# Indice

# Capitolo 1
# Introduzione

## 1.1 Agente intelligente

Si definisce **agente intelligente**, o semplicemente **agente**, qualsiasi entitá in grado di percepire l'ambiente in cui si trova mediante sensori e modificando tale ambiente compiendo delle azioni, mappando percezioni ad azioni. Con **ambiente** si intende la parte di universo a disposizione delle percezioni dell'agente e da questa influenzabile. L'intelligenza artificiale é definibile come lo studio degli agenti.

> Un essere umano puó essere modellato come un agente, potendo percepire l'ambiente tramite occhi, orecchie e altri organi e agendo su di esso per mezzo dei suoi arti. Allo stesso modo, un robot puó essere modellato come un agente, percependo l'ambiente attraverso telecamere o sensori infrarossi e agendo su di esso mediante appendici e/o motori elettrici. Infine, anche un programma per computer puó essere modellato come un agente, se si considera l'input umano (tramite tastiera, mouse, touchscreen o voce) come percezione ed il suo output (scrivere su un file, mostrare un contenuto a schermo, generare un suono, eccetera) come azione compiuta sull'ambiente.

La sequenza di percezioni di un agente é la storia completa di tutto ció che l'agente ha percepito. In generale, la scelta dell'azione compiuta da un agente in un certo istante dipende dalla sua conoscenza a priori e/o dall'intera sequenza di percezioni precedente. Formalmente, il comportamento di un agente é descritto da una funzione agente che mappa sequenze di percezioni in azioni: $f : \mathrm{Pow}(P) \to A$. Tale funzione é un concetto astratto, una caratterizzazione *esterna* di un agente: *internamente*, la funzione agente di un agente intelligente é implementata da un **programma agente**; tale funzione viene eseguita da un dispositivo elettronico dotato di sensori di sorta, chiamato **architettura**.

Un **agente razionale** é un agente che "fa la scelta giusta". La nozione di "scelta giusta" comunemente adottata nel campo dell'intelligenza artificiale é il **consequenzialismo**: il comportamento dell'agente é valutato sulla base delle conseguenze delle sue azioni. Se un agente, in relazione ad un certa percezione, compie una azione desiderabile dal punto di vista dell'utilizzatore, allora tale agente ha compiuto la "scelta giusta", ed é definibile agente razionale. La nozione di desiderabilitá viene descritta da una **misura di prestazione** che valuta ogni sequenza di stati in cui l'ambiente si trova. In genere, é preferibile definire una misura di prestazione rispetto a ció che si vuole accada all'ambiente piuttosto che rispetto al modo in cui ci si aspetta che funzioni.

É allora possibile fornire una definizione operativa di agente razionale: per ogni possibile sequenza di percezioni, un agente razionale sceglierá di compiere l'azione che, sulla base delle percezioni precedenti e sulla base della conoscenza che possiede a priori, restituisce il massimo valore possibile in termini di misura di prestazione. Si noti come "razionale" non significhi "onniscente", ovvero in grado di prevedere con assoluta certezza ció che accadrá in futuro, dato che questo é realisticamente impossibile; un agente razionale deve limitarsi a compiere azioni che massimizzano la prestazione *attesa*.

La definizione di agente razionale sopra presentata prevede che questo possieda anche una qualche nozione di **apprendimento**: per quanto la sua configurazione iniziale possa essere fissata, questa puó venire modificata e potenziata con l'esperienza. Nel caso in cui l'ambiente sia interamente conosciuto a priori, l'agente non ha alcuna forma di apprendimento, limitandosi a compiere le azioni preimpostate.

Un agente che compie azioni esclusivamente sulla base della sua conoscenza a priori e non fa uso di apprendimento si dice che non é **autonomo**. Un agente razionale dovrebbe invece essere autonomo, ovvero partire sí da una base di conoscenza pregressa ma, attraverso l'apprendimento, colmarne le lacune. Dopo abbastanza esperienza, ci si aspetta che un agente razionale diventi di fatto indipendente dalla sua conoscenza a priori.

É possibile classificare gli ambienti rispetto a cinque metriche informali, utili a ragionare sulla difficoltá del problema e sulla modalitá risolutiva da adottare:

- **Accessibile** o **inaccessibile**. Un ambiente é tanto accessibile quanto un agente é in grado di ottenere le informazioni sul suo stato di cui necessita con completa accuratezza. Un ambiente puó essere inaccessibile perché i sensori dell'agente non sono precisi oppure perché parte dell'ambiente é del tutto preclusa ai sensori dell'agente. Gli ambienti nel mondo reale hanno necessariamente un certo grado di inaccessibilitá;
- **Deterministico** o **non deterministico**. Un ambiente é deterministico (in riferimento alle azioni dell'agente) se la sua evoluzione é completamente determinata dal suo stato attuale e dalle azioni dell'agente. Un ambiente é non deterministico se la sua evoluzione é anche influenzata da forze al di lá dell'agente. Il mondo fisico da modellare ha sempre un certo grado di non determinismo;
- **Episodico** o **sequenziale**. In un ambiente episodico l'esperienza di un agente puó essere divisa in step atomici dove la scelta di un azione dipende esclusivamente dalla percezione attuale. In un ambiente sequenziale le azioni che un agente compie possono dipendere del tutto o in parte da quali azioni sono state prese in precedenza;
- **Statico** o **dinamico**. Un ambiente é statico se non subisce modifiche mentre l'agente sta deliberando, altrimenti é dinamico;
- **Discreto** o **continuo**. Un ambiente é discreto se il numero di stati in cui questo puó trovarsi é finito, ovvero se é possibile (almeno in linea teorica) enumerare tutti i suoi possibili stati, altrimenti é continuo. Essendo i computer discreti per definizione, modellare un ambiente continuo attraverso un sistema automatico richiederá sempre un certo grado di approssimazione.

