NeoClassic

un linguaggio per la rappresentazione della conoscenza

Introduzione

NeoClassic [1] [2] è un sistema per la rappresentazione della conoscenza basato sulla logica descrittiva che affonda le radici nel paradigma di KL-ONE [3] e permette di:

- definire concetti strutturati e organizzarli in una tassonomia;
- creare e manipolare istanze individuali di tali concetti;
- derivare conoscenza tramite meccanismi di inferenza di completamento, sussunzione e regole di concatenazione in avanti (*forward chaining rules*)

NeoClassic fa uso di quattro tipi di oggetti:

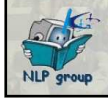
CONCETTI
INDIVIDUI
RUOLI
REGOLE

Concetti

- Un concetto corrisponde ad una *descrizione* alla quale viene associato un identificatore (nome del concetto).
- Ogni concetto ha almeno un genitore.
- Un concetto può essere *primitivo* o *definito.*

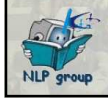
Individui

Rappresentano gli oggetti del dominio. Agli individui vengono assegnate proprietà asserendo che i loro "ruoli" sono "riempiti" da altri individui e/o classificandoli come istanze di concetti (ereditando le proprietà del concetto). Ogni individuo è necessariamente sussunto da un qualche concetto e classificato all'interno della tassonomia



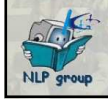
Ruoli

Rappresentano le proprietà dei concetti e degli individui. Sono usati nelle descrizioni e per mettere gli individui in relazione tra loro. I ruoli di un individuo possono essere "riempiti" da altri individui o avere i potenziali "riempitori" vincolati da determinate restrizioni.



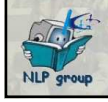
Regole

Sono regole di concatenazione in avanti composte da un concetto antecedente e un concetto conseguente. Sono utilizzate per ricavare *informazione derivata* dalla base di conoscenza.



Concetti

(*definiti* e *primitivi*)



Esempio di concetto primitivo (1)

Vogliamo definire il concetto di PERSONAGGIO: non siamo in grado di darne una descrizione completa, ma vogliamo definirlo come qualcosa che, *tra le altre cose*, possiede un "*nome*" ed un "*anno di nascita*":

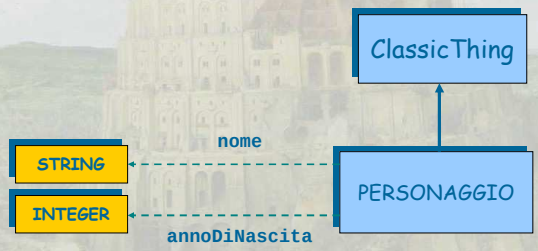
```
(createConcept PERSONAGGIO (and ClassicThing
      (all nome STRING)
      (all annoDiNascita INTEGER)) true)
```

con "true" definisco un concetto primitivo

in cui nome ed annoDiNascita costituiscono delle proprietà chiamate *ruoli* (che devono essere precedentemente definiti).

NLP group

Esempio di concetto primitivo (2)




```
(createConcept PERSONAGGIO (and ClassicThing
      (all nome STRING)
      (all annoDiNascita INTEGER)) true)
```

NLP group

Esempio di concetto primitivo (3)

Il concetto PERSONAGGIO è primitivo e fornisce condizioni necessarie ma non sufficienti di appartenenza, in altre parole:

- qualcosa che viene definito essere un PERSONAGGIO sarà dotato di un nome ed un anno di nascita (condizione necessaria)

- qualcosa che ha un nome ed un anno di nascita può non essere classificato come PERSONAGGIO (condizione non sufficiente)

NLP group

Esempio di concetto definito (1)

