Università degli Studi di Roma "La Sapienza"

Facoltà di Ingegneria

Corso di Laurea Specialistica in Ingegneria Informatica

" Traduzione di diagrammi UML in Protégé e confronto tra DL e DL-LiteA"

Corso di "Seminari di Ingegneria Del Software"

Prof. Giuseppe De Giacomo

INDICE

1. Introduzione

Il presente lavoro ha come obiettivo la creazione di ontologie in OWL-DL a partire da diagrammi UML utilizzando come editor Protégé, un tool che consente di implementare schemi concettuali.

La passi per la progettazione di ontologie sono :

– definire le classi e organizzarle in una gerarchia tassonomica (sottoclassi-superclassi);
– definire le proprietà (object e datatype) per ciascuna classe
– definere i vincoli associati a ciascuna classe

A partire dai compiti di "Progettazione del Software", per ciascun diagramma UML è stata creata l'ontologia corrispondente ovvero una descrizione formale esplicita dei concetti del dominio.
Si è cercato il modo di tradurre gli elementi che caratterizzano i diagrammi UML: le classi, le associazioni, gli attributi e i vincoli, nei corrispondenti concetti, proprietà e restrizioni delle ontologie.
Sulle ontologie create, è stato utilizzato un ragionatore RACER (o classificatore), basato sulla Logica Descrittiva, che offre servizi per ragionare (fare inferenza) sulle basi di conoscenza.
I controlli che sono stati effettuati sono: il controllo di consistenza delle classi dell'ontologia e la classificazione dell'ontologia che consente di ottenere la *gerarchia desunta* delle classi, che può essere diversa da quella dichiarata durante la creazione.

Successivamente è stato utilizzato Swoop, un editor per ontologie OWL, che offre la possibilità di visualizzare le ontologie in varie formati, per generare la sintassi astratta.
E' stata effettuata un'analisi di questa sintassi e sono stati messi in evidenza i limiti del potere espressivo di OWL-DL.
In particolare si è evidenziato che con OWL-DL non è possibile esprimere gli attributi delle associazioni in quanto OWL non permette di inserire una *datatype property* come subproperty di una *object property*.

A partire da questa deduzione, la fase successiva è stata quella di studiare le caratteristiche di un nuovo linguaggio, DL-LiteA , che ha un potere espressivo diverso di OWL-DL, e che a differenza di questo invece consente di poter esprimere attributi di associazioni.
I file .owl ottenuti con Protégé sono stati tradotti in DL-liteA (attraverso un traduttore che restitutisce un file .xml) .

Dall'analisi di questi file si è potuto confrontare il potere espressivo dei due linguaggi.

2. Ontologie

La parola Ontologia può rivestire significati diversi a seconda dell'ambito dove viene utilizzata.
Nella filosofia è legato alla teoria dell'esistenza che si occupa di dare una risposta alle grandi domande del tipo:
Cos'è l'esistenza? Quali proprietà possono spiegarla? Come queste proprietà la spiegano?

Nell'Intelligenza Artificiale, la parola ontologia è usata per catturare la conoscenza su un dominio di interesse attraverso una descrizione formale ed esplicita di un insieme di concetti e delle relazioni che intercorrono tra essi. Un'ontologia cioè rappresenta il *modello concettuale* di un mondo, la struttura formale di un pezzo di realtà percepita ed organizzata da chi modella e tenta di formulare uno schema concettuale esaustivo e rigoroso nell'ambito di un certo dominio.

Gli elementi che costituiscono un'ontologia sono:

- Classi (concetti generali del dominio di interesse)
- Relazioni tra le classi
- Proprietà assegnate a ciascun concetto che ne descrivono vari attributi o proprietà
- Restrizioni sulle proprietà circa i valori che possono assumere

2.1 DL

Le *Description Logics* (DL) sono una famiglia di linguaggi per la rappresentazione della conoscenza, usate per modellare un dominio applicativo in modo strutturato.
Le logiche descrittive sono logiche specificatamente disegnate per modellare e per ragionare su: oggetti, classi (chiamati concetti in DLs) e relazioni binarie (chiamate ruoli inDLs).
Una logica Descrittiva è caratterizzata da:

- un linguaggio descrittivo (per esprimere concetti e ruoli)
- un meccanisco per specificare la conoscenza circa concetti e ruoli(TBox)
- un meccanisco per specificare le proprietà degli oggetti(ABox)
- un set di servizi di inferenza per ragionare sulla base di conoscenza

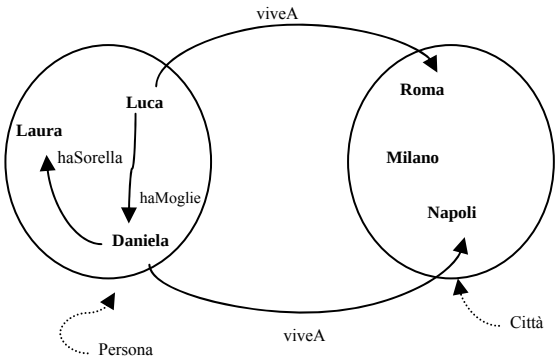
2.2 OWL

Il linguaggio più diffuso per la descrizione delle ontologie è l'OWL, definito nel 2004 dal World Wide Web Consortium.
Il linguaggio Owl è rilasciato in tre diverse versioni, che hanno una complessità e potere espressivo diverso. Ciascuna versione include ed espande la precedente.

- Owl-Lite è la versione sintatticamente più semplice, attraverso cui è possibile definire gerarchie di classi e vincoli poco complessi.
- Owl-DL è una versione intermedia, offre un potere espressivo più elevato e mantiene la completezza computazionale e la decidibilità
- Owl-Full offre la massima espressività ma non offre nessuna garanzia circa la completezza e la decidibilità .

In particolare la versione OWL-DL è così chiamata per la sua corrispondenza con le Logiche Descrittive ed è quella che è stata utilizzata in questa tesina.

I componenti principali di un'ontologia OWL sono tre: individui, proprietà e classi.
Gli *individui* rappresentano gli oggetti nel dominio di interesse, le *proprietà* sono relazioni binarie (ovvero che collegano due oggetti per volta) tra individui, le *classi* sono gruppi di individui.
Le classi OWL possono essere organizzate in gerarchie di superclassi e sottoclassi dette *tassonomie*.



Rappresentazione schematica di classi, proprietà, individui

3. Protégé

Protégé è un editor di ontologie gratuito ed open source, sviluppato da un gruppo di lavoro dell'università di Stanford. Basato su Java è estendibile grazie a numerose API e plug-in.
L'home page di Protégé è: http://protege.stanford.edu.
Protégé supporta due principali metodi di modellazione di ontologie ed è quindi disponibile nelle due versioni: Protégé-Frames e Protégé-OWL.
Nell'ambito di questa tesina viene fatto riferimento solamente a Protégé-OWL che consente di costruire ontologie per il Semantic Web con il linguaggio OWL descrivendo classi, proprietà, istanze.

4. UML in Protégé

Un Class Diagram Uml viene utilizzato per la rappresentazione di informazioni di un dominio di interesse.
Gli elementi che vengono modellati in un Class Diagramm Uml sono

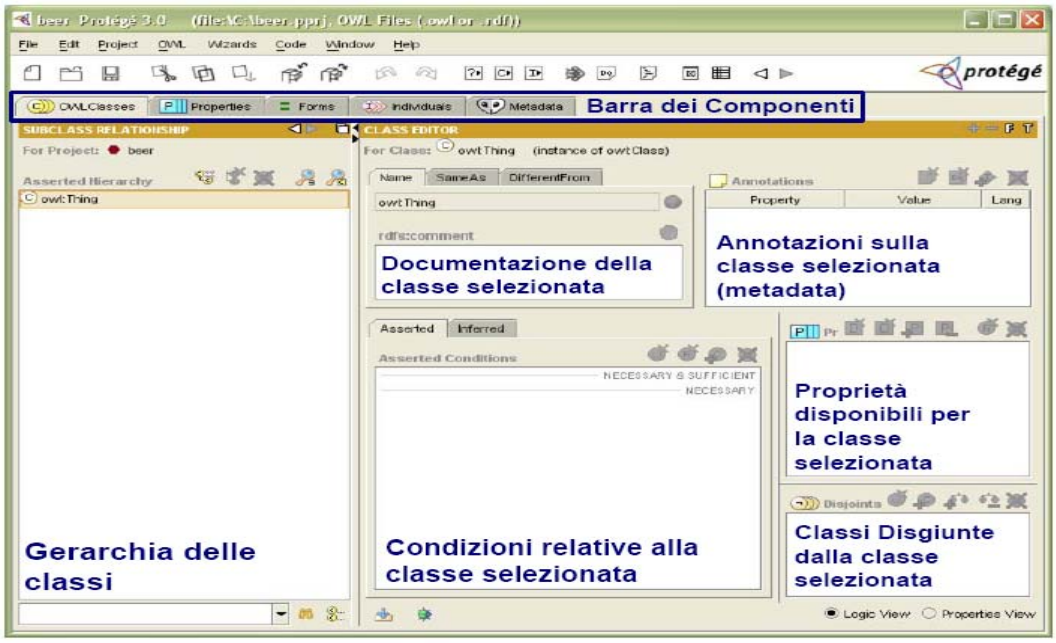
- classi ovvero gruppi di oggetti
- sottoclassi ovvero relazioni ISA e generalizzazioni
- associazioni ovvero relazioni tra le classi
- attributi e operazioni ovvero proprietà delle classi e delle associazioni
- cardinalità ovvero limite inferiore e superiore di un'associazione

Esaminiamo come è possibile tradurre questi elementi opportunamente in Protégé.

2.1 Classi → Classes

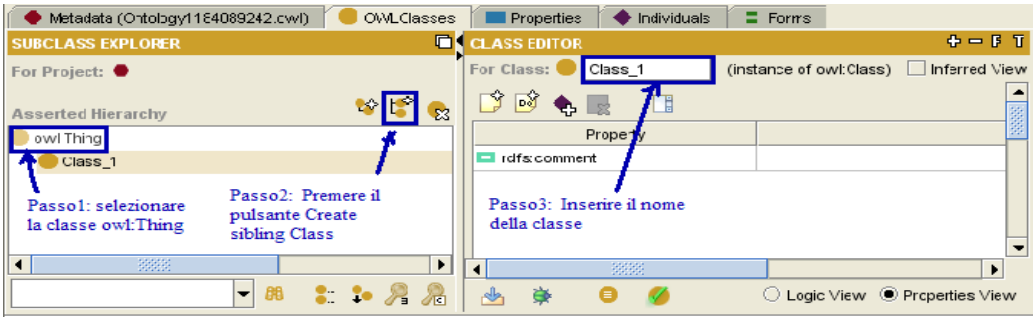
Le classi UML rappresentano un set di oggetti che posseggono le stesse caratteristiche.
Le classi possono essere organizzate in gerarchie di superclassi e sottoclassi.
Le sottoclassi ereditano tutte le proprietà delle superclassi e le specializzano attraverso le loro proprietà: quindi tutti i membri di una sottoclasse sono anche membri della superclasse (ma non vale il viceversa).

Ciascuna classe UML viene tradotta come una classe OWL in Protégé: vengono specificate le condizioni che devono essere soddisfatte da un individuo affinché appartenga a quella classe.
Tutte le classi devono essere inserite alla voce Class della barra dei Componenti.
L'ontologia vuota contiene la sola classe *owl:Thing* da cui vengono derivate tutte le altre classi.



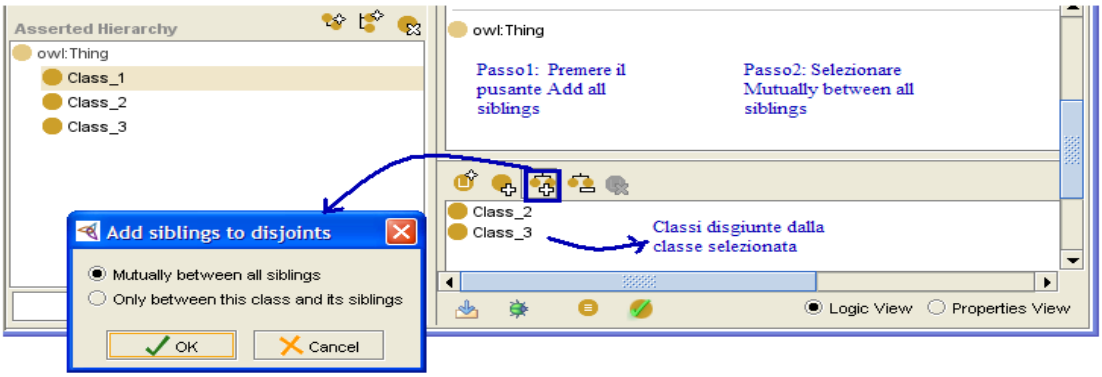
L'interfaccia utente di Protégé con il pannello delle classi attivo.

Per inserire una nuova classe nell'ontologia è sufficiente posizionarsi sulla una classe premere il pulsante *Create SiblingClass* e definire il nome della classe.



Creare una nuova classe.

Utilizzando lo stesso procedimento possono essere dichiarate le altre classi appartenenti al diagramma.
OWL non fa alcuna assunzione sulla disgiunzione tra classi ed assume che le classi possano sovrapporsi, se non dichiariamo esplicitamente il contrario.
Le classi in UML sono tra loro *disgiunte* (se non sono classi derivate), ovvero un individuo appartenente ad una non può contemporaneamente appartenere ad un'altra di esse.
La disgiunzione può essere dichiarata selezionando una qualsiasi delle classi sorelle e premendo il pulsante *Add all Siblings* nella sezione *Classi Disgiunte.*
Selezionando l'opzione *Mutually between all Siblings* dalla dialog box che appare ci si assicura, inoltre, non solo che la classe attuale sia disgiunta da tutte le sorelle, ma che anche ogni altra sorella sia disgiunta dalle altre.
Queste classi sono derivate dalla classe *owl:Thing.*



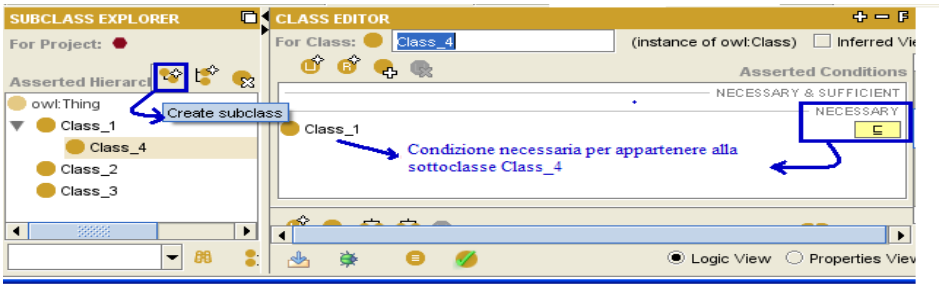
Rendere disgiunte tutte le classi derivate dalla stessa super-classe.

Possiamo associare ad ogni classe delle condizioni che possono essere necessarie o necessarie e sufficienti.
Se una classe è descritta attraverso condizioni necessarie, viene detta primitiva, e queste indicano delle restrizioni sugli individui che possono appartenere ad una classe.("se un individuo appartiene alla classe deve avere le seguenti caratteristiche").

Se una classe è descritta attraverso condizioni necessarie e sufficienti, viene detta definita, e queste indicano delle restrizioni sugli individui che possono appartenere ad una classe.(“se un individuo ha le seguenti caratteristiche esso fa parte di questa classe”).

2.2 Classi Derivate → Classes

Nei diagrammi UML possiamo rappresentare classi derivate tramite relazioni is-a o tramite generalizzazioni.
Per definire una relazione is-a tra classi , bisogna posizionarsi sulla superclasse e premere il pulsante *Create SubClass* e specificare il nome della classe.



Creare una tassonomia .

La condizioni necessaria indica che la condizione per cui un individuo appartenga ad una classe è che esso appartenga anche alla sua super-classe o, meglio, all’insieme delle sue super-classi (OWL supporta l’ereditarietà multipla).
Ogni sottoclasse, inoltre, eredita tutte le condizioni delle superclassi che vengono automaticamente inserite nel gruppo *Inherited*.

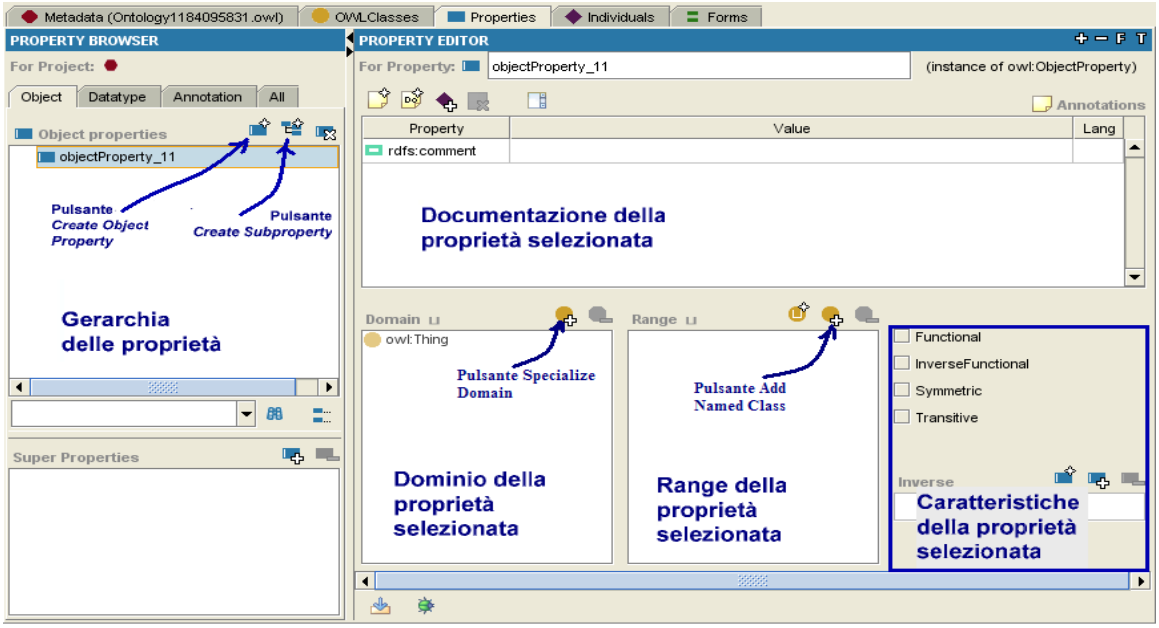
Per definire una generalizzazione tra classi(disgiunta e completa), oltre a definire le sottoclassi della superclasse bisogna assicurarsi che le sottoclassi allo stesso livello siano tra loro mutuamente disgiunte, applicando il procedimento sopra descritto.
Inoltre la superclasse deve essere l’unione delle sottoclassi e questo deve essere descritto tra le condizione necessaria e sufficiente ed ereditato dalle sottoclassi.

Un gerarchia di classi può essere chiamata tassonomia.

2.3 Associazioni → Object Properties

Nei diagrammi UML le associazione definiscono un insieme di legami simili tra coppie di oggetti che appartengono a due classi prefissate A e B.
Le associazioni vengono tradotte come Object Properties in Protègè.

Per definire le proprietà in Protégé è necessario accedere all’omonimo pannello, attraverso la barra dei componenti.



Il pannello *Properties* di Protégé – Object Properties.

Questo pannello è suddiviso in due sezioni principali: la parte sinistra mostra la gerarchia delle proprietà mentre la parte destra consente di caratterizzare la proprietà selezionata. Anche le proprietà possono infatti essere organizzate in gerarchie, dove ogni sotto-proprietà specializza la proprietà da cui è derivata.
Inserire una nuova proprietà è del tutto simile ad inserire una nuova classe ma inoltre bisogna definire il dominio e il range.
Il *dominio* di una proprietà è la classe (o l'insieme di classi) ai cui individui appartenenti si può applicare la proprietà.
Il *range* (o codominio) di una proprietà è invece la classe (o l'insieme di classi) i cui individui appartenenti possono essere valori della proprietà.
In altre parole, una proprietà collega individui appartenenti ad un dominio ad individui appartenenti ad un range.
Le sezioni *Dominio* e *Range* del pannello *Properties* hanno la funzione di raccogliere queste informazioni. Per inserirvi una nuova classe è necessario utilizzare il pulsante *Add Named Class* e scegliere la classe da inserire dalla dialog-box che appare.

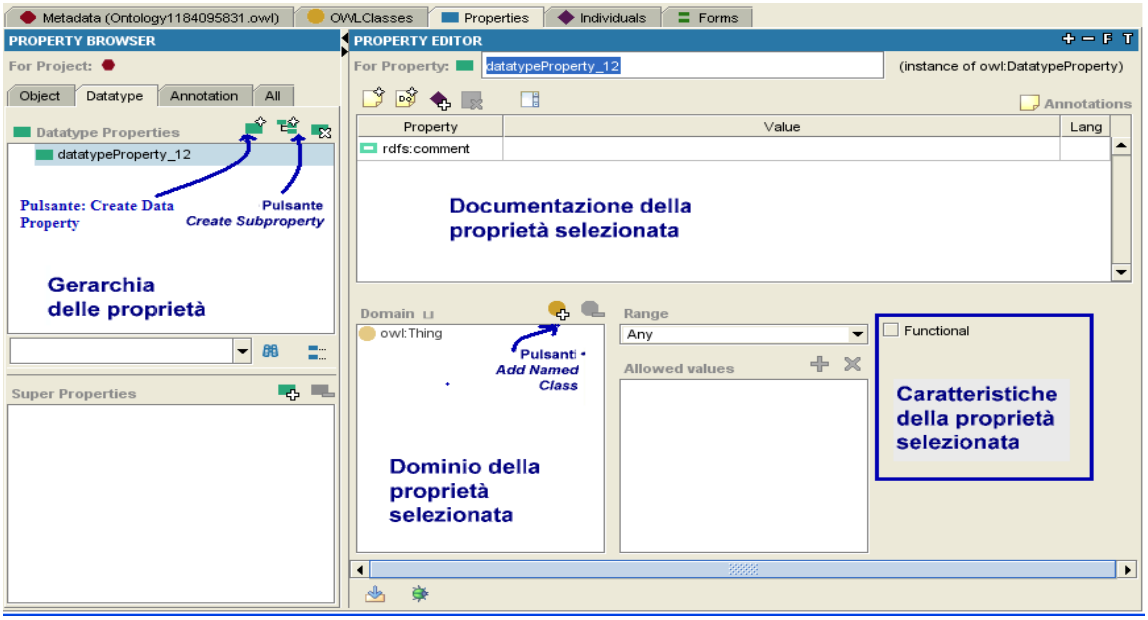
Le proprietà OWL possono avere diverse caratteristiche, tra cui la simmetria, la transitività, e la funzionalità. È possibile inoltre dichiarare che alcune proprietà sono inverse di altre.
Se una proprietà P è simmetrica relaziona l'individuo *a* con l'individuo *b* allora si può desumere che la proprietà P relazioni l'individuo *b* con l'individuo *a* anche se non esplicitamente dichiarato.

Se una proprietà P dichiarata transitiva relaziona l'individuo *a* con l'individuo *b* e l'individuo *b* con l'individuo *c* allora si può desumere che la proprietà P relazioni l'individuo *a* con l'individuo *c* anche se non esplicitamente dichiarato.
Se una proprietà P è dichiarata funzionale allora, dato un individuo *a*, esiste al più un individuo *b* che può essere relazionato ad *a* attraverso P. Ciò significa che se P relaziona l'individuo *a* con l'individuo *b* e l'individuo *a* con l'individuo *c* allora *b* e *c* sono lo stesso individuo.
Ogni Object Property può avere una corrispondente proprietà inversa. Se una proprietà collega un individuo a con un individuo b allora la proprietà inversa collega un individuo b con un individuo a.

2.4 Attributi → Datatype Properties

A ciascuna classe UML possono essere associate delle proprietà comuni (attributi) e i domini dei valori che possono essere assunti.
Gli attributi vengono tradotti utilizzando datatype properties.

Per definire le Datatype Properties in OWL Protégé è necessario accedere all'omonimo pannello, attraverso la barra dei componenti e selezionare la sezione specifica.
L'inserimento è analogo a quello per inserire Object Properties: anche in questo caso è possibile specificare il dominio, ovvero la classe a cui l'attributo appartiene e il range che in questo caso è il tipo associato all'attributo (string, int, data, etc..).
Per i datatype, però, l'unica caratteristica che può essere associata è di tipo functional(ad indicare che ogni individuo della classe ha associato un solo attributo)



Il pannello *Properties* di Protégé – Datatype Properties.

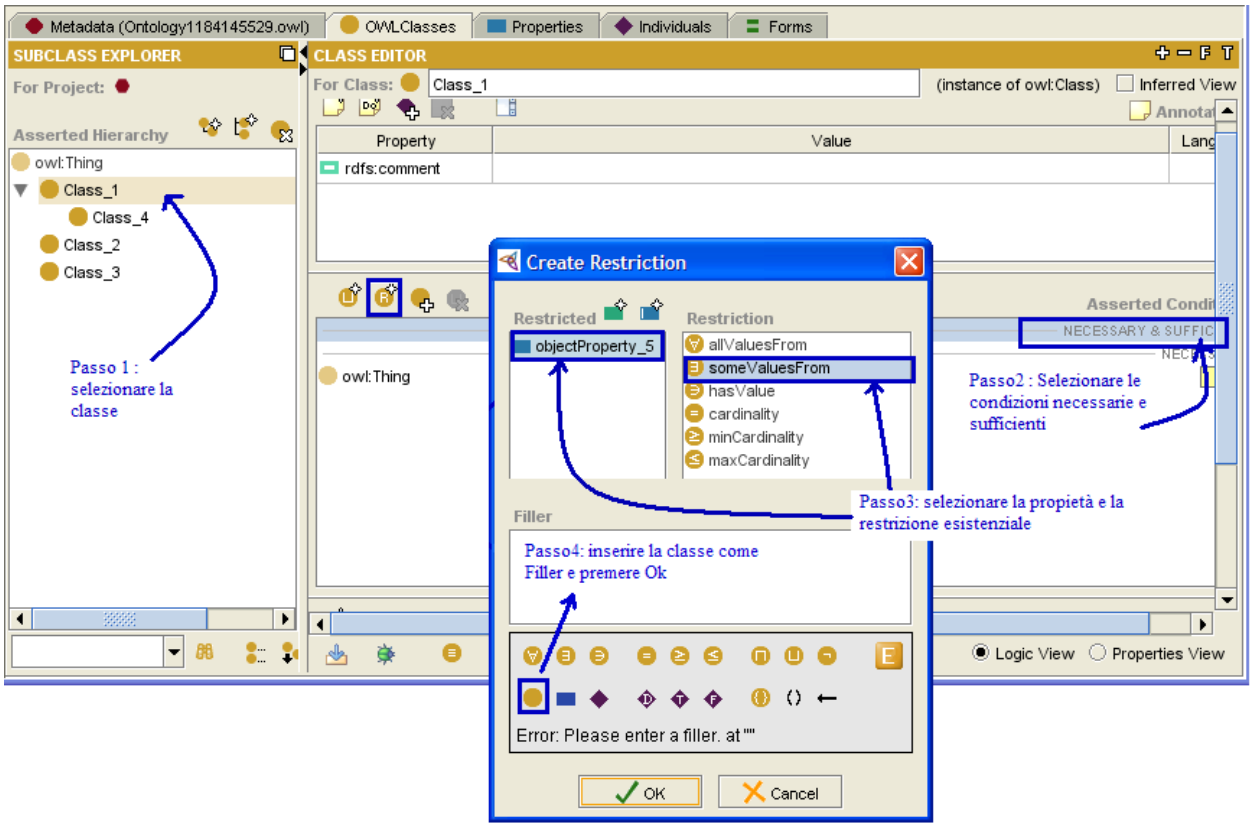
Non è possibile esprimere attributi relativi ad associazioni, e le operazioni relative a classi

2.5 Cardinalità → Restriction

In UML le cardinalità vengono utilizzate per indicare la partecipazione delle classi alle

associazioni attraverso una coppia (min,max).

Possiamo esprimere questi concetti attraverso delle restrizioni.
-- cardinalità (0,*) : è quella che viene indicata di default e perciò non è necessario indicarla attraverso restrizioni cardinali
-- cardinalità (1,*) : viene espressa tramite una "restrizione esistenziale" in OWL che indica che l'insieme di individui, per una data proprietà, hanno *almeno* relazioni con individui di una specifica classe.
-- cardinalità (1,1) : la cardinalità minima viene espressa attraverso una "restrizione esistenziale" (come precedentemente) mentre la cardinalità massima viene indicata attraverso la proprietà Functional (se a è in relazione con b, e se a e in relazione con c allora b e c coincidono).



Creare una nuova restrizione esistenziale.

E'necessario selezionare la classe di partenza dell'associazione, selezionare le condizioni necessarie e sufficienti (in modo che la restrizione venga indicata tra queste), e premere il pulsante *Create Restriction.* Nell'apposita palette che compare, occorre selezionare la proprietà coinvolta, ∃ dall'elenco delle restrizioni e inserire il nome della classe d'arrivo nel filler. Premendo Ok viene inserita la restrizione tra le condizioni necessarie e sufficienti del tipo:
∃ objectProperty_5 some Class_2.

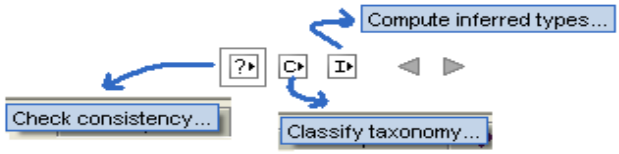
La "restrizione universale", invece, può essere utilizzata per indicare l'insieme di individui che, per una data proprietà, hanno *al più* relazioni con individui di una specifica classe.

5. Il reasoner Racer

I reasoner (o classificatori) sono programmi che offrono un insieme di servizi per ragionare (fare inferenza) sulle basi di conoscenza.
Esistono diverse tipologie di reasoner, ogni tipologia è compatibile con un determinato formalismo di rappresentazione della conoscenza.
Le ontologie realizzate in OWL-DL, in particolare, possono essere elaborate da reasoner basati sulla Logica Descrittiva.
Il ragionatore utilizzato è **Racer** realizzato dall' Università di Amburgo. I suoi realizzatori sono: Ralf Möller o Volker Haarslev.

E' possibile scaricare il file *Racer Server* un file compresso che contiene un seguibile. Cliccando due volte sull'icona viene lanciato ed è in esecuzione.
A questo punto i servizi offerti da Racer sono attivi e possono essere invocati via socket (la porta di default è la 8088) o via HTTP (la porta di default è la 8080).
È possibile scegliere una porta di comunicazione diversa per entrambe le interfacce attraverso le opzioni a linea di comando *-p* e *-http*.
L'interfaccia HTTP di Racer supporta la connessione da parte di client conformi allo standard DIG sviluppato dal DL Implementation Group e adottato da diversi software basati sulle logiche descrittive tra cui, appunto, Protégé.
Una volta in esecuzione, dunque, Racer può essere invocato da Protégé attraverso HTTP, sfruttando lo standard DIG.

Le principali funzionalità offerte da Racer sono:
- il controllo di consistenza delle classi dell'ontologia.
- la classificazione delle tassonomie

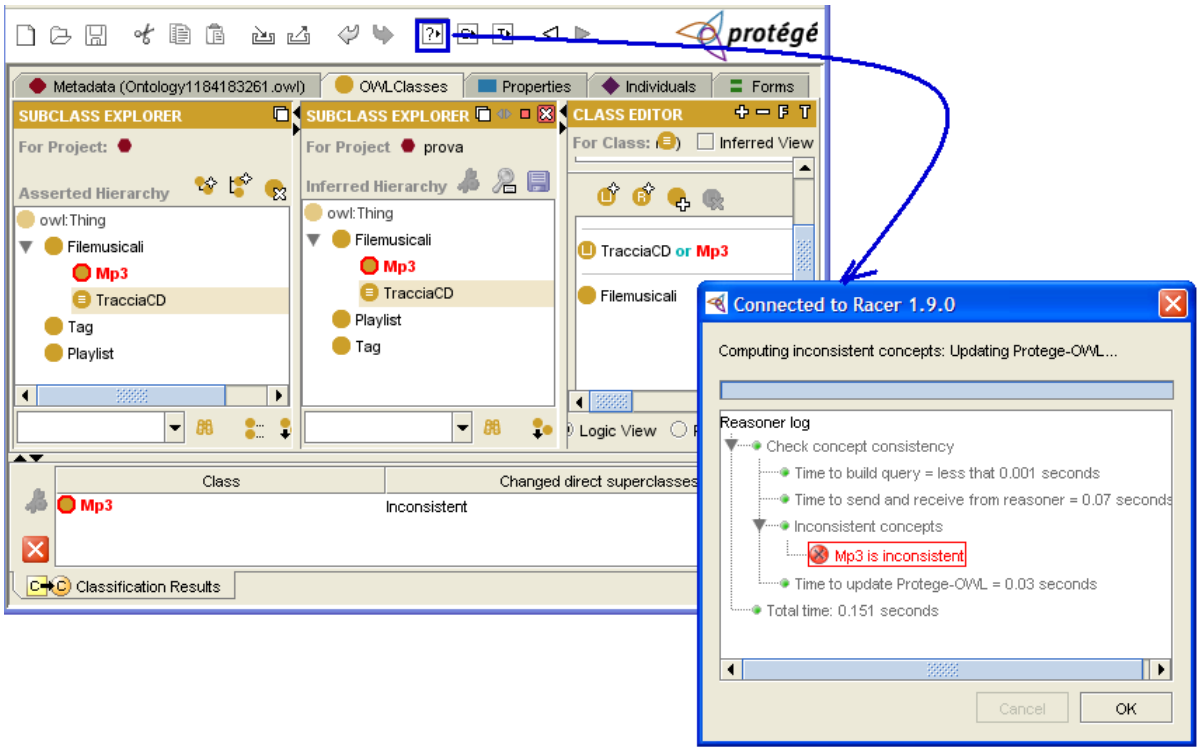


I pulsanti di Protégé per invocare i servizi del reasoner.

Una classe è detta consistente se possono esistere individui appartenenti ad essa.
Se un'ontologia ha delle classi inconsistenti, vuol dire che è stato fatto qualche errore in fase di modellazione che è opportuno effettuare delle correzioni.

Sulla barra degli strumenti di Protègè è presente il pulsante *Check Consistency* per eseguire il controllo di consistenza di un'ontologia, Protégé passa l'ontologia a Racer che la analizza e ne evidenzia eventuali classi inconsistenti.

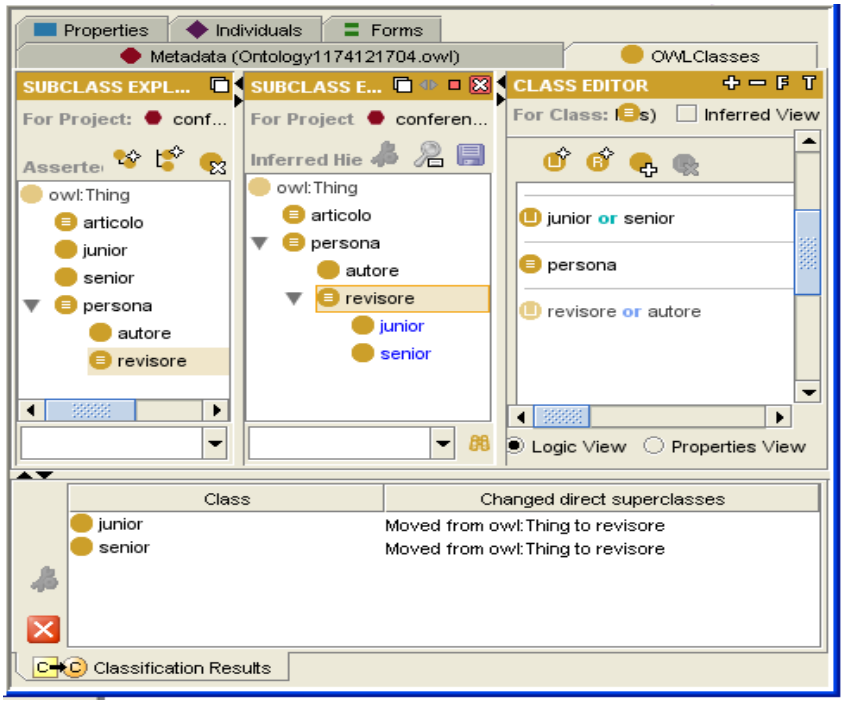
Esempio:
Per la classe TracciaCd è stato effettuato un errore nella definizione della condizione necessaria e sufficiente. Tramite il controllo della consistenza si evidenzia che la classe Mp3, disgiunta da essa, si rivela essere inconsistente, ovvero nessun elemento può appartenere a essa.



La classe MP3 risulta essere inconsistente.

Si è detto che una classe è detta primitiva se è descritta solo attraverso condizioni *necessarie*.
Le condizioni necessarie servono a creare restrizioni sugli individui che possono appartenere ad una classe. In altre parole esse servono ad asserire: "se un individuo appartiene a questa classe allora deve avere queste caratteristiche".
Una classe può essere descritta anche attraverso condizioni *necessarie e sufficienti*. Tali condizioni servono ad asserire: "se un individuo ha queste caratteristiche allora esso fa parte di questa classe".
Le classi definite sono classi descritte attraverso condizioni necessarie e sufficienti.
Una funzionalità che può essere utile per capire ancor meglio la differenza tra classi primitive e classi definite è la classificazione dell'ontologia.
Essa consente di ottenere la *gerarchia desunta* delle classi dell'ontologia che può essere diversa da quella dichiarata dal suo creatore.
Per eseguire la classificazione da Protégé è necessario premere il pulsante *Classif Taxonomy* sulla barra degli strumenti.
Protégé passa l'ontologia a Racer che la analizza e genera la gerarchia desunta che viene visualizzata accanto a quella dichiarata dal creatore.

Esempio:
La classe Junior e la classe Senior sono state definite tra le classi, ed è specificata la condizione necessaria e sufficienti che indica che la classe revisore è data dall'unione di queste due (queste però non sono state inserite come sue sottoclassi). Ciò che è implicito nelle condizioni delle due classi viene dedotto e reso esplicito dal reasoner.



Gerarchia asserita e desunta dell'ontologia della classe revisore.

6. Swoop – Sintassi Astratta

Swoop è un semplice, scalabile browser ed editor per ontologie OWL.
Esso usa URIs che sono i principali identificatori per ontologie, classi proprietà e individui per supportare la navigazione ipertestuale tra le ontologie.
Inoltre Swoop offre la possibilità di visualizzare le ontologie in varie formati da rappresentazioni sintattiche (Abstract Syntax) a presentazioni RDF/XLM.

In questa tesina Swoop è stato utilizzato per caricare i file .owl ottenuti dalla precedente traduzione da UML con Protégé (file .owl), e produrre un file .txt con la rappresentazione delle ontologie nella sintassi astratta. Nel seguente schema viene riportato un confronto tra la sintassi astratta e la corrispondente sintassi DL.

Abstract Syntax	DL Syntax
Class(A partial $C_1 \dots C_n$)	$A \sqsubseteq C_1 \sqcap \dots \sqcap C_n$
Class(A complete $C_1 \dots C_n$)	$A \equiv C_1 \sqcap \dots \sqcap C_n$
EnumeratedClass(A $o_1 \dots o_n$)	$A \equiv \{o_1\} \sqcup \dots \sqcup \{o_n\}$
SubClassOf(C_1 C_2)	$C_1 \sqsubseteq C_2$
EquivalentClasses($C_1 \dots C_n$)	$C_1 \equiv \dots \equiv C_n$
DisjointClasses($C_1 \dots C_n$)	$C_i \sqcap C_j \sqsubseteq \bot, i \not\equiv j$
Datatype(D)	
ObjectProperty(R super(R_1)...super(R_n)	$R \sqsubseteq R_i$
domain(C_1)...domain(C_m)	$\geqslant 1\, R \sqsubseteq C_i$
range(C_1)...range(C_ℓ)	$\top \sqsubseteq \forall R.C_i$
[inverseOf(R_0)]	$R \equiv R_0^-$
[Symmetric]	$R \equiv R^-$
[Functional]	$\top \sqsubseteq \leqslant 1\, R$
[InverseFunctional]	$\top \sqsubseteq \leqslant 1\, R^-$
[Transitive])	$Tr(R)$
SubPropertyOf(R_1 R_2)	$R_1 \sqsubseteq R_2$
EquivalentProperties($R_1 \dots R_n$)	$R_1 \equiv \dots \equiv R_n$
DatatypeProperty(U super(U_1)...super(U_n)	$U \sqsubseteq U_i$
domain(C_1)...domain(C_m)	$\geqslant 1\, U \sqsubseteq C_i$
range(D_1)...range(D_ℓ)	$\top \sqsubseteq \forall U.D_i$
[Functional])	$\top \sqsubseteq \leqslant 1\, U$
SubPropertyOf(U_1 U_2)	$U_1 \sqsubseteq U_2$
EquivalentProperties($U_1 \dots U_n$)	$U_1 \equiv \dots \equiv U_n$
AnnotationProperty(S)	
OntologyProperty(S)	
Individual(o type(C_1)...type(C_n)	$o \in C_i$
value(R_1 o_1)...value(R_n o_n)	$\langle o, o_i \rangle \in R_i$
value(U_1 v_1)...value(U_n v_n))	$\langle o, v_i \rangle \in U_i$
SameIndividual($o_1 \dots o_n$)	$\{o_1\} \equiv \dots \equiv \{o_n\}$
DifferentIndividuals($o_1 \dots o_n$)	$\{o_i\} \sqsubseteq \neg\{o_j\}, i \not\equiv j$

La sintassi astratta e la corrispondente sintassi DL

8. DL-LiteA

Un'ontologia permette di rappresentare il livello intensionale di un dominio applicativo fornendo concettualizzazioni comuni.
Oltre al livello dati di un sistema informativo, viene imposto un livello concettuale che permette di avere una vista concettuale delle informazioni del sistema astraendo come tali informazioni sono mantenute nel livello dati del sistema stesso.
Mentre le ontologie vengono utilizzate per realizzare il livello concettuale, i dbms relazionali vengono utilizzate per la gestione del livello dati.