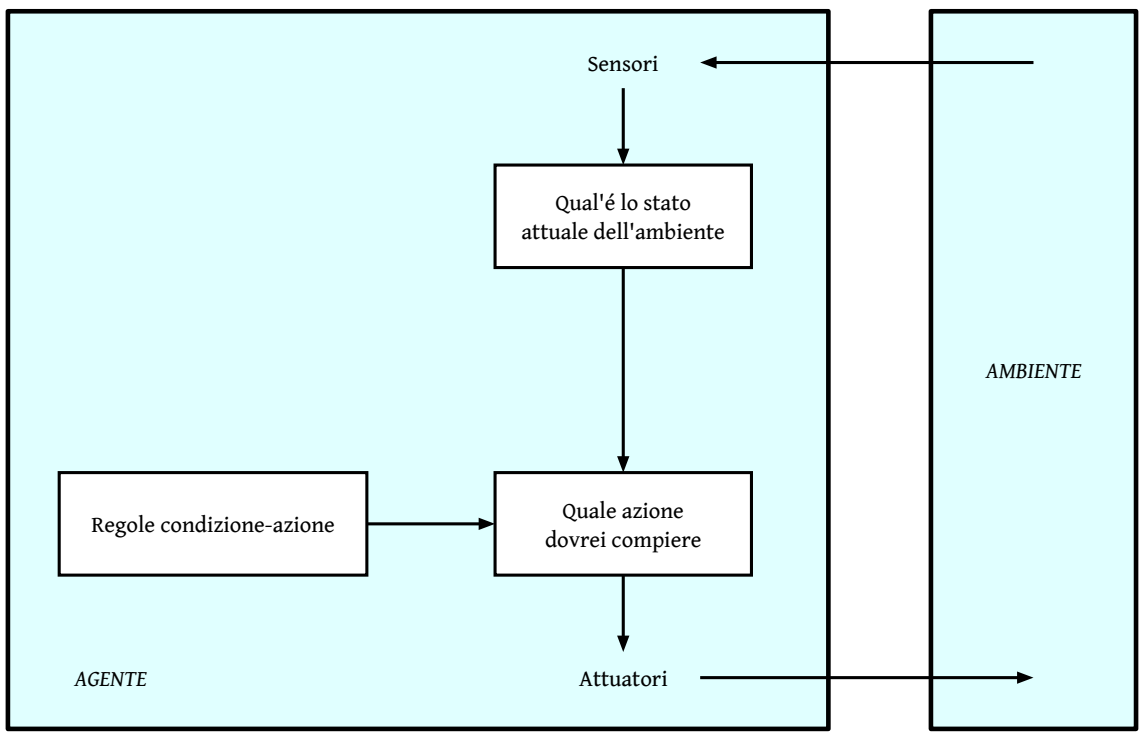
- Si consideri come ambiente il gioco degli scacchi e come agenti i giocatori umani (si assuma che le mosse non abbiano alcun limite di tempo). Tale ambiente é:
  1. Accessibile, perché ciascun giocatore ha completa conoscenza dello stato della partita;
  2. Deterministico, perché l'evoluzione degli stati dipende esclusivamente da quali mosse scelgono di compiere i giocatori;
  3. Sequenziale, perché le mosse di un giocatore possono anche dipendere da quali mosse ha compiuto in precedenza;
  4. Statico, perché durante l'esecuzione di una mossa e durante la scelta della stessa lo stato della partita rimane invariato;
  5. Discreto, perché il numero di possibili stati in cui la partita puó trovarsi é finito.
- Si consideri come ambiente le strade di una cittá e come agente un sistema di guida automatico per automobili. Tale ambiente é:
  1. Inaccessibile, perché non é possibile conoscere l'intero stato del traffico di tutta la cittá in ciascun istante;
  2. Non deterministico, perché l'evoluzione del traffico non dipende esclusivamente dalle scelte dell'agente;
  3. Sequenziale, perché la scelta di quale strada percorrere puó dipendere anche da quali strade ha percorso in precedenza;
  4. Dinamico, perché lo stato della cittá e del traffico cambiano anche mentre l'agente é in movimento;
  5. Continuo, perché lo stato della cittá e del traffico si modificano costantemente.

Gli agenti intelligenti possono essere informalmente classificati in quattro categorie, di crescente ordine di complessitá.

### 1.1.1 Agenti con riflessi semplici

Gli agenti piú facili da realizzare sono gli **agenti con riflessi semplici**. Questi agenti non hanno alcun modello dell'ambiente: scelgono che azione compiere esclusivamente sulla base della percezione attuale e non hanno cognizione delle percezioni precedenti.
Agenti di questo tipo scelgono che azioni compiere seguendo **regole condizione-azione**: *se* si verifica una certa condizione, *allora* viene compiuta l'azione associata a tale condizione.
Una rappresentazione schematica di un agente con riflessi semplici é presentata in basso. La funzione `INTERPRET-INPUT` genera una descrizione astratta della percezione ricevuta dall'agente, mentre la funzione `RULE-MATCH` restituisce la prima azione associata a tale rappresentazione di percezione nel set di regole `rules`.

```
rules <= set of condition-action rules

function SIMPLE-REFLEX-AGENT(percept)
 state <= INTERPRETER-INPUT(percept)
 rule <= RULE-MATCH(state, rules)
 action <= rule.action
 return action
```



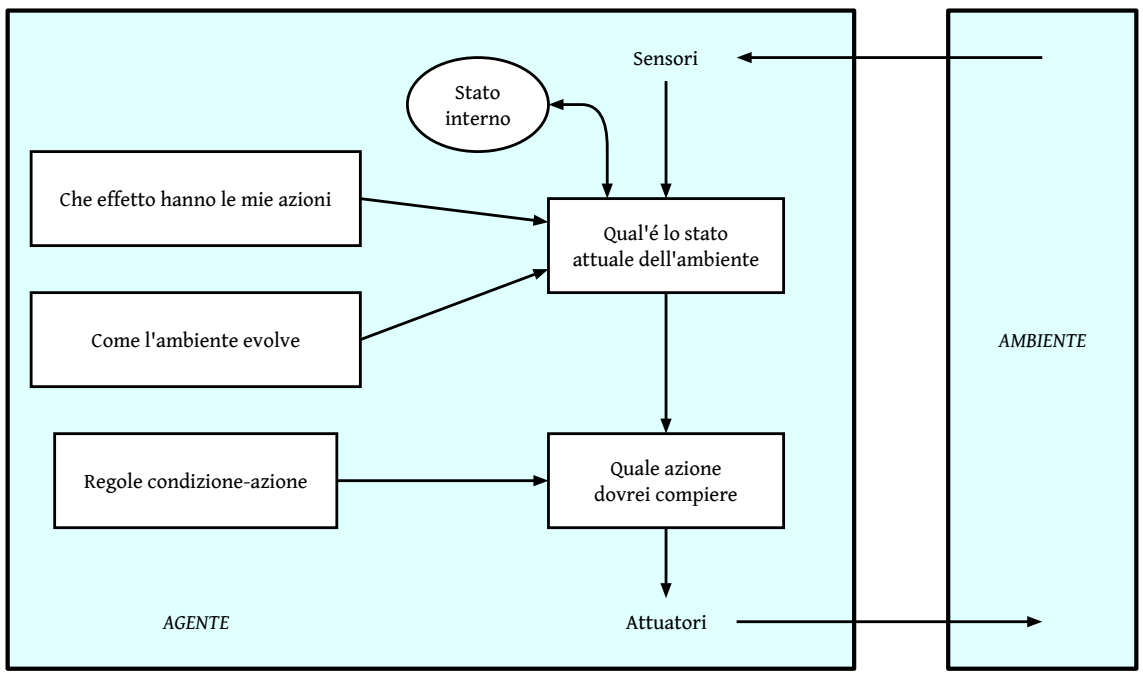

Gli agenti con riflessi semplici hanno una intelligenza limitata. Infatti, agenti di questo tipo operano correttamente solamente se l'azione da compiere che massimizza la funzione di prestazione puó essere determinata solo sulla base delle proprie percezioni, ovvero se l'ambiente é completamente accessibile. Se nella propria conoscenza a priori sono presenti errori o se l'ambiente é accessibile solo in parte, l'agente sará destinato ad operare in maniera non razionale.

