precedenza la differentia che distingue le classi nella parte sinistra è che le istanze devono essere donne:

- si possono concepire anche una classe *squadra di calcio femminile* e una classe *squadra femminile*, pur non avendo identificatori specifici in Wikidata;
- non esiste in Wikidata la classe *squadra di calcio maschile*: le squadre possono tesserare donne, anche se poi nessuna gioca effettivamente nella squadra maschile.

La *gerarchia delle classi* costituisce un grafo aciclico orientato con archi dalle sottoclassi alle rispettive super-classi immediate.

Si tratta di un *reticolo*: sono inutili definizioni con cicli, dato che si creerebbero equivalenze. Si noti che la metodologia non porta necessariamente a gerarchie ad albero, alcuni oggetti possono appartenere a più classi distinte e ogni classe non ha una sola super-classe più specifica.

Esempio — Si considerino le seguenti definizioni:

- **rettangolo**: quadrilatero con tutti angoli retti al suo interno;
- **rombo**: quadrilatero con lati di pari lunghezza;
- **quadrato**: quadrilatero con lati di pari lunghezza e tutti gli angoli retti:
 - è sia **rettangolo** sia un **rombo** (super-classi più specifiche): potrebbe essere definito come **rettangolo** con lati di pari lunghezza oppure come **rombo** con gli angoli interni retti;
 - un **quadrilatero** è una **figura piana** costituita da quattro lati (segmenti) quindi un **quadrato** è una **figura piana** con quattro lati di pari lunghezza e angoli interni retti.

Raramente le gerarchie naturali sono definibili come alberi, ad esempio come la tassonomia di Linneo, per ragioni evolutive. Storicamente imporre forzatamente una struttura ad albero ha avuto scarso successo.

Può essere definita una struttura aciclica, con una radice, detta **thing**:

- definendo ogni classe in termini di una super-classe (genus) e attributi (differentia) con una *forma normale* costituita da **thing** congiunta con gli attributi, ottenuta espandendo ogni super-classe con la sua definizione;
- se gli attributi sono coppie proprietà-valore allora in forma normale, una classe è sotto-classe di un'altra per inclusione, come **quadrato** nell'esempio precedente.

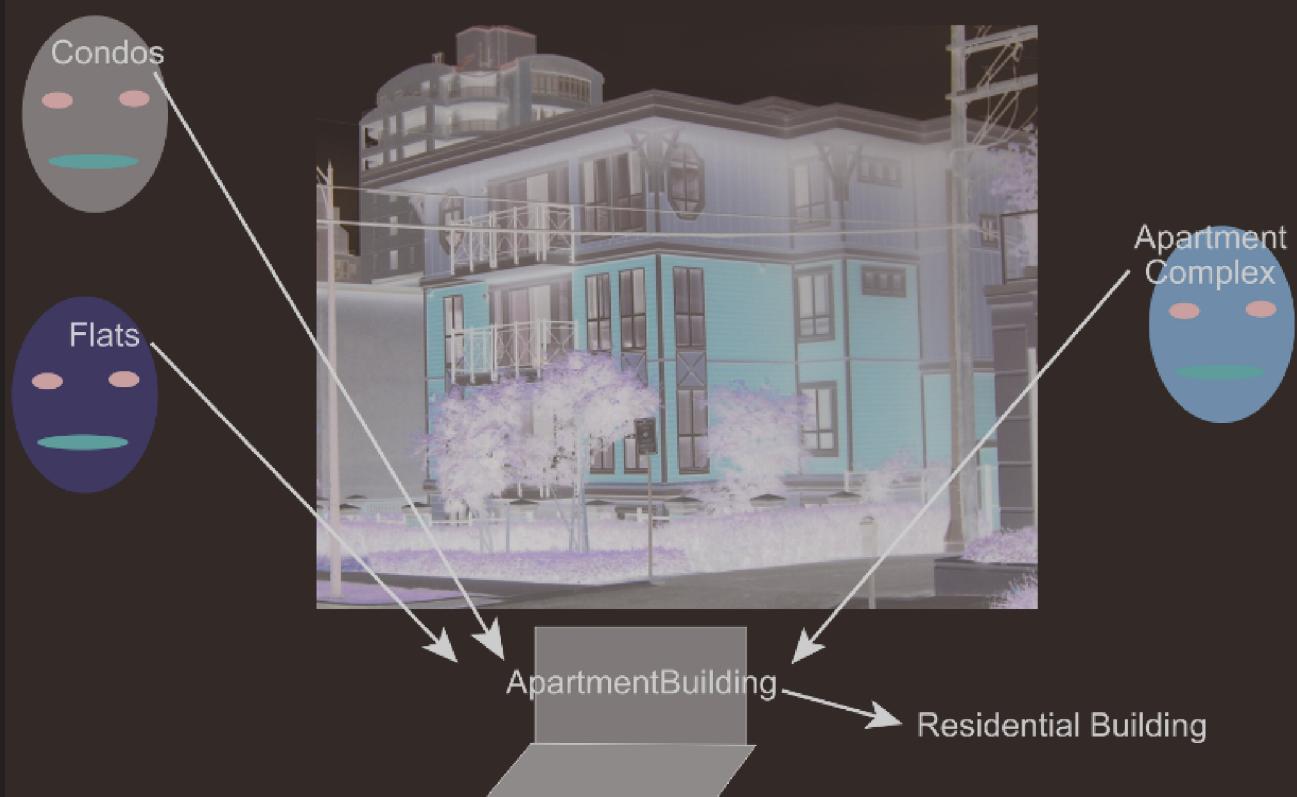
In seguito saranno presentati altri costrutti di classe più espressivi.

