

Progetto Web Semantico  
***Marathon TV Party Ontology***  
***(mtparty)***  
Web Semantico  
a.a 2022-2023

Marica Pasquali - 0000982560  
(marica.pasquali@studio.unibo.it)

29 maggio 2023

# Indice

<b>Introduzione</b>	<b>3</b>
<b>1 Ontologie esistenti utilizzate</b>	<b>4</b>
<b>2 Descrizione dell'ontologia modellata</b>	<b>6</b>
2.1 TVParty . . . . .	8
2.2 Movie/SerieTV . . . . .	16
2.3 Agents . . . . .	19
2.3.1 Organizer . . . . .	20
2.3.2 Participant . . . . .	20
2.3.3 Seller . . . . .	21
2.4 Location . . . . .	23
2.5 Selling Object . . . . .	26
2.5.1 Ticket . . . . .	26
2.5.2 Food . . . . .	33
<b>3 SWRL</b>	<b>41</b>
3.1 Rule 1 e Rule 2 . . . . .	41
3.2 Rule 3 . . . . .	41
3.3 Rule 4 e Rule 5 . . . . .	41
3.4 Rule 6 . . . . .	42
<b>4 Queries</b>	<b>43</b>
4.1 Lista dei biglietti venduti validi per un TV Party pubblico . . . .	44
4.2 Incassi dei TV Party Pubblici . . . . .	45
4.3 Genere (movie/serie tv) più frequente . . . . .	47
4.4 Lista dei luoghi più gettonati per TV Party . . . . .	48
4.5 Rating medio di ogni TVParty . . . . .	49
4.6 Lista ordinata degli show in ogni TV Party . . . . .	50
<b>Conclusioni</b>	<b>51</b>
<b>Terminologia del Knowledge Graph (KG)</b>	<b>53</b>
<b>Acronimi</b>	<b>55</b>

<b>Glossario</b>	<b>56</b>
<b>Sitografia</b>	<b>59</b>

# Introduzione

Chi di noi non è andato al cinema o al un drive in o semplicemente è andato a casa di un amica/o, fidanzata/o o semplicemente in famiglia per vedere un film, un saga di film o una serie tv? Non essendo noi stessi gli organizzatori di questi tipi di eventi, la nostra unica preoccupazione è se il film o la serie tv ci possa interessare e quindi valga la pena “sprecare” il nostro tempo per vederlo/a; ma esiste una grande quantità di “conoscenza” riguardante l’organizzazione di questi eventi.

In questa relazione viene esplorata questa “conoscenza” attraverso la modellazione dell’ontologia *Marathon TV Party* (***mtparty***) riguardante l’organizzazione di eventi inerenti alla visioni di film o serie tv.

# Capitolo 1

## Ontologie esistenti utilizzate

L'ontologia *Marathon TV Party* (*mtparty*) è stata scritta in “*Terse RDF Triple Language (Turtle)*” utilizzando l'editor Protégé.

I prefissi utilizzati sono: *rdf*, *rdfs*, *xsd*, *owl*, *dc*, *dcterms*, *foaf*, *schema*, *event*, *fo*, *gr*, *geo* e *time*.

L'ontologia *Dublic Core*[1] (*dc* e *dcterms*) è utilizzata per descrivere l'ontologia (titolo, creatore, descrizione, ecc.).

*owl*, *rdf*, *rdfs* e *xsd* sono utilizzate per definire le classi, proprietà e restrizioni nella definizione dei concetti dell'ontologia.

L'ontologia *Friend Of A Friend*[3] (*foaf*) è un'ontologia utilizzata per descrivere sia le relazioni tra le persone che le caratteristiche dell'individuo stesso. In questo caso il suo utilizzo si è reso necessario per modellare gli agenti coinvolti.

L'ontologia *schema.org* (*schema*) è un vocabolario di definizioni standard che devono essere utilizzati dai webmaster per creare risultati multimediali su un determinato argomento di interesse. In questo caso sono stati utilizzati alcuni dei suoi termini per definire i luoghi in cui si svolge l'evento, i film o le serie tv da guardare.

Per non creare da zero l'ontologia sono state importate alcune ontologie per poter utilizzare le proprie definizioni e quindi facilitare la modellazione.

- **Event Ontology** [2] (*event*) : ontologia incentrata sulla nozione di evento. La definizione di evento è la seguente:

Events are primarily linguistic or cognitive in nature. That is, the world does not really contain events. Rather, events are the way by which agents classify certain useful and relevant patterns of change.

E' stata utilizzata come base per modellare l'evento che riguarda la visione di film o serie tv; nel capitolo *Descrizione dell'ontologia modellata* viene illustrato meglio il suo utilizzo.

- **GoodRelations Ontology** [5] (*gr*): ontologia standard per la descrizione di risorse impiegate nei comuni siti di commercio elettronico: prodotti, prezzi, magazzini, compagnie. E' stata utilizzata per descrivere la compravendita dei biglietti e del cibo.
- **Food Ontology** [4] (*fo*): l'ontologia è un'estensione di **GoodRelations Ontology** per la descrizione di alimenti e prodotti alimentari. E' stata utilizzata per lo stesso motivo.
- **Time Ontology** [12] (*time*): ontologia dei concetti temporali, per descrivere le proprietà temporali delle risorse nel mondo o descritte nelle pagine web. Questa ontologia viene importata direttamente da *Event Ontology*.

## Capitolo 2

# Descrizione dell'ontologia modellata

L'ontologia *Marathon TV Party* (*mtparty*) è stata sviluppata con l'intento di modellare la “conoscenza” per l'organizzazione di eventi che riguardano le maratone di film o serie tv.

La modellazione dell'ontologia è partita dall'estensione del *Event Ontology* (figura 2.1) che ha permesso di avere già una struttura solida da cui partire.

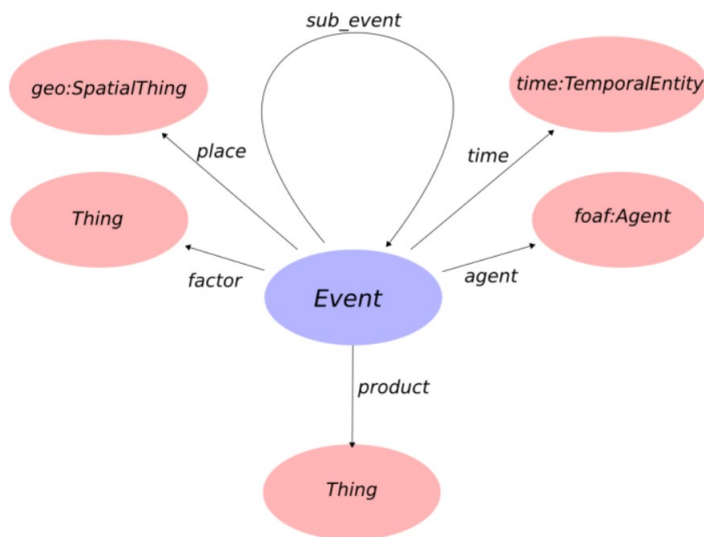
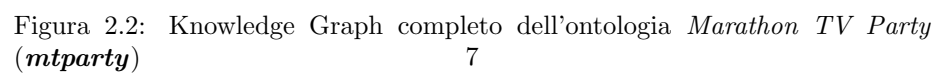


Figura 2.1: Knowledge Graph di *Event Ontology*

Di seguito è illustrato il Knowledge Graph dell'ontologia modellata.





Di seguito vengono descritte le varie componenti dell'ontologia modellata.

## 2.1 TVParty

**TVParty** rappresenta *l'evento della "maratona di film o serie tv" che è accaduto o che deve ancora accadere, in un luogo fisico oppure on-line in una certa data e ora*. La maratona viene ideata e organizzata da almeno un organizzatore (descritto nella sezione *Agents*).

Per la componente temporale è stata riutilizzata la classe **time:TemporalEntity** (parte della *Time Ontology*), la quale viene collegata a **TVParty** con l'object-property **startsTVPartyAt**.

```
1 mtparty:startsTVPartyAt rdf:type owl:ObjectProperty ;  
2                           rdfs:subPropertyOf time:hasTime ;  
3                           rdf:type owl:FunctionalProperty ;  
4                           rdfs:domain mtparty:TVParty .
```

Per la componente spaziale (descritta nel dettaglio nella sezione *Location*) sono state riutilizzate le classi **schema:Place** e **schema:VirtualLocation**, che fanno parte della *Schema.org Ontology*.

Per la lista di film o serie tv da proiettare durante l'evento è stata creata la classe **TVPartyShowSequence**, la quale viene approfondita nelle sezione *Movie/SerieTV*.

The screenshot shows a web interface for the class **TVParty**. At the top, there is a yellow header with the text "Description: TVParty". Below this, there are two main sections. The first section, "Equivalent To", shows a plus icon and a yellow circle with an equals sign, followed by the text "Event". The second section, "SubClass Of", shows a plus icon and a list of properties with yellow circles. The properties are: "hasBudget max 1 'Price specification'", "hasStatus exactly 1 TVPartyStatus", "organizer min 1 Organizer", "physicalPlace max 1 schema:Place", "startsTVPartyAt max 1 'Temporal entity'", "tvParty Show Sequence max 1 TVParty Show Sequence", and "virtualPlace max 1 schema:VirtualLocation".

Figura 2.3: Descrizione della classe **TVParty**

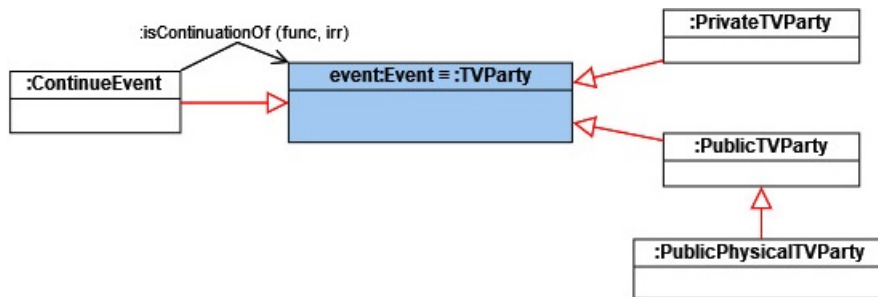


Figura 2.4: Gerarchia di **TVParty**

Come mostrato in figura 2.4 la classe **TVParty** è stata divisa in 2 differenti sottoclassi “disjoint” :

- **PrivateTVParty** è un **TVParty** che si potrebbe svolgere una residenza di un o più partecipanti.

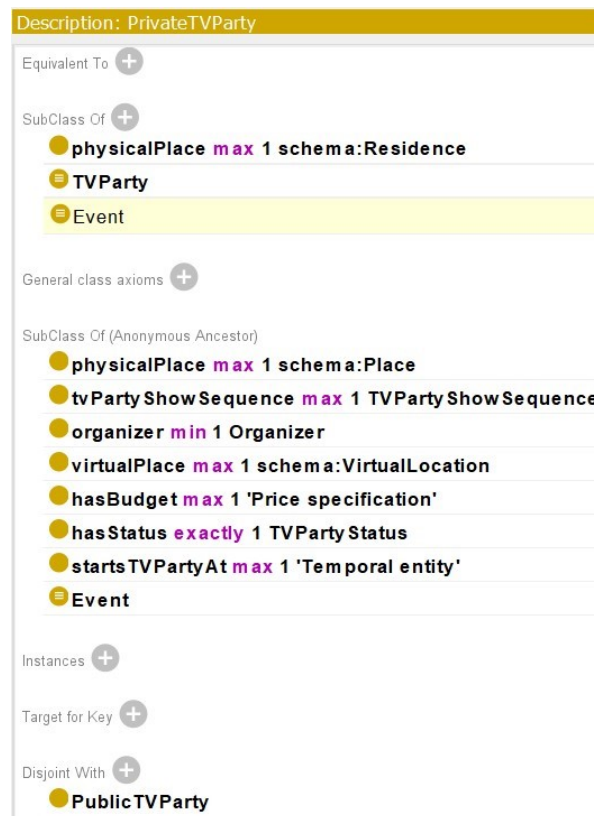


Figura 2.5: Descrizione della classe **PrivateTVParty**

- **PublicTVParty** è un **TVParty** che si potrebbe svolgere in un luogo pubblico (esempio: cinema, in luoghi dove è possibile il drive-in, ecc..).

