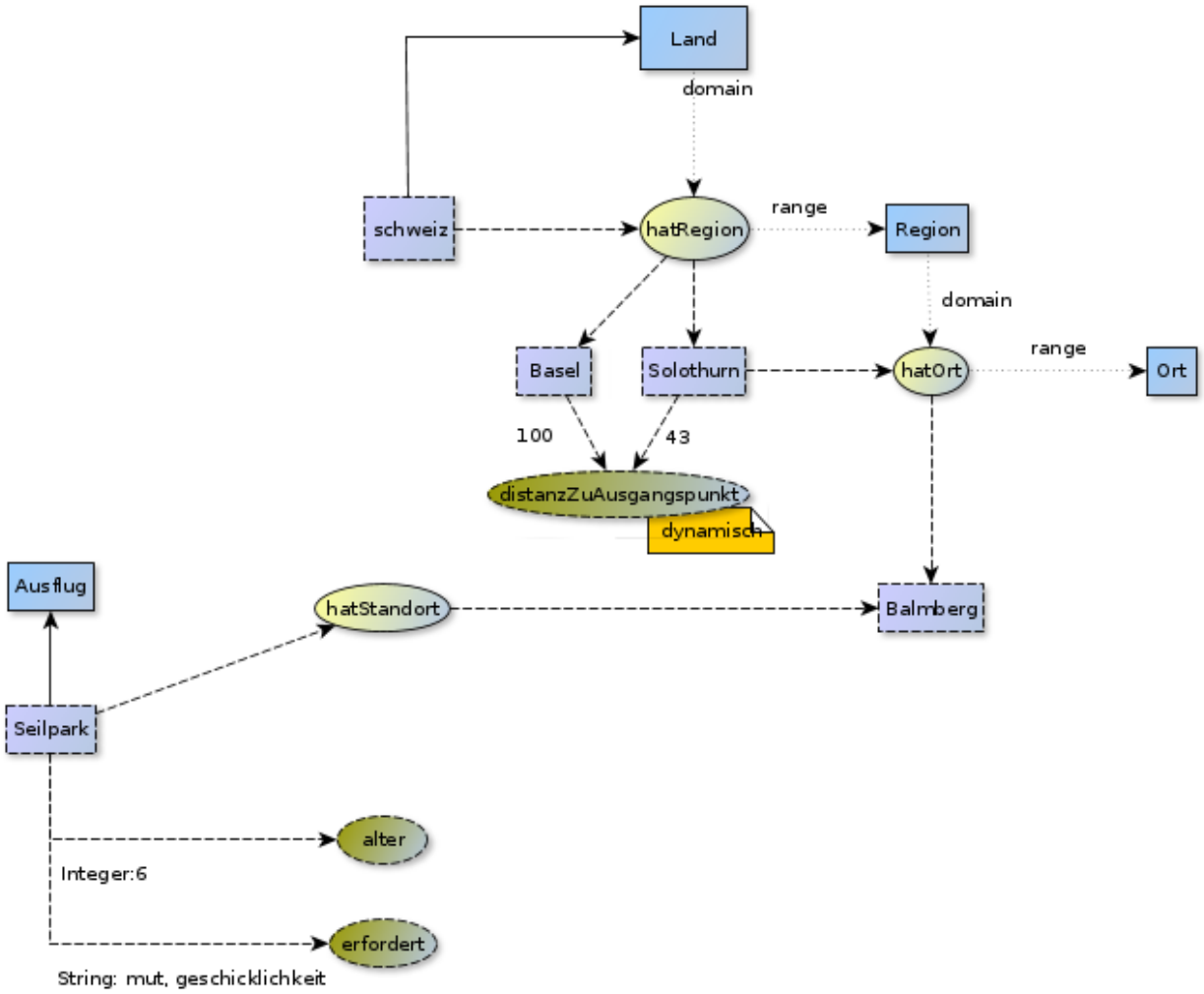
Punkt null; nachdem wir entschieden haben unsere Domäne zu ändern fangen wir also nochmal von vorne an.
Wir haben uns entschieden mit einem einfachen Beispiel zu beginnen, und uns im ersten Schritt auf die Schweiz zu beschränke.

Erster Fall: Familie Muster plant einen eintägige Ausflug. Die Kinder sind in einem alter in dem Sie immer beschäftigt sein müssen:

Kriterien:

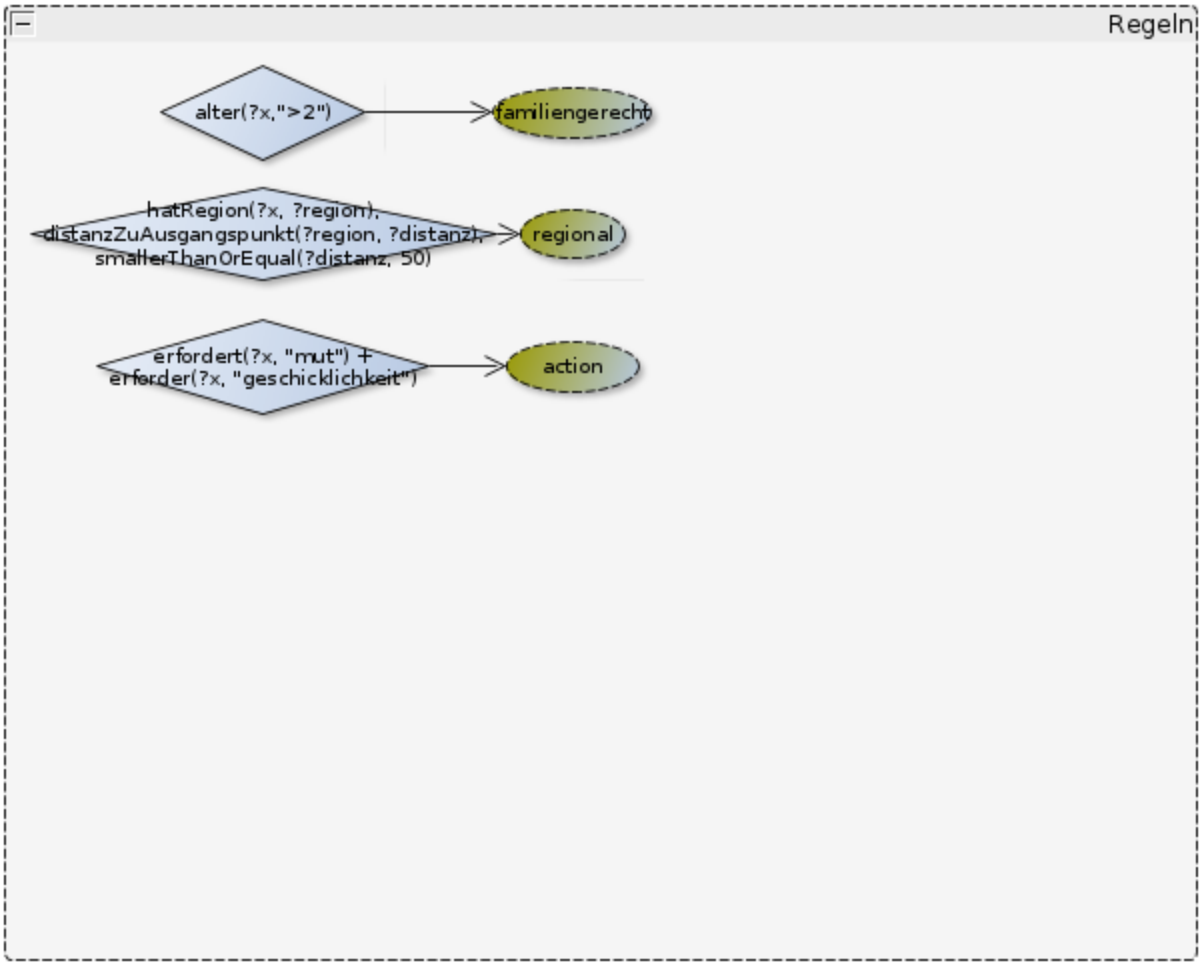
- Familienfreundlich
- regional
- action

daraus entsteht folgendes Expertensystem:

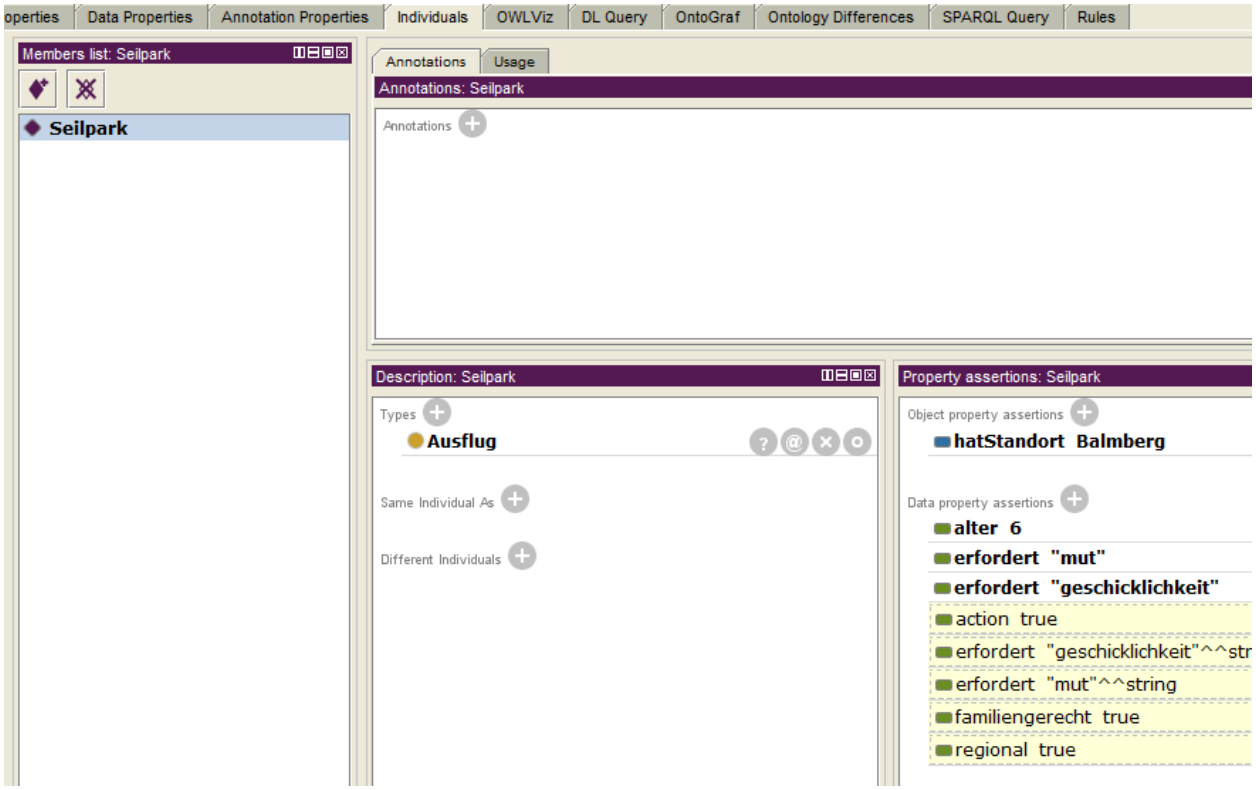


dynamisch: distanzZuAusgangspunkt könnte zu einem späteren Zeitpunkt dynamisch (zb mithilfe von Googlemaps) berechnet werden. Für unser Beispiel wurde Bern als Ausgangspunkt festgelegt und die Distanzen fix festgelegt.

Um das Ziehen von Schlüssen zuzulassen, wurden folgende Regeln definiert:



Protege kann schon einige Schlüsse ziehen:



Stellen wir in Stardog die entsprechende Regel erhalten wir die gewünschte Antwort.

```
Select * where
{
  ?object :familiengerecht true;
        :regional true;
        :action true.
  }
```

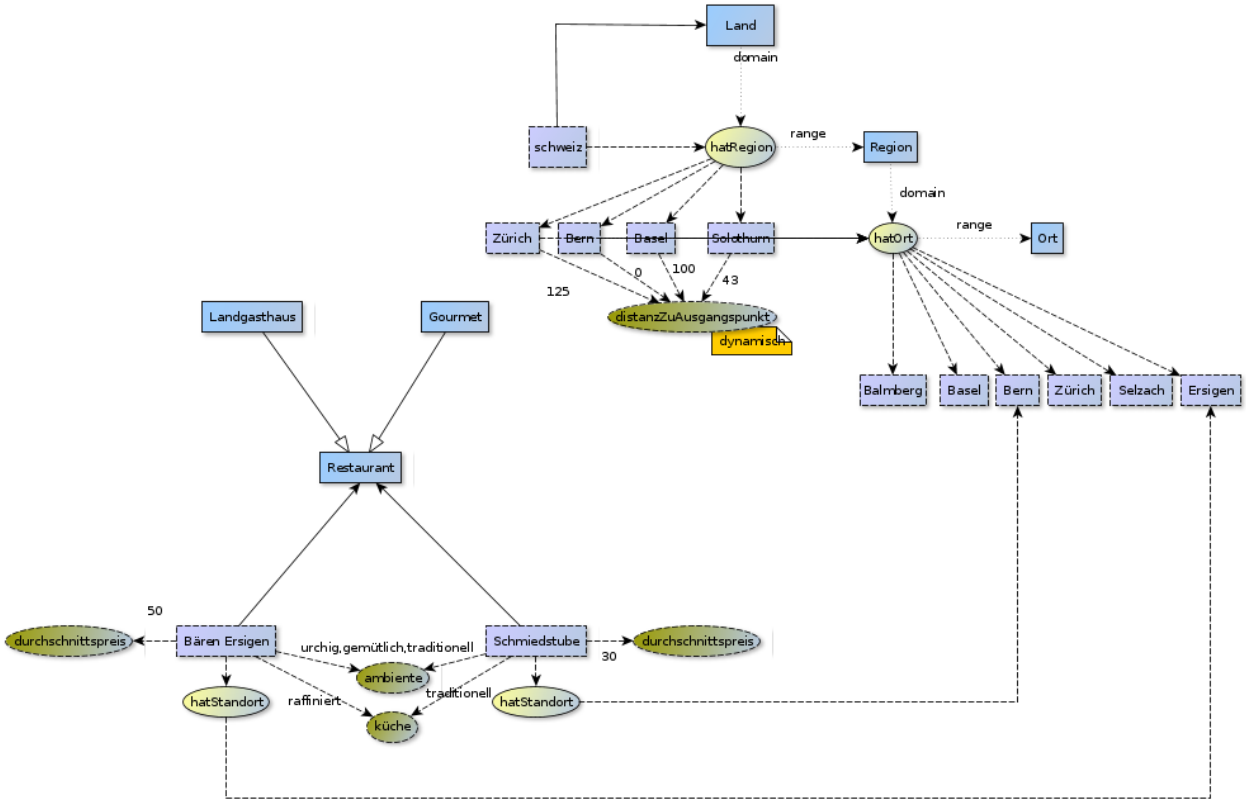
Zweiter Fall

Ein Startup-Unternehmen möchte einen Teamevent mit anschliessendem Abendessen in einem Landgasthof veranstalten. Dabei soll der Teamevent Nervenkitzel bieten.