3 Ontologie e Condivisione della Conoscenza

Diversi fattori rendono complessa la costruzione di grandi sistemi basati su conoscenza. La conoscenza spesso progettata integrando *più sorgenti*. Queste, a loro volta, potrebbero non condividere uno stesso modo di analizzare il mondo, ognuna potrebbe semplificarlo secondo specifiche priorità originate in campi diversi e con terminologie peculiari. Inoltre i sistemi *evolvono* nel tempo e risulta difficile anticipare distinzioni che potranno essere utili in futuro. Il mondo appare spesso *indistinto* e i progettisti devono scegliere *individui e relazioni* da rappresentare accordandosi su una decomposizione conveniente del dominio. Risulta difficile tenere presenti contemporaneamente sia la propria *notazione* con il suo *significato* sia quella altrui. Dato un simbolo utilizzato, se ne deve saper determinare il significato ovvero, dato un concetto, occorre saper determinare con quale simbolo rappresentarlo: il concetto potrebbe essere stato già definito e associato a un certo simbolo oppure, in alternativa occorre determinare quali siano concetti correlati con cui definirlo.

Per condividere e comunicare conoscenza è importante sviluppare un *vocabolario* con un suo *significato* comunemente accettato. Una **concettualizzazione** è l'associazione tra un vocabolario di simboli usati nella macchina e individui / relazioni del mondo che si intende rappresentare. Essa costituisce una particolare *astrazione* del mondo e la sua *notazione*. In *piccolo* si tratta di una forma di documentazione curata dal progettista, spesso informale ma difficilmente scalabile. Nel caso di *grandi sistemi* occorre una concettualizzazione *condivisa*.

Esempio — Un'ontologia riguardante le *mappe stradali* potrebbe specificare che **ApartmentBuilding** rappresenti edifici con appartamenti, non necessariamente con una definizione formale, ma almeno in modo che gli *altri* ne comprendano la natura. Altri potrebbero usare nomi/simboli diversi, **Condos**, **Flats** o **Apartment Complex**, ma essere comunque in grado di ritrovare il simbolo appropriato nell'ontologia. Si devono creare associazioni fra il concetto, il suo simbolo e il suo significato.



Mediante l'uso di *assiomi* si può vincolare l'uso dei simboli, ad esempio:

- specificando che gli **Apartment Building** siano **Building**, quindi *opere* costruite da uomini;
- imponendo qualche restrizione sulle dimensioni degli edifici in modo da escludere, ad esempio, **Box** o intere **City**;

- dichiarando che un edificio non possa trovarsi simultaneamente in locazioni geograficamente distanti, dato che se si staccasse una sua parte spostandola in un posto diverso, non sarebbe più un edificio singolo;
- essendo un **Apartment Building** anche un **Building**, le relative restrizioni si applicano anche ad ognuna delle sue istanze.

Un'**ontologia**, generalmente definita *indipendentemente* dalle sue applicazioni, richiede l'accordo di una comunità sul significato dei simboli che vi occorrono.

Componenti:

- *vocabolario* delle categorie di cose da rappresentare: classi e proprietà;
- *organizzazione* delle categorie, ad esempio la gerarchia d'ereditarietà definita attraverso proprietà speciali come **subClassOf** o **subPropertyOf**, o altre modalità;
- insieme di *assiomi* che vincolano la definizione di alcuni simboli in modo da riflettere meglio il significato inteso, ad esempio la transitività d'una proprietà, restrizioni su dominio, codominio, o sul numero di valori che la proprietà può assumere per individuo; a volte alcune relazioni sono definite in termini di più relazioni *primitive*, che non hanno una vera e propria definizione (intensionale).

Non serve provvedere a definire altri individui ancora *sconosciuti* in fase progettazione, bastano quelli ritenuti necessari e da condividere per la specifica applicazione come, ad esempio, i giorni della settimana o i colori.