Recenti ricerche di base sono state fatte per capire quali frammenti di OWL ,OWL-DL o OWL-Lite sia adatto con il formalismo per la rappresentazione delle ontologie in questo contesto.
Il risultato di questo lavoro è quello che nessuno delle varianti di OWL sia adatto, se non con delle restrizioni che garantiscano ragionamenti polinomiali
Tra queste logiche, maggior interesse è stato rivolto alla famiglia DL-lite che tiene conto di congiuntive query con complessità dei dati logspace e che tengono conto del DBMS che gestisce il livello dati.

Maggiormente, due linguaggi nella famiglia DL-lite posseggono questa proprietà.
Il Primo è DL-Lite F il quale permette di specificare le principali proprietà di modelli concettuali includendo cicliche asserzioni , Isa di concetti, inversa di ruoli, tipi di ruoli, obbligatorie partecipazioni a ruoli, e funzionali restrizioni di ruoli.
Il secondo è DL-lite R il quale è in grado di catturare completamente, (il DL frammento di) RDFS ed ha in aggiunta la capacità'di specificare obbligatorie partecipazioni a ruoli e disgiunzioni tra concetti e ruoli.
Il linguaggio ottenuto unendo le caratteristiche di DL-lite R e DL-lite F , non ha complessità dei dati logspace perciò perdendo la caratteristica computazionale più interessante.

Questa logica descrittiva DL-lite-FR, che combina le principali caratteristiche di due DLlite presentati DLlite-F e DLlite-R, pone però le basi per il linguaggio Dl-lite-A.

Tutte le logiche della famiglia DL-lite permettono di rappresentare l'universo di *discorsi* in termini di concetti denotando gruppi di oggetti e ruoli, denotando relazioni binarie tra oggetti.

In aggiunta DL-lite-FR permette di usare:

- valori-domini, domini concreti denotando set di valori
- attributi di concetto denotando relazioni binarie tra oggetti e valori
- attributi di ruolo denotando relazioni binarie tra coppie di oggetti e valori

Ovviamente, un attributo di ruolo può anche essere visto come una relazione ternaria che lega due oggetti e un valore.

Per specificare il linguaggio vengono utilizzate le seguenti notazioni :

- A denota concetti atomici, B concetti base, C concetti generali
- D denota un valore-dominio atomico, E un valore-dominio di base, F un valore-dominio generale
- P denota ruoli atomici, Q ruoli base, R ruoli generali
- U_C denota attributo di concetto atomico, V_C attributo concettuale generale
- U_R denota attributo di ruolo atomico, V_R attributo di ruolo generale
- T_C denota un concetto universale, T_D valore-dominio universale

Dato un attributo concettuale U_C (rispettivamente un attributo di ruolo U_R) viene chiamato dominio di Uc (risp. U_R), denotato con $\delta(U_C)$ (risp. $\delta(U_R)$), il set di oggetti che Uc (risp U_R), relaziona con valori e chiamiamo range di Uc (risp. U_R), denotato con $\rho(U_C)$ (risp. $\rho(U_R)$), il set di valori che Uc (risp. U_R) relaziona con gli oggetti.
Denotiamo che il con $\delta_F(U_C)$ (risp. $\delta_F(U_R)$)il set di oggetti che Uc (risp. U_R) relaziona con valori in un dominio dei valori F.

Le espressioni DL-liteFR sono definite come seguono:

-- <u>espressioni di concetto</u>
$B ::= A \mid \exists Q \mid \delta(U_C)$
$C ::= Tc \mid B \mid \neg B \mid \exists Q.C \mid \delta_F(U_C) \mid \exists \delta_F(UR) \mid \exists \delta_F(UR)^{-}$

-- <u>espressioni di value-domain</u>
$E ::= D \mid \rho(U_C) \mid \rho(U_R)$
$F ::= TD \mid E \mid \neg E \mid$ *rdfDataType*

-- <u>espressioni di attributo</u>
$Vc ::= Uc \mid \neg Uc$
$V_R ::= U_R \mid \neg U_R$

-- <u>espressioni di attributo</u>
$Q ::= P \mid \neg P \mid \delta(U_R) \mid \delta(U_R)^{-}$
$R ::= Q \mid \neg Q \mid \delta F(U_R) \mid \delta F(U_R)^{-}$

Una base di conoscenza (KB) K= <T, A> di DL-lite-FR è costituita da due componenti:
una TBox T usata per rappresentare la conoscenza definita ed immutabile sul dominio rappresentato, ovvero la conoscenza intensionale , e una ABox A usata per rappresentare la conoscenza relativa al problema attuale specifico, ovvero la conoscenza estensionale.

Le assersioni della TBox in DL-lite-FR sono della forma:

$B \sqsubseteq C$ asserzione di inclusione di concetto
$Q \sqsubseteq R$ asserzione di inclusione di ruolo
$E \sqsubseteq F$ asserzione di inclusione di value-domain
$U_C \sqsubseteq V_C$ asserzione di inclusione di attributo di concetto
$U_R \sqsubseteq V_R$ asserzione di inclusione di attributo di ruolo

(funct P) asserzione di funzionalità di ruolo
(funct P^{-}) asserzione di funzionalità di ruolo inverso
(funct U_C) asserzione di funzionalità di attributo di concetto
(funct U_R) asserzione di funzionalità di attributo di ruolo

Un *concetto di inclusione di asserzione* esprime che un concetto di base B e *subsunto* da un generale concetto C. Analogamente per gli altri tipi di inclusione asserzione.

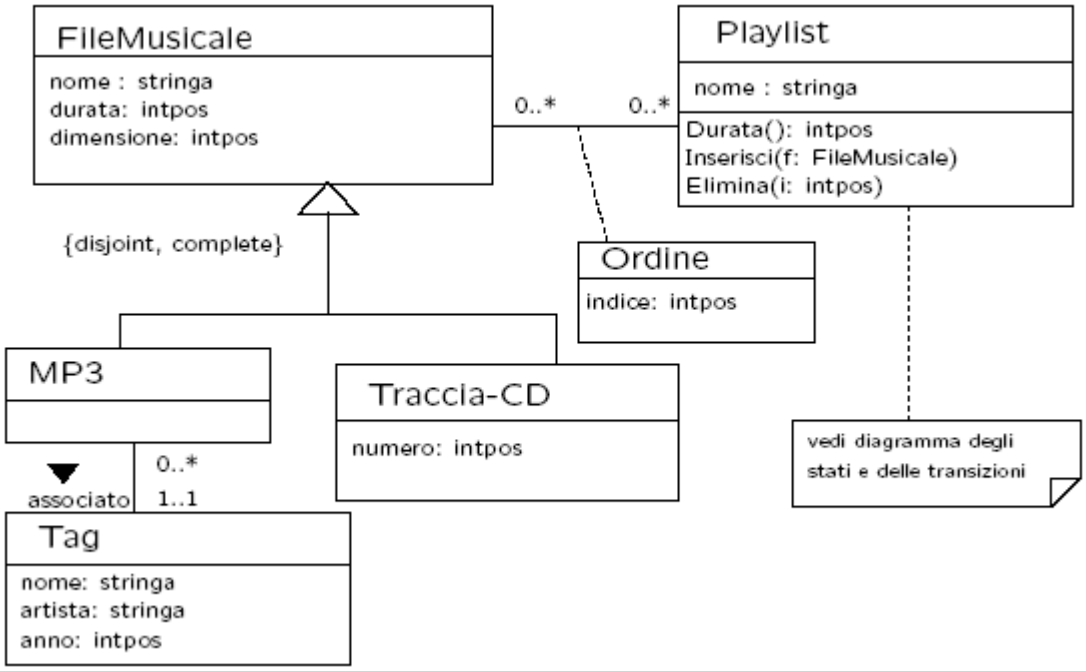
Per la ABox possiamo definire due alfabeti Γ_o e Γ_υ.
Γ_o sono chiamati costanti di oggetti e sono usati per denotare oggetti mentre i simboli in Γ_υ costanti di valori, usati per denotare valori dati.
La semantica in DL-liteFR è data in termini di interpretazioni FOL ($I = (\Delta^I, \cdot^I)$).

Una base di conoscenza DL-Lite-A è una coppia < T,A >, dove A è un DL-lite-FR ABox, e T è a DL-Lite- FR TBox che soddisfa le seguenti condizioni:

1) Per ogni ruolo atomico e inverso di un ruolo atomico Q compare in un concetto della forma $\exists Q.C$, le asserzioni (funct Q) e (funct Q^{-}) non sono in T
2) Per ogni asserzione di inclusione di ruolo $Q \sqsubseteq R$ in T , dove R è un ruolo atomico o l'inverso di un ruolo atomico, l'asserzione (funct R) e (funct R^{-}) non sono in T ;
3) Per ogni asserzione di inclusione di attributo di concetto $U_C \sqsubseteq V_C$ in T , dove V_C è un attributo di concetto atomico, l'asserzione (funct V_C) non sono in T ;
4) Per ogni asserzione di inclusione di attributo di ruolo $U_R \sqsubseteq V_R$ in T , dove Vr è un attributo di ruolo, le asserzioni (funct V_R) non sono in T .

Una DL-LiteA TBox impone la condizione che ogni ruolo funzionale non può essere specializzato usandolo sul lato destro del ruolo inclusione asserzione, la stessa condizione è anche imposta su ogni attributo funzionale(ruolo o concetto)
Può essere mostrato che funzionalità specifiche in una DL-LiteA TBox non sono implicitamente propagate nella TBox e questo permette LOGSPACE query answering.

Compito d'esame 1 Aprile 2005



OWL-DL

Classi: FileMusicali, Playlist, Mp3, TracciaCd,Tag
Proprietà: nome_filemusicali, durata, dimensione, numero, nome_tag, artista, anno, nome_playlist
Relazioni: associato, ordine

Non è possibile esprimere le operazioni Durata(), Inserisci(), Elimina() e l'attributo della relazione Indice.

CLASSI

- **File musicali**

L'oggetto "File Musicale" è stato tradotto in una classe Protègè che ha come proprietà gli attributi 'nome', 'durata', 'dimensione' tradotti come Dataproperty.
La classe File Musicale è superclasse nella generalizzazione ed ha come sottoclassi Mp3, TracciaCD (che sono tra di loro disgiunte).
La generalizzazione è disgiunta e completa e questo vincolo è stato espresso attraverso la restrizione TracciaCD U Mp3 inserita nelle condizioni necessarie e sufficienti.
La classe partecipa alla relazione 'inverse_of_ ordine' con la classe Playlist, con la cardinalità 0..* che è quella che viene indicata di default e perciò non è necessario indicarla attraverso restrizioni cardinali

Class(a: **Filemusicali** complete unionOf(a: **Mp3** a: **TracciaCD**))
DisjointClasses(a: **Tag** a: **Filemusicali**)
DisjointClasses(a: **Playlist** a: **Filemusicali**)

La classe Filemusicali è *complete:* è descritta utilizzando condizioni necessarie e sufficienti.
E' l'unione delle Mp3, TracciaCD ed è disgiunta dalle classi Tag e Playlist .

- **Mp3**

L'oggetto "Mp3" è stato tradotto in una classe Protègè la quale è una sottoclasse di FileMusicali da cui eredita le proprietà ed è disgiunta con la classe TracciaCD.
Queste classi sono tra di loro disgiunte e complete quindi un individuo che appartiene alla classe Mp3 non può appartenere alla classe TracciaCD.
La classe partecipa alla relazione 'associato' con la classe Tag, con cardinalità 1..1.
La cardinalità minima viene espressa tramite una restrizione di cardinlità mentre la cardinalità massima con proprietà Functional sulla relazione.

Class(a: **Mp3** complete
    restriction(a:associato someValuesFrom(a:Tag)))
Class(a: **Mp3** partial a: **Filemusicali**)
DisjointClasses (a: **Mp3** a: **TracciaCD**)

La classe Mp3 è definite *complete* perché ha tra le condizioni necessarie e sufficienti una restrizione ad indicare il vincolo di cardianiltà minima con il quale partecipa alla relazione 'associato' con la classe Tag.

La classe Mp3 è definita *partial* perché ha almeno una condizione necessaria inoltre è disgiunta dalla classe TracciaCD.

- **Traccia CD**

L'oggetto "TracciaCD" è stato tradotto in una classe Protègè che è una sottoclasse di FileMusicali da cui eredita le proprietà ed è disgiunta con la classe Mp3.
Queste classi sono tra di loro disgiunte e complete quindi un individuo che appartiene alla classe Mp3 non può appartenere alla classe TracciaCD.

Class(a: **TracciaCD** partial a: **Filemusicali**)
DisjointClasses(a: **Mp3** a: **TracciaCD**)

La classe TracciaCD è definita *partial* perché ha almeno una condizione necessaria inoltre è disgiunta dalla classe Mp3.

- **Playlist**

L'oggetto "Playlist" è stato tradotto in una classe Protègè che ha come proprietà l'attributo: nome tradotta come dataproperty con range string.
La classe partecipa alla relazione ordine con la cardinalità 0..* che è quella che viene indicata di default e perciò non è necessario indicarla attraverso restrizioni cardinali.

Class(a: **Playlist** partial)
DisjointClasses(a: **Playlist** a: **Filemusicali**)
DisjointClasses(a: **Tag** a: **Playlist**)

La classe Playlist è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalle classi Filemusicali e Tag .

- **Tag**

L'oggetto "Tag" è stato tradotto in una classe Protègè che ha come proprietà attributo: 'nome', 'artista' e 'anno' tradotti come Dataproperty.
La classe partecipa alla relazione 'inverse_of_asociato' con la classe Mp3, con la cardinalità 0..* che è quella che viene indicata di default e perciò non è necessario indicarla attraverso restrizioni cardinali.

Class(a:**Tag** partial)
DisjointClasses(a: **Tag** a: **Filemusicali**)
DisjointClasses(a: **Tag** a: **Playlist**)

La classe Tag è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalle classi Filemusicali e Playlist .

PROPRIETA' (DataProperty)

Gli attributi della classe sono stati tradotti come DataProperty e hanno come la proprietà Functional

poiché sono associati ad un solo valore per individuo.

- L'attributo nome ha come dominio la classe Filemusicali e come range string mentre gli attributi durata e dimensione hanno come dominio la classe Filemusicali e come range intero.

DatatypeProperty(a: **nome_filemusicale** Functional
domain(a: **Filemusicali**)
range(xsd: string))

DatatypeProperty(a: **dimensione** Functional
domain(a: **Filemusicali**)
range(xsd: int))

DatatypeProperty(a: **durata** Functional
domain(a: **Filemusicali**)
range(xsd: int)

- Gli attributi 'nome' e 'artista' hanno come dominio la classe Tag e come range string mentre l'attributo 'anno' ha come dominio la classe Tag e come range intero.

DatatypeProperty(a: **nome**_tag Functional
domain(a: **Tag**)
range(xsd: string))

DatatypeProperty(a: **artista** Functional
domain(a: **Tag**)
range(xsd: string))

DatatypeProperty(a: **anno** Functional
domain(a: **Tag**)
range(xsd: int))

- L'attributo 'nome' ha come dominio la classe Playlist e come range string

DatatypeProperty(a: **nome_playlist** Functional
domain(a: **Playlist**)
range(xsd: string))

- L'attributo 'numero' ha come dominio la classe TracciaCD e come range intero

DatatypeProperty(a: **numero** Functional
domain(a: **TracciaCD**)
range(xsd: int))

<u>RELAZIONI</u> (ObjectProperty)

Le associazioni sono state tradotte in relazioni tra le classi (ObjectProperty) e per ogni relazione è stata definita una relazione inversa:

ordine ←→inverse_of_ordine
associato←→inverse_of_associato

- **ordine**

L'associazione 'ordine' viene tradotta con un ObjectProperty che ha come dominio la classe Playlist come range la classe Filemusicali.

ObjectProperty(a:**ordine**
inverseOf(a:**inverse_of_ordine**)
domain(a:**Playlist**)
range(a:**Filemusicali**))

La sua relazione inversa è 'inverse_of_ordine' ha come dominio la classe Filemusicali e come range la classe Playlist.

ObjectProperty(a: **inverse_of_ordine**
inverseOf(a: **ordine**)
domain(a: **Filemusicali**)
range(a: **Playlist**))

- **associato**

L'associazione 'associato' viene tradotta con un ObjectProperty che ha come dominio la classe Mp3 come range la classe Tag ed è stata definita Functional per indicare il vincolo di cardinalità 1..1, per la cardinalità massima (se l'mp3 a ha come tag b, e se l'mp3 a ha come tag c, allora il tag b e il tag c coincidono.

ObjectProperty(a: **associato** Functional
inverseOf(a: **inverse_of_associato**)
domain(a: **Mp3**)
range(a: **Tag**))

La sua relazione inversa è inverse_of_ordine_da ha come dominio la classe Tag e come range la classe Mp3 e come proprietà inverseFunctional in quanto la relazione inversa è Functional.

ObjectProperty(a:**inverse_of_associato** InverseFunctional
inverseOf(a: **associato**)
domain(a: **Tag**)
range(a: **Mp3**))

DL-LiteA

Asserzioni di inclusione di concetti

Filemusicali ⊑ ¬ Playlist
Filemusicali ⊑ ¬ Tag
Playlist ⊆ ¬ Filemusicali
Playlist ⊑ ¬ Tag
Tag ⊑ ¬ Playlist
Tag ⊑ ¬ Filemusicali
TracciaCD ⊑Filemusicali
TracciaCD ⊑ ¬ Mp3
Mp3 ⊑ Filemusicali
Mp3 ⊑ ¬ TracciaCD

Filemusicali ⊑ δ(nome_filemusicale)
Filemusicali ⊑ δ(durata)
Filemusicali ⊑ δ(dimensione)
TracciaCD ⊑ δ(numero)
Playlist ⊑ δ(nome_playlist)
Tag ⊑ δ(nome_tag)
Tag ⊑ δ(artista)
Tag ⊑ δ(anno)

∃ ordine ⊑ Playlist
∃ $(\text{ordine})^{-}$ ⊑ Filemusicali
∃ inverse_of_ ordine ⊑ Filemusicali
∃ $(\text{inverse\_of\_ ordine})^{-}$ ⊑ Playlist
∃ associato ⊑ Mp3
∃ $(\text{associato})^{-}$ ⊑ Tag
∃ inverse_of_ associato ⊑ Tag
∃ $(\text{inverse\_of\_ associato})^{-}$ ⊑ Mp3

Asserzioni di funzionalità di attributi di concetto

(funct nome_filemusicale)
(funct durata)
(funct dimensione)
(funct numero)
(funct nome_playlist)
(funct nome_tag)
(funct artista)
(funct anno)

Asserzioni di inclusione di domini di valori

ρ(nome_filemusicale) ⊑ xs:string

ρ(durata) ⊑ xs:int
ρ(dimensione) ⊑ xs:int
ρ(numero) ⊑ xs:int
ρ(nome_playlist) ⊑ xs:string
ρ(nome_tag) ⊑ xs:string
ρ(artista) ⊑ xs:string
ρ(anno) ⊑ xs:int

Asserzioni di funzionalità di ruolo

(funct associato)

Asserzioni di inclusione di ruoli

ordine ⊑ (inverse_of_ordine)$^{-}$
inverse_of_ ordine ⊑ (ordine)$^{-}$
associato ⊑ (inverse_of_ associato)$^{-}$
inverse_of_ associato ⊑ (associato)$^{-}$

- **Alfabeto**

L'alfabeto che descrive lo schema UML è formato dai seguenti elementi:
da concetti generali (atomicC), attributi di concetti (atomicCA) , ruoli generali (atomicR).

<alphabet>

<atomicC> **Filemusicali** </atomicC>
<atomicC> **Mp3** </atomicC>
<atomicC> **TracciaCD** </atomicC>
<atomicC> **Tag** </atomicC>
<atomicC> **Playlist** </atomicC>

<atomicCA> **nome_filemusicale** </atomicCA>
<atomicCA> **dimensione** </atomicCA>
<atomicCA> **durata** </atomicCA>
<atomicCA> **numero** </atomicCA>
<atomicCA> **nome_tag** </atomicCA>
<atomicCA> **artista** </atomicCA>
<atomicCA> **anno** </atomicCA>
<atomicCA> **nome_playlist** </atomicCA>

<atomicR> **ordine** </atomicR>
<atomicR> **inverse_of_ordine** </atomicR>
<atomicR> **associato** </atomicR>
<atomicR> **inverse_of_associato** </atomicR>

</alphabet>

Le Classi Filemusicali, Mp3, TracciaCD, Tag, Playlist sono concetti generali.
Le proprietà delle classi nome_filemusicale, dimensione, durata, numero, nome_tag, artista, anno, nome_playlist sono attributi di concetto.
Le relazioni ordine, inverse_of_ordine, associato, inverse_of_ associato, sono ruoli generali.

• **Tbox**

La Tbox è usata per rappresentare la conoscenza intensionale attraverso assezioni di inclusione.

-- Filemusicali ⊑ ¬ Playlist

```
<inclusionAssertion>
        <basicC>
              <atomicC> Filemusicali </atomicC>
        </basicC>
        <generalC>
              <signedC sign="negative">
                  <basicC>
                        <atomicC> Playlist </atomicC>
                  </basicC>
              </signedC>
        </generalC>
  </inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni Filemusicali non è una Playlist (sign="**negative**" vuole indicare la negazione del concetto).

-- Playlist ⊑ ¬ Filemusicali

```
<inclusionAssertion>
        <basicC>
              <atomicC> Playlist </atomicC>
        </basicC>
        <generalC>
              <signedC sign="negative">
                  <basicC>
                        <atomicC> Filemusicali </atomicC>
                  </basicC>
              </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni Playlist non è un Filemusicali (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di inclusione di concetto (Filemusicali ⊑ ¬ Playlist , Playlist ⊑ ¬ Filemusicali) stanno ad indicare che i concetti Filemusicali e Playlist sono disgiunti.

-- Filemusicali ⊑ ¬ Tag

```
<inclusionAssertion>
        <basicC>
            <atomicC> Filemusicali </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Tag </atomicC>
                </basicC>
            </signedC>
        </generalC>
  </inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni Filemusicali non è un Tag (sign="**negative**" vuole indicare la negazione del concetto).

-- Tag ⊑ ¬ Filemusicali

```
<inclusionAssertion>
        <basicC>
            <atomicC> Tag </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Filemusicali </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni Tag non è un Filemusicali (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di inclusione di concetto (Filemusicali ⊑ ¬Tag, Tag ⊑ ¬ Filemusicali) stanno ad indicare che i concetti Tag e Filemusicali sono disgiunti.

-- Playlist ⊑ ¬ Tag

```
<inclusionAssertion>
        <basicC>
            <atomicC> Playlist </atomicC>
        </basicC>
        <generalC>
```

```
            <signedC sign="negative">
                <basicC>
                    <atomicC> Tag </atomicC>
                </basicC>
            </signedC>
        </generalC>
  </inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni Playlist non è un Tag (sign="**negative**" vuole indicare la negazione del concetto).

-- Tag ⊑ ¬ Playlist

```
<inclusionAssertion>
        <basicC>
            <atomicC> Tag </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Playlist </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni Tag non è un Playlist (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di inclusione di concetto (Playlist ⊑ ¬Tag, Tag ⊑ ¬ Playlist) stanno ad indicare che i concetti Tag e Playlist sono disgiunti.

-- TracciaCD ⊑ Filemusicali

```
<inclusionAssertion>
        <basicC>
          <atomicC> TracciaCD </atomicC>
        </basicC>
        <generalC>
          <signedC sign="positive">
            <basicC>
              <atomicC> Filemusicali </atomicC>
            </basicC>
          </signedC>
        </generalC>
</inclusionAssertion>
```

L’asserzione di inclusione vuole indicare che ogni concetto TracciaCD è un Filemusicali.

Questa asserzione sta ad indicare che la classe TracciaCD è una sottoclasse di Filemusicali.

-- Mp3 ⊑ Filemusicali

```
<inclusionAssertion>
        <basicC>
           <atomicC> Mp3 </atomicC>
        </basicC>
        <generalC>
           <signedC sign="positive">
              <basicC>
                 <atomicC> Filemusicali </atomicC>
              </basicC>
           </signedC>
        </generalC>
</inclusionAssertion>
```

L’asserzione di inclusione vuole indicare che ogni concetto Mp3 è una Filemusicali.

Questa asserzione sta ad indicare che la classe Mp3 è una sottoclasse di Filemusicali.

Queste asserzioni stanno ad indicare che i concetti generali Mp3 e TracciaCD sono derivati dal concetto di Filemusicali.

-- TracciaCD ⊑ ¬ Mp3

```
<inclusionAssertion>
        <basicC>
              <atomicC> TracciaCD </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                        <atomicC> Mp3 </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

L’asserzione di inclusione vuole indicare che ogni TracciaCD non è un Mp3 (sign="**negative**" vuole indicare la negazione del concetto).

-- Mp3 ⊑ ¬ TracciaCD

```
<inclusionAssertion>
```

```
        <basicC>
            <atomicC> Mp3 </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> TracciaCD </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni Mp3 non è una TracciaCD (sign="**negative**" vuole indicare la negazione del concetto).

*Queste due asserzioni di inclusione di concetto (Mp3 ⊑ ¬ TracciaCD **,** TracciaCD ⊑ ¬ Mp3) stanno ad indicare che i concetti generali Mp3 e TracciaCD sono tra di loro disguinti.*

-- funct(nome_filemusicale)

```
<funct>
        <atomicCA> nome_filemusicale </atomicCA>
</funct>
```

L'attributo di concetto, 'nome_filemusicale**'**, è stato definito funzionale.

-- Filemusicali ⊑ δ(nome_filemusicale)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Filemusicali </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> nome_filemusicale </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Filemusicali ha associato un attributo di concetto 'nome_filemusicale'.
Dato un attributo di concetto 'nome_filemusicale' chiamiamo dominio di 'nome_filemusicale' denotato con δ(nome_filemusicale) il set di oggetti che 'nome_filemusicale' collega a valori.

-- ρ(nome_filemusicale) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> nome_filemusicale </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell'attributo di concetto 'nome_filemusicale' è un dominio di valori predefinito xs:string.

-- funct(durata)

```
<funct>
    <atomicCA> durata </atomicCA>
</funct>
```

L'attributo di concetto, 'durata**'**, è stato definito funzionale.

-- Filemusicali ⊑ δ(durata)

```
<inclusionAssertion>
    <basicC>
        <atomicC> Filemusicali </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> durata </atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto Filemusicali ha associato un attributo di concetto 'durata'.
Dato un attributo di concetto 'durata' chiamiamo dominio di 'durata' denotato con δ(durata) il set di oggetti che 'durata' collega a valori.

-- ρ(durata) ⊑ xs:int

```
<inclusionAssertion>
    <basicV>
```

```
            <ARange>
                <atomicCA> durata </atomicCA>
            </ARange>
    </basicV>
    <generalV>
            <predefinedV> xs:int </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell'attributo di concetto 'durata' è un dominio di valori predefinito xs:int.

-- funct(dimensione)

```
<funct>
        <atomicCA> dimensione </atomicCA>
</funct>
```

L'attributo di concetto, 'dimensione**'**, è stato definito funzionale.

-- Filemusicali ⊑ δ(dimensione)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Filemusicali </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> dimensione </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Filemusicali ha associato un attributo di concetto 'dimensione'.
Dato un attributo di concetto 'dimensione' chiamiamo dominio di 'dimensione' denotato con δ('dimensione') il set di oggetti che 'dimensione' collega a valori.

-- ρ(dimensione) ⊑ xs:int

```
<inclusionAssertion>
    <basicV>
            <ARange>
                <atomicCA> dimensione </atomicCA>
            </ARange>
    </basicV>
```

```
    <generalV>
        <predefinedV> xs:int </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell'attributo di concetto 'dimensione' è un dominio di valori predefinito xs:int.

-- funct(numero)

```
<funct>
    <atomicCA> numero </atomicCA>
</funct>
```

L'attributo di concetto, 'numero', è stato definito funzionale.

-- TracciaCD ⊑ δ(numero)

```
<inclusionAssertion>
    <basicC>
        <atomicC> TracciaCD </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> numero </atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto TracciaCD ha associato un attributo di concetto 'numero'.
Dato un attributo di concetto 'numero' chiamiamo dominio di 'numero' denotato con δ(numero) il set di oggetti che 'numero' collega a valori.

-- ρ(numero) ⊑ xs:int

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> numero </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:int </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell'attributo di concetto 'numero' è un dominio di valori predefinito xs:int.

-- funct(nome_playlist)

```
<funct>
        <atomicCA> nome_playlist </atomicCA>
</funct>
```

L'attributo di concetto, 'nome_playlist', è stato definito funzionale.

-- Playlist ⊑ δ(nome_playlist)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Playlist </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> nome_playlist </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Playlist ha associato un attributo di concetto 'nome_playlist'.
Dato un attributo di concetto 'nome' chiamiamo dominio di 'nome_playlist' denotato con δ(nome_playlist) il set di oggetti che 'nome_playlist' collega a valori.

-- ρ(nome_playlist) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> nome_playlist </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell'attributo di concetto 'nome_playlist' è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di 'nome_playlist' chiamiamo range di 'nome_list' denotato con ρ(nome_list) il set di valori che 'nome_list' collega a oggetti.

-- funct(nome_tag)

```
<funct>
        <atomicCA> nome_tag </atomicCA>
</funct>
```

L'attributo di concetto, 'nome_tag', è stato definito funzionale.

-- Tag ⊑ δ(nome_tag)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Tag </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> nome_tag </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Tag ha associato un attributo di concetto 'nome_tag'.
Dato un attributo di concetto 'nome' chiamiamo dominio di 'nome_tag' denotato con δ(nome_tag) il set di oggetti che 'nome_tag' collega a valori.

-- ρ(nome_tag) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> nome_tag </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell'attributo di concetto 'nome_tag' è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di 'nome_tag' chiamiamo range di 'nome_tag' denotato con ρ(nome_tag) il set di valori che 'nome_tag' collega a oggetti.

-- funct(artista)

```
<funct>
        <atomicCA> artista </atomicCA>
</funct>
```

L'attributo di concetto, 'artista', è stato definito funzionale.

-- Tag ⊑ δ(artista)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Tag </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> artista </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Tag ha associato un attributo di concetto 'artista'.
Dato un attributo di concetto 'nome' chiamiamo dominio di 'artista' denotato con δ(artista) il set di oggetti che 'artista' collega a valori.

-- ρ(artista) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> artista </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell'attributo di concetto 'artista' è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di 'artista' chiamiamo range di 'artista' denotato con ρ(artista) il set di valori che 'artista' collega a oggetti.

-- funct(anno)

```
<funct>
        <atomicCA> anno </atomicCA>
</funct>
```

L'attributo di concetto, 'anno', è stato definito funzionale.

-- Tag ⊑ δ(anno)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Tag </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> anno </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Tag ha associato un attributo di concetto 'anno'.
Dato un attributo di concetto 'nome' chiamiamo dominio di 'anno' denotato con δ(anno) il set di oggetti che 'anno' collega a valori.

-- ρ(anno) ⊑ xs:int

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> anno </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:int </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell'attributo di concetto 'anno' è un dominio di valori predefinito xs:int.
Dato un attributo di concetto di 'anno' chiamiamo range di 'anno' denotato con ρ(anno) il set di valori che 'anno' collega a oggetti.

-- **ordine ⊑ (inverse_of_ordine)⁻**

```
<inclusionAssertion>
        <basicR dir="direct">
                <atomicR> ordine </atomicR>
        </basicR>
        <generalR>
                <signedR sign="positive">
                    <basicR dir="inverse">
                            <atomicR> inverse_of_ordine </atomicR>
                    </basicR>
                </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘ordine’ è un’istanza del ruolo inverso di ‘ inverse_of_ordine’.

Questa asserzione di inclusione di ruolo (ordine ⊑ (inverse_of_f ordine)⁻) sta ad indicare che il ruolo ‘inverse_of_ordine’ è la relazione inversa della relazione ‘ordine’.

-- **inverse_of_ ordine ⊑ (ordine)⁻**

```
<inclusionAssertion>
        <basicR dir="direct">
                <atomicR> inverse_of_ordine </atomicR>
        </basicR>
        <generalR>
                <signedR sign="positive">
                        <basicR dir="inverse">
                                <atomicR> ordine </atomicR>
                        </basicR>
                </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ ordine’ è un’istanza del ruolo inverso di ‘ordine’.

Questa asserzione di inclusione di ruolo (inverse_of_ ordine ⊑ (ordine)⁻)sta ad indicare che il ruolo ‘ordine’ è la relazione inversa della relazione ‘inverse_of_ordine’.

-- **∃ ordine ⊑ Playlist**

```
<inclusionAssertion>
        <basicC>
                <exists>
                    <basicR dir="direct">
                            <atomicR> ordine </atomicR>
```

```
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Playlist </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘ordine’ è un’ istanza del concetto Playlist.

-- ∃ (ordine)¯ ⊑ Filemusicali

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> ordine </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Filemusicali </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso ‘ordine’ è un’ istanza del concetto Filemusicali.

-- ∃ inverse_of_ ordine ⊑ Filemusicali

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> inverse_of_ordine </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
```

```
                <basicC>
                    <atomicC> Filemusicali </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_ordine’ è un’ istanza del concetto Filemusicali.

-- ∃ (inverse_of_ ordine)¯ ⊑ Playlist

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> inverse_of_ordine </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Playlist </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso ‘inverse_of_ordine’ è un’ istanza del concetto Playlist.

-- funct(associato)

```
<funct>
        <basicR dir="direct">
            <atomicR> associato </atomicR>
        </basicR>
</funct>
```

Il ruolo di concetto, ‘associato’, è stato definito funzionale.

Questa asserzione di funzionalità di ruolo (funct associato) sta ad indicare la cardinalità (1,1) della relazione associato.

-- associato ⊑ (inverse_of_ associato)¯

```
<inclusionAssertion>
        <basicR dir="direct">
```

```
            <atomicR> associato </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> inverse_of_associato </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘associato’ è un’istanza del ruolo inverso di ‘ inverse_of_ associato’.

Questa asserzione di inclusione di ruolo (assistito ⊑ (inverse_of_assistito)⁻)sta ad indicare che il ruolo ‘inverse_of_assistito’ è la relazione inversa della relazione ‘assistito’.

-- inverse_of_ associato ⊑ (associato)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> inverse_of_associato </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> associato </atomicR>
                </basicR>
            </signedR>
        </generalR>
    </inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ associato’ è un’istanza del ruolo inverso di ‘associato’.

Questa asserzione di inclusione di ruolo (inverse_of_assistito ⊑ (assistito)⁻)sta ad indicare che il ruolo ‘assistito’ è la relazione inversa della relazione invers di ‘inverse_of_assistito’.

-- ∃ associato ⊑ Mp3

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> associato </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
```

```
            <signedC sign="positive">
                <basicC>
                    <atomicC> Mp3 </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘associato’ è un’ istanza del concetto Mp3.

-- ∃ (associato)¯ ⊑ Tag

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> associato </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Tag </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso ‘associato’ è un’ istanza del concetto Tag.

-- ∃ inverse_of_ associato ⊑ Tag

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> inverse_of_associato </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Tag</atomicC>
                </basicC>
             </signedC>
```

```
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_associato’ è un’ istanza del concetto Tag.

-- ∃ (inverse_of_ associato)⁻ ⊑ Mp3

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                        <atomicR> inverse_of_associato </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Mp3</atomicC>
                </basicC>
             </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘inverse_of_associato’ è un’ istanza del concetto Mp3.

Conclusioni:

1) *<!-- UnionOf not totally expressible in DL-liteA -->*

   La classe Filemusicali è la generalizzazione completa e disgiunta delle classi Mp3 eTracciaCD Espressa in DL-lite con l’espressione:

   Class (a: Filemusicali complete
   unionOf(a: Mp3 a: TracciaCD))

   In DL-liteA possimo esprimere le proprietà di disgiunzione tramite la seguente asserzione di inclusione di concetti: TracciaCD ⊑ ¬ Mp3, Mp3 ⊑ ¬ TracciaCD e le proprietà di completezza con le seguenti asserzioni di inclusione di concetti Mp3 ⊑ Filemusicali
   L’unione di queste tre classi non è completamente esprimibile in DL-liteA.

2) *Attributo di ruolo in DL-liteA*

   In DL-liteA è possibile esprimere attributi di ruolo denotando la relazione binaria tra coppie di oggetti e valori (non esprimibile in DL-lite).
   La relazione ordine presenta un attributo di relazione ‘indice’ di tipo intero.

Questo può essere espresso attraverso un attrobuto di ruolo ‘indice’ per il ruolo ‘ordine’.
Le espressioni sono le seguenti:

Asserzione funzionale di attributo di ruolo:

(funct indice)

L’attributo di ruolo indice è funzionale

Asserzioni di inclusione di concetti:

Playlist $\sqsubseteq \exists \delta(\text{indice})$
Filemusicali $\sqsubseteq \exists \delta(\text{indice})^{-}$

Ogni Playlist deve partecipare al ruolo ‘ordine’ (dominio di indice)avendo associato un attributo di ruolo ‘indice’.
Dato un attributo di ruolo ‘indice’ chiamiamo dominio di ‘indice’ denotato con δ(indice) il set di oggetti che ‘indice’ collega a valori.
Ogni Filemusicali deve partecipare al ruolo ‘inverse_of_ordine’ (dominio inverso di indice) avendo associato un attributo di ruolo ‘indice’.

Asserzioni di inclusione di ruolo:

$\delta(\text{indice}) \sqsubseteq \text{ordine}$
$\delta(\text{indice})^{-} \sqsubseteq \text{inverse\_of\_ordine}$

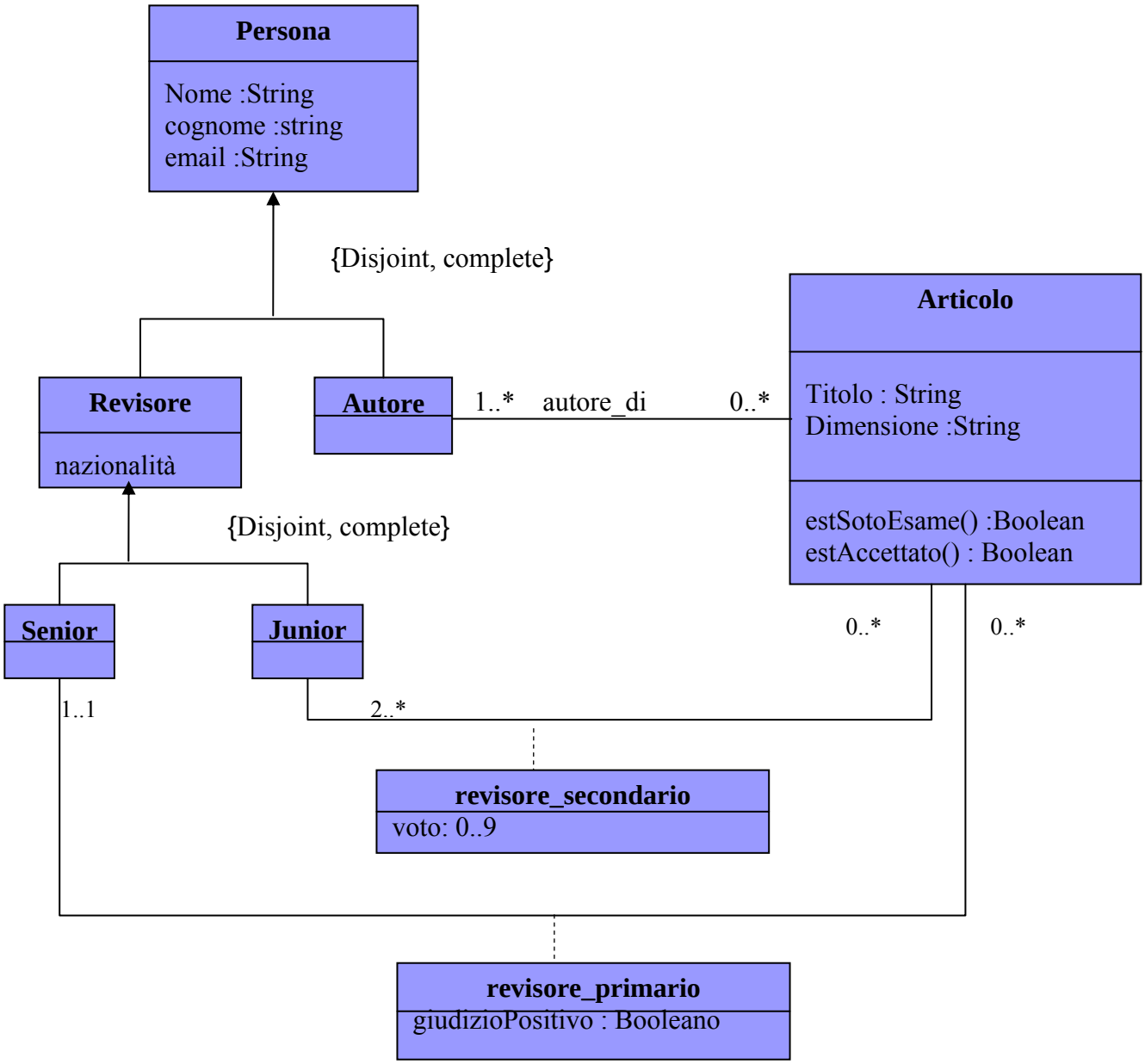
Il dominio dell’attributo di ruolo ‘indice’ è il ruolo ‘ordine’
Il dominio inverso dell’attributo di ruolo ‘indice’ è il ruolo ‘inverse_of_ordine’

Asserzioni di inclusione di dominio di valori:

$\rho(\text{indice}) \sqsubseteq \text{xs:int}$

Il range dell’attr

Compito d'esame 15 Aprile 2005



OWL-DL

Classi : Persona, Revisore, Autore, Senior, Junior, Articolo
Relazioni: autore_di, revisore_primario, revisore_secondario.
Proprietà: nome, cognome, email, nazionalità, titolo, dimensione.

Non è possibile esprimere le operazioni estSottoEsame(), estAccettato() e gli attributi di associazione voto e giudizioPositivo.