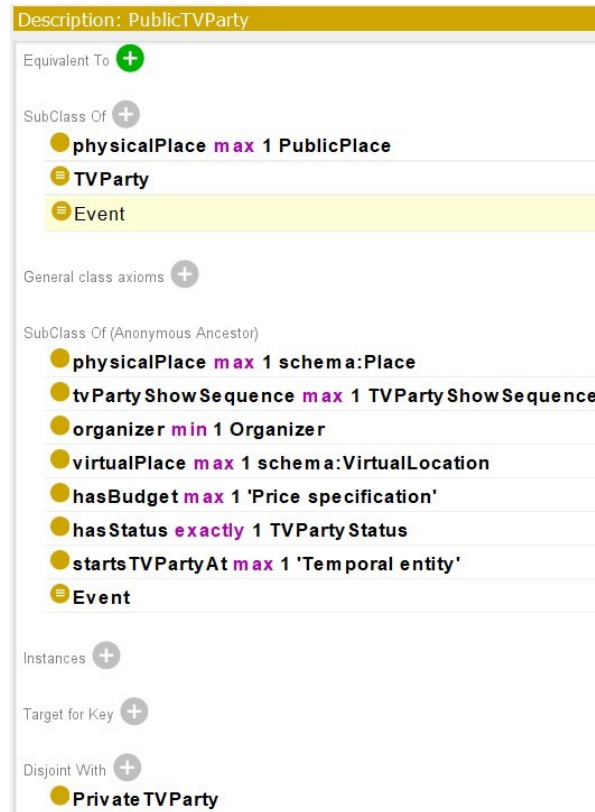


Figura 2.6: Descrizione della classe **PublicTVParty**

E' stata creata la classe **PublicPhysicalTVParty** con la seguente definizione :

*PublicTVParty and (physicalPlace exactly 1  
PublicPlace)*

E' stata creata anche la classe **ContinueEvent** che rappresenta *sempre un evento, ma è la continuazione di un altro evento* (la object-properties **isContinuationOf**, la quale è inversa di **event : sub\_event**).

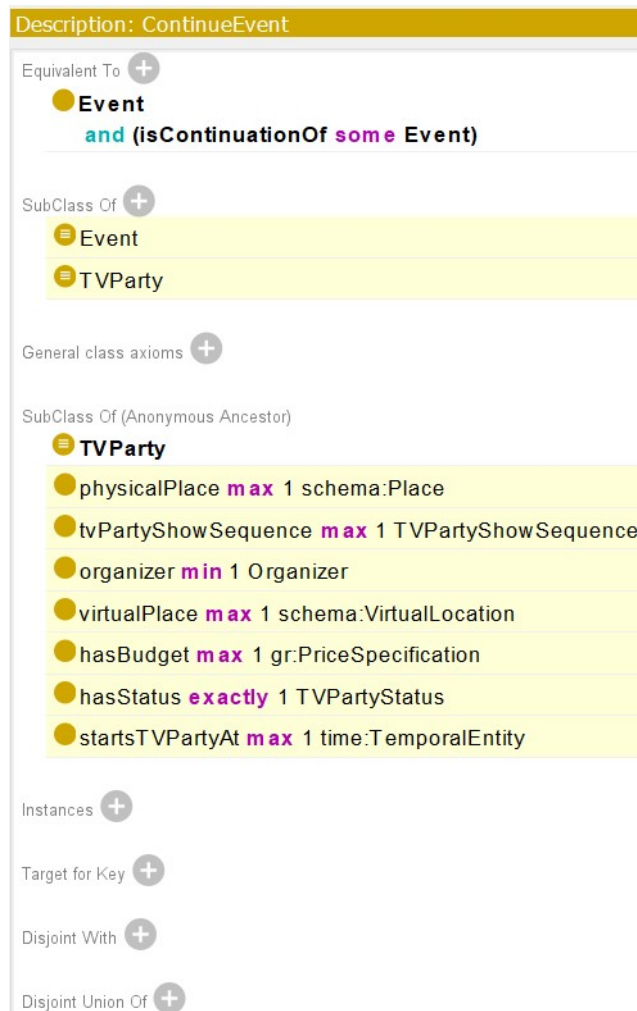


Figura 2.7: Descrizione della classe **ContinueEvent**

## TVPartyStatus

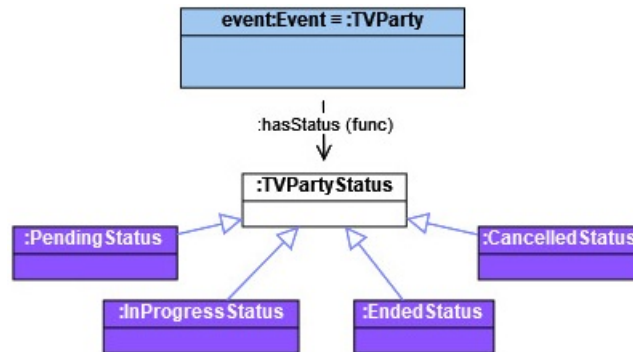


Figura 2.8: Knowledge Graph parziale della classe **TVPartyStatus**

**TVPartyStatus** (figura 2.8) rappresenta *lo stato del TV Party*. Per definirlo su un TV Party si utilizza l'object-property **hasStatus**.

---

```
1 mtparty:hasStatus rdf:type owl:ObjectProperty ,
2                       owl:FunctionalProperty ;
3                       rdfs:domain mtparty:TVParty ;
4                       rdfs:range mtparty:TVPartyStatus .
```

---

Sono state create le seguenti istanze:

- **PendingStatus**: stato che indica che TVParty deve ancora avvenire. Si dovrebbe mettere come istanza di default quando l'evento viene creato.
- **InProgressStatus**: stato che indica che TV Party si sta svolgendo in questo momento.
- **EndedStatus**: stato che indica che TV Party si è svolto ed è finito.
- **CancelledStatus**: stato che indica che TV Party non si è svolto come da programma.

## Budget

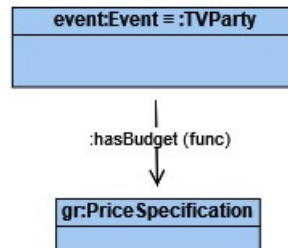


Figura 2.9: Knowledge Graph parziale del concetto di *Budget*

Un **TVParty** potrebbe avere bisogno di avere un budget da rispettare ossia un limite di denaro da spendere .

Questo concetto è stato modellato attraverso il riutilizzo della classe **gr:PriceSpecification** definita nel *Good Relations Ontology*. Essa permette di specificare un massimo attraverso la data-property **gr:hasMaxCurrencyValue** e il budget speso fino a quel momento attraverso la data-property **gr:hasCurrencyValue**.

## Reaction

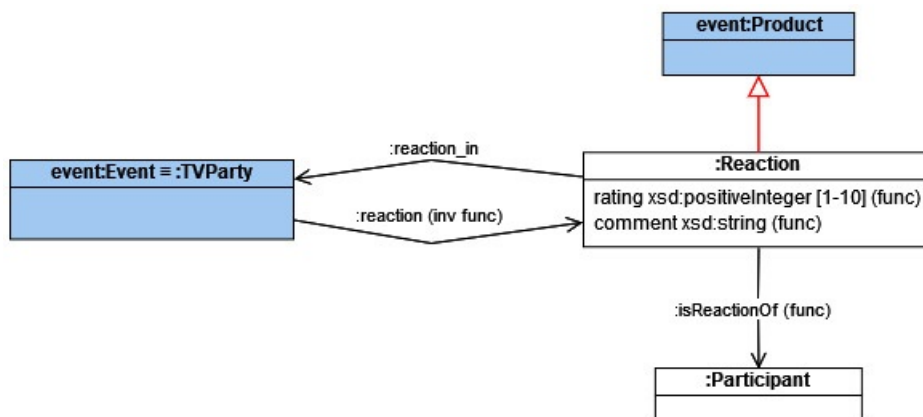


Figura 2.10: Knowledge Graph parziale della classe **Reaction**

**Reaction** (figura 2.10) rappresenta il concetto di “recensione” di un **TVParty** (object-property **reaction\_in**) che deve appartenere a un solo partecipante (object-property **isReactionOf**).

La **Reaction** è una sotto classe di **event:Product** perchè è un qualcosa che l’evento produce e quindi anche le object-property **reaction** e **reaction\_in** sono state modellate come sotto proprietà rispettivamente delle object-property **event:product** e **event:produced\_in**.

---

```

1 mtparty:reaction rdf:type owl:ObjectProperty ;
2   rdfs:subPropertyOf event:product ;
3   owl:inverseOf mtparty:reaction_in ;
4   rdf:type owl:InverseFunctionalProperty ;
5   rdfs:range mtparty:Reaction .
6
7 mtparty:reaction_in rdf:type owl:ObjectProperty .

```

---

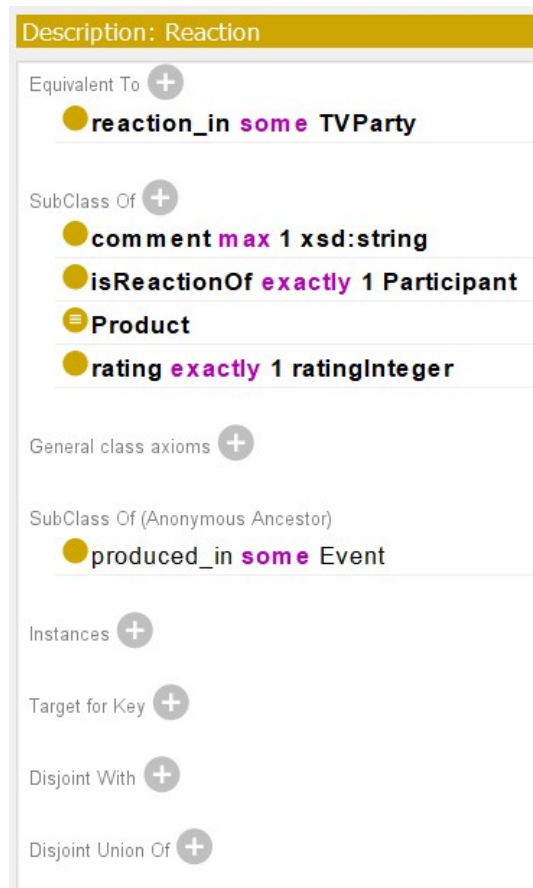


Figura 2.11: Descrizione della classe **Reaction**

Un partecipante può recensire l'evento tramite una valutazione numerica obbligatoria, rappresentata dalla data-property **rating**, mentre è libero di lasciare un commento rappresentato dalla data-property **comment**. Il range della data-property **rating** è ristretto a un intervallo da 1 a 10 tramite il datatype modellato: **ratingInteger** (figura 2.12).

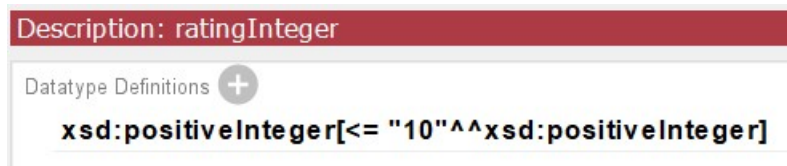


Figura 2.12: Definizione del Datatype **ratingInteger**



## 2.2 Movie/SerieTV

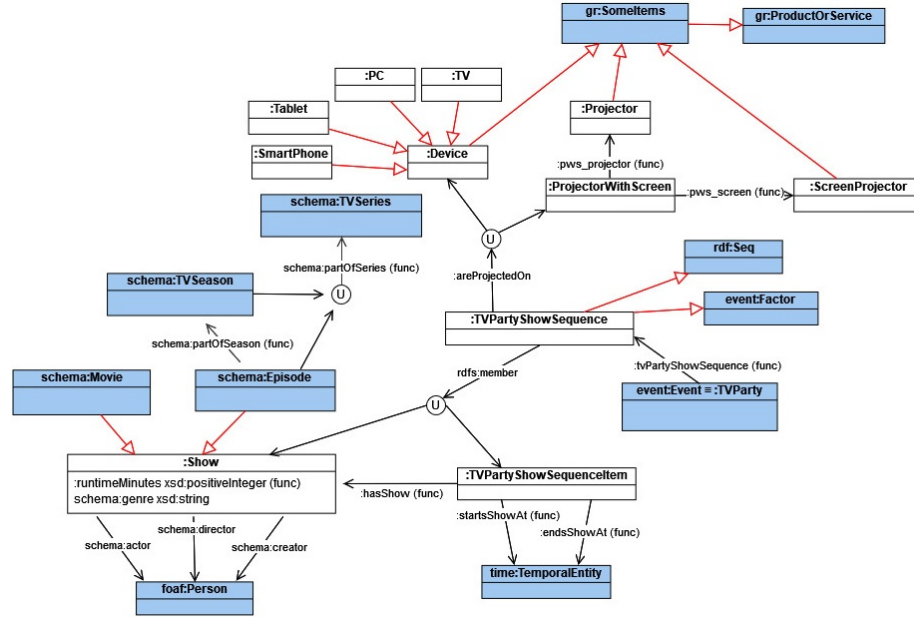


Figura 2.13: Knowledge Graph parziale della classe **TVPartyShowSequence**

Per definire gli show che devono essere proiettati al **TVParty** è stata creata la classe **TVPartyShowSequence** (figura 2.14) come sotto classe di **rdf:Seq** per rappresentare la sequenza ordinata dei film o di episodi di una serie tv.

