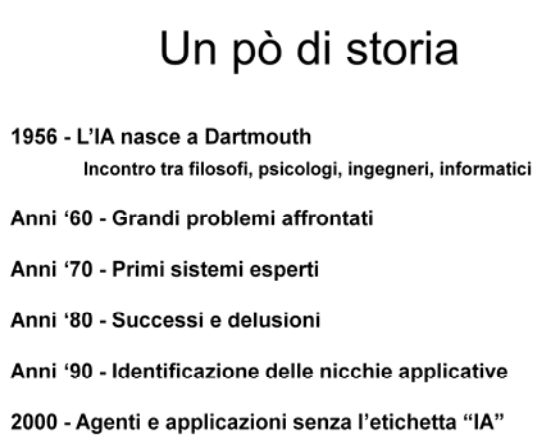
Note del Corso di
Ingegneria della Conoscenza

Parte1: Knowledge representation

Andrea Bonarini

Dipartimento di Elettronica e Informazione
Politecnico di Milano
Via Ponzio 35/5
20133 Milano

http://www.elet.polimi.it/people/bonarini
bonarini@elet.polimi.it
02 2399 3525



Iniziamo il nostro percorso con una breve storia dello sviluppo dell'Intelligenza Artificiale, che ci permette di capire anche quali sono i problemi affrontati e risolti, e quelli ancora aperti.

Nel **1956** si tiene al Dartmouth College un gruppo di lavoro a cui partecipano psicologi, filosofi, ingegneri, informatici, Viene fondata una nuova disciplina scientifica: l'Intelligenza Artificiale. Prevalente in quei tempi era l'obiettivo di costruire una macchina che potesse simulare o emulare il comportamento umano intelligente. Distinguiamo tra ***simulazione***, il cui scopo è riprodurre i meccanismi di funzionamento di un sistema, ed ***emulazione***, il cui scopo è di ottenere un sistema che fornisce le stesse prestazioni di un'altro, senza necessariamente riprodurre le modalità con cui questo opera.

Negli **anni '60** vengono affrontati grandi temi, in vari settori dell'intelligenza, dai giochi, alla risoluzione di equazioni, alla pianificazione, al supporto alle decisioni, fino a formulare piani di ricerca per avere un General Problem Solver.

Negli **anni '70** si hanno i primi sistemi esperti nei campi della diagnosi, dell'analisi di dati, della risoluzione di equazioni simboliche. Tutti erano in grado di affrontare con successo problemi molto limitati, e, quasi sempre, in un ambiente diverso da quello in cui doveva operare il sistema (laboratorio).

Negli **anni '80** si hanno i primi successi industriali, ma, allo stesso tempo, anche le prime grandi delusioni, dovute soprattutto al desiderio di risolvere problemi più grossi di quanto la tecnologia del momento o lo strumento stesso fosse in grado di risolvere.

Negli **anni '90** sono ormai identificate delle chiare nicchie applicative, ed in queste la tecnologia dei sistemi esperti si assesta e si sviluppa, arricchendo sempre di più le sue caratteristiche ingegneristiche.

Oggi si parla di applicazioni AI in diversi settori e si cominciano ad avere veri e propri agenti autonomi.

Cos’è un modello?

1.
2.
3.
4.

Questo spazio serve per annotare le definizioni che emergono a lezione

A lezione, in questo momento viene chiesta una definizione del termine ***modello***. Questa attività ha lo scopo di fare emergere l’opinione che ognuno di noi ha circa questo termine.

Questo serve da un lato a dare un punto di partenza che renderà poi possibile una valutazione dell’evoluzione nell’apprendimento del concetto. Dall’altro lato mette in evidenza alcune caratteristiche dei modelli, che emergono dall’analisi delle definizioni date.

Cos'è un modello

Rappresentazione

- Utile a uno scopo
- Necessariamente diversa dal rapresentato

Il modello è una rappresentazione di qualcos'altro, che sia utile a uno scopo, e quindi sia stata progettata o scelta per raggiungere lo scopo.

Ad esempio, un modello matematico ha lo scopo di rappresentare sinteticamente una relazione tra variabili in modo che dato il valore di alcune di esse si possa ricavare il valore di altre. Con I modelli matematici di due funzioni si può facilmente calcolare in maniera esatta la loro intersezione. Il grafico delle stesse funzioni permette immediatamente di percepirne delle proprietà, ma permette un calcolo più approssimato delle intersezioni.

Una rappresentazione tabellare delle stesse funzioni è approssimata, ma permette di risparmiare il computo dei valori rappresentati…

La rappresentazione è necessariamente diversa dal rappresentato, ne cattura solo gli aspetti rilevanti, scelti dal modellista. Questa differenza può portare a problemi qualora si voglia usare il modello per scopi diversi da quelli che permette la sua qualità. IN particolare, per i modelli che vedremo, la differenza tra la realtà del mondo e la sua rappresentazione porterà problemi di incertezza e approssimazione che andranno risolti rappresentando esplicitamente queste stesse qualità

Cos’è la Conoscenza?

1.

2.

3.

4.

Questo spazio serve per annotare le definizioni che emergono a lezione

A lezione, in questo momento viene chiesta una definizione del termine ***conoscenza***. Questa attività ha lo scopo di fare emergere l’opinione che ognuno di noi ha circa questo termine.

Questo serve da un lato a dare un punto di partenza che renderà poi possibile una valutazione dell’evoluzione nell’apprendimento del concetto. Dall’altro lato mette in evidenza alcune caratteristiche della conoscenza, che emergono dall’analisi delle definizioni date.