Ancora piú problematica é la situazione in cui agenti con riflessi semplici entrano in loop infiniti, dato che non sono in grado di determinarli. L'unica contromisura che possono adottare é randomizzare le proprie azioni, dato che in questo modo si riduce la probabilitá che l'agente compia le stesse azioni piú volte di fila. Tuttavia, sebbene questo approccio possa mettere una pezza al problema del loop infinito in maniera semplice, in genere comporta uno spreco di risorse, e pertanto risulta difficilmente in un comportamento razionale da parte dell'agente.

### 1.1.2 Agenti con riflessi, ma basati su un modello

Il modo piú efficiente per risolvere il problema dell'avere a che fare con un agente parzialmente accessibile é tenere traccia della parte di ambiente di cui questo non ha conoscenza. Ovvero, l'agente dovrebbe avere una qualche sorta di **stato interno** che dipende dalle percezioni che questo ha captato in precedenza, di modo da avere informazioni su alcuni degli stati diversi da quello corrente. Agenti di questo tipo sono detti **agenti con riflessi ma basati su un modello**.
Aggiornare periodicamente tale stato interno richiede che l'agente possieda due forme di conoscenza. Innanzitutto, é necessario avere informazioni relative al modo in cui l'ambiente si evolve nel tempo, sia in termini di come le azioni dell'agente influenzano l'ambiente che in termini di come l'ambiente si evolve in maniera indipendente dall'agente. Questo corpo di informazioni prende il nome di **modello di transizione**. Inoltre, é necessario avere informazioni relative a come l'evoluzione dell'ambiente si riflette sulle percezioni dell'agente, nel complesso chiamate **modello sensoriale**.
Una rappresentazione schematica di un agente con riflessi ma basati su un modello é presentata in basso, dove la funzione `UPDATE-STATE` aggiorna lo stato interno dell'agente prima di restituire l'azione da compiere.

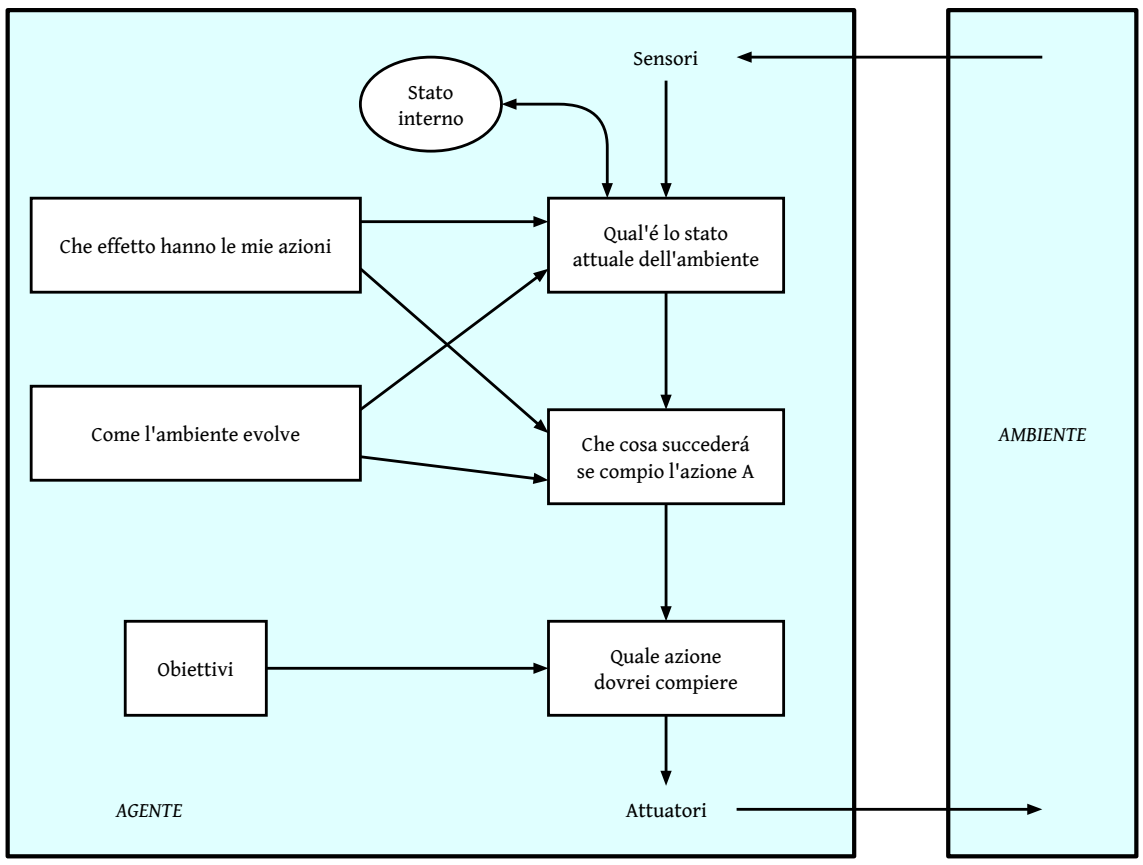
```
state <= the agent's current conception of the environment state
transition_model <= a description on how the next state depends on the current state and action
sensor_model <= a description on how the current world state is reflected in the agent's percepts
rules <= set of condition-action rules
action <= the most recent action (starts NULL)


function MODEL-BASED-REFLEX-AGENT(percept)
 state <= UPDATE-STATE(state, action, percept, transition_model, sensor_model)
 rule <= RULE-MATCH(state, rules)
 action <= rule.action
 return action
```




Si noti come difficilmente un agente con riflesso basato su un modello puó determinare con certezza lo stato attuale dell'ambiente. In genere, un agente puó limitarsi ad averne una descrizione parziale.