Alla classe **TVPartyShowSequence** è possibile anche specificare dove sono state “visualizzati” gli show (object-property **areProjectedOn**) attraverso

- un **Device** (es: TV, PC, SmartPhone o Tablet)
- un proiettore (**Projector**) con associato uno schermo bianco (**ScreenProjector**).

Le classi **Device**, **Projector** e **ScreenProjector** sono state definite come sotto classi di **gr:SomeItems** del *Good Relations Ontology*.

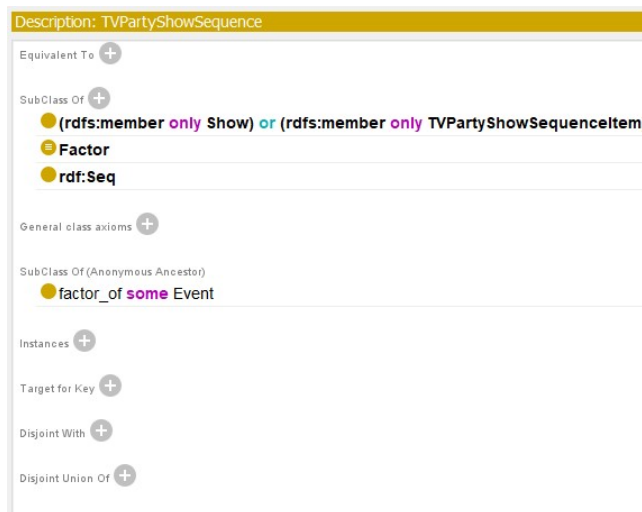


Figura 2.14: Descrizione della classe **TVPartyShowSequence**

La classe **TVPartyShowSequence** è anche sotto classe di **event:Factor** perché è usato come un *passive factor* nell'evento e quindi anche la object-property **tvPartyShowSequence** è stata modellata come sotto proprietà della object-property **event:factor**.

Per rappresentare i film e gli episodi di una serie tv sono state sfruttate le classi definite da *schema.org*: **schema:Movie** e **schema:Episode**. Per avere un concetto più generale si è creata la classe **Show** che rappresenta o un episodio o a un film.

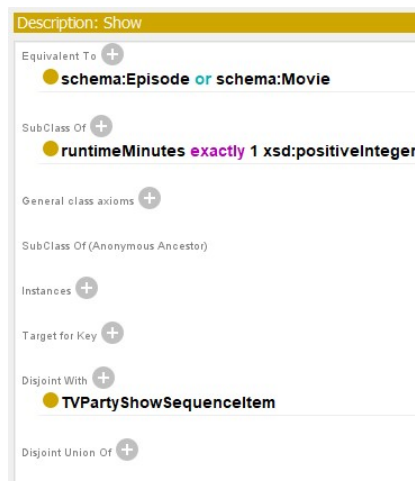


Figura 2.15: Descrizione della classe **Show**

Le proprietà dei film e degli episodi sono le stesse definite da *schema.org* ma nella figura 2.13 sono state riportate le più rilevanti: **schema:genre**, **schema:actor**, **schema:director** e **schema:creator**. E' stata aggiunta la data-property **runtimeMinutes** che permette di definire la durata (in minuti) di uno show.

Nel caso si volesse creare una scaletta con gli orari di proiezione dei vari show si è creata una classe **TVPartyShowSequenceItem** che possiede le object-property **startsShowAt** che permette di impostare l'orario di inizio e **endsShowAt** che permette di impostare l'orario di fine dello show.

---

```
1 mtparty:startsShowAt rdf:type owl:ObjectProperty ;
2                       rdfs:subPropertyOf time:hasTime ;
3                       rdf:type owl:FunctionalProperty ;
4                       rdfs:domain mtparty:TVPartyShowSequenceItem .
5
6 mtparty:endsShowAt rdf:type owl:ObjectProperty ;
7                   rdfs:subPropertyOf time:hasTime ;
8                   rdf:type owl:FunctionalProperty ;
9                   rdfs:domain mtparty:TVPartyShowSequenceItem .
```

---

## 2.3 Agents

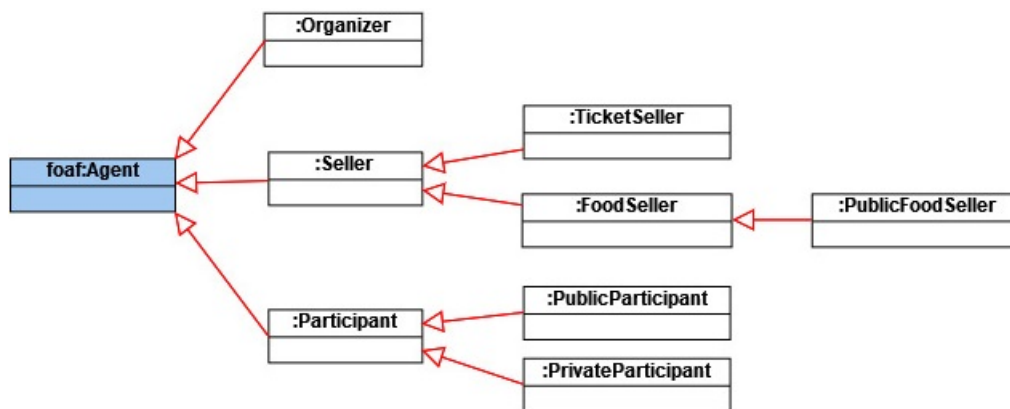


Figura 2.16: Gerarchia degli agenti coinvolti

Un **TVPparty** ha molti *active agent* coinvolti come mostrato in figura 2.16. Si è voluto essere più generali possibili sfruttando la classe della *FOAF Ontology* **foaf:Agent** per rappresentarli in modo tale da essere liberi di creare istanze che possono essere o persone (**foaf:Person**) o gruppi (**foaf:Group**) o organizzazioni (**foaf:Organization**).

La *Event Ontology* collega l'evento con gli agenti tramite l'object-property **event:agent**.

Nell'ontologia creata sono state modellate le seguenti sotto proprietà che come range hanno le rispettive classi che rappresentano gli agenti (descritte più avanti).



Figura 2.17: Sotto proprietà di **event:agent** e della sua inversa **event:agent\_in**

Di seguito sono illustrati in dettaglio i vari agenti modellati.

### 2.3.1 Organizer

Come detto in precedenza un **TVParty** deve avere almeno un organizzatore ossia *un agente che ha l'idea del evento e l'organizza*. In questa modellazione è rappresentato dalla classe **Organizer** così definita:

---

```
1 mtparty:Organizer rdf:type owl:Class ;
2   owl:equivalentClass [
3     rdf:type owl:Restriction ;
4     owl:onProperty mtparty:organizer_in ;
5     owl:someValuesFrom mtparty:TVParty
6   ] ;
7   rdfs:subClassOf foaf:Agent .
```

---

### 2.3.2 Participant

Un **TVParty** può avere agenti che rappresentano i partecipanti. In questa ontologia il concetto di “partecipante” è stato modellato con la classe **Participant**. La classe è stata specializzata in due sotto classi (**PublicParticipant** e **PrivateParticipant**) così da distinguere da chi è andato rispettivamente a un TV Party pubblico oppure a un TV Party privato .

---

```
1 mtparty:Participant rdf:type owl:Class ;
2   owl:equivalentClass [
3     rdf:type owl:Restriction ;
4     owl:onProperty mtparty:participant_in ;
5     owl:someValuesFrom mtparty:TVParty
6   ] ;
7   rdfs:subClassOf foaf:Agent .
8
9 mtparty:PublicParticipant rdf:type owl:Class ;
10  owl:equivalentClass [
11    rdf:type owl:Restriction ;
12    owl:onProperty mtparty:participant_in ;
13    owl:someValuesFrom mtparty:PublicTVParty
14  ] ;
15  rdfs:subClassOf
16    mtparty:Participant ,
17    [ rdf:type owl:Restriction ;
18      owl:onProperty mtparty:paysEntrance ;
19      owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
20      owl:onClass gr:PriceSpecification
21    ] .
22
23 mtparty:PrivateParticipant rdf:type owl:Class ;
24  owl:equivalentClass [
25    rdf:type owl:Restriction ;
26    owl:onProperty mtparty:participant_in ;
27    owl:someValuesFrom mtparty:PrivateTVParty
28  ] ;
29  rdfs:subClassOf mtparty:Participant .
```

---

Un partecipante per partecipare deve o pagare l'ingresso oppure una persona può compere uno o più biglietti per entrare al TVParty pubblico. Questo viene discusso nel dettaglio nella sotto sezione Ticket.

### 2.3.3 Seller

In questa ontologia è stata modellato anche il concetto di “venditore” (*un agente che “vende/offre” qualcosa*) rappresentato dalla classe **Seller**.

---

```

1 mtparty: Seller rdf:type owl:Class ;
2               owl:equivalentClass [
3                   rdf:type owl:Restriction ;
4                   owl:onProperty gr:offers ;
5                   owl:someValuesFrom gr:Offering
6               ] ;
7               rdfs:subClassOf foaf:Agent .

```

---

La condizione necessaria e sufficiente per essere considerato un **Seller** è quello di vendere “qualcosa” come dice la definizione così è stata sfruttata l’object-property **gr:offers** definita nel *GoodRelations Ontology*. Sono state create anche delle sotto proprietà in base a cosa vende:



Figura 2.18: Sotto proprietà dell’object-property **gr:offers**

La classe **Seller** è stata specializzata in 2 sottoclassi: **TicketSeller** e **FoodSeller**.

#### TicketSeller

Il **TicketSeller** rappresenta *l’ agente che vende o offre biglietti per un TV Party pubblico*.

Per il range dell’object-property **sellsTicket** è stata modellata la classe **TicketOffering** descritta in dettaglio nella sezione Selling Object.

---

```

1 mtparty: TicketSeller rdf:type owl:Class ;
2               owl:equivalentClass [
3                   rdf:type owl:Restriction ;
4                   owl:onProperty mtparty:ticketSeller_in ;
5                   owl:someValuesFrom mtparty:PublicTVParty
6               ] ;
7               rdfs:subClassOf mtparty: Seller.
8
9 mtparty: sellsTicket rdf:type owl:ObjectProperty ;
10               rdfs:subPropertyOf gr:offers ;
11               rdf:type owl:InverseFunctionalProperty ;
12               rdfs:domain mtparty: TicketSeller ;
13               rdfs:range mtparty: TicketOffering .

```

---

## FoodSeller

Il **FoodSeller** rappresenta *l' agente che vende o offre del cibo*. Per il range dell'object-property **sellsFood** è stata modellata la classe **FoodOffering** descritta in dettaglio nella sezione Selling Object.

Si è creata anche la classe **PublicFoodSeller** che rappresenta *l' agente che vende o offre del cibo in un TV Party pubblico che si svolge in un luogo fisico*.

---

```
1 mtparty:FoodSeller rdf:type owl:Class ;
2                       rdfs:subClassOf mtparty:Seller .
3
4 mtparty:PublicFoodSeller rdf:type owl:Class ;
5                           owl:equivalentClass [
6                               rdf:type owl:Restriction ;
7                               owl:onProperty mtparty:foodseller_in ;
8                               owl:someValuesFrom mtparty:PublicPhysicalTVParty
9                           ] ;
10                          rdfs:subClassOf mtparty:FoodSeller .
```

---

Questa modellazione del *foodseller* è stata ideata per fare in modo che un partecipante posso comprare del cibo da un venditore esterno al TV Party (soprattutto in un TV Party privato). Questo concetto viene spiegato nel dettaglio nella sezione Selling Object.

## 2.4 Location

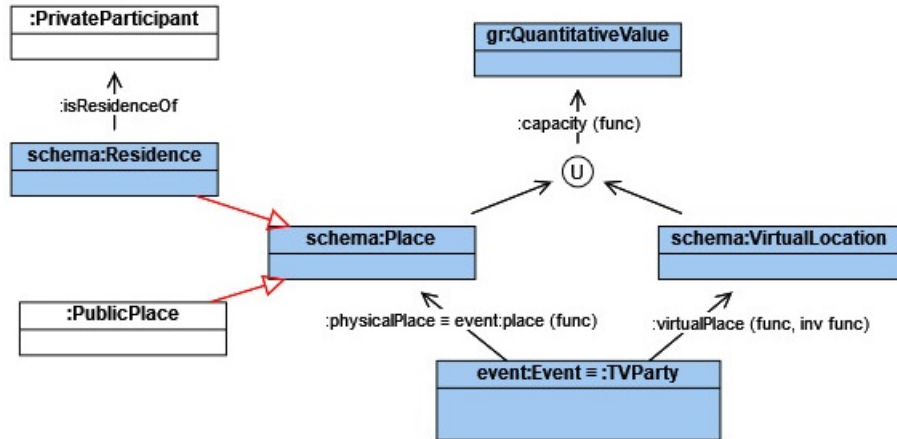


Figura 2.19: Knowledge Graph parziale della *Location*

Un **TVParty** può avere luogo in un luogo fisico rappresentato dalla classe **schema:Place** oppure on-line rappresentato dalla classe **schema:VirtualLocation**.