Cos'è la Conoscenza

- Strutture Dati
- Relazioni
- Possibilità di ragionamento

- Generazione di nuovi dati

La conoscenza è quindi costituita da dati strutturati (informazioni) collegati da relazioni, sui quali è possibile svolgere delle attività di ragionamento che permettono di ricavare ulteriore conoscenza.

Le ***strutture dati*** permettono di dare una descrizione organica delle informazioni. DI solito assumono una forma ben definita a priori che ha lo scopo di guidare l'identificazione e la definizione delle caratteristiche rilevanti della conoscenza da rappresentare.

Le ***relazioni*** tra i dati permettono di creare quei legami tra le informazioni rappresentate con le strutture dati, che serviranno per ragionare sui dati e per ricavare nuova conoscenza.

Perchè la conoscenza sia tale, deve essere possibile effettuare del ***ragionamento***, cioè utilizzare la conoscenza a disposizione e rappresentata per ricavare in forma esplicita nuova conoscenza.

Cos'è l'Ingegneria della Conoscenza

Insieme di tecniche
(metodologia)
per realizzare
sistemi basati sulla conoscenza
(KBS - Knowledge-based Systems)

L'***Ingegneria della Conoscenza*** è una metodologia che supporta lo sviluppo di sistemi informatici basati sulla conoscenza.

L'importanza di una metodologia risulta evidente per permettere uno sviluppo efficace ed efficiente di sistemi basati sulla conoscenza (KBS - Knowledge-Based Systems), limitando la necessità di abilità speciali dei singoli operatori per ottenere il successo nello sviluppo di tali sistemi.

In particolare gli strumenti a cui accenneremo nel seguito permettono di impostare un progetto di KBS, senza dover riscoprire ogni volta soluzioni a problemi già affrontati da altri in passato.

Classi di applicazioni

- Problem Solving
- Games
- Diagnosi
- Pianificazione
- Controllo
- Classificazione
- Apprendimento Automatico
- Progettazione Automatica
- Perception
- Interazione Uomo-macchina
- …

Le classi di applicazioni per sistemi esperti sono molteplici. ne trattiamo qui solo alcune, quelle più diffuse o storicamente significative.

Con il termine ***Problem Solving*** si intendeva nei primi anni dell'IA una generica attività di risoluzione dei problemi: il termine non identifica una vera e propria classe di applicazioni.

Tra i primi KBS troviamo m olti sistemi in grado di giocare con vari esiti diversi ***giochi***, quali: scacchi, dama, ecc.

Le applicazioni di ***diagnosi*** sono tutt'oggi tra le più diffuse. Si tratta di identificare le cause di malfunzionamenti a partire dalla loro osservazione. La maggior parte dei KBS per la diagnosi si occupano con successo di applicazioni industriali, mentre alcuni dei primi KBS erano utilizzati per diagnosi mediche, anche se non hanno mai incontrato la fiducia dell'utenza.

Con il termine ***pianificazione*** interdiamo la formulazione di un piano per l'utilizzo di risorse, che queste siano risorse per la produzione industriale o, magari, azioni per un robot. Anche questo settore è tutt'oggi attivo, grazie alla efficienza dei KBS rispetto a sistemi di ricerca e ottimizzazione non basatisulla conoscenza

Il ***controllo intelligente*** ha incontrato nell'ultimo decennio un crescente interesse per la sua flessibilità, facilità di implementazione e per i risultati che può ottenere su certe classi di problemi difficilmente affrontabili con tecniche classiche. Rientra in questa categoria anche il controllo fuzzy, basato su regole di inferenza che possono rappresentare concetti approssimati in modo preciso.

Classificazione significa identificare la classe (categoria) a cui appartiene un certo esemplare descritto in maniera anche parziale e incompleta. è una tecnica usata, ad es., nell'interpretazione di dati.

L'***apprendimento automatico*** è una vasta classe di applicazioni il cui scopo è l'accrescimento della conoscenza presente in un KBS tramite l'interazione diretta con il mondo esterno. Molti sforzi sono attualmente fatti in questa direzione, nella speranza di ridurre i costi dello sviluppo di KBS, migliorandone la qualità

La ***progettazione automatica*** (o Intelligent CAD) prevede che un KBS articolato supporti il progettista nelle diverse fasi dello sviluppo di un progetto complesso. è stata utilizzata con successo per progettare aerei, navette spaziali e costruzioni edili.

La ***percezione*** attraverso sensori diversi dalla semplice tastiera è ciò che potrebbe permettere ad un calcolatore di interagire autonomamente con il mondo reale. Anche in questo settore, data la complessità dei dati a disposizione e la necessità di interpretarli in modo intelligente, i KBS giocano un ruolo rilevante.

Il settore dell'***'nterazione uomo-macchina*** è stato uno dei primi affrontati con tecniche di IA. Si tratta di permettere ad un calcolatore di capire di cosa ha bisogno il suo utente e di interagire con esso in modo opportuno, efficace e piacevole per l'utente stesso.

Cosa faremo?

Individuazione di:

- caratteristiche della conoscenza
- modalità di rappresentazione basate su espressioni simboliche
- obiettivi di sistemi basati sulla conoscenza

Identificare le ***caratteristiche*** della conoscenza permette di strutturare il processo di acquisizione della conoscenza e di utilizzare metodologie specifiche per l'acquisizione e la rappresentazione di questi aspetti caratteristici.

Ogni aspetto della conoscenza può essere rappresentato con diverse ***modalità*** più o meno valide in generale o per casi particolari. In questo corso vedremo alcune modalità basate sull'utilizzo di espressioni simboliche, cioè sull'uso di simboli simili al quelli usati nel linguaggio naturale per rappresentare concetti.