Esempio — sistema per il settore immobiliare:

- I clienti possono descrivere una *sistemazione desiderata*;
- Il sistema potrà:
 - cercare su più KB le sistemazioni più consone;
 - contattare gli utenti quando si libera una sistemazione appropriata.
- Considerare case indipendenti e condomini come edifici residenziali può essere utile per suggerire di affittare una casa / un appartamento, ma non un intero condominio (edificio):
 - si può definire il concetto di *unità abitativa*, gruppo di stanze in cui si convive, che costituisce la tipica offerta da parte delle agenzie immobiliari: singola stanza, o persino una parte / posto letto.
- Basta definire conoscenza (più) stabile e generale: descrizioni/offerte reali sono ignoti in fase di progettazione. Casi limite, non previsti in partenza, o spesso non definiti chiaramente, possono essere delineati in seguito nel corso dell'evoluzione dell'ontologia.

Usi delle Ontologie

Lo scopo *primario* delle ontologie è *documentare* il significato dei simboli, associare simboli (nella macchina) a concetti (nella mente del progettista):

- dato un simbolo, si usa l'ontologia per determinare cosa significhi;
- dovendo rappresentare un concetto, si sfrutta l'ontologia per trovare il simbolo appropriato o per evidenziare la necessità di una nuova definizione.

Scopi *secondari* sono l'*inferenza* o individuare le eventuali *contraddizioni* tramite l'uso degli assiomi.

Il problema principale nella progettazione è definire l'*organizzazione* dei concetti (simboli) in modo che la macchina possa inferire conoscenza utile dai fatti asseriti.

3.1 Web Semantico

Il **Web Semantico** costituisce una prospettiva che prevede la distribuzione di *conoscenza* attraverso l'infrastruttura del Web (server): a differenza dei documenti HTML del Web, destinati alle persone, si mira a conoscenza *interpretabile anche dalle macchine*, con una semantica esplicitata attraverso la definizione di ontologie formali che permettano il ragionamento automatico.

Gli ingredienti-base sono costituiti dagli identificatori URI utilizzati in rappresentazioni / linguaggi quali RDF (SPARQL), RDF-Schema e OWL.

Interoperabilità

XML (Extensible Markup Language) è un linguaggio che fornisce una *sintassi* progettata per l'elaborazione automatica leggibile anche dalle persone. Si tratta di un linguaggio testuale, con elementi costituiti da *tag* che possono essere organizzati gerarchicamente. La sintassi, anche complessa (grammatiche libere), prevede al livello basilare ambiti di tag della forma <tag...> oppure <tag...> ... </tag>.

A livello *semantico* occorre adottare ontologie di riferimento, riusando quelle esistenti anche per svilupparne di nuove in proprio. Ai fini dell'*interoperabilità semantica*, si dovrebbe tendere a riusare ontologie *standard* per i domini interessati e/o definire *mapping* dalla propria verso ontologie standard. Diversi sforzi sono stati fatti in passato tesi alla costruzione di grandi ontologie universali come, ad esempio, Cyc.

Identificatori

Un **URI** (*Uniform Resource Identifier*) identifica *univocamente* una *risorsa*, ovvero qualsiasi cosa debba essere identificata: individui, classi e proprietà. La sintassi è basata su quella degli URL: <`url#name`> dove `url` è l'indirizzo di una pagina Web, con HTTP si può *negoziare* il tipo di risposta (HTML/RDF) in base alla richiesta del client; si usano anche IRI, con un set-caratteri esteso (Unicode). Alcuni esempi sono i seguenti:

- individuo <<http://example.org/#spiderman>>
- proprietà <<http://xmlns.com/foaf/0.1/name>>

Un URI si può esprimere in forma *abbreviata*, detta *CURIE*, `abbr:name`, dove `abbr` è un *prefisso* dell'URI completo (*namespace*) dichiarato localmente. Ogni URI avrà un significato convenzionale condiviso determinato dal suo uso.

Esempio — L'ontologia **foaf** (*friend-of-a-friend*) serve a pubblicare informazioni su persone, reti di amici, ...

- gli URI di foaf sono associati al *namespace* <http://xmlns.com/foaf/0.1/>, abbreviato tipicamente con il prefisso `foaf`;
- `foaf:name` è la proprietà che correla una persona a una rappresentazione del suo nome (una stringa); essa consentirà di sapere esattamente a quale proprietà ci si riferisce; si noti che l'URI della proprietà è formato premettendo il prefisso del namespace;
- `foaf:knows` serve a per collegare una persona ai suoi amici;
- ecc.

RDF (*Resource Description Framework*) definisce un *modello di dati* in forma di *triple*:

`individuo–proprietà–valore`

La sintassi (serializzazione) è tipicamente basata sull'**XML**, in al caso si parla di **RDF/XML**, ma esistono altri formati più leggibili come **JASON-LD**, **N3** o **TURTLE**; in quest'ultimo si scrive:

Un esempio concreto potrebbe essere il seguente:

<http://example.org/#spiderman> <http://xmlns.com/foaf/0.1/name> "Uomo Ragno"@it.