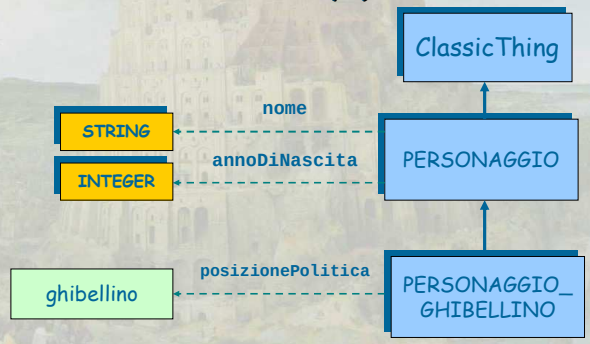
Quando è disponibile una descrizione esaustiva è possibile creare un concetto definito che fornisce condizioni necessarie e sufficienti. Ad esempio il concetto PERSONAGGIO_GHIBELLINO è definito come un PERSONAGGIO con un ruolo posizionePolitica riempito dall'individuo "ghibellino":

```
(createConcept PERSONAGGIO_GHIBELLINO(and PERSONAGGIO
      (fills posizionePolitica ghibellino))
```

La creazione di un individuo che sia un PERSONAGGIO e che abbia il valore ghibellino come *filler* (riempitore) del ruolo **posizionePolitica** sarà classificato come istanza di PERSONAGGIO_GHIBELLINO.

NLP group

Esempio di concetto definito (2)




```
(createConcept PERSONAGGIO_GHIBELLINO(and PERSONAGGIO
    (fills posizionePolitica ghibellino))
```

Esempio di concetto definito (3)

PERSONAGGIO_GHIBELLINO è un concetto definito per cui:

- un individuo definito come PERSONAGGIO_GHIBELLINO sarà anche un PERSONAGGIO e avrà il ruolo posizionePolitica riempito da "ghibellino" (condizione necessaria).
- un individuo definito come PERSONAGGIO che abbia il ruolo posizionePolitica riempito da ghibellino sarà classificato come PERSONAGGIO_GHIBELLINO (condizione sufficiente).

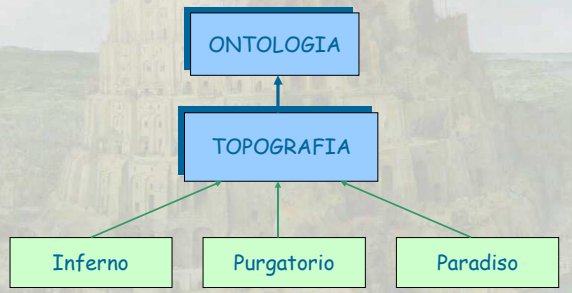
Concetti definiti per *enumerazione* (1)

E' possibile definire un concetto enumerando esplicitamente gli individui che ne costituiranno le uniche istanze possibili.

Ad esempio:

```
(createConcept TOPOGRAFIA (and ONTOLOGIA
    (oneOf Inferno Purgatorio Paradiso)))
```

Concetti definiti per *enumerazione* (2)




```
(createConcept TOPOGRAFIA (and ONTOLOGIA
    (oneOf Inferno Purgatorio Paradiso)))
```

Individui, ruoli e restrizioni di ruolo

Per definire le proprietà:

Le proprietà dei concetti e degli individui sono definite con i ruoli, ad esempio, per il concetto PERSONAGGIO:

```
(createRole notoCome)
(createRole noteBiografiche)
```

aggiungendo **true** si ottiene un *attributo*, ad esempio:

```
(createRole annoDiNascita true)
```

definisce il ruolo **annoDiNascita** con la particolarità che può avere al più un riempitore → è un *attributo*.

Restrizioni di ruolo

I quattro operatori **all**, **atLeast**, **atMost** e **fills** formano le restrizioni di ruolo, cioè impongono vincoli ai riempitori di un ruolo.

Vediamo un esempio in cui defineremo passo per passo il concetto di EPISODIO_IN_TOSCANA, cominciando dalle definizioni dei concetti EPISODIO, CITTÀ e CITTÀ_TOSCANA, seguite dalla definizione del ruolo luogo e da tutte le altre informazioni necessarie.