CLASSI

- **Persona**

L'oggetto "Persona" è stato tradotto in una classe Protègè che ha come proprietà gli attributi della classe: nome, cognome, email.
La classe Persona è superclasse nella generalizzazione che ha come sottoclassi Revisore, Autore (che sono tra di loro disgiunte).
La generalizzazione è completa e questo vincolo è stato espresso attraverso la restrizione : Revisore U Autore inserita nelle condizioni necessarie e sufficienti.

Class(a:**Persona** complete unionOf(a:**Autore** a:**Revisore**))
DisjointClasses(a:**Persona** a:**Articolo**)

La classe Persona è *complete:* è descritta utilizzando condizioni necessarie e sufficienti.
E' l'unione della classe Autore e Revisore ed è disgiunta dalla classe Articolo.

- **Revisore**

L'oggetto "Revisore" è stato tradotto in una classe Protègè ed è una sottoclasse di Persona da cui eredita le proprietà ed è disgiunta con la classe Autore.
Queste due classi sono tra di loro disgiunte e complete quindi un individuo che appartiene alla classe Revisore non può appartenere alla classe Autore.
Inoltre ha come sua proprietà l'attributo della classe: Nazionalità.
La classe "Revisore" è superclasse nella generalizzazione che ha come sottoclassi Junior e Senior che sono tra di loro disgiunte.
La generalizzazione è completa e questo vincolo è stato espresso attraverso la restrizione : Junior U Senior inserita nelle condizioni necessarie e sufficienti.

Class(a: **Revisore** complete unionOf(a: **Senior** a: **Junior**))
Class(a: **Revisore** partial a: **Persona**)
DisjointClasses(a:**Autore** a:**Revisore**)

La classe Revisore è definita *complete* perché è descritta utilizzando condizioni necessarie e sufficienti ed è l'unione delle classi Senior e Junior.
La classe Revisore è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalla classe Autore.

- **Autore**

L'oggetto "Autore" è stato tradotto in una classe Protègè ed è una sottoclasse di Persona da cui eredita le proprietà ed è disgiunta con la classe Revisore.
Queste due classi sono tra di loro disgiunte e complete quindi un individuo che appartiene alla classe Revisore non può appartenere alla classe Autore.
La classe partecipa alla relazione autore_di con cardinalità 0..* (un autore può scrivere da 0 a n articoli).
La cardinalità 0..* è quella che viene indicata di default e perciò non è necessario indicarla attraverso restrizioni cardinali.

Class(a:**Autore** partial a:**Persona**)
DisjointClasses(a:**Autore** a:**Revisore**)

La classe Autore è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalla classe Revisore.

- **Senior:**

L'oggetto "Senior" è stato tradotto in una classe Protègè ed è una sottoclasse di Revisore (sottoclasse di Persona) da cui eredita le proprietà ed è disgiunta con la classe Junior.
Queste due classi sono tra di loro disgiunte e complete quindi un individuo che appartiene alla classe Senior non può appartenere alla classe Junior.
La classe partecipa alla relazione revisore_primario con cardinalità 0..* (un revisore-senior può essere revisore primario di 0 a n articoli).
La cardinalità 0..* è quella che viene indicata di default e perciò non è necessario indicarla attraverso restrizioni cardinali.

Class(a: **Senior** partial a: **Revisore**)
DisjointClasses(a:**Junior** a:**Senior**)

La classe Senior è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalla classe Junior.

- **Junior:**

L'oggetto "Junior" è stato tradotto in una classe Protègè ed è una sottoclasse di Revisore (sottoclasse di Persona) da cui eredita le proprietà ed è disgiunta con la classe Senior.
Queste due classi sono tra di loro disgiunte e complete quindi un individuo che appartiene alla classe Junior non può appartenere alla classe Senior.
La classe partecipa all'associazione revisore_secondario con cardinalità 0..* (un revisore-secondario può essere revisore secondario di 0 a n articoli).
La cardinalità 0..* è quella che viene indicata di default e perciò non è necessario indicarla attraverso restrizioni cardinali.

Class(a: **Junior** partial a: **Revisore**)
DisjointClasses(a:**Junior** a:**Senior**)

La classe Junior è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalla classe Senior.

- **Articolo:**

L'oggetto "Articolo" è stato tradotto in una classe Protègè che ha come proprietà gli attributi della classe: titolo e dimensione.
La classe partecipa alla relazione inverse_of_autore_di con la classe Autore con cardinalità 1..* (ogni articolo deve avere almeno un Autore), inverse_of_revisore_primario con la classe Senior con cardinalità 1..1 (ogni articolo deve avere un revisore-senior), inverse_of_revisore_secondario con la classe Junior con cardinalità 2..* (ogni articolo deve avere almeno 2 revisori-junior).

Class(a:**Articolo** complete
intersectionOf(
restriction(a:inverse_of_revisore_primario minCardinality(1)) -
restriction(a:inverse_of_autore_di minCardinality(1))
restriction(a:inverse_of_revisore_secondario minCardinality(2))))

La classe Articolo è una classe *complete:* è descritta utilizzando condizioni necessarie e sufficienti. Alla classe vengono poste delle restrizioni di cardinalità per esprimere le cardinalità minime con cui partecipa alle relazioni.

PROPRIETA' (DataProperty)

Gli attributi della classe sono stati tradotti come DataProperty e hanno come la proprietà Functional poiché sono associati ad un solo valore per ogni individuo.

- Gli attributi 'nome', 'cognome', 'email' della classe vengono tradotti come DatatypeProperty il cui dominio è Persona e il range è string e sono stati posti functional.

DatatypeProperty(a:**nome** Functional
domain(a:**Persona**)
range(xsd:string))

DatatypeProperty(a:**cognome** Functional
domain(a:**Persona**)
range(xsd:string))

DatatypeProperty(a:**email** Functional
domain(a:**Persona**)
range(xsd:string))

- La classe Autore eredita gli attributi della superclasse Persona ed ha come attributo 'nazionalità' che ha come dominio la classe Revisore e come range string ed è stato posto Functional

DatatypeProperty(a:**nazionalità** Functional
domain(a:**Revisore**)
range(xsd:string))

- L'attributo 'titolo' ha come dominio la classe Articolo e range string, l'attributo dimensione ha come dominio la classe Articolo e range float ed sono stati posti Functional

DatatypeProperty(a:**titolo** Functional

domain(a:**Articolo**)
range(xsd:string))

DatatypeProperty(a:**dimensione** Functional
domain(a:**Articolo**)
range(xsd:float))

RELAZIONI(ObjectProperty)

Le associazioni sono state tradotte in relazioni tra le classi (ObjectProperty) e per ogni relazione è stata definita una relazione inversa:

autore_di ←→ inverse_of_autore_di
revisore_primario ←→ inverse_of_revisore_primario
revisore_secondario ←→ inverse_of_revisore_secondario

Le associazioni revisore_primario e revisore_secondario hanno entrambe un attributo (voto, giudizioPositivo) che non può essere tradotto come Datatype di ObjectProperty e questo non è possibile in OWL.

- **autore_di:**

L'associazione autore_di viene tradotta con un ObjectProperty che ha come dominio la classe Autore e come range la classe Articolo.

ObjectProperty(a:**autore_di**
inverseOf(a:inverse_of_autore_di)
domain(a:**Autore**)
range(a:**Articolo**))

La sua relazione inversa è inverse_of_autore_di ha come dominio la classe Articolo e come range la classe Autore.

ObjectProperty(a:**inverse_of_autore_di**
inverseOf(a:autore_di)
domain(a:**Articolo**)
range(a:**Autore**))

- **revisore_primario:**

L'associazione revisore_primario viene tradotta con un ObjectProperty che ha come dominio la classe Senior e come range la classe Articolo e come proprietà inverseFunctional in quanto la relazione inversa è Functional.

ObjectProperty(a:**revisore_primario InverseFunctional**
inverseOf(a:inverse_of_revisore_primario)
domain(a:**Senior**)
range(a:**Articolo**))

La sua relazione inversa è inverse_of_ revisore_primario ha come dominio la classe Articolo e come range la classe Senior ed è stata definita Functional per indicare il vincolo di cardinalità per il limite superiore (se l'articolo a ha come revisore primario b, e se l'articolo a ha come revisore primario c, allora il revisore primario b e c coincidono).

ObjectProperty(a:**inverse_of_revisore_primario** Functional
inverseOf(a:revisore_primario)
domain(a:**Articolo**)
range(a:**Senior**))

- **revisore_secondario:**

L'associazione revisore_secondario viene tradotta con un ObjectProperty che ha come dominio la classe Junior e come range la classe Articolo.

ObjectProperty(a:**revisore_secondario**
inverseOf(a:inverse_of_revisore_secondario)
domain(a:**Junior**)
range(a:**Articolo**))

La sua relazione inversa è inverse_of_ revisore_secondario ha come dominio la classe Junior e come range la classe Articolo.

ObjectProperty(a:inverse**_of_revisore_secondario**
inverseOf(a:revisore_secondario)
domain(a:**Articolo**)
range(a:**Junior**))

DL-LiteA

Asserzioni di inclusione di concetti

Articolo ⊑ ¬ Persona
Persona ⊑ ¬ Articolo
Revisore ⊑ Persona
Revisore ⊑ ¬ Autore
Autore ⊑ Persona
Autore ⊑ ¬ Revisore
Junior ⊑ Revisore
Junior ⊑ ¬ Senior
Senior ⊑ Revisore
Senior ⊑ ¬ Junior

Persona ⊑ δ(nome)
Persona ⊑ δ(cognome)
Persona ⊑ δ(email)
Revisore ⊑ δ(nazionalità)
Articolo ⊑ δ(titolo)
Articolo ⊑ δ(dimensione)

∃ autore_di ⊑ Autore
∃ (autore_di)$^{-}$ ⊑ Articolo
∃ inverse_of_ autore_di ⊑ Articolo
∃ (inverse_of_ autore_di)$^{-}$ ⊑ Autore

∃ revisore_primario ⊑ Senior
∃ (revisore_primario)$^{-}$ ⊑ Articolo
∃ inverse_of_ revisore_primario ⊑ Articolo
∃ (inverse_of_ revisore_primario)$^{-}$ ⊑ Senior

∃ revisore_secondario ⊑ Junior
∃ (revisore_secondario)$^{-}$ ⊑ Articolo
∃ inverse_of_ revisore_secondario ⊑ Articolo
∃ (inverse_of_ revisore_secondario)$^{-}$ ⊑ Junior

Articolo ⊑ ∃ inverse_of_autore_di. Autore
Articolo ⊑ ∃ inverse_of_revisore_primario. Senior

Asserzioni di funzionalità di attributi di concetto
(funct nome)
(funct cognome)
(funct email)
(funct nazionalità)
(funct titolo)
(funct dimensione)

Asserzioni di funzionalità di ruolo

(funct inverse_of_revisore_primario)

Asserzioni di inclusione di domini di valori

ρ(nome) ⊑ xs:string
ρ(cognome) ⊑ xs:string
ρ(email) ⊑ xs:string
ρ(nazionalità) ⊑ xs:string
ρ(titolo) ⊑ xs:string
ρ(dimensione) ⊑ xs:float

Asserzioni di inclusione di ruoli

autore_di ⊑ (inverse_of_ autore_di)$^{-}$
inverse_of_ autore_di ⊑ (autore_di)$^{-}$
revisore_primario ⊑ (inverse_of_ revisore_primario)$^{-}$
inverse_of_ revisore_primario ⊑ (revisore_primario)$^{-}$
revisore_secondario ⊑ (inverse_of_ revisore_secondario)$^{-}$
inverse_of_ revisore_secondario ⊑ (revisore_secondario)$^{-}$

- **Alfabeto**

L'alfabeto che descrive lo schema UML è formato dai seguenti elementi:
da concetti generali (atomicC), attributi di concetti (atomicCA) , ruoli generali (atomicR).

<alphabet>

<atomicC> **Persona** </atomicC>
<atomicC> **Revisore** </atomicC>
<atomicC> **Autore** </atomicC>
<atomicC> **Junior** </atomicC>
<atomicC> **Senior** </atomicC>
<atomicC> **Articolo** </atomicC>

<atomicCA> **nome** </atomicCA>
<atomicCA> **cognome** </atomicCA>
<atomicCA> **email** </atomicCA>
<atomicCA> **nazionalità** </atomicCA>
<atomicCA> **dimensione** </atomicCA>
<atomicCA> **titolo** </atomicCA>

<atomicR> **autore_di** </atomicR>
<atomicR> **inverse_of_autore_di** </atomicR>
<atomicR> **revisore_primario** </atomicR>
<atomicR> **inverse_of_revisore_primario** </atomicR>
<atomicR> **revisore_secondario** </atomicR>
<atomicR> **inverse_of_revisore_secondario** </atomicR>

</alphabet>

Le Classi Persona, Revisore, Autore, Junior, Senior, Articolo sono concetti generali.
Le proprietà delle classi nome, cognome, email, nazionalità, dimensione, titolo sono attributi di concetto.
Le relazioni autore_di, inverse_of_autore_di, revisore_primario, inverse_of_ revisore_primario, revisore secondario, inverse_of_ revisore secondario sono ruoli generali.

• **Tbox**

La Tbox è usata per rappresentare la conoscenza intensionale attraverso assezioni di inclusione.

-- Articolo ⊑ ¬ Persona

```
<inclusionAssertion>
        <basicC>
            <atomicC> Articolo </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Persona </atomicC>
                </basicC>
            </signedC>
        </generalC>
 </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Articolo non è una Persona (sign="**negative**" vuole indicare la negazione del concetto).

-- Persona ⊑ ¬ Articolo

```
<inclusionAssertion>
        <basicC>
            <atomicC> Persona </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Articolo </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Persona non è un Articolo (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni (Articolo ⊑ ¬Persona , Persona ⊑ ¬ Articolo) stanno ad indicare che i concetti Articolo e Persona sono disgiunti.

-- Revisore ⊑ Persona

```
<inclusionAssertion>
        <basicC>
           <atomicC> Revisore </atomicC>
        </basicC>
        <generalC>
           <signedC sign="positive">
              <basicC>
                 <atomicC> Persona </atomicC>
              </basicC>
           </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni concetto Revisore è una Persona.

Questa asserzione di inclusione di concetto sta ad indicare che la classe Revisore è una sottoclasse di Persona.

-- Autore ⊑ Persona

```
<inclusionAssertion>
        <basicC>
              <atomicC> Autore </atomicC>
        </basicC>
        <generalC>
           <signedC sign="positive">
                <basicC>
                     <atomicC> Persona </atomicC>
                </basicC>
           </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Revisore è una Persona.

Questa asserzione di inclusione di concetto sta ad indicare che la classe Autore è una sottoclasse di Persona.

Queste asserzioni (Revisore ⊑ Persona e Autore ⊑ Persona) stanno ad indicare che i concetti generali Autore e Revisore sono derivati dal concetto di Persona.

-- Revisore ⊑ ¬ Autore

```
<inclusionAssertion>
        <basicC>
            <atomicC> Revisore </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Autore </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Revisore non è un Autore (sign="**negative**" vuole indicare la negazione del concetto).

-- Autore ⊑ ¬ Revisore

```
<inclusionAssertion>
        <basicC>
            <atomicC> Autore </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Revisore </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Revisore non è un Autore (sign="**negative**" vuole indicare la negazione del concetto).

Queste due asserzioni (Revisore ⊑ ¬ Autore e Autore ⊑ ¬ Revisore) stanno ad indicare che i concetti generali Revisore e Autore sono tra di loro disguinti.

-- Junior ⊑ Revisore

```
<inclusionAssertion>
        <basicC>
            <atomicC> Junior</atomicC>
        </basicC>
        <generalC>
```

```
            <signedC sign="positive">
               <basicC>
                     <atomicC>Revisore</atomicC>
               </basicC>
            </signedC>
         </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni revisore Junion è un Revisore.

Questa asserzione di inclusione di concetto sta ad indicare che la classe Junion è una sottoclasse di Revisore.

-- Senior ⊑ Revisore

```
<inclusionAssertion>
         <basicC>
               <atomicC> Senior</atomicC>
         </basicC>
         <generalC>
            <signedC sign="positive">
               <basicC>
                     <atomicC>Revisore</atomicC>
               </basicC>
            </signedC>
         </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni revisore Senior è un Revisore.

Questa asserzione di inclusione di concetto sta ad indicare che la classe Junion è una sottoclasse di Revisore.

Queste asserzioni di inclusione di concetto (Junior ⊑ Revisore e Senior ⊑ Revisore) stanno ad indicare che i concetti generali Junior e Senior sono derivati dal concetto di Revisore.

-- Junior ⊑ ¬ Senior

```
<inclusionAssertion>
         <basicC>
               <atomicC> Junior </atomicC>
         </basicC>
         <generalC>
               <signedC sign="negative">
                  <basicC>
                        <atomicC> Senior </atomicC>
                  </basicC>
```

```
            </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni revisore Junior non è un revisore Senior (sign="**negative**" vuole indicare la negazione del concetto).

-- Senior ⊑ ¬ Junior

```
<inclusionAssertion>
        <basicC>
            <atomicC> Senior </atomicC>
         </basicC>
        <generalC>
            <signedC sign="negative">
               <basicC>
                    <atomicC>Junior</atomicC>
               </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni revisore Senior non è un revisore Junior (sign="**negative**" vuole indicare la negazione del concetto).

Queste due asserzioni di concetto(Junior ⊑ ¬ Senior e Senior ⊑ ¬ Junior)
stanno ad indicare che i concetti generali Junior e Senior sono tra di loro disguinti.

-- (funct nome)

```
<funct>
        <atomicCA> nome </atomicCA>
</funct>
```

L'attributo di concetto, 'nome', è stato definito funzionale.

-- Persona ⊑ δ(nome)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Persona </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> nome </atomicCA>
```

```
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Persona ha associato un attributo di concetto ‘nome’.
Dato un attributo di concetto ‘nome’ chiamiamo dominio di ‘nome’ denotato con δ(nome) il set di oggetti che ‘nome’ collega a valori.

-- ρ(nome) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> nome </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘nome’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘nome’ chiamiamo range di ‘nome’ denotato con ρ(nome) il set di valori che ‘nome’ collega a oggetti.

-- (funct cognome)

```
<funct>
    <atomicCA> cognome </atomicCA>
</funct>
```

L’attributo di concetto, ‘cognome’, è stato definito funzionale.

-- Persona ⊑ δ(cognome)

```
<inclusionAssertion>
    <basicC>
        <atomicC> Persona </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> cognome </atomicCA>
                </CADomain>
```

```
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Persona ha associato un attributo di concetto ‘cognome’.
Dato un attributo di concetto ‘cognome’ chiamiamo dominio di ‘cognome’ denotato con δ(cognome) il set di oggetti che ‘cognome’ collega a valori.

-- ρ(cognome) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA>cognome</atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV>xs:string</predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘cognome’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘cognome’ chiamiamo range di ‘cognome’ denotato con ρ(cognome) il set di valori che ‘cognome’ collega a oggetti.

-- (funct email)

```
<funct>
        <atomicCA> email </atomicCA>
</funct>
```

L’attributo di concetto, ‘email’, è stato definito funzionale.

-- Persona ⊑ δ(email)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Persona </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> email </atomicCA>
                    </CADomain>
                </basicC>
```

```
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Persona ha associato un attributo di concetto ‘email’.
Dato un attributo di concetto ‘email’ chiamiamo dominio di ‘email’ denotato con δ(email) il set di oggetti che ‘email’ collega a valori.

-- ρ(email) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> email </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘email’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘email’ chiamiamo range di ‘email’ denotato con ρ(email) il set di valori che ‘email’ collega a oggetti.

-- (funct nazionalità)

```
<funct>
        <atomicCA> nazionalità </atomicCA>
</funct>
```

L’attributo di concetto, ‘nazionalità’, è stato definito funzionale.

-- Revisore ⊑ δ(nazionalità)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Revisore </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> nazionalità </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
```

```
        </generalC>
</inclusionAssertion>
```

Ogni concetto Revisore ha associato un attributo di concetto ‘nazionalità’.
Dato un attributo di concetto ‘nazionalità’ chiamiamo dominio di ‘nazionalità’ denotato con δ(nazionalità) il set di oggetti che ‘nazionalità’ collega a valori.

-- ρ(nazionalità) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> nazionalità </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘nazionalità’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘nazionalità’ chiamiamo range di ‘nazionalità’ denotato con ρ(nazionalità) il set di valori che ‘nazionalità’ collega a oggetti.

-- (funct titolo)

```
<funct>
        <atomicCA> titolo </atomicCA>
</funct>
```

L’attributo di concetto, ‘titolo’, è stato definito funzionale.

-- Articolo ⊑ δ(titolo)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Articolo </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> titolo </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
```

</inclusionAssertion>

Ogni concetto Articolo ha associato un attributo di concetto ‘titolo’.
Dato un attributo di concetto ‘titolo’ chiamiamo dominio di ‘titolo’ denotato con δ(titolo) il set di oggetti che ‘titolo’ collega a valori.

-- ρ(titolo) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> titolo </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘titolo’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘titolo’ chiamiamo range di ‘titolo’ denotato con ρ(titolo) il set di valori che ‘titolo’ collega a oggetti.

-- (funct dimensione)

```
<funct>
    <atomicCA> dimensione </atomicCA>
</funct>
```

L’attributo di concetto, ‘dimensione’, è stato definito funzionale.

-- Articolo ⊑ δ(dimensione)

```
<inclusionAssertion>
    <basicC>
        <atomicC> Articolo </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> dimensione </atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto Articolo ha associato un attributo di concetto ‘dimensione’.
Dato un attributo di concetto ‘dimensione’ chiamiamo dominio di ‘dimensione’ denotato con δ(dimensione) il set di oggetti che ‘dimensione’ collega a valori.

-- ρ(dimensione) ⊑ xs:float

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> dimensione </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:float </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘dimensione’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘dimensione’ chiamiamo range di ‘dimensione’ denotato con ρ(dimensione) il set di valori che ‘dimensione’ collega a oggetti.

-- autore_di ⊑ (inverse_of_ autore_di)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> autore_di </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> inverse_of_autore_di </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘autore_di’ è un’istanza del ruolo inverso di ‘ inverse_of_ autore_di’.

Questa asserzione di inclusione di ruoli sta ad indicare che il ruolo ‘inverse_of_ autore_di’ è il ruolo inverso del ruolo ‘autore_di’.

-- inverse_of_ autore_di ⊑ (autore_di)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
```

```
            <atomicR> inverse_of_autore_di </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> autore_di </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ autore_di’ è un’istanza del ruolo inverso di ‘autore_di’.

Questa asserzione di inclusione di ruoli sta ad indicare che il ruolo ‘autore_di’ è il ruolo inverso del ruolo ‘inverse_of_‘autore_di’.

-- ∃ autore_di ⊑ Autore

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> autore_di </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Autore </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘autore_di’ è un’ istanza del concetto Autore.

-- ∃ (autore_di)⁻ ⊑ Articolo

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> autore_di </atomicR>
                </basicR>
            </exists>
        </basicC>
```

```
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Articolo </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘autore_di’ è un’ istanza del concetto Articolo.

-- ∃ inverse_of_ autore_di ⊑ Articolo

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> inverse_of_autore_di </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Articolo </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_autore_di’ è un’ istanza del concetto Articolo.

-- ∃ (inverse_of_ autore_di)⁻ ⊑ Autore

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> inverse_of_autore_di </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Autore </atomicC>
                </basicC>
```

```
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘inverse_of_autore_di’ è un’ istanza del concetto Autore

-- Articolo ⊑ ∃ inverse_of_autore_di. Autore

```
<inclusionAssertion>
        <basicC>
            <atomicC> Articolo </atomicC>
        </basicC>
        <generalC>
            <qualifiedExists>
                <basicR dir="direct">
                    <atomicR> inverse_of_autore_di </atomicR>
                </basicR>
                <generalC>
                    <signedC sign="positive">
                        <basicC>
                            <atomicC>Autore</atomicC>
                        </basicC>
                    </signedC>
                </generalC>
            </qualifiedExists>
        </generalC>
</inclusionAssertion>
```

Ogni Articolo possiede (‘inverse_of_autore_di’) almeno un Autore

Questa asserzione (Articolo ⊑ ∃ inverse_of_autore_di. Autore) sta ad indicare che la classe Articolo partecipa alla relazione inverse_of_autore_di con la classe Autore con cardinalità minima 1.

-- (funct inverse_of_revisore_primario)

```
<funct>
        <basicR dir="direct">
            <atomicR> inverse_of_revisore_primario </atomicR>
        </basicR>
</funct>
```

Il ruolo di concetto, ‘inverse_of_revisore_primario’, è stato definito funzionale.

-- revisore_primario ⊑ (inverse_of_ revisore_primario)⁻

```
<inclusionAssertion>
```

```
        <basicR dir="direct">
            <atomicR> revisore_primario </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> inverse_of_revisore_primario </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘revisore_primario’ è un’istanza del ruolo inverso di ‘ inverse_of_ revisore_primario’.

Questa asserzione di inclusione di ruolo sta ad indicare che il ruolo ‘inverse_of_ revisore_primario’ è il ruolo inverso del ruolo ‘revisore_primario’.

-- inverse_of_ revisore_primario ⊑ (revisore_primario)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> inverse_of_revisore_primario </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> revisore_primario </atomicR>
                </basicR>
            </signedR>
        </generalR>
    </inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ revisore_primario’ è un’istanza del ruolo inverso di ‘revisore_primario’.

Questa asserzione di inclusione di ruolo sta ad indicare che il ruolo inverso ‘inverse_of_ revisore_primario’ è il ruolo inverso del ruolo ‘revisore_primario’

-- ∃ revisore_primario ⊑ Senior

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> revisore_primario </atomicR>
                </basicR>
```

```
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Senior </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘revisore_primario’ è un’ istanza del concetto Senior.

-- ∃ (revisore_primario)¯ ⊑ Articolo

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> revisore_primario </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Articolo </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘revisore_primario’ è un’ istanza del concetto Senior.

-- ∃ inverse_of_ revisore_primario ⊑ Articolo

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> inverse_of_revisore_primario </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
```

```
                    <atomicC> Articolo </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_revisore_primario’ è un’ istanza del concetto Articolo

-- ∃ (inverse_of_ revisore_primario)¯ ⊑ Senior

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                        <atomicR> inverse_of_revisore_primario </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Senior </atomicC>
                </basicC>
             </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo di ‘inverse_of_revisore_primario’ è un’ istanza del concetto Senior.

-- Articolo ⊑ ∃ inverse_of_revisore_primario. Senior

```
<inclusionAssertion>
        <basicC>
             <atomicC> Articolo </atomicC>
        </basicC>
        <generalC>
             <qualifiedExists>
                <basicR dir="direct">
                        <atomicR> inverse_of_revisore_primario</atomicR>
                </basicR>
                <generalC>
                        <signedC sign="positive">
                            <basicC>
                                    <atomicC>Senior</atomicC>
                            </basicC>
                        </signedC>
                </generalC>
```

```
            </qualifiedExists>
        </generalC>
</inclusionAssertion>
```

Ogni Articolo possiede (‘inverse_of_revisore_primario’) almeno un revisore Senior

Questa asserzione (Articolo ⊑ ∃ inverse_of_revisore_primario. Senior) sta ad indicare che la classe Articolo partecipa alla relazione inverse_of_revisore_primario con la classe Senior con cardinalità minima 1

-- revisore_secondario ⊑ (inverse_of_ revisore_secondario)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> revisore_secondario </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> inverse_of_revisore_secondario </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘revisore_secondario’ è un’istanza del ruolo inverso di ‘ inverse_of_ revisore_secondario’.

Questa asserzione di ruolo sta ad indicare che il ruolo ‘inverse_of_ revisore_secondario’ è il ruolo inverso del ruolo ‘revisore_secondario’.

-- inverse_of_ revisore_secondario ⊑ (revisore_secondario)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> inverse_of_revisore_secondario </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> revisore_secondario </atomicR>
                </basicR>
            </signedR>
        </generalR>
    </inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ revisore_secondario’ è un’istanza del ruolo inverso di ‘revisore_secondario’.

Questa asserzione di ruolo sta ad indicare che il ruolo ‘revisore_secondario’‘è il ruolo inverso del ruolo inverse_of_ revisore_secondario’.

-- ∃ revisore_secondario ⊑ Junior

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                        <atomicR> revisore_secondario </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                 <basicC>
                        <atomicC> Junior </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘revisore_secondario’ è un’ istanza del concetto Junior .

-- ∃ (revisore_secondario)¯ ⊑ Articolo

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                        <atomicR> revisore_secondario </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Articolo </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso ‘revisore_secondario’ è un’ istanza del concetto Articolo.

-- ∃ inverse_of_ revisore_secondario ⊑ Articolo

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                        <atomicR> inverse_of_revisore_secondario </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Articolo </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_revisore_secondario’ è un’ istanza del concetto Articolo.

-- ∃ (inverse_of_ revisore_secondario)⁻ ⊑ Junior

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                        <atomicR> inverse_of_revisore_secondario </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Junior </atomicC>
                </basicC>
             </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘inverse_of_revisore_secondario’ è un’ istanza del concetto Junior .

Conclusioni:

1) *<!-- UnionOf not totally expressible in DL-liteA -->*

La classe Persona è la generalizzazione completa e disgiunta delle classi Autore e Revisore.

Class (a: Persona complete
unionOf(a: Autore a: Revisore))

In DL-liteA possimo esprimere le proprietà di disgiunzione tramite la seguente asserzione di inclusione di concetti: Revisore ⊑ ¬ Autore e Autore ⊑ ¬ Revisore
le proprietà di completezza con le seguenti asserzioni di inclusione di concetti Autore ⊑ Persona e Revisore ⊑ Persona

L'unione di queste classi non è completamente esprimibile in DL-liteA.

2) *<!-- UnionOf not totally expressible in DL-liteA -->*

La classe Revisore è la generalizzazione completa e disgiunta delle classi Junior e Senior.

Class (a: Revisore complete
unionOf(a: Junior a: Senior))

In DL-liteA possimo esprimere le proprietà di disgiunzione tramite la seguente asserzione di inclusione di concetti: Junior ⊑ ¬ Senior e Senior ⊑ ¬ Junior
le proprietà di completezza con le seguenti asserzioni di inclusione di concetti Junior ⊑ Revisore Senior ⊑ Revisore

L'unione di queste classi non è completamente esprimibile in DL-liteA.

3) *<!-- IntersectionOf not totally expressible in DL-liteA -->*

La classe Articolo è espressa in DL-lite come l'intersezione di:
la restrizione (che ha lo scopo di indicare la cardinalita minima della relazione inverse_of_ autore_di con la classe Autore)
la restrizione (che ha lo scopo di indicare la cardinalita minima della relazione inverse_of_ revisore_secondario)
la restrizione (che ha lo scopo di indicare la cardinalita minima della relazione inverse_of_ revisore_primario con la classe Senior)

Class(a:Articolo complete
intersectionOf(restriction(a:inverse_of_autore_di someValuesFrom(a:Autore))
restriction(a:inverse_of_revisore_secondario minCardinality(2))
restriction(a:inverse_of_revisore_primariosomeValuesFrom(a:Senior)))

In DL-liteA il concetto di intersezione non è totalmente esprimibile.

4) *<!-- Cardinality different from 1 not expressible in DL-liteA-->*

La classe Utente partecipa relazione 'possiede' con la classe Contratto la cardinalita massima di 2. Il Dl-lite questo viene espresso tramite una restizione:

restriction(a:inverse_of_revisore_secondario minCardinality(2))

In Dl-liteA la cardinalità dfferente da 1 non è esprimibile.

5) *<!-- NOT IN DL-LiteA!There is a conflict :the atomic Role "inverse_of_autore_di"is qualified exists then it cannot be functional*
-->

La base di conoscenza in DL-lite-A K=<T,A>, è ottenuta attraverso delle restrizioni: ovvero T (TBox) è una DL-lite$_{FR}$ Tbox che soddisfa determinate condizioni. In particolare, in questo caso:
-- Per ogni ruolo atomico e inverso di un ruolo atomico Q compare in un concetto della forma $\exists Q.C$, le asserzioni (funct Q) e (funct Q^{-}) non sono in T.

Poiché il ruolo inverse_of_autore_di compare nella seguente asserzione:
Articolo $\sqsubseteq \exists$ inverse_of_autore_di. Autore e quindi è quantificato esistenzialmente, l'asserzione (funct inverse_of_autore_di) non è in T, e quindi il ruolo non può essere funzionale.

6) *Attributo di ruolo in DL-LiteA*

In DL-lite$_A$ è possibile esprimere attributi di ruolo denotando la relazione binaria tra coppie di oggetti e valori (non esprimibile in DL-lite).
La relazione revisore_primario presenta un attributo di relazione 'giudizioPositivo' di tipo intero.
Questo può essere espresso attraverso un attributo di ruolo 'giudizioPositivo' per il ruolo 'revisore_primario'.
Le espressioni sono le seguenti:

Asserzione funzionale di attributo di ruolo:

(funct giudizioPositivo)

L'attributo di ruolo 'giudizioPositivo' è funzionale

<u>Asserzioni di inclusione di concetti:</u>

Senior $\sqsubseteq \exists \delta$(giudizioPositivo)
Articolo $\sqsubseteq \exists \delta(\text{giudizioPositivo})^{-}$

Ogni Senior deve partecipare al ruolo revisore_primario (dominio di giudizioPositivo) avendo associato un attributo di ruolo ‘giudizioPositivo’.
Dato un attributo di ruolo ‘giudizioPositivo’ chiamiamo dominio di ‘giudizioPositivo’ denotato con δ(giudizioPositivo) il set di oggetti che ‘giudizioPositivo’ collega a valori.
Ogni Articolo deve partecipare al ruolo ‘inverse_of_ revisore_primario’ (dominio inverso di giudizioPositivo) avendo associato un attributo di ruolo ‘giudizioPositivo’.

Asserzioni di inclusione di ruolo:

$\delta(\text{giudizioPositivo}) \sqsubseteq \text{revisore\_primario}$
$\delta(\text{giudizioPositivo})^{-} \sqsubseteq \text{inverse\_of\_revisore\_primario}$

Il dominio dell’attributo di ruolo ‘giudizioPositivo’ è il ruolo ‘revisore_primario’
Il dominio inverso dell’attributo di ruolo ‘giudizioPositivo’ è il ruolo ‘inverse_of_revisore_primario’

Asserzioni di inclusione di dominio di valori:

$\rho(\text{giudizioPositivo}) \sqsubseteq \text{xs:bool}$

Il range dell’attributo di ruolo ‘giudizioPositivo’ è xs:bool

7) *Attributo di ruolo in DL-LiteA*

In DL-liteA è possibile esprimere attributi di ruolo denotando la relazione binaria tra coppie di oggetti e valori (non esprimibile in DL-lite).
La relazione revisore_secondario presenta un attributo di relazione ‘voto’ di tipo intero.
Questo può essere espresso attraverso un attributo di ruolo ‘voto’ per il ruolo ‘revisore_secondario’.
Le espressioni sono le seguenti:

Asserzione funzionale di attributo di ruolo:

(funct voto)

L’attributo di ruolo ‘voto’ è funzionale

Asserzioni di inclusione di concetti:

$\text{Junior} \sqsubseteq \exists\, \delta(\text{voto})$
$\text{Articolo} \sqsubseteq \exists\, \delta(\text{voto})^{-}$

Ogni Junior deve partecipare al ruolo revisore_secondario (dominio di voto) avendo associato un attributo di ruolo ‘voto’.
Dato un attributo di ruolo ‘voto’ chiamiamo dominio di ‘voto’ denotato con δ(voto) il set di oggetti che ‘voto’ collega a valori.

Ogni Articolo deve partecipare al ruolo ‘inverse_of_ revisore_secondario’ (dominio inverso di voto) avendo associato un attributo di ruolo ‘voto’.

Asserzioni di inclusione di ruolo:

$\delta(\text{voto}) \sqsubseteq \text{revisore\_secondario}$
$\delta(\text{voto})^{-} \sqsubseteq \text{inverse\_of\_revisore\_secondario}$

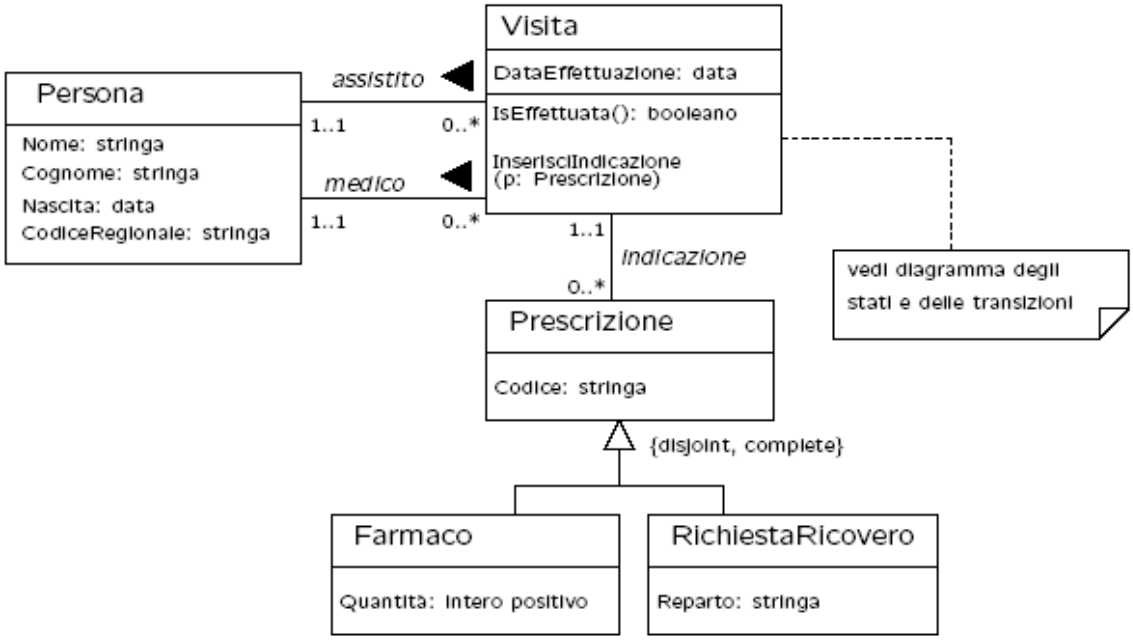
Il dominio dell’attributo di ruolo ‘voto’ è il ruolo ‘revisore_secondario’
Il dominio inverso dell’attributo di ruolo ‘voto’ è il ruolo ‘inverse_of_revisore_secondario’

Asserzioni di inclusione di dominio di valori:

$\rho(\text{voto}) \sqsubseteq \text{xs:int}$

Il range dell’attributo di ruolo ‘voto’ è xs:int

Compito d’esame 22 giugno 2005



OWL_DL

Classi: Persona, Visita, Prescrizione, Farmaco, RichietaRicovero
Proprietà: nome, cognome, nascita, codiceRegione, dataEffettazione, codice, quantità, reparto
Relazioni: assistito, medico, indicazione

CLASSI

- **Persona**

L'oggetto "Persona" è stato tradotto in una classe Protègè che ha come proprietà gli attributi 'nome', 'cognome', 'nascita', 'codiceregione' tradotti come Dataproperty..
La classe partecipa alla relazione 'inverse_of_assistito' con la classe Visita con la cardinalità 0..*,e alla relazione 'inverse_of_medico' con la classe Visita con la cardinalità 0..*.Queste cardinalità sono che è quelle che vengono indicate di default e perciò non è necessario indicarle attraverso restrizioni cardinali.

Class(a: **Persona** partial)
DisjointClasses(a: **Persona** a: **Prescrizione**)
DisjointClasses(a:**Visita** a: **Persona**)

La classe Persona è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalle classi Prescrizione e Visita.

- **Visita**

L'oggetto "Visita" è stato tradotto in una classe Protègè che ha come proprietà l'attributo 'dataeffettuazione' tradotti come Dataproperty che ha come range date.
La classe partecipa alla relazione 'assistito' con la classe Persona con la cardinalità 1..*, e alla relazione 'medico' con la classe Persona con la cardinalità 1..*.
La cardinalità minima viene espressa tramite una restriction esisteziale mentre la cardinalità massima con proprietà Functional sulle relazioni.

Class(a: **Visita** complete
intersectionOf(
restriction(a: **assistito** someValuesFrom(a: **Persona**))
restriction(a: **medico** someValuesFrom(a: **Persona**))))
DisjointClasses(a: V**isita** a: **Prescrizione**)
DisjointClasses(a:**Visita** a: **Persona**)

La classe Visita è definita *complete* perché ha tra le condizioni necessarie e sufficienti restrizione ad indicare il vincolo di cardinalità minima, ed è disgiunta dalla classe Persona e Prescrizione.

- **Prescrizione**

L'oggetto "Prescrizione" è stato tradotto in una classe Protègè che ha come proprietà l'attributo 'codice' tradotti come Dataproperty.
La classe Prescrizione è superclasse nella generalizzazione ed ha come sottoclassi Farmaco e RichiestaRicovero (che sono tra di loro disgiunte).

La generalizzazione è disgiunta e completa e questo vincolo è stato espresso attraverso la restrizione Farmaco U RichiestaRicovero inserita nelle condizioni necessarie e sufficienti.
La classe partecipa alla relazione 'inverse_prescrizione con la classe Visita, con la cardinalità 1..1
La cardinalità minima viene espressa tramite la restriction esistenziale mentre la cardinalità massima con proprietà Functional sulla relazione.

Class(a: **Prescrizione** complete
intersectionOf(
unionOf(a: **Farmaco** a: **RichiestaRicovero**)
restriction(a: **inverse_of_indicazione** someValuesFrom(a:**Visita**))))
DisjointClasses(a: V**isita** a: **Prescrizione**)
DisjointClasses(a: **Persona** a: **Prescrizione**)

La classe Prescrizione è *complete:* è descritta utilizzando condizioni necessarie e sufficienti.
E' l'unione delle classi Farmaco d Richiesta ricovero ed è disgiunta dalle classi Visita e Persona .

- **Farmaco**

L'oggetto "Farmaco" è stato tradotto in una classe Protègè ed è una sottoclasse di Prescrizione da cui eredita le proprietà ed ha come suo attributo Quantità tradotto come Dataproperty
La classe è disgiunta con la classe Richiestaricovero.
Queste classi sono tra di loro disgiunte e complete quindi un individuo che appartiene alla classe Farmaco non può appartenere alla classe RichiestaRicovero .

Class(a: **Farmaco** partial a: **Prescrizione**)
DisjointClasses(a:**Farmaco** a:**RichiestaRicovero**)

La classe Farmaco è definita *partial* perché ha almeno una condizione necessaria inoltre è disgiunta dalla classe RichiestaRicovero.

- **RichiestaRicovero**

L'oggetto "RichiestaRicovero" è stato tradotto in una classe Protègè ed è una sottoclasse di Prescrizione da cui eredita le proprietà ed ha come suo attributo Reparto tradotto come Dataproperty
La classe è disgiunta con la classe Farmaco.
Queste classi sono tra di loro disgiunte e complete quindi un individuo che appartiene alla classe Farmaco non può appartenere alla classe RichiestaRicovero .

Class(a: **RichiestaRicovero** partial a: **Prescrizione**)
DisjointClasses(a:**Farmaco** a:**RichiestaRicovero**)

La classe RichiestaRicovero è definita *partial* perché ha almeno una condizione necessaria inoltre è disgiunta dalla classe Farmac.

PROPRIETA' (DataProperty)

Gli attributi della classe sono stati tradotti come DataProperty e hanno come la proprietà Functional poiché sono associati ad un solo valore per individuo.

- Gli attributi 'nome', 'cognome', 'codiceregione' hanno come dominio la classe Persona e come range string mentre l'attributo ' ha come dominio la classe Persona e come range date.

DatatypeProperty(a: **nome** Functional
domain(a: **Persona**)
range(xsd: string))

DatatypeProperty(a: **cognome** Functional
domain(a: **Persona**)
range(xsd: string))

DatatypeProperty(a: **nascita** Functional
domain(a: **Persona**)
range(xsd: date))

DatatypeProperty(a: **codiceregione** Functional
domain(a: **Persona**)
range(xsd: string))

- L'attributo 'dataeffettuazione' ha come dominio la classe Visita e come range date

DatatypeProperty(a: **dataeffettuazione** Functional
domain(a: **Visita**)
range(xsd: date))

- L'attributo 'codice' ha come dominio la classe Prescrizione e come range string

DatatypeProperty(a: **codice** Functional
domain(a: **Prescrizione**)
range(xsd: string))

- L'attributo 'quantita' ha come dominio la classe Farmaco e come range int

DatatypeProperty(a: **quantita** Functional
domain(a: **Farmaco**)
range(xsd: int))

- L'attributo 'reparto' ha come dominio la classe RichiestaRicovero e come range string

DatatypeProperty(a: **reparto** Functional
domain(a: **RichiestaRicovero**)
range(xsd: string))

<u>RELAZIONI</u> (ObjectProperty)

Le associazioni sono state tradotte in relazioni tra le classi (ObjectProperty) e per ogni relazione è stata definita una relazione inversa:

assistito ←→ inverse_of_assistito
medico ←→ inverse_of_medico
indicazioni←→inverse_of_indicazioni

- **assistito**

L'associazione 'assistito' viene tradotta con un ObjectProperty che ha come dominio la classe Visita come range la classe Persona ed è stata definita Functional per indicare il vincolo di cardinalità 1..1, per la cardinalità massima (se la visita a ha come assistito la persona b, e se la visita a ha come assistito la persona c, allora b e c coincidono).

ObjectProperty(a: **assistito** Functional
inverseOf(a: **inverse_of_assistito**)
domain(a: **Visita**)
range(a: **Persona**))

La sua relazione inversa è inverse_of_assistito ha come dominio la classe Persona e come range la classe Visita e come proprietà inverseFunctional in quanto la relazione inversa è Functional.

ObjectProperty(a: **inverse_of_assistito** InverseFunctional
inverseOf(a: **assistito**)
domain(a: **Persona**)
range(a: **Visita**))

- **medico**

L'associazione 'medico' viene tradotta con un ObjectProperty che ha come dominio la classe Visita come range la classe Persona ed è stata definita Functional per indicare il vincolo di cardinalità 1..1, per la cardinalità massima (se la visita a ha come medico la persona b, e se la visita a ha come medico la persona c, allora b e c coincidono).

ObjectProperty(a: **medico** Functional
inverseOf(a: **inverse_of_medico**)
domain(a: **Visita**)
range(a: **Persona**))

La sua relazione inversa è inverse_of_medico ha come dominio la classe Persona e come range la classe Visita e come proprietà inverseFunctional in quanto la relazione inversa è Functional.

ObjectProperty(a: **inverse_of_medico** InverseFunctional
inverseOf(a: **medico**)
domain(a: **Persona**)
range(a: **Visita**))

• **prescrizione**

L'associazione 'prescrizione' viene tradotta con un ObjectProperty che ha come dominio la classe Visita come range la classe Prescrizione ed è stata inverseFunctional in quanto la relazione inversa è Functional.

ObjectProperty(a: **indicazione** InverseFunctional
inverseOf(a: **inverse_of_indicazione**)
domain(a: **Visita**)
range(a: **Prescrizione**))

La sua relazione inversa è 'inverse_of_prescrizione' ha come dominio la classe Prescrizione e come range la classe Visita ed è stata definita Functional per indicare il vincolo di cardinalità 1..1, per la cardinalità massima (se la visita a ha come prescrizione b, e se la visita a ha come prescrizione la c, allora b e c coincidono).

ObjectProperty(a: **inverse_of_indicazione** Functional
inverseOf(a: **indicazione**)
domain(a: **Prescrizione**)
range(a: **Visita**))

.

DL-LiteA

Asserzioni di inclusione di concetti

Persona ⊑ ¬ Prescrizione
Persona ⊑ ¬ Visita
Prescrizione ⊑ ¬ Visita
Prescrizione ⊑ ¬Persona
Visita ⊑ ¬Persona
Visita⊑ ¬ Prescrizione
Farmaco ⊑ Prescrizione
Farmaco ⊑ ¬ RichiestaRicovero
RichiestaRicovero ⊑ Prescrizione
RichiestaRicovero ⊑ ¬ Farmaco

Persona ⊑ δ(nome)
Persona ⊑ δ(cognome)
Persona ⊑ δ(nascita)
Persona ⊑ δ(codiceregione)
Visita ⊑ δ(dataeffettuazione)
Prescrizione ⊑ δ(codice)
Farmaco ⊑ δ(quantita)
RichiestaRicovero ⊑ δ(reparto)

∃ assistito ⊑ Visita
∃ (assistito)$^{-}$ ⊑ Persona
∃ inverse_of_ assistito ⊑ Persona
∃ (inverse_of_ assistito)$^{-}$ ⊑ Visita
∃ medico ⊑ Visita
∃ (medico)$^{-}$ ⊑ Persona
∃ inverse_of_ medico ⊑ Persona
∃ (inverse_of_ medico)$^{-}$ ⊑ Visita
∃ indicazione ⊑ Visita
∃ (indicazione)$^{-}$ ⊑ Prescrizione
∃ inverse_of_ indicazione ⊑ Prescrizione
∃ (inverse_of_ indicazione)$^{-}$ ⊑ Visita

Visita ⊑ ∃ assistito. Persona
Visita ⊑ ∃ medico. Persona
Prescrizione ⊑ ∃ inverse_of_indicazione. Visita

Asserzioni di funzionalità di attributi di concetto

(funct nome)
(funct cognome)
(funct nascita)
(funct codiceregione)
(funct dataeffettuazione)

(funct codice)
(funct quantita)
(funct reparto)

Asserzioni di funzionalità di ruolo

(funct medico)
(funct inverse_of_indicazione)

Asserzioni di inclusione di domini di valori

ρ(nome) ⊑ xs:string
ρ(cognome) ⊑ xs:string
ρ(nascita) ⊑ xs:date
ρ(codiceregione) ⊑ xs:string
ρ(dataeffettuazione) ⊑ xs:date
ρ(codice) ⊑ xs:string
ρ(quantita) ⊑ xs:int
ρ(reparto) ⊑ xs:string

Asserzioni di inclusione di ruoli

assistito ⊑ (inverse_of_ assistito)$^{-}$
inverse_of_ assistito ⊑ (assistito)$^{-}$
medico ⊑ (inverse_of_ medico)$^{-}$
inverse_of_ medico ⊑ (medico)$^{-}$
indicazione ⊑ (inverse_of_ indicazione)$^{-}$
inverse_of_ indicazione ⊑ (indicazione)$^{-}$

- **Alfabeto**

L'alfabeto che descrive lo schema UML è formato dai seguenti elementi:
da concetti generali (atomicC), attributi di concetti (atomicCA) , ruoli generali (atomicR).

<alphabet>

<atomicC> **Persona** </atomicC>
<atomicC> **Visita** </atomicC>
<atomicC> **Prescrizione** </atomicC>
<atomicC> **RichiestaRicovero** </atomicC>
<atomicC> **Farmaco** </atomicC>

<atomicCA> **nome** </atomicCA>
<atomicCA> **cognome** </atomicCA>
<atomicCA> **nascita** </atomicCA>
<atomicCA> **codiceregione** </atomicCA>
<atomicCA> **dataeffettuazione**</atomicCA>
<atomicCA> **codice** </atomicCA>
<atomicCA> **reparto** </atomicCA>
<atomicCA> **quantita** </atomicCA>

<atomicR> **medico** </atomicR>
<atomicR> **inverse_of_medico** </atomicR>
<atomicR> **assistito** </atomicR>
<atomicR> **inverse_of_assistito** </atomicR>
<atomicR> **indicazione** </atomicR>
<atomicR> **inverse_of_indicazione** </atomicR>

</alphabet>

Le Classi Persona, Visita, Prescrizione, RichiestaRicovero, Farmaco sono concetti generali.
Le proprietà delle classi nome, cognome, nascita, codiceregione, dataeffettuazione, codice,reparto,quantita sono attributi di concetto.
Le relazioni medico, inverse_of_ medico, assistito, inverse_of_assistito, indicazione, inverse_of_ indicazione sono ruoli generali.

• **Tbox**

La Tbox è usata per rappresentare la conoscenza intensionale attraverso assezioni di inclusione.

-- Persona ⊑ ¬ Prescrizione