È ammessa la *reificazione di enunciati*, formule logiche arbitrarie, quindi in generale, non è decidibile. Ciò non costituisce sempre un problema: significa solo non poter limitare il tempo necessario a un dato calcolo, analogamente a quanto accade con i programmi logici quando si ammettono le funzioni.

SPARQL è il nome del protocollo e del linguaggio per l'interrogazione di KB costituite da grafi di triple RDF. A tale scopo si utilizzano specifici *endpoint*, quali, ad esempio: DBpedia, Min. Istruzione, INPS, Regione Puglia, ... Le query sono definite da *pattern* costituiti da triple contenenti URI o *variabili*:

Esempio — Testabile attraverso l'endpoint di DBpedia

```
select ?canzone ?data where {
    dbr:The_Sugarcubes dbo:formerBandMember ?componente .
    ?componente rdf:type yago:Female109619168 .
    ?canzone dbo:artist ?componente .
    ?canzone dbo:releaseDate ?data .
} limit 3
```

risultati:

```
http://dbpedia.org/resource/Cosmogony\_\(song\) 2011-07-19
http://dbpedia.org/resource/Hunter\_\(Björk\_song\) 1998-10-05
http://dbpedia.org/resource/Declare\_Independence 2008-01-01
```

RDF-S (*RDF Schema*) consente di definire risorse, classi, proprietà, in termini di altre risorse, ad esempio è possibile *restringere* domini e range di proprietà. Esso fornisce anche *contenitori*: set, sequenze e alternative.

Esempio — Uso di RDF e RDF-S

```
@prefix : < http://www.example.org/sample.rdfs# >.
@prefix rdf: < http://www.w3.org/1999/02/22-rdf-syntax-ns# >.
@prefix rdfs: < http://www.w3.org/2000/01/rdf-schema# >.

:Dog      rdfs:subClassOf :Animal.
:Person   rdfs:subClassOf :Animal.
:hasChild rdfs:range :Animal;
            rdfs:domain :Animal.
:hasSon   rdfs:subPropertyOf :hasChild.

:Max      rdf:type :Dog.
:Abel     a :Person.
:Adam     :hasSon :Abel.
```

3.2 Logiche Descrittive

I linguaggi ontologici come **OWL** si basano sulle **logiche descrittive** [DL] che servono a descrivere *concetti* (classi), *ruoli* (proprietà) e *individui*. L'idea fondante è la distinzione nella *base di conoscenza* di due parti $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$:

- \mathcal{T} costituisce la parte **terminologica** o **TBox** che descrive la terminologia, ossia assiomi per la definizione del significato dei simboli. Ad esempio, in notazione logico-matematica:

$$C \sqsubseteq (C_1 \sqcap \exists R_1. C_2) \sqcup \forall R_2. (C_3 \sqcap \neg C_4)$$

Per la descrizione in **OWL** degli assiomi sono possibili diverse sintassi. Essa viene definita in fase di progettazione del sistema e ne specifica l'ontologia tende, quindi, ad essere stabile, a meno che non si debba modificare il significato del vocabolario (caso raro).

- \mathcal{A} costituisce la parte **asserzionale** o **ABox** che specifica verità fattuali. Ad esempio in DL:

$$C(a), R(a, b), P(b, v)$$

facilmente descrivibili come asserzioni in forma di *triple* **RDF**. Esse riguardano conoscenza su situazioni contingenti, lo *stato del mondo*, di solito completamente noto solo al momento dell'utilizzo (*runtime*).

3.2.1 OWL

OWL (*Web Ontology Language*) è linguaggio con cui vengono rappresentate le ontologie pubblicate attraverso il Web.

Con **OWL** è possibile descrivere mondi/domini in termini di:

- **Individui** — *entità* del mondo che si descrive, e.g., l'URI di una data casa o una particolare prenotazione.
- **Classi** — *insiemi di individui*, tutte le entità reali o potenziali che potrebbero appartenervi; ad esempio **House** insieme di tutte le cose, anche future, classificabili come *case*.
- **Proprietà** — *relazioni* che descrivono individui associandoli ad altri *individui*, **object property**, oppure a *dati*, *valori* di tipi predefiniti, come gli interi o le stringhe, dette in tal caso **datatype property**:
 - ad esempio **nextTo** relazione tra case e **onStreet** tra case e vie;
 - ad esempio **streetName** può associare vie a stringhe (loro nomi).

Le *varianti* di OWL differiscono nelle restrizioni su classi/proprietà e nell'implementazione:

- in **OWL-DL** una classe non può essere un individuo o una proprietà e una proprietà non è un individuo;
- in **OWL-Full**, individui, proprietà e classi *non* necessariamente *disgiunti*.

