The OntoWeb Evaluation Experiment for Ontology Editors: Using Protégé-2000 to Represent the Travel Domain

Natalya F. Noy
Stanford Medical Informatics, Stanford University
noy@smi.stanford.edu

1 Design decisions

Our goal was to represent the domain as close to its natural-language description as possible. We tried to introduce only the concepts and relations that were necessary and sufficient to represent the facts in the description. We did not add any other common-sense facts about the domain.

1.1 Assumption that we have made in interpreting the domain description

Several facts in the description allowed more than one interpretation. Here is a list of assumptions that we have made:

- Prices for flights (business and economy class) are constant for a particular flight and do not change from day to day. In other words, it always costs the same to fly from Madrid to New York on the flight number UA345
- Other trips (not just flights) also have arrival and departure city, arrival and departure time, port (airport, station) of arrival/departure, etc.
- The description said: "We know that each model of transport belongs only to one kind of transportation (e.g., it's either a plane, or a bus, or a car, etc.)." We interpreted this sentence to mean that each maker of transportation manufactures only one type of transportation. For example, if Boeing makes planes, it cannot make trains.

We believe that other statements in the description were unambiguous and there was only one possible interpretation.

1.2 Classes and slots

Figure 1 shows elements of the class structure and some relations among classes.

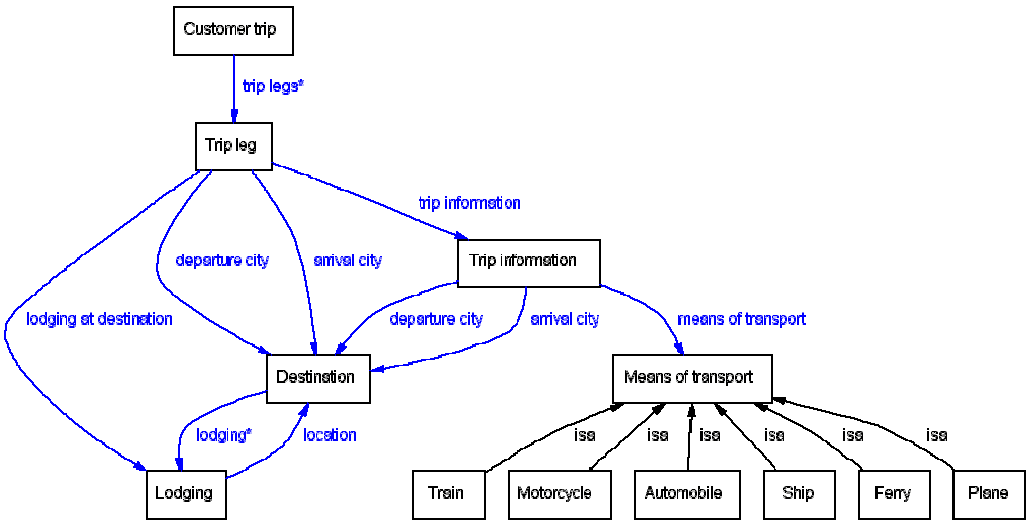

Figure 1. Elements of the class structure and relations. Boxes represent classes and arrows represent relations.

We start defining a customer's trip as an instance of the class `Customer trip`. Each instance of this class contains the customer's `name` and points to one or more legs of the trip. Each `trip leg` is an instance of the class `Trip leg` describing `departure` and `arrival time` and `departure and arrival cities`. It points to more specific `trip information`: the specific flight the customer is taking on that trip, or specific train, or the car he is renting. There is a constraint indicating that the arrival and departure cities for the trip leg must be the same as the arrival and departure cities in the corresponding `Trip information` instance.

An instance of `Trip information` represents information about particular flights, train rides, etc. That is, an instance of this class could be flight UA455 that leaves Paris at 9am and arrives to NY at 1pm every day. The `Flight` subclass of `Trip information` will include prices for economy and business class, and a flight number. We assume that this information does not change from day to day.

Arrival and departure cities on the trips are instances of the `Destination` class. In addition to the city `name`, its `country` and `continent` (we need the latter for one of the constraints), instance of the `Destination` class describes `local transport` in the city, `points of interest`, and a list of available `lodging` options. The options for the local transport are the default values for the `local transport` slot at the destination. The list of available lodging options contains instances of the class `Lodging`. In addition, each destination has a Boolean slot indicating whether it `has an airport`.