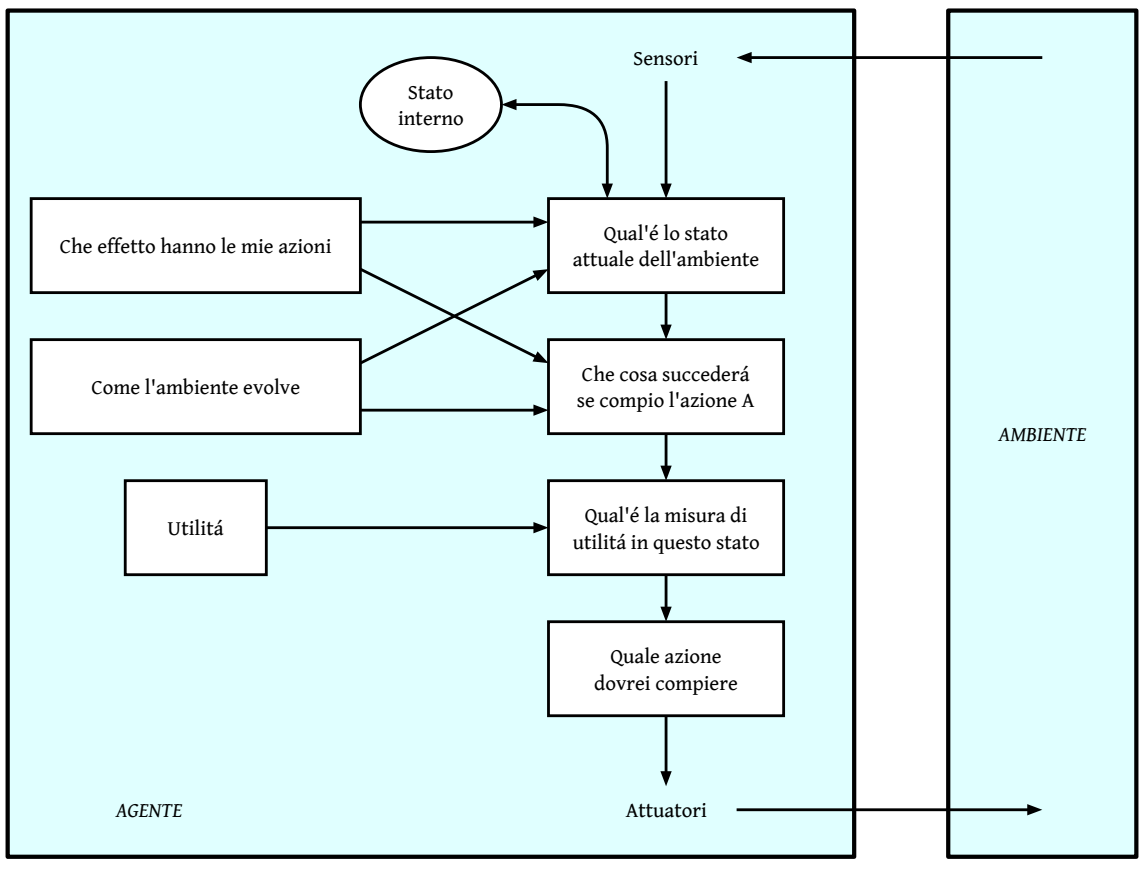
### 1.1.3 Agenti basati su un modello, ma basati su obiettivi

Vi sono situazioni in cui la scelta di quale sia l'azione migliore da compiere da parte di un agente dipenda anche da un qualche tipo di obiettivo a lungo termine. Non sempre questo obiettivo viene raggiunto nell'operare una sola azione, ma puó richiedere diverse azioni intermedie. In agenti di questo tipo, la medesima azione ed il medesimo stato interno possono risultare in azioni diverse se é diverso l'obiettivo.




### 1.1.4 Agenti basati su un modello e guidati da utilitá

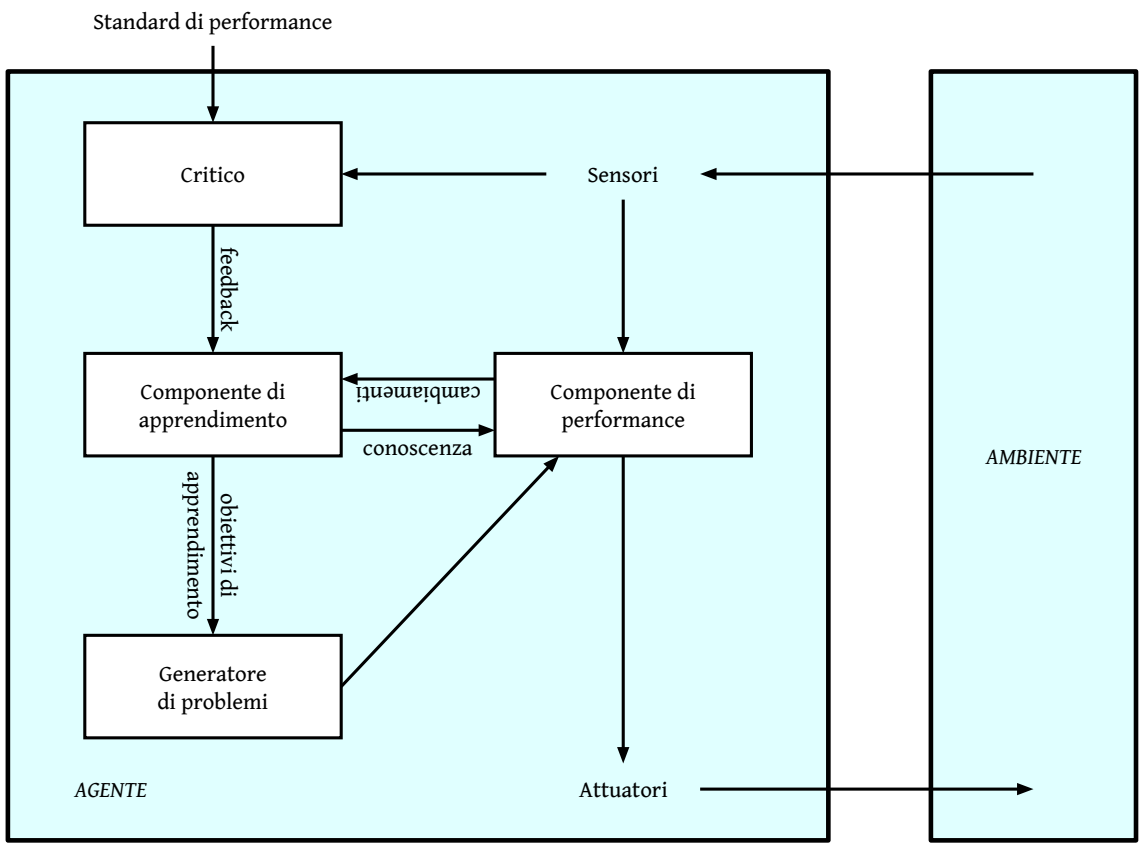
Non sempre é possibile costruire un agente razionale semplicemente spingendolo a raggiungere un obiettivo. Infatti, se tale obiettivo puó essere raggiunto tramite diverse sequenze di azioni, una potrebbe essere preferibile ad un'altra. Inoltre, un agente potrebbe dover perseguire piú obiettivi contemporaneamente fra di loro incompatibili, ovvero compiere azioni che lo "avvicinano" ad un obiettivo ma al contempo "allontanarlo" da un altro.
Un obiettivo permette di discriminare gli stati dell'ambiente esclusivamente come "favorevoli" e "sfavorevoli", senza alcuna sfumatura nel mezzo. Un migliore approccio prevede invece di introdurre una misura di **utilitá**, che influenza la scelta dell'agente nello scegliere quale azione compiere (insieme alla misura di prestazione, all'obiettivo da seguire e dal proprio stato interno).
La misura di utilitá permette all'agente di, nel dover perseguire piú obiettivi fra di loro incompatibili, scegliere l'azione che comporta il miglior compromesso nell'avanzamento di tutti loro. Inoltre, non sempre la struttura dell'ambiente garantisce che sia possibile raggiungere con assoluta certezza un obiettivo semplicemente eseguendo le azioni appropriate; anche in questo caso, la misura di utilitá permette di valutare quanto sia "conveniente" per l'agente compiere una certa azione in vista di un determinato obiettivo sulla base di quanto sia ragionevole che tale obiettivo venga effettivamente raggiunto.