Identificando gli obiettivi che un KBS deve perseguire si capisce quali degli aspetti della conoscenza coinvolta sono significativi e come si possono rappresentare per risolvere efficacemente il problema affrontato.

Nel seguito della dispensa affronteremo i seguenti temi:

- caratteristiche della conoscenza
- modalità di rappresentazione basate su espressioni simboliche
- obiettivi di sistemi basati sulla conoscenza

... ed ora un gioco ...

Cos'è un G00324?

A questo punto viene proposto un gioco allo scopo di far capire quali possono essere le caratteristiche della conoscenza che vengono considerate, ad esempio, per un sistema esperto di classificazione e riconoscimento. L'aula gioca il ruolo del "sistema esperto" e cerca di riconoscere l'oggetto proposto dal docente ponendo domande a cui vengono date risposte semplici e non articolate.

Normalmente emergono alcune delle caratteristiche della conoscenza: concetti di classe ed esemplare, strutturazione in gerarchie, esistenza di attributi con valori multipli, ambigui, incerti. Emergono anche diverse strutture inferenziali.

Questo è solo uno dei numerosi task cui un KBS può essere dedicato. Inoltre la conoscenza usata per far domande (e capire le risposte) è molto di più di quella normalmente presente in un KBS.

SI riporta di seguito una tipica sequenza di domande e risposte. In italico sono riportate le caratteristiche della conoscenza che sono state considerate.

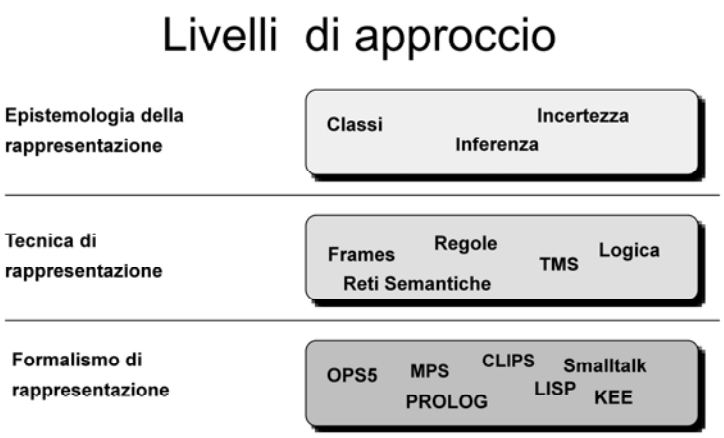
1. è una cosa, un vegetale o un animale?
   animale *(classe)*
2. Che colore ha?
   rosa, nero e bianco*(attributo a valore multiplo)*
3. Quante zampe ha?
   2 *(attributo a valore numerico)*
4. Quanto pesa?
   circa 20 Kg *(attributo a valore approssimato)*
5. Vola?
   no *(attributo a valore in contrasto con le aspettative)*
6. è un uccello?
   si *(specificazione di classe)*
7. è uno struzzo?
   si *(specificazione di classe)*
8. Ma uno struzzo pesa più di 20kg!
   questo no, è un piccolo! *(conoscenza specifica, esemplare)*

Caratteristiche della conoscenza

- Classi
- Esemplari
- Proprietà
- Relazioni e gerarchie
- Inferenza
- Verità e incertezza

Quelle che abbiamo citato sopra sono solo alcune delle caratteristiche più importanti della conoscenza.

Verranno affrontate nel seguito, identificando possibili problemi nella loro acquisizione e modalità per affrontare tali problemi.



Sopra sono riportati i livelli ai quali si affronta il processo di acquisizione, formalizzazione e rappresentazione della conoscenza.

Per ogni livello sono riportati alcuni esempi dei concetti esaminati a quel livello.

Il processo di definizione della conoscenza per un sistema basato sulla conoscenza segue un cammino che va dal livello più altoa quello più specifico

A livello ***epistemologico*** si cerca di identificare ***cosa*** occorre rappresentare, il significato della conoscenza che occorre per l'applicazione da affrontare, e, in particolare, quali sono le sue caratteristiche. In altri termini si cerca di identificare la struttura della conoscenza, la suddivisione in classi ed esemplari, quali sono i processi inferenziali coinvolti, quali sono le qualità rilevanti della conoscenza, come ad esempio, l'incertezza ad essa collegata. Tutte queste caratteristiche dipendono non solo dalla conoscenza stessa, ma anche dall'uso che se ne intende fare per l'applicazione specifica.

A livello della ***tecnica di rappresentazione*** occorre decidere ***come*** rappresentare il modello concettuale ottenuto lavorando a livello epistemologico. Occorre qui scegliere, quali tra gli approcci possibili è più adeguato a rappresentare formalmente la conoscenza a disposizione.

Infine si giunge alla scelta del ***formalismo di rappresentazione***, cioè di una specifica implementazione di una tecnicadi rappresentazione, tenendo conto di tutte le sue caratteristiche tecniche e di opportunità per il progetto. è solo a questo punto che si comincia a programmare il sistema basato sulla conoscenza.

Terminologia

–Entità

generico elemento del mondo reale che vogliamo rappresentare
(Fred, Struzzo, Camminare, La Mangiata Del 3 aprile 2003, KR)

–Unit

struttura dati concettuale che aggrega informazioni su una entità
(#Fred, #Struzzo, #Camminare, #LaMangiataDel3Aprile2003, #KR)

–Slot

Struttura dati identificata da un nome che contiene dei valori di proprietà
($Colore, $DeponeUova, $StartingTime)

Introduciamo ora la terminologia necessaria per stabilire un terreno comune per intendersi. Le definizioni dei tre termini che useremo in questa fase sono date sopra.