Esempio di concetto

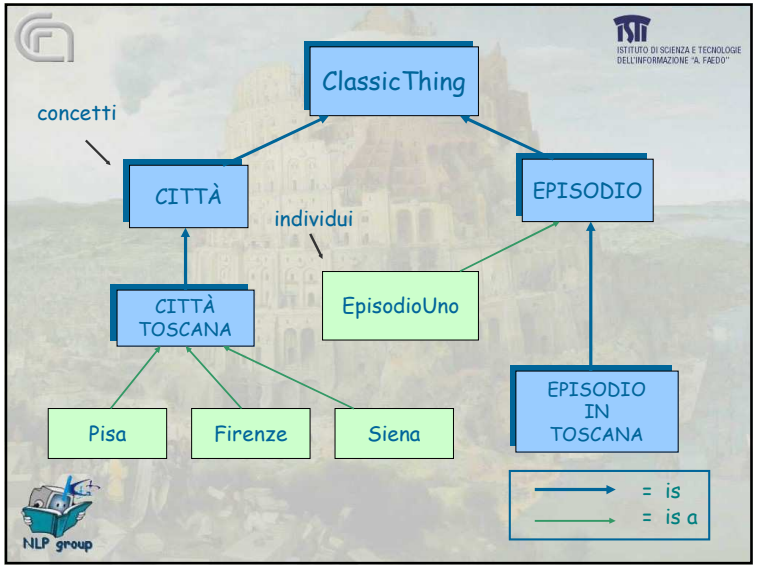
- `(createConcept EPISODIO ClassicThing true)`
- `(createConcept CITTÀ ClassicThing true)`
- `(createConcept CITTÀ_TOSCANA (and CITTÀ (oneOf Firenze Pisa Siena)))`
- `(createRole luogo)`
- `(createConcept EPISODIO_IN_TOSCANA (and EPISODIO (all luogo CITTÀ_TOSCANA)))`

restrizione di valore

A questo punto è possibile creare un individuo "episodio": se ad esempio non sappiamo dove abbia avuto luogo possiamo definirlo per il momento come semplice istanza del concetto EPISODIO…

```
(createIndividual EpisodioUno EPISODIO)
```

La tassonomia di concetti e individui mantenuta da NeoClassic avrà la struttura seguente…



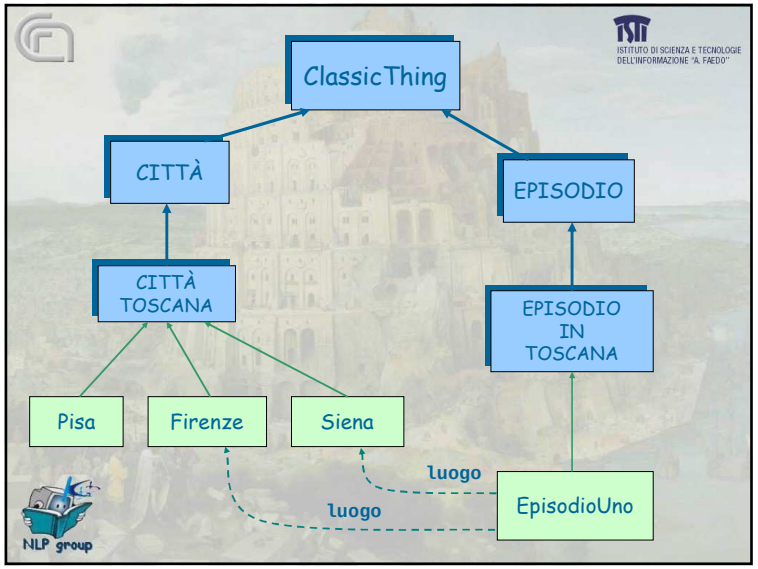
Aggiungere informazione

E' possibile in un momento successivo alla creazione di un individuo aggiungervi o togliervi informazione, ad esempio:

```
(addToldInformation EpisodioUno (and
      (fills luogo Firenze)
      (fills luogo Siena)))

(closeRole EpisodioUno luogo)
```