Con l'introduzione di **OWL2**, vi sono *profili* orientati verso specifiche applicazioni che limitano i costrutti usabili, per garantire l'efficienza dell'*inferenza*:

- **OWL2 EL** consente descrizioni con molti dettagli strutturali, utili, ad esempio, con le grandi ontologie biomediche;
- **OWL2 QL** pensato come front-end per i *linguaggi di interrogazione* per DB;
- **OWL2 RL** progettato per l'uso di regole.

Default in OWL

- non si assume la *UNA*: due nomi/URI non denotano necessariamente individui o classi diversi;
- si assume una semantica di *mondo aperto*: non si può assumere che tutti i fatti rilevanti siano stati dichiarati (*conoscenza completa*).

Le principali **classi predefinite** e i **costruttori di classe** di **OWL** sono elencati nella seguente tabella. La loro semantica è data dall'**estensione** corrispondente (insieme di individui). Nella tabella che segue, le varie C_i sono classi, p è una generica proprietà, la *tripla* $x p y$ corrisponde all'atomo $p(x, y)$:

Classe	Estensione
<code>owl:Thing</code>	tutti gli individui
<code>owl:Nothing</code>	nessun individuo
<code>owl:ObjectIntersectionOf(C_1, \dots, C_k)</code>	individui in $C_1 \cap \dots \cap C_k$
<code>owl:ObjectUnionOf(C_1, \dots, C_k)</code>	individui in $C_1 \cup \dots \cup C_k$
<code>owl:ObjectComplementOf(C)</code>	individui non in C
<code>owl:ObjectOneOf(I_1, \dots, I_k)</code>	I_1, \dots, I_k
<code>owl:ObjectHasValue(p, v)</code>	$\{x \mid p(x, v)\}$
<code>owl:ObjectAllValuesFrom(p, C)</code>	$\{x \mid \forall y : xPy \rightarrow y \in C\}$
<code>owl:ObjectSomeValuesFrom(p, C)</code>	$\{x \mid \exists y \in C : xPy\}$
<code>owl:ObjectMinCardinality(n, p, C)</code>	$\{x \mid \#\{y \mid xPy \wedge y \in C\} \geq n\}$
<code>owl:ObjectMaxCardinality(n, p, C)</code>	$\{x \mid \#\{y \mid xPy \wedge y \in C\} \leq n\}$
<code>owl:ObjectHasSelf(p)</code>	$\{x \mid xPx\}$

I **predicati predefiniti OWL** hanno un'interpretazione prefissata (alcuni erano già disponibili in RDF e RDF-S). Nelle seguenti tabelle si userà la **notazione funzionale**, inoltre x e y s'intendono universalmente quantificate:

Predicato	Semantica
<code>rdf:type(I, C)</code> <code>owl:ClassAssertion(C, I)</code>	$I \in C$
<code>rdfs:subClassOf(C_1, C_2)</code> <code>owl:SubClassOf(C_1, C_2)</code>	$C_1 \subseteq C_2$
<code>rdfs:domain(p, C)</code> <code>owl:ObjectPropertyDomain(p, C)</code>	se $p(x, y)$ allora $x \in C$
<code>rdfs:range(p, C)</code> <code>owl:ObjectPropertyRange(p, C)</code>	se $p(x, y)$ allora $y \in C$
<code>owl:EquivalentClass(C_1, C_2, \dots, C_k)</code>	$C_i \equiv C_j$ per ogni i, j
<code>owl:DisjointClass(C_1, C_2, \dots, C_k)</code>	$C_i \cap C_j = \emptyset$ per ogni $i \neq j$

Predicato	Semantica
<code>rdfs:subPropertyOf(p_1, p_2)</code>	$p_1(x, y)$ implica $p_2(x, y)$
<code>owl:EquivalentObjectProperties(p_1, p_2)</code>	$p_1(x, y)$ sse $p_2(x, y)$
<code>owl:DisjointObjectProperties(p_1, p_2)</code>	$p_1(x, y)$ implica $\neg p_2(x, y)$
<code>owl:InverseObjectProperties(p_1, p_2)</code>	$p_1(x, y)$ sse $p_2(y, x)$
<code>owl:SameIndividual(I_1, \dots, I_n)</code>	$\forall j \forall k \quad I_j = I_k$
<code>owl:DifferentIndividuals(I_1, \dots, I_n)</code>	$\forall j \forall k \quad j \neq k$ implica $I_j \neq I_k$
<code>owl:InverseFunctionalObjectProperty(p)</code>	se $p(x_1, y)$ e $p(x_2, y)$ allora $x_1 = x_2$
<code>owl:TransitiveObjectProperty(p)</code>	se $p(x, y)$ e $p(y, z)$ allora $p(x, z)$
<code>owl:SymmetricObjectProperty(p)</code>	se $p(x, y)$ allora $p(y, x)$
<code>owl:AsymmetricObjectProperty(p)</code>	$p(x, y)$ implica $\neg p(y, x)$
<code>owl:ReflectiveObjectProperty(p)</code>	$p(x, x)$ per ogni x
<code>owl:IrreflexiveObjectProperty(p)</code>	$\neg p(x, x)$ per ogni x

Sono possibili altre sintassi, basate su XML, TURTLE, eccetera.