Si noti che con il temine ***entità*** denotiamo qualcosa di cui possiamo parlare, che possiamo identificare e descrivere. Con ***unit*** denotiamo una rappresentazione dell'entità in un certo linguaggio formale. Si è voluto dare al termine (in inglese) e a nomi che denotano le entità (singole parole precedute da #) una forma che permettano di distinguerle chiaramente dal termine "entità" (in italiano) e dalla modalità di riferimento per esse usate (frasi in linguaggio naturale). Questo per distingure chiaramente le entità, ciò che vogliamo descrivere, dalle unit, il modo che utilizziao per descriverle.

Sulla stessa linea è la scelta fatta per il termine ***slot***, che è usato per indicare il modo di rappresentare proprietà delle entità. Si noti che lo slot è costituito da un nome (con identificatore preceduto dal simbolo $) e da un valore della proprietà.

I termini qui introdotti, non sono universalmente riconosciuti, ma identificano alcuni dei concetti di base utili per la rappresentazione della conoscenza. Nel seguito introdurremo un linguaggio semi formale per la definizione di elemeti di conoscenza, che permette di rappresentare bene gli aspetti descrittivi della conoscenza, detti ***aspetti dichiarativi***

Il concetto di "classe"

Una "classe" è l'insieme di tutte le entità che hanno caratteristiche comuni.

Es.: la classe degli struzzi, la classe degli Eventi, ...

Nella definizione di classe si pone l'accento sul fatto che si tratta di un ***insieme*** di entità. Come tutti gli insiemi, viene utilizzato per discriminare gli elementi ad essa appartenenti rispetto al resto degli elementi che possono essere presi in considerazione.

Per poter effettuare questa discriminazione è essenziale che all'insieme sia associata una modalità che permetta di valutare l'appartenenza di un elemento generico all'insieme stesso. Questa modalità è generalmente chiamata funzione d'appartenenza, in quanto è pensata come una funzione che, applicata ad un elemento generico del'universo del discorso (nel nostro caso un'entità), fornisce un grado di appartenenza di tale elemento all'insieme (nel nostro caso una classe).

Quando si parla di classi, la funzione d'appartenenza è funzione delle proprietà della classe. In altri termini: tanto più un'entità ha proprietà coerenti con quelle che descrivono la classe, quanto più tale entità appartiene alla classe stessa.

A cosa serve una classe ?

Definisce le **proprietà comuni** agli elementi dell'insieme.

Es.: Tutti gli struzzi sono uccelli, non volano, sono alti mediamente circa 1,50 metri,...

E' alla base del processo inferenziale di **classificazione.**

Es.: Se di Fred so che è un uccello alto 1,50 metri, e che non vola, posso assumere che sia uno struzzo

Permette di fare assunzioni per default.

Es.: Se di Fred so solo che è uno struzzo, posso assumere tutte le sue proprietà comuni a tutti gli struzzi.

Il fatto che una classe permetta di definire delle ***proprietà comuni*** ad un insieme di elementi è di importanza fondamentare per un'efficace gestione dela conoscenza. In altri termini si riescono così a raggruppare diverse entità in un'unica entità (classe) e a ragionare in modo efficiente ed in termini generali sulla classe.

Tra le altre cose, il concetto di classe permette di effettuare un processo inferenziale detto appunto di ***classificazione***, cioè di ricondurre un'entità ad un certo insieme. Questo permette di aumentare la conoscenza a disposizione, in quanto, nel momento in cui si sa che un'entità appartiene ad una certa classe, si può assumere (per ***default***, cioè finché non viene provato il contrario) che quell'entità possegga tutte le proprietà che descrivono la classe, anche se, in realtà, si conoscono direttamente solo alcune di esse.

Come rappresentiamo una classe?

Una classe è una entità

```
Unit Struzzi
    $IsA (#Collection)
    $GeneralizedBy  (#Uccelli)
    $SpecializedBy  (#StruzziAfricani)
    $Instances (#Fred #Mary)
    $InstProperties (($Height 1.50)($Flies False))
```

Dato che di una classe possiamo parlare, che siamo in grado di descriverla, anche una classe è un'entità.

Pertanto la rappresentiamo con una unit.

Introduciamo qui un semplice ***linguaggio di rappresentazione della conoscenza*** che vuole dare un aspetto formale allarappresentazione di unit. Si tratta di un esempio di un possibile linguaggio, e verrà utilizzato come linguaggio di riferimento in questa trattazione.

Dopo la parola chiave **Unit** che identifica il fatto che stiamo rappresentando una unit, abbiamo il nome della unit stessa (in questo caso, **Struzzi**).

Di seguito abbiamo degli **slot** che descrivono l'entità associata a tale nome. Il nome degli slot inizia con il carattere $. Il contenuto degli slot è indicato generalmente come una lista (uno o più termini racchiusi tra parentesi). I termini che descrivono la proprietà rappresentata dallo slot possono essere riferimenti ad altre unit, oppure valori numerici o booleani, oppure stringhe di caratteri racchiuse tra ", che danno una descrizione in linguaggio naturale del valore della proprietà.

Per quanto riguarda l'uso di unit per la descrizione di una classe, abbiamo un certo numero di slot standard.

Lo slot **$Is_A** indica il fatto che la Unit rappresenta una collezione di elementi (un insieme).

$GeneralizedBy fornisce la lista di Unit che descrivono classi che generalizzano quella in oggetto (che contengono, cioè tutti i suoi elementi, in quanto sono descritte da un sottoinsieme delle proprietà della classe corrente).