### 1.1.5 Agenti che apprendono

Gli agenti piú interessanti sono indubbiamente quelli in grado di **apprendere**; tutti i tipi di agenti presentati finora possono essere costruiti come agenti che apprendono. Il notevole vantaggio che presentano é che possono operare in un ambiente del tutto sconosciuto apprendendo da questo, di modo da compiere le azioni migliori anche in situazioni dove lo stesso designer non ha modo di poter prevedere quali queste possano essere.
Un agente in grado di apprendere puó essere concettualmente suddiviso in quattro componenti:

- La **componente di apprendimento**, che si occupa di migliorare la performance dell'agente;
- La **componente di performance**, che sceglie quale azione compiere sulla base delle percezioni e dello stato di conoscenza interno. Di fatto, questa componente costituiva l'intero agente dei modelli precedenti;
- Il **critico**, che informa la componente di apprendimento di quanto l'agente si sta comportando in maniera ottimale (razionale) sulla base di uno standard di performance prestabilito. Questa componente é necessaria perché le percezioni, di per loro, non sono in grado di informare l'agente sull'ottimalitá del proprio comportamento;
- Il **generatore di problemi**, che suggerisce azioni all'agente che possono comportare nuove ed informative esperienze. Questa componente é necessaria perché se l'agente si affidasse esclusivamente alla componente di performance sceglierebbe sempre le azioni migliori sulla base della sua conoscenza attuale, che non sono necessariamente complete. Il generatore di problemi puó portare l'agente a compiere azioni che possono potenzialmente essere localmente subottimali ma che sul lungo termine possono portare a compiere azioni ancora migliori.



# Capitolo 2
# Intelligenza artificiale simbolica

## 2.1 Knowledge representation and reasoning

Gli esseri umani sono in grado di compiere azioni anche sulla base del fatto che possiedono delle **conoscenze** utilizzate per operare dei **ragionamenti** su una **rappresentazione** interna della conoscenza. Nel campo della AI questo si traduce nella costruzione di **agenti basati sulla conoscenza**.
Il componente principale di un agente basato sulla conoscenza é la **base di conoscenza**, o KB. Una KB é composta da un insieme di **fatti**, che rappresentano delle asserzioni sul mondo. Un agente basato sulla conoscenza deve essere in grado di fare **inferenze**, ovvero essere in grado di aggiungere dei nuovi fatti alla KB sulla base di quelli presenti applicando delle **regole**. Affinché questo sia possibile, é necessario che alcuni fatti siano presenti nella KB fin da subito. Questi vengono detti **assiomi**; l'unione di tutti gli assiomi prende il nome di **conoscenza pregressa** (**background knowledge**).
Sia i fatti (le asserzioni sul mondo) che le regole (le trasformazioni che aggiungono nuovi fatti alla KB sulla base di quelli presenti) vengono espressi in genere espressi in linguaggi specifici. Tali linguaggi sono detti **linguaggi di Knowledge Representation and Reasoning**, o **linguaggi KRR** (**linguaggi di rappresentazione della conoscenza**). Un linguaggio KRR deve necessariamente basarsi su una qualche formalizzazione della logica, e ci si chiede allora quale formalizzazione della logica potrebbe ben adattarsi ad essere quella utilizzata dagli agenti basati sulla conoscenza.
La logica proposizionale (logica di ordine zero) puó venire scartata subito: nonostante abbia il pregio di essere decidibile, é troppo semplicistica, dato che non supporta i quantificatori universali "per ogni" e "esiste". Un miglior candidato potrebbe allora essere la logica proposizionale (logica del primo ordine), ma anche questa presenta dei problemi:

- *Decidibilitá*. Come mostrato dai Teoremi di Incompletezza di Godel, la logica proposizionale é **indecidibile**, ovvero non tutte le formule possono essere provate vere o false all'interno della logica stessa [1]. Questo significa che un sistema di deduzione automatico, essendo limitato dall'Halting Problem, potrebbe rimanere eternamente bloccato nel computare se una data proposizione segua dalle premesse senza essere in grado di fornire una risposta;
- *Complessitá*. La logica proposizionale é estremamente espressiva, pertanto alcune inferenze possono richiedere molto tempo computazionale (per quanto finito) per essere completate;
- *Approssimazione*. Per lo stesso motivo, non tutte le proprietá della logica proposizionale sono strettamente necessarie nel campo della IA. Cercare di implementarle tutte risulterebbe in uno spreco di risorse e nella costruzione di un sistema di deduzione inefficiente.

La scelta di un formalismo logico adatto al campo delle IA sembrerebbe allora ricadere in una logica che si trovi "nel mezzo" fra la logica proposizionale e la logica predicativa.