La tassonomia sarà aggiornata riclassificando **EpisodioUno** come istanza di EPISODIO_IN_TOSCANA…



Altre restrizioni

Oltre ad "**all**" e a "**fills**" gli altri due operatori di restrizione sono:

- "**atMost**" e
- "**atLeast**"

utilizzati per definire descrizioni con vincoli sul numero dei riempitori di un certo ruolo, ad esempio, volendo definire il concetto di un episodio che ha coinvolto un'unica città…

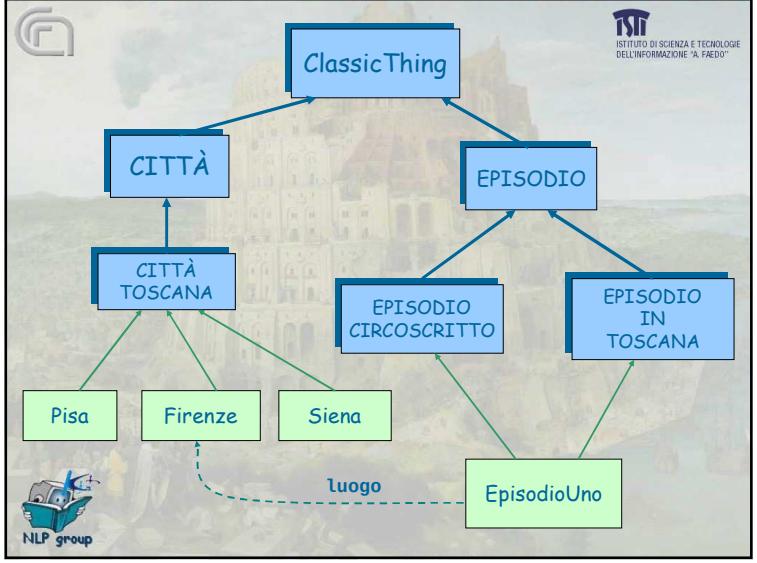
```
(createConcept EPISODIO_CIRCOSCRITTO (and EPISODIO
      (atMost 1 luogo)
      (atLeast 1 luogo))
```

restrizioni di numero

Volendo modificare l'individuo **EpisodioUno** specificando che ha avuto luogo soltanto a Firenze si procede in questo modo…

```
(uncloseRole EpisodioUno luogo)
(removeToldInformation EpisodioUno
      (fills luogo Siena))
(closeRole EpisodioUno luogo)
```

ottenendo…



Considerazioni

Con questo semplice esempio abbiamo evidenziato due caratteristiche chiave di NC :

- ***deduttivo***: è in grado di dedurre automaticamente fatti;
- ***incrementale***: descrizioni incomplete possono essere raffinate non appena si acquisisce nuova conoscenza.

inoltre…

E' possibile specificare restrizioni di tipo procedurale tramite operatori specifici (*testC* e *testH*) che permettono l'uso di procedure di controllo complesse scritte nel linguaggio host (nel caso di NeoClassic il *C++*), ad esempio, per definire il concetto PERSONAGGIO_DI_MEZZA_ETÀ:

```
(createConcept PERSONAGGIO_DI_MEZZA_ETÀ
      (and PERSONAGGIO (all età (and integer
            (testH checkIntervallo 30 60)))))
```

in cui **checkIntervallo** è una funzione C++ che verifica che l'età sia compresa tra 30 e 60.

un altro esempio…

Dopo aver definito un ruolo che rappresenti l'atteggiamento di Dante nei confronti di un personaggio:

```
(createRole atteggiamentoDante)
```

Definiamo il concetto PERSONAGGIO_ANTIPATICO_A_DANTE facendo uso della procedura utente **checkAntipatia** la cui unica funzione è cercare tra i riempitori del ruolo valori rappresentanti sentimenti negativi:

```
(createConcept PERSONAGGIO_ANTIPATICO_A_DANTE
  (and PERSONAGGIO
    (testC checkAntipatia atteggiamentoDante)))
```

