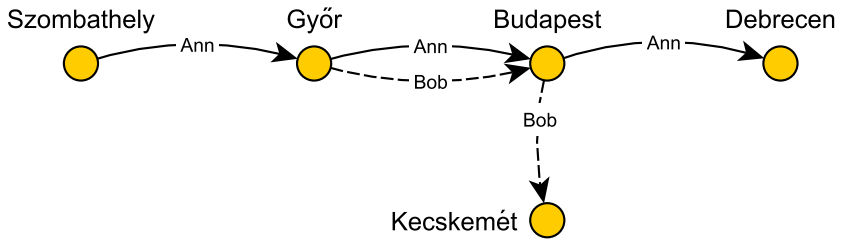
1st Seminar – Structural Modelling – Solutions

We are designing a social transportation service where anyone can announce planned car trips. Others are notified of this through our system and they can join as passengers (even for just a short fragment of the trip) if they share the fuel cost with the owner of the car. A car trip is split into one or more fragments. The owner doesn't necessarily drive throughout the whole trip, it's up for debate, although he or she must be present for every fragment of the trip.
Up until now our service operated in a closed development environment, trips were organised in an ad-hoc manner, and related data weren't saved systematically. We would like to publish our service soon. Information related to trip organizing must be made available on the website, thus we need a way to keep records of such information.

1 Graph-Based Structure Modelling

We would like to design the data model of our system, so we put together some common scenarios based on observations of previous trips.

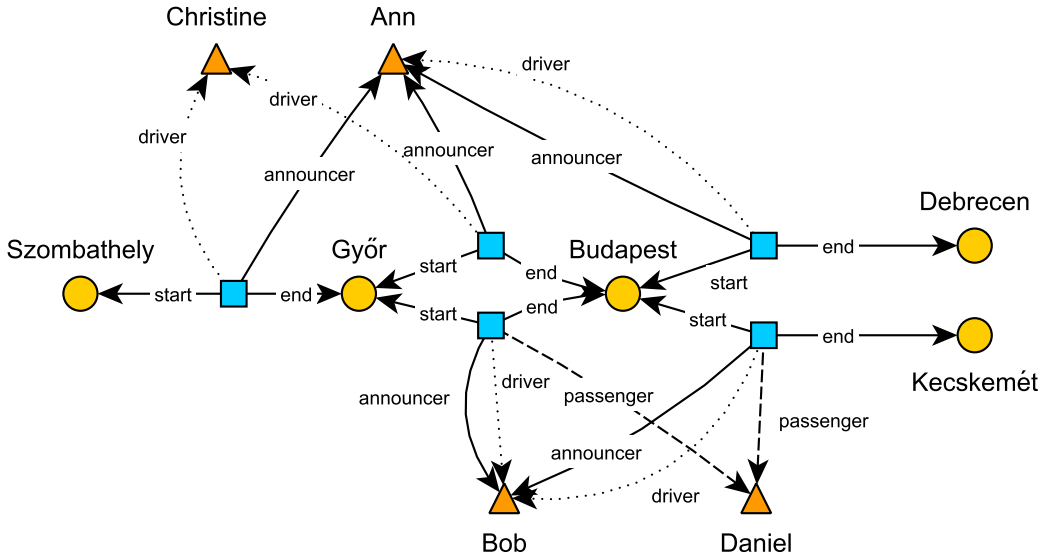
a. Ann plans a trip from Szombathely to Debrecen through Győr and Budapest. Bob takes a trip from Győr to Budapest and then from there to Kecskemét. Construct a graph model based on these scenarios according to the given relations!
Solution
We simply have to create an instance graph based on the scenarios. We can do this in more than one way, for example the cities can be the nodes of the graph and the trip fragments taken by the individual people can be the directed arcs between nodes (labelled with the name of the person). We can also model this in an other way – but more on that later. The dashed arc means that it belongs to Bob, but since it's labelled with Bob's name, it could also be a simple (non-dashed) arc. Throughout the solution's of this seminar we follow the convention to use different types of arcs to denote different relations.



