Brief Article

The Author

January 10, 2014

1 Task Knowledge

2 Inference Knowledge

As inference model we use a modified version of the Configuration design template, because given predefined components we need to find and assembly that satisfies the requirements. The inference model deriving from this task can be found in .

inference	Input	Output	Description
specify	requirements	sketal design	the function look-up the default sketal design: the basic structure of a trip day (heavy activity during the morning, relaxing afternoon, evening and meal).
optionalize	needs of the customers	preferences, requirements, constraints	the needs and desires are translated into preferences ("I would like to have time for shopping and visit many cultural places. I am not interested so much in food places"), requirements ("I want a quiet trip") and contraints ("In Rome I want to visit the *Colosseum* and avoid *Piazza di Spagna*").
propose	preferences and requirements, sketal design slots	filled sketal design	fill the slots of the sketal design with locations that fits the preferences and requirements.

verify	contraints, extension design	the list of violated contraints	it checks with the help of the internal contraints and those supplied by the user whether the current configuration is internally consistent. If the verification fails, it produces the violated contraints as an additional output
select	fix actions list	fix action	It simply selects an action from the fix actions list generated by the critique function.
modify	itinerary design, fix actions list	fixed itinerary design	it applies the fix actions to the design.
critique	itinerary, violations, customer's inputs	fix actions list	it creates a series of actions which will fix the violations of the contraints, following also the customer's inputs. For example the contraint "I absolutely want to visit the *Colosseum*" will produce the action "Insert the *Colosseum* into the itinerary".

3 Domain knowledge

3.1 Domain schema

The domain schema can be found in figure... This schema seems complicated, for this reason every model is explained in the following list:

Client

The client who goes to the travel agency. He could be a quite person who normally wants to visit a lot of things or very few (activeness). The clients are categorized by their age because ??

Preference

Each client needs to specify a list of preferences, valued from 1 to 5, where 1 is "I'm not so interested" and 5 is "I love to do it!". These preferences are related to the itinerary we want to create, consequently if the same clients wants to create another itinerary, it will specify again all the preferences he wants in this second trip.

Contraint

Each client needs to specify a list of contraints that have to be fulfilled. As for the *Preference*, they are related to the single itinerary.

Itinerary

This represents the itinerary we want to create. It is composed by a fixed number of *Day* and it is related to a *Client* who has specified his own list of *Contraint* and *Preference*. If there will be kids in the itinerary, the system needs to select some *Location* that could entertain them. This is a requirement.
The method *selectLocation* takes a list of *Preference* and produces a list of *Location* that could fit this preferences.

Day

This describes a day of the itinerary.

Timeslot

A timeslot is a fixed part of a day. The division of the day came from the expert interview.

Location

This model represents the point of interests that a customer could visit. The attribute *rating* describes the quality of this place, *intensive* describes if the place is not for quite people. The method *distance* takes two locations and returns the distance between them. It is useful in order to create the combination of locations to visit during a trip.

3.2 Rule types

Listing 1: Rules

```
RULE TYPE constraint-rule;
    DESCRIPTION: "rule stating the relation between client and
        the choice for a location in the itinerary, by means of
        defining strict boundaries that must be respected.";
ANTECEDENT: Client;
```

```
CONSEQUENT: Itinerary;
CONNECTION-SYMBOL: restricts;
END RULE-TYPE constraint-rule;

RULE TYPE requirement-rule;
    DESCRIPTION: "rule stating the relation between the client
       and the choice for a location in the itinerary, by means
        of defining boundaries that should be respected.";
ANTECEDENT: Client;
CONSEQUENT: Itinerary;
CONNECTION-SYMBOL: requires;
END RULE-TYPE requirement-rule;

RULE TYPE preference-rule;
    DESCRIPTION: "rule stating the relation between the client
       and the choice for a location in the itinerary, by means
        of defining preferences that could be satisfied with
       probability X (calculated on the input values) .";
ANTECEDENT: Client;
CONSEQUENT: Itinerary;
CONNECTION-SYMBOL: prefers-with-probability;
END RULE-TYPE preference-rule;
```

Here are presented also some example in order to better understand all the rule types.

Listing 2: The client wants to include a destination into the itinerary.

```
client.constraint.location.name=A AND client.constraint.type=
   include
RESTRICTS
∃itinerary.day.timeslot, timeslot.location.name=A;
```

Listing 3: The client is a quite person

```
client.quiet=true, client.needsFreeTime=true, client.active=1
REQUIRES
itinerary.day.timeslot.location, location.intensive=false;
itinerary.day.timeslot, timeslot.location=NULL;
∑_{i=1}^{n-1} i.distance(i+1) < δ, ∀i ∈ location;
```

Listing 4: The client expresses four preferences with four ranges (from 1 to 5). The method selectLocation will compose the itinerary selecting the locations that fits the preferences. For example it could select 3 shopping 1 gastronomy and 1 cultural locations.

```
Var A, B, C, D; client.preference;
A.type=shopping AND A.range=x
B.type=cultural AND B.range=y
C.type=gastronomy AND C.range=w
D.type=nightlife AND D.range=z
PREFERS-WITH-PROBABILITY
∀itinerary.day.timeslot, timeslot.location=selectLocation(A, B,
    C, D);
```

3.3 Knowledge Base

4 Scenarios