Regole

NeoClassic supporta tre tipi di *regole di concatenazione in avanti*:

1. **regole semplici**
2. **regole con descrizione "calcolata"**
3. **regole con riempitore "calcolato"**

Regole semplici

Una regola semplice consiste di un antecedente (un concetto) e un conseguente (una descrizione o un concetto):

- data una regola con il concetto c_1 come antecedente e il concetto c_2 come conseguente allora, non appena l'individuo i_1 viene riconosciuto come appartenente a c_1, la regola viene "accesa" ed i_1 considerato anche appartenente a c_2 .

Esempio di regola semplice

Per illustrare l'uso delle regole riprendiamo ed estendiamo la definizione dei concetti relativi ai personaggi…

```
(createRole nome true)
(createRole annoDiNascita true)
(createRole luogoDiNascita true)
(createRole posizionePolitica true)

(createConcept PERSONAGGIO (and ClassicThing
      (all nome STRING)
      (all annoDiNascita INTEGER)) true)
(createConcept PERSONAGGIO_GHIBELLINO (and PERSONAGGIO
      (fills posizionePolitica ghibellino)))
(createConcept PERSONAGGIO_TOSCANO (and PERSONAGGIO
      (all luogoDiNascita (oneOf Firenze Siena Pisa))))
```

Se abbiamo bisogno di asserire il fatto che un qualunque personaggio ghibellino è originario di Firenze o Siena (una conclusione maturata ad un certo punto dell'evoluzione della base di conoscenza) possiamo usare la seguente regola…

antecedente

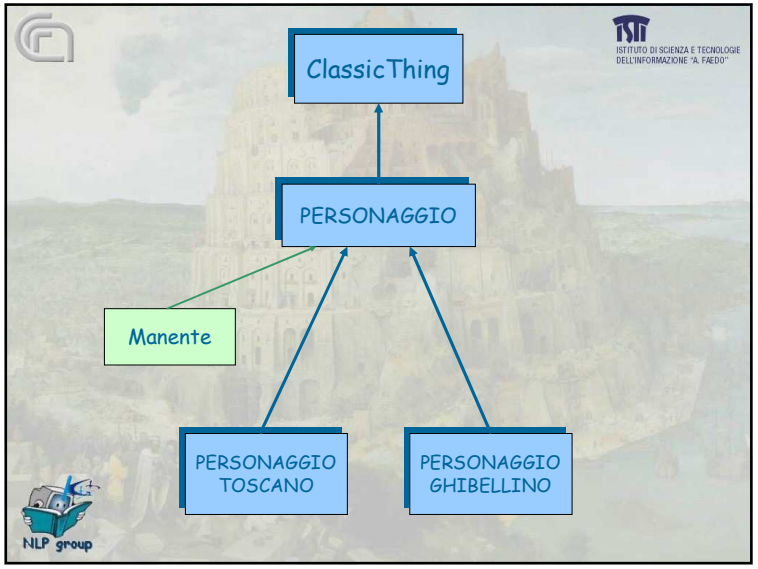
```
(createRule regolaUno PERSONAGGIO_GHIBELLINO
      (all luogoNascita (oneOf Firenze Siena)))
```

conseguente

Quali sono le conseguenze? Proviamo a creare il seguente individuo…


```
(createIndividual Manente (and PERSONAGGIO
      (fills nome "Manente-degli-Uberti")
      (fills annoDiNascita 1210)))
```