Il luogo è collegato a **TVParty** attraverso una delle seguenti object-property: **physicalPlace** o **virtualPlace**.

```

1 mtparty:physicalPlace rdf:type owl:ObjectProperty ;
2                       rdfs:subPropertyOf mtparty:location ;
3                       rdf:type owl:FunctionalProperty ;
4                       rdfs:range schema:Place ;
5                       owl:propertyDisjointWith mtparty:virtualPlace .
6
7 mtparty:virtualPlace  rdf:type owl:ObjectProperty ;
8                       rdfs:subPropertyOf mtparty:location ;
9                       rdf:type owl:FunctionalProperty ,
10                              owl:InverseFunctionalProperty ;
11                       rdfs:range schema:VirtualLocation .
12
13 mtparty:location rdf:type owl:ObjectProperty ,
14                   owl:FunctionalProperty ;
15                   rdfs:domain mtparty:TVParty .

```

Per un dato luogo che sia fisico o virtuale si può anche specificare quante persone può contenere attraverso l'object-property **capacity**, la quale ha come range la classe **gr:QuantitativeValue** definita nella *GoodRelations Ontology*.

Non si è creata una sotto classe dove l'unità di misura è "C62" (unità di misura che rappresenta "l'unità") perché si potrebbe voler definire delle proprie unità di misure che rappresentano le "persone".



## schema:Place

**schema:Place** è caratterizzato da un indirizzo (object-property **schema:address**) oppure dalle coordinate geografiche: latitudine (data-property **geo:lat**) e longitudine (data-property **geo:long**).

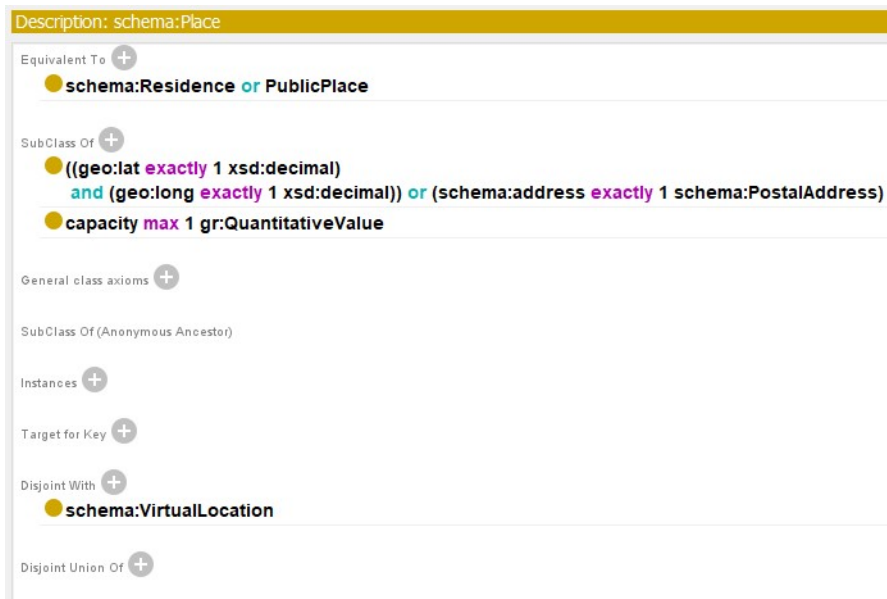


Figura 2.20: Descrizione della *schema:Place*

**schema:Place** è stato suddiviso in 2 sotto classi: **PublicPlace** e **schema:Residence**. **schema:Residence** rappresenta *il luogo dove vive almeno uno dei partecipanti* espresso attraverso l'object-property **isResidenceOf**. Invece **PublicPlace** rappresenta tutti quei *luoghi che non sono schema:Residence*, per esempio luoghi dove è possibile il drive-in oppure cinema ecc.

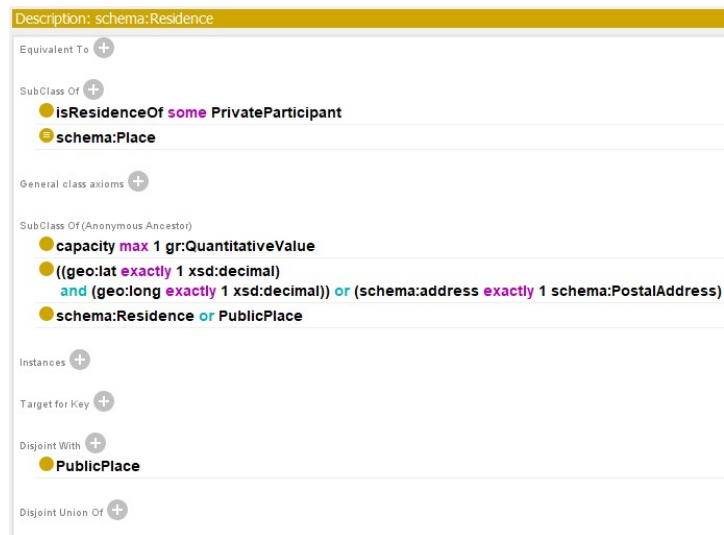


Figura 2.21: Descrizione della *schema:Residence*

## schema:VirtualLocation

**schema:VirtualLocation** deve avere un indirizzo virtuale ossia un URL, questo è possibile attraverso la data-property **schema:url** che ha come range il datatype **xsd:anyURI**.

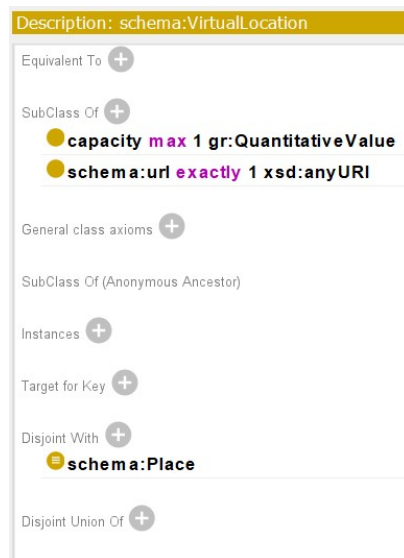


Figura 2.22: Descrizione della *schema:VirtualLocation*

## 2.5 Selling Object

La componente della compra-vendita in questa ontologia è stata modellata attraverso un'altra ontologia ossia *GoodRelations Ontology*. Le classi principali riusate sono **gr:ProductOrService** e **gr:Offering** che hanno permesso di non rimodellare da zero rispettivamente il concetto di *prodotto* e di *offerta*.

Di seguito vengono illustrate le modellazione della compra-vendita dei biglietti per entrare a un TV Party pubblico e la compra-vendita di cibo.

### 2.5.1 Ticket

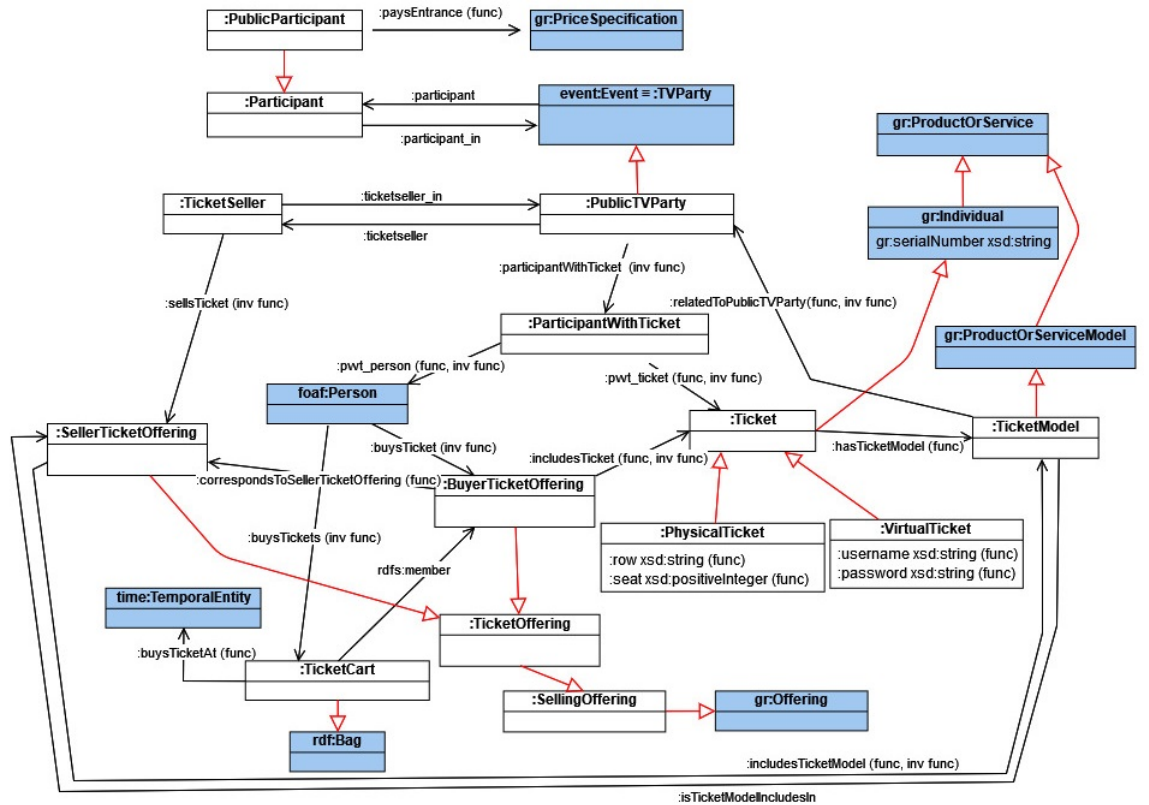


Figura 2.23: Knowledge Graph parziale riguardante *Ticket*

In un **PublicTVParty** una persona (**foaf:Person**) può prenotare uno o più posti attraverso l'acquisto di biglietti oppure può pagare direttamente l'entrata e trovare dei posti liberi.

## Pagamento all'entrata

Per il pagamento all'entrata è stata riusata la classe **gr:PriceSpecification**, la quale viene direttamente collegata all'agente **PublicParticipant** attraverso l'object-property **paysEntrance**.

Di seguito è riportato un esempio di utilizzo.

```
1 :tvparty a mtparty:PublicTVParty
2       mtparty:seller [ a mtparty:Seller ;
3       gr:offers [
4       a gr:Offering ;
5       gr:hasPriceSpecification [
6       a gr:UnitPriceSpecification ;
7       gr:hasCurrency "EUR" ;
8       gr:hasCurrencyValue "5"^^xsd:float ;
9       gr:hasUnitOfMeasurement "C62"
10      ]
11      rdfs:comment "Tv Party Entrance Offering" .
12    ]
13  ];
14  mtparty:participant :pGroup .
15
16 :pGroup a foaf:Group ,
17         mtparty:PublicParticipant ;
18  mtparty:paysEntrance [
19  a gr:PriceSpecification ;
20  gr:hasCurrency "EUR" ;
21  gr:hasCurrencyValue "25"^^xsd:float ;
22  ] ;
23  rdfs:comment "Group of 5 persons" .
```

## Compra-vendita di biglietti

La classe **Ticket** (sotto classe di **gr:Individual**) rappresenta *il biglietto con cui una persona può partecipare al TV Party pubblico*. Questo collegamento avviene tramite l'object-property **participantWithTicket** che ha come range la classe **ParticipantWithTicket** definita come segue:

*(pwt\_person exactly 1 foaf:Person) and  
(pwt\_ticket exactly 1 Ticket)*

L'object-property **pwt\_person** permette di capire chi ha utilizzato il biglietto (**pwt\_ticket some Ticket**).

Grazie all'utilizzo della classe **ParticipantWithTicket**, si può dedurre che se esiste la “catena” di object-property illustrata nella figura 2.24, al TV Party pubblico partecipa la persona che utilizza il biglietto.



Figura 2.24: Property Chain dell'object-property participant

In base al tipo di **TVParty** (in un luogo fisico oppure on-line) si sono modellate 2 tipi di biglietti (sottoclassi di **Ticket**): **PhysicalTicket** e **VirtualTicket**.

- **PhysicalTicket** rappresenta *il biglietto fisico dove è indicato il posto prenotato (row e seat)*.