The `Lodging` class has two subclasses—`Hotel` and `Bed&Breakfast`. Each instance of `Lodging` points back to the `Destination` (the slot `location` is inverse of the slot `lodging` at the `Destination` class). Each `Hotel` instance has a required slot indicating its `star rating`. Each `Lodging` instance points to an instance of the `Room facilities` class describing individual rooms.

A class `Means of transport` represents different transport options for customer's travels. Specific means of transport are subclasses of this class. Each instance has a `make` and `model`. Hence, we can represent makes and models of particular planes, automobiles, etc. The `Means of transport` class is *abstract* to indicate that every instance of this class must be an instance of one of its subclasses. We attached a PAL axiom to this class expressing the constraint on makes and models: each maker produces only one type of transportation (see the Assumption above). Specific makes and models of planes, cars, etc. are instances of this class.
Instances of `Trip information` point to instances of the `Means of transport` class indicating which model of a plane, ship, train, is used for a particular flight, voyage, train ride, respectively.
There is a class `Distance` table which contains pairs of distances between destinations.

1.3 Constraints

The three constraints describing when customers would prefer to travel by train or car are PAL constraints.
The first constraint is "we know that it is not possible to go from America to Europe by train, car, bike nor motorbike." To express this fact, we attach the following PAL axiom to the `Trip information` class:

```
(forall ?trip
   (=> (and (name (continent ('arrival city' ?trip)) "Europe")
       (name (continent ('departure city' ?trip)) "North America"))
       (instance-of ('means of transport' ?trip) Plane      )))
```

A similar axiom expresses the constraint for the opposite direction (from Europe to North America).
The second constrain is "If distance between two cities is between 400 and 800 miles, and there is no airport close to one of them, the customer will prefer going by car or by train." We attach the following PAL axiom to the `Trip Leg` class:

```
(forall ?tripleg
   (=> (exists ?distance
          (and (to ?distance ('arrival city' ?tripleg))
          (from ?distance ('departure city' ?tripleg))
          (> ('distance in miles' ?distance) 400)
          (< ('distance in miles' ?distance) 800)
          ('has airport' ('arrival city' ?tripleg) FALSE)
          ('has airport' ('departure city' ?tripleg) FALSE)))
          (or (instance-of ('means of transport'
                                      ('trip information' ?tripleg))
                           Automobile)
      (instance-of ('means of transport' ('trip information' ?tripleg))
                   Train))))
```

To express the last constraint "The customer also prefer to go by car or train if he hates travel by plane.", we attach an axiom to the `Customer trip` class:

```
(forall ?customer
   (=> ('hates planes' ?customer true)
     (forall ?tripleg
      (=> ('trip legs' ?customer ?tripleg)
          (or (instance-of ('means of transport'
                                  ('trip information' ?tripleg))
                           Automobile)
```

```
(instance-of ('means of transport'
                          ('trip information' ?tripleg))
                    Train))))))
```

1.4 Instances

To represent a specific trip, we create an instance of `Customer trip` (Figure 2). It has pointers to three trip legs. Each leg points to a Trip information instance describing specific flights for the trips to and from Madrid. We do not specify means of transportation for the New York-Washington leg.