L'individuo **Manente** sarà classificato sotto il concetto PERSONAGGIO…

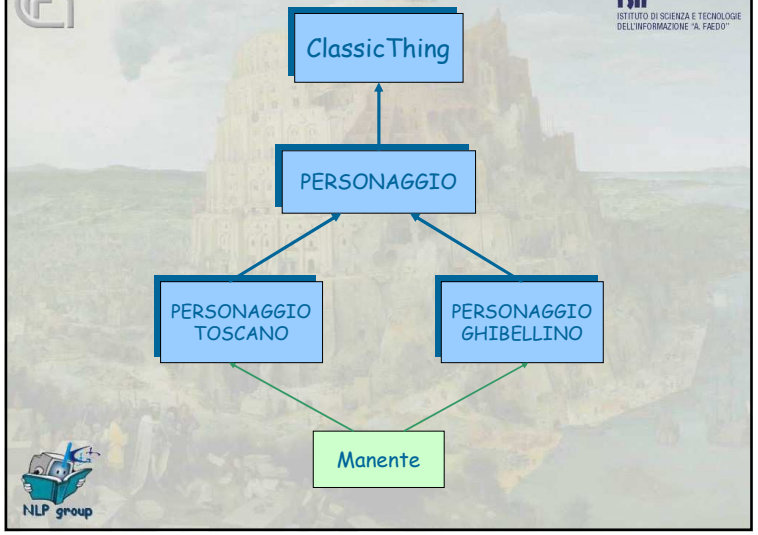


Se in seguito apprendiamo che Manente "Farinata" degli Uberti era ghibellino, possiamo aggiungere tale informazione con…

```
(addToldInformation Manente
    (fills posizionePolitica ghibellino))
```

L'informazione aggiunta porta ad una serie di conseguenze a catena…

1. L'individuo Manente viene riclassificato sotto il concetto PERSONAGGIO_GHIBELLINO, quindi…
2. … soddisfa l'antecedente della regola che di conseguenza viene "accesa" e …
3. … la descrizione contenuta nella parte destra della regola viene applicata all'individuo Manente che quindi risulta avere luogo di nascita Firenze o Siena e quindi, infine…
4. … viene classificato anche sotto il concetto PERSONAGGIO_TOSCANO.



Regole con descrizione calcolata

La parte destra di questo tipo di regole è calcolata da una funzione (con relativi parametri) al momento dell'accensione:

- il conseguente della regola non deve essere specificato al momento della creazione ma viene calcolato in base alla informazioni acquisite dalla base di conoscenza fino al momento dell'attivazione.

Regole con riempitore calcolato

Simili alle regole con descrizione calcolata ma oltre alla funzione ed ai parametri prendono un *ruolo*:

- la funzione restituisce una lista di riempitori per il ruolo al momento dell'accensione della regola.

L'inferenza in NeoClassic

I processi di inferenza supportati da NeoClassic possono essere suddivisi in tre categorie:

- **completamento**
- **classificazione e sussunzione**
- **applicazione di regole**

Vediamo in dettaglio le prime due…

Inferenza: completamento

- **ereditarietà**: le restrizioni che si applicano alle istanze di un concetto devono applicarsi anche alle istanze delle specializzazioni di tale concetto:
  - se *A* è un'istanza di *B* e *B* è una sottoclasse di *C* allora *A* "eredita" tutte le proprietà di *C;*
- **combinazione**: restrizioni su concetti e individui possono essere logicamente combinate per formare restrizioni più stringenti:
  - se *A* è definito come istanza dei concetti *B* e *C* esso sarà dotato della descrizione ottenuta combinando - in and - le descrizione di *B* e *C*
- **propagazione**: l'informazione associata ad un individuo viene propagata agli altri individui in relazione con esso:
  - se ad esempio *A* riempie il ruolo *r* di *B* e *B* è istanza di un concetto che vincola il ruolo *r* delle proprie istanze ad essere riempite soltanto da istanze del concetto *D*, è possibile concludere che *A* è un'istanza di *D*.

- **individuazione di contraddizioni**: NeoClassic è in grado di riconoscere affermazioni contraddittorie riguardo ad un individuo;
- **individuazione di concetti incoerenti**: nel caso in cui le restrizioni applicate ad un concetto si combinano per formare una contraddizione logica.