```
<inclusionAssertion>
        <basicC>
              <atomicC> Persona </atomicC>
        </basicC>
        <generalC>
              <signedC sign="negative">
                  <basicC>
                        <atomicC> Prescrizione </atomicC>
                  </basicC>
              </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni Persona non è un Prescrizione (sign="**negative**" vuole indicare la negazione del concetto).

-- Prescrizione ⊑ ¬ Persona

```
<inclusionAssertion>
        <basicC>
              <atomicC> Prescrizione </atomicC>
        </basicC>
        <generalC>
              <signedC sign="negative">
                  <basicC>
                        <atomicC> Persona </atomicC>
                  </basicC>
              </signedC>
        </generalC>
 </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Prescrizione non è una Persona (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di inclusione di concetto (Persona ⊑ ¬ Prescrizione, Prescrizione ⊑ ¬Persona) stanno ad indicare la disgiunzione tra i concetti Prescrizione e Persona.

-- Persona ⊑ ¬ Visita

```
<inclusionAssertion>
        <basicC>
                <atomicC> Persona </atomicC>
        </basicC>
        <generalC>
                <signedC sign="negative">
                    <basicC>
                            <atomicC> Visita </atomicC>
                    </basicC>
                </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Persona non è un Visita (sign="**negative**" vuole indicare la negazione del concetto).

-- Visita ⊑ ¬ Persona

```
<inclusionAssertion>
        <basicC>
                <atomicC> Visita </atomicC>
        </basicC>
        <generalC>
                <signedC sign="negative">
                    <basicC>
                            <atomicC> Persona </atomicC>
                    </basicC>
                </signedC>
        </generalC>
 </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Visita non è una Persona (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di concetto (Visita ⊑ ¬Persona **,** *Persona ⊑ ¬ Visita) stanno ad indicare che i concetti Visita e Persona sono disgiunti.*

-- Visita⊑ ¬ Prescrizione

```
<inclusionAssertion>
        <basicC>
                <atomicC> Visita </atomicC>
        </basicC>
        <generalC>
```

```
            <signedC sign="negative">
                <basicC>
                    <atomicC> Prescrizione </atomicC>
                </basicC>
            </signedC>
        </generalC>
  </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Visita non è una Prescrizione (sign="**negative**" vuole indicare la negazione del concetto).

-- Prescrizione ⊑ ¬ Visita

```
<inclusionAssertion>
        <basicC>
            <atomicC> Prescrizione </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Visita </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Prescrizione non è un Visita (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di concetto (Visita ⊑ ¬Prescrizione , Prescrizione ⊑ ¬ Visita) stanno ad indicare che i concetti Visita e Prescrizione sono disgiunti.

-- Farmaco ⊑ Prescrizione

```
<inclusionAssertion>
        <basicC>
            <atomicC> Farmaco </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Prescrizione </atomicC>
                </basicC>
           </signedC>
        </generalC>
</inclusionAssertion>
```

L’asserzione di inclusione di concetto vuole indicare che ogni concetto Farmaco è una Prescrizione.

Questa asserzione di concetto (Farmaco ⊑ Prescrizione)sta ad indicare che la classe Farmaco è una sottoclasse di Prescrizione.

-- RichiestaRicovero ⊑ Prescrizione

```
<inclusionAssertion>
        <basicC>
              <atomicC>RichiestaRicovero </atomicC>
        </basicC>
        <generalC>
          <signedC sign="positive">
               <basicC>
                    <atomicC> Prescrizione </atomicC>
               </basicC>
          </signedC>
        </generalC>
</inclusionAssertion>
```

L’asserzione di inclusione di concetto vuole indicare che ogni Visita è una Prescrizione.

Questa asserzione RichiestaRicovero ⊑ Prescrizione sta ad indicare che la classe RichiestaRicovero è una sottoclasse di Prescrizione.

Queste asserzioni di inclusione di concetto stanno ad indicare che i concetti generali RichiestaRicovero e Revisore sono derivati dal concetto di Prescrizione.

-- Farmaco ⊑ ¬ RichiestaRicovero

```
<inclusionAssertion>
        <basicC>
             <atomicC> Farmaco </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                      <atomicC>RichiestaRicovero </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

L’asserzione di inclusione di concetto vuole indicare che ogni Farmaco non è unRichiestaRicovero (sign="**negative**" vuole indicare la negazione del concetto).

-- RichiestaRicovero ⊑ ¬ Farmaco

```
<inclusionAssertion>
        <basicC>
            <atomicC>RichiestaRicovero </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Farmaco </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Farmaco non è unRichiestaRicovero (sign="**negative**" vuole indicare la negazione del concetto).

Queste due asserzioni di inclusione di concetto (Farmaco ⊑ ¬ RichiestaRicovero e RichiestaRicovero ⊑ ¬ Farmaco) stanno ad indicare che i concetti generali Farmaco e RichiestaRicovero sono tra di loro disguinti.

-- (funct nome)

```
<funct>
        <atomicCA> nome </atomicCA>
</funct>
```

L'attributo di concetto, 'nome', è stato definito funzionale.

-- Persona ⊑ δ(nome)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Persona </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> nome </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Persona ha associato un attributo di concetto ‘nome’.
Dato un attributo di concetto ‘nome’ chiamiamo dominio di ‘nome’ denotato con δ(nome) il set di oggetti che ‘nome’ collega a valori.

-- ρ(nome) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> nome </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘nome’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘nome’ chiamiamo range di ‘nome’ denotato con ρ(nome) il set di valori che ‘nome’ collega a oggetti.

-- (funct cognome)

```
<funct>
    <atomicCA> cognome </atomicCA>
</funct>
```

L’attributo di concetto, ‘cognome’, è stato definito funzionale.

-- Persona ⊑ δ(cognome)

```
<inclusionAssertion>
    <basicC>
        <atomicC> Persona </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> cognome </atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto Persona ha associato un attributo di concetto ‘cognome’.
Dato un attributo di concetto ‘cognome’ chiamiamo dominio di ‘cognome’ denotato con δ(cognome) il set di oggetti che ‘cognome’ collega a valori.

-- ρ(cognome) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA>cognome</atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV>xs:string</predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘cognome’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘cognome’ chiamiamo range di ‘cognome’ denotato con ρ(cognome) il set di valori che ‘cognome’ collega a oggetti.

-- (funct nascita)

```
<funct>
        <atomicCA> nascita </atomicCA>
</funct>
```

L’attributo di concetto, ‘nascita’, è stato definito funzionale.

-- Persona ⊑ δ(nascita)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Persona </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> nascita </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Persona ha associato un attributo di concetto ‘nascita’.

Dato un attributo di concetto ‘nascita’ chiamiamo dominio di ‘nascita’ denotato con δ(nascita) il set di oggetti che ‘nascita’ collega a valori.

-- ρ(nascita) ⊑ xs:date

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> nascita </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:date </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘nascita’ è un dominio di valori predefinito xs:date.
Dato un attributo di concetto di ‘nascita’ chiamiamo range di ‘nascita’ denotato con ρ(nascita) il set di valori che ‘nascita’ collega a oggetti.

-- (funct codiceregione)

```
<funct>
    <atomicCA> codiceregione </atomicCA>
</funct>
```

L’attributo di concetto, ‘codiceregione’, è stato definito funzionale.

-- Persona ⊑ δ(codiceregione)

```
<inclusionAssertion>
    <basicC>
        <atomicC> Persona </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> codiceregione </atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto Persona ha associato un attributo di concetto ‘codiceregione’.

Dato un attributo di concetto ‘codiceregione’ chiamiamo dominio di ‘codiceregione’ denotato con δ(codiceregione) il set di oggetti che ‘codiceregione’ collega a valori.

-- ρ(codiceregione) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> codiceregione </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘codiceregione’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘codiceregione’ chiamiamo range di ‘codiceregione’ denotato con ρ(codiceregione) il set di valori che ‘codiceregione’ collega a oggetti.

-- (funct dataeffettuazione)

```
<funct>
    <atomicCA> dataeffettuazione </atomicCA>
</funct>
```

L’attributo di concetto, ‘dataeffettuazione’, è stato definito funzionale.

-- Visita ⊑ δ(dataeffettuazione)

```
<inclusionAssertion>
    <basicC>
        <atomicC> Visita </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> dataeffettuazione </atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto Visita ha associato un attributo di concetto ‘dataeffettuazione’.

Dato un attributo di concetto ‘dataeffettuazione’ chiamiamo dominio di ‘dataeffettuazione’ denotato con δ(dataeffettuazione) il set di oggetti che ‘dataeffettuazione’ collega a valori.

-- ρ(dataeffettuazione) ⊑ xs:date

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> dataeffettuazione </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:date </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘dataeffettuazione’ è un dominio di valori predefinito xs:date.
Dato un attributo di concetto di ‘dataeffettuazione’ chiamiamo range di ‘dataeffettuazione’ denotato con ρ(dataeffettuazione) il set di valori che ‘dataeffettuazione’ collega a oggetti.

-- (funct codice)

```
<funct>
    <atomicCA> codice </atomicCA>
</funct>
```

L’attributo di concetto, ‘codice’, è stato definito funzionale.

-- Prescrizione ⊑ δ(codice)

```
<inclusionAssertion>
    <basicC>
        <atomicC> Prescrizione </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> codice </atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto Prescrizione ha associato un attributo di concetto ‘codice’.

Dato un attributo di concetto ‘codice’ chiamiamo dominio di ‘codice’ denotato con δ(codice) il set di oggetti che ‘codice’ collega a valori.

-- ρ(codice) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> codice </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘codice’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘codice’ chiamiamo range di ‘codice’ denotato con ρ(codice) il set di valori che ‘codice’ collega a oggetti.

-- (funct quantita)

```
<funct>
    <atomicCA> quantita </atomicCA>
</funct>
```

L’attributo di concetto, ‘quantita’, è stato definito funzionale.

-- Farmaco ⊑ δ(quantita)

```
<inclusionAssertion>
    <basicC>
        <atomicC> Farmaco </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> quantita </atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto Farmaco ha associato un attributo di concetto ‘quantita’.

Dato un attributo di concetto ‘quantita’ chiamiamo dominio di ‘quantita’ denotato con δ(quantita) il set di oggetti che ‘quantita’ collega a valori.

-- ρ(quantita) ⊑ xs:int

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> quantita </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:int </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘quantita’ è un dominio di valori predefinito xs:int.
Dato un attributo di concetto di ‘quantita’ chiamiamo range di ‘quantita’ denotato con ρ(quantita) il set di valori che ‘quantita’ collega a oggetti.

-- (funct reparto)

```
<funct>
    <atomicCA> reparto </atomicCA>
</funct>
```

L’attributo di concetto, ‘reparto’, è stato definito funzionale.

-- RichiestaRicovero ⊑ δ(reparto)

```
<inclusionAssertion>
    <basicC>
        <atomicC> RichiestaRicovero </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> reparto </atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto RichiestaRicovero ha associato un attributo di concetto ‘reparto’.

Dato un attributo di concetto ‘reparto’ chiamiamo dominio di ‘reparto’ denotato con δ(reparto) il set di oggetti che ‘reparto’ collega a valori.

-- ρ(reparto) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> reparto </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘reparto’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘reparto’ chiamiamo range di ‘reparto’ denotato con ρ(reparto) il set di valori che ‘reparto’ collega a oggetti.

-- assistito ⊑ (inverse_of_ assistito)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> assistito </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> inverse_of_assistito </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘assistito’ è un’istanza del ruolo inverso di ‘ inverse_of_ assistito’.

Questa asserzione di inclusione di ruolo (assistito ⊑ (inverse_of_assistito)⁻)sta ad indicare che il ruolo ‘inverse_of_assistito’ è la relazione inversa della relazione ‘assistito’.

-- inverse_of_ assistito ⊑ (assistito)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> inverse_of_assistito </atomicR>
        </basicR>
```

```
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> assistito </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ assistito’ è un’istanza del ruolo inverso di ‘assistito’.

Questa asserzione di inclusione di ruolo (inverse_of_ assistito ⊑ (assistito)¯)
sta ad indicare che il ruolo ‘assistito’ è la relazione inversa della relazione ‘inverse_of_assistito

-- ∃ assistito ⊑ Visita

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> assistito </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Visita </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘assistito’ è un’ istanza del concetto Visita.

-- ∃ (assistito)¯ ⊑ Persona

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> assistito </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
```

```
                <basicC>
                    <atomicC> Persona </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso ‘assistito’ è un’ istanza del concetto Persona.

-- ∃ inverse_of_ assistito ⊑ Persona

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> inverse_of_assistito </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Persona </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_assistito’ è un’ istanza del concetto Persona.

-- ∃ (inverse_of_ assistito)⁻ ⊑ Visita

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> inverse_of_assistito </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Visita </atomicC>
                </basicC>
            </signedC>
        </generalC>
```

</inclusionAssertion>

La prima componente del ruolo inverso di ‘inverse_of_assistito’ è un’ istanza del concetto Visita.

-- Visita ⊑ ∃ assistito. Persona

```
<inclusionAssertion>
        <basicC>
                <atomicC> Visita </atomicC>
        </basicC>
        <generalC>
                <qualifiedExists>
                        <basicR dir="direct">
                                <atomicR> assistito </atomicR>
                        </basicR>
                        <generalC>
                                <signedC sign="positive">
                                    <basicC>
                                            <atomicC>Persona</atomicC>
                                    </basicC>
                                </signedC>
                        </generalC>
                 </qualifiedExists>
        </generalC>
</inclusionAssertion>
```

Ogni Visita possiede (‘assistito’) almeno una Persona

Questa asserzione (Visita ⊑ inverse_of_ assistito. Persona) sta ad indicare che la classe Visita partecipa alla relazione inverse_of_formato_da con la classe Visita con cardinalità minima 1

-- (funct medico)

```
<funct>
      <basicR dir="direct">
              <atomicR> medico </atomicR>
      </basicR>
</funct>
```

Il ruolo atomico, ‘medico’, è stato definito funzionale.

-- medico ⊑ (inverse_of_ medico)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
                <atomicR> medico </atomicR>
        </basicR>
```

```
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> inverse_of_medico </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘medico’ è un’istanza del ruolo inverso di ‘ inverse_of_ medico’.

Questa asserzione di inclusione di ruolo (medico ⊑ (inverse_of_medico)⁻)sta ad indicare che il ruolo ‘inverse_of_medico’ è la relazione inversa della relazione ‘medico’.

-- inverse_of_ medico ⊑ (medico)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> inverse_of_medico </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> medico </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ medico’ è un’istanza del ruolo inverso di ‘assistito’.

Questa asserzione di inclusione di ruolo (inverse_of_ medico ⊑ (medico)⁻)
sta ad indicare che il ruolo ‘medico’ è la relazione inversa della relazione ‘inverse_of_medico’.

-- ∃ medico ⊑ Visita

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> medico </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
```

```
                <basicC>
                    <atomicC> Visista </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘medico’ è un’ istanza del concetto Visita.

-- ∃ (medico)¯ ⊑ Persona

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> medico </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Persona </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘medico’ è un’ istanza del concetto Persona

-- ∃ inverse_of_ medico ⊑ Persona

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> inverse_of_medico </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Persona </atomicC>
                </basicC>
            </signedC>
        </generalC>
```

</inclusionAssertion>

La prima componente del ruolo ‘inverse_of_medico’ è un’ istanza del concetto Persona

-- ∃ (inverse_of_ medico)¯ ⊑ Visita

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                        <atomicR> inverse_of_medico </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                 <basicC>
                        <atomicC> Visita </atomicC>
                 </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘inverse_of_medico’ è un’ istanza del concetto Visita

-- Visita ⊑ ∃ medico. Persona

```
<inclusionAssertion>
        <basicC>
             <atomicC> Visita </atomicC>
        </basicC>
        <generalC>
             <qualifiedExists>
                  <basicR dir="direct">
                        <atomicR> medico </atomicR>
                  </basicR>
                  <generalC>
                        <signedC sign="positive">
                             <basicC>
                                  <atomicC>Persona</atomicC>
                             </basicC>
                        </signedC>
                  </generalC>
             </qualifiedExists>
        </generalC>
</inclusionAssertion>
```

Ogni Visita possiede (‘medico’) almeno un Persona

Questa asserzione (Visita ⊑ ∃ inverse_of_medico. Persona) sta ad indicare che la classe Visita partecipa alla relazione inverse_of_formato_da con la classe Visita con cardinalità minima 1

-- (funct inverse_of_indicazione)

```
<funct>
      <basicR dir="direct">
            <atomicR> inverse_of_indicazione </atomicR>
      </basicR>
</funct>
```

Il ruolo atomico, ‘inverse_of_indicazione’, è stato definito funzionale

Questa asserzione di funzionalità di ruolo (funct inverse_of_indicazione) sta ad indicare la cardinalità (1,1) della relazione ‘inverse_of_indicazione’.

-- indicazione ⊑ (inverse_of_ indicazione)⁻

```
<inclusionAssertion>
      <basicR dir="direct">
            <atomicR> indicazione </atomicR>
      </basicR>
      <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                      <atomicR> inverse_of_indicazione </atomicR>
                </basicR>
            </signedR>
      </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘indicazione’ è un’istanza del ruolo inverso di ‘ inverse_of_ indicazione’.

Questa asserzione di inclusione di ruolo sta ad indicare che il ruolo ‘inverse_of_ indicazione’ è il ruolo inverso del ruolo ‘indicazione’.

-- inverse_of_ indicazione ⊑ (indicazione)⁻

```
<inclusionAssertion>
      <basicR dir="direct">
            <atomicR> inverse_of_indicazione </atomicR>
      </basicR>
      <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
```

```
                    <atomicR> indicazione </atomicR>
                </basicR>
            </signedR>
        </generalR>
    </inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ indicazione’ è un’istanza del ruolo inverso di ‘indicazione’.

Questa asserzione di inclusione di ruolo (inverse_of_indicazione ⊑ (indicazione)¯)
sta ad indicare che il ruolo ‘indicazione’ è la relazione inversa della relazione ‘inverse_of_ indicazione’.

-- ∃ indicazione ⊑ Visita

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> indicazione </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Visita </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘indicazione’ è un’ istanza del concetto Visita.

-- ∃ (indicazione)¯ ⊑ Prescrizione

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> indicazione </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Prescrizione </atomicC>
                </basicC>
```

```
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘indicazione’ è un’ istanza del concetto Prescrizione.

-- ∃ inverse_of_ indicazione ⊑ Prescrizione

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> inverse_of_indicazione </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Prescrizione </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_indicazione’ è un’ istanza del concetto Prescrizione.

--∃ (inverse_of_ indicazione)¯ ⊑ Visita

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> inverse_of_indicazione </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Visita </atomicC>
                </basicC>
             </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘inverse_of_indicazione’ è un’ istanza del concetto Visita.

-- Prescrizione ⊑ ∃ inverse_of_indicazione. Visita

```
<inclusionAssertion>
        <basicC>
                <atomicC> Prescrizione </atomicC>
        </basicC>
        <generalC>
                <qualifiedExists>
                        <basicR dir="direct">
                                <atomicR> inverse_of_indicazione</atomicR>
                        </basicR>
                        <generalC>
                                <signedC sign="positive">
                                    <basicC>
                                            <atomicC> Visita </atomicC>
                                    </basicC>
                                </signedC>
                        </generalC>
                 </qualifiedExists>
        </generalC>
</inclusionAssertion>
```

Ogni Prescrizione possiede (‘inverse_of_indicazione’) almeno unaVisita

Questa asserzione (Prescrizione ⊑ ∃ inverse_of_indicazione. Visita) sta ad indicare che la classe Prescrizione partecipa alla relazione inverse_of_ indicazione con la classe Visita con cardinalità minima 1

Conclusioni:

1) *<!-- IntersectionOf not totally expressible in DL-liteA -->*

La classe Visita è espressa in DL-lite come l’intersezione di:
la restrizione (che ha lo scopo di indicare la cardinalita minima della relazione assistito con la classe Persona)
la restrizione (che ha lo scopo di indicare la cardinalita minima della relazione medico con la classe Persona)

```
Class(a:Visita complete
        intersectionOf( restriction(a: assistito someValuesFrom(a: Persona))
                        restriction(a: medico someValuesFrom(a: Persona))))
```

In DL-liteA il concetto di intersezione non è totalmente esprimibile.

2) *<!-- NOT IN DL-LiteA!There is a conflict :the atomic Role 'assistito' is qualified exists then it cannot be functional*
-->

La base di conoscenza in DL-lite-A K=<T,A>, è ottenuta attraverso delle restrizioni: ovvero T (TBox) è una DL-liteFR Tbox che soddisfa determinate condizioni. In particolare, in questo caso:
-- Per ogni ruolo atomico e inverso di un ruolo atomico Q compare in un concetto della forma ∃Q.C, le asserzioni (funct Q) e (funct Q^{-}) non sono in T.

Poiché il ruolo inverse_of_ effettua compare nella seguente asserzione:
Visita ⊑ ∃ assistito. Persona e quindi è quantificato esistenzialmente,
l'asserzione (funct assistito) non è in T, e quindi il ruolo non può essere funzionale.

3) *<!-- IntersectionOf not totally expressible in DL-liteA -->*

La classe Prescrizione è espressa in DL-lite come l'intersezione di:
l'unione di classi(Farmaco , RichiestaRicovero)
la restrizione (che ha lo scopo di indicare la cardinalita minima della relazione inverse_of_indicazione con la classe Visita)

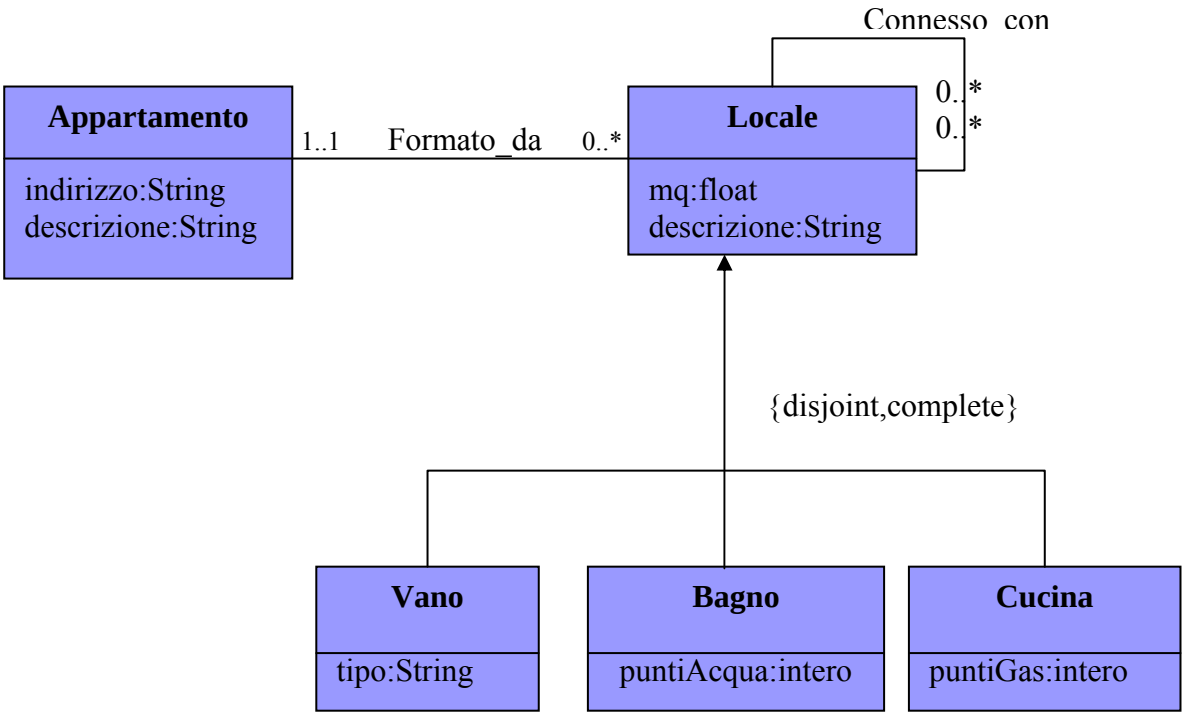
```
Class( a: Prescrizione complete
        intersectionOf( unionOf(a: Farmaco a: RichiestaRicovero)
                        restriction(a:inverse_of_indicazione someValuesFrom(a:Visita))))
```

In DL-liteA il concetto di intersezione non è totalmente esprimibile.

4) *<!-- UnionOf not totally expressible in DL-liteA -->*

La classe Prescrizione è è la generalizzazione completa e disgiunta delle classi (Farmaco , RichiestaRicovero)
In DL-liteA possimo esprimere le proprietà di disgiunzione tramite la seguente asserzione di inclusione di concetti: Farmaco ⊑ ¬ RichiestaRicovero e RichiestaRicovero ⊑ ¬ Farmaco
e le proprietà di completezza con le seguenti asserzioni di inclusione di concetti
Farmaco ⊑ Prescrizione, RichiestaRicovero ⊑ Prescrizione

Compito d’esame 12 Settembre 2005



OWL-DL

Classi: Appartamento, Locale, Vano, Bagno, Cucina
Relazioni: formato_da, connesso_con
Proprietà: indirizzo, descrizione_app, mq, descrizione_loc, tipo, puntiAcqua, puntiGas

CLASSI

- **Appartamento**

L'oggetto "Appartamento" è stato tradotto in una classe Protègè che ha come proprietà gli attributi : indirizzo, descrizione.
La classe partecipa alla relazione formato_da con la cardinalità 0..*

Class(a:**Appartamento** partial)
DisjointClasses(a:**Appartamento** a:**Locale**)

La classe Appartamento è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalla classe Locale.

- **Locale**

L'oggetto "Locale" è stato tradotto in una classe Protègè che ha come proprietà gli attributi della classe: mq e descrizione.
La classe Locale è superclasse nella generalizzazione che ha come sottoclassi Vano, Bagno, Cucina (che sono tra di loro disgiunte).
La generalizzazione è completa e questo vincolo è stato espresso attraverso la restrizione :
Vano U Bagno U Cucina inserita nelle condizioni necessarie e sufficienti.
La classe partecipa alla relazione connesso_con.con se stessa con cardinalità 0..*.
Quest'ultima viene tradotta con la proprietà simmetrica ovvero un locale a è connesso con un locale b allora il locale b è connesso con il locale a. e la proprietà transitiva (se il locale a e connesso con il locale b e il locale b è connesso con il locale c allora il locale a è connesso con il locale c).
La cardinalità 0..* è quella che viene indicata di default e perciò non è necessario indicarla attraverso restrizioni cardinali.
Inoltre partecipa alla relazione inverse_of_formato_da con la classe Appartamento con cardinalità 1..1 (un locale può appartenere ad un unico appartamento).La cardinalità minima viene espressa tramite la restrizione.

Class(a:**Locale** complete
intersectionOf(
unionOf(a:**Vano** a:**Cucina** a:**Bagno**)
restriction(a:inverse_of_formato_da someValuesFrom(a:**Appartamento**))))

DisjointClasses(a:**Appartamento** a:**Locale**)

La classe Locale è *complete:* è descritta utilizzando condizioni necessarie e sufficienti.
E' l'unione delle classi Vano, Bagno, Cucina ed è disgiunta dalla classe Appartamento.

• **Vano:**
L'oggetto "Vano" è stato tradotto in una classe Protègè ed è una sottoclasse di Locale da cui eredita le proprietà ed è disgiunta con le classi Bagno e Cucina.
Queste classi sono tra di loro disgiunte e complete quindi un individuo che appartiene alla classe Vano non può appartenere alla classe Bagno e Cucina.

Class(a:**Vano** partial a:**Locale**)
DisjointClasses(a:**Vano** a:**Cucina**)
DisjointClasses(a:**Vano** a:**Bagno**)

La classe Vano è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalla classe Cucina e Bagno.

• **Bagno:**
L'oggetto "Bagno" è stato tradotto in una classe Protègè ed è una sottoclasse di Locale da cui eredita le proprietà ed è disgiunta con le classi Vano e Cucina.
Queste classi sono tra di loro disgiunte e complete quindi un individuo che appartiene alla classe Bagno non può appartenere alla classe Vano e Cucina.

Class(a:**Bagno** partial a:**Locale**)
DisjointClasses(a:**Bagno** a:**Cucina**)
DisjointClasses(a:**Vano** a:**Bagno**)

La classe Bagno è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalla classe Vano e Cucina.

• **Cucina:**
L'oggetto "Cucina" è stato tradotto in una classe Protègè ed è una sottoclasse di Locale da cui eredita le proprietà ed è disgiunta con le classi Vano e Bagno.
Queste classi sono tra di loro disgiunte e complete quindi un individuo che appartiene alla classe Cucina non può appartenere alla classe Vano e Bagno.

Class(a:**Cucina** partial a:**Locale**)
DisjointClasses(a:**Bagno** a:**Cucina**)
DisjointClasses(a:**Vano** a:**Cucina**)

La classe Cucina è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalla classe Vano e Bagno.

PROPRIETA' (DataProperty)

Gli attributi della classe sono stati tradotti come DataProperty e hanno come la proprietà Functional poiché sono associati ad un solo valore per ogni individuo.

- Gli attributi indirizzo e descrizione_appartamento hanno come dominio la classe Appartamento e come range string

DatatypeProperty(a:**indirizzo** Functional

domain(a:**Appartamento**)
range(xsd:string))

DatatypeProperty(a:**desc_appartamento** Functional
domain(a:**Appartamento**)
range(xsd:string))

- Gli attributi mq e descrizione_locale hanno come dominio la classe Locale e come range rispettivamente string e float

DatatypeProperty(a:**desc_locale** Functional
domain(a:**Locale**)
range(xsd:string))

DatatypeProperty(a:**mq** Functional
domain(a:**Locale**)
range(xsd:float))

- L'attributo puntiAcqua ha come dominio la classe Bagno e come range int

DatatypeProperty(a:**puntiAcqua** Functional
domain(a:**Bagno**)
range(xsd:int))

- L'attributo puntiGas ha come dominio la classe Cucina e come range int

DatatypeProperty(a:**puntiGas** Functional
domain(a:**Cucina**)
range(xsd:int))

- L'attributo tipo ha come dominio la classe Vano e come range string

DatatypeProperty(a:**tipo** Functional
domain(a:**Vano**)
range(xsd:string))

<u>RELAZIONI</u> (ObjectProperty)

Le associazioni sono state tradotte in relazioni tra le classi (ObjectProperty) e per ogni relazione è stata definita una relazione inversa:

formato_da←→inverse_of_formato_da
connesso_con←→connesso_con

• **formato_da**

L'associazione formato_da viene tradotta con un ObjectProperty che ha come dominio la classe Appartamento come range la classe Locale e come proprietà inverseFunctional in quanto la relazione inversa è Functional..

ObjectProperty(a:**formato_da** InverseFunctional
inverseOf(a:inverse_of_formato_da)
domain(a:**Appartamento**)
range(a:**Locale**))

La sua relazione inversa è inverse_of_formato_da ha come dominio la classe Locale e come range la classe Appartamento ed è stata definita Functional per indicare il vincolo di cardinalità per il limite superiore (se il locale a ha come appartamento b, e se il locale a ha come appartamento c, allora l'appartamento b e c coincidono).

ObjectProperty(a:**inverse_of_formato_da** Functional
inverseOf(a:formato_da)
domain(a:**Locale**)
range(a:**Appartamento**))

• **connesso_con**

La relazione connesso_con viene tradotta con un ObjectProperty che ha come dominio la classe Locale come range la classe Locale. E' una relazione che collega elementi della stessa classe Locale. Per esprimere questo è stata utilizzata la proprietà simmetrica (se il locale a e connesso con il locale b allora il locale b è connesso con il locale a) e la proprietà transitiva (se il locale a e connesso con il locale b e il locale b è connesso con il locale c allora il locale a è connesso con il locale c). La relazione inversa è proprio se stessa.

ObjectProperty(a:**connesso_con** Transitive Symmetric
inverseOf(a:connesso_con)
domain(a:**Locale**)
range(a:**Locale**))

DL-LiteA

Asserzioni di inclusione di concetti

Appartamento ⊑ ¬ Locale
Locale ⊑ ¬ Appartamento
Vano ⊑ Locale
Vano ⊑ ¬ Bagno
Vano ⊑ ¬ Cucina
Bagno ⊑ Locale
Bagno ⊑ ¬ Cucina
Bagno ⊑ ¬ Vano
Cucina ⊑ Locale
Cucina ⊑ ¬ Vano
Cucina ⊑ ¬ Bagno

Appartamento ⊑ δ(desc_appartamento)
Appartamento ⊑ δ(indirizzo)
Locale ⊑ δ(desc_locale)
Locale ⊑ δ(mq)
Vano ⊑ δ(tipo)
Bagno ⊑ δ(puntiAcqua)
Cucina ⊑ δ(puntiGas)

∃ formato_da ⊑ Appartamento
∃ $(\text{formato\_da})^{-}$ ⊑ Locale
∃ inverse_of_formato_da ⊑ Locale
∃ $(\text{inverse\_of\_formato\_da})^{-}$ ⊑ Appartamento
∃ connesso_con ⊑ Locale
∃ $(\text{connesso\_con})^{-}$ ⊑ Locale

Locale ⊑ ∃ inverse_of_formato_da.Appartamento

Asserzioni di funzionalità di attributi di concetto

funct (indirizzo)
funct (desc_appartamento)
funct (mq)
funct (desc_locale)
funct (tipo)
funct (puntiacqua)
funct (puntiGas)

Asserzioni di inclusione di domini di valori

ρ(desc_appartamento) ⊑ xs:string
ρ(indirizzo) ⊑ xs:string

ρ(desc_locale) ⊑ xs:string
ρ(mq) ⊑ xs:float
ρ(tipo) ⊑ xs:string
ρ(puntiAcqua) ⊑ xs:int
ρ(puntiGas) ⊑ xs:int

Asserzioni di inclusione di ruoli

formato_da ⊑ (inverse_of_formato_da)$^{-}$
inverse_of_formato_da ⊑ (formato_da)$^{-}$
connesso_con ⊑ (connesso_con)$^{-}$

- **Alfabeto**

L'alfabeto che descrive lo schema UML è formato dai seguenti elementi:
da concetti generali (atomicC), attributi di concetti (atomicCA) , ruoli generali (atomicR).

<alphabet>

<atomicC> **Appartamento** </atomicC>
<atomicC> **Locale** </atomicC>
<atomicC> **Vano** </atomicC>
<atomicC> **Bagno** </atomicC>
<atomicC> **Cucina** </atomicC>

<atomicCA> **indirizzo** </atomicCA>
<atomicCA> **desc_appartamento** </atomicCA>
<atomicCA> **mq** </atomicCA>
<atomicCA> **desc_locale** </atomicCA>
<atomicCA> **tipo** </atomicCA>
<atomicCA> **puntiAcqua** </atomicCA>
<atomicCA> **puntiGas** </atomicCA>

<atomicR> **formato_da** </atomicR>
<atomicR> **inverse_of_formato_da** </atomicR>
<atomicR> **connesso_con** </atomicR>

</alphabet>

Le classi Appartamento, Locale, Vano, Bagno, Cucina sono concetti atomici.
Le proprietà delle classi indirizzo, desc_appartamento, desc_locale, mq, tipo, puntiAcqua, puntiGas sono attributi di concetto.
Le relazioni formato_da, inverse_of_formato_da, connesso_con, sono ruoli generali.

• **Tbox**

La Tbox è usata per rappresentare la conoscenza intensionale attraverso asserzioni di inclusione.

-- Appartamento ⊑ ¬ Locale

```
<inclusionAssertion>
    <basicC>
          <atomicC> Appartamento </atomicC>
    </basicC>
    <generalC>
          <signedC sign="negative">
               <basicC>
                    <atomicC> Locale </atomicC>
               </basicC>
          </signedC>
    </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni concetto di Appartamento non è un concetto di Locale (sign="**negative**" vuole indicare la negazione del concetto).