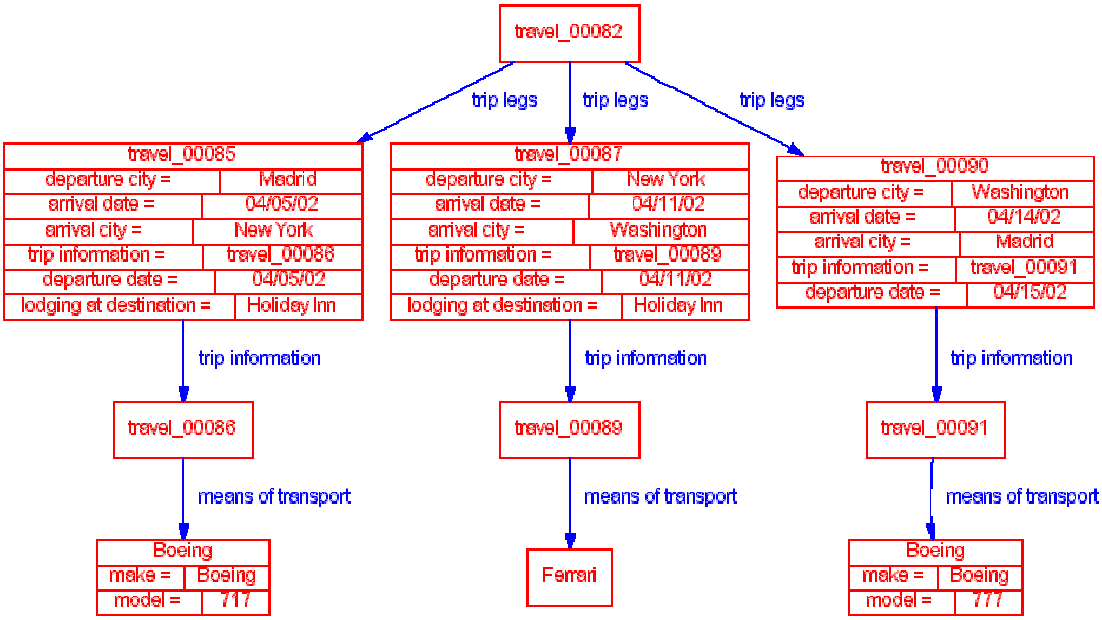

Figure 2. The customer traveling from Madrid to the US

2 Discussion

We were able to represent most of the facts from the description. We found that we had to revise the class structure significantly twice: First, when we got to the instance definition, we learned that a trip can have several legs and therefore had to add an intermediate `Trip leg` concept. Then, in order to express some of the additional constraints, we needed to add a number of new attributes to many of the classes and introduce the `Continent` class.

The class structure ended up being somewhat complicated. We believe that this complexity resulted from some of the requirements in the description: that customers can have several legs in one trip, using different means of transportation for each of them, that we define ticket prices for each flight, etc. However, these complexities exist in the real life and a real-life ontology would probably have been even more complicated.

We used many of the available knowledge-modeling primitives:

- *inverse slots* to link lodging and location

- *default values* to indicate default list of options for local transport at the destination. Designers can change this list for a specific destination since not all the towns have metro, for example; and some may have trams.
- *slots as first class objects* to attach the same slots to different classes. The slots `arrival city` and `departure city` are attached both to the `Trip leg` and `Trip information` classes. The slot `name` is attached to several classes as well.
- *abstract classes* to indicate that the subclasses of the `Means of transport` class enumerate all the possible means of transport

We used the Ontoviz plug-in to visualize relationships between classes and instances graphically (and to generate figures in this report). Being able to see the resulting structure in a graph, helped a lot in analyzing the emerging ontology.

We also used the Protégé Axiom Language to express domain constraints that could not be expressed in the frame formalism directly. In addition to the three constraints in the domain description, we specified a PAL axiom linking arrival and departure cities in the instances of `Trip information` and `Trip leg`. We also used a PAL axiom to express the fact that lodging at destination must be located in the same city as the destination.

There were several facts in the domain description that we did not represent. First, we did not represent the following fact: "From all of them, the travel agency is specially interested in flights, as it is the means of transport mostly used by its customers" In our representation, when we fill in the value for the means of transport slot (in the Trip information class), we put in *instances* of specific planes, trains, etc. thus, we cannot seta preferred *class* of transport.

"The most common destinations are ….". If there was only one most common destination, we could have put it as default value for destination. However, selecting some of the destinations as more common and some others as less common (given that we then set the corresponding slot value to only one of those destinations) was not possible.

We used the Protégé RDFS backend to generate RDF. Since the backend is designed to store all the information that is necessary to restore a complete project, we did not need any other format.

3 Conclusions

We were able to represent most of the information in the domain description. To do that, we used many of the knowledge-modeling features available in Protégé, such as inverse slots, default values, slots as first-class objects, abstract classes. Features that we lacked included more flexible default (or some other mechanism) to support preferences and prototypical instances.