Esempio — Costruttori di classe

- `ObjectHasValue(country_of_citizenship, Q16)`
classe dei cittadini del Canada (Q16);
- `ObjectSomeValuesFrom(country_of_citizenship, Q113489728)`
classe delle persone con cittadinanza in un paese membro dell'OECD
Organization for Economic Co-operation and Development, dove Q113489728 identifica la classe dei paesi membri dell'OECD; in caso di più cittadinanze, almeno una deve riguardare un paese dell'OECD;
- `MinCardinality(2, country_of_citizenship, Q113489728)`
classe di persone con cittadinanza in 2 o più paesi dell'OECD.

Enunciati

I costruttori di classe vanno usati negli *enunciati*, ad esempio per dire che un individuo appartiene a tale classe o che una classe sia equivalente a un'altra.

OWL non può essere espresso, in modo equivalente, attraverso clausole definite; per dire che tutti gli elementi di *S* hanno il valore *v* per un dato predicato *p*, si dice che *S* è sottoinsieme di quello di tutte le cose con valore *v* per *p* (*assioma di inclusione*, \sqsubseteq).

Esempio — Definizione di edificio con appartamenti (*apartment building*): *edificio residenziale* con più unità abitative che sono date in affitto; si noti la differenza con il *condominio* (unità vendute separatamente) o la *casa* (una sola unità).

```
FunctionalObjectProperty(numberOfunits)
ObjectPropertyDomain(numberOfunits, ResidentialBuilding)
ObjectPropertyRange(numberOfunits, ObjectOneOf(two, one, moreThanTwo))

FunctionalObjectProperty(ownership)
ObjectPropertyDomain(numberOfunits, ResidentialBuilding)
ObjectPropertyRange(numberOfunits, ObjectOneOf(rental, ownerOccupied, coop))

EquivalentClasses(ApartmentBuilding,
ObjectIntersectionOf(
    ResidentialBuilding,
    ObjectHasValue(numberOfunits, moreThanTwo),
    ObjectHasValue(ownership, rental)))
```

- definizione utile a rispondere a query sulle loro proprietà (ereditate)

Fra gli altri costrutti disponibili, c'è il *costruttore di proprietà* `owl:ObjectInverseOf(p)` utile a rappresentare la proprietà inversa di *p*, denotata anche con p^{-1} :

$$p^{-1}(y, x) \text{ sse } p(x, y)$$

vale solo per *object-property*: le *datatype-property* non hanno proprietà inverse, in quanto i valori di *tipi concreti* non possono fungere da soggetto di triple.

Ci sono anche classi *datatype* corrispondenti ai codomini di proprietà, come `owl:DataSomeValuesFrom` e `owl:EquivalentDataProperties`, con definizioni analoghe a quelle per proprietà su individui ma con domini *concreti* come range, presi in prestito da **XML-SCHEMA (xsd:)**.

Infine, sono disponibili anche costrutti per definire proprietà, *commenti*, *annotazioni*, *versioning* e *import* da altre ontologie.

Ontologie di Dominio

Una **ontologia di dominio** riguarda un particolare dominio d'interesse: molte delle ontologie esistenti riguardano un dominio ristretto definito per specifiche applicazioni.

Linee Guida per definire ontologie di dominio sono le seguenti:

- Se possibile, conviene usare ontologie *esistenti*: la KB potrà interagire con altre che le adottano;
- Se esiste un'ontologia che non corrisponda esattamente ai requisiti, la si può *importare* facendo poi delle *aggiunte* in modo da non partire da zero: se la propria ontologia include e migliora l'altra, altri la vorranno adottare, aumentando l'interoperabilità della loro applicazione;
- Occorre assicurarsi che l'ontologia *si integri* con ontologie affini, tentando di assicurare l'uso della stessa terminologia per le stesse cose (ad esempio un'ontologia sui resort potrebbe essere integrata con altre su cibi, spiagge, sport, ecc.);
- Tentare di conformarsi a ontologie di *livello superiore* poiché questo rende molto più facile l'integrazione della conoscenza di/con altre fonti;
- Nel progettare una nuova ontologia, conviene consultarsi con altri *utenti potenziali*, ciò la rende più utile e più facilmente adottabile;
- Vanno seguite le *convenzioni* di denominazione, ad esempio chiamare una classe con il nome singolare delle istanze **Resort** e non **Resorts** o, peggio, **ResortConcept**
 - pensare all'uso delle classi / proprietà
 - di un individuo **r1** meglio dire che "è un **Resort**" rispetto a "è un **Resorts**" oppure "è un **ResortConcept**"
- Specificare la corrispondenza tra ontologie: a volte serve un *allineamento* quando le ontologie sono sviluppate separatamente, evitando, se possibile, la ridondanza che le rende più complicate da usare.