Inferenza: classificazione e sussunzione

- **classificazione di concetti**: possono essere individuati tutti i concetti più generali e più specifici di un certo concetto per mezzo del meccanismo della **sussunzione**;
- **classificazione di individui**: possono essere individuati tutti i concetti ai quali appartiene un certo individuo;

Utilità di NeoClassic

Per concludere la presentazione riassumiamo quelle che possono essere le caratteristiche decisive nella scelta (o nel rifiuto) di NeoClassic come sistema di rappresentazione della conoscenza...

- **orientato agli oggetti**: tutti gli individui hanno un'unica e immutabile identità acquisita al momento della creazione;
- **terminologico**: tramite le definizioni di concetti, ben si presta alla descrizione di oggetti complessi; d'altra parte non è appropriato per gestire asserzioni complesse che necessitano di quantificatori o disgiunzioni;
- **deduttivo**: non è una semplice base di dati passiva ma permette di dedurre fatti in base alle richieste presentate;
- **incrementale**: sono ammesse descrizioni di individui parziali e incomplete;
- **supporta ritrattazione di conoscenza**: tiene traccia delle dipendenze tra i fatti e permette che certi fatti siano ritrattati;

- **supporta regole**: applicate con la concatenazione in avanti e accese ogni volta che un individuo soddisfa l'antecedente;
- **supporta test procedurali**: concetti troppo complessi per essere espressi nel linguaggio di NeoClassic possono essere descritti tramite procedure scritte nel linguaggio ospite;
- **buona integrazione con il linguaggio ospite**: è possibile gestire le strutture del linguaggio ospite come istanze delle rispettive classi senza dover essere mappate in individui di NeoClassic;

Queste caratteristiche fanno di NeoClassic un sistema adatto per:

- domini dove è necessario organizzare un gran numero di oggetti che possono essere naturalmente rappresentati in termini di "caratteristiche" e "ruoli";
- applicazioni di recupero d'informazione dove ogni oggetto ha una descrizione complessa e le interrogazioni si presentano come descrizioni di oggetti aventi una certa struttura;
- applicazioni che coinvolgono basi di dati con schemi che evolvono nel tempo;

e in particolare ...

- applicazioni che coinvolgono descrizioni che evolvono incrementalmente nel tempo, a partire da descrizioni parziali.

infatti NeoClassic permette di:

- asserire a quanti oggetti un individuo è associato tramite un ruolo senza conoscere la natura di tali oggetti (es: "ogni personaggio ha almeno un oggetto associato a sé stesso tramite il ruolo **epoca**")

- descrivere i riempitori dei ruoli senza conoscerli (es: "tutti i riempitori del ruolo **`luogo-di-nascita`** sono città")

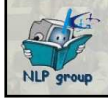
- informazione incompleta può essere raffinata man mano che si acquisisce nuova conoscenza (es: il luogo di nascita di un personaggio era inizialmente una città della Toscana che in seguito si scopre essere Firenze)

- gli individui possono essere riclassificati dopo l'acquisizione di nuova conoscenza (es: Farinata era classificato come PERSONAGGIO e in seguito specializzato come PERSONAGGIO-GHIBELLINO)

Naturalmente ci sono applicazioni che non si prestano all'uso di NeoClassic...

Quando non usare NeoClassic

- applicazioni che richiedono manipolazioni di entità matematiche come tuple, sequenze, ecc.
- recupero di informazione semplice da basi di dati;
- applicazioni che richiedono condizioni complesse nell'antecedente delle regole;
- applicazioni che richiedono aggiornamenti frequenti della KB.



Riferimenti

- [1] R. Brachmann, A. Borgida, D. McGuinnesse e P. Patel-Schneider. 'Reducing' CLASSIC to practice: Knowledge representation theory meets reality. In KR'92 Principles of Knowledge Representation and Reasoning. Proceedings of the Third International Conference, pages 247--258.
- [2] NeoClassic Knowledge Representation System Tutorial. http://www.bell-labs.com/project/classic/papers/NeoTut/NeoTut1.html
- [3] Ronald J. Brachman. A Structural Paradigm for Representing Knowledge, Ph.D. thesis, Harvard University, USA, 1977.