-- Locale ⊑ ¬ Appartamento

```
<inclusionAssertion>
    <basicC>
          <atomicC> Locale </atomicC>
    </basicC>
    <generalC>
          <signedC sign="negative">
               <basicC>
                    <atomicC> Appartamento </atomicC>
               </basicC>
          </signedC>
    </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni concetto di Locale non è un concetto di Appartamento (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di inclusione di concetto (Appartamento ⊑ ¬Locale , Locale ⊑¬ Appartamento) stanno ad indicare la disgiunzione tra i concetti Appartamento e Locale.

-- Vano ⊑ Locale

```
<inclusionAssertion>
```

```
    <basicC>
        <atomicC> Vano </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <atomicC> Locale </atomicC>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni concetto di Vano è un concetto di Locale.

Questa asserzione sta ad indicare che la classe Vano è una sottoclasse di Locale.

-- Vano ⊑ ¬ Bagno

```
<inclusionAssertion>
    <basicC>
        <atomicC> Vano </atomicC>
    </basicC>
    <generalC>
        <signedC sign="negative">
            <basicC>
                <atomicC> Bagno </atomicC>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni concetto di Vano non è un concetto di Bagno (sign="**negative**" vuole indicare la negazione del concetto).

-- Vano ⊑ ¬ Cucina

```
<inclusionAssertion>
    <basicC>
        <atomicC> Vano </atomicC>
    </basicC>
    <generalC>
        <signedC sign="negative">
            <basicC>
                <atomicC> Cucina </atomicC>
            </basicC>
        </signedC>
```

```
    </generalC>
</inclusionAssertion>
```

L’asserzione di inclusione vuole indicare che ogni concetto di Vano non è un concetto di Cucina (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di concetto (Vano ⊑ ¬Bagno , Vano ⊑ ¬ Cucina) stanno ad indicare la disgiunzione tra il concetto di Vano e i concetti Bagno e Cucina.

-- Bagno ⊑ Locale

```
<inclusionAssertion>
    <basicC>
        <atomicC> Bagno </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <atomicC> Locale </atomicC>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

L’asserzione di inclusione concetto di vuole indicare che ogni concetto di Bagno è un concetto di Locale.

Questa asserzione sta ad indicare che la classe Bagno è una sottoclasse di Locale.

-- Bagno ⊑ ¬ Cucina

```
<inclusionAssertion>
    <basicC>
        <atomicC> Bagno </atomicC>
    </basicC>
    <generalC>
        <signedC sign="negative">
            <basicC>
                <atomicC> Cucina </atomicC>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

L’asserzione di inclusione di concetto vuole indicare che ogni concetto di Bagno non è un concetto di Cucina (sign="**negative**" vuole indicare la negazione del concetto).

-- Bagno ⊑ ¬ Vano

```
<inclusionAssertion>
    <basicC>
          <atomicC> Bagno </atomicC>
    </basicC>
    <generalC>
          <signedC sign="negative">
               <basicC>
                    <atomicC> Vano </atomicC>
               </basicC>
          </signedC>
    </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni concetto di Bagno non è un concetto di Vano (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di inclusione di concetto (Bagno ⊑ ¬ Vano **,** *Bagno ⊑ ¬ Cucina) stanno ad indicare la disgiunzione tra il concetto di Vano e i concetti Bagno e Cucina.*

-- Cucina ⊑ Locale

```
<inclusionAssertion>
    <basicC>
          <atomicC> Cucina </atomicC>
    </basicC>
    <generalC>
          <signedC sign="positive">
               <basicC>
                    <atomicC> Locale </atomicC>
               </basicC>
          </signedC>
     </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto di vuole indicare che ogni concetto di Cucina è un concetto di Locale.

Questa asserzione sta ad indicare che la classe Cucina è una sottoclasse di Locale.

-- Cucina ⊑ ¬ Vano

```
<inclusionAssertion>
    <basicC>
          <atomicC> Cucina </atomicC>
```

```
    </basicC>
    <generalC>
          <signedC sign="negative">
               <basicC>
                    <atomicC> Vano </atomicC>
               </basicC>
          </signedC>
    </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto di vuole indicare che ogni concetto di Cucina non è un concetto di Vano(sign="**negative**" vuole indicare la negazione del concetto).

-- Cucina ⊑ ¬ Bagno

```
<inclusionAssertion>
    <basicC>
          <atomicC> Cucina </atomicC>
    </basicC>
    <generalC>
          <signedC sign="negative">
               <basicC>
                    <atomicC> Bagno </atomicC>
               </basicC>
          </signedC>
    </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni Cucina non è un Bagno (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di inclusione di concetto (Cucina ⊑ ¬ Vano , Cucina ⊑ ¬ Bagno) stanno ad indicare la disgiunzione tra il concetto di Cucina e i concetti Bagno e Vano.

-- funct (indirizzo)

```
<funct>
      <atomicCA> indirizzo </atomicCA>
</funct>
```

L'attributo di concetto, 'indirizzo', è stato definito funzionale.

-- Appartamento ⊑ δ(indirizzo)

```
<inclusionAssertion>
    <basicC>
          <atomicC> Appartamento </atomicC>
```

```
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> indirizzo </atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto Appartamento ha associato un attributo di concetto, ‘indirizzo’.
Dato un attributo di concetto, ‘indirizzo’, chiamiamo dominio di ‘indirizzo’ denotato con δ(indirizzo) il set di oggetti che ‘indirizzo’ collega a valori.

-- ρ(indirizzo) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> indirizzo </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV>xs:string</predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘indirizzo’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto,‘indirizzo’ , chiamiamo range di ‘indirizzo’ denotato con ρ(indirizzo) il set di valori che ‘indirizzo’ collega a oggetti.

-- funct (desc_appartamento)

```
<funct>
    <atomicCA> desc_appartamento </atomicCA>
</funct>
```

L’attributo di concetto, ‘desc_appartamento’, è stato definito funzionale.

-- Appartamento ⊑ δ(desc_appartamento)

```
<inclusionAssertion>
    <basicC>
        <atomicC>Appartamento</atomicC>
    </basicC>
```

```
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA>desc_appartamento</atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto Appartamento ha associato un attributo di concetto ‘desc_appartamento’.
Dato un attributo di concetto ‘desc_appartamento’ chiamiamo dominio di ‘desc_appartamento’ denotato con δ(desc_appartamento) il set di oggetti che ‘desc_appartamento’ collega a valori.

-- ρ(desc_appartamento) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> desc_appartamento </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘desc_appartamento’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘desc_appartamento’ chiamiamo range di ‘desc_appartamento’ denotato con ρ(desc_appartamento) il set di valori che ‘desc_appartamento’ collega a oggetti.

-- funct (mq)

```
<funct>
    <atomicCA> mq </atomicCA>
</funct>
```

L’attributo di concetto, ‘mq’, è stato definito funzionale.

-- Locale ⊑ δ(mq)

```
<inclusionAssertion>
    <basicC>
        <atomicC> Locale </atomicC>
    </basicC>
    <generalC>
```

```
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> mq </atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto di Locale ha associato un attributo di concetto, ‘mq’.
Dato un attributo di concetto ‘mq ‘chiamiamo dominio di ‘mq’ denotato con δ(mq) il set di oggetti che ‘mq’ collega a valori.

-- ρ(mq) ⊑ xs:float

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> mq </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV>xs:float</predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘mq’ è un dominio di valori predefinito xs:float.
Dato un attributo di concetto ‘mq’ chiamiamo range di ‘mq’ denotato con ρ(mq) il set di valori che ‘mq’ collega a oggetti.

-- funct (desc_locale)

```
<funct>
    <atomicCA> desc_locale </atomicCA>
</funct>
```

L’attributo di concetto, ‘desc_locale’, è stato definito funzionale.

-- Locale ⊑ δ(desc_locale)

```
<inclusionAssertion>
    <basicC>
        <atomicC> Locale </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
```

```
            <basicC>
                <CADomain>
                    <atomicCA> desc_locale </atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto di Locale ha associato un attributo di concetto, ‘desc_locale’.
Dato un attributo di concetto desc_locale chiamiamo dominio di ‘desc_locale’ denotato con δ(desc_locale) il set di oggetti che ‘desc_locale’ collega a valori.

-- ρ(desc_locale) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> desc_locale </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘desc_locale’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto ‘desc_locale’ chiamiamo range di ‘desc_locale’ denotato con ρ(desc_locale) il set di valori che ‘desc_locale’ collega a oggetti.

-- funct (tipo)

```
<funct>
    <atomicCA> tipo </atomicCA>
</funct>
```

L’attributo di concetto, ‘tipo’, è stato definito funzionale.

-- Vano ⊑ δ(tipo)

```
<inclusionAssertion>
    <basicC>
        <atomicC> Vano </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
```

```
                    <CADomain>
                        <atomicCA> tipo </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto di Vano ha associato un attributo di concetto, ‘tipo’.
Dato un attributo di concetto tipo chiamiamo dominio di ‘tipo’ denotato con δ(tipo) il set di oggetti che ‘tipo’ collega a valori.

-- ρ(tipo) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> tipo </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV>xs:string</predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘tipo’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto ‘tipo’ chiamiamo range di ‘tipo’ denotato con ρ(tipo) il set di valori che ‘tipo’ collega a oggetti.

-- funct (puntiacqua)

```
<funct>
    <atomicCA> puntiacqua </atomicCA>
</funct>
```

L’attributo di concetto, ‘puntiacqua’, è stato definito funzionale.

-- Bagno ⊑ δ(puntiacqua)

```
<inclusionAssertion>
    <basicC>
        <atomicC> Bagno </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
```

```
                        <atomicCA> puntiacqua </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto di Bagno ha associato un attributo di concetto, ‘puntiacqua’.
Dato un attributo di concetto ‘puntiacqua’ chiamiamo dominio di ‘puntiacqua’ denotato con δ(puntiacqua) il set di oggetti che ‘puntiacqua’ collega a valori.

-- ρ(puntiacqua) ⊑ xs:int

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> puntiacqua </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:int </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘puntiacqua’ è un dominio di valori predefinito xs:int.
Dato un attributo di concetto ‘puntiacqua’ chiamiamo range di indirizzo denotato con ρ(puntiacqua) il set di valori che ‘puntiacqua’ collega a oggetti.

-- funct (puntiGas)

```
<funct>
    <atomicCA> puntiGas </atomicCA>
</funct>
```

L’attributo di concetto, ‘puntiGas’, è stato definito funzionale.

-- Cucina ⊑ δ(puntiGas)

```
<inclusionAssertion>
    <basicC>
        <atomicC> Cucina </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> puntiGas </atomicCA>
```

```
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto di Cucina ha associato un attributo di concetto, ‘puntiGas’.
Dato un attributo di concetto ‘puntiGas’ chiamiamo dominio di ‘puntiGas’ denotato con δ(puntiGas) il set di oggetti che ‘puntiGas’ collega a valori.

-- ρ(puntiGas) ⊑ xs:int

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> puntiGas </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:int </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘puntiGas’ è un dominio di valori predefinito xs:int.
Dato un attributo di concetto ‘puntiGas’ chiamiamo range di indirizzo denotato con ρ(puntiGas) il set di valori che ‘puntiGas’ collega a oggetti.

-- formato_da ⊑ (inverse_of_formato_da)⁻

```
<inclusionAssertion>
    <basicR dir="direct">
        <atomicR> formato_da </atomicR>
     </basicR>
    <generalR>
         <signedR sign="positive">
             <basicR dir="inverse">
                    <atomicR> inverse_of_formato_da </atomicR>
              </basicR>
             </signedR>
     </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘formato_da’ è un’istanza del ruolo inverso di ‘ inverse_of_formato_da’.

Questa asserzione di inclusione di ruolo (formato_da ⊑ (inverse_of_formato_da)⁻)sta ad indicare che il ruolo ‘inverse_of_formato_da’ è la relazione inversa della relazione ‘formato_da’.

-- inverse_of_ formato_da ⊑ (formato_da)⁻

```
<inclusionAssertion>
    <basicR dir="direct">
                                <atomicR> inverse_of_ formato_da </atomicR>
     </basicR>
    <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                        <atomicR> formato_da </atomicR>
                 </basicR>
             </signedR>
     </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_formato_da’ è un’istanza del ruolo inverso di ‘formato_da’.

Questa asserzione di inclusione di ruolo (inverse_of_formato_da ⊑ (formato_da)⁻)
sta ad indicare che il ruolo ‘formato_da’ è la relazione inversa della relazione ‘inverse_of_formato_da’.

-- ∃ formato_da ⊑ Appartamento

```
<inclusionAssertion>
    <basicC>
            <exists>
                <basicR dir="direct">
                        <atomicR> formato_da </atomicR>
                </basicR>
            </exists>
    </basicC>
    <generalC>
             <signedC sign="positive">
                <basicC>
                        <atomicC> Appartamento </atomicC>
                </basicC>
             </signedC>
    </generalC>
 </inclusionAssertion>
```

La prima componente del ruolo ‘formato_da’ è un’ istanza del concetto Appartamento.

-- ∃ (formato_da)⁻ ⊑ Locale

```
<inclusionAssertion>
    <basicC>
```

```
            <exists>
                <basicR dir="inverse">
                        <atomicR> formato_da </atomicR>
                </basicR>
            </exists>
    </basicC>
    <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Locale </atomicC>
                </basicC>
            </signedC>
    </generalC>
 </inclusionAssertion>
```

La prima componente del ruolo inverso di‘formato_da’ è un’ istanza del concetto Locale. (Locale appartiene a..)

-- ∃ inverse_of_formato_da ⊑ Locale

```
<inclusionAssertion>
    <basicC>
            <exists>
                <basicR dir="direct">
                        <atomicR> inverse_of_formato_da </atomicR>
                </basicR>
            </exists>
    </basicC>
    <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Locale </atomicC>
                </basicC>
            </signedC>
    </generalC>
 </inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_formato_da’ è un’ istanza del concetto Locale. (Locale appartiene a..)

-- ∃ (inverse_of_formato_da)⁻ ⊑ Appartamento

```
 <inclusionAssertion>
    <basicC>
            <exists>
                <basicR dir="inverse">
                        <atomicR>inverse_of_formato_da</atomicR>
```

```
                </basicR>
            </exists>
    </basicC>
    <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC>Appartamento</atomicC>
                </basicC>
            </signedC>
    </generalC>
 </inclusionAssertion>
```

La prima componente del ruolo inverso di ‘inverse_of_formato_da’ è un’ istanza del concetto Appartamento.

-- Locale ⊑ ∃ inverse_of_formato_da.Appartamento

```
<inclusionAssertion>
    <basicC>
            <atomicC> Locale </atomicC>
    </basicC>
    <generalC>
            <qualifiedExists>
                    <basicR dir="direct">
                            <atomicR> inverse_of_formato_da </atomicR>
                    </basicR>
                    <generalC>
                            <signedC sign="positive">
                                <basicC>
                                        <atomicC> Appartamento </atomicC>
                                </basicC>
                            </signedC>
                    </generalC>
            </qualifiedExists>
    </generalC>
 </inclusionAssertion>
```

Ogni Locale appartiene(‘inverse_of_formato _da’) almeno ad un Appartamento.

Questa asserzione (Locale ⊑ ∃ inverse_of_formato_da.Appartamento) sta ad indicare che la classe Locale partecipa alla relazione inverse_of_formato_da con la classe Appartamento con cardinalità minima 1 (un locale deve appartenere almeno ad un appartamento).

-- connesso_con ⊑ (connesso_con)⁻

```
<inclusionAssertion>
    <basicR dir="direct">
```

```
                    <atomicR> connesso_con </atomicR>
    </basicR>
    <generalR>
        <signedR sign="positive">
            <basicR dir="inverse">
                <atomicR> connesso_con </atomicR>
            </basicR>
        </signedR>
    </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘connesso_con’ è un’istanza del ruolo inverso di ‘inverse_of_ connesso_con’.

Questa asserzione di inclusione di ruolo (connesso_con ⊑ (connesso_con)⁻) sta ad indicare che ‘inverse_of_ connesso_con’ è la relazione inversa della relazione ‘connesso_con’.

-- ∃ connesso_con ⊑ Locale

```
<inclusionAssertion>
    <basicC>
        <exists>
            <basicR dir="direct">
                <atomicR> connesso_con </atomicR>
            </basicR>
        </exists>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <atomicC> Locale </atomicC>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘connesso_con’ è un’ istanza del concetto Locale.

-- ∃ (connesso_con)⁻ ⊑ Locale

```
<inclusionAssertion>
    <basicC>
        <exists>
            <basicR dir="inverse">
                <atomicR> connesso_con </atomicR>
```

```
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Locale </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di 'connesso_con' è un' istanza del concetto Locale.

Conclusioni:

1) *<!-- IntersectionOf not totally expressible in DL-liteA -->*

La classe Locale è espressa in DL-lite come l'intersezione di:
l'unione di tre classi(Vano, Cucina, Bagno)
la restrizione (che ha lo scopo di indicare la cardinalita minima della relazione inverse_of_ formato_da con la classe Appartamento)

```
Class( a: Locale complete
    intersectionOf ( unionOf(a: Vano a: Cucina a: Bagno)
                    restriction(a:inverse_of_formato_da someValuesFrom(a:Appartamento))))
```

In DL-liteA il concetto di intersezione non è totalmente esprimibile.

2) *<!-- UnionOf not totally expressible in DL-liteA -->*

La classe Locale è la generalizzazione completa e disgiunta delle classi Vano, Bagno, Cucina. In DL-liteA possimo esprimere le proprietà di disgiunzione tramite la seguente asserzione di inclusione di concetti: Vano ⊑ ¬Bagno, Vano ⊑ ¬ Cucina, Bagno ⊑ ¬ Cucina, Bagno ⊑ ¬ Vano Cucina ⊑ ¬ Vano, Cucina ⊑ ¬ Bagno e le proprietà di completezza con le seguenti asserzioni di inclusione di concetti Cucina ⊑ Locale, Bagno ⊑ Locale

L'unione di queste tre classi non è completamente esprimibile in DL-liteA.

3) *<!-- TransitiveProperty not expressible in DL-liteA-->*

La relazione 'connesso_con' ha come dominio la classe Locale e come Range la classe Locale per cui valgono le proprietà di simmetria e di transitività:

ObjectProperty(a:connesso_con Transitive Symmetric
inverseOf(a:connesso_con)
domain(a:Locale)
range(a:Locale))

La proprietà di transitività del ruolo non è esprimibile in DL-liteA.

.

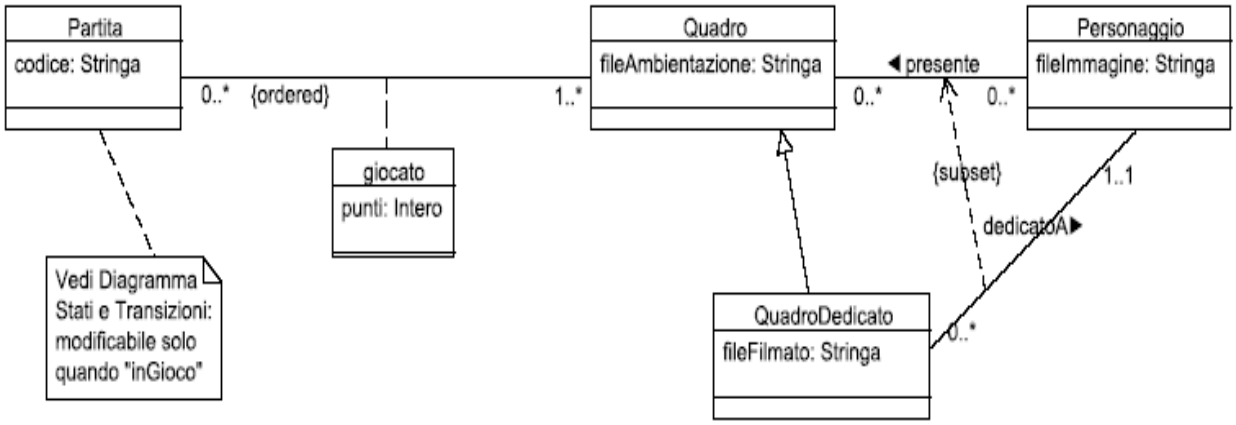
4) *<!-- NOT IN DL-LiteA!There is a conflict :the atomic Role 'inverse_of_formato_da' is qualified exists then it cannot be functional-->*

La base di conoscenza in DL-lite-A K=<T,A>, è ottenuta attraverso delle restrizioni: ovvero T (TBox) è una DL-lite$_{FR}$ Tbox che soddisfa determinate condizioni. In particolare, in questo caso:
-- Per ogni ruolo atomico e inverso di un ruolo atomico Q compare in un concetto della forma ∃Q.C, le asserzioni (funct Q) e (funct Q^{-}) non sono in T.

Poiché il ruolo i inverse_of_ formato_da compare nella seguente asserzione:

Locale ⊑ inverse_of_formato_da.Appartamento e quindi è quantificato esistenzialmente, l'asserzione (funct inverse_of_ formato_da) non è in T, e quindi il ruolo non può essere funzionale.

Compito d’esame 28 marzo 2006



OWL-DL

Classi: Partita, Quadro, QuadroDedicato, Personaggio
Proprietà: codice, fileAmbientazione, fileAnimazione, fileImmagine
Relazioni: giocato, presente, dedicatoA

Non è possibile esprimere l'attributo 'punti' dell'associazione 'giocato'. Per poter esprimere 'ordered' bisognerebbe introdurre una nuova relazione 'ordine' che ha come attributo di ruolo 'numero'.

CLASSI

- **Partita**

L'oggetto "Partita" è stato tradotto in una classe Protègè che ha come proprietà l'attributo: 'codice'.
La classe partecipa alla relazione ordina con la classe Quadro con cardinalità 1..*.

Class(a: **Partita** complete
    restriction(a: **giocato** someValuesFrom(a: **Quadro**)))
DisjointClasses(a: **Quadro** a: **Partita**)
DisjointClasses(a: **Partita** a: **Personaggio**)

La classe Partita è definita *complete* perché ha tra le condizioni necessarie e sufficienti le restrizioni ad indicare il vincolo di cardianiltà minima con il quale partecipa alla relazione 'giocato' con la classe Quadro.

- **Quadro**

L'oggetto "Quadro" è stato tradotto in una classe Protègè che ha come proprietà l'attributo: fileAmbientazione che viene tradotto come dataproperty con range string.
La classe è superclasse della classe QuadroSpeciale.
La classe partecipa alla relazione 'presente' con la classe Personaggio con cardinalità 0..* è quella che viene indicata di default e perciò non è necessario indicarla attraverso restrizioni cardinali.

Class(a: **Quadro** partial)
DisjointClasses(a: **Quadro** a: **Partita**)
DisjointClasses(a: **Quadro** a: **Personaggio**)

La classe Quadro è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalla classe Partita e Personaggio.

- **QuadroDedicato**

L'oggetto "QuadroDedicato" è stato tradotto in una classe Protègè che ha come proprietà l'attributo fileFilmato che viene tradotto come dataproperty con range string.
La classe è sottoclasse della classe Quadro(relazione ISA).
La classe partecipa alla relazione 'dedicatoA' con la classe Personaggio con cardinalità 1..1(ogni QuadroDedicato è dedicato ad un solo Personaggio).
(che è una relazione derivata della relazione contiene)

Class(a: **QuadroDedicato** complete
restriction(a: **dedicatoA** someValuesFrom(a: **Personaggio**)))
Class(a: **QuadroDedicato** partial a: **Quadro**)

La classe QuadroDedicato è *complete:* è descritta utilizzando ha tra le condizioni necessarie e sufficienti le restrizioni ad indicare il vincolo di cardianiltà minima con il quale partecipa alla relazione 'dedicatoA' con la classe Personaggio.
La classe QuadroDedicato è definita *partial* perché ha almeno una condizione necessaria.

• **Personaggio**
L'oggetto "Personaggio" è stato tradotto in una classe Protègè che ha come proprietà l'attributo: fileImmagine tradotto come dataproperty con range string.
La classe partecipa alla relazione 'inverse_of_presente' con la classe Quadro con cardinalità 0..*, e alla relazione 'inverse_of_dedidataA' con la classe QuadroDedicato con cardinalità 0..*,

Class(a: **Personaggio** partial)
DisjointClasses(a: **Quadro** a: **Personaggio**)
DisjointClasses(a: **Partita** a: **Personaggio**)

La classe Partita è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta con le classi Quadro e Partita.

PROPRIETA' (DataProperty)

Gli attributi della classe sono stati tradotti come DataProperty e hanno come la proprietà Functional poiché sono associati ad un solo valore per ogni individuo.

- L'attributo 'codice' ha come dominio la classe Partita e come range string.

DatatypeProperty(a: **codice** Functional
domain(a: **Partita**)
range(xsd: string))

- L'attributo 'fileAmbientazione' ha come dominio la classe Quadro e come range string e viene ereditato dalla classe QuadroDedicato.

DatatypeProperty(a: **fileAmbientazione** Functional
domain(a: **Quadro**)
range(xsd: string))

- L'attributo 'titolo' ha come dominio la classe QuadroDedicato e come range string, la classe eredita anche gli attributi della superclasse Quadro.

DatatypeProperty(a: **fileFilmato** Functional
domain(a: **QuadroDedicato**)
range(xsd: string))

- L'attributi 'fileImmagine' ha come dominio la classe Personggio e come range string.

DatatypeProperty(a: **fileImmagine** Functional
domain(a: **Personaggio**)
range(xsd: string))

RELAZIONI(ObjectProperty)

Le associazioni sono state tradotte in relazioni tra le classi (ObjectProperty) e per ogni relazione è stata definita una relazione inversa:

giocato ←→ inverse_of_giocato
presente ←→ inverse_of_presente
dedicatoA ←→ inverse_of_dedicatoA

ObjectProperty(a:dedicatoA Functional
inverseOf(a:inverse_of_dedicatoA)
domain(a:QuadroDedicato)
range(a:Personaggio))

ObjectProperty(a:inverse_of_dedicatoA InverseFunctional
inverseOf(a:dedicatoA)
domain(a:Personaggio)
range(a:QuadroDedicato))

- **giocato:**

L'associazione contiene viene tradotta con un ObjectProperty che ha come dominio la classe Partita e come range la classe Quadro.

ObjectProperty(a: **giocato**
inverseOf(a: **inverse_of_giocato**)
domain(a: **Partita**)
range(a: **Quadro**))

La sua relazione inversa è inverse_of_giocato ha come dominio la classe Quadro e come range la classe Partita.

ObjectProperty(a: **inverse_of_giocato**
inverseOf(a: **giocato**)
domain(a: **Quadro**)
range(a: **Partita**))

- **presente:**

L'associazione 'presente' viene tradotta con un ObjectProperty che ha come dominio la classe Quadro e come range la classe Personaggio.

ObjectProperty(a: **presente**
inverseOf(a: **inverse_of_presente**)
domain(a: **Quadro**)

range(a: **Personaggio**))

La sua relazione inversa è 'inverse_of_presente' ha come dominio la classe Personaggio e come range la classe Quadro.

ObjectProperty(a: **inverse_of_presente**
inverseOf(a: **presente**)
domain(a: **Personaggio**)
range(a: **Quadro**))

- **dedicatoA:**

L'associazione 'dedicatoA' viene tradotta con un ObjectProperty che ha come dominio la classe QuadroDedicato e come range la classe Personaggio.

ObjectProperty(a: **dedicatoA**
inverseOf(a: **inverse_of_dedicatoA**)
domain(a: **QuadroDedicato**)
range(a: **Personaggio**))

La sua relazione inversa è 'inverse_of_presente' ha come dominio la classe Personaggio e come range la classe QuadroDedicato.

ObjectProperty(a: **inverse_of_dedicatoA**
inverseOf(a: **dedicatoA**)
domain(a: **Personaggio**)
range(a: **QuadroDedicato**))

L'associazione 'dedicatoA' e derivata dall'associazione 'presente' pertanto viene definita come subproperty ed eredità tutte le sue caratteristiche(analogamente per la proprietà inversa):

SubPropertyOf(a: **dedicatoA** a: **presente**)
SubPropertyOf(a: **inverse_of_dedicatoA** a: **inverse_of_presente**)

DL- LiteA

Asserzioni di inclusione di concetti

Partita ⊑ ¬ Quadro
Quadro ⊑ ¬ Partita
Partita ⊑ ¬ Personaggio
Personaggio ⊑ ¬ Partita
Personaggio ⊑ ¬ Quadro
Quadro ⊑ ¬ Personaggio
QuadroDedicato ⊑ Quadro

Partita ⊑ δ(codice)
Personaggio ⊑ δ(fileImmagine)
Quadro ⊑ δ(fileAmbientazione)
QuadroDedicato ⊑ δ(fileFilmato)

∃ giocato ⊑ Partita
∃ (giocato)$^{-}$ ⊑ Quadro
∃ inverse_of_ giocato ⊑ Quadro
∃ (inverse_of_ giocato)$^{-}$ ⊑ Partita

∃ presente ⊑Quadro
∃ (presente)$^{-}$ ⊑Personaggio
∃ inverse_of_ presente ⊑ Personaggio
∃ (inverse_of_ presente)$^{-}$ ⊑ Quadro

∃ dedicatoA ⊑ QuadroDedicato
∃ (dedicatoA)$^{-}$ ⊑ Personaggio
∃ inverse_of_ dedicatoA ⊑ Personaggio
∃ (inverse_of_ dedicatoA)$^{-}$ ⊑ QuadroDedicato

Asserzioni di funzionalità di attributi di concetto

funct(codice)
funct(fileAmbientazione)
funct(fileImmagine)
funct(fileFilmato)

Asserzioni di funzionalità di ruolo

funct(dedicatoA)

Asserzioni di inclusione di domini di valori

ρ(codice) ⊑ xs:string

ρ(fileAmbientazione) ⊑ xs:string
ρ(fileImmagine) ⊑ xs:string
ρ(fileFilmato) ⊑ xs:string

Asserzioni di inclusione di ruoli

giocato ⊑ (inverse_of_ giocato)$^{-}$
inverse_of_ giocato ⊑ (giocato)$^{-}$

presente ⊑ (inverse_of_ presente)$^{-}$
inverse_of_ presente ⊑ (presente)$^{-}$

dedicatoA ⊑ (inverse_of_ dedicatoA)$^{-}$
inverse_of_ dedicatoA ⊑ (dedicatoA)$^{-}$

dedicatoA ⊑ presente
inverse_of_dedicatoA ⊑ inverse_of_presente

- **Alfabeto**

L'alfabeto che descrive lo schema UML è formato dai seguenti elementi:
da concetti generali (atomicC), attributi di concetti (atomicCA) , ruoli generali (atomicR).

<alphabet>

<atomicC> **Partita** </atomicC>
<atomicC> **Quadro** </atomicC>
<atomicC> **QuadroDedicato** </atomicC>
<atomicC> **Personaggio** </atomicC>

<atomicCA> **codice** </atomicCA>
<atomicCA> **fileAmbientazione** </atomicCA>
<atomicCA> **fileFilmato** </atomicCA>
<atomicCA> **fileImmagine** </atomicCA>

<atomicR> **giocato** </atomicR>
<atomicR> **inverse_of_giocato** </atomicR>
<atomicR> **presente** </atomicR>
<atomicR> **inverse_of_presente** </atomicR>
<atomicR> **dedicatoA** </atomicR>
<atomicR> **inverse_of_dedicatoA** </atomicR>

</alphabet>

Le Classi Partita, Quadro, QuadroDedicato, Personaggio sono concetti generali.
Le proprietà delle classi codice, fileAmbientazione, fileFilmato, fileImmagine sono attributi di concetto.
Le relazioni giocato, inverse_of_ giocato, presente, inverse_of_ presente, dedicatoA, inverse_of_ dedicatoA sono ruoli generali.

• **Tbox**

La Tbox è usata per rappresentare la conoscenza intensionale attraverso assezioni di inclusione.

-- Partita ⊑ ¬ Quadro

```
<inclusionAssertion>
        <basicC>
            <atomicC> Partita </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Quadro </atomicC>
                </basicC>
            </signedC>
        </generalC>
 </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Partita non è una Quadro (sign="**negative**" vuole indicare la negazione del concetto).

-- Quadro ⊑ ¬ Partita

```
<inclusionAssertion>
        <basicC>
            <atomicC> Quadro </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Partita </atomicC>
                </basicC>
            </signedC>
        </generalC>
 </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Quadro non è una Partita (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di inclusione di concetto (Partita ⊑ ¬ Quadro , Quadro ⊑ ¬ Partita) stanno ad indicare che i concetti Partita e Quadro sono disgiunti

-- Partita ⊑ ¬ Personaggio

```
<inclusionAssertion>
        <basicC>
            <atomicC> Partita </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Personaggio </atomicC>
                </basicC>
            </signedC>
        </generalC>
 </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Partita non è una Personaggio (sign="**negative**" vuole indicare la negazione del concetto).

-- Personaggio ⊑ ¬ Partita

```
<inclusionAssertion>
        <basicC>
            <atomicC> Personaggio </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Partita </atomicC>
                </basicC>
            </signedC>
        </generalC>
 </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Partita non è una Personaggio (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di inclusione di concetto (Partita ⊑ ¬ Personaggio **,** *Quadro ⊑ ¬ Personaggio) stanno ad indicare che i concetti Partita e Personaggio sono disgiunti*

-- Personaggio ⊑ ¬ Quadro

```
<inclusionAssertion>
        <basicC>
            <atomicC> Personaggio </atomicC>
        </basicC>
```

```
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Quadro </atomicC>
                </basicC>
            </signedC>
        </generalC>
  </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Partita non è una Quadro (sign="**negative**" vuole indicare la negazione del concetto).

-- Quadro ⊑ ¬ Personaggio

```
<inclusionAssertion>
        <basicC>
            <atomicC> Quadro </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Personaggio </atomicC>
                </basicC>
            </signedC>
        </generalC>
  </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Quadro non è un Personaggio (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di inclusione di concetto (Personaggio ⊑ ¬ Quadro , Quadro ⊑ ¬ Personaggio) stanno ad indicare che i concetti Personaggio e Quadro sono disgiunti

-- QuadroDedicato ⊑ Quadro

```
<inclusionAssertion>
        <basicC>
           <atomicC> QuadroDedicato </atomicC>
        </basicC>
        <generalC>
           <signedC sign="positive">
              <basicC>
                 <atomicC> Quadro </atomicC>
              </basicC>
           </signedC>
        </generalC>
</inclusionAssertion>
```

L’asserzione di inclusione di concetto vuole indicare che ogni concetto QuadroDedicato è una Quadro.

Questa asserzione di inclusione di concetto (QuadroDedicato ⊑ Quadro) sta ad indicare che la classe QuadroDedicato è una sottoclasse di Quadro.

-- funct(codice)

```
<funct>
        <atomicCA> codice </atomicCA>
</funct>
```

L’attributo di concetto, ‘codice’, è stato definito funzionale.

-- Partita ⊑ δ(codice)

```
<inclusionAssertion>
        <basicC>
              <atomicC> Partita </atomicC>
        </basicC>
        <generalC>
              <signedC sign="positive">
                   <basicC>
                         <CADomain>
                               <atomicCA> codice </atomicCA>
                         </CADomain>
                   </basicC>
              </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Fotografia ha associato un attributo di concetto ‘codice’.
Dato un attributo di concetto ‘nome’ chiamiamo dominio di ‘codice’ denotato con δ(codice) il set di oggetti che ‘codice’ collega a valori.

-- ρ(codice) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
          <ARange>
               <atomicCA> codice </atomicCA>
          </ARange>
    </basicV>
    <generalV>
          <predefinedV> xs:string </predefinedV>
    </generalV>
```

</inclusionAssertion>

Il range dell'attributo di concetto 'codice' è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di 'codice' chiamiamo range di 'codice' denotato con ρ(codice) il set di valori che 'codice' collega a oggetti.

-- funct(fileAmbientazione)

```
<funct>
        <atomicCA> fileAmbientazione </atomicCA>
</funct>
```

L'attributo di concetto, 'fileAmbientazione', è stato definito funzionale.

-- Quadro ⊑ δ(fileAmbientazione)

```
<inclusionAssertion>
        <basicC>
              <atomicC> Quadro </atomicC>
        </basicC>
        <generalC>
              <signedC sign="positive">
                  <basicC>
                        <CADomain>
                              <atomicCA> fileAmbientazione </atomicCA>
                        </CADomain>
                  </basicC>
              </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Quadro ha associato un attributo di concetto 'fileAmbientazione'.
Dato un attributo di concetto 'tipo' chiamiamo dominio di 'fileAmbientazione' denotato con δ(fileAmbientazione) il set di oggetti che 'fileAmbientazione' collega a valori.

-- ρ(fileAmbientazione) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
          <ARange>
              <atomicCA> fileAmbientazione </atomicCA>
          </ARange>
    </basicV>
    <generalV>
          <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell'attributo di concetto 'fileAmbientazione' è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di 'fileAmbientazione' chiamiamo range di 'fileAmbientazione' denotato con ρ(fileAmbientazione) il set di valori che 'fileAmbientazione' collega a oggetti.

-- funct(fileImmagine)

```
<funct>
        <atomicCA> fileImmagine </atomicCA>
</funct>
```

L'attributo di concetto, 'fileImmagine', è stato definito funzionale.

-- Personaggio ⊑ δ(fileImmagine)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Personaggio </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> fileImmagine </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Personaggio ha associato un attributo di concetto 'fileImmagine'.
Dato un attributo di concetto 'nome' chiamiamo dominio di 'fileImmagine' denotato con δ(fileImmagine) il set di oggetti che 'fileImmagine' collega a valori.

-- ρ(fileImmagine) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
            <ARange>
                <atomicCA>fileImmagine</atomicCA>
            </ARange>
    </basicV>
    <generalV>
            <predefinedV>xs:string</predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell'attributo di concetto 'fileImmagine' è un dominio di valori predefinito xs:string.

Dato un attributo di concetto di ‘fileImmagine’ chiamiamo range di ‘fileImmagine’ denotato con ρ(fileImmagine) il set di valori che ‘fileImmagine’ collega a oggetti.

-- funct(fileFilmato)

```
<funct>
        <atomicCA> fileFilmato </atomicCA>
</funct>
```

L’attributo di concetto, ‘fileFilmato’, è stato definito funzionale.

-- QuadroDedicato ⊑ δ(fileFilmato)

```
<inclusionAssertion>
        <basicC>
            <atomicC> QuadroDedicato </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> fileFilmato </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto QuadroDedicato ha associato un attributo di concetto ‘fileFilmato’.
Dato un attributo di concetto ‘nome’ chiamiamo dominio di ‘fileFilmato’ denotato con δ(fileFilmato) il set di oggetti che ‘fileFilmato’ collega a valori.

-- ρ(fileFilmato) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA>fileFilmato</atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV>xs:string</predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘fileFilmato’ è un dominio di valori predefinito xs:string.