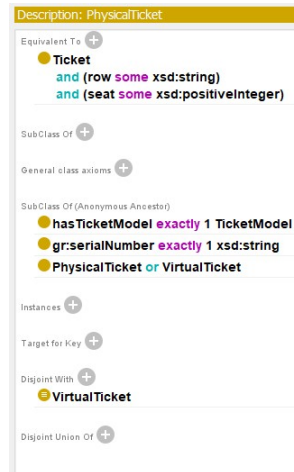


Figura 2.25: Descrizione della classe PhysicalTicket

- **VirtualTicket** rappresenta *il biglietto virtuale dove sono indicate le credenziali di accesso (username e password)*.

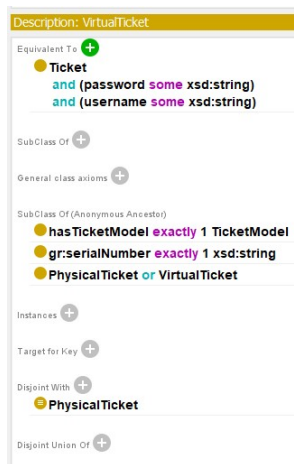


Figura 2.26: Descrizione della classe VirtualTicket

Non esiste un vincolo che afferma che una persona che acquista il biglietto non possa cederlo a qualcun altro.

Per l'acquisto di un biglietto è stata seguita la logica del *GoodRelations Ontology* ossia un business entity (`gr:BusinessEntity`) può “offrire” (`gr:offers`) una certa offerta (`gr:Offering`) mentre un'altra può “comprare” (`gr:seeks`) l'offerta relativa a uno o più prodotti/servizi (`gr:ProductOrService`).

Sono state create due tipi di classe relative all'offerta dei biglietti:

- **BuyerTicketOffering** rappresenta *l'offerta relativa a un biglietto (**Ticket**) che può comprare una persona.*

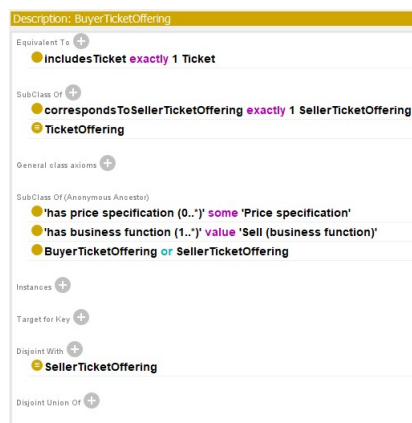


Figura 2.27: Descrizione della classe BuyerTicketOffering

- **SellerTicketOffering** rappresenta *l'offerta relativa al TV Party pubblico (**TicketModel** sotto classe di `gr:ProductOrServiceModel`) che può vendere un **TicketSeller**.*

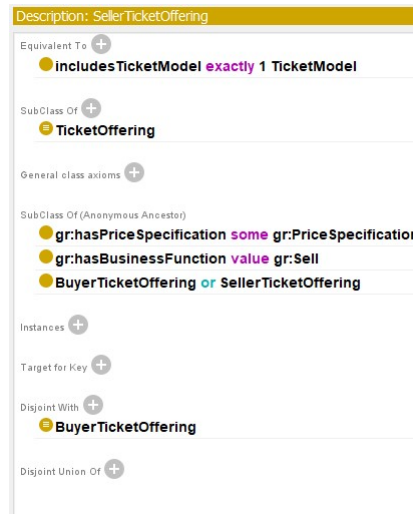


Figura 2.28: Descrizione della classe SellerTicketOffering

L'offerta **BuyerTicketOffering** è collegata con l'offerta **SellerTicketOffering** attraverso l'object-property **correspondsToSellerTicketOffering**.

```
1 mtparty:correspondsToSellerTicketOffering
2     rdf:type owl:ObjectProperty ,
3         owl:FunctionalProperty ;
4     rdfs:domain mtparty:BuyerTicketOffering ;
5     rdfs:range mtparty:SellerTicketOffering .
```

Acquisto dei biglietti è definito tramite le object-property **buysTicket** oppure **buysTickets**.

```
1 mtparty:buysTicket rdf:type owl:ObjectProperty ;
2     rdfs:subPropertyOf gr:seeks ;
3     owl:inverseOf mtparty:isBoughtTicketBy ;
4     rdf:type owl:InverseFunctionalProperty ;
5     rdfs:domain foaf:Person ;
6     rdfs:range mtparty:BuyerTicketOffering .
7
8 mtparty:buysTickets rdf:type owl:ObjectProperty ;
9     owl:inverseOf mtparty:isTicketCartOf ;
10    rdf:type owl:InverseFunctionalProperty ;
11    rdfs:domain foaf:Person ;
12    rdfs:range mtparty:TicketCart .
```

Nell'ontologia è stata modellata la classe **TicketCart** (sotto classe di **rdf:Bag**) che rappresenta *il carrello dei biglietti acquistati* e permette anche di sapere quando è avvenuto l'acquisto tramite l'object-property **buysTicketAt**.

Di seguito è riportato un esempio di utilizzo di alcune componenti modellate che riguardano l'acquisto e l'utilizzo dei biglietti in un TVParty pubblico.

```
1 :ticketPrice a gr:UnitPriceSpecification ;
2     gr:hasCurrency "EUR" ;
3     gr:hasCurrencyValue "8.0"^^xsd:float ;
4     gr:hasUnitOfMeasurement "C62" ;
5     rdfs:comment "Full Price of Tv Party".
6
7 :ticketReducedPrice a gr:UnitPriceSpecification ;
8     gr:hasCurrency "EUR" ;
9     gr:hasCurrencyValue "5.0"^^xsd:float ;
10    gr:hasUnitOfMeasurement "C62" ;
11    rdfs:comment "Reduced Price of Tv Party".
12
13 :ticketModelOfTVParty01 a mtparty:TicketModel ;
14     mtparty:relatedToPublicTVParty :tvparty01
15     rdfs:comment "Ticket Model of TV Party".
16
17 :ticketSellerOffering01 a mtparty:SellerTicketOffering ;
18     gr:acceptedPaymentMethods gr:Cash ,
19                               gr:MasterCard ,
20                               gr:PayPal ;
21     gr:hasPriceSpecification :ticketPrice;
22     mtparty:includesTicketModel :ticketModelOfTVParty01 ;
23     rdfs:comment "Seller Offering number 1 about TV Party".
24
25 :ticketSellerOffering02 a mtparty:SellerTicketOffering ;
26     gr:acceptedPaymentMethods gr:Cash;
27     gr:hasPriceSpecification :ticketReducedPrice;
28     mtparty:includesTicketModel :ticketModelOfTVParty01 ;
29     rdfs:comment "Seller Offering number 2 about TV Party".
30
31 :ticketseller01 a mtparty:TicketSeller ;
32     mtparty:sellsTicket :ticketSellerOffering01,
33                        :ticketSellerOffering02 ;
34     rdfs:comment "Ticket seller of TV Party".
35
36 :ticket01 a mtparty:PhysicalTicket ;
37     gr:serialNumber "AA1122WW" ;
38     mtparty:hasTicketModel :ticketModelOfTVParty01 ;
39     mtparty:row "A" ;
40     mtparty:seat 1 ;
41     rdfs:comment "Ticket row A seat 1 of TV Party".
42
43
44 :ticket02 a mtparty:PhysicalTicket ;
45     gr:serialNumber "AA1122WZ" ;
46     mtparty:hasTicketModel :ticketModelOfTVParty01 ;
47     mtparty:row "A" ;
48     mtparty:seat 2 ;
49     rdfs:comment "Ticket row A seat 2 of TV Party".
50
51
52
53
```



```

54 :agent00Cart a mtparty:TicketCart ;
55     rdfs:member [
56         a mtparty:BuyerTicketOffering ;
57         mtparty:includesTicket :ticket01;
58         gr:hasPriceSpecification :ticketReducedPrice;
59         mtparty:correspondsToSellerTicketOffering :ticketSellerOffering02
60     ] ,
61     [
62         a mtparty:BuyerTicketOffering ;
63         mtparty:includesTicket :ticket02;
64         gr:hasPriceSpecification :ticketPrice, [
65             a gr:PaymentChargeSpecification ;
66             gr:appliesToPaymentMethod gr:PayPal
67         ];
68         mtparty:correspondsToSellerTicketOffering :ticketSellerOffering01
69     ] ;
70     rdfs:comment "Ticket cart of agent00".
71
72 :agent00 a foaf:Person ;
73     mtparty:buysTickets :agent00Cart;
74     rdfs:comment "person that buys tickets".
75
76
77 :tvparty01 a mtparty:PublicTVParty ;
78     mtparty:ticketseller :ticketseller01 ;
79     mtparty:participantWithTicket [
80         a mtparty:ParticipantWithTicket ;
81         mtparty:pwt_person [rdf:type foaf:Person] ;
82         mtparty:pwt_ticket :ticket01
83     ];
84     mtparty:participantWithTicket [
85         a mtparty:ParticipantWithTicket ;
86         mtparty:pwt_person :agent00 ;
87         mtparty:pwt_ticket :ticket02
88     ];
89     rdfs:comment "Marathon TV Party".

```

---

## 2.5.2 Food

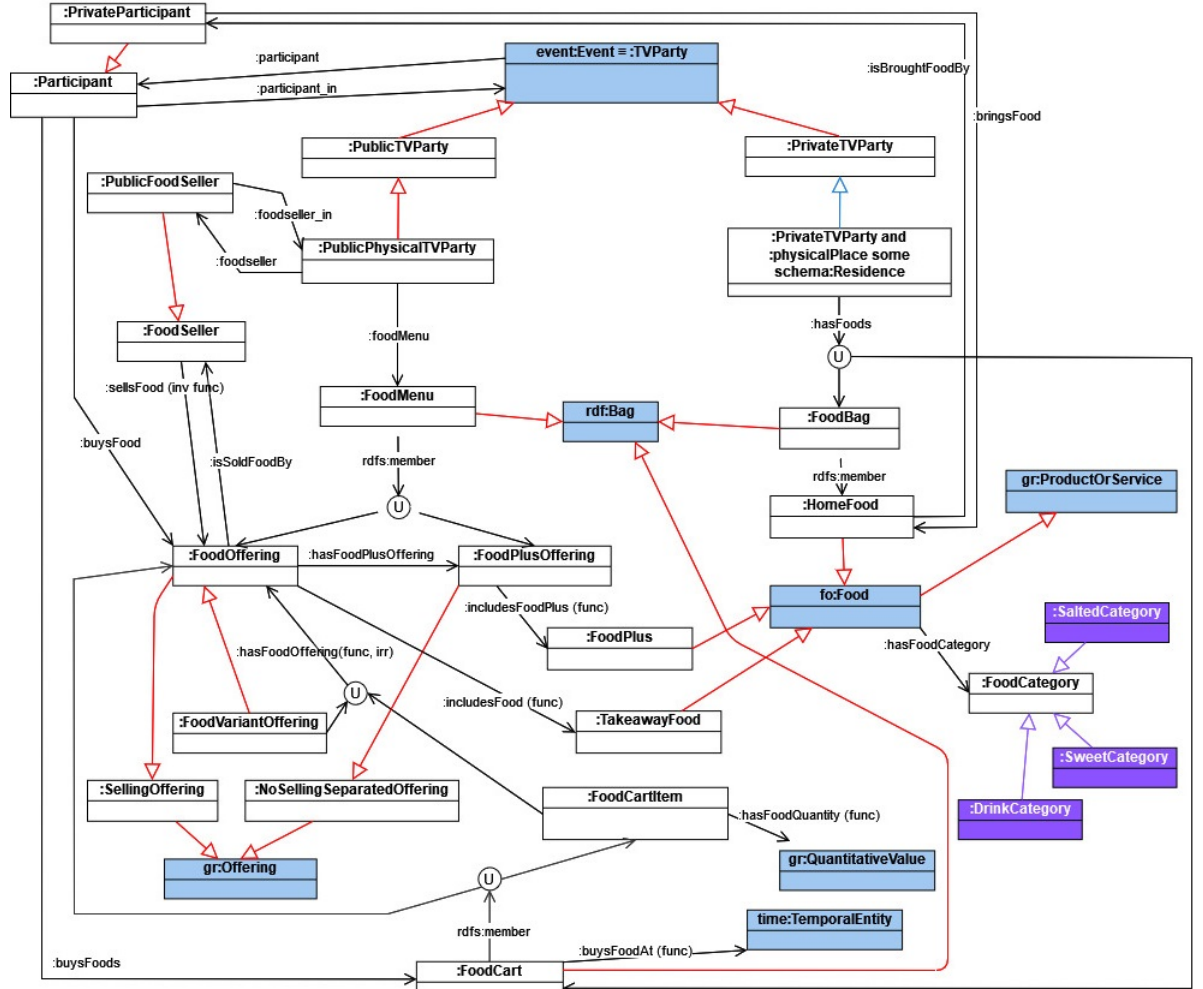


Figura 2.29: Knowledge Graph parziale riguardante *Food*

Per il concetto di cibo è stata riutilizzata la classe **fo:Food** definita nel *Food Ontology*, che come si nota in figura 2.29 è una sotto classe di **gr:ProductOrService** del *GoodRelations Ontology*.