Analogamente, **$SpecializedBy** è associato alla lista delle classi che specializzano la classe corrente (che contengono cioè solo alcuni dei suoi elementi, caratterizzati da un maggior n umero di proprietà).

Segue la lista degli esemplari della classe in oggetto (**$Instances**)

Segue quindi la lista delle proprietà comuni agli elementi della classe (**$InstProperties**). Questa lista contiene liste composte da un nome di slot, ed un eventuale valore di default per quello slot. Al posto del e di default può andare una lista

(type <riferimento_a_classe>)

che indica che i valori dello slot sono tipizzati, cioè appartengono alla classe indicata.

Problemi relativi alle classi

- Classi e prototipi:
  - quali *proprietà* sono sicuramente comuni a tutti gli esemplari di una classe?
  - default values
- Attribuzione di esemplari ad una classe:
  - quando e come è possibile?

Solo alcune delle proprietà che normalmente vengono usate per descrivere una classe sono realmente comuni a tutti gli esemplari ad essa afferenti. Si pensi, ad esempio, alla classe degli uccelli: sembra naturale ed utile definire come proprietà comune ai suoi esemplari il fatto che volino. Tuttavia esistono diversi uccelli che non volano.

In realtà, dal punto di vista formale, possiamo distinguere due tipi di proprietà:

- proprietà caratteristiche, che devono cioè essere necessariamente possedute da tutti gli esemplari di una classe;
- proprietà tipiche, che cioé descrivono la maggor parte degli elementi della classe, ma non necessariamente tutti.

Le uniche proprietà caratteristiche sono quelle definite per convenzione (ad esempio le proprietà che permettono agli zoologi di identificare classi e specie animali).

La maggior parte delle proprietà utilizzate nella pratica della rappresentazione della conoscenza sono invece proprietà tipiche, utili per effettuare diversi processi inferenziali che non sarebbero possibili usando le poche proprietà caratteristiche di solito presenti nelle applicazioni. Spesso, quindi, non si fa distinzione tra i due tipi di proprietà.

Quando invece fosse necessario distinguere, una possibilità è quella di lasciare le sole proprietà caratteristiche nella definizione della classe, ed associare alla classe uno o più prototipi, cioè la descrizione di uno o più esemplari tipici della classe. Ad esempio, un prototipo per la classe degli uccelli potrebbe essere descritto con la proprietà ($FliesP True)

A questi discorsi è correlato il problema relativo alla classificazione di esemplari. Nella pratica applicativa, raramente abbiamo a disposizione tutte le informazioni necessarie e sufficienti per determinare in modo univoco l’appartenenwza di un certo esemplare ad una classe. In tal caso, occorre sviluppare dele tecniche che permettano di qualificare l’appartenenza di un esemplare ad una classe.

Il concetto di "esemplare"

Un esemplare è una **singola entità**.

Es.: lo struzzo Fred,
l'azione di mangiare un panino il 21 aprile 2003 alle 13,
l'insieme delle sedie, ...

Un esemplare è un'entità singola, descritto da sue proprietà caratteristiche.

Di solito gli esemplari sono messi in relazione con una o più classi.

Il fatto che un esemplare possa ***appartenere a più classi*** è possibile solo quando le classificazioni sono fatte secondo criteri diversi. Ad esempio, uno struzzo può essere classificato come **uccello**, ma anche come **corridore,** o come **animale del deserto**. Le due classi nascono da criteri di classificazione diversi: la prima nasce da una classificazione "zoologica", mentre le altre seconda nascono da una classificazione basata sulle caratteristiche degli esemplari (nel nostro caso caratteristiche di locomozione e di luogo di presenza).

A cosa serve un esemplare?

Raggruppa tutte le **caratteristiche specifiche** di una singola entità

È collegato a una classe.

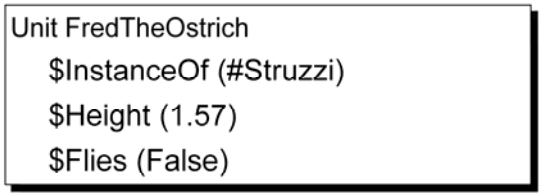
Il concetto di ***esemplare*** serve per poter descrivere singole entità con le loro caratteristiche specifiche.

Le proprietà che descrivono un esemplare possono essere le stesse che si trovano nella sua classe od in un prototipo, oppure possono essre ulteriori proprietà caratteristiche dell'esemplare specifico.

Ad esempio di Frida, possiamo dire che è un'anatra (e quindi, è un uccello, vola, ecc.) che abita in un preciso nido, che è alta esattamente 37 cm., che in un certo istante è in un certo posto definito, che è ammalata (e quindi, magari, per ora non vola), ecc.

Come rappresentiamo un esemplare?

Un esemplare è un'entità



Un esemplare è un'entità: può dunque essere rappresentato con una unit.

Slot caratteristico di una unit che descrive un esemplare è $InstanceOf, che fornisce un riferimento alle classi di cui l'esemplare è parte.

La maggior parte degli altri slot sono solitamente quelli indicati nello slot $InstProperties delle classi di appartenenza.

Problemi relativi agli esemplari

- Identificazione degli esemplari:
  il nome è sufficiente?
- Numero degli esemplari:
  ha senso avere un grande numero di esemplari della stessa classe?

Il primo problema è relatvo alla presenza di ***più esemplari distinti dal solo nome***, ma descritti tutti nello stesso modo. Questa situazione è caratteristica in carenza di informazione, quando cioè la conoscenza che permetterebbe di distinguere diversi esemplari non è disponibile. Quando si verifica una situazione di questo genere, il progettista dovrebbe chiedersi se effettivamente ha senso avere diversi semplari, o, dualmente, perchè ritiene che debbano esserci diversi esemplari distinti dal solo nome. Così operando s riesce solitamente a far emergere caratteristiche che permettano di rappresentare conoscenza che a sua volta permetta di distinguere i due esemplari.