## 2.2 Knowledge Graphs

Un **Knowledge Graph** (**KG**) é un grafo diretto ed etichettato il cui scopo é riportare e trasmettere conoscenze sul mondo reale. I nodi del grafo rappresentano delle **entitá**, ovvero degli oggetti che appartengono al mondo di interesse, mentre gli archi del grafo rappresentano delle **relazioni** che intercorrono fra queste entitá.
Con "conoscenza" si intende genericamente qualsiasi cosa sia *nota*: tale conoscenza puó essere ricavata da dal mondo che il grafo vuole modellare oppure estratta dal grafo stesso. La conoscenza puó essere composta sia da semplici asserzioni che coinvolgono due entitá ("A possiede/fa uso di/fa parte di/... B") oppure asserzioni che coinvolgono gruppi di entitá ("tutti i membri di A possiedono/fanno uso/fanno parte di/... B"). Le asserzioni semplici sono riportate come etichette degli archi del grafo: se esiste un arco fra i nodi A e B, significa che A e B sono legati dalla relazione che etichetta l'arco che li unisce.
Formalmente, un Knowledge Graph é definito a partire dalla quintupla $\langle E, L, T, P, A \rangle$:

- Un insieme $E$ di simboli, che rappresentano gli identificativi associati alle entitá;
- Un insieme $L$ di **letterali**, che rappresentano tutti i dati "grezzi" che il modello necessita di rappresentare (stringhe, numeri, eccettera);
- Un insieme $T$ di tipi;
- Un insieme $P$ di simboli di relazione;
- Un insieme $A$ di assiomi.

A loro volta, gli assiomi vengono distinti in due sottogruppi:

- I fatti, ovvero assiomi che riguardano le singole entita. Indicano:
  - Se una certa entitá appartiene ad un certo tipo, ovvero $t(e) \mid t(l)$ con $e \in E$ e $l \in L$;
  - Se due entitá sono legate da una certa relazione, ovvero $r(e_1, e_2) \mid r(e, l)$ con $e_i \in E$ e $l \in L$.

1. Piú correttamente, si dice che la logica proposizionale é **semidecidibile**, in quanto é sempre possibile dimostrare se una proposizione é vera sulla base delle premesse ma non é sempre possibile dimostrare se sia falsa.

- Gli assiomi generali, ovvero assiomi che non riguardano singole entitá ma riguardano classi. La loro espressivitá dipende dal linguaggio logico a cui il KG fa riferimento, ma in genere sono nella forma $\forall x(t_1(x) \rightarrow t_2(x))$, ovvero che specificano una relazione di ordine parziale rispetto ai tipi.

Essendo un KG un grafo, é possibile studiarne le proprietá tipiche dei grafi (simmetria, antisimmetria, transitivitá, eccetera) e metterle in relazione con il significato che hanno nel modello che questi rappresentano. É inoltre possibile *visitare* il grafo per ricavare informazioni piú elaborate di quelle riportate nei soli archi.

## 2.3 Resource Description Framework

**Resource Description Framework** (**RDF**) é un modello di dati strutturato a grafo; sebbene inizialmente concepito per il web (é infatti parte di un insieme di protocolli piú grande noto come **Semantic Web Stack**), trova uso anche come formato per la rappresentazione della conoscenza. RDF é un modello di dati pensato per descrivere risorse. Con **risorsa** si intende qualsiasi entitá a cui sia possibile associare un'identitá, che siano entitá virtuali (pagine web, siti web, file, ... ), entitá concrete (libri, persone, luoghi, ... ) o entitá astratte (specie animali, categorie, ere geologiche, ... ) [2].

### 2.3.1 Termini

Ad una risorsa RDF viene fatto riferimento attraverso un **termine**. In genere, un termine non é una semplice stringa di caratteri, dato che é facile trovarsi in una situazione in cui diversi dataset si riferiscono con lo stesso nome a risorse diverse, in particolare se tali dataset provengono dal Web da fonti diverse. In RDF esistono tre tipi di termini: **IRI**, **letterali** e **nodi blank**.
Un IRI (**International Resource Identifier**) é una stringa di caratteri Unicode che identifica univocamente una qualsiasi risorsa; se due risorse hanno lo stesso IRI, allora sono in realtá la stessa risorsa. Gli IRI sono un superset degli **URI** (**Unique Resource Identifier**), che hanno la medesima funzione ma sono limitati ai soli caratteri ASCII.
Gli URI costituiscono a loro volta un soprainsieme sia degli **URL** (**Universal Resource Locator**) sia degli **URN** (**Uniform Resource Name**). Il primo serve ad indicare la locazione di una risorsa (sul web), mentre il secondo il nome proprio della risorsa, scritto con una sintassi specifica. Pertanto, ad una risorsa é possibile riferirsi indifferentemente per locazione (URL) o per nome (URN).
Si noti come gli IRI risolvono il problema di avere a che fare con risorse diverse aventi lo stesso nome, ma non risolvono il problema inverso, ovvero dove IRI distinti si riferiscono alla stessa risorsa. RDF permette che una situazione di questo tipo si verifichi, ma in genere é preferibile risolvere questo tipo di conflitti adottando uno degli IRI che si riferiscono alla stessa risorsa a discapito degli altri.
Piú IRI provenienti dallo stesso dataset hanno molto spesso dei prefissi in comune. Inoltre, gli IRI tendono ad essere stringhe molto lunghe. Per risolvere entrambi i problemi, RDF ammette la possibilitá di assegnare delle abbreviazioni ai prefissi.
Si noti inoltre come gli IRI non possano fornire (in maniera efficiente) informazioni relative a descrizioni, date, valori numerici, ecc ... Tali dati sono riportati come letterali. In RDF, un letterale é costituito dalle seguenti tre componenti:

- Una **forma lessicale**, ovvero una stringa di caratteri Unicode;
- Un **datatype IRI** che indica il tipo di dato del letterale, definendo un dominio di possibili valori che questo puó assumere. Viene preceduto da "^^";
- Un **language tag** che indica la lingua in cui il termine viene espresso. Viene preceduto da "@"

I letterali piú semplici sono quelli composti dalla sola forma lessicale; il datatype ed il language tag sono opzionali, ma spesso utili a dare l'interpretazione corretta del letterale a cui si riferiscono. I tipi di dato definiti da RDF sono un sottoinsieme dallo standard XSD, a cui si aggiungono i tipi di dato `rdf:XML` e `rdf:XMLLiteral` propri di RDF. Questi possono essere raggruppati in quattro categorie:

- **Booleani**, ( `xsd:boolean` );
- **Numerici**, sia interi ( `xsd:decimal` , `xsd:byte` , `xsd:unsignedInt` , ecc ... ) che razionali ( `xsd:float` e `xsd:double` );
- **Temporali**, che siano istanti di tempo ( `xsd:time` , ... ), lassi di tempo ( `xsd:duration` , ... ) o una data specifica ( `xsd:gDay` , `xsd:gMonth` , `xsd:gYear` , ... );
- **Testuali**, sequenze di caratteri generiche ( `xsd:string` ) oppure conformi rispetto ad una certa sintassi ( `rdf:XML` , `rdf:XMLLiteral` , `xsd:anyURI` , ecc ... ).