Editor OWL

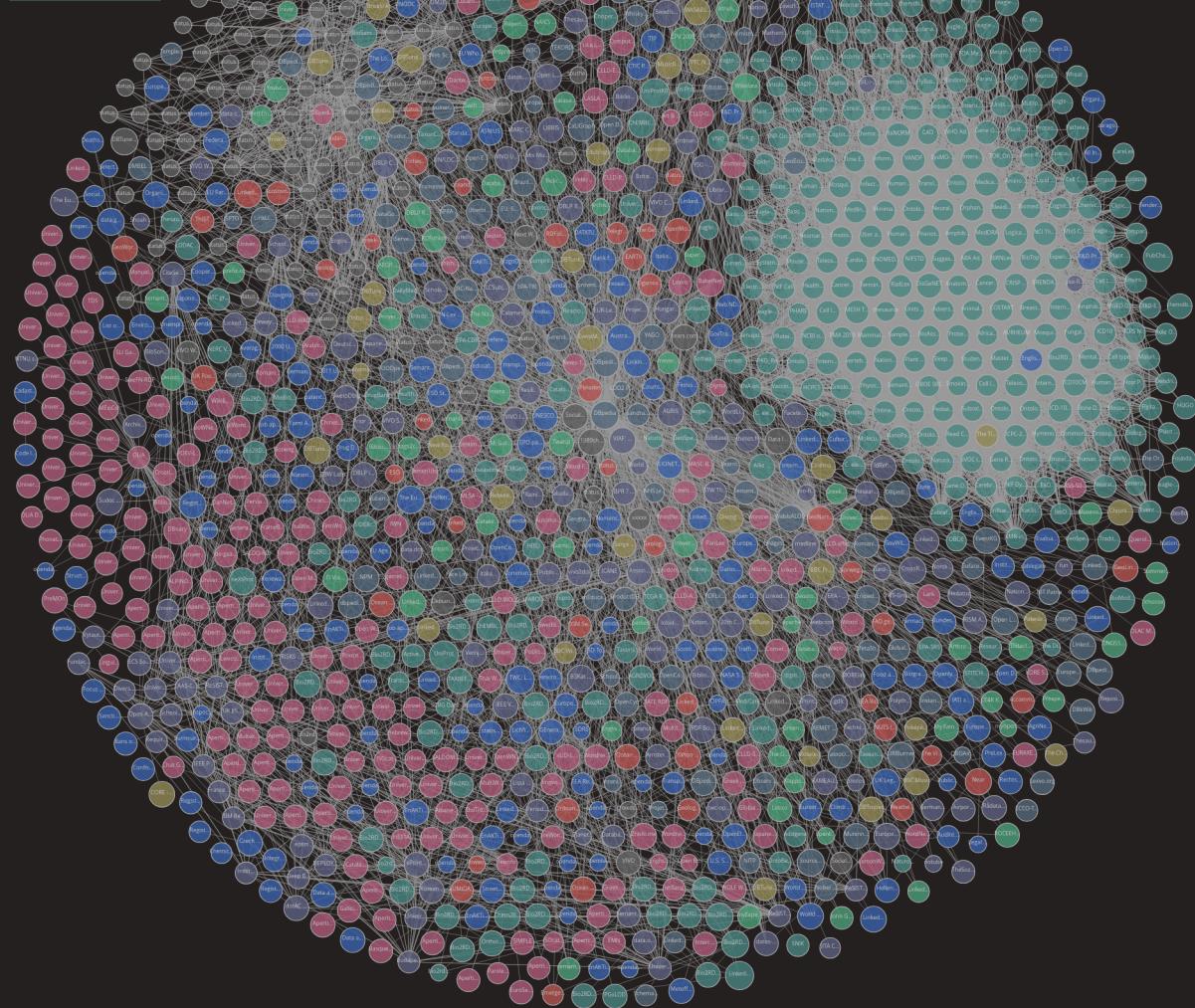
Gli editor di ontologie, come Protégé forniscono un modo per definire ontologie a un *livello di astrazione* appropriato:

- volendo usare un concetto, ne facilitano la *ricerca* nella terminologia o può segnalarne l'eventuale assenza;
- aiutano a determinare immediatamente il *significato* di un termine;
- permettono controlli di *correttezza* sulle ontologie, ad esempio sulla corrispondenza con l'interpretazione intesa per i termini
- facilitano la *riusabilità* spingendo a usare il più possibile un linguaggio standard.