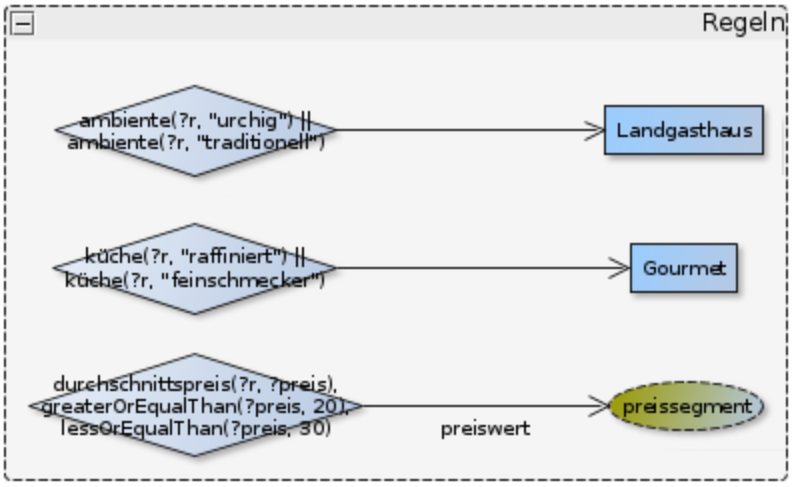
Kriterien:

- nervenkitzel
- teamevent
- preiswert
- Landgasthof

Expertensystem:



Regeln:



Mit der passenden Anfrage wird das Restaurant Schmiedstube in Bern gefunden

```
SELECT
       ?ausflug ?standort ?resti ?standort2
WHERE {
  ?ausflug :teamevent true;
           :nervenkitzel true;
               :hatStandort ?standort.
  ?region :hatOrt ?standort.

  ?resti a :Landgasthaus;
               :hatStandort ?standort2;
               :preissegment "preiswert".
  ?region :hatOrt ?standort2.


  Filter(regex(str(?region),"Bern","i"))

}
```

ohne Preiseinschränkung würde zusätzlich Bären Ersigen gefunden

Dritter Fall

Emil möchte seine Frau am Jahrestag überraschen. Sie müssen aber ausgerechnet an diesem Tag zu seiner Familie nach Basel zum Mittagessen. Deshalb sucht er in dieser Region nach einem Ausflug mit entspannung und zum Abschluss ein spezielles Essen.

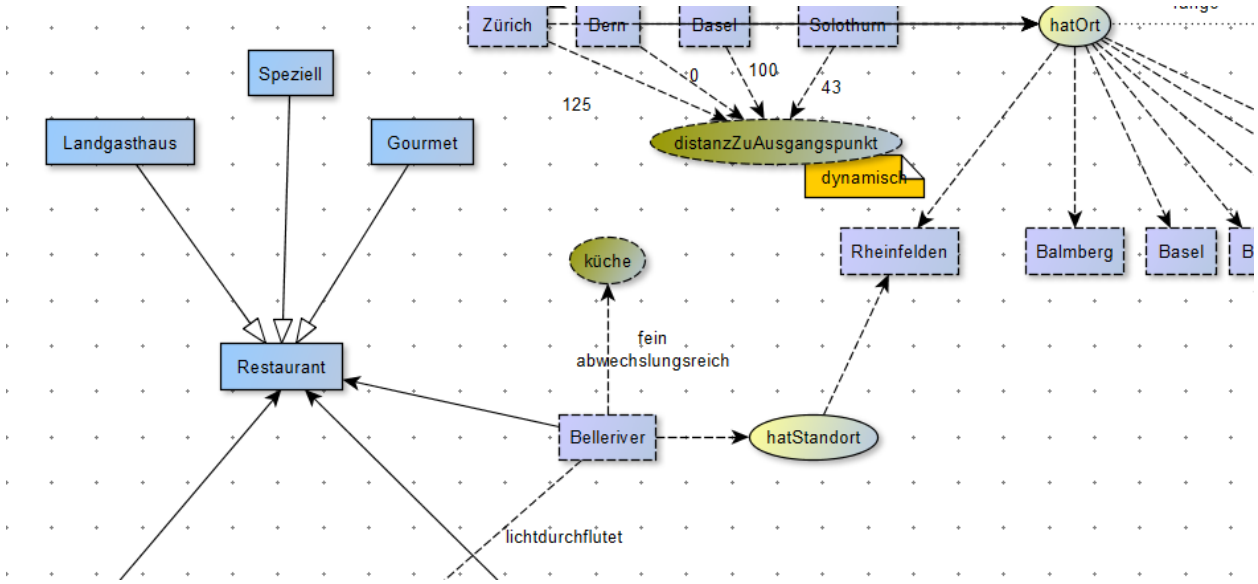
Kriterien:

- entspannung

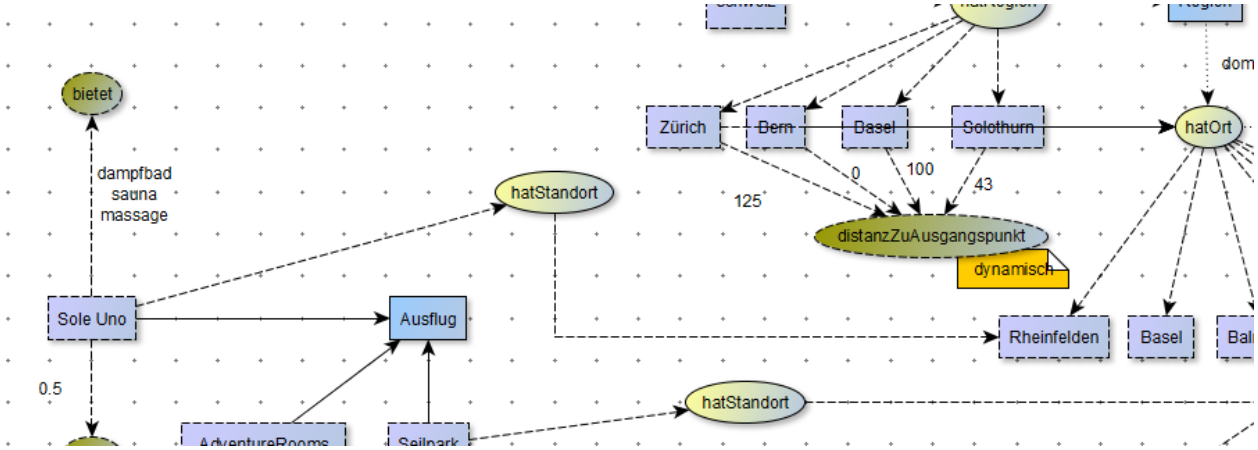
- romantik
- spezielle Küche

Sowohl die Ontologie wie auch die Regeln mussten entsprechend erweitert werden:

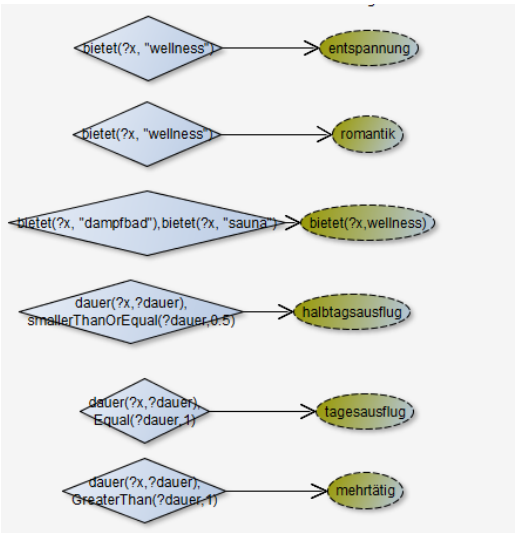
Restaurant

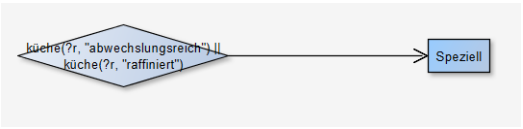


Ausflug:



Regeln



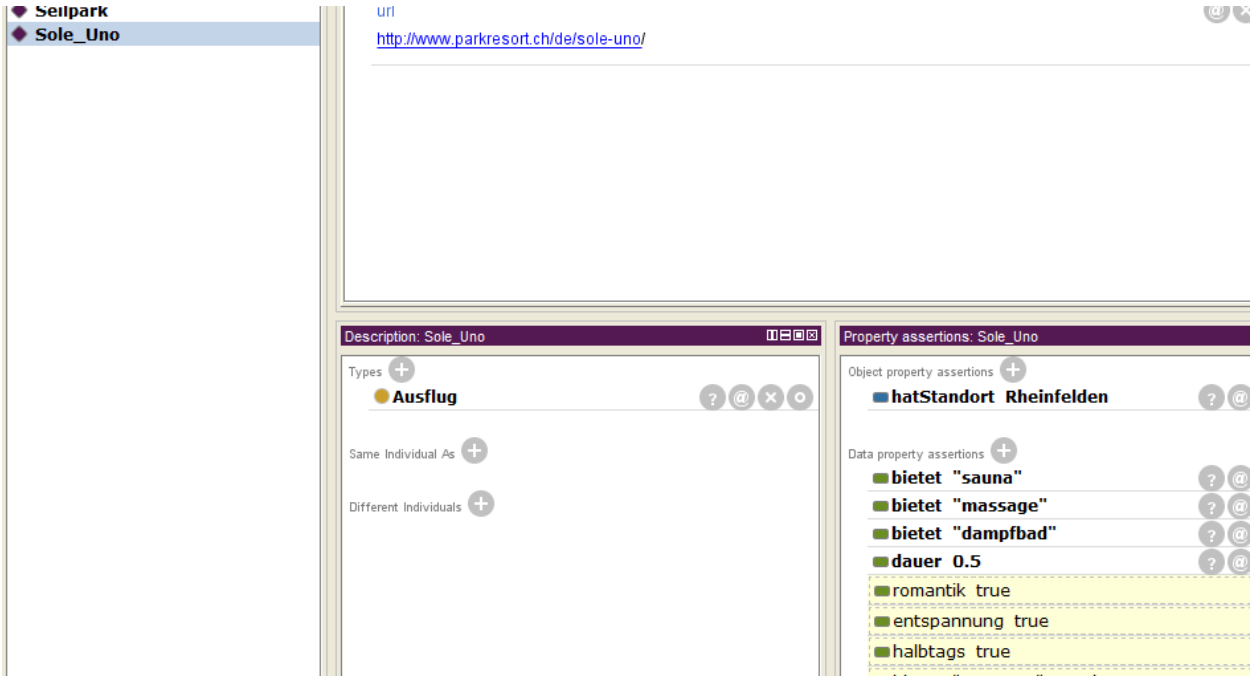


mit der passenden Anfrage wird das Sole Uno und das Restaurant Belleriver gefunden

```
SELECT
        ?ausflug ?standort ?resti ?standort2
WHERE {
  ?ausflug :romantik true;
           :entspannung true;
           :halbtags true;
               :hatStandort ?standort.
  ?region :hatOrt ?standort.

  ?resti a :Speziell;
               :hatStandort ?standort;



  Filter(regex(str(?region),"Basel","i"))

}
```

ACHTUNG: Stardogfehler, kann aus einem DataProperty nicht nicht auf das gleiche dataproperty schliessen: bietet(?x,”sauna”), bietet(?x,”dampfbad”) -> bietet(?x,”wellness”)

Auf Protege kann der Reasoner den Schluss ziehen, dass das Sole Uno in Rheinfelden Wellness bietet. Daraus wiederum schliesst er, dass Romantik und Entspannung zutreffen.

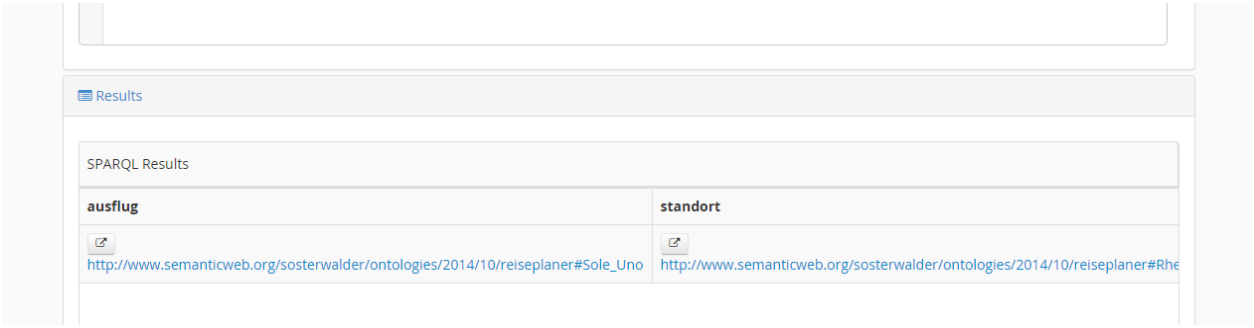


Stardog kann dies nicht ableiten. Anscheinend eben wegen dem oben erklärten Fehler.

Workaround: wir führen ein Dataproperty wellness vom Typ Boolean ein mit der Regel:

```
bietet(?x,”sauna”), bietet(?x,”dampfbad”) -> wellness(?x,true)
wellness(?x,true) -> entspannung(?x,true)
wellness(?x,true) -> romantik(?x,true)
```

So kann Stardog die Anfrage richtig Beantworten:

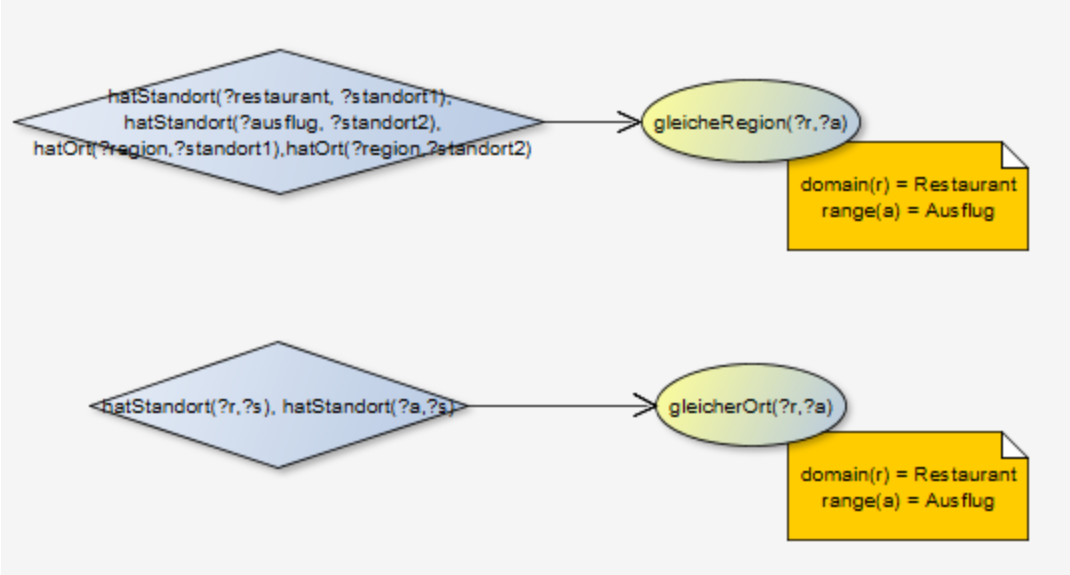


Wichtig: In diesem Fall wurde der erste Versuch gestartet eine Dauer für die Ausflüge mit einzubeziehen. Diese Regeln sind aber noch nicht fertig ausgereift. In einem weiteren Schritt soll analysiert werden, was da möglich ist.