Alcuni tipi di dato sono derivati da altri tipi di dato, ovvero restringono i valori ammissibili dal dato da cui derivano ad un sottoinsieme piú piccolo (e piú specifico); i tipi di dato che non derivano da altri sono detti **primitivi**. Inoltre, mentre alcuni tipi di dato (come `xsd:decimal` ) hanno una cardinalitá infinita numerabile, altri (come `xsd:unsignedLong` ) hanno un numero finito di valori ammissibili.
Se ad un letterale non é associato un tipo di dato, si assume che sia di tipo `xsd:string` ; l'unica eccezione sono i letterali che presentano un language tag, a cui viene implicitamente assegnato il tipo `rdf:langString` . Sebbene RDF ammetta la possibilitá di definire dei tipi di dato custom, non fornisce un meccanismo standard per riportare esplicitamente che tale tipo di dato derivi da un altro, o per definire un dominio di valori ammissibili.
Vi sono situazioni in cui é preferibile che una certa risorsa non venga identificata per mezzo di un IRI, ad esempio perché un'informazione é mancante oppure perché non é rilevante. RDF gestisce tali casistiche per mezzo dei **blank nodes**, che per convenzione hanno come prefisso il

2. Si noti come questo comporti che diverse risorse che in un dataset RDF sono presenti quasi con certezza debbano venire reimplementati da capo. Fortunatamente, esistono servizi che si prefiggono di "standardizzare" la definizione di diverse risorse, di modo che siano riutilizzabili. Un esempio di questi é FOAF: `xmlns.com/foaf/spec` .

carattere "_". Se una risorsa é identificata da un blank node, significa che tale risorsa esiste, ma non si ha modo o interesse di assegnarle un nome. I blank node operano come variabili esistenziali locali al loro dataset; due blank node di due dataset distinti si riferiscono a due risorse distinte.

### 2.3.2 Triple

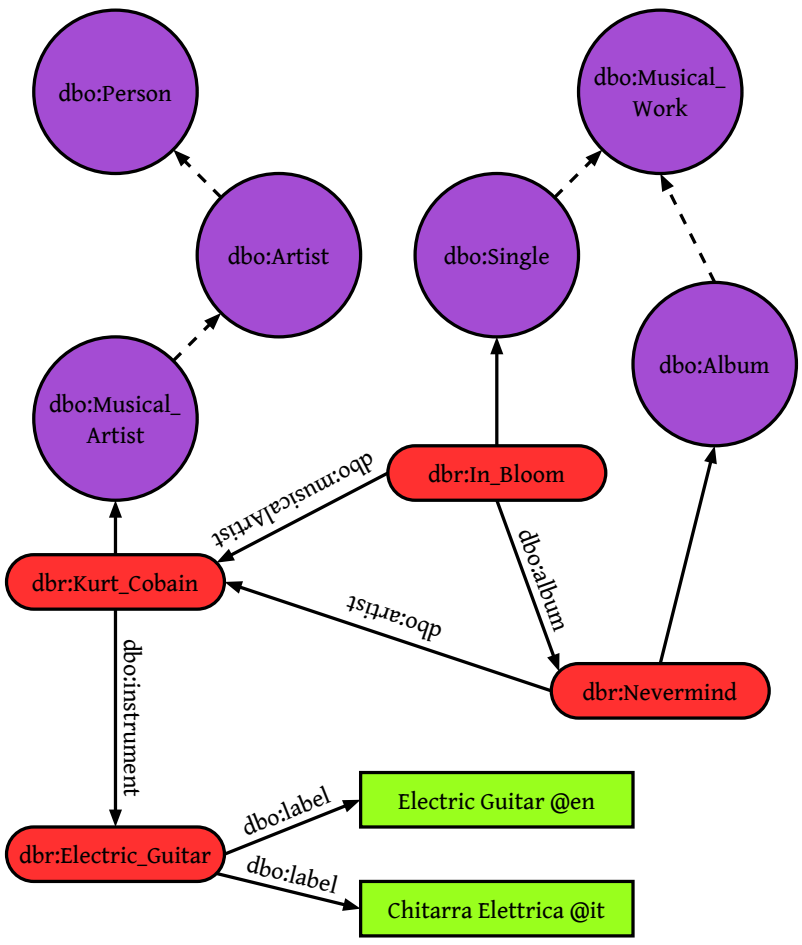
Mentre i termini RDF sono usati per identificare le risorse, le **triple** sono usati per descriverle. Una tripla RDF é nella forma soggetto-predicato-oggetto [3], dove tutti e tre gli elementi sono termini RDF. Nello specifico, il soggetto deve essere un IRI o un blank node, il predicato deve essere un IRI e l'oggetto puó essere di qualsiasi tipo di termine.

```
ex:Boston ex:hasPopulation "646000"^^xsd:integer          ex:VoynichManuscript ex:hasAuthor _:b
```

Queste restrizioni sono in linea con lo scopo che RDF si prefissa. Ai predicati deve necessariamente venire fornito un nome, dato che l'informazione "un soggetto ed un oggetto sono legati da un predicato ignoto" non é particolarmente rilevante. Inoltre, tale nome deve essere unico, perché i predicati devono poter essere univocamente identificati in qualsiasi dataset. Infine, per RDF, i letterali sono risorse di minore importanza rispetto agli IRI, pertanto sarebbe poco sensato averli come soggetto di una tripla.
Sebbene, per convenzione, il soggetto di una tripla sia la "risorsa primaria" che viene descritta dalla tripla stessa, la distinzione é del tutto arbitraria, in quanto é possibile invertire l'ordine del soggetto e dell'oggetto di una tripla per ottenerne una che descrive la stessa cosa.