Dato un attributo di concetto di ‘fileFilmato’ chiamiamo range di ‘fileFilmato’ denotato con ρ(fileFilmato) il set di valori che ‘fileFilmato’ collega a oggetti.

-- giocato ⊑ (inverse_of_ giocato)¯

```
<inclusionAssertion>
        <basicR dir="direct">
                <atomicR> giocato </atomicR>
        </basicR>
        <generalR>
                <signedR sign="positive">
                    <basicR dir="inverse">
                            <atomicR> inverse_of_giocato </atomicR>
                    </basicR>
                </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘giocato’ è un’istanza del ruolo inverso di ‘ inverse_of_ giocato’.

Questa asserzione di inclusione di ruolo (giocato ⊑ (inverse_giocato)¯) sta ad indicare che il ruolo ‘inverse_of_giocato’ è la relazione inversa della relazione ‘giocato’

-- inverse_of_ giocato ⊑ (giocato)¯

```
<inclusionAssertion>
        <basicR dir="direct">
                <atomicR> inverse_of_giocato </atomicR>
        </basicR>
        <generalR>
                <signedR sign="positive">
                        <basicR dir="inverse">
                                <atomicR> giocato </atomicR>
                        </basicR>
                </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ giocato’ è un’istanza del ruolo inverso di ‘giocato’.

Questa asserzione di inclusione di ruolo (inverse_of_*giocato ⊑ (giocato)¯) sta ad indicare che il ruolo ‘giocato’ è la relazione inversa della relazione ‘inverse_of_giocato*’.

-- ∃ giocato ⊑ Partita

```
<inclusionAssertion>
        <basicC>
                <exists>
                    <basicR dir="direct">
                            <atomicR> giocato </atomicR>
                    </basicR>
                </exists>
        </basicC>
        <generalC>
                <signedC sign="positive">
                    <basicC>
                            <atomicC> Partita </atomicC>
                    </basicC>
                </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘giocato’ è un’ istanza del concetto Partita.

-- ∃ (giocato)¯ ⊑ Quadro

```
<inclusionAssertion>
        <basicC>
                <exists>
                    <basicR dir="inverse">
                            <atomicR> giocato </atomicR>
                    </basicR>
                </exists>
        </basicC>
        <generalC>
                <signedC sign="positive">
                    <basicC>
                            <atomicC> Quadro </atomicC>
                    </basicC>
                </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso ‘giocato’ è un’ istanza del concetto Quadro.

-- ∃ inverse_of_ giocato ⊑ Quadro

```
<inclusionAssertion>
        <basicC>
                <exists>
                    <basicR dir="direct">
                            <atomicR> inverse_of_giocato </atomicR>
```

```
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">                    <basicC>
                    <atomicC> Quadro </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_giocato’ è un’ istanza del concetto Quadro.

-- ∃ (inverse_of_ giocato)¯ ⊑ Partita

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                        <atomicR> inverse_of_giocato </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Partita</atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘inverse_of_giocato’ è un’ istanza del concetto Partita.

-- presente ⊑ (inverse_of_ presente)¯

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> presente </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                        <atomicR> inverse_of_presente </atomicR>
                </basicR>
            </signedR>
        </generalR>
```

</inclusionAssertion>

Ogni istanza del ruolo ‘presente’ è un’istanza del ruolo inverso di ‘ inverse_of_ presente’.

Questo ad indicare che il ruolo ‘inverse_of_ presente’ è il ruolo inverso del ruolo ‘presente’.

-- inverse_of_ presente ⊑ (presente)¯

```
<inclusionAssertion>
        <basicR dir="direct">
              <atomicR> inverse_of_presente </atomicR>
        </basicR>
        <generalR>
              <signedR sign="positive">
                  <basicR dir="inverse">
                        <atomicR> presente </atomicR>
                  </basicR>
              </signedR>
        </generalR>
      </inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ presente’ è un’istanza del ruolo inverso di ‘presente’.

Questo ad indicare che il ruolo inverso di ‘presente’ è il ruolo inverso del ruolo ‘inverse_of_ presente’.

-- ∃ presente ⊑ Quadro

```
<inclusionAssertion>
        <basicC>
              <exists>
                  <basicR dir="direct">
                        <atomicR> presente </atomicR>
                  </basicR>
              </exists>
        </basicC>
        <generalC>
              <signedC sign="positive">
                   <basicC>
                        <atomicC> Quadro </atomicC>
                  </basicC>
              </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘presente’ è un’ istanza del concetto Quadro.

-- ∃ (presente)¯ ⊑ Personaggio

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> presente </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Personaggio </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso ‘presente’ è un’ istanza del concetto Personaggio.

-- ∃ inverse_of_ presente ⊑ Personaggio

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> inverse_of_presente </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Personaggio </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_presente’ è un’ istanza del concetto Personaggio.

-- ∃ (inverse_of_ presente)¯ ⊑ Quadro

```
<inclusionAssertion>
        <basicC>
```

```
            <exists>
                <basicR dir="inverse">
                        <atomicR> inverse_of_presente </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Quadro </atomicC>
                </basicC>
             </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘inverse_of_presente’ è un’ istanza del concetto Quadro.

-- funct(dedicatoA)

```
<funct>
      <basicR dir="direct">
            <atomicR> dedicatoA </atomicR>
      </basicR>
</funct>
```

Il ruolo, ‘dedicatoA’, è stato definito funzionale.

Questa asserzione di funzionalità di ruolo (funct dedicatoA) sta ad indicare la cardinalità (1,1) della relazione dedicatoA.

-- dedicatoA ⊑ (inverse_of_ dedicatoA)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
              <atomicR> dedicatoA </atomicR>
        </basicR>
        <generalR>
              <signedR sign="positive">
                  <basicR dir="inverse">
                        <atomicR> inverse_of_dedicatoA </atomicR>
                  </basicR>
              </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘dedicatoA’ è un’istanza del ruolo inverso di ‘ inverse_of_ dedicatoA’.

Questo ad indicare che il ruolo ‘inverse_of_ dedicatoA’ è il ruolo inverso del ruolo ‘dedicatoA’.

-- inverse_of_ dedicatoA ⊑ (dedicatoA)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
              <atomicR> inverse_of_dedicatoA </atomicR>
        </basicR>
        <generalR>
              <signedR sign="positive">
                   <basicR dir="inverse">
                         <atomicR> dedicatoA </atomicR>
                   </basicR>
              </signedR>
        </generalR>
      </inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ dedicatoA’ è un’istanza del ruolo inverso di ‘dedicatoA’.

Questo ad indicare che il ruolo ‘dedicatoA’ è il ruolo inverso del ruolo ‘inverse_of_dedicatoA’.

-- ∃ dedicatoA ⊑ QuadroDedicato

```
<inclusionAssertion>
        <basicC>
              <exists>
                   <basicR dir="direct">
                         <atomicR> dedicatoA </atomicR>
                   </basicR>
              </exists>
        </basicC>
        <generalC>
              <signedC sign="positive">
                    <basicC>
                         <atomicC> QuadroDedicato </atomicC>
                   </basicC>
              </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘dedicatoA’ è un’ istanza del concetto QuadroDedicato.

-- ∃ (dedicatoA)⁻ ⊑ Personaggio

```
<inclusionAssertion>
        <basicC>
```

```
            <exists>
                <basicR dir="inverse">
                    <atomicR> dedicatoA </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Personaggio </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso‘dedicatoA’ è un’ istanza del concetto Personaggio.

-- ∃ inverse_of_ dedicatoA ⊑ Personaggio

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> inverse_of_dedicatoA </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Personaggio </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_dedicatoA’ è un’ istanza del concetto Personaggio.

-- ∃ (inverse_of_ dedicatoA)⁻ ⊑ QuadroDedicato

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> inverse_of_dedicatoA </atomicR>
                </basicR>
            </exists>
```

```
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> QuadroDedicato </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘inverse_of_dedicatoA’ è un’ istanza del concetto QuadroDedicato.

-- dedicatoA ⊑ presente

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> dedicatoA </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="direct">
                    <atomicR> presente </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘dedicatoA’ è un’istanza del ruolo ‘presente’.

-- inverse_of_dedicatoA ⊑ inverse_of_presente

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> inverse_of_dedicatoA </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="direct">
                    <atomicR> inverse_of_presente </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_dedicatoA’ è un’istanza del ruolo ‘inverse_of_presente’.

Conclusione:

1) Attributo di ruolo in DL-liteA

In DL-liteA è possibile esprimere attributi di ruolo denotando la relazione binaria tra coppie di oggetti e valori (non esprimibile in DL-lite).
La relazione giocato presenta un attributo di relazione 'punti' di tipo intero.
Questo può essere espresso attraverso un attrobuto di ruolo 'punti' per il ruolo 'giocato'.
Le espressioni sono le seguenti:

Asserzione funzionale di attributo di ruolo:

(funct punti)

L'attributo di ruolo punti è funzionale

Asserzioni di inclusione di concetti:

Partita $\sqsubseteq \exists \delta$(punti)
Quadro $\sqsubseteq \exists \delta(\text{punti})^{-}$

Ogni Partita deve partecipare al ruolo 'giocato' (dominio di punti)avendo associato un attributo di ruolo 'punti'.
Dato un attributo di ruolo 'punti' chiamiamo dominio di 'punti' denotato con δ(punti) il set di oggetti che 'punti' collega a valori.
Ogni Quadro deve partecipare al ruolo 'inverse_of_giocato' (dominio inverso di punti) avendo associato un attributo di ruolo 'punti'.

Asserzioni di inclusione di ruolo:

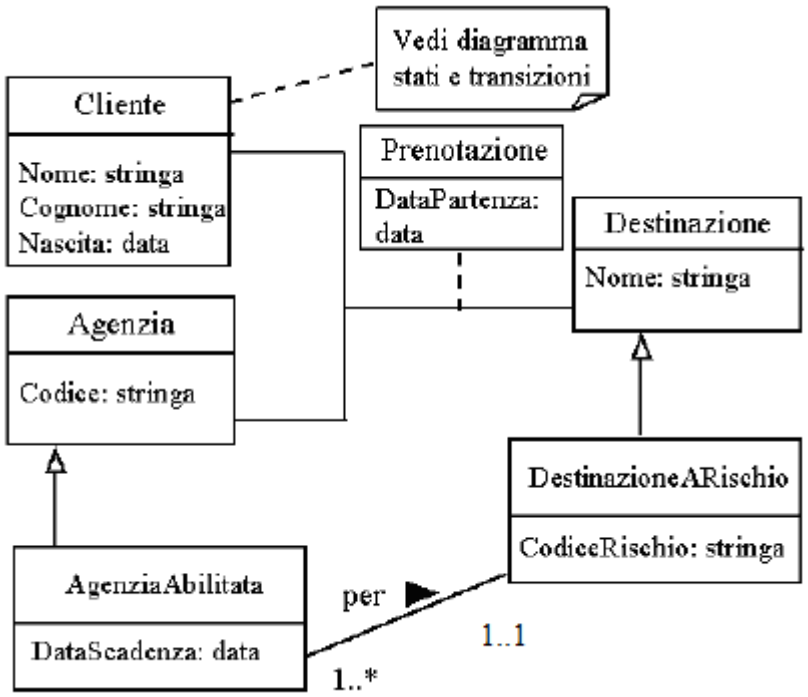
δ(punti) $\sqsubseteq$ giocato
$\delta(\text{punti})^{-} \sqsubseteq$ inverse_of_giocato

Il dominio dell'attributo di ruolo 'punti' è il ruolo 'giocato'
Il dominio inverso dell'attributo di ruolo 'punti' è il ruolo 'inverse_of_giocato'

Asserzioni di inclusione di dominio di valori:

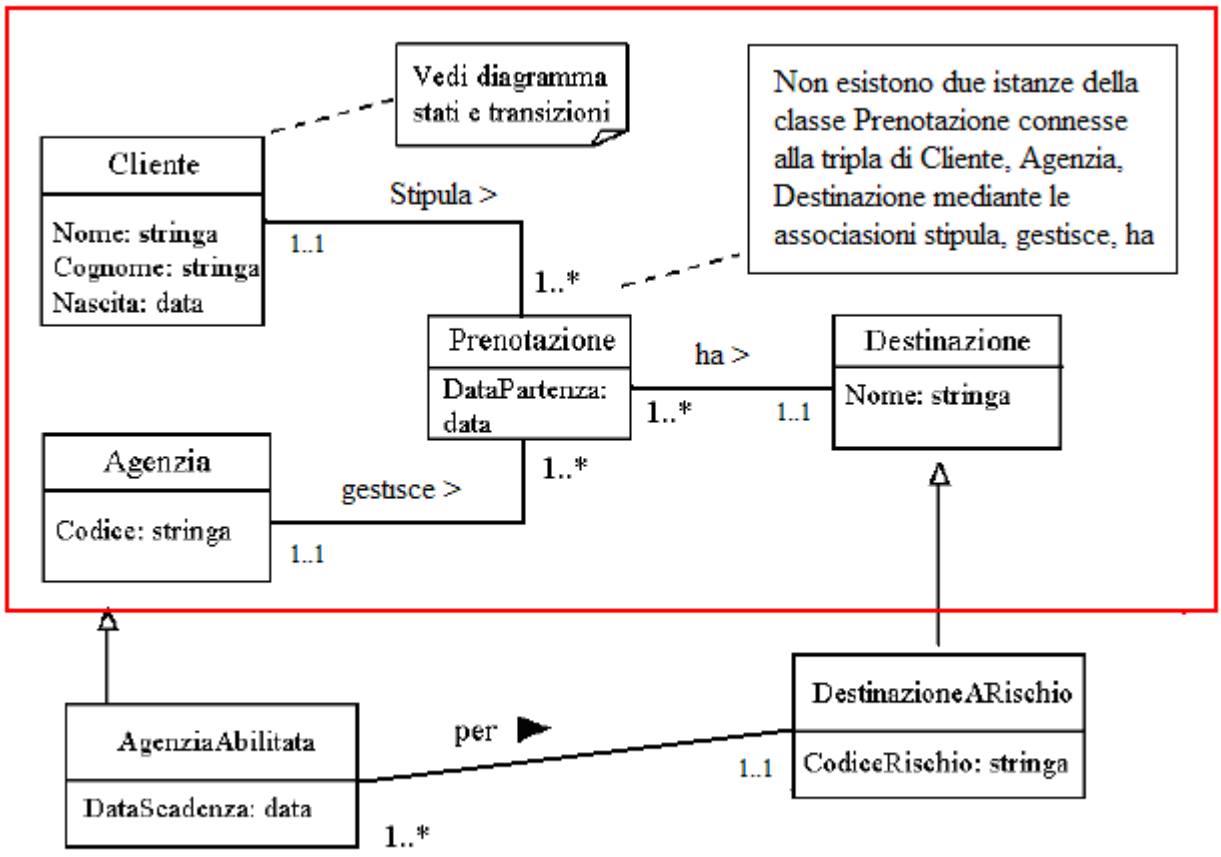
ρ(punti) $\sqsubseteq$ xs:int
Il range dell'attributo di ruolo 'punti' è xs:int

Compito d'esame 20 Aprile 2006



La relazione ternaria Prenotazione non può essere tradotta in Protégé;
per questo è stata trasformata la relazione 'Prenotazione' in classe e utilizzate le relazioni binarie 'stipula' tra la classi Cliente e Prenotazione, 'gestisce' tra la classe Agenzia e Prenotazione e 'ha' tra la classe Prenotazione e Destinazione.

Relazione ternaria ‘prenotazione’ → Relazione binaria ‘stipula’,’gestisce’,’ha’



OWL-DL

Classi: Cliente, Prenotazione, Agenzia, AgenziaAbilitata, Destinazione, DestinazioneRischio
Proprietà: nomeCliente, cognome, nascita, codice, dataScadenza, dataPartenza,
nomeDestinazione,CodiceRischio
Relazione: stipula, gestisce, ha, per

Non è possibile esprimere l’attributo di ruolo ‘dataPartenza’

CLASSI

- **Cliente**

L'oggetto "Cliente" è stato tradotto in una classe Protègè che ha come proprietà gli attributi della classe: nomeCliente, cognome, nascita tradotta come dataproperty con range rispettivamente string e date.

Class(a:**Cliente** complete
restriction(a:stipula someValuesFrom(a:Prenotazione)))
DisjointClasses(a:**Cliente** a: **Agenzia**)
DisjointClasses(a:**Cliente** a: **Prenotazione**)
DisjointClasses(a:**Cliente** a: **Destinazione**)

La classe Cliente è *complete:* è descritta utilizzando condizioni necessarie e sufficienti ed è disgiunta dalle classi Agenzia, Prenotazione, Destinazione.

Alla classe viene posta la restrizione di cardinalità per esprimere la cardinalità minima con cui partecipa alle relazione 'stipula' con la classe Prenotazione ad esprimere che il Cliente stipula almeno una Prenotazione.

- **Agenzia**

L'oggetto "Agenzia" è stato tradotto in una classe Protègè che ha come proprietà l'attributo 'codice' che viene tradotto come dataproperty con range string.
La classe è superclasse della classe AgenziaAbilitata.
La classe partecipa alla relazione 'gestisce' con la classe Prenotazione con cardinalità 1..* che viene espressa tramite la restrizione di cardinalità.

Class(a: **Agenzia** complete
restriction(a: gestisce someValuesFrom(a: **Prenotazione**)))
DisjointClasses(a: **Agenzia** a: **Cliente**)
DisjointClasses(a: **Agenzia** a: **Prenotazione**)
DisjointClasses(a: **Agenzia** a: **Destinazione**)

La classe Agenzia è definita *complete:* è descritta utilizzando condizioni necessarie e sufficienti ed è disgiunta dalle classi Cliente, Prenotazione, Destinazione.

- **AgenziaAbilitata**

L'oggetto "AgenziaAbilitata" è stato tradotto in una classe Protègè che ha come proprietà l'attributo dataScadenza che viene tradotto come dataproperty con range date.
La classe è sottoclasse della classe Agenzia(relazione ISA).
La classe partecipa alla relazione 'per' con la classe DestinazioneRischio con cardinalità 1..1(ogni AgenziaAbilitata è per una Destinazione a Rischio).

Class(a: **AgenziaAbilitata** complete
restriction(a: per someValuesFrom(a: **DestinazioneRischio**)))
Class(a: **AgenziaAbilitata** partial a: **Agenzia**)

La classe AgenziaAbilitata è *complete:* è descritta utilizzando ha tra le condizioni necessarie e sufficienti le restrizioni ad indicare il vincolo di cardianiltà minima con il quale partecipa alla relazione 'per' con la classe DestinazioneRischio.
La classe AgenziaAbilitata è definita *partial* perché ha almeno una condizione necessaria.

• **Prenotazione**
L'oggetto "Prenotazione" è stato tradotto in una classe Protègè che ha come proprietà gli attributi della classe: dataPartenza tradotta come dataproperty con range rispettivamente date.

Class(a: **Prenotazione** complete
intersectionOf(restriction(a:inverse_of_stipula someValuesFrom(a: **Cliente**))
restriction(a:ha someValuesFrom(a: **Destinazione**))
restriction(a:inverse_of_gestisce someValuesFrom(a: **Agenzia**))))
DisjointClasses(a:**Prenotazione** a: **Agenzia**)
DisjointClasses(a:**Prenotazione** a: **Prenotazione**)
DisjointClasses(a:**Prenotazione** a: **Destinazione**)

La classe Prenotazione è *complete:* è descritta utilizzando condizioni necessarie e sufficienti ed è disgiunta dalle classi Agenzia, Prenotazione, Destinazione. .
Alla classe vengono poste delle restrizioni di cardinalità per esprimere le cardinalità minime con cui partecipa alle relazioni 'inverse_of_stipula' con la classe Cliente, 'ha'con la classe Destinazione, 'inverse_of_gestisce' con la classe Agenzia.

• **Destinazione**
L'oggetto "Destinazione" è stato tradotto in una classe Protègè che ha come proprietà l'attributo 'nomeDestinazione' che viene tradotto come dataproperty con range string.
La classe è superclasse della classe DestinazioneRischio.
La classe partecipa alla relazione 'inverse_of_ha' con la classe Prenotazione con cardinalità 1..* che viene espressa tramite la restrizione di cardinalità.

Class(a: **Destinazione** complete
restriction(a: inverse_of_ha someValuesFrom(a: **Prenotazione**)))
DisjointClasses(a: **Destinazione** a: **Cliente**)
DisjointClasses(a: **Destinazione** a: **Prenotazione**)
DisjointClasses(a: **Destinazione** a: **Agenzia**)

La classe Destinazione è definita *complete:* è descritta utilizzando condizioni necessarie e sufficienti ed è disgiunta dalle classi Cliente, Prenotazione, Agenzia.

• **DestinazioneRischio**
L'oggetto "DestinazioneRischio" è stato tradotto in una classe Protègè che ha come proprietà l'attributo codiceRischio che viene tradotto come dataproperty con range string.
La classe è sottoclasse della classe Destinazione(relazione ISA).
La classe partecipa alla relazione 'inverse_of_per' con la classe agenziaAbilitata con cardinalità 1..*.

Class(a: **DestinazioneRischio** complete

restriction(a: inverse_of_per someValuesFrom(a: **AgenziaAbilitata**)))
Class(a: **DestinazioneRischio** partial a: **Destinazione**)

La classe DestinazioneRischio è *complete:* è descritta utilizzando ha tra le condizioni necessarie e sufficienti le restrizioni ad indicare il vincolo di cardianiltà minima con il quale partecipa alla relazione 'inverse_of_per' con la classe AgenziaAbilitata.
La classe DestinazioneRischio è definita *partial* perché ha almeno una condizione necessaria.

PROPRIETA' (DataProperty)

Gli attributi della classe sono stati tradotti come DataProperty e hanno come la proprietà Functional poiché sono associati ad un solo valore per ogni individuo.

- Gli attributi nomeCliente e cognome hanno come dominio la classe Cliente e come range string

DatatypeProperty(a: **nomeCliente** Functional
domain(a:**Cliente**)
range(xsd:string))

DatatypeProperty(a: **cognome** Functional
domain(a:**Cliente**)
range(xsd:string))

DatatypeProperty(a: **nascita** Functional
domain(a:**Cliente**)
range(xsd:date))

- L'attributo codice ha come dominio la classe Agenzia e come range string.

DatatypeProperty(a: **codice** Functional
domain(a:**Agenzia**)
range(xsd:string))

- L'attributo dataScadenza ha come dominio la classe AgenziaAbiltata e come range string.

DatatypeProperty(a: **dataScadenza** Functional
domain(a:**AgenziaAbilitata**)
range(xsd:string))

- L'attributo dataPartenza ha come dominio la classe Prenotazione e come range date.

DatatypeProperty(a:**dataPartenza** Functional
domain(a: Prenotazione)
range(xsd:date))

- L'attributo nomeDestinazione ha come dominio la classe Destinazione e come range string.

DatatypeProperty(a:**nomeDestinazione** Functional

domain(a: **Destinazione**)
range(xsd:string))

- L'attributo codiceRischio ha come dominio la classe DestinazioneRischio e come range string.

DatatypeProperty(a: **codiceRischio** Functional
domain(a: **DestinazioneRischio**)
range(xsd:string))

<u>RELAZIONI</u> (ObjectProperty)

Le associazioni sono state tradotte in relazioni tra le classi (ObjectProperty) e per ogni relazione è stata definita una relazione inversa:

stipula←→inverse_of_ stipula
gestisce←→inverse_of_gestisce
per←→inverse_of_per
ha←→inverse_of_ha

- **stipula**

L'associazione 'stipula' viene tradotta con un ObjectProperty che ha come dominio la classe Cliente come range la classe Prenotazione e come proprietà inverseFunctional in quanto la relazione inversa è Functional..

ObjectProperty(a:**stipula** InverseFunctional
inverseOf(a:**inverse_of_stipula**)
domain(a:**Cliente**)
range(a:**Prenotazione**))

La sua relazione inversa è 'inverse_of_stipula' ha come dominio la classe Prenotazione e come range la classe Cliente. E' stata definita Functional per indicare il vincolo di cardinalità 1..1, per la cardinalità massima.

ObjectProperty(a: **inverse_of_stipula** Functional
inverseOf(a: **stipula**)
domain(a: **Prenotazione**)
range(a: **Cliente**))

- **gestisce**

L'associazione 'gestisce' viene tradotta con un ObjectProperty che ha come dominio la classe Agenzia come range la classe Prenotazione e come proprietà inverseFunctional in quanto la relazione inversa è Functional..

ObjectProperty(a:**gestisce** InverseFunctional
inverseOf(a:**inverse_of_gestisce**)
domain(a:**Agenzia**)
range(a:**Prenotazione**))

La sua relazione inversa è ‘inverse_of_gestisce’ ha come dominio la classe Prenotazione e come range la classe Agenzia. E’ stata definita Functional per indicare il vincolo di cardinalità 1..1, per la cardinalità massima.

ObjectProperty(a: **inverse_of_gestisce** Functional
inverseOf(a: **gestisce**)
domain(a: **Prenotazione**)
range(a: **Agenzia**))

- **ha**

L’associazione ‘ha’ viene tradotta con un ObjectProperty che ha come dominio la classe Prenotazione come range la classe Destinazione. E’ stata definita Functional per indicare il vincolo di cardinalità 1..1, per la cardinalità massima.

ObjectProperty(a:**ha** InverseFunctional
inverseOf(a:**inverse_of_ha**)
domain(a:**Prenotazione**)
range(a:**Destinazione**))

La sua relazione inversa è ‘inverse_of_ha’ ha come dominio la classe Destinazione e come range la classe Prenotazione e come proprietà inverseFunctional in quanto la relazione inversa è Functional..

ObjectProperty(a: **inverse_of_ha** Functional
inverseOf(a: **ha**)
domain(a: **Destinazione**)
range(a: **Prenotazione**))

- **per**

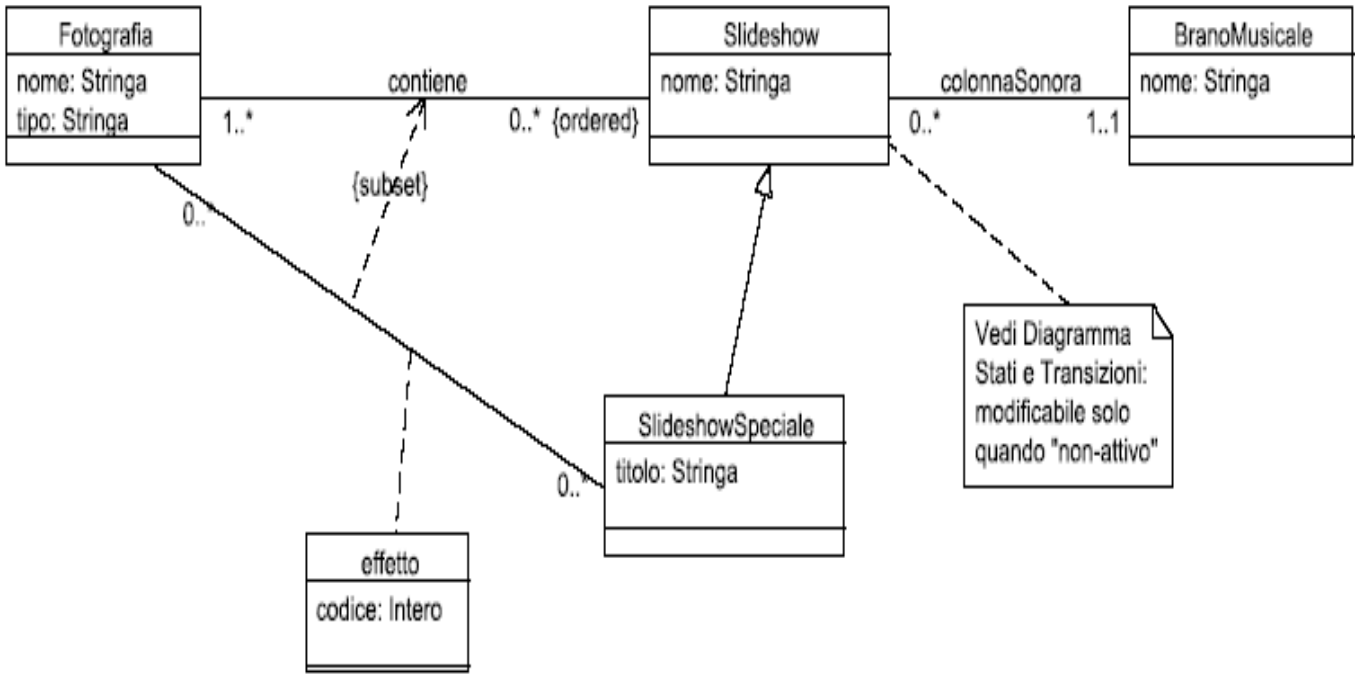
L’associazione ‘per’ viene tradotta con un ObjectProperty che per come dominio la classe AgenziaAbilitata come range la classe DestinazioneRischio. E’ stata definita Functional per indicare il vincolo di cardinalità 1..1, per la cardinalità massima.

ObjectProperty(a:**per** InverseFunctional
inverseOf(a:**inverse_of_per**)
domain(a:**AgenziaAbilitata**)
range(a:**DestinazioneRischio**))

La sua relazione inversa è ‘inverse_of_per’ ha come dominio la classe DestinazioneRischio e come range la classe AgenziaAbilitata e come proprietà inverseFunctional in quanto la relazione inversa è Functional..

ObjectProperty(a: **inverse_of_per** Functional
inverseOf(a: **per**)
domain(a: **DestinazioneRischio**)
range(a: **AgenziaAbilitata**))

Compito d’esame 7 Luglio 2006



OWL-DL

Classi: Fotografia, Slideshow, SlideshowSpeciale, Branomusicale
Proprietà: nome_foto, nome_slide, nome_brano, titolo_slide
Relazioni: contiene, effetto, colonnaSonora

Non è possibile esprimere l'attributo di rulo 'codice' , inoltre per poter esprimere 'ordered' bisgnerebbe introdurre una nuova relazione con un attributo (numero) che ne stabilisca l'ordinamento

CLASSI

- **Fotografia**

L'oggetto "Fotografia" è stato tradotto in una classe Protègè che ha come proprietà l'attributo: nome, tipo.
La classe partecipa alla relazione inverse_of_contiene con la classe Slideshow con cardinalità 0..*, e alla relazione inverse_of_effetto con la classe SlideshowSpeciale con cardinalità 0..* (che è una relazione derivata da inverse_of_contiene)

Class(a: **Fotografia** partial)
DisjointClasses(a: **Branomusicale** a: **Fotografia**)
DisjointClasses(a: **Slideshow** a: **Fotografia**)

La classe Fotografia è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalla classe Slideshow e Branomusicale.

- **Slideshow**

L'oggetto "Slideshow" è stato tradotto in una classe Protègè che ha come proprietà l'attributo: nome tradotta come dataproperty con range string.
La classe è superclasse della classe SlideshowSpeciale.
La classe partecipa alla relazione contiene con la classe Fotografia con cardinalità 1..*, e alla relazione colonnasonora con la classe Branomusicale con cardinalità 1..1 che vengono espresse mediante restrizioni per quanto riguarda la cardinalità minima.

Class(a: **Slideshow** complete
intersectionOf(
restriction(a: **colonnasonora** someValuesFrom(a: **Branomusicale**))
restriction(a: **contiene** someValuesFrom(a: **Fotografia**))))
DisjointClasses(a: **Slideshow** a: **Fotografia**)

La classe Slideshow è definita *complete* perché ha tra le condizioni necessarie e sufficienti le restrizioni ad indicare il vincolo di cardianiltà minima con il quale partecipa alla relazione colonnasonora con la classe Branomusicale e il vincolo di cardianiltà minima con il quale partecipa alla relazione contiene con la classe Fotografia.

- **SlideshowSpeciale**

L'oggetto "SlideshowSpeciale" è stato tradotto in una classe Protègè che ha come proprietà l'attributo: titolo tradotta come dataproperty con range string.
La classe è sottoclasse della classe Slideshow (relazione ISA).
La classe partecipa alla relazione effetto con la classe Fotografia con cardinalità 0..* (che è una relazione derivata della relazione contiene)

Class(a: **SlideshowSpeciale** partial a: **Slideshow**)

La classe SlideshowSpeciale è definita *partial* perché ha almeno una condizione necessaria

- **Branomusicale**

L'oggetto "Branomusicale" è stato tradotto in una classe Protègè che ha come proprietà l'attributo: nome tradotta come dataproperty con range string.
La classe partecipa alla relazione inverse_of_colonnasonora con la classe Slideshow con cardinalità 0..*,

Class(a: **Branomusicale** partial)
DisjointClasses(a: **Branomusicale** a: **Fotografia**)
DisjointClasses(a: **Branomusicale** a: **Slideshow**)

La classe Branomusicale è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta con le classi Fotografia e Slideshow.

PROPRIETA' (DataProperty)

Gli attributi della classe sono stati tradotti come DataProperty e hanno come la proprietà Functional poiché sono associati ad un solo valore per ogni individuo.

- L'attributo nome_brano ha come dominio la classe Branomusicale e come range string.

DatatypeProperty(a: **nome_brano** Functional
domain(a: **Branomusicale**)
range(xsd: string))

- L'attributo nome_slide ha come dominio la classe Slideshow e come range string e viene ereditato dalla classe SlideshowSpeciale

DatatypeProperty(a: **nome_slide** Functional
domain(a: **Slideshow**)
range(xsd: string))

- L'attributo titolo ha come dominio la classe SlideshowSpeciale e come range string, la classe eredita anche gli attributi della superclasse Slideshow.

DatatypeProperty(a: **titolo** Functional
domain(a: **SlideshowSpeciale**)
range(xsd: string))

- Gli attributi nome_foto e tipo_foto hanno come dominio la classe Fotografia e come range string.

DatatypeProperty(a: **nome_foto** Functional
domain(a: **Fotografia**)
range(xsd: string))

DatatypeProperty(a: **tipo_foto** Functional
domain(a: **Fotografia**)
range(xsd: string))

RELAZIONI(ObjectProperty)

Le associazioni sono state tradotte in relazioni tra le classi (ObjectProperty) e per ogni relazione è stata definita una relazione inversa:

contiene ←→ inverse_of_contiene
effetto ←→ inverse_of_revisore_effetto
colonnasonora ←→ inverse_of_colonnasonora

L'associazione effetto ha un attributo (codice) che non può essere tradotto come Datatype di ObjectProperty e questo non è possibile in Protègè.

- **contiene:**

L'associazione contiene viene tradotta con un ObjectProperty che ha come dominio la classe Slideshow e come range la classe Fotografia.

ObjectProperty(a: **contiene**
inverseOf(a: **inverse_of_contiene**)
domain(a: **Slideshow**)
range(a: **Fotografia**))

La sua relazione inversa è inverse_of_contiene_di ha come dominio la classe Fotografia e come range la classe Slideshow.

ObjectProperty(a: **inverse_of_contiene**
inverseOf(a: **contiene**)
domain(a: **Fotografia**)
range(a: **Slideshow**))

• **effetto**
L'associazione effetto viene tradotta con un ObjectProperty che ha come dominio la classe SlideshowSpeciale e come range Fotografia .

ObjectProperty(a: **effetto**
inverseOf(a: **inverse_of_effetto**)
domain(a**: SlideshowSpeciale**))
range((a: **Fotografia**))

La sua relazione inversa è inverse_of_effetto_di ha come dominio la classe Fotografia e come range la classe SlideshowSpeciale.

ObjectProperty(a: **inverse_of_effetto**
inverseOf(a: **effetto**)
domain(a: **Fotografia**)
range(a: **SlideshowSpeciale**))

L'associazione 'effetto' è derivata dall'associazione 'contiene' pertanto viene definita come subproperty ed eredità tutte le sue caratteristiche(analogamente per la proprietà inversa):

SubPropertyOf(a: **effetto** a: **contiene**)
SubPropertyOf(a: **inverse_of_effetto** a: **inverse_of_contiene**)

• **colonnasonora:**
L'associazione colonnasonora viene tradotta con un ObjectProperty che ha come dominio la classe Slideshow e come range la classe Branomusicale.

ObjectProperty(a: **colonnasonora** Functional
inverseOf(a: **inverse_of_colonnasonora**)
domain(a: **Slideshow**)
range(a:**Branomusicale**))

La sua relazione inversa è inverse_of_colonnasonora ha come dominio la classe Branomusicale e come range la classe Slideshow.

ObjectProperty(a:**inverse_of_colonnasonora** InverseFunctional
inverseOf(a:**colonnasonora**)
domain(a:**Branomusicale**)
range(a:**Slideshow**))

DL-LiteA

Asserzioni di inclusione di concetti

Fotografia ⊑ ¬ Slideshow
Fotografia ⊑ ¬ Branomusicale
SlideshowSpeciale ⊑ Slideshow
Slideshow ⊑ ¬ Branomusicale
Slideshow ⊑ ¬ Fotografia
Branomusicale ⊑ ¬ Fotografia
Branomusicale ⊑ ¬ Slideshow

Fotografia ⊑ δ(nome_foto)
Fotografia ⊑ δ(tipo_foto)
Slideshow ⊑ δ(nome_slide)
SlideshowSpeciale ⊑ δ(titolo)
Branomusicale ⊑ δ(nome_brano)

∃ contiene ⊑ Slideshow
∃ (contiene)$^{-}$ ⊑ Fotografia
∃ inverse_of_ contiene ⊑ Fotografia
∃ (inverse_of_ contiene)$^{-}$ ⊑ Slideshow
∃ colonnasonora ⊑ Slideshow
∃ (colonnasonora)$^{-}$ ⊑ Branomusicale
∃ inverse_of_ colonnasonora ⊑Branomusicale
∃ (inverse_of_ colonnasonora)$^{-}$ ⊑ Slideshow

∃ effetto ⊑ SlideshowSpeciale
∃ (effetto)$^{-}$ ⊑ Fotografia
∃ inverse_of_ effetto ⊑ Fotografia
∃ (inverse_of_ effetto)$^{-}$ ⊑ SlideshowSpeciale

Slideshow ⊑ ∃ contiene. Fotografia
Slideshow ⊑ ∃ colonnasonora. Branomusicale

Asserzioni di funzionalità di attributi di concetto

(funct nome_foto)
(funct tipo_foto)
(funct nome_slide)
(funct titolo)
(funct nome_brano)

Asserzioni di funzionalità di ruolo

(funct colonnasonora)

Asserzioni di inclusione di domini di valori

ρ(nome_foto) ⊑ xs:string
ρ(tipo_foto) ⊑ xs:string
ρ(nome_slide) ⊑ xs:string
ρ(titolo) ⊑ xs:string
ρ(nome_brano) ⊑ xs:string

Asserzioni di inclusione di ruoli

contiene ⊑ (inverse_of_ contiene)$^{-}$
inverse_of_ contiene ⊑ (contiene)$^{-}$
colonnasonora ⊑ (inverse_of_ colonnasonora)$^{-}$
inverse_of_ colonnasonora ⊑ (colonnasonora)$^{-}$
effetto ⊑ (inverse_of_ effetto)$^{-}$
inverse_of_effetto ⊑ (effetto)$^{-}$
effetto ⊑ contiene
inverse_of_effetto ⊑ inverse_of_contiene

- **Alfabeto**

L'alfabeto che descrive lo schema UML è formato dai seguenti elementi:
da concetti generali (atomicC), attributi di concetti (atomicCA) , ruoli generali (atomicR).

<alphabet>

<atomicC> **Fotografia** </atomicC>
<atomicC> **Slideshow** </atomicC>
<atomicC> **SlideshowSpeciale** </atomicC>
<atomicC> **Branomusicale** </atomicC>

<atomicCA> **nome_foto** </atomicCA>
<atomicCA> **tipo_foto** </atomicCA>
<atomicCA> **nome_slide** </atomicCA>
<atomicCA> **titolo** </atomicCA>
<atomicCA> **nome_brano** </atomicCA>

<atomicR> **contiene** </atomicR>
<atomicR> **inverse_of_contiene** </atomicR>
<atomicR> **effetto** </atomicR>
<atomicR> **inverse_of_effetto** </atomicR>
<atomicR> **colonnasonora** </atomicR>
<atomicR> **inverse_of_colonnasonora** </atomicR>

</alphabet>

Le Classi Fotografia, Slideshow, SlideshowSpeciale, Branomusicale sono concetti generali.
Le proprietà delle classi nome_foto, tipo_foto, nome_slide, titolo, nome_brano sono attributi di concetto.
Le relazioni contiene, inverse_of_ contiene, effetto, inverse_of_ effetto, colonnasonora, inverse_of_ colonnasonora sono ruoli generali.

• **Tbox**

La Tbox è usata per rappresentare la conoscenza intensionale attraverso assezioni di inclusione.

-- Fotografia ⊑ ¬ Slideshow

```
<inclusionAssertion>
        <basicC>
            <atomicC> Fotografia </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Slideshow </atomicC>
                </basicC>
            </signedC>
        </generalC>
 </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Fotografia non è una Slideshow (sign="**negative**" vuole indicare la negazione del concetto).

-- Slideshow ⊑ ¬ Fotografia

```
<inclusionAssertion>
        <basicC>
            <atomicC> Slideshow </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Fotografia </atomicC>
                </basicC>
            </signedC>
        </generalC>
 </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Slideshow non è una Fotografia (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di inclusione di concetto (Fotografia ⊑¬ Slideshow , Slideshow ⊑ ¬ Fotografia) stanno ad indicare che i concetti Fotografia e Slideshow sono disgiunti

-- Fotografia ⊑ ¬ Branomusicale

```
<inclusionAssertion>
        <basicC>
            <atomicC> Fotografia </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Branomusicale </atomicC>
                </basicC>
            </signedC>
        </generalC>
  </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Fotografia non è una Branomusicale (sign="**negative**" vuole indicare la negazione del concetto).

-- Branomusicale ⊑ ¬ Fotografia

```
<inclusionAssertion>
        <basicC>
            <atomicC> Branomusicale </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Fotografia </atomicC>
                </basicC>
            </signedC>
        </generalC>
  </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Fotografia non è una Branomusicale (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di inclusione di concetto (Fotografia ⊑ ¬ Branomusicale , Slideshow ⊑ ¬ Branomusicale) stanno ad indicare che i concetti Articolo e Persona sono disgiunti

-- Branomusicale ⊑ ¬ Slideshow

```
<inclusionAssertion>
        <basicC>
            <atomicC> Branomusicale </atomicC>
        </basicC>
        <generalC>
```

```
            <signedC sign="negative">
                <basicC>
                    <atomicC> Slideshow </atomicC>
                </basicC>
            </signedC>
        </generalC>
  </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Fotografia non è una Slideshow (sign="**negative**" vuole indicare la negazione del concetto).