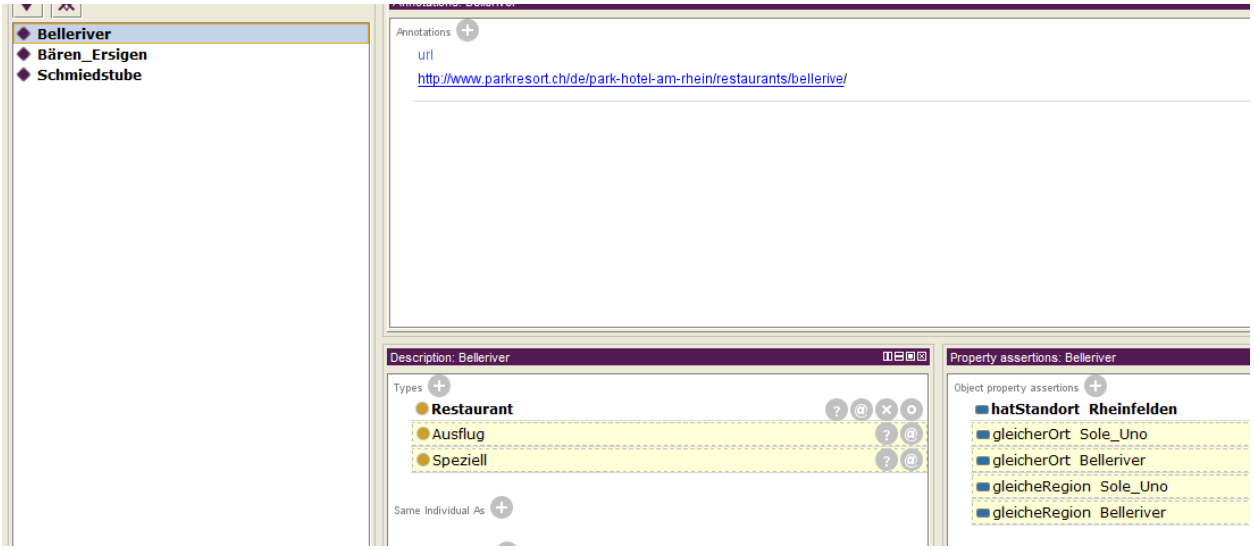
> *Die Grundidee der Dauer ist folgendermassen:*
> *jeder Ausflug hat eine Zeitdauer.*
> *dauer < 1 > halbtags (dauer(?x, ?dauer), lessThan(?dauer,1) -> halbtags(?x, true))*
> *aus dauer = 1 -> ganztags*
> *aus dauer grösser 1 -> mehrtägig*

gleicherOrt/ gleicheRegion

Bei der Besprechung der ersten 3 Fällen ist uns aufgefallen, dass wir bei diesen Beispielen je nach gutdünken entschieden haben, ob wir nach einem Restaurant suchen welches sich im gleichen Ort oder in der gleichen Region befindet. Das funktioniert so natürlich nicht. Aus diesem Grund versuche ich hier die ObjectPropertys gleicherOrt/gleicheRegion einzuführen, welche durch eine Regeln besetzt werden.



gemäss Reasoning in Protegé scheint das so zu funktionieren:

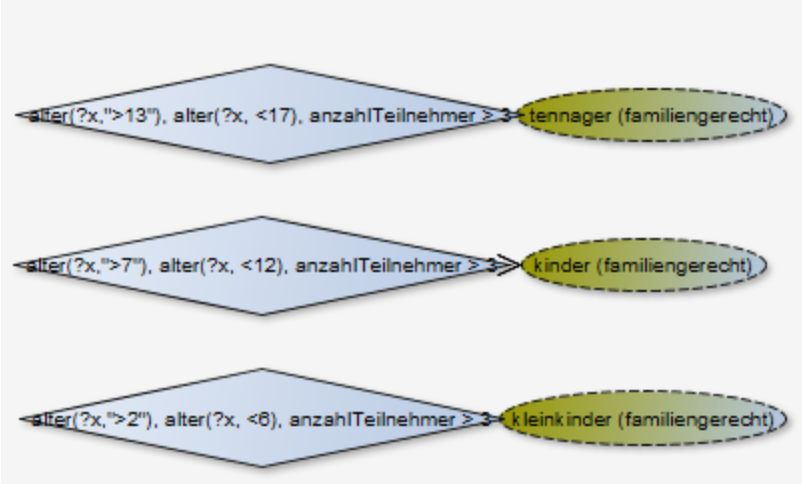


Je nachdem müssen die Regeln noch so erweitert werden, dass man angibt ob es sich um ein Restaurant handeln muss. Ich denke aber nicht dass das nötig ist.