Sono state create delle sotto classi di **fo:Food** “disjoint”: **HomeFood**, **TakeawayFood** e **FoodPlus**.

- **HomeFood** può rappresentare *cibo fatto in casa oppure un concetto più generale di cibo ossia senza dover specificare chi l’ha venduto ma semplicemente chi l’ha portato al TV Party*. Questa classe può essere usata solo in un TV Party privato .

- **TakeawayFood** rappresenta *il cibo comprato da un partecipante e venduto da un venditore (**FoodSeller**) attraverso l'offerta corrispondente (**FoodOffering**)*.
- **FoodPlus** rappresenta *il cibo che può essere aggiunto a un **TakeawayFood***. Questo è possibile attraverso l'object-property **hasFoodPlusOffering** sull'offerta **FoodOffering**. Questo tipo di cibo non può essere venduto separatamente, ciò è rappresentato dalla restrizione

```
gr:hasBusinessFunction value :SellNotSeparated
```

sull'offerta relativa (**FoodPlusOffering**).

**SellNotSeparated** è un'istanza di **gr:BusinessFunction** che è stata modellata.

E' stata creata anche la classe **FoodVariantOffering** (figura 2.30) che rappresenta una **FoodOffering** che ha in più una **FoodPlusOffering**.

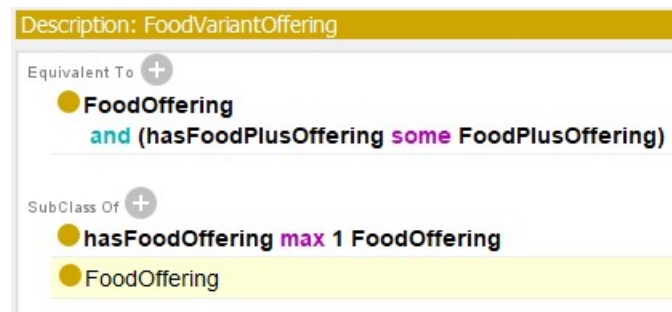


Figura 2.30: Descrizione della classe **FoodVariantOffering**

Acquisto di cibo è definito tramite le object-property **buysFood** oppure **buysFoods**.

```

1 mtparty:buysFood rdf:type owl:ObjectProperty ;
2   rdfs:subPropertyOf gr:seeks ;
3   owl:inverseOf mtparty:isBoughtFoodBy ;
4   rdfs:domain mtparty:Participant ;
5   rdfs:range mtparty:FoodOffering ;
6   owl:propertyChainAxiom ( mtparty:buysFoods
7                               rdfs:member
8                               ) .
9
10
```

```

11 mtparty:buysFoods rdf:type owl:ObjectProperty ;
12     owl:inverseOf mtparty:isFoodCartOf ;
13     rdfs:domain mtparty:Participant ;
14     rdfs:range mtparty:FoodCart .

```

---

Nell'ontologia è stata modellata la classe **FoodCart** (sotto classe di **rdf:Bag**) che rappresenta *il carrello del cibo acquistato* e permette anche di sapere quando è avvenuto l'acquisto tramite l'object-property **buysFoodAt**.

Il **FoodCart** può contenere direttamente l'offerta **FoodOffering** oppure la classe **FoodCartItem** che rappresenta *un FoodOffering in una certa quantità*.

Di seguito si illustra un esempio di utilizzo della classe FoodCartItem:

```

1 @prefix pto: <http://www.productontology.org/id/> .
2
3 :bottleWaterOffering a mtparty:FoodOffering ;
4     gr:hasEligibleQuantity [
5         a gr:QuantitativeValue ;
6         gr:hasUnitOfMeasurement "LTR" ;
7         gr:hasValue "0.5"^^xsd:float ;
8     ] ;
9     gr:hasPriceSpecification [
10         a gr:UnitPriceSpecification ;
11         gr:hasCurrency "EUR" ;
12         gr:hasCurrencyValue "1"^^xsd:float ;
13         gr:hasUnitOfMeasurement "C62" ;
14     ] ;
15     mtparty:includesFood [
16         a pto:Bottled_water,
17         mtparty:TakeawayFood ;
18     ] ;
19     rdfs:label "Bottle of Water by 1/2 litre Offering" .
20
21 :cartItem a mtparty:FoodCartItem ;
22     mtparty:hasFoodQuantity [
23         a gr:QuantitativeValue ;
24         gr:hasUnitOfMeasurement "C62" ;
25         gr:hasValue 5 ;
26     ] ;
27     mtparty:hasFoodOffering :waterOffering .
28
29 :cart a mtparty:FoodCart ;
30     rdfs:member :cartItem ;
31     rdfs:comment "Cart with 5 bottle of water by 1/2 litre".

```

---

Il **PublicPhysicalTVParty** mette ha disposizione menu con le offerte del cibo (**FoodMenu**) offerto dai venditori associati (**PublicFoodSeller**).

Di seguito si illustra un esempio di utilizzo delle varie componenti per quanto riguarda un TV Party pubblico :

```
1 @prefix pto: <http://www.productontology.org/id/> .
2
3 :pFs a mtparty:PublicFoodSeller ;
4     mtparty:sellsFood :popcornSmallOffering.
5
6 :popcorn a pto:Popcorn,
7     mtparty:TakeawayFood ;
8     mtparty:hasFoodCategory mtparty:SaltedCategory ;
9     rdfs:label "Popcorn" .
10
11 :popcornSmallOffering a mtparty:FoodOffering ;
12     gr:acceptedPaymentMethods gr:Cash ;
13     gr:hasEligibleQuantity [
14         a gr:QuantitativeValue ;
15         gr:hasUnitOfMeasurement "GRM" ;
16         gr:hasValue 50 ;
17     ] ;
18     gr:hasPriceSpecification [
19         a gr:UnitPriceSpecification ;
20         gr:hasCurrency "EUR" ;
21         gr:hasCurrencyValue "1.5"^^xsd:float ;
22         gr:hasUnitOfMeasurement "C62" ;
23     ] ;
24     mtparty:includesFood :popcorn ;
25     rdfs:label "Popcorn (small) Offering" .
26
27 :butter a pto:Butter,
28     mtparty:FoodPlus ;
29     mtparty:hasFoodCategory mtparty:SaltedCategory .
30
31 :butterOffering a mtparty:FoodPlusOffering ;
32     gr:hasEligibleQuantity [
33         a gr:QuantitativeValue ;
34         gr:hasUnitOfMeasurement "GRM" ;
35         gr:hasValue 5 ;
36     ] ;
37     gr:hasPriceSpecification [
38         a gr:UnitPriceSpecification ;
39         gr:hasCurrency "EUR" ;
40         gr:hasCurrencyValue "0.5"^^xsd:float ;
41         gr:hasUnitOfMeasurement "C62" ;
42     ] ;
43     mtparty:includesFoodPlus :butter ;
44     rdfs:label "Butter Offering" .
45
46 :popcornSmallWithButterOffering a mtparty:FoodVariantOffering ;
47     mtparty:hasFoodOffering :popcornSmallOffering ;
48     mtparty:hasFoodPlusOffering :butterOffering ;
49     rdfs:label "Popcorn (small) with butter Offering" .
50
51
52
53
54
```

```

55 :puTVParty a mtparty:PublicTVParty ;
56     :foodMenu [
57         a mtparty:FoodMenu ;
58         rdfs:member :popcornSmallOffering;
59         rdfs:member :butterOffering;
60     ];
61     :foodseller :pFs;
62     :participant :p.
63
64 :p a mtparty:PublicParticipant ;
65     mtparty:buysFoods [
66         a mtparty:Foodcart ;
67         rdfs:member :popcornSmallOffering;
68         rdfs:member [
69             a mtparty:FoodCartItem;
70             mtparty:hasFoodOffering :popcornSmallWithButterOffering ;
71             mtparty:hasFoodQuantity [
72                 a gr:QuantitativeValue ;
73                 gr:hasUnitOfMeasurement "C62" ;
74                 gr:hasValue 2 ;
75             ]
76         ] ;
77         rdfs:label "Food cart with 1 popcorn small and 2 popcorn small with butter."
78     ] .

```

---

In un **PrivateTVParty** and **physicalPlace** some **Residence** i partecipanti possono portare qualcosa da mangiare da casa (attraverso la classe **FoodBag**) oppure ordinare d'asporto durante il TV party (attraverso la classe **FoodCart**).

Di seguito un esempio di utilizzo di **FoodCart** nel accezione di *ordinare d'asporto durante il TV party*.

---

```

1 @prefix pto: <http://www.productontology.org/id/> .
2
3 :pizzaOffering a mtparty:FoodOffering ;
4                 mtparty:includesFood :pizza ;
5                 mtparty:isSoldFoodBy [
6                 a mtparty:FoodSeller ;
7                 rdfs:label "Pizzeria 360"
8                 ];
9                 gr:hasPriceSpecification [
10                a gr:UnitPriceSpecification;
11                gr:hasCurrency "EUR" ;
12                gr:hasCurrencyValue "5.50"^^xsd:float ;
13                gr:hasUnitOfMeasurement "C62";
14                ].
15
16 :pizza a pto:Pizza,
17         mtparty:TakeawayFood ;
18         gr:name "Pizza margherita" ;
19         gr:weight [
20         a gr:QuantitativeValue;
21         gr:hasValue "350"^^xsd:int;
22         gr:hasUnitOfMeasurement "GRM";
23         ];
24         fo:containsIngredient [a fo:Ingredient; rdfs:label "pomodoro"] ,
25                               [a fo:Ingredient; rdfs:label "mozzarella"],
26                               [a fo:Ingredient; rdfs:label "basilico"].
27
28
29 :cart a mtparty:FoodCart;
30     rdfs:member [
31     a mtparty:FoodCartItem ;
32     mtparty:hasFoodOffering :pizzaOffering ;
33     mtparty:hasFoodQuantity [
34     a gr:QuantitativeValue;
35     gr:hasValue "5"^^xsd:int;
36     gr:hasUnitOfMeasurement "C62";
37     ];
38     rdfs:comment "Cart with 5 pizze. So 5*5.50 = 27.5 EUR"
39     ] .
40
41 :pTVParty a mtparty:PrivateTVParty ;
42         mtparty:hasFoods :cart .

```

---

La classe **FoodBag** rappresenta 2 concetti semantici differenti :

- *quale cibo uno o più partecipanti hanno portato al TV Party privato.*  
Di seguito se ne illustra un esempio:

---

```
1 @prefix pto: <http://www.productontology.org/id/> .
2
3 :prTVParty a mtparty:PrivateTVParty ;
4             mtparty:foodBag [
5                 a mtparty:FoodBag ;
6                 rdfs:member :homeFood;
7             ] ;
8             mtparty:participant :particiant01,
9                                 :particiant02.
10
11 :homeFood a mtparty:HomeFood ,
12            pto:Beer ;
13            mtparty:isBroughtFoodBy :particiant01,
14                                    :particiant02 ;
15            gr:quantitativeProductOrServiceProperty [
16                a gr:QuantitativeValueFloat ;
17                gr:hasValue "3"^^xsd:float;
18                gr:hasUnitOfMeasurement "LTR"^^xsd:string
19            ] .
20
21 :particiant01 a foaf:Person ,
22              mtparty:PrivateParticipant.
23
24 :particiant02 a foaf:Group ,
25              mtparty:PrivateParticipant ;
26              rdfs:comment "Group of 3 persons".
```

---

- *quale cibo portare al TV Party privato.*  
Di seguito se ne illustra un esempio:

---

```
1 @prefix pto: <http://www.productontology.org/id/> .
2
3 :prTVParty a mtparty:PrivateTVParty ;
4             mtparty:foodBag [
5                 a mtparty:FoodBag ;
6                 rdfs:member [
7                     a gr:SomeItems ,
8                     pto:Beer ;
9                     gr:quantitativeProductOrServiceProperty [
10                         a gr:QuantitativeValueFloat ;
11                         gr:hasValue "6"^^xsd:float;
12                         gr:hasUnitOfMeasurement "LTR"^^xsd:string
13                     ]
14                 ]
15             ];
16             mtparty:participant :particiant01.
17
18
19
20
21
22
```



```
23 :particiant01 a foaf:Person ,
24     mtparty:PrivateParticipant ;
25     mtparty:bringsFood [
26         a mtparty:HomeFood ,
27         pto:Beer ;
28         gr:quantitativeProductOrServiceProperty [
29             a gr:QuantitativeValueFloat ;
30             gr:hasValue "3"^^xsd:float;
31             gr:hasUnitOfMeasurement "LTR"^^xsd:string
32         ]
33     ].
```

---

## Capitolo 3

# SWRL

Per permettere al reasoner di inferire maggiori informazioni riguardanti l'ontologia sono state realizzate alcune regole Semantic Web Rule Language (SWRL).