-- Slideshow ⊑ ¬ Branomusicale

```
<inclusionAssertion>
        <basicC>
            <atomicC> Slideshow </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Branomusicale </atomicC>
                </basicC>
            </signedC>
        </generalC>
  </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Slideshow non è un Branomusicale (sign="**negative**" vuole indicare la negazione del concetto).

Queste asserzioni di inclusione di concetto (Branomusicale ⊑ ¬ Slideshow , Slideshow ⊑ ¬ Branomusicale) stanno ad indicare che i concetti Branomusicale e Slideshow sono disgiunti

-- SlideshowSpeciale ⊑ Slideshow

```
<inclusionAssertion>
        <basicC>
            <atomicC> SlideshowSpeciale </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Slideshow </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni concetto SlideshowSpeciale è una Slideshow.

Questa asserzione di inclusione di concetto(SlideshowSpeciale ⊑ Slideshow) sta ad indicare che la classe SlideshowSpeciale è una sottoclasse di Slideshow.

-- funct(nome_foto)

```
<funct>
        <atomicCA> nome_foto </atomicCA>
</funct>
```

L'attributo di concetto, 'nome_foto', è stato definito funzionale.

-- Fotografia ⊑ δ(nome_foto)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Fotografia </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> nome_foto </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Fotografia ha associato un attributo di concetto 'nome_foto'.
Dato un attributo di concetto 'nome' chiamiamo dominio di 'nome_foto' denotato con δ(nome_foto) il set di oggetti che 'nome_foto' collega a valori.

-- ρ(nome_foto) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA>nome_foto</atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV>xs:string</predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell'attributo di concetto 'nome_foto' è un dominio di valori predefinito xs:string.

Dato un attributo di concetto di ‘nome_foto’ chiamiamo range di ‘nome_foto’ denotato con ρ(nome_foto) il set di valori che ‘nome_foto’ collega a oggetti.

-- funct(tipo_foto)

```
<funct>
        <atomicCA> tipo_foto </atomicCA>
</funct>
```

L’attributo di concetto, ‘tipo_foto’, è stato definito funzionale.

-- Fotografia ⊑ δ(tipo_foto)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Fotografia </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> tipo_foto </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Fotografia ha associato un attributo di concetto ‘tipo_foto’.
Dato un attributo di concetto ‘tipo’ chiamiamo dominio di ‘tipo_foto’ denotato con δ(tipo_foto) il set di oggetti che ‘tipo_foto’ collega a valori.

-- ρ(tipo_foto) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> tipo_foto </atomicCA>
        </ARange>
    </basicV>
    <generalV>
            <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘tipo_foto’ è un dominio di valori predefinito xs:string.

Dato un attributo di concetto di ‘tipo_foto’ chiamiamo range di ‘tipo_foto’ denotato con ρ(tipo_foto) il set di valori che ‘tipo_foto’ collega a oggetti.

-- funct(nome_slide)

```
<funct>
        <atomicCA> nome_slide </atomicCA>
</funct>
```

L’attributo di concetto, ‘nome_slide’, è stato definito funzionale.

-- Slideshow ⊑ δ(nome_slide)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Slideshow </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> nome_slide </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Slideshow ha associato un attributo di concetto ‘nome_slide’.
Dato un attributo di concetto ‘nome’ chiamiamo dominio di ‘nome_slide’ denotato con δ(nome_slide) il set di oggetti che ‘nome_slide’ collega a valori.

-- ρ(nome_slide) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA>nome_slide</atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV>xs:string</predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘nome_slide’ è un dominio di valori predefinito xs:string.

Dato un attributo di concetto di ‘nome_slide’ chiamiamo range di ‘nome_slide’ denotato con ρ(nome_slide) il set di valori che ‘nome_slide’ collega a oggetti.

-- funct(titolo)

```
<funct>
        <atomicCA> titolo </atomicCA>
</funct>
```

L’attributo di concetto, ‘titolo’, è stato definito funzionale.

-- SlideshowSpeciale ⊑ δ(titolo)

```
<inclusionAssertion>
        <basicC>
            <atomicC> SlideshowSpeciale </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> titolo </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto SlideshowSpeciale ha associato un attributo di concetto ‘titolo’.
Dato un attributo di concetto ‘nome’ chiamiamo dominio di ‘titolo’ denotato con δ(titolo) il set di oggetti che ‘titolo’ collega a valori.

-- ρ(titolo) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> titolo </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘titolo’ è un dominio di valori predefinito xs:string.

Dato un attributo di concetto di ‘titolo’ chiamiamo range di ‘titolo’ denotato con ρ(titolo) il set di valori che ‘titolo’ collega a oggetti.

-- funct(nome_brano)

```
<funct>
        <atomicCA> nome_brano </atomicCA>
</funct>
```

L’attributo di concetto, ‘nome_brano’, è stato definito funzionale.

-- Branomusicale ⊑ δ(nome_brano)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Branomusicale </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> nome_brano </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Branomusicale ha associato un attributo di concetto ‘nome_brano’.
Dato un attributo di concetto ‘nome_brano’ chiamiamo dominio di ‘nome_brano’ denotato con δ(nome_brano) il set di oggetti che ‘nome_brano’ collega a valori.

-- ρ(nome_brano)⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA>nome_brano</atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV>xs:string</predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘nome_brano’ è un dominio di valori predefinito xs:string.

Dato un attributo di concetto di ‘nome_brano’ chiamiamo range di ‘nome_brano’ denotato con ρ(nome_brano) il set di valori che ‘nome_brano’ collega a oggetti.

-- contiene ⊑ (inverse_of_ contiene)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
                <atomicR> contiene </atomicR>
        </basicR>
        <generalR>
                <signedR sign="positive">
                    <basicR dir="inverse">
                            <atomicR> inverse_of_contiene </atomicR>
                    </basicR>
                </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘contiene’ è un’istanza del ruolo inverso di ‘ inverse_of_ contiene’.

Questa asserzione di inclusione di ruolo (contiene ⊑ (inverse_contiene)⁻) sta ad indicare che il ruolo ‘inverse_of_contiene’ è la relazione inversa della relazione ‘contiene’.

-- inverse_of_ contiene ⊑ (contiene)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
                <atomicR> inverse_of_contiene </atomicR>
        </basicR>
        <generalR>
                <signedR sign="positive">
                        <basicR dir="inverse">
                                <atomicR> contiene </atomicR>
                        </basicR>
                </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ contiene’ è un’istanza del ruolo inverso di ‘contiene’.

Questa asserzione di inclusione di ruolo (inverse_of_*contiene ⊑ (contiene)⁻) sta ad indicare che il ruolo ‘contiene’ è la relazione inversa della relazione ‘inverse_of_contiene*’.

-- ∃ contiene ⊑ Slideshow

```
<inclusionAssertion>
        <basicC>
                <exists>
                    <basicR dir="direct">
                            <atomicR> contiene </atomicR>
                    </basicR>
                </exists>
        </basicC>
        <generalC>
                <signedC sign="positive">
                    <basicC>
                            <atomicC> Slideshow </atomicC>
                    </basicC>
                </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘contiene’ è un’ istanza del concetto Slideshow.

-- ∃ (contiene)¯ ⊑ Fotografia

```
<inclusionAssertion>
        <basicC>
                <exists>
                    <basicR dir="inverse">
                            <atomicR> contiene </atomicR>
                    </basicR>
                </exists>
        </basicC>
        <generalC>
                <signedC sign="positive">
                    <basicC>
                            <atomicC> Fotografia </atomicC>
                    </basicC>
                </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso ‘contiene’ è un’ istanza del concetto Fotografia.

-- ∃ inverse_of_ contiene ⊑ Fotografia

```
<inclusionAssertion>
        <basicC>
                <exists>
                    <basicR dir="direct">
                            <atomicR> inverse_of_contiene </atomicR>
```

```
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Fotografia </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_contiene’ è un’ istanza del concetto Fotografia.

-- ∃ (inverse_of_ contiene)¯ ⊑ Slideshow

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> inverse_of_contiene </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Slideshow </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘inverse_of_contiene’ è un’ istanza del concetto Slidehow.

-- Slideshow ⊑ ∃ contiene. Fotografia

```
<inclusionAssertion>
        <basicC>
            <atomicC> Slideshow </atomicC>
        </basicC>
        <generalC>
            <qualifiedExists>
                <basicR dir="direct">
                    <atomicR> contiene </atomicR>
                </basicR>
```

```
                    <generalC>
                        <signedC sign="positive">
                            <basicC>
                                <atomicC>Fotografia</atomicC>
                            </basicC>
                        </signedC>
                    </generalC>
                </qualifiedExists>
        </generalC>
</inclusionAssertion>
```

Ogni Slideshow contiene almeno un Fotografia

Questa asserzione (Slideshow ⊑ ∃ contiene. Fotografia) sta ad indicare che la classe Slideshow partecipa alla relazione contiene con la classe Fotografia con cardinalità minima 1 (una Slideshow contiene almeno ad una Fotografia).

-- funct(colonnasonora)

```
<funct>
        <basicR dir="direct">
           <atomicR> colonnasonora </atomicR>
        </basicR>
</funct>
```

Il ruolo, 'colonnasonora', è stato definito funzionale.

Questa asserzione di funzionalità di ruolo (funct colonnasonora) sta ad indicare la cardinalità (1,1) della relazione ' colonnasonora'

-- colonnasonora ⊑ (inverse_of_ colonnasonora)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
                <atomicR> colonnasonora </atomicR>
        </basicR>
        <generalR>
                <signedR sign="positive">
                    <basicR dir="inverse">
                            <atomicR> inverse_of_colonnasonora </atomicR>
                    </basicR>
                </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo 'colonnasonora' è un'istanza del ruolo inverso di ' inverse_of_ colonnasonora'.

Questa asserzione di inclusione di ruolo (colonnasonora ⊑ ∃ (inverse_of_ colonnasonora)⁻) sta ad indicare che il ruolo 'inverse_of_ colonnasonora' è la relazione inversa della relazione 'colonnasonora'.

-- inverse_of_ colonnasonora ⊑ (colonnasonora)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
              <atomicR> inverse_of_colonnasonora </atomicR>
        </basicR>
        <generalR>
              <signedR sign="positive">
                   <basicR dir="inverse">
                         <atomicR> colonnasonora </atomicR>
                   </basicR>
              </signedR>
        </generalR>
      </inclusionAssertion>
```

Ogni istanza del ruolo 'inverse_of_ colonnasonora' è un'istanza del ruolo inverso di 'colonnasonora'.

Questa asserzione di inclusione di ruolo (inverse_of_ colonnasonora ⊑ (colonnasonora)⁻) sta ad indicare che il ruolo 'colonnasonora' è la relazione inversa della relazione 'colonnasonora'.

-- ∃ colonnasonora ⊑ Slideshow

```
<inclusionAssertion>
        <basicC>
              <exists>
                   <basicR dir="direct">
                         <atomicR> colonnasonora </atomicR>
                   </basicR>
              </exists>
        </basicC>
        <generalC>
              <signedC sign="positive">
                    <basicC>
                         <atomicC> Slideshow </atomicC>
                   </basicC>
              </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo 'colonnasonora' è un' istanza del concetto Slideshow.

-- ∃ (colonnasonora)¯ ⊑ Branomusicale

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> colonnasonora </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Branomusicale </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_colonnasonora’ è un’ istanza del concetto Branomusicale

-- ∃ inverse_of_ colonnasonora ⊑ Branomusicale

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> inverse_of_colonnasonora </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Branomusicale </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_colonnasonora’ è un’ istanza del concetto Branomusicale.

-- ∃ (inverse_of_ colonnasonora)¯ ⊑ Slideshow

```
<inclusionAssertion>
        <basicC>
```

```
            <exists>
                <basicR dir="inverse">
                        <atomicR> inverse_of_colonnasonora </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Slideshow </atomicC>
                </basicC>
             </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di 'inverse_of_colonnasonora' è un' istanza del concetto Slideshow.

-- Slideshow ⊑ ∃ colonnasonora. Branomusicale

```
<inclusionAssertion>
        <basicC>
             <atomicC> Slideshow </atomicC>
        </basicC>
        <generalC>
             <qualifiedExists>
                  <basicR dir="direct">
                        <atomicR> colonnasonora</atomicR>
                  </basicR>
                  <generalC>
                        <signedC sign="positive">
                            <basicC>
                                  <atomicC>Branomusicale</atomicC>
                            </basicC>
                        </signedC>
                  </generalC>
             </qualifiedExists>
        </generalC>
</inclusionAssertion>
```

Ogni SlideShow possiede come colonnasonora almeno un Branomusicale.

Questa asserzione (Slideshow ⊑ ∃ colonnasonora. Branomusicale) sta ad indicare che la classe Slideshow partecipa alla relazione colonnasonora con la classe Branomusicale con cardinalità minima 1 (una Slideshow ha per colonnasonora da almeno ad un Branomusicale).

-- effetto ⊑ (inverse_of_ effetto)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
                <atomicR> effetto </atomicR>
        </basicR>
        <generalR>
                <signedR sign="positive">
                    <basicR dir="inverse">
                            <atomicR> inverse_of_effetto </atomicR>
                    </basicR>
                </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘effetto’ è un’istanza del ruolo inverso di ‘ inverse_of_ effetto’.

Questa asserzione di inclusione di ruolo (effetto ⊑ (inverse_of_effetto)⁻)sta ad indicare che il ruolo ‘inverse_of_effetto’ è la relazione inversa della relazione ‘effetto’.

-- inverse_of_ effetto ⊑ (effetto)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
                <atomicR> inverse_of_effetto </atomicR>
        </basicR>
        <generalR>
                <signedR sign="positive">
                    <basicR dir="inverse">
                            <atomicR> effetto </atomicR>
                    </basicR>
                </signedR>
        </generalR>
      </inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ effetto’ è un’istanza del ruolo inverso di ‘effetto’.

Questa asserzione di inclusione di ruolo (inverse_of_effetto ⊑ (effetto)⁻) sta ad indicare che il ruolo ‘effetto’ è la relazione inversa della relazione ‘inverse_of_effetto’.

-- ∃ effetto ⊑ SlideshowSpeciale

```
<inclusionAssertion>
        <basicC>
              <exists>
                 <basicR dir="direct">
```

```
                    <atomicR> effetto </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> SlideshowSpeciale </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘effetto’ è un’ istanza del concetto SlideshowSpeciale.

-- ∃ (effetto)¯ ⊑ Fotografia

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> effetto </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Fotografia </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del inverso ruolo ‘effetto’ è un’ istanza del concetto Fotografia.

-- ∃ inverse_of_ effetto ⊑ Fotografia

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> inverse_of_effetto </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
```

```
            <signedC sign="positive">
                <basicC>
                    <atomicC> Fotografia </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del inverso ruolo ‘inverse_of_effetto’ è un’ istanza del concetto Fotografia.

-- ∃ (inverse_of_ effetto)⁻ ⊑ SlideshowSpeciale

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> inverse_of_effetto </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> SlideshowSpeciale </atomicC>
                </basicC>
             </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del inverso ruolo inverso di ‘inverse_of_effetto’ è un’ istanza del concetto SlideshowSpeciale.

-- effetto ⊑ contiene

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> effetto </atomicR>
        </basicR>
        <generalR>
             <signedR sign="positive">
                <basicR dir="direct">
                    <atomicR> contiene </atomicR>
                </basicR>
             </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘effetto’ è un’istanza del ruolo ‘contiene’.

-- inverse_of_effetto ⊑ inverse_of_contiene

<inclusionAssertion>
<basicR dir="**direct**">
<atomicR> **inverse_of_effetto** </atomicR>
</basicR>
<generalR>
<signedR sign="**positive**">
<basicR dir="**direct**">
<atomicR> **inverse_of_contiene** </atomicR>
</basicR>
</signedR>
</generalR>
</inclusionAssertion>

Ogni istanza del ruolo ‘inverse_of_effetto’ è un’istanza del ruolo ‘inverse_of_contiene’.

Conclusioni:

1) *<!-- IntersectionOf not totally expressible in DL-liteA -->*

La classe Slideshow è espressa in DL come l’intersezione di:
la restrizione (che ha lo scopo di indicare la cardinalita minima della relazione ‘contiene’ con la classe Fotografia)
la restrizione (che ha lo scopo di indicare la cardinalita minima della relazione ‘colonnasonora’ con la classe Branomusicale)

Class(a: Slideshow complete
intersectionOf(restriction(a: contiene someValuesFrom(a: Fotografia))
restriction(a: colonnasonora someValuesFrom(a: Branomusicale))))

In DL-liteA il concetto di intersezione non è totalmente esprimibile.

2) *<!-- NOT IN DL-LiteA!There is a conflict :the atomic Role 'contiene' is qualified exists then it cannot be functional*
-->

La base di conoscenza in DL-lite-A K=<T,A>, è ottenuta attraverso delle restrizioni: ovvero T (TBox) è una DL-lite$_{FR}$ Tbox che soddisfa determinate condizioni. In particolare, in questo caso:
-- Per ogni ruolo atomico e inverso di un ruolo atomico Q compare in un concetto della forma ∃Q.C, le asserzioni (funct Q) e (funct Q^{-}) non sono in T.

Poiché il ruolo ‘contiene’ compare nella seguente asserzione:
Slideshow $\sqsubseteq \exists$ contiene. Fotografia e quindi è quantificato esistenzialmente,
l’asserzione (funct contiene) non è in T, e quindi il ruolo non può essere funzionale.

3) *Attributo di ruolo*

In DL-lite$_A$ è possibile esprimere attributi di ruolo denotando la relazione binaria tra coppie di oggetti e valori (non esprimibile in DL-lite).
La relazione ‘effetto’ presenta un attributo di relazione ‘codice’ di tipo intero.
Questo può essere espresso attraverso un attributo di ruolo ‘codice’ per il ruolo ‘effetto’.
Le espressioni sono le seguenti:

Asserzione funzionale di attributo di ruolo:

(funct codice)

L’attributo di ruolo ‘codice’ è funzionale

Asserzioni di inclusione di concetti:

SlideshowSpeciale $\sqsubseteq \exists \delta$(codice)
Fotografia $\sqsubseteq \exists \delta(\text{codice})^{-}$

Ogni SlideshowSpeciale deve partecipare al ruolo effetto (dominio di codice) avendo associato un attributo di ruolo ‘codice’.
Dato un attributo di ruolo ‘codice’ chiamiamo dominio di ‘codice’ denotato con δ(codice) il set di oggetti che ‘codice’ collega a valori.
Ogni Fotografia deve partecipare al ruolo ‘inverse_of_ effetto’ (dominio inverso di codice) avendo associato un attributo di ruolo ‘codice’.

Asserzioni di inclusione di ruolo:

δ(codice) $\sqsubseteq$ effetto
$\delta(\text{codice})^{-} \sqsubseteq$ inverse_of_effetto

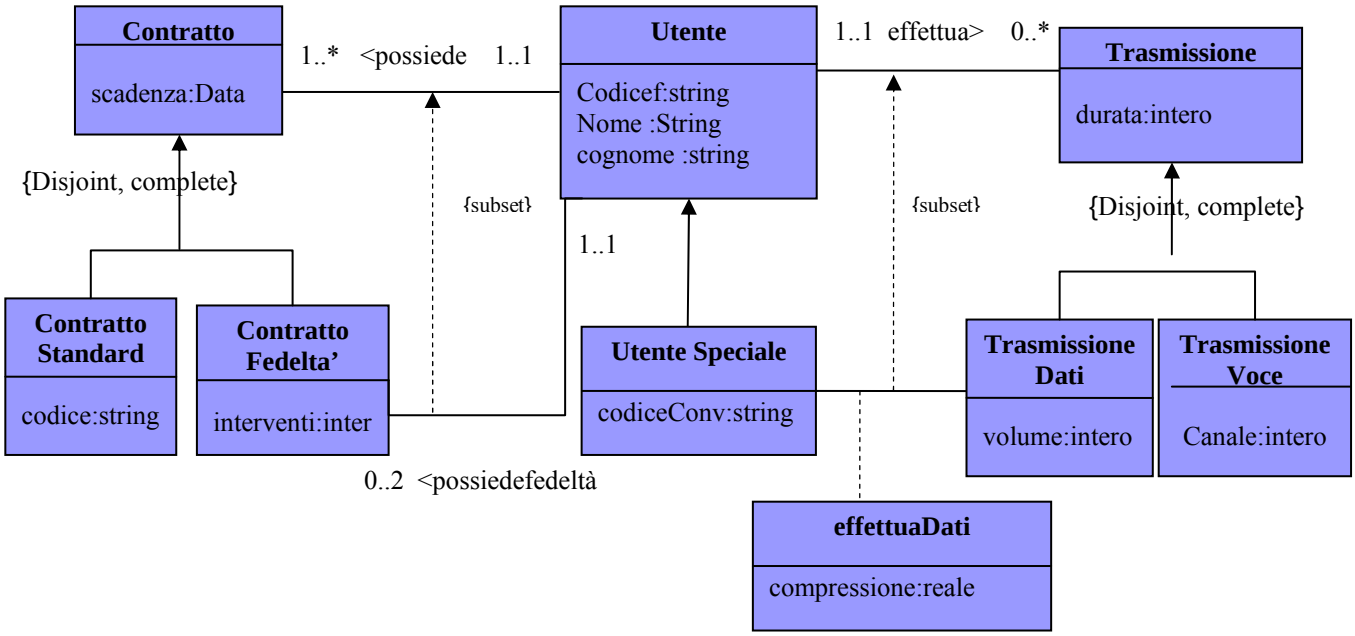
Il dominio dell’attributo di ruolo ‘codice’ è il ruolo ‘effetto’
Il dominio inverso dell’attributo di ruolo ‘codice’ è il ruolo ‘inverse_of_effetto’

Asserzioni di inclusione di dominio di valori:

ρ(codice) $\sqsubseteq$ xs:int

Il range dell’attributo di ruolo ‘codice’ è xs:int

Compito d’esame 13 Settembre 2006



OWL-DL

Classe: Contratto, ContrattoStandard, ContrattoFedeltà, Utente, UtenteSpeciale, Trasmissione, TrasmissioneDati, TrasmissioneVoce
Proprietà: scadenza, codice, interventi, codicef, nome, utente, ccodiceConv, durata, volume,canale
Relazioni: possiede, possiedefedeltà, effetta, effettuaDati

Non è possibile definire l'attributo 'conpressione' della relazione 'effettuaDati'

CLASSI

- **Contratto**

L'oggetto "Contratto" è stato tradotto in una classe Protègè che ha come proprietà l'attributo della classe: scadenza tradotta come dataproperty con range date.
La classe Contratto è superclasse nella generalizzazione che ha come sottoclassi ContrattoStandard, ContrattoSpeciale (che sono tra di loro disgiunte).
La generalizzazione è completa e questo vincolo è stato espresso attraverso la restrizione : ContrattoStandard U ContrattoSpeciale inserita nelle condizioni necessarie e sufficienti.

Class(a:**Contratto** complete
intersectionOf
(restriction(a:inverse_of_possiede minCardinality(1))
unionOf(a:**ContrattoStandard** a:**ContrattoFedelta**)))
DisjointClasses(a:**Trasmissioni** a:**Contratto**)
DisjointClasses(a:**Contratto** a:**Utente**)

La classe Contratto è *complete:* è descritta utilizzando condizioni necessarie e sufficienti.
E' l'unione della classe ContrattoStandard e ContrattoSpeciale ed è disgiunta dalle classi Trasmissioni e Utente .
Alla classe viene posta la restrizione di cardinalità per esprimere la cardinalità minima con cui partecipa alle relazione inversa inverse_of_possiede, ad esprimere che il contratto è posseduto da almeno un utente (poiché la cardinalità è 1..1 la cardinlità massima viene espressa aggiungendo alla relazione la proprietà functional) .

- **ContrattoStandard:**

L'oggetto "ContrattoStandard" è stato tradotto in una classe Protègè ed è una sottoclasse di Contratto da cui eredita la proprietà scadenza ed ha come sua proprietà codice tradotta come dataproperty con range string.
La classe è disgiunta con la classe ContrattoSpeciale.
Queste due classi sono tra di loro disgiunte e complete quindi un contratto può essere o speciale o standard.

Class(a:**ContrattoStandard** partial a:**Contratto**)
DisjointClasses(a:**ContrattoStandard** a:**ContrattoFedelta**)

La classe ContrattoStandard è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalla classe ContrattoFedelta.

• **ContrattoSpeciale:**
L'oggetto "ContrattoSpeciale" è stato tradotto in una classe Protègè ed è una sottoclasse di Contratto da cui eredita la proprietà scadenza ed ha come sua proprietà interventi tradotta come dataproperty con range int.
La classe è disgiunta con la classe ContrattoStandard.
Queste due classi sono tra di loro disgiunte e complete quindi un contratto può essere o speciale o standard.
La classe partecipa alla relazione inversa inverse_of_possiede_fedelta con cardinalità 1..1 (un contratto fedeltà è posseduto per fedeltà da uno solo utente).
Class(a:**ContrattoFedelta** complete
restriction(a:**inverse_of_possiede_fedelta** minCardinality(1)))
Class(a:**ContrattoFedelta** partial a:**Contratto**)
DisjointClasses(a:**ContrattoStandard** a:**ContrattoFedelta**)

La classe ContrattoFedelta è *complete:* è descritta utilizzando condizioni necessarie e sufficienti.
La classe ContratoFedelta è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalla classe ContrattoStandard.
Alla classe viene posta la restrizione di cardinalità per esprimere la cardinalità minima con cui partecipa alle relazione inversa inverse_of_possiede_fedeltà, ad esprimere che il contrattofedeltà è posseduto da almeno un utente (poiché la cardinalità è 1..1 la cardinalità massima viene espressa aggiungendo alla relazione la proprietà functional) .

• **Trasmissioni**
L'oggetto "Trasmissioni" è stato tradotto in una classe Protègè che ha come proprietà l'attributo della classe: durata tradotto come dataproperty con range int.
La classe Trasmissioni è superclasse nella generalizzazione che ha come sottoclassi TrasmissioniVoce, TrasmissioneDati (che sono tra di loro disgiunte).
La generalizzazione è completa e questo vincolo è stato espresso attraverso la restrizione : TrasmissioniVoce U TrasmissioneDati inserita nelle condizioni necessarie e sufficienti.

Class(a: **Trasmissioni** complete
intersectionOf
(restriction(a: inverse_of_effettua minCardinality(1))
unionOf(a: **TrasmissioniDati a:TrasmissioniVoce**)))
DisjointClasses(a:**Trasmissioni** a:**Contratto**)
DisjointClasses(a:**Trasmissioni a:Utente**)

La classe Trasmissioni è *complete:* è descritta utilizzando condizioni necessarie e sufficienti.
E' l'unione della classe TrasmissioniDati e TrasmissioniVoce ed è disgiunta dalle classi Contratto e Utente .
Alla classe viene posta la restrizione di cardinalità per esprimere la cardinalità minima con cui partecipa alle relazione inversa inverse_of_effettua, ad espremere che la trasmissione è effettuata da un solo utente (poiché la cardinalità è 1..1 la cardinlità massima viene espressa aggiungendo alla relazione la proprietà functional) .

• **TrasmissioniDati:**

L'oggetto "TrasmissioniDati" è stato tradotto in una classe Protègè ed è una sottoclasse di Trasmissioni da cui eredita la proprietà durata ed ha come sua proprietà volume tradotta come dataproperty con range int.
La classe è disgiunta con la classe TrasmissioniVoce.
Queste due classi sono tra di loro disgiunte e complete quindi una trasmissione può essere voce o dati.
La classe partecipa alla relazione inversa inverse_of_ effettua_dati con cardinalità 1..1 (una trasmissione dati è effettuta da un utente speciale).

Class(a:**TrasmissioniDati** complete
restriction(a: **inverse_of_ effettua_dati** minCardinality(1)))
Class(a:**TrasmissioniDati** partial a:**Trasmissioni**)
DisjointClasses(a:**TrasmissioniDati** a:**TrasmissioniVoce**)

La classe TrasmissioniDati è *complete:* è descritta utilizzando condizioni necessarie e sufficienti.
La classe TrasmissioniDati è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalla classe TrasmissioniVoce.

- **TrasmissioniVoce:**

L'oggetto "TrasmissioniVoce" è stato tradotto in una classe Protègè ed è una sottoclasse di Trasmissioni da cui eredita la proprietà durata ed ha come sua proprietà canale tradotta come dataproperty con range string.
La classe è disgiunta con la classe TrasmissioniDati.
Queste due classi sono tra di loro disgiunte e complete quindi una trasmissione può essere voce o dati.

Class(a:**TrasmissioniVoce** partial a:**Trasmissioni**)
DisjointClasses(a:**TrasmissioniDati** a:**TrasmissioniVoce**)

La classe TrasmissioniVoce è definita *partial* perché ha almeno una condizione necessaria ed è disgiunta dalla classe TrasmissioniDati.

- **Utente**

L'oggetto "Utente" è stato tradotto in una classe Protègè che ha come proprietà gli attributi della classe: codicefiscale, nome,cognome tradotti come dataproperty con range string.
La classe Utente è superclasse nella I-sa e ha come sottoclasse UtenteSpeciale.

Class(a:**Utente** complete
intersectionOf(
restriction(a:**possiede** minCardinality(1))
restriction(a:**possiede_fedelta** maxCardinality(2))))
DisjointClasses(a:**Utente** a:**Contratto**)
DisjointClasses(a:**Trasmissioni** a:**Utente**)
.
La classe Utente è *complete:* è descritta utilizzando condizioni necessarie e sufficienti ed è disgiunta dalle classi Contratto e Trasmissioni .

Alla classe vengono poste le restrizioni di cardinalità per esprimere la cardinalità minima con cui partecipa alle relazione possiede, ad esprimere che utente possiede, e la cardinalità massima con cui partecipa alla relazione possiede_fedelta (un utente possiede_fedelta al più due contratti fedeltà) .

- **UtenteSpeciale:**

L'oggetto "UtenteSpeciale" è stato tradotto in una classe Protègè ed è una sottoclasse di Utente da cui eredita le proprietà ed ha come sua proprietà CodiceConvenzione tradotta come dataproperty con range string.
La classe partecipa alla relazione effettua_dati con cardinalità 0..*, quella che viene indicata di default e perciò non è necessario indicarla attraverso restrizioni cardinali..

Class(a:**UtenteSpeciale** partial a:**Utente**)

La classe Utentespeciale è definita *partial* perché ha almeno una condizione necessaria.

PROPRIETA' (DataProperty)

Gli attributi della classe sono stati tradotti come DataProperty e hanno come la proprietà Functional poiché sono associati ad un solo valore per ogni individuo.

- Gli attributi codicefiscale nome,cognome hanno come dominio la classe Utente come range string e vengono ereditati dalla classe UtenteSpeciale.

DatatypeProperty(a: **codicefiscale** Functional
domain(a: **Utente**)
range(xsd: string))

DatatypeProperty(a: **cognome** Functional
domain(a: **Utente**)
range(xsd: string))

DatatypeProperty(a: **nome** Functional
domain(a: **Utente**)
range(xsd: string))

- L'attributo codiceConvenzione ha come dominio la classe UtenteSpeciale e come range string,la classe eredita anche gli attributi della superclasse Utente.

DatatypeProperty(a: **codiceConvenzione** Functional
domain(a: **UtenteSpeciale**)
range(xsd: string))

- L'attributo scadenza ha come dominio la classe Contratto e come range date ereditato dalle sottoclassi Contrattostandard e ContarttoSpeciale

DatatypeProperty(a: **scadenza** Functional
domain(a: **Contratto**)
range(xsd: date))

- L'attributo codice ha come dominio la classe ContrattoStandard e come range string la classe eredita anche gli attributi della superclasse Contratto.

DatatypeProperty(a: **codice** Functional
domain(a: **ContrattoStandard**)
range(xsd: string))

- L'attributo interventi ha come dominio la classe ContrattoSpeciale e come range string la classe eredita anche gli attributi della superclasse Contratto.

DatatypeProperty(a: **interventi** Functional
domain(a: **ContrattoFedelta**)
range(xsd: int))

- L'attributo durata ha come dominio la classe Trasmissione e come range int ereditato dalle sottoclassi TrasmissioneDati e TrasmissioneVoce.

DatatypeProperty(a: **durata** Functional
domain(a: **Trasmissioni**)
range(xsd: int))

- L'attributo volume ha come dominio la classe TrasmisssioneDati e come range int, la classe eredita anche gli attributi della superclasse Trasmissioni.

DatatypeProperty(a: **volume** Functional
domain(a:**TrasmissioniDati**)
range(xsd: int))

- L'attributo canale ha come dominio la classe TrasmisssioneVoce e come range int, la classe eredita anche gli attributi della superclasse Trasmissioni.

DatatypeProperty(a: **canale** Functional
domain(a: **TrasmissioniVoce**)
range(xsd: int))

RELAZIONI(ObjectProperty)

Le associazioni sono state tradotte in relazioni tra le classi (ObjectProperty) e per ogni relazione è stata definita una relazione inversa:

effettua ←→ inverse_of_effettua
effettua_dati ←→ inverse_of_effettua_dati
possiede ←→ inverse_of_possiede
possiede_fedelta ←→ inverse_of_possiede_possiede_fedelta

• **effettua:**
L'associazione effettua viene tradotta con un ObjectProperty che ha come dominio la classe Utente e come range la classe Trasmissioni e ed è stata definita Functional per indicare il vincolo di cardinalità per il limite superiore

ObjectProperty(a: **effettua** InverseFunctional
inverseOf(a: **inverse_of_effettua**)
domain(a: **Utente**)
range(a: **Trasmissioni**))

La sua relazione inversa è inverse_of_ effettua ha come dominio la classe Trasmissioni e come range la classe Utente ed ha come proprietà inverseFunctional in quanto la relazione inversa è Functional.

ObjectProperty(a: **inverse_of_effettua** Functional
inverseOf(a: **effettua**)
domain(a: **Trasmissioni**)
range(a: **Utente**))

• **effettua_dati:**
L'associazione effettua_dati viene tradotta con un ObjectProperty che ha come dominio la classe UtenteSpeciale e come range la classe TrasmissioniDati e come proprietà inverseFunctional in quanto la relazione inversa è Functional.

ObjectProperty(a: **effettua_dati** InverseFunctional
inverseOf(a: **inverse_of_effettua_dati**)
domain(a: **UtenteSpeciale**)
range(a: **TrasmissioniDati**))

La sua relazione inversa è inverse_of_ effettua_dati ha come dominio la classe TrasmissioniDati e come range la classe UtenteSpeciale ed è stata definita Functional per indicare il vincolo di cardinalità per il limite superiore

ObjectProperty(a: **inverse_of_effettua_dati** Functional
inverseOf(a: **effettua_dati**)
domain(a: **TrasmissioniDati**)
range(a: **UtenteSpeciale**))

L'associazione 'effettua_dati' è derivata dall'associazione 'effettua' pertanto viene definita come subproperty ed eredità tutte le sue caratteristiche(analogamente per la proprietà inversa):

SubPropertyOf(a: **effettua_dati** a: **effettua**)
SubPropertyOf(a: **inverse_of_effettua_dati** a: **inverse_of_effettua**)

• **possiede:**
L'associazione possiede viene tradotta con un ObjectProperty che ha come dominio la classe Utente e come range la classe Contratto e ed ha come proprietà inverseFunctional in quanto la relazione inversa è Functional.

ObjectProperty(a:**possiede** InverseFunctional
inverseOf(a: **inverse_of_possiede**)
domain(a:**Utente**)
range(a:**Contratto**))

La sua relazione inversa è inverse_of_ possiede ha come dominio la classe Contratto e come range la classe Utente ed è stata definita Functional per indicare il vincolo di cardinalità per il limite superiore

ObjectProperty(a: **inverse_of_possiede** Functional
inverseOf(a: **possiede**)
domain(a: **Contratto**)
range(a: **Utente**))

• **possiede_fedeltà:**
L'associazione 'possiede_fedeltà' viene tradotta con un ObjectProperty che ha come dominio la classe Utente e come range la classe ContrattoFedelta

ObjectProperty(a:**possiede_fedelta**
inverseOf(a:**inverse_of_possiede_fedelta**)
domain(a:**Utente**)
range(a:**ContrattoFedelta**))

La sua relazione inversa è inverse_of_ possiede_fedelta ha come dominio la classe ContrattoFedelta e come range la classe Utente

ObjectProperty(a: **inverse_of_possiede_fedelta**
inverseOf(a: **possiede_fedelta**)
domain(a: **ContrattoFedelta**)
range(a: **Utente**))

L'associazione 'possiede_fedelta' è derivata dall'associazione 'possiede' pertanto viene definita come subproperty ed eredità tutte le sue caratteristiche(analogamente per la proprietà inversa):

SubPropertyOf(a: **possiede_fedelta** a: **possiede**)
SubPropertyOf(a: **inverse_of_possiede_fedelta** a: **inverse_of_possiede**)

DL-LiteA

Asserzioni di inclusione di concetti

Contratto ⊑ ¬ Trasmissione
Trasmissione ⊑ ¬ Contratto
Contratto ⊑ ¬ Utente
Utente ⊑ ¬ Contratto
Trasmissioni ⊑ ¬ Utente
Utente ⊑ ¬ Trasmissioni
ContrattoFedelta ⊑ Contratto
ContrattoStandard ⊑ Contratto
ContrattoStandard ⊑ ¬ ContrattoFedelta
ContrattoFedelta ⊑ ¬ ContrattoStandard
TrasmissioneVoce ⊑ Trasmissione
TrasmissioneDati ⊑ Trasmissione
TrasmissioneDati ⊑ ¬ TrasmissioneVoce
TrasmissioneVoce ⊑ ¬ TrasmissioneDati
UtenteSpeciale ⊑ Utente

Contratto ⊑ δ(scadenza)
ContrattoStandard ⊑ δ(codice)
Utente ⊑ δ(codicefiscale)
Utente ⊑ δ(nome)
Utente ⊑ δ(cognome)
UtenteSpeciale ⊑ δ(codiceConvenienza)
Trasmissioni ⊑ δ(durata)
TrasmissioniDati ⊑ δ(volume)
TrasmissioniVoce ⊑ δ(canale)

∃ possiede ⊑ Utente
∃ $(possiede)^{-}$ ⊑ Contratto
∃ inverse_of_ possiede ⊑ Contratto
∃ $(inverse\_of\_\ possiede)^{-}$ ⊑ Utente

∃ possiede_fedelta ⊑ Utente
∃ $(possiede\_fedelta)^{-}$ ⊑ ContrattoFedelta
∃ inverse_of_ possiede_fedelta ⊑ ContrattoFedelta

∃ (inverse_of_ possiede_fedelta)¯ ⊑ Utente

∃ effettua ⊑ Utente
∃ (effettua)¯ ⊑ Trasmissioni
∃ inverse_of_ effettua ⊑ Trasmissioni
∃ (inverse_of_ effettua)¯ ⊑ Utente

∃ effettua_dati ⊑ UtenteSpeciale
∃ (effettua_dati)¯ ⊑ TrasmissioniDati
∃ inverse_of_effettua_dati ⊑ TrasmissioniDati
∃ (inverse_of_effettua_dati)¯ ⊑ UtenteSpeciale

Utente ⊑ ∃ possiede.Contratto
Contratto ⊑ ∃ inverse_of_ possiede. Utente
Trasmissioni ⊑ ∃ inverse_of_ effettua. Utente

Asserzioni di funzionalità di attributi di concetto

(funct scadenza)
(funct codice)
(funct interventi)
(funct codicefiscale)
(funct nome)
(funct cognome)
(funct codiceConvenienza)
(funct durata)
(funct volume)
(funct canale)

Asserzioni di funzionalità di ruolo

(funct inverse_of_ effettua_dati)

Asserzioni di inclusione di domini di valori

ρ(scadenza) ⊑ xs:date
ρ(codice) ⊑ xs:string
ρ(interventi) ⊑ xs:int
ρ(codicefiscale) ⊑ xs:string
ρ(nome) ⊑ xs:string
ρ(cognome) ⊑ xs:string
ρ(codiceConvenienza) ⊑ xs:string
ρ(durata) ⊑ xs:int
ρ(volume) ⊑ xs:int
ρ(canale) ⊑ xs:int

Asserzioni di inclusione di ruoli

possiede $\sqsubseteq$ (inverse_of_ possiede)$^{-}$
inverse_of_ possiede $\sqsubseteq$ (possiede)$^{-}$
possiede_fedelta $\sqsubseteq$ (inverse_of_ possiede_fedelta)$^{-}$
inverse_of_ possiede_fedelta $\sqsubseteq$ (possiede_fedelta)$^{-}$
effettua $\sqsubseteq$ (inverse_of_ effettua)$^{-}$
inverse_of_ effettua $\sqsubseteq$ (effettua)$^{-}$
effettua_dati $\sqsubseteq$ (inverse_of_ effettua_dati)$^{-}$
inverse_of_ effettua_dati $\sqsubseteq$ (effettua_dati)$^{-}$

possiede_fedelta $\sqsubseteq$ possiede
inverse_of_possiede_fedelta $\sqsubseteq$ inverse_of_possiede
effettua_dati $\sqsubseteq$ effettua
inverse_of_ effettua_dati $\sqsubseteq$ inverse_of_ effettua

- **Alfabeto**

L'alfabeto che descrive lo schema UML è formato dai seguenti elementi:
da concetti generali (atomicC), attributi di concetti (atomicCA) , ruoli generali (atomicR).