### 2.3.3 Grafi

Un insieme di triple RDF costituisce un **grafo RDF**. Il nome grafo deriva dall'osservazione che ciascuna tripla RDF puó essere rappresentata in maniera equivalente come una coppia di nodi di un grafo uniti da un arco: l'etichetta di tale arco é il predicato della tripla, il soggetto é il nodo di partenza dell'arco e l'oggetto é il nodo di arrivo. Piú triple RDF danno allora vita ad un grafo diretto ed etichettato.
Il fatto che RDF sia un modello di dati strutturato a grafo lo rende molto flessibile. Infatti, per introdurre nuovi predicati in un grafo RDF é sufficiente aggiungere un arco che ha tale predicato come etichetta, cosí come per introdurre nuovi soggetti o oggetti é sufficiente aggiungere dei nodi. Similmente, due grafi diversi (che corrispondono a due dataset diversi) possono essere unificati in maniera diretta mediante l'operazione di unione sui due insiemi di triple; l'unica eccezione sono i grafi che contengono dei blank node, perché il loro significato dipende dal grafo in cui si trovano, ed é quindi necessario prendere misure aggiuntive.




3. La struttura segue quella delle lingue anglosassoni.

## 2.4 Sintassi: N-triples e Turtle

RDF fornisce solamente una semantica per le triple, ma ne non specifica una sintassi. A tal scopo, sono state definite diverse sintassi per le triple RDF, in genere fra loro intercompatibili.
Una rappresentazione testuale estremamente semplice é **N-triples**; questa prevede di riportare per intero ciascun elemento di ogni tripla, una tripla per riga, terminandole con un punto. Le tre componenti di ciascuna tripla ed il punto alla fine della tripla sono separate da uno o piú spazi. Se un elemento é un IRI, viene riportato fra parentesi angolate, mentre se é un letterale viene riportato fra doppi apici. I blank node, i language tag ed i datatype IRI vengono riportati come di consueto. Una riga che inizia con il carattere "#" viene interpretata come un commento.

```
# Le tre triple riportate di seguito sono sintatticamente valide per N-triples

<http://www.example.org/alice>                 <http://schema.org/knows>          <http://www.example.org/bob>   .
_:dave                                         <http://xmlns.com/foaf/0.1/name>   "Dave Beckett"^^xsd:string     .
<http://www.w3.org/2001/sw/RDFCore/ntriples/>  <http://purl.org/dc/terms/title>   "N-Triples"@en-US              .
```

N-triples é tanto intuitivo quanto poco leggibile, perché gli IRI sono sempre riportati per intero, e gli IRI tendono ad essere molto lunghi. Una rappresentazione testuale leggermente piú complessa é **Turtle**, che eredita la sintassi di N-triples estendendola ed aggiungendovi delle abbreviazioni per migliorarne la leggibilitá.
Ai prefissi puó essere associata una parola chiave mediante la direttiva `@prefix:` . Se due triple consecutive hanno in comune il soggetto, é possibile terminare la prima con un punto e virgola e non riportare il soggetto nella seconda. Se due triple consecutive hanno in comune sia il soggetto che il predicato, é possibile terminare la prima con una virgola e non riportare soggetto e predicato nella seconda.

```
# Turtle permette di definire triple RDF molto piú facilmente rispetto a N-triples

@prefix dbr:   <http://dbpedia.org/resource/>   .
@prefix dbo:   <http://dbpedia.org/ontology/>   .

dbr:Kurt_Cobain   dbo:instrument      dbr:Electric_guitar .
dbr:In_Bloom      dbo:musicalArtist   dbr:Kurt_Cobain     ;
                  dbo:album           dbr:Nevermind       .
dbr:Nevermind     dbo:artist          dbr:Kurt_Cobain     .
```

## 2.5 SPARQL Protocol And RDF Query Language

Avendo a disposizione un grafo RDF, ci si chiede come sia possibile formulare domande sullo stesso, ad esempio determinare se esiste una tripla in cui figura un certo IRI. Dato che porre questo tipo di domande in linguaggio naturale è di difficile interpretazione per una macchina, queste vanno riformulate in un **linguaggio di query**. In particolare, un linguaggio di query appositamente pensato per estrarre informazioni da grafi RDF è **SPARQL (SPARQL Protocol And RDF Query Language)** [4].
Una query SPARQL è costituita dalle seguenti sei componenti, non tutte strettamente obbligatorie:

1. *Dichiarazione dei prefissi*. Similmente a Turtle, è possibile dichiarare dei prefissi mediante la direttiva `PREFIX` , seguita dal nome scelto per il prefisso e dall'URI a cui il prefisso è associato;
2. *Costruzione del dataset*. mediante la direttiva `FROM` è possibile specificare su quale/i grafo/i si vuole operare la query. Se vengono specificati più grafi, la query verrà operata sulla loro unione;
3. *Tipo di query*. SPARQL supporta quattro tipi di query:

   - `SELECT` , che restituisce il risultato della query sotto forma di tabella. Questa supporta l'eliminazione delle soluzioni duplicate per mezzo delle direttive `REDUCED` (possono essere rimosse) e `DISTINCT` (devono essere rimosse);
   - `ASK` , che restituisce true se la query ha un risultato non nullo e false altrimenti;
   - `CONSTRUCT` , che restituisce il risultato della query sotto forma di (sotto) grafo;
   - `DESCRIBE` , che restituisce il risultato della query sotto forma di grafo che descrive termini e soluzioni.
4. *Pattern*. La direttiva `WHERE` specifica il pattern che discrimina un elemento del grafo che è parte della soluzione da uno che non lo è;
5. *Aggregazione*. Le direttive `GROUP BY` e `HAVING` , analoghe alle direttive omonime di SQL permettono di raggruppare o di filtrare gli elementi della soluzione secondo specifiche regole;

4. Sia il nome che la struttura delle query di SPARQL hanno molto in comune con **SQL**, che è invece un linguaggio di query per database relazionali.

6. *Modificatori della soluzione.* Alcune direttive permettono di modificare gli elementi della soluzione disponendoli secondo un certo ordine ( `ORDER BY` ) oppure restituendone solo una parte.

La nozione più importante nel linguaggio SPARQL è il **pattern di tripla RDF**. Questa è di fatto analoga ad una tripla RDF, ma oltre ad ammettere IRI, letterali e nodi blank può contenere anche **variabili di query**, che ha il carattere "?" come prefisso. Tale pattern viene riportato nel quarto campo di una query SPARQL dopo la direttiva `WHERE` .
I risultati delle query SPARQL sono ottenuti mediante **pattern matching**: quando un client SPARQL invia una query ad un server SPARQL, questo analizza il pattern di tripla contenuto nella query e restituisce tutte le triple dei grafi specificati nella query che coincidono con tale pattern. Nello specifico, gli IRI ed i letterali hanno un match solamente con, rispettivamente, un IRI ed un letterale a loro identico, mentre i blank node e le variabili di query hanno un match con qualsiasi termine. La differenza fra i due sta nel fatto che i termini che hanno un match con una variabile di query possono venire restituiti come parte della soluzione, mentre quelli che hanno un match con un blank node non possono.