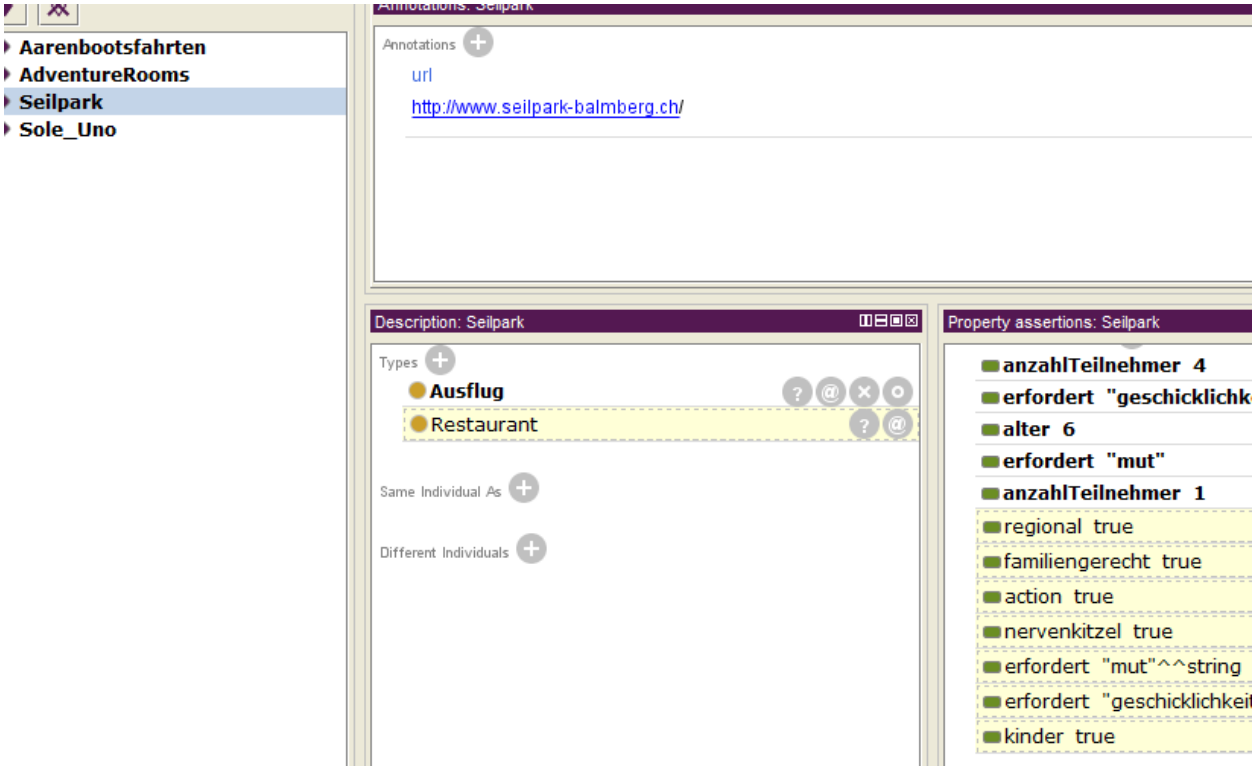
Familiengerecht erweitern

Familiengerecht wurde im ersten Beispiel seehr einfach gehalten. Das müssen wir ersten noch einschränken (mit anzahl Personen) Ausserdem macht es Sinn Familiengerecht mit Subproperties zu versehen und so je nach alter der Kinder zu unterscheiden:

Dazu wurden die SubDataPropertys tennager, kinder und kleinkinder unter familiengerecht eingeführt und die folgenden Regeln erstellt:



Dies schein wunderbar zu funktionieren, in Protegé wird weiterhin gezeigt das Seilpark ein familiengerechter ausflug ist. es ist nun zusätzlich noch spezifiziert das es für kinder geeignet ist.



Anzahl Teilnehmer

AnzahlTeilnehmer ist so wie es bisher verwendet wurde problematisch, weil es kein von - bis gibt. Ein Event kann ja sowohl für einzelpersonen als auch für Gruppen geeignet sein. Dies kann abgedeckt werden indem man entweder AnzahTeilnehmerVon und AnzahlTeilnehmerBis einführt. Oder indem dem Ausflug die Property AnzahlTeilnehmer zweimal zuweisst. Einmal das minimum 1 ein anderes mal mit 4, dann werden Sowohl Gruppen als auch Familien als auch einzelpersonen gefunden. -> Im Moment hab ich das so umgesetzt.

-> Ich glaube es macht aber mehr sinn mit von bis. Das können wir aber am Mittwoch zusammen besprechen.

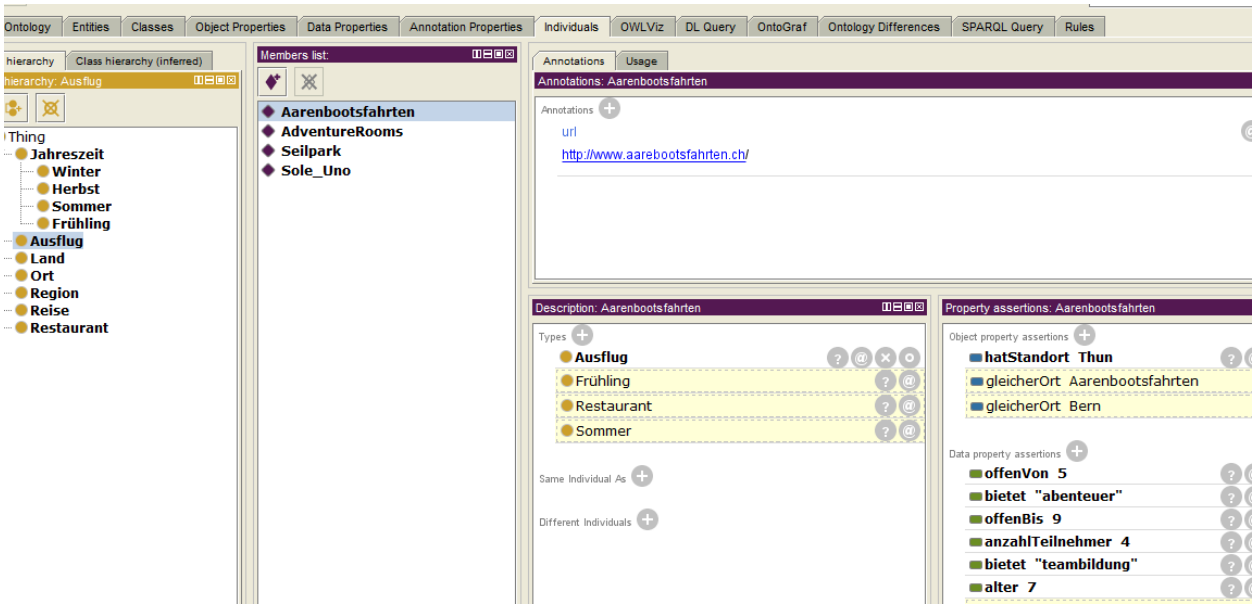
Saison

Saionall
Frühling
Sommer
Herbst

Winter
mit definition der Jahreszeiten + date.today oder so in der programmlogik?
Es kann aber in einem ersten Schritt so gelöst werden das die
Saison ausgewählt wird, also als Ojectproperty.
Das ist aber langweilig so. Idee mit offenVon/offenBis in monaten 1-12
und definiert zwischen wo und wo welche Saison ist?

offenVon(?x,?von), greaterThanOrEqual(?von,3),lessThan(?von,7) -> Frühling(?x)||
offenBis(?x,?bis), greaterThenOrEqual(?bis,3),lessThen(?bis,7) -> frühling
offenVon(?x,?von) greaterThenOrEqual(?von,7),lessThen(?von10) -> sommer ||
offenBis(?x,?bis), greaterThenOrEqual(?bis,7),lessThen(?bis,10) -> sommer
offenVon(?x,?von) greaterThenOrEqual(?von,10),lessThenOrEqual(?von,12) -> herbst
offenBis(?x,?bis), greaterThenOrEqual(?bis,10),lessThen(?bis,12) -> herbst
offenVon(?x,?von) greaterThenOrEqual(?von,12) -> winter ||
offenVOn(?x,?von),greaterThenOrEqual(?von,1)lessThen(?von3) -> winter ||
offenBis(?x,?bis) greaterThenOrEqual(?bis,1),lessThen(?bis,3) -> winter

Auf den ersten Blick scheint das so zu funktionieren:

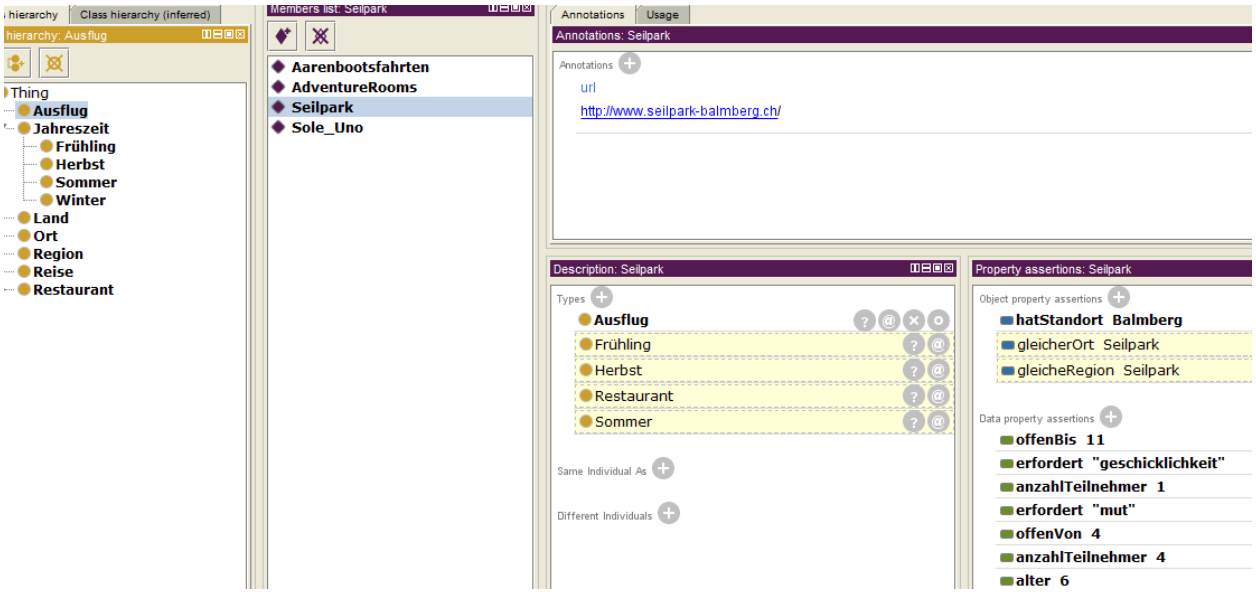


Das ist so natürllich nicht ganz richtig:
offenVon(?x,?von), greaterThenOrEqual(?von,3),lessThen(?von,7)
offenVon(?x,?von), offenBis(?x,?bis),lessThanOrEqual(?von,3), greaterThen(?bis,3) ->
Frühling(?x)

offenVon(?x,?von), greaterThanOrEqual(?von,7),lessThan(?von,10)
offenVon(?x,?von), offenBis(?x,?bis),lessThanOrEqual(?von,7), greaterThan(?bis,7) ->
Sommer(?x)

offenVon(?x,?von), greaterThanOrEqual(?von,10),lessThan(?von,12)
offenVon(?x,?von), offenBis(?x,?bis),lessThanOrEqual(?von,10), greaterThan(?bis,10) -> Herbst(?x)

offenVon(?x,?von), greaterThanOrEqual(?von,1),lessThan(?von,3)-> Winter(?x) ||
offenVon(?x,?von), greaterThanOrEqual(?von,12)> Winter(?x)



Saison nochmal überarbeitet:

Die einzelnes Jahreszeiten als Individien und mit Object Propertys; Regeln ansich bleibt aber gleich.
Rule: offenVon(?x, ?von), offenBis(?x, ?bis), lessThanOrEqual(?von, 3), greaterThan(?bis, 3) -> hatJahreszeit(?x, frühling)

Anzeige:

Domain/ Range
Damit die Abfrage zum anzeigen aller möglichen Propertys richtig gemacht werden kann müssen diese als Domain an die RoutElemente (Ausflug/ Restaurant) gebunden werden.

Ausflug:

DataPropertys:

- familiengerecht
    - kinder
    - kleinkinder
    - teenager
- action
- teamevent
- nervenkitzel
- entspannung
- romantik
- wellness
- regional
- reisedauer:
    - halbtags/
    - tagesausflug/
    - mehrtägig

```
SELECT *
WHERE
{
	?subject rdfs:domain rp:Ausflug.
	?subject  rdfs:subPropertyOf  ?super.
}
Order By ?super  ?subject
```

-> dies führt zu einer Ordnung nach der Superproperty. Mit Hilfe dieser kann meiner Meinung nach mittels Programlogik eine übersichtliche Auswahl der Propertys erzeugt werden. (if ?super isnot topDataProperty… oder so)

Klassen

Ausserdem gibt es folgende Klassen die spezifiziert werden können:

- Saisonall
    - Frühling
    - Sommer
    - Herbst
    - Winter

Die folgende Abfrage gibt uns alle Jahreszeiten zurück.

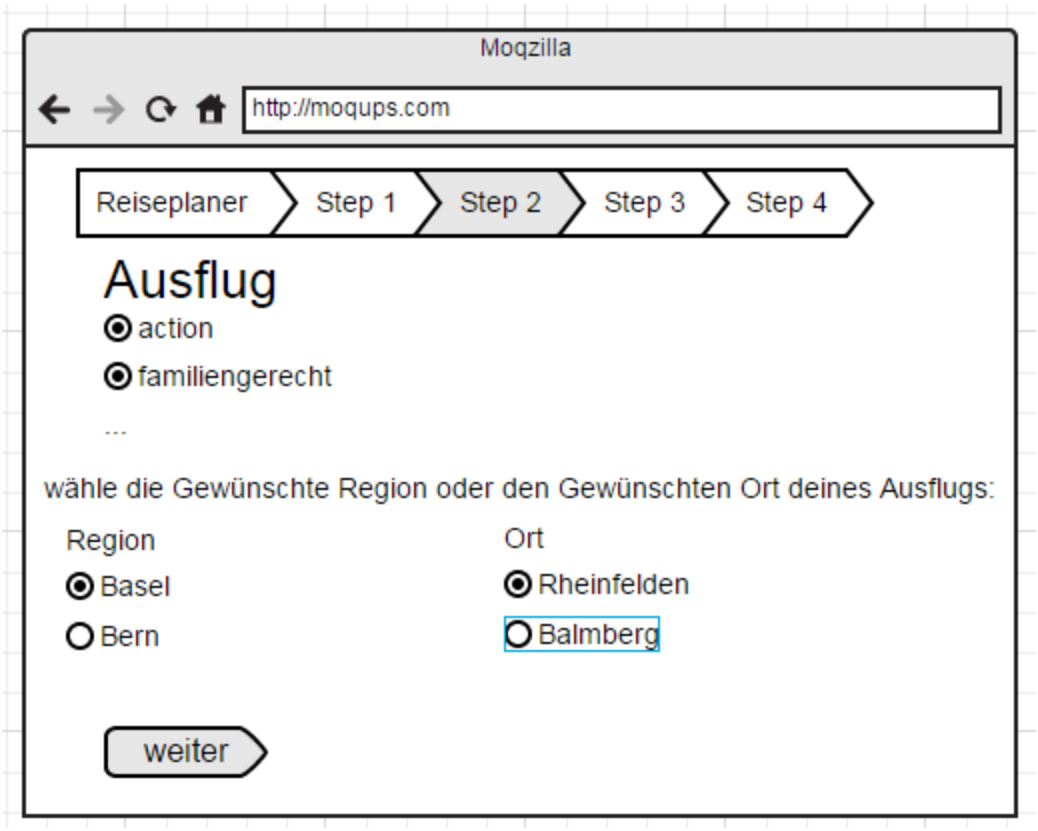
```
SELECT
	*
WHERE {
	?j  rdfs:subClassOf rp:Jahreszeit
}
```

Dieser Teil muss irgendwie abgesetzt sein. Entweder als eigener Step (dann würde ich auch die Orte als eigener Step nehmen) oder wie in der Unteren Skizze für Standort angedacht auf dem gleiche Step.

ObjectProperty:

- hatStandort

Um dem Benutzter die Möglichkeit zu bieten Ort rsp Region auszuwählen würde ich das irgendwie so machen:




```
SELECT *
WHERE
{
?subject a rp:Region.
}
```

das kann ich leider nicht prüfen, da Stardog wieder mal nicht läuft. und so wie ichs in Erinnerung habe ist genau das die Abfrageart die Protege nicht richtig beantwortet..