Web of Data

Il **Web of Data** è una *prospettiva* del Web Semantico tesa a sfruttare l'infrastruttura del Web per creare grafo globale di sorgenti *distribuite* di dati collegati.

La semantica dei dati viene resa disponibile anche alle macchine sulla base di ontologie **RDF/OWL**, e quindi attraverso query **SPARQL**, o veri e propri servizi di *ragionamento* automatico (tramite *reasoner*). A tale scopo sono disponibili anche server ad hoc detti *triple store* o altre forme di DB NoSQL, come ad esempio *document DB* basati sulla serializzazione JSON-LD.



The Linked Open Data Cloud from lod-cloud.net



Da lod-cloud.net

I principi-guida dei **Linked Data** [HB11] sono i seguenti:

1. l'uso di *URI* come nomi per le cose;
2. in particolare *URI HTTP* per permetterne la ricerca sull'infrastruttura del Web;
3. le informazioni vengono fornite usando linguaggi *standard* quali **RDF**, **SPARQL**, ...
4. l'uso di *link* verso URI di risorse esterne che abilitano la scoperta di altra conoscenza.

La qualità degli *Open Data* può essere valutata in termini di una categorizzazione 1-5 stelle di TBL [HB11].

5★ — Linked Open Data

- navigabili attraverso strumenti come LodLive, ad esempio LOD su film girati in Puglia.

3.3 Ontologie Top-Level

Ogni ontologia di dominio implicitamente o esplicitamente assume un'ontologia di più alto livello alla quale conformarsi.

Ad esempio l'ontologia degli appartamenti può essere riusata da chi voglia definire una KB che si riferisca a cose che possono comparire in mappe, assumendo che gli edifici siano già definiti altrove.

Tipicamente, in un'ontologia **top-level** si troveranno definizioni a un livello di astrazione molto elevato. Essa fornirà una categorizzazione di base alla quale altre ontologie (su domini diversi) potranno conformarsi esplicitamente per facilitare la loro *integrazione*; inoltre consentirà a diverse applicazioni di riferirsi a più KB, ognuna delle quali potrà adottare ontologie diverse.

Progettare un'ontologia top-level è difficile ma il suo uso aiuta a connettere le diverse basi di conoscenza.

Riferimenti Bibliografici

- [PM23] D. Poole, A. Mackworth: *Artificial Intelligence: Foundations of Computational Agents*. 3/e. Cambridge University Press. 2023 [Ch.16]
- [H+21] A. Hogan, et al.: *Knowledge Graphs*. ACM Comput. Surv. 54(4): 71:1-71:37. ACM 2021
- [HB11] T. Heath & C. Bizer *Linked Data: Evolving the Web into a Global Data Space* Morgan & Claypool. 2011. [sito]
- [HKR09] P. Hitzler, M. Krötzsch, S. Rudolph: *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC. 2009 [sito]
- [PMG98] D. Poole, A. Mackworth, R. Goebel: *Computational Intelligence: A Logical Approach*. Oxford University Press. 1998
- [RN20] S.J. Russell, P. Norvig: *Artificial Intelligence*. Pearson. 4th Ed. 2020 (ch. 9-10)
- [Sow99] J. Sowa: *Knowledge Representation: Logical, Philosophical, and Computational Foundations* Brooks Cole/Cengage. 1999
- [BL04] R. J. Brachman and H. J. Levesque: *Knowledge representation and reasoning*. Morgan Kaufmann. 2004
- [BHL01] T. Berners-Lee, J. Hendler and O. Lassila: *The semantic web: a new form of web content that is meaningful to computers will unleash a revolution of new possibilities*. Scientific American May, pp. 28–37 2001
- [JvHH15] K. Janowicz, F. van Harmelen, J. A. Hendler and P. Hitzler: *Why the data train needs semantic rails*. AI Magazine 36 (1), pp. 5–14. 2015
- [Kow14] R. A. Kowalski: *Logic for problem solving, revisited*. Books on Demand. 2014

Link

- [DL] Description Logics Homepage
- [Protégé] Protégé standalone o anche via Web
- [JSON-LD] sito-base
- [Owlready] Programmazione *ontology-oriented*: Owlready2 — libreria Python per OWL/RDF

Altro:

- Semantic Web presso il W3C
- IRI
- RDF e RDF-Schema: Notation3 (N3), N-Triples, Turtle, TriG
- OWL2
- SWRL *linguaggio a regole* basato su OWL e RULEML
- Interrogazione: SPARQL
- Inferenza
- Linked Data: linkeddata.org, Linked Open Vocabularies, wikidata, DBpedia e relativo endpoint