Il secondo problema citato si riferisce al ***numero di esemplari***. Nelle applicazioni, avere un alto numero di esemplari per una certa classe può essere indice di carenza di conoscenza che permetta di specificare meglio la classe. In questo caso occorre chiedersi se non sia possibile o conveniente identificare ulteriori elementi comuni ad un sottoinsieme degli elementi appartenenti alla classe, in modo da partizionare ulteriormente la classe in sottoclassi.

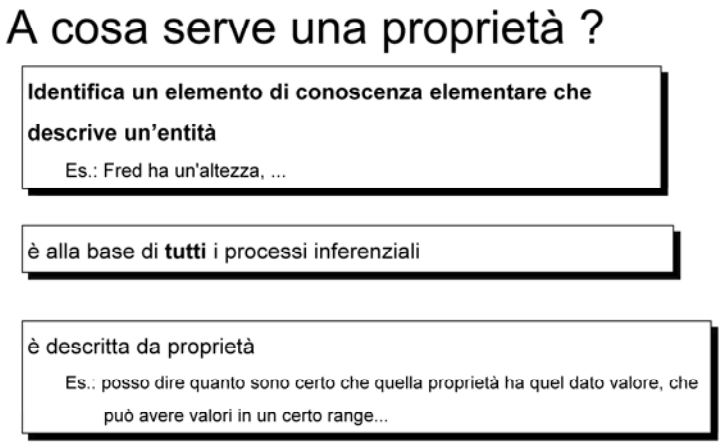
Il concetto di “proprietà“

Una proprietà è un elemento di descrizione per un’entità

Es.: uno struzzo ha la proprietà altezza,

la classe degli struzzi ha la proprietà is-a, ...

Le proprietà permettono di descrivere le entità.



Tutti i processi inferenziali lavorano su descrizioni di entità date in termini di proprietà, quindi il concetto di proprietà è alla base di tutti i processi inferenziali.

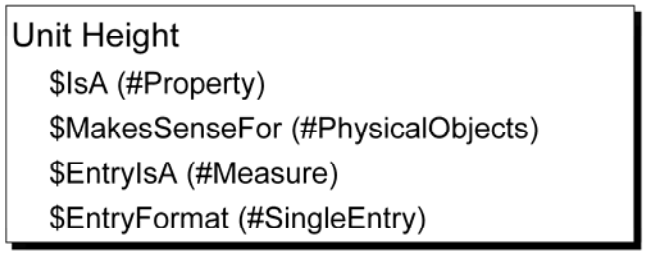
Occorre notare che, trattanandosi di un concetto di cui si può parlare, la proprietà è a sua volta un'entità, descrivibile da proprietà per essa caratteristiche. Ad esempio il tipo di valori tipoco per una proprietà, il livello di affidabilità che abbiamo circa il fatto che un esemplare abbia davvero una proprietà con un certo valore, ecc.

Per quanto detto finora è evidente che il concetto di proprietà è fondamentale per la rappresentazione della conoscenza: permette non solo di descrivere entità, ma anche di creare relazioni fra di loro.

Finora abbiamo indicato le proprietà con degli slot.

Come rappresentiamo una proprietà?

Una proprietà è una entità



Si può dunque rappresentare unà proprietà con una unit, che la descrive.

Ad esempio di può dire per quali classi ha senso la proprietà, il tipo dei valori che può assumere (misura, numero, boolean, colore, ecc.), il formato dei valori (singolo valore, lista, lista ordinata, ecc.)

Quetsta descrizione, anche se può sembrare pesante, è di grande importanza nella costruzione di un modello concettuale della conoscenza utilizzata in un sistema esperto. Infatti, descrivendo anche le proprietà, si eliminano possibili ambiguità e incompletezza di informazione.

Nell'uso pratico del linguaggio che stiamo introducendo, si può anche utilizzare una forma abbreviata della descrizione di proprietà, che fornisce un insieme di informazioni utili nel momento in cui viene definito uno slot. Ad esempio, se la proprietà #BirthDate viene descritta nella seguente unit con lo slot $BirthDate, si può indicare direttamente al posto del valore un termine che viene inteso come tipo di valore, la cui struttura viene intesa come formato di valore, indicando addirittura qui gli elementi essenziali di tale struttura.

```
Unit Citizen
        $IsA (#Collection)
        $GeneralizedBy  (#Person)
        $SpecializedBy  (#Worker #Student)
        $Instances (#Fred #Mary)
        $InstProperties (($BirthDate
                        (TYPE #Date
                            ($InstProperties
                                ($Day (TYPE #Day))
                                ($Month (TYPE #Month ))
                                ($Year (TYPE #Year))))))
```

Problemi relativi alle proprietà

Definizione di classi o definizione di proprietà?

Es: "Le auto dei pompieri sono rosse" si può rappresentare con:

- La proprietà $RedP nella classe #AutoDeiPompieri
- La proprietà $Color nella classe #AutoDeiPompieri
- La classe #RedThing
- La classe #RedCar

Il problema citato si riferisce alla scelta di rappresentazione di proprietà come slot di classi, o come slot caratteristici di classi specializzate.

Nell'esempio riportato abbiamo diverse alternative.

La prima prevede di avere uno slot booleano nella classe in oggetto (#AutoDeiPompieri) che dica se l'auto dei pompieri è rossa o no.

La seconda prevede di definire uno slot più generale ($Color), che permetterebbe di definire diversi colori per le auto dei pompieri.