### 3.1 Rule 1 e Rule 2

---

```
1 PrivateTVParty(?x) ^ ContinueEvent(?y) ^ isContinuationOf(?y, ?x)
2 -> PrivateTVParty(?y)
3
4 PublicTVParty(?x) ^ ContinueEvent(?y) ^ isContinuationOf(?y, ?x)
5 -> PublicTVParty(?y)
```

---

Queste due regole permettono di capire se un evento `ContinueEvent` è un evento privato (`PrivateTVParty`) oppure pubblico (`PublicTVParty`).

### 3.2 Rule 3

---

```
1 PublicPhysicalTVParty(?e) ^ FoodMenu(?fm) ^ foodMenu(?e, ?fm) ^
2 FoodOffering(?f) ^ rdfs:member(?fm, ?f) ^
3 foaf:Agent(?a) ^ sellsFood(?f, ?a)
4 -> foodseller(?e, ?a)
```

---

Questa regola permette di capire se un venditore di cibo (`FoodSeller`) ha “partecipato” a un evento pubblico in un luogo fisico.

### 3.3 Rule 4 e Rule 5

---

```
1 TVParty(?e) ^ TVPartyShowSequence(?s) ^ tvPartyShowSequence(?e, ?s) ^
2 TVPartyShowSequenceItem(?i) ^ rdf:_1(?s, ?i) ^
3 time:TemporalEntity(?t) ^ startsTVPartyAt(?e, ?t)
4 -> startsShowAt(?i, ?t)
5
```

---

```

6 TVParty(?e) ^ TVPartyShowSequence(?s) ^ tvPartyShowSequence(?e, ?s) ^
7 TVPartyShowSequenceItem(?i) ^ rdf:_1(?s, ?i) ^
8 time:TemporalEntity(?t) ^ startsShowAt(?i, ?t)
9 -> startsTVPartyAt(?e, ?t)

```

---

Queste due regole permettono di assegnare la data e l'orario di inizio del primo show che equivale l'inizio del TV Party.

### 3.4 Rule 6

```

1 PrivateTVParty(?e) ^ Organizer(?o) ^ organizer(?e, ?o)
2 -> participant(?e, ?o)

```

---

Questa regola permette di affermare che l'organizzatore di un TV Party privato partecipante all'evento.

## Capitolo 4

# Queries

Per separare la “componente terminologica” (T-Box) dalla “componente assertiva” (A-Box) è stata creata una nuova ontologia con prefisso `sdb`:

`<http://www.semanticweb.org/marica/ontologies/marathonparty/sample-db/>`  
(file *marathonparty-sample-db.ttl* contenuta nella cartella *datastore*), la quale importa l’ontologia modellata, *mtparty*.

Dopo aver completato il popolamento dell’ontologia è stato possibile formulare e testare diverse tipologie di query attraverso il linguaggio Simple Protocol and RDF Query Language (SPARQL) utilizzando come IDE, *Stardog Studio*. E’ stato utilizzato *Stardog* perchè ha un supporto più esteso di SPARQL, ad esempio supporta i blank node, funzioni COALESCE (restituisce il valore della prima espressione che valuta senza errori), sequence path , alternative paths.

Le query descritte necessitano dell’utilizzo del Reasoner.

## Prefix

Per evitare di ripetere i **PREFIX** che sono stati utilizzati per lo sviluppo delle query SPARQL, vengono mostrati solamente una volta qui di seguito.

---

```
1 PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
2 PREFIX schema: <http://schema.org/>
3 PREFIX gr: <http://purl.org/goodrelations/v1#>
4 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
5 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
6 PREFIX mtparty: <http://www.semanticweb.org/marica/ontologies/marathonparty/>
7 PREFIX sdb: <http://www.semanticweb.org/marica/ontologies/marathonparty/sample-db/>
```

---

## 4.1 Lista dei biglietti venduti validi per un TV Party pubblico

Questa query permette di visualizzare i biglietti venduti da un venditore autorizzato per un determinato TVParty pubblico (i.e. sdb:tvparty01).

```
1 SELECT ?TVParty ?ticket ?ticketseller (COALESCE(?val, "NO_VALID") AS ?validity)
2 WHERE {
3   ?ticketseller mtparty:sellsTicket ?sto.
4   ?tm mtparty:relatedToPublicTVParty ?TVParty.
5   ?sto mtparty:includesTicketModel ?tm.
6   ?bto mtparty:correspondsToSellerTicketOffering ?sto;
7     mtparty:includesTicket ?ticket.
8   ?ticket mtparty:hasTicketModel ?tm.
9
10  OPTIONAL {
11    ?ticketseller mtparty:ticketseller_in ?TVParty.
12    BIND(( "VALID" ) AS ?val).
13  }
14
15  FILTER (?TVParty = sdb:tvparty01)
16 }
```

### Risultato

TVParty	ticket	ticketseller	validity
sdb:tvparty01	sdb:ticket01	sdb:sellerT01	"VALID"
sdb:tvparty01	sdb:ticket02	sdb:sellerT01	"VALID"
sdb:tvparty01	sdb:ticket03	sdb:sellerT02	"NO VALID"

## 4.2 Incassi dei TV Party Pubblici

Questa query permette di visualizzare gli incassi totali (?proceedsTotal) di tutti i TVParty pubblici.

Gli incassi sono suddivisi in:

- incasso derivato dalle offerte di entrata (?proceedsEntrance)
- incasso derivato dalle offerte dei biglietti venduti (?proceedsTicket)
- incasso derivato dalle offerte del cibo venduto (?proceedsFood)

---

```
1 SELECT ?TVParty
2     (SUM(COALESCE(?entrancepcv,0)) AS ?proceedsEntrance)
3     (SUM(COALESCE(?btopcv,0)) AS ?proceedsTicket)
4     (SUM(COALESCE(?bfopcv,0)) AS ?proceedsFood)
5     (SUM(COALESCE(?partial_proceeds,0)) AS ?proceedsTotal)
6 WHERE {
7     ?TVParty a mtparty:PublicTVParty.
8     {
9         ?TVParty mtparty:seller [
10             gr:offers [
11                 gr:hasPriceSpecification [
12                     gr:hasCurrencyValue ?entrancepcv
13                 ]
14             ]
15         ].
16         ?agent mtparty:participant_in ?TVParty;
17             mtparty:paysEntrance [gr:hasCurrencyValue ?entrancepcv].
18
19         BIND ((?entrancepcv) AS ?partial_proceeds).
20     }
21     UNION {
22         ?agent mtparty:buysTicket|(mtparty:buysTickets/rdfs:member) ?bto.
23         ?bto mtparty:includesTicket ?t ;
24             mtparty:correspondsToSellerTicketOffering ?sto ;
25             gr:hasPriceSpecification [gr:hasCurrencyValue ?btopcv].
26         ?sto mtparty:includesTicketModel ?tm .
27         ?ts mtparty:sellsTicket ?sto ;
28             mtparty:ticketseller_in ?TVParty .
29         ?tm mtparty:relatedToPublicTVParty ?TVParty .
30         ?t mtparty:hasTicketModel ?tm .
31
32         BIND ((?btopcv) AS ?partial_proceeds).
33     }
34     UNION {
35         ?agent mtparty:participant_in ?TVParty .
36         ?TVParty mtparty:foodMenu [rdfs:member ?bfo];
37             mtparty:foodseller [mtparty:sellsFood ?bfo].
38         {
39             ?agent mtparty:buysFood ?bfo .
40             ?bfo gr:hasPriceSpecification [gr:hasCurrencyValue ?bfopcv].
41         }
42     }
43     UNION {
44
```

```

45         ?agent mtparty:buysFood ?bfoI .
46         ?bfoI mtparty:hasFoodQuantity [gr:hasValue ?bfoIQV];
47             mtparty:hasFoodOffering ?bfo.
48         ?bfo gr:hasPriceSpecification [gr:hasCurrencyValue ?partial_bfopcv].
49         BIND ((?bfoIQV * ?partial_bfopcv) AS ?bfopcv).
50     }
51
52     BIND ((?bfopcv) AS ?partial_proceeds).
53 }
54 }
55 GROUP BY ?TVParty

```

---

## Risultato

TVParty	proceedsEntrance	proceedsTicket	proceedsFood	proceedsTotal
sdb:tvparty01	0	13.0	4.0	17.0
sdb:tvparty02	16.0	0	0	16.0

### 4.3 Genere (movie/serie tv) più frequente

Questa query permette di capire qual è il genere più frequente per dei TV Party.

```
1 SELECT ?genre (COUNT(?genre) AS ?frequenctGenreShow)
2 WHERE {
3   {
4     ?show a schema:Movie.
5     ?TVParty mtparty:tvPartyShowSequence [
6       rdfs:member|(rdfs:member/mtparty:hasShow) ?show
7     ].
8     ?show schema:genre ?genre.
9   }
10  UNION {
11    ?show a schema:Episode .
12    ?TVParty mtparty:tvPartyShowSequence [
13      rdfs:member|(rdfs:member/mtparty:hasShow) ?show
14    ].
15    { ?show schema:genre ?genre. }
16    UNION
17    {
18      ?show schema:partOfSeason ?season .
19      { ?season schema:genre ?genre. }
20      UNION
21      {
22        ?season schema:partOfSeries ?serie .
23        ?serie schema:genre ?genre .
24      }
25    }
26  }
27 }
28 GROUP BY ?genre
29 ORDER BY DESC(?frequenctGenreShow) ASC(?genre)
```

#### Risultato

genre	frequenctGenreShow
"crime"	6
"horror"	6
"drama"	3
"fantasy"	3
"mystery"	3
"sci-fi"	3
"action"	2
"thriller"	2



## 4.4 Lista dei luoghi più gettonati per TV Party

Questa query permette di visualizzare i luoghi più frequenti come location di un TVParty.

---

```
1 SELECT ?place ?placeLabel (COUNT(?place) AS ?frequencyTVParty)
2 WHERE {
3   {
4     ?TVParty mtparty:physicalPlace ?place.
5     OPTIONAL {
6       ?place rdfs:label ?pl.
7       BIND (STR(?pl) AS ?placeLabel)
8     }
9   }
10  UNION
11  {
12    ?TVParty mtparty:virtualPlace [ schema:url ?pl ].
13    BIND (REPLACE(STR(?pl), "^(http[s]?://[^:/]+)(.*)$" , "$1") AS ?place).
14  }
15 }
16 }
17 GROUP BY ?place ?placeLabel
18 ORDER BY DESC (?frequencyTVParty)
```

---

### Risultato

place	placeLabel	frequencyTVParty
sdb:placePR01	"Residence of agent08"	2
"https://stream.example.com"		2
sdb:placePP01	"Cinedream Faenza"	1
sdb:placePP02	"Drive-in Imola"	1

## 4.5 Rating medio di ogni TVParty

Questa query permette di capire quale TVParty ha riscosso più successo attraverso la media delle valutazioni (?avgRating).

```
1 SELECT ?TVParty
2       (COALESCE(AVG(?rating), 0) AS ?avgRating)
3       (COUNT(?reaction) AS ?numReaction)
4 WHERE {
5     ?TVParty a mtparty:TVParty.
6     OPTIONAL{
7         ?TVParty mtparty:reaction ?reaction;
8                 mtparty:participant ?participant.
9         ?reaction mtparty:isReactionOf ?participant;
10                mtparty:rating ?rating.
11     }
12 }
13 GROUP BY ?TVParty
14 ORDER BY DESC(?avgRating)
```

### Risultato

TVParty	avgRating	numReaction
sdb:tvparty04	10	1
sdb:tvparty01	8.333333333333334	3
sdb:tvparty06	0	0
sdb:tvparty05	0	0
sdb:tvparty03	0	0
sdb:tvparty02	0	0

## 4.6 Lista ordinata degli show in ogni TV Party

Questa query permette di visualizzare, per ogni TV Party, la lista ordinata completa di ogni show proiettati.

```
1 SELECT ?TVParty ?position ?showLabel ?seasonLabel ?serieLabel
2 WHERE {
3   FILTER (STRSTARTS(STR(?membership), STR(rdf:_)))
4   ?TVParty mtparty:tvPartyShowSequence ?tvPartyShowSeq.
5   {
6     ?tvPartyShowSeq ?membership ?showId.
7   }
8   UNION
9   {
10    ?tvPartyShowSeq ?membership ?showItem.
11    ?showItem mtparty:hasShow ?showId.
12  }
13  OPTIONAL { ?showId rdfs:label ?showLabel . }
14
15  OPTIONAL {
16    {
17      ?showId schema:partOfSeason ?seasonId .
18      ?seasonId schema:partOfSeries ?serieId .
19    } UNION {
20      ?showId schema:partOfSeries ?serieId .
21    }
22    OPTIONAL { ?seasonId rdfs:label ?seasonLabel . }
23    OPTIONAL { ?serieId rdfs:label ?serieLabel . }
24  }
25  BIND((xsd:integer(REPLACE(STR(?membership), STR(rdf:_), ""))) AS ?position)
26 }
27 ORDER BY ?TVParty ?position
```

### Risultato

TVParty	position	showLabel	seasonLabel	serieLabel
sdb:tvparty01	1	"Underworld"		
sdb:tvparty01	2	"Underworld: Evolution"		
sdb:tvparty01	3	"Underworld: Rise of the Lycans"		
sdb:tvparty04	1	"Fringe 1x01 Pilot"	"Fringe season 1"	"Fringe"
sdb:tvparty04	2	"Fringe 1x02 The Same Old Story"	"Fringe season 1"	"Fringe"
sdb:tvparty04	3	"Fringe 1x03 The Ghost Network"	"Fringe season 1"	"Fringe"

# Conclusioni

L'ontologia *mtparty* ha lo scopo di sviluppare un modello per la rappresentazione dell'organizzazione e della gestione di un o più eventi di visioni di film o serie tv .