Orte und Regionen würde ich Im Gegensatz zu den Propertys auch mit Oder Verknüpfungen für die Abfrage zusammenhängen.
Dies kann folgendermassen im Filter umgesetzt werden:

```
        Filter(regex(str(?ort),"Basel","i") || regex(str(?ort),"Bern","i"))
```

Restaurant:

- preissegment
- gleicherOrt
- gleicheRegion

```
SELECT
        *
WHERE {
        ?r rdfs:domain rp:Restaurant
}
```

evt. würde ich da noch mehr kriterien einbauen? oder die Unterklassen von Restaurant noch mehr ausbauen?
Klassen:

- Restaurant
    - Landgasthaus/
    - Speziell
    - Gourmet

```
SELECT
        *
WHERE {
        ?j  rdfs:subClassOf rp:Restaurant
}
```

ObjectProperty

- gleicherOrt/ gleiche Region

Diese Eigenschaften werden hier direkt mit den DataPropertys angezeigt. Ich finde das macht eigendlich schon noch sin. Vielleicht kann man das noch intelligenter benennen

Abfragen zusammenfügen

Das wird wahrscheinlich noch ein ziemlicher brocken;
wie kann man die Anfragen dynamisch intelligen zusammen bauen? So auf den ersten Blick denke ich, dass wird ne ziemliche ansammlung von if abfragen:

- wenn eine DataPropterty gesetzt ist dann…

- wenn eine Region gesetzt ist dann…
- wenn ein Ort gesetzt ist dann..

- wenn auch Restaurant gewählt ist dann…
- wenn gleicherOrt/region gesetzt ist dann…

Das überleg ich mir vielleicht heute noch. Jetzt gehe ich erst mal Trainieren ;-)
-> Viel Spass Sven ;-)

Überarbeitet:

Allgemein:
- Blacklist einführen
-

DataProperty:
- Ok mit Domain; subProperty baum ist für die anzeige noch Problematisch

ObejectProperty:

```
var queryString = '\
  select distinct\
    (strafter(str(?rel), "#") AS ?relation) \
    (strafter(str(?ziel), "#") AS ?property) \
  where {\
    ?ind ?rel ?ziel.  \
    ?ind a :' + reiseModel.get('routeName') + '. \
    ?rel a owl:ObjectProperty.
```

-> Problem: man kann nur ?property dynamisch anzeigen. Lösung für den TItel finden.
- Idee: mit eigenem Loop für ObjectProperty in index.html?

SubKlasse:
- gibt es eigendlich nur bei Restaurant; das ist “ einfach” mit subklass von routeName
- SELECT

```
        *
WHERE {
        ?j  rdfs:subClassOf :<routeName>
}
```

Überarbeitung Modellierung Mira:

Fragen:

- wieso meinen die Restaurants sie seien auch Ausflüge?
- und die Ausflüge sie seien auch Restaurants?

bei regional waren die Domänen Restaurant und Ausflug zugeweisen. Anscheinend schliesst der Reasoner bei der Domäne draus, dass ein Individum das die entsprechende Property hat (also zb regional) sämtlichen Klassen der Domäne entspricht; also Restaurant und Ausflug. -> Lösung: Eine Domäne kann nur zugeweisen werden wenn sie eindeutig ist.

- wieso meinen alle anderen Klassen das sie eine subklasse von Jahreszeit seien?
  - Das meinen Sie nur in Protege; mit einer Stardog Abfrage gibt es die richtige Ausgabe.
- wieso wird bei regional auch AdventureRooms in Zürich angezeigt?

Restaurant:

- Grafik sauber; evt noch sämtliche Preissegment Regeln Eintragen

Ausflug:

- Grafik Onto ok;
- evt Preisdifferenzierung auch bei Ausflügen! -> überlegen ob die gleichen Regeln verwendet werden könnten? dann müsste nur die Domains erweitert werden. Ansonsten müsen neue Regeln hinzugefügt werden.

Erweiterung Benutzeroberfläche:
Werden mehrer eigenschaften des gleichen objectPropertys ausgewählt müssen diese mit Union verbunden werden:

```
SELECT
(strafter(str(?obj), "#") AS ?travel)
(strafter(str(?loc), "#") AS ?location)
?url

WHERE {
  ?obj a :Restaurant.
  ?obj :url ?url.
  ?obj :hatStandort ?loc.
  {?obj :hatPreis :gehoben.}
  UNION
  {?obj :hatPreis :preiswert.}

}
```

23.12.2014 Weitermodellierung:

Preisdifferenzierung bei Ausflügen:

- Eigene Regeln; Preisdifferenzierung bei Ausflügen würde ich anderst setzten zb:
    - günstig: 0-20
    - Preiswert: 20-50
    - mittel: 50 - 100
    - gehoben: 50-150
    - exklusiv: 150-x
- wenn das noch eingeführt wird, muss grundsätzlich gecheckt werden, ob bei den Restaurant Regeln auch eingegrenzt ist, dass dies nur Bei Restaurants gilt
    - Bei den Ausflug Regeln müsste das auch sichergestellt werden

Öffnungszeiten

sind doch nicht so easy wie gedacht, denn:

- ich könnte von Montag bis Sonntag jeden Tag (in zahlen 1-7) angeben, an dem der Ausflug geöffnet hat und das ganze dann als liste wie bei Jahreszeiten ausfüllen.
    - Das ist aber nicht wahnsinnig lustig
    - ich kann nicht die angeben welche geschlossen sind, weil soviel ich weiss keine Regel für verneinung gewählt werden kann
- Ich glaube vorallem herr eckerel hat eigendlich nach Öffnungszeiten in sinn von Datum gefragt. Was natürlich wieder absolut nicht geht ohne Programmlogik.
    - Dort könnte man das mit einer Liste für jeden Ausflug lösen

- OK, ich kanns doch mit swrl:notEqual machen, das ist aber auch noch nit sooo lustig...aber ich machs jezt mal so:

DataProperty:
    Ruhetag: (1-7)
ObectProperty:
    Reisetage (die Möglichen Tage können hier angehäckelt werden)
Individuen:
    Montag-Sonntag

Regeln:
    ruhetag(?a,?t), swrl:NotEqual(?t,1) -> Reisetage(Montag)

Rule: ruhetag(?x,?t), swrlb:notEqual(?t,1) -> Reisetage(?x, Montag)

-> So ist das ganze glaub garnicht so schlecht. Aber bei den Abfragen mit Reisetagen müsste auch mit Union die Sparql abfrage zusammengestellt werden.

-> so muss man einfach wenn es Täglich geöffnet hat ruhetag zb tag 8 setzten. Weil ja nicht auf nicht abgefragt werden kann. Das ist schon ein bisschen ein gebastel.

Ausserdem ist es Grundsätzlich ein bisschen unsinnig weil alle unsere Ausflüge Täglich offen haben.

-> Insgesamt kann aber Reisetage einfach gleich abgehandelt werden wie Jahreszeiten

Zoo Basel

Zur veranschaulichung der bestehenden Eigenschaften und Funktionalitäten wurde noch ein neuer Ausflug hinzugefügt.
Der Zoo Basel
Neue Spezialitäten:

- der Zooausflug kann sowohl einen halben als auch einen ganen Tag dauern, somit wird er sowohl bei den halb als auch bei den ganztagesausflügen angezeigt
- neues Folgeproperty: lehrreich dies tritt ein wenn sowohl informationen als auch unterhaltung geboten werden.
- auch das alter ist sowohl mit 4 als auch mit 12 besetzt. Damit wird gesagt das sich der ausflug für Kinder in jedem Alter eignet

Rule: bietet(?a,"unterhaltung"),bietet(?a,"informationen") -> lehrreich(?a,true)