La terza alternativa presuppone che venga definita una classe delle cose rosse, caratterizzata dal fatto che tutti i suoi elementi hanno la proprietà descritta dallo slot ($Color #Red). In questo caso #AutoDeiPompieri sarebbe una specializzazione anche di #RedThing.

L'ultima alternativa prevede che venga definita la classe delle auto rosse, di cui #AutoDeiPompieri è una specializzazione.

Quale di queste alternative è la migliore? Non esiste una migliore in termini assoluti, ma la scelta dipende dallo scopo con cui si sta costruendo la base della conoscenza. Ad esempio, se dovremo ragionare sui colori degli oggetti (magari perchè il nosro sistema deve supportare il riconoscimento visivo) potrebbe andare bene la seconda proposta, in quanto, probabilmente, tutti gli oggetti che il nostro sistema tratterà sono caratterizzati da una proprietà #Color. In questo caso la prima soluzione sarebbe rigida (il sistema dovrebbe operare con molte proprietà relative ai diversi colori), mentre le ultime due soluzioni non sembrano particolarmente interessanti, in quanto non occorrerebbe ragionare sui concetti descritti dalle classi in questione.

Invece, la penultima proposta andrebbe bene se dovessimo ragionare circa gli oggetti rossi (ad esempio per suggerire di trattarle in un certo modo all'interno di un'immagine pubblicitaria), mentre l'ultima soluzione potrebbe essere la mgiliore nel caso volessimo rappresentare il pericolo che può essere associato alle macchine rosse (sia sia le auto dei pompieri che le Ferrari vanno veloci).

Si ricorda l'importanza di generare in fase di rappresentazione della conoscenza diverse alternative, superando il blocco che spesso esiste e che ci focalizza sulla prima soluzione trovata. Come in tutti i progetti per poter effettuare delle scelte occorre avere a disposizione diverse alternative. Senza alternative non c'è scelta.

Il concetto di gerarchia

Una gerarchia è una **relazione di ordinamento tra classi** secondo una certa proprietà.

Es.: la gerarchia spec/gen (le classiche tassonomie), la gerarchia part-of, la gerarchia more-relevant, ...

Una gerarchia è una generica relazione di ordinamento in relazione ad una certa proprietà.

Le gerarchie a cui siamo abituati a pensare (si pensi alle gerarchie di classificazione tradizionali ome quelle delle Scienze Naturali), sono gerarchie sviluppate secondo le proprietà di specializzazione e generalizzazione.

Altre gerarchie possono essere definite, ad esempio, secondo la proprietà #PartOf, che crea una relazione tra le parti di un'entità e l'entità stessa.

Altra gerarchia comunemente usata è quella definita attraverso la proprietà #instOf, tra esemplari e classi.

A cosa serve una gerarchia ?

Per realizzare il mecanismo inferenziale di eredità.

Es.: La classe degli struzzi eredita da quella degli uccelli il fatto che tutti i suoi esemplari hanno le piume (gerarchia spec/gen)

Le parti di una entità composta ereditano la localizzazione della entità stessa (gerarchia part-of). ....

Il concetto di relazione gerarchica serve per realizzare il meccanismo inferenziale dell'***eredità***: una classe eredita valori di certe proprietà dalle classi che la precedono nella gerarchia.

Ad esempio, nella gerarchia di generalizzazione e specializzazione, le classi più specifiche ereditano le proprietà che descrivono gli esemplari delle classi più generali (nella nostra rappresentazione, il contenuto dello slot $InstProperties).

Nella gerarchia di parte e sottoparte, viene ereditato il valore che descrive il proprietario (chi possiede un oggetto possiede anche le sue parti) e la locazione (dove sta un oggetto stanno anche le sue parti).

Qualche parola in più sull' eredità

Eredità multipla

Es.: Uno struzzo è un uccello, ma anche un corridore...

Overriding

Es.: Gli struzzi non volano, eppure sono uccelli...

In generale, un'entità può essere in relazione gerarchica con diverse altre entità. Nell'esempio riportato l'entità descritta dalla unit #Struzzo è in relazione gerarchica con #Uccello, ma anche con #Corridore, che appartengono a diverse gerarchie (anche se tutte di Specializzazione e Generalizzazione)

Di solito, nel meccanismo di eredità, è adottata una politica in cui le proprietà definite ai livelli inferiori dela gerarchia hanno la meglio in caso di conflitto, rispetto alle analoghe proprietà definite ai livelli superiori. Questo meccanismo si chiama (sovrascrittura (overriding) dei valori" e viene attuato quando le proprietà considerate non sono quelle caratteristiche, ma quelle tipiche delle classi.

Test #1: rappresentazione dichiarativa

Rappresentiamo la conoscenza contenuta in questo testo:

Un ingegnere è una persona.
Aristide è un ingegnere.
Gli ingegneri sono laureati.
Aristide si è laureato il 20 luglio 2005.
Ad Aristide piace ballare.
Agli ingegneri piace il computer e non piace ballare.

In questo esercizio viene richiesto di rappresentare con il formalismo delle unit la conoscenza contenuta nelle frasi sopra citate.
Siete invitati a generare, ove possibile, più di un'alternativa e a valutare le varie alternative generate.