La modellazione di questa ontologia ha permesso di approfondire e di esporre le conoscenze in ambito di Web Semantico.  
In particolare si è posta l'attenzione sui seguenti elementi:

**Resource Description Framework (RDF)** è lo strumento base proposto da W3C per la codifica, lo scambio e il riutilizzo di metadati strutturati e consente l'interoperabilità semantica tra applicazioni che condividono le informazioni sul Web. Inoltre descrive i concetti e le relazioni su di essi attraverso l'introduzione di triple (soggetto-predicato-oggetto), e consente la costruzione di query basate su triple patterns, congiunzioni logiche, disgiunzioni logiche, e pattern opzionali.

RDF è stato utilizzato per modellare le classi e le proprietà che compongono l'ontologia.

**Resource Description Framework Schema (RDFS)** è un insieme di classi e proprietà RDF costituente un'estensione del vocabolario base di RDF.  
**Web Ontology Language (OWL)** è un linguaggio di markup per rappresentare esplicitamente significato e semantica di termini con vocabolari e relazioni tra gli stessi. Esistono varie versioni del linguaggio, che differiscono molto tra di loro.

RDFS e OWL sono stati utilizzati per arricchire l'espressività delle affermazioni modellate nell'ontologia RDF.

**SWRL** è un linguaggio proposto per il Semantic Web che può essere utilizzato per esprimere regole e logica, combinando OWL DL o OWL Lite con un sottoinsieme del Rule Markup Language (a sua volta un sottoinsieme di Datalog).

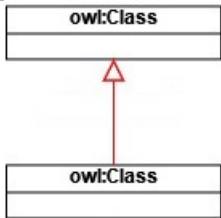
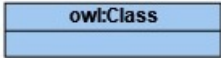
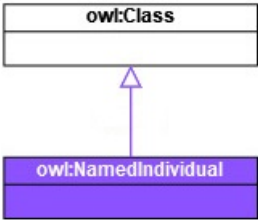
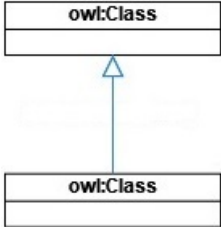
SWRL è stato utilizzato per realizzare regole che aggiungano ulteriori informazioni non definibili attraverso OWL.

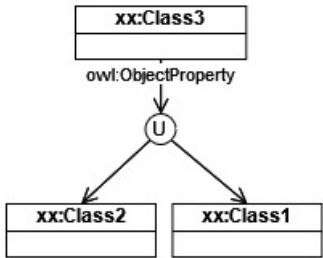
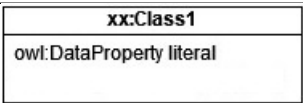
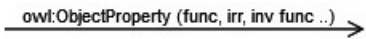
**SPARQL** è un linguaggio di interrogazione per dati rappresentati tramite il RDF. SPARQL trova i risultati dell'interrogazione attraverso il pattern mat-

*ching all'interno del grafo che deve soddisfare ciò che è stato specificato nella clausola **WHERE** della query.*

SPARQL è stato utilizzato per realizzare le interrogazioni per estrapolare informazioni riguardo l'ontologia realizzata.

# Terminologia del KG

Simbolo	Significato
	<p> rappresenta il costrutto di rdf:  <i>rdf:subClassOf</i> . </p>
	<p> rappresenta una classe esterna  ossia una classe definita in  un'altra ontologia. </p>
	<p> rappresenta il costrutto di rdf:  <i>rdf:type</i> su un individuo. </p>
	<p> la classe in basso non è definita  nell'ontologia ma è comunque  una sottoclasse della classe più  in alto. </p>
$xx:Class1 \equiv yy:Class2$	<p> rappresenta il costrutto di owl:  <i>owl:equivalentClass</i>. </p>

	<p> rappresenta il costrutto di owl:  <i>owl:unionOf</i> . </p>
	<p> rappresenta <i>owl:DataProperty</i> . </p>
	<p> rappresenta <i>owl:ObjectProperty</i>  con alcune caratteristiche: </p> <ul style="list-style-type: none"> <li>• <i>func</i> : proprietà funzionale</li> <li>• <i>inv func</i>: proprietà inversamente proporzionale</li> <li>• <i>irr</i> : proprietà irriflessiva</li> </ul>

# Acronimi

**FOAF** Friend Of A Friend. 4, 59

**KG** Knowledge Graph. 1, 53

**OWL** Web Ontology Language. 51

**RDF** Resource Description Framework. 51

**RDFS** Resource Description Framework Schema. 51

**SPARQL** Simple Protocol and RDF Query Language. 43, 51, 52

**SWRL** Semantic Web Rule Language. 41, 51

**Turtle** Terse RDF Triple Language. 4



# Glossario

**event:Factor** Qualsiasi cosa usato come fattore in un evento. 17

**event:Product** Qualsiasi cosa prodotto da un evento. 14

**foaf:Agent** è la classe degli agenti (ad es. persona, gruppo, software o artefatto fisico); cose che fanno cose.  
Ha come sotto classi **foaf:Person**, **foaf:Organization** e **foaf:Group**. 19

**foaf:Group** rappresenta una raccolta di singoli agenti (e può essa stessa svolgere il ruolo di agente, cioè qualcosa che può eseguire azioni). 19, 56

**foaf:Organization** rappresenta una sorta di agente corrispondente a istituzioni sociali come aziende, società ecc.. 19, 56

**foaf:Person** rappresenta le persone. Non si fanno distinzioni sul fatto che siano vivi, morti, reali o immaginari. 19, 26, 56

**gr:BusinessEntity** rappresenta l'agente legale che fa (o cerca) una particolare offerta. Questo può essere un ente legale o una persona. Per maggiori informazioni: <http://www.heppnetz.de/ontologies/goodrelations/v1#BusinessEntity>. 29, 56, 57

**gr:BusinessFunction** specifica il tipo di attività o accesso (ovvero il pacchetto di diritti) offerto da gr:BusinessEntity su gr:ProductOrService tramite gr:Offering. Tipici sono la vendita, il noleggio o il leasing, la manutenzione o la riparazione, la produzione/produzione, il riciclaggio/smaltimento, l'ingegneria/costruzione o l'installazione.. 34, 57

**gr:Individual** rappresenta un prodotto reale o un'istanza di servizio, cioè un singolo oggetto o azione identificabile che crea qualche aumento di utilità (in senso economico) per l'individuo che possiede o utilizza questo stesso oggetto (prodotto) o per l'individuo nel cui favorire questa stessa azione (servizio). Prodotti o servizi sono tipi di beni in senso economico. 27, 57

**gr:Offering** rappresenta l'annuncio pubblico, non necessariamente vincolante, non necessariamente esclusivo, da parte di una `gr:BusinessEntity` per fornire (o ricercare) una determinata `gr:BusinessFunction` per un determinato ad un determinato target di riferimento. Un'offerta è specificata dal tipo di prodotto o servizio o pacchetto a cui si riferisce, quale funzione aziendale viene offerta (vendita, affitto, ...) e un insieme di proprietà commerciali. . 26, 29, 56

**gr:PriceSpecification** rappresenta la superclasse di tutte le specifiche di prezzo. Per maggiori informazioni: <http://www.heppnetz.de/ontologies/goodrelations/v1#PriceSpecification>. 13, 27

**gr:ProductOrService** è la superclasse di tutte le classi che descrivono i tipi di prodotti o servizi, per natura o scopo. Esempi di tali sottoclassi sono "televisore", "aspirapolvere", ecc. Un'istanza di questa classe può essere un prodotto o un servizio effettivo (`gr:Individual`), un'istanza segnaposto per istanze sconosciute di una merce prodotta in serie (`gr:SomeItems`) o una specifica del modello/prototipo (`gr:ProductOrServiceModel`). In caso di dubbio, usa `gr:SomeItems`. . 26, 29, 33, 56, 57

**gr:ProductOrServiceModel** rappresenta un'entità intangibile che specifica alcune caratteristiche di un gruppo di prodotti simili, solitamente prodotti in serie, nel senso di un prototipo. Nel caso di prodotti prodotti in serie, esiste una relazione `gr:hasMakeAndModel` tra il prodotto o servizio effettivo (`gr:Individual` o `gr:SomeItems`) e il prototipo (`gr:ProductOrServiceModel`). 29, 57

**gr:QuantitativeValue** è un intervallo numerico che rappresenta l'intervallo di una certa proprietà in termini di limite inferiore e superiore per un particolare `gr:ProductOrService`. Deve essere interpretato in combinazione con la rispettiva unità di misura. La maggior parte dei valori quantitativi sono intervalli anche se in pratica sono spesso trattati come un singolo valore.. 23

**gr:SomeItems** rappresenta un placeholder instance per istanze sconosciute di una merce prodotta in serie. Questo viene utilizzato come soluzione alternativa dal punto di vista computazionale per tali istanze che non sono esposte individualmente sul Web ma semplicemente dichiarate di esistere (ovvero, che sono quantificate esistenzialmente).. 16, 57

**rdf:Bag** è usata convenzionalmente per indicare a un lettore umano che il contenitore è destinato ad essere non ordinato. 30, 35

**rdf:Seq** è usata convenzionalmente per indicare a un lettore umano che l'ordinamento numerico delle proprietà di appartenenza al contenitore deve essere significativo . 16

**Reasoner** è un software in grado di svolgere dei ragionamenti su delle basi di conoscenza adeguatamente formalizzate. Il ragionamento, in questo caso, è inteso come la capacità di elaborare la base di conoscenza secondo alcune regole, in modo da validare ed analizzare la base di conoscenza stessa. Le possibilità del ragionatore dipendono strettamente dal linguaggio adoperato per formalizzare la base di conoscenza. 43

**schema:Place** entità che hanno un'estensione fisica in qualche modo fissa. 8, 23, 24

**schema:Residence** il luogo in cui vive una persona. 24, 25

**schema:VirtualLocation** un luogo online o virtuale per partecipare agli eventi. Sebbene un luogo virtuale possa essere utilizzato come luogo di un evento, i luoghi virtuali non devono essere confusi con i luoghi fisici nel mondo reale. 8, 23, 25

**time:TemporalEntity** rappresenta un intervallo temporale o un istante di tempo. 8

# Sitografia

- [1] *Dublic Core*. URL: <https://www.dublincore.org/>.
- [2] *Event Ontology*. URL: <https://motools.sourceforge.net/event/event.html>.
- [3] *Friend Of A Friend Ontology*. URL: <http://xmlns.com/foaf/0.1/>.
- [4] *Food Ontology*. URL: <https://github.com/ailabitmo/food-ontology>.
- [5] *Good Relations Ontology*. URL: <http://www.heppnetz.de/ontologies/goodrelations/v1.html>.
- [6] *RDF Schema*. URL: [https://it.wikipedia.org/wiki/RDF\\_Schemae](https://it.wikipedia.org/wiki/RDF_Schemae).
- [7] *Resource Description Framework*. URL: [https://it.wikipedia.org/wiki/Resource\\_Description\\_Framework](https://it.wikipedia.org/wiki/Resource_Description_Framework).
- [8] *Schema Ontology*. URL: <https://schema.org/>.
- [9] *Semantic Web Rule Language*. URL: [https://en.wikipedia.org/wiki/Semantic\\_Web\\_Rule\\_Language](https://en.wikipedia.org/wiki/Semantic_Web_Rule_Language).
- [10] *SPARQL*. URL: <https://it.wikipedia.org/wiki/SPARQL>.
- [11] *SPARQL Stardog*. URL: <https://docs.stardog.com/getting-started-series/getting-started-1>.
- [12] *Time Ontology*. URL: <https://www.w3.org/TR/owl-time/>.
- [13] *Web Ontology Language*. URL: [https://it.wikipedia.org/wiki/Web\\_Ontology\\_Language](https://it.wikipedia.org/wiki/Web_Ontology_Language).