<alphabet>

<atomicC> **Contratto** </atomicC>
<atomicC> **ContrattoFedelta** </atomicC>
<atomicC> **ContrattoStandard** </atomicC>
<atomicC> **Utente** </atomicC>
<atomicC> **UtenteSpeciale** </atomicC>
<atomicC> **Trasmissioni** </atomicC>
<atomicC> **TrasmissioniVoce** </atomicC>
<atomicC> **TrasmissioniDati** </atomicC>

<atomicCA> **scadenza** </atomicCA>
<atomicCA> **codice** </atomicCA>
<atomicCA> **interventi** </atomicCA>
<atomicCA> **codicefiscale** </atomicCA>
<atomicCA> **nome** </atomicCA>
<atomicCA> **cognome** </atomicCA>
<atomicCA> **codiceConvenienza** </atomicCA>
<atomicCA> **durata** </atomicCA>
<atomicCA> **volume** </atomicCA>
<atomicCA> **canale**</atomicCA>

<atomicR> **possiede** </atomicR>
<atomicR> **inverse_of_possiede** </atomicR>
<atomicR> **possiede_fedelta** </atomicR>
<atomicR> **inverse_of_possiede_fedelta** </atomicR>
<atomicR> **effettua** </atomicR>
<atomicR> **inverse_of_effettua** </atomicR>
<atomicR> **effettua_dati** </atomicR>
<atomicR> **inverse_of_ effettua_dati** </atomicR>

</alphabet>

Le Classi Contratto, ContrattoFedeltà, ContrattoStandard, Utente, UtenteSpeciale, Trasmissione, TrasmissioneDati, TrasmissioneVoce sono concetti generali.
Le proprietà delle classi scadenza, codice, interventi, codiceficale, nome, cognome, codiceConvenienza, durata, volume, canale sono attributi di concetto.
Le relazioni possiede, inverse_of_ possiede, possiede_fedelta, inverse_of_ possiede_ fedelta, effettua, inverse_of_effettua, effettua_dati, inverse_of_effettua_dati sono ruoli generali.

• **Tbox**

La Tbox è usata per rappresentare la conoscenza intensionale attraverso assezioni di inclusione.

-- Contratto ⊑ ¬ Trasmissione

```
<inclusionAssertion>
        <basicC>
            <atomicC> Contratto </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Trasmissione </atomicC>
                </basicC>
            </signedC>
        </generalC>
 </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Contratto non è una Trasmissione (sign="**negative**" vuole indicare la negazione del concetto).
Questa asserzione sta ad indicare che le classi Contratto e Trasmissione sono classi disgiunte.

-- Trasmissione ⊑ ¬ Contratto

```
<inclusionAssertion>
        <basicC>
            <atomicC> Contratto </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Contratto </atomicC>
                </basicC>
            </signedC>
        </generalC>
 </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Trasmissione non è un Contratto (sign="**negative**" vuole indicare la negazione del concetto).
Questa asserzione sta ad indicare che le classi Trasmissione e Contratto sono classi disgiunte.

Queste asserzioni di concetto (Contratto ⊑ ¬Trasmissione ,Trasmissione ⊑ ¬ Contratto) stanno ad indicare che i concetti Contratto e Trasmissione sono disgiunti.

-- Contratto ⊑ ¬ Utente

```
<inclusionAssertion>
        <basicC>
              <atomicC> Contratto </atomicC>
        </basicC>
        <generalC>
              <signedC sign="negative">
                  <basicC>
                        <atomicC> Utente </atomicC>
                  </basicC>
              </signedC>
        </generalC>
  </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Contratto non è una Utente (sign="**negative**" vuole indicare la negazione del concetto).
Questa asserzione sta ad indicare che le classi Contratto e Utente sono classi disgiunte.

-- Utente ⊑ ¬ Contratto

```
<inclusionAssertion>
        <basicC>
              <atomicC> Utente </atomicC>
        </basicC>
        <generalC>
              <signedC sign="negative">
                  <basicC>
                        <atomicC> Contratto </atomicC>
                  </basicC>
              </signedC>
        </generalC>
  </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Utente non è una Contratto (sign="**negative**" vuole indicare la negazione del concetto).
Questa asserzione sta ad indicare che le classi Utente e Contratto sono classi disgiunte.

Queste asserzioni ***(****Contratto ⊑ ¬Utente ,Utente ⊑ ¬ Contratto) stanno ad indicare che i concetti Contratto e Utente sono disgiunti.*

-- Trasmissioni ⊑ ¬ Utente

```
<inclusionAssertion>
        <basicC>
              <atomicC> Trasmissioni </atomicC>
```

```
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Utente </atomicC>
                </basicC>
            </signedC>
        </generalC>
  </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Trasmissioni non è una Utente (sign="**negative**" vuole indicare la negazione del concetto).
Questa asserzione sta ad indicare che le classi Trasmissioni e Utente sono classi disgiunte.

-- Utente ⊑ ¬ Trasmissioni

```
<inclusionAssertion>
        <basicC>
            <atomicC> Utente </atomicC>
        </basicC>
        <generalC>
            <signedC sign="negative">
                <basicC>
                    <atomicC> Trasmissioni </atomicC>
                </basicC>
            </signedC>
        </generalC>
  </inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni Utente non è una Trasmissioni (sign="**negative**" vuole indicare la negazione del concetto).
Questa asserzione sta ad indicare che le classi Utente e Trasmissioni sono classi disgiunte.

Queste asserzioni (Trasmissioni ⊑ ¬Utente ,Utente ⊑ ¬ Trasmissioni) stanno ad indicare che i concetti Trasmissioni e Utente sono disgiunti.

-- ContrattoFedelta ⊑ Contratto

```
<inclusionAssertion>
        <basicC>
           <atomicC>ContrattoFedelta </atomicC>
        </basicC>
        <generalC>
           <signedC sign="positive">
              <basicC>
                 <atomicC> Contratto </atomicC>
              </basicC>
```

```
        </signedC>
    </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni concetto ContrattoFedelta è una Contratto.

Questa asserzione di inclusione di concetto sta ad indicare che la classe ContrattoFedelta è una sottoclasse di Contratto.

-- ContrattoStandard ⊑ Contratto

```
<inclusionAssertion>
    <basicC>
        <atomicC> ContrattoStandard </atomicC>
    </basicC>
    <generalC>
      <signedC sign="positive">
        <basicC>
            <atomicC> Contratto </atomicC>
        </basicC>
      </signedC>
    </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione di concetto vuole indicare che ogni ContrattoStandard è una Contratto.

Questa asserzione di inclusione di concetto sta ad indicare che la classe ContrattoStandard è una sottoclasse di Contratto.

Queste asserzioni stanno ad indicare che i concetti generali ContrattoStandard e ContrattoFedeltà sono derivati dal concetto di Contratto.

-- ContrattoStandard ⊑ ¬ ContrattoFedelta

```
<inclusionAssertion>
    <basicC>
        <atomicC> ContrattoStandard </atomicC>
    </basicC>
    <generalC>
      <signedC sign="negative">
        <basicC>
            <atomicC> ContrattoFedelta </atomicC>
        </basicC>
      </signedC>
    </generalC>
</inclusionAssertion>
```

L’asserzione di inclusione di concetto vuole indicare che ogni ContrattoStandard non è un ContrattoFedelta (sign="**negative**" vuole indicare la negazione del concetto).

-- ContrattoFedelta ⊑ ¬ ContrattoStandard

```
<inclusionAssertion>
        <basicC>
              <atomicC> ContrattoFedelta </atomicC>
        </basicC>
        <generalC>
              <signedC sign="negative">
                   <basicC>
                         <atomicC> ContrattoStandard </atomicC>
                   </basicC>
              </signedC>
        </generalC>
</inclusionAssertion>
```

L’asserzione di inclusione di concetto vuole indicare che ogni ContrattoStandard non è un ContrattoFedelta (sign="**negative**" vuole indicare la negazione del concetto).

Queste due asserzioni (ContrattoStandard ⊑ ¬ ContrattoFedelta e ContrattoFedelta ⊑ ¬ ContrattoStandard) stanno ad indicare che i concetti generali ContrattoStandard e ContrattoFedelta sono tra di loro disguinti.

-- TrasmissioneVoce ⊑ Trasmissione

```
<inclusionAssertion>
        <basicC>
              <atomicC> TrasmissioneVoce</atomicC>
        </basicC>
        <generalC>
           <signedC sign="positive">
              <basicC>
                    <atomicC> Trasmissione </atomicC>
              </basicC>
           </signedC>
        </generalC>
</inclusionAssertion>
```

L’asserzione di inclusione vuole indicare che ogni TrasmissioneVoce è una Trasmissione.

Questa asserzione sta ad indicare che la classe TrasmissioneVoce è una sottoclasse di Trasmissione.

-- TrasmissioneDati ⊑ Trasmissione

```
<inclusionAssertion>
        <basicC>
              <atomicC> TrasmissioneDati</atomicC>
        </basicC>
        <generalC>
          <signedC sign="positive">
            <basicC>
                  <atomicC> Trasmissione </atomicC>
            </basicC>
          </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni TrasmissioneDati è una Trasmissione.

Questa asserzione sta ad indicare che la classe TrasmissioneDati è una sottoclasse di Trasmissione.

-- TrasmissioneDati ⊑ ¬ TrasmissioneVoce

```
<inclusionAssertion>
        <basicC>
              <atomicC> TrasmissioneDati </atomicC>
        </basicC>
        <generalC>
          <signedC sign="negative">
            <basicC>
                  <atomicC>TrasmissioneVoce</atomicC>
            </basicC>
          </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni TrasmissioneDati non è una TrasmissioneVoce (sign="**negative**" vuole indicare la negazione del concetto).

-- TrasmissioneVoce ⊑ ¬ TrasmissioneDati

```
<inclusionAssertion>
        <basicC>
              <atomicC> TrasmissioneVoce</atomicC>
        </basicC>
        <generalC>
          <signedC sign="negative">
            <basicC>
```

```
                    <atomicC>TrasmissioneDati</atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni TrasmissioneVoce non è una TrasmissioneDati (sign="**negative**" vuole indicare la negazione del concetto).

Queste due asserzioni (TrasmissioneDati⊆ ¬ TrasmissioneVoce, TrasmissioneVoce⊆ ¬ TrasmissioneDati) stanno ad indicare che i concetti generali TrasmissioneVoce e TrasmissioneDati sono tra di loro disguinti.

-- UtenteSpeciale ⊑ Utente

```
<inclusionAssertion>
        <basicC>
            <atomicC> UtenteSpeciale</atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC>Utente</atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

L'asserzione di inclusione vuole indicare che ogni UtenteSpeciale è un Utente.

-- (funct scadenza)

```
<funct>
        <atomicCA> scadenza </atomicCA>
</funct>
```

L'attributo di concetto, 'scadenza', è stato definito funzionale.

-- Contratto ⊑ δ(scadenza)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Contratto </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
```

```
                    <CADomain>
                        <atomicCA> scadenza </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Contratto ha associato un attributo di concetto ‘scadenza’.
Dato un attributo di concetto ‘scadenza’ chiamiamo dominio di ‘scadenza’ denotato con δ(scadenza) il set di oggetti che ‘scadenza’ collega a valori.

-- ρ(scadenza) ⊑ xs:date

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> scadenza </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:date </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘scadenza’ è un dominio di valori predefinito xs:date.
Dato un attributo di concetto di ‘scadenza’ chiamiamo range di ‘scadenza’ denotato con ρ(scadenza) il set di valori che ‘scadenza’ collega a oggetti.

-- (funct codice)

```
<funct>
        <atomicCA> codice </atomicCA>
</funct>
```

L’attributo di concetto, ‘codice’, è stato definito funzionale.

-- ContrattoStandard ⊑ δ(codice)

```
<inclusionAssertion>
        <basicC>
            <atomicC> ContrattoStandard </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
```

```
                        <atomicCA> codice </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto ContrattoStandard ha associato un attributo di concetto ‘codice’.
Dato un attributo di concetto ‘codice’ chiamiamo dominio di ‘codice’ denotato con δ(codice) il set di oggetti che ‘codice’ collega a valori.

-- ρ(codice) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> codice </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘codice’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘codice’ chiamiamo range di ‘codice’ denotato con ρ(codice) il set di valori che ‘codice’ collega a oggetti.

-- (funct interventi)

```
<funct>
    <atomicCA> interventi </atomicCA>
</funct>
```

L’attributo di concetto, ‘interventi’, è stato definito funzionale.

-- ContrattoFedelta ⊑ δ(interventi)

```
<inclusionAssertion>
    <basicC>
        <atomicC> ContrattoFedelta </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> interventi </atomicCA>
```

```
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto ContrattoFedelta ha associato un attributo di concetto ‘interventi’.
Dato un attributo di concetto ‘interventi’ chiamiamo dominio di ‘interventi’ denotato con δ(interventi) il set di oggetti che ‘interventi’ collega a valori.

-- ρ(interventi) ⊑ xs:int

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> interventi </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:int </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘interventi’ è un dominio di valori predefinito xs:int.
Dato un attributo di concetto di ‘interventi’ chiamiamo range di ‘interventi’ denotato con ρ(interventi) il set di valori che ‘interventi’ collega a oggetti.

-- (funct codicefiscale)

```
<funct>
        <atomicCA> codicefiscale </atomicCA>
</funct>
```

L’attributo di concetto, ‘codicefiscale’, è stato definito funzionale.

-- Utente ⊑ δ(codicefiscale)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Utente </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> codicefiscale </atomicCA>
                    </CADomain>
```

```
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Utente ha associato un attributo di concetto ‘codicefiscale’.
Dato un attributo di concetto ‘codicefiscale’ chiamiamo dominio di ‘codicefiscale’ denotato con δ(codicefiscale) il set di oggetti che ‘codicefiscale’ collega a valori.

-- ρ(codicefiscale) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> codicefiscale atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘codicefiscale’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘codicefiscale’ chiamiamo range di ‘codicefiscale’ denotato con ρ(codicefiscale) il set di valori che ‘codicefiscale’ collega a oggetti.

-- (funct nome)

```
<funct>
        <atomicCA> nome </atomicCA>
</funct>
```

L’attributo di concetto, ‘nome’, è stato definito funzionale.

-- Utente ⊑ δ(nome)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Utente </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> nome </atomicCA>
                    </CADomain>
                </basicC>
```

```
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto Utente ha associato un attributo di concetto ‘nome’.
Dato un attributo di concetto ‘nome’ chiamiamo dominio di ‘nome’ denotato con δ(nome) il set di oggetti che ‘nome’ collega a valori.

-- ρ(nome) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> nome </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘nome’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘nome’ chiamiamo range di ‘nome’ denotato con ρ(nome) il set di valori che ‘nome’ collega a oggetti.

-- (funct cognome)

```
<funct>
        <atomicCA> cognome </atomicCA>
</funct>
```

L’attributo di concetto, ‘cognome’, è stato definito funzionale.

-- Utente ⊑ δ(cognome)

```
<inclusionAssertion>
        <basicC>
            <atomicC> Utente </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> cognome </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
```

```
        </generalC>
</inclusionAssertion>
```

Ogni concetto Utente ha associato un attributo di concetto ‘cognome’.
Dato un attributo di concetto ‘cognome’ chiamiamo dominio di ‘cognome’ denotato con δ(cognome) il set di oggetti che ‘cognome’ collega a valori.

-- ρ(cognome) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> cognome </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘cognome’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘cognome’ chiamiamo range di ‘cognome’ denotato con ρ(cognome) il set di valori che ‘cognome’ collega a oggetti.

-- (funct codiceConvenienza)

```
<funct>
        <atomicCA> codiceConvenzione </atomicCA>
</funct>
```

L’attributo di concetto, ‘codiceConvenzione’, è stato definito funzionale.

-- UtenteSpeciale ⊑ δ(codiceConvenienza)

```
<inclusionAssertion>
        <basicC>
            <atomicC> UtenteSpecile </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> codiceConvenienza </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
```

</inclusionAssertion>

Ogni concetto UtenteSpeciale ha associato un attributo di concetto ‘codiceConvenienza’.
Dato un attributo di concetto ‘codiceConvenienza’ chiamiamo dominio di ‘codiceConvenienza’ denotato con δ(codiceConvenienza) il set di oggetti che ‘codiceConvenienza’ collega a valori.

-- ρ(codiceConvenienza) ⊑ xs:string

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> codiceConvenienza </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:string </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘codiceConvenienza’ è un dominio di valori predefinito xs:string.
Dato un attributo di concetto di ‘codiceConvenienza’ chiamiamo range di ‘codiceConvenienza’ denotato con ρ(codiceConvenienza) il set di valori che ‘codiceConvenienza’ collega a oggetti.

-- (funct durata)

```
<funct>
    <atomicCA> durata </atomicCA>
</funct>
```

L’attributo di concetto, ‘durata’, è stato definito funzionale.

-- Trasmissioni ⊑ δ(durata)

```
<inclusionAssertion>
    <basicC>
        <atomicC> Trasmissioni </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> durata </atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto Trasmissioni ha associato un attributo di concetto ‘durata’.
Dato un attributo di concetto ‘durata’ chiamiamo dominio di ‘durata’ denotato con δ(durata) il set di oggetti che ‘durata’ collega a valori.

-- ρ(durata) ⊑ xs:int

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> durata </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:int </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘durata’ è un dominio di valori predefinito xs:int.
Dato un attributo di concetto di ‘durata’ chiamiamo range di ‘durata’ denotato con ρ(durata) il set di valori che ‘durata’ collega a oggetti.

-- (funct volume)

```
<funct>
    <atomicCA> volume </atomicCA>
</funct>
```

L’attributo di concetto, ‘volume’, è stato definito funzionale.

-- TrasmissioniDati ⊑ δ(volume)

```
<inclusionAssertion>
    <basicC>
        <atomicC> TrasmissioniDati </atomicC>
    </basicC>
    <generalC>
        <signedC sign="positive">
            <basicC>
                <CADomain>
                    <atomicCA> volume </atomicCA>
                </CADomain>
            </basicC>
        </signedC>
    </generalC>
</inclusionAssertion>
```

Ogni concetto TrasmissioniDati ha associato un attributo di concetto ‘volume’.
Dato un attributo di concetto ‘volume’ chiamiamo dominio di ‘volume’ denotato con δ(volume) il set di oggetti che ‘volume’ collega a valori.

-- ρ(volume) ⊑ xs:int

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> volume </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:int </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘volume’ è un dominio di valori predefinito xs:int.
Dato un attributo di concetto di ‘volume’ chiamiamo range di ‘volume’ denotato con ρ(volume) il set di valori che ‘volume’ collega a oggetti.

-- (funct canale)

```
<funct>
        <atomicCA> canale </atomicCA>
</funct>
```

L’attributo di concetto, ‘canale’, è stato definito funzionale.

-- TrasmissioniVoce ⊑ δ(canale)

```
<inclusionAssertion>
        <basicC>
            <atomicC> TrasmissioniVoce </atomicC>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <CADomain>
                        <atomicCA> canale </atomicCA>
                    </CADomain>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

Ogni concetto TrasmissioniVoce ha associato un attributo di concetto ‘canale’.

Dato un attributo di concetto ‘canale’ chiamiamo dominio di ‘canale’ denotato con δ(canale) il set di oggetti che ‘canale’ collega a valori.

-- ρ(canale) ⊑ xs:int

```
<inclusionAssertion>
    <basicV>
        <ARange>
            <atomicCA> canale </atomicCA>
        </ARange>
    </basicV>
    <generalV>
        <predefinedV> xs:int </predefinedV>
    </generalV>
</inclusionAssertion>
```

Il range dell’attributo di concetto ‘canale’ è un dominio di valori predefinito xs:int.
Dato un attributo di concetto di ‘canale’ chiamiamo range di ‘canale’ denotato con ρ(canale) il set di valori che ‘canale’ collega a oggetti.

-- possiede ⊑ (inverse_of_ possiede)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> possiede </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> inverse_of_possiede </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘possiede’ è un’istanza del ruolo inverso di ‘ inverse_of_ possiede’.

Questa asserzione di inclusione di ruoli sta ad indicare che il ruolo ‘inverse_of_ possiede’ è il ruolo inverso del ruolo ‘possiede’.

-- inverse_of_ possiede ⊑ (possiede)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> inverse_of_possiede </atomicR>
        </basicR>
        <generalR>
```

```
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> possiede </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ possiede’ è un’istanza del ruolo inverso di ‘possiede’.

Questa asserzione di inclusione di ruoli sta ad indicare che il ruolo ‘possiede’ è il ruolo inverso del ruolo ‘inverse_of_possiede’.

-- ∃ possiede ⊑ Utente

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                    <atomicR> possiede </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Utente </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘possiede’ è un’ istanza del concetto Utente.

-- ∃ (possiede)$^{-}$ ⊑ Contratto

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                    <atomicR> possiede </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
```

```
                    <atomicC> Contratto </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘possiede’ è un’ istanza del concetto Contratto.

-- ∃ inverse_of_ possiede ⊑ Contratto

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                        <atomicR> inverse_of_possiede </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Contratto </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_possiede’ è un’ istanza del concetto Contratto.

-- ∃ (inverse_of_ possiede)¯ ⊑ Utente

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                        <atomicR> inverse_of_possiede </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Utente </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘inverse_of_possiede’ è un’ istanza del concetto Utente.

-- Utente ⊑ ∃ possiede.Contratto

```
<inclusionAssertion>
        <basicC>
                <atomicC> Utente </atomicC>
        </basicC>
        <generalC>
                <qualifiedExists>
                        <basicR dir="direct">
                                <atomicR> possiede </atomicR>
                        </basicR>
                        <generalC>
                                <signedC sign="positive">
                                    <basicC>
                                            <atomicC> Contratto </atomicC>
                                    </basicC>
                                </signedC>
                        </generalC>
                 </qualifiedExists>
        </generalC>
</inclusionAssertion>
```

Ogni Utente possiede almeno un Contratto

Questa asserzione (Utente ⊑ ∃possiede.Contratto) sta ad indicare che la classe Utente partecipa alla relazione possiede con la classe Contratto con cardinalità minima 1 (un utente deve possedere almeno ad un contratto).

-- Contratto ⊑ ∃ inverse_of_ possiede. Utente

```
<inclusionAssertion>
        <basicC>
                <atomicC> Contratto </atomicC>
        </basicC>
        <generalC>
                <qualifiedExists>
                        <basicR dir="direct">
                                <atomicR> inverse_of_ possiede </atomicR>
                        </basicR>
                        <generalC>
                                <signedC sign="positive">
                                    <basicC>
                                            <atomicC> Utente </atomicC>
                                    </basicC>
                                </signedC>
```

```
                </generalC>
            </qualifiedExists>
        </generalC>
</inclusionAssertion>
```

Ogni Contratto è posseduto da almeno un Utente.

Questa asserzione (Contratto⊑ ∃ inverse_of_ possiede. Utente) sta ad indicare che la classe Contratto partecipa alla relazione inverse_of_possiede con la classe Utente con cardinalità minima 1

-- possiede_fedelta ⊑ (inverse_of_ possiede_fedelta)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> possiede_fedelta </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> inverse_of_possiede_fedelta </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘possiede_fedelta’ è un’istanza del ruolo inverso di ‘ inverse_of_ possiede_fedelta’.

Questa asserzione di inclusione di ruolo sta ad indicare che il ruolo ‘inverse_of_ possiede_fedelta’ è il ruolo inverso del ruolo ‘possiede_fedelta’.

-- inverse_of_ possiede_fedelta ⊑ (possiede_fedelta)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> inverse_of_possiede_fedelta </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> possiede_fedelta </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Questa asserzione di inclusione di ruolo sta ad indicare che il ruolo ‘ possiede_fedelta’ è il ruolo inverso del ruolo ‘inverse_of_possiede_fedelta’.

∃ possiede_fedelta ⊑ Utente

```
<inclusionAssertion>
        <basicC>
                <exists>
                    <basicR dir="direct">
                            <atomicR> possiede_fedelta </atomicR>
                    </basicR>
                </exists>
        </basicC>
        <generalC>
                <signedC sign="positive">
                    <basicC>
                            <atomicC> Utente </atomicC>
                    </basicC>
                </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo di ‘possiede_fedelta’ è un’ istanza del concetto Utente.

-- ∃ (possiede_fedelta)⁻ ⊑ ContrattoFedelta

```
<inclusionAssertion>
        <basicC>
                <exists>
                    <basicR dir="inverse">
                            <atomicR> possiede_fedelta </atomicR>
                    </basicR>
                </exists>
        </basicC>
        <generalC>
                <signedC sign="positive">
                    <basicC>
                            <atomicC> ContrattoFedelta </atomicC>
                    </basicC>
                </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘possiede_fedelta’ è un’ istanza del concetto ContrattoFedelta.

-- ∃ inverse_of_ possiede_fedelta ⊑ ContrattoFedelta

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                        <atomicR> inverse_of_possiede </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> ContrattoFedelta </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo di ‘inverse_of_possiede’ è un’ istanza del concetto ContrattoFedelta.

-- ∃ (inverse_of_ possiede_fedelta)¯ ⊑ Utente

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                        <atomicR> inverse_of_possiede_fedelta </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Utente </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘inverse_of_possiede’ è un’ istanza del concetto Utente

-- possiede_fedelta ⊑ possiede

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> possiede_fedelta </atomicR>
```

```
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="direct">
                    <atomicR> possiede </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘possiede_fedelta’ è un’istanza del ruolo ‘possiede’.

-- inverse_of_possiede_fedelta ⊑ inverse_of_possiede

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> inverse_of_possiede_fedelta </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="direct">
                    <atomicR> inverse_of_possiede </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_possiede_fedelta’ è un’istanza del ruolo ‘inverse_of_possiede’.

-- effettua ⊑ (inverse_of_ effettua)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> effettua </atomicR>
        </basicR>
        <generalR>
            <signedR sign="positive">
                <basicR dir="inverse">
                    <atomicR> inverse_of_effettua </atomicR>
                </basicR>
            </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘effettua’ è un’istanza del ruolo inverso di ‘ inverse_of_ effettua’.

Questa asserzione di inclusione di ruolo sta ad indicare che il ruolo 'inverse_of_ effettua' è il ruolo inverso del ruolo 'effettua'.

-- inverse_of_ effettua ⊑ (effettua)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
                <atomicR> inverse_of_effettua </atomicR>
        </basicR>
        <generalR>
                <signedR sign="positive">
                        <basicR dir="inverse">
                                <atomicR> effettua </atomicR>
                        </basicR>
                </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo 'inverse_of_effettua' è un'istanza del ruolo inverso di 'effettua'.

Questa asserzione di inclusione di ruolo sta ad indicare che il ruolo 'effettua' è il ruolo inverso del ruolo 'inverse_of_effettua'.

-- ∃ effettua ⊑ Utente

```
<inclusionAssertion>
        <basicC>
                <exists>
                    <basicR dir="direct">
                            <atomicR> effettua</atomicR>
                    </basicR>
                </exists>
        </basicC>
        <generalC>
                <signedC sign="positive">
                    <basicC>
                            <atomicC> Utente </atomicC>
                    </basicC>
                </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo 'effetta' è un' istanza del concetto Utente.

-- ∃ (effettua)⁻ ⊑ Trasmissioni

```
<inclusionAssertion>
```

```
        <basicC>
            <exists>
                <basicR dir="inverse">
                        <atomicR> effettua </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Trasmissioni </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso ‘effetta’ è un’ istanza del concetto Trasmissioni

-- ∃ inverse_of_ effettua ⊑ Trasmissioni

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="direct">
                        <atomicR> inverse_of_effettua </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> Trasmissioni </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_effetta’ è un’ istanza del concetto Trasmissioni

-- ∃ (inverse_of_ effettua)¯ ⊑ Utente

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                        <atomicR> inverse_of_effettua </atomicR>
                </basicR>
            </exists>
```

```
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                    <atomicC> Utente </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso di ‘inverse_of_effetta’ è un’ istanza del concetto Utente

-- Trasmissioni ⊑ ∃ inverse_of_ effettua. Utente

```
<inclusionAssertion>
        <basicC>
            <atomicC> Trasmissioni </atomicC>
        </basicC>
        <generalC>
            <qualifiedExists>
                <basicR dir="direct">
                    <atomicR> inverse_of_ effettua </atomicR>
                </basicR>
                <generalC>
                    <signedC sign="positive">
                        <basicC>
                            <atomicC> Utente </atomicC>
                        </basicC>
                    </signedC>
                </generalC>
            </qualifiedExists>
        </generalC>
</inclusionAssertion>
```

Ogni Trasmissioni è effettuata (inverse_of_ effettua) almeno un Utente

Questa asserzione (Trasmissioni ⊑ inverse_of_ effettua. Utente) sta ad indicare che la classe Trasmissioni partecipa alla relazione inverse_of_ effettua. con la classe Utente con cardinalità minima 1

-- funct(inverse_of_ effettua_dati)

```
<funct>
    <basicR dir="direct">
        <atomicCA> inverse_of_ effettua_dati </atomicCA>
    </basicR >
</funct>
```

Il ruolo, ‘inverse_of_ effettua_dati’, è stato definito funzionale.

Questa asserzione di funzionalità di ruolo (funct inverse_of_ effettua_dati) sta ad indicare la cardinalità (1,1) della relazione ‘inverse_of_ effettua_dati’.

-- effettua_dati ⊑ (inverse_of_ effettua_dati)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
                <atomicR> effettua_dati </atomicR>
        </basicR>
        <generalR>
                <signedR sign="positive">
                    <basicR dir="inverse">
                            <atomicR> inverse_of_effettua_dati </atomicR>
                    </basicR>
                </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘effettua_dati’ è un’istanza del ruolo inverso di ‘ inverse_of_ effettua_dati’.

Questa asserzione di inclusione di ruolo sta ad indicare che il ruolo ‘inverse_of_ effettua_dati’ è il ruolo inverso del ruolo ‘effettua_dati’.

-- inverse_of_ effettua_dati ⊑ (effettua_dati)⁻

```
<inclusionAssertion>
        <basicR dir="direct">
                <atomicR> inverse_of_effettua_dati </atomicR>
        </basicR>
        <generalR>
                <signedR sign="positive">
                        <basicR dir="inverse">
                                <atomicR> effettua_dati </atomicR>
                        </basicR>
                </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_effettua_dati’ è un’istanza del ruolo inverso di ‘effettua_dati’.

Questa asserzione di inclusione di ruolo sta ad indicare che il ruolo ‘effettua_dati’ è il ruolo inverso del ruolo ‘inverse_of_effettua_dati’.

-- ∃ effettua_dati ⊑ UtenteSpeciale

```
<inclusionAssertion>
        <basicC>
                <exists>
                    <basicR dir="direct">
                            <atomicR> effettua_dati </atomicR>
                    </basicR>
                </exists>
        </basicC>
        <generalC>
                <signedC sign="positive">
                    <basicC>
                            <atomicC> UtenteSpeciale </atomicC>
                    </basicC>
                </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘effetta_dati’ è un’ istanza del concetto UtenteSpeciale

-- ∃ (effettua_dati)⁻ ⊑ TrasmissioniDati

```
<inclusionAssertion>
        <basicC>
                <exists>
                    <basicR dir="inverse">
                            <atomicR> effettua_dati </atomicR>
                    </basicR>
                </exists>
        </basicC>
        <generalC>
                <signedC sign="positive">
                    <basicC>
                            <atomicC> TrasmissioniDati </atomicC>
                    </basicC>
                </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso ‘effetta_dati’ è un’ istanza del concetto TrasmissioniDati.

-- ∃ inverse_of_effettua_dati ⊑ TrasmissioniDati

```
<inclusionAssertion>
```

```
        <basicC>
            <exists>
                <basicR dir= "direct">
                        <atomicR> inverse_of_effettua_dati </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> TrasmissioniDati </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo ‘inverse_of_effetta_dati’ è un’ istanza del concetto TrasmissioniDati.

-- ∃ (inverse_of_effettua_dati)¯ ⊑ UtenteSpeciale

```
<inclusionAssertion>
        <basicC>
            <exists>
                <basicR dir="inverse">
                        <atomicR> inverse_of_effettua_dati </atomicR>
                </basicR>
            </exists>
        </basicC>
        <generalC>
            <signedC sign="positive">
                <basicC>
                        <atomicC> UtenteSpeciale </atomicC>
                </basicC>
            </signedC>
        </generalC>
</inclusionAssertion>
```

La prima componente del ruolo inverso ‘inverse_of_effetta_dati’ è un’ istanza del concetto UtenteSpeciale.

-- effettua_dati ⊑ effettua

```
<inclusionAssertion>
        <basicR dir="direct">
            <atomicR> effettua_dati</atomicR>
        </basicR>
```

```
        <generalR>
                <signedR sign="positive">
                    <basicR dir="direct">
                            <atomicR> effettua</atomicR>
                    </basicR>
                </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘effettua_dati’ è un’istanza del ruolo ‘effettua’.

-- inverse_of_ effettua_dati ⊑ inverse_of_effettua

```
<inclusionAssertion>
        <basicR dir="direct">
              <atomicR>inverse_of_effettua_dati</atomicR>
        </basicR>
        <generalR>
                <signedR sign="positive">
                    <basicR dir="direct">
                            <atomicR>inverse_of_effettua</atomicR>
                    </basicR>
                </signedR>
        </generalR>
</inclusionAssertion>
```

Ogni istanza del ruolo ‘inverse_of_ effettua_dati’ è un’istanza del ruolo ‘inverse_of_ effettua’.

Conclusioni:

1) *<!-- IntersectionOf not totally expressible in DL-liteA -->*

La classe Trasmissioni è espressa in DL-lite come l’intersezione di:
l’unione di classi(TrasmissioniDati, TrasmissioniVoce)
la restrizione (che ha lo scopo di indicare la cardinalita minima della relazione inverse_of_ effettua con la classe Appartamento)

```
Class( a: Trasmissioni complete
        intersectionOf(restriction(a: inverse_of_effettua someValuesFrom(a: Utente))
        unionOf(a: TrasmissioniDati a: TrasmissioniVoce)))
```

In DL-liteA il concetto di intersezione non è totalmente esprimibile.

2) *<!-- UnionOf not totally expressible in DL-liteA -->*

La classe Trasmissioni è la generalizzazione completa e disgiunta delle classi TrasmissioniDati

TrasmissioniVoce.
In DL-liteA possimo esprimere le proprietà di disgiunzione tramite la seguente asserzione di inclusione di concetti: TrasmissioneDati ⊑ ¬ TrasmissioneVoce e TrasmissioneVoce ⊑ ¬ TrasmissioneDati e le proprietà di completezza con le seguenti asserzioni di inclusione di concetti TrasmissioneDati ⊑ Trasmissione, TrasmissioneVoce ⊑ Trasmissione, Bagno ⊑ Locale

L'unione di queste classi non è completamente esprimibile in DL-liteA.

3) *<!-- IntersectionOf not totally expressible in DL-liteA -->*

La classe Trasmissioni è espressa in DL-lite come l'intersezione di:
l'unione di classi(TrasmissioniDati, TrasmissioniVoce)
la restrizione (che ha lo scopo di indicare la cardinalita massima della relazione 'possiede' con la classe Contratto)

```
Class( a: Utente complete
        intersectionOf(restriction(a: possiede someValuesFrom(a: Contratto))
        restriction(a: possiede_fedelta maxCardinality(2)) ))
```

In DL-liteA il concetto di intersezione non è totalmente esprimibile.

4) *<!-- Cardinality different from 1 not expressible in DL-liteA-->*

La classe Utente partecipa relazione 'possiede' con la classe Contratto la cardinalita massima di 2. Il Dl-lite questo viene espresso tramite una restizione:

```
restriction(a: possiede_fedelta maxCardinality(2))
```

In Dl-liteA la cardinalità dfferente da 1 non è esprimibile.

5) *<!-- NOT IN DL-LiteA!There is a conflict :the atomic Role 'inverse_of_effettua' is qualified exists then it cannot be functional -->*

La base di conoscenza in DL-lite-A K=<T,A>, è ottenuta attraverso delle restrizioni: ovvero T (TBox) è una DL-lite$_{FR}$ Tbox che soddisfa determinate condizioni. In particolare, in questo caso:
-- Per ogni ruolo atomico e inverso di un ruolo atomico Q compare in un concetto della forma ∃Q.C, le asserzioni (funct Q) e (funct Q^{-}) non sono in T.

Poiché il ruolo inverse_of_ effettua compare nella seguente asserzione:
Trasmissioni ⊑ ∃ inverse_of_ effettua. Utente e quindi è quantificato esistenzialmente, l'asserzione (funct inverse_of_effettua) non è in T, e quindi il ruolo non può essere funzionale.

6) *<!-- NOT IN DL-LiteA!there is a conflict : the atomic Role 'effettua' cannot be functional if exists the ISA between 'effettua_dati' and 'effettua'*

-->

<!-- NOT IN DL-LiteA!there is a conflict : the atomic Role ' inverse_of_effettua' cannot be functional if exists the ISA between ' inverse_of_effettua_dati' and ' nverse_of_effettua' -->

La base di conoscenza in DL-lite-A K=<T,A>, è ottenuta attraverso delle restrizioni: ovvero T (TBox) è una DL-liteFR Tbox che soddisfa determinate condizioni. In particolare, in questo caso:

-- Per ogni asserzione di inclusione di ruolo Q ⊑ R in T , dove R è un ruolo atomico o l'inverso di un ruolo atomico, l'asserzione (funct R) e (funct R^{-}) non sono in T.

Poiché il ruolo effettua compare nella seguente asserzione:
effettua_dati ⊑ effettua
inverse_of_ effettua_dati ⊑ inverse_of_ effettua
l'asserzione (funct effettua) ed (funct inverse_of_effettua) non è in T, e quindi i ruoli non possono essere funzionali.

7) *<!-- NOT IN DL-LiteA!there is a conflict : the atomic Role 'possiede' cannot be functional if exists the ISA between 'possiede_fedelta' and 'possiede' -->*

<!-- NOT IN DL-LiteA!there is a conflict : the atomic Role 'inverse_of_possiede' cannot be functional if exists the ISA between 'inverse_of_possiede_fedelta' and 'inverse_of_possiede' -->

La base di conoscenza in DL-lite-A K=<T,A>, è ottenuta attraverso delle restrizioni: ovvero T (TBox) è una DL-liteFR Tbox che soddisfa determinate condizioni. In particolare, in questo caso:

-- Per ogni asserzione di inclusione di ruolo Q ⊑ R in T , dove R è un ruolo atomico o l'inverso di un ruolo atomico, l'asserzione (funct R) e (funct R^{-}) non sono in T.

Poiché il ruolo effettua compare nella seguente asserzione:
possiede_fedelta ⊑ possiede
inverse_of_ possiede_fedelta ⊑ inverse_of_ possiede
l'asserzione (funct possiede) ed (funct inverse_of_ possiede) non è in T, e quindi i ruoli non possono essere funzionali.

8) *Attributo di ruolo*

In DL-liteA è possibile esprimere attributi di ruolo denotando la relazione binaria tra coppie di oggetti e valori (non esprimibile in DL-lite).
La relazione effettuaDati presenta un attributo di relazione 'compressione' di tipo intero. Questo può essere espresso attraverso un attributo di ruolo 'compressione' per il ruolo 'effettuaDati'.
Le espressioni sono le seguenti:

Asserzione funzionale di attributo di ruolo:

(funct compressione)

L'attributo di ruolo 'compressione' è funzionale

Asserzioni di inclusione di concetti:

UtenteSpeciale $\sqsubseteq \exists \delta$(compressione)
TrasmissioneDati $\sqsubseteq \exists \delta(\text{compressione})^{-}$

Ogni UtenteSpeciale deve partecipare al ruolo effettuaDati (dominio di compressione) avendo associato un attributo di ruolo 'compressione'.
Dato un attributo di ruolo 'compressione' chiamiamo dominio di 'compressione' denotato con δ(compressione) il set di oggetti che 'compressione' collega a valori.
Ogni TrasmissioneDati deve partecipare al ruolo 'inverse_of_ effettuaDati' (dominio inverso di compressione) avendo associato un attributo di ruolo 'compressione'.

Asserzioni di inclusione di ruolo:

δ(compressione) $\sqsubseteq$ effettuaDati
$\delta(\text{compressione})^{-} \sqsubseteq$ inverse_of_effettuaDati

Il dominio dell'attributo di ruolo 'compressione' è il ruolo 'effettuaDati'
Il dominio inverso dell'attributo di ruolo 'compressione' è il ruolo 'inverse_of_effettuaDati'

Asserzioni di inclusione di dominio di valori:

ρ(compressione) $\sqsubseteq$ xs:int

Il range dell'attributo di ruolo 'compressione' è xs:int

9. Conclusioni

Il presente lavoro mette in evidenza come DL e DL-liteA abbiano un potere espressivo diverso: alcuni concetti possono essere espressi con l'uno e non con altro e viceversa.

In conclusione vengono qui riassunti i limiti di ciascun linguaggio

in DL:
- Non si possono esplicitare gli attributi delle relazioni in quanto OWL non permette di inserire una *datatype property* come subproperty di una *objectproperty.*
- Non è possibile rappresentare le operazioni delle classi
- Non è possibile rappresentare associazioni ternarie tra le classi a meno che non venga ristrutturato il diagramma UML sostituendole con due associazioni binarie

- Non è possibile rappresentare ‘ordered’ che definisce una classe come un insieme di elementi ordinati di un’altra classe. Per poter rappresentare questo concetto si potrebbe però creare una nuova relazione che leghi queste due classi e che abbia come attributo didellarelazione ‘indice’.

in DL-LiteA:

- L’unione non è totalmente esprimibile: per poter esprimere una generalizzazione completa e disgiunta bisogna specificare che una classe è superclasse, e che le sottoclassi sono contenute nella superclasse e che l’una non può essere contenuta nell’altra.
- L’intersezione non è completamente esprimibile
- La proprietà transitva tra le classi legate da una relazione non è esprimibile
- Non è possibile esprimere la cardinalità differente da 1
- Se viene utilizzato un quantificatore esistenziale per esprimere la cardinalità di un’associazione allora questa non può essere definita *functional*
- Se da un’associazione se ne ha un’altra che deriva da questa, allora questa non può essere definita *functional*

Riferimenti bibliografici

- G. De Giacomo, dispense del corso di “Seminari di ingegneria del software” anno accademico 2005-2006, http://www.dis.uniroma1.it/~degiacomo
- Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe, ProtégéOWLTutorial, http://protege.stanford.edu/ www.w3.org/2004/OWL/
- Matthew Horridge,Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe - The University Of Manchester, Stanford University, http://www.coode.org/resources/tutorials/ProtegeOWLTutorial.pdf
- D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider, “The Description Logic Handbook” edited by Franz Baader.
- N. Capuano, “Ontologie e OWL: Teoria e Pratica”
- D. Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Riccardo Rosati “Linking Data to Ontologies: The Description Logic DL-LiteA”