Possibili alternative.
La prima frase é una definizione di classe-sottoclasse.
La seconda é una definizione di esemplare-classe
La terza può essere:

- Classe #Ingegneri, generalizzata dalla classe #Laureati, che contiene slot per $DataDiLaurea, $TipoLaurea, ecc.
- Ogni esemplare di #Ingegneri ha uno slot $LaureatoP

La quarta potrebbe permettere di pensare che anche la classe dei laureati, oltre ad Aristide, ha lo slot $DataDiLaurea, che è una data.
La quinta è una specifica caratteristica di Aristide, ma potrebbe essere pensata come caratteristica di tutte le #Person potrebbe avere come default anche #Ballare. Abbiamo qui l'alternativa tra varie scelte di rappresentazione:

- una proprietà che raggruppa le cose che piacciono ed una che raggruppa le cose che non piacciono: in questo caso il meccanismo inferenziale deve gestire l'incompatibilità di presenza
- una proprietà che raggruppa le cose rispetto a cui si sa che piacciono o no, che contiene anche dei simboli per indicare quelle che non piacciono: anche qui il meccanismo inferenziale deve poter gestire eventuali incompatibilità

I problemi escono con l'ultima frase...

- Un ingegnere e' una persona

Unit Persona
$IsA(#Collection)
$SpecializedBy(#Ingegnere)
$InstProperties(($Name))

Unit Ingegnere
$IsA(#Collection)
$GeneralizedBy(#Persona)

- Aristide e' un ingegnere

Unit Ingegnere
$IsA(#Collection)
$GeneralizedBy(#Persona)
$Instances(#Aristide)

Unit Aristide
$InstanceOf(#Ingegnere)
$Name('Aristide')

Unit Name
$IsA(#Property)
$MakesSenseFor(#Persona)
$EntryType(#String)

- Gli ingegneri sono laureati

```
Unit Persona
$IsA(#Collection)
$SpecializedBy(#Laureato)
$InstProperties(($Name))

Unit Ingegnere
$IsA(#Collection)
$GeneralizedBy(#Laureato)
$InstProperties (($TipoLaurea 'Ingegneria'))
```

```
Unit Laureato
$IsA(#Collection)
$SpecializedBy(#Ingegnere)
$GeneralizedBy(#Persona)
$InstProperties(
        ($LaureatoP True)
        ($TipoLaurea)
        ($DataLaurea
          ($EntryType #Date)))
```

- Aristide si è laureato il 20 luglio 2005

```
Unit Aristide
$InstanceOf(#Ingegnere)
$Name('Aristide')
$DataLaurea(($Giorno 20)
                ($Mese 'luglio')($Anno 2005))
```

- Ad Aristide piace ballare

Unit Persona
$IsA(#Collection)
$SpecializedBy(#Laureato)
$InstProperties(($Name)($Likes ($EntryFormat #List)))

Unit Aristide
$InstanceOf(#Ingegnere)
$Name('Aristide')
$DataLaurea(($Giorno 20)($Mese 'luglio')($Anno 2005))
$Likes((#Ballare))

Agli ingegneri piace il computer e non piace ballare.

```
Unit Persona
$IsA(#Collection)
$SpecializedBy(#Laureato)
$InstProperties(($Name)($Likes ($EntryFormat #List)) )($DisLikes ($EntryFormat
    #List)))

Unit Aristide
$InstanceOf(#Ingegnere)
$Name('Aristide')
$DataLaurea(($Giorno 20)($Mese 'luglio')($Anno 2005))
$Likes((#Ballare))

Unit Ingegnere
$IsA(#Collection)
$GeneralizedBy(#Laureato)
```

Cos'è un'inferenza?

Ricavare conoscenza da conoscenza già codificata

Abbiamo finora visto soprattutto gli aspetti dichiarativi della rappresentazione della conoscenza. Vediamo ora le strutture inferenziali che ci permettono di ricavare conoscenza partendo da conoscenza già codificata.

è interessante, anche per le strutture inferenziali, identificare diversi tipi di strutture, in modo da potersi focalizzare, in fase di costruzione del modello concettuale, sulle strutture che sono veramente usate nell'applicazione specifica.

In Cyc, il sistema che abbiamo citato precedentemente, sono state identificate ben 24 diverse strutture inferenziali. Noi ne vedremo solo le principali.

Vedremo anche che la forma più generale è quella che si traduce in regole. Ha comunque interesse vedere anche strutture più specifiche, perchè queste aiutano ad identificare delle particolari necessità all'interno di un'applicazione.

Rappresentazione di strutture inferenziali: esempio

SI Inverso
If (<ref1> (<slot-name> <ref2>))
Then (<ref2> (<inv-slot-name><ref1>))

Ad esempio, nel caso di HasFather e FatherOf, abbiamo

SI Inverso
If (?X ($HasFather ?Y))
Then (?Y ($FatherOf ?X))

Questo e' solo un esempio di come sia possibile definire strutture inferenziali con ruoli interessanti e contenuto piu' specifico delle generali regole di inferenza.

Test #2: Inferenza

In questo testo è presente solo una parte della conoscenza che servirebbe per realizzare un sistema in grado di diagnosticare guasti di un'auto. Rappresentiamola e cerchiamo di completarla.

I motivi per cui un'auto puo' non partire sono: batteria scarica, benzina terminata, motorino di avviamento bloccato, antifurto inserito.

Inoltre, occorre rappresentare le cause per cui si verificano queste condizioni, ...

In questo esercizio emergono diverse caratteristiche sia per quanto riguarda la conoscenza dichiarativa (classi, esemplari proprietà) sia per quanto riguarda i processi inferenziali coinvolti.

Almeno due sono i possibili approcci alla soluzione del problema. Nel primo viene descritta la struttura dell'oggetto della diagnosi e, partendo da quella, si giunge alla definizione dei meccanismi inferenziali che vi operano. Nel secondo approccio, invece, ci si focalizza su concetti quali guasto e osservazione, giungendo attraverso questi alla definizione dell'oggetto della diagnosi e dei meccanismi inferenziali necessari.