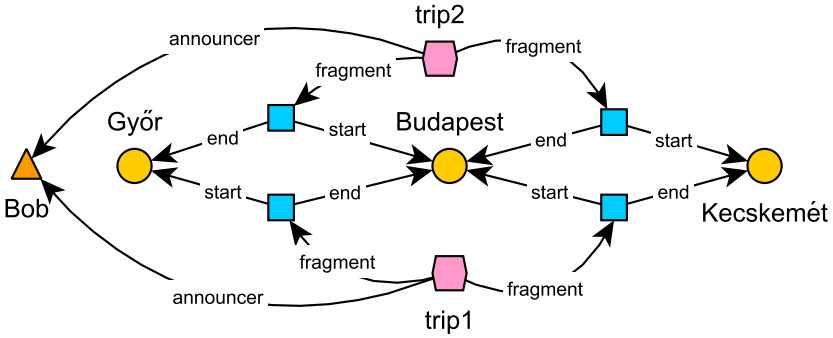
b. Daniel is from Győr and he doesn't own a car. Which graph operation can be used to determine where Daniel can go from Győr by joining others as a passenger? (Suppose that the other drivers are flexible regarding the time of the trip.)
Solution
We would like to find the nodes that are reachable from Győr through directed arcs. If we would like to exclude transfers from one car to another (because in that case we would have to take into account the time of arrival and departure, which are currently not parts of our model) then we have to consider the paths consisting only of arcs of the same label. We can acquire these paths with a graph traversal algorithm, like BFS or DFS. Or in other words: we have to calculate the *transitive closure* of the graph starting from node Győr.

c. Christine decided that she will travel with Ann from Szombathely to Budapest; Daniel requested passage from Győr to Kecskemét. Since Ann worked late last night she would like to sleep after the departure so Christine will drive to Budapest. Bob himself will drive throughout his trip. Extend the graph with this knowledge.
Solution
The start point, end point, driver and passengers of a trip fragment have complex relations between them. In order to represent these relations as a graph we introduce two new node types to represent persons (▲) and trip fragments (■). Then we can connect the other elements to the new nodes. So basically we transformed the previous arcs (*Ann, Bob*) to new nodes and new arcs. It's important to note that there can be multiple relations (arcs) between a trip fragment node

and a person node, like ”announcer”, ”driver” and ”passenger”. The ”driver” relation also means that the person is present during the trip fragment like the passengers (implicit knowledge).
Side note: We could model these relations in an easier way using hypergraphs. Although it's not easy to represent a hypergrapgh graphically.



d. This isn't Bob's first social trip. He announced a trip from Kecskemét to Győr through Budapest. Extend the graph again with this knowledge.
Solution
Now we strictly extend the graph (and reposition some arcs). We introduce a node type to represent trips ( ) that groups the trip fragments (”fragment” relation) and which is connected with its announcer (”announcer” relation). This solution only includes the extended part of the graph, the remaining parts of the previous graph should be modified accordingly.
Side note: Our graph doesn't model temporal properties, i.e. it doesn't distinguish past trips from future trips. But our graph clearly shows that there is no Budapest-Győr-Budapest trip announced.



e. Which graph operation can provide a simplified view of the latest graph, which only shows the announcers, the trips and the fragments of the trips? What will be the properties of the result graph?
Solution
We *filter* the graph based on node types (*Person, Trip, Fragment*) and arc labels (*announcer, fragment*, and maybe *start* and *end*, to also include the cities). The resulting graph will be a tree/forest graph or a DAG, depending whether trips may have common fragments.

2 Attribute Modelling

We gathered some data about previous trips which are summarized in the following table.
After a trip cars and passengers can be rated on a scale from 1 to 5 based on how pleasant they made the journey. The table above includes the ratings gathered so far.

# of ratings	total	category	name	password	plate	smokes	A/C	payment	DL number
6	24	car			ABC-123		no		
17	71	person	Ann	qwe		no			KL2048
16	49	person	Bob	pass		yes			MN4096
14	45	person	Christine	12345		yes		card	
1	5	person	Daniel	friend		no		transfer	
0	0	car			DEF-456		yes		
7	31	person	Emily	2501		no		card	
2	8	person	Francis	appletree		no		Bitcoin	

Table 1: The `Trips` table.

a. Besides the ratings of passengers and cars, available air conditioning and smoking habit can be important factors when someone chooses from the available trips. However the users must not see the driver's password and drivers licence number, and must not know the method of payment used by other passengers. Which operation (known from attribute modelling) can provide the necessary information for this view?
Solution
We need to use the *projection* operation which keeps the desired attributes (columns) and discards everything else.
In SQL:
```
SELECT "# of ratings", "total", "category", "name", "smokes", "A/C"
FROM Trips
```

b. Ranking is based on the average of ratings and not the sum of ratings. Which operation makes it possible to extend the attribute model with the result of this calculation?
Solution
Let's add a new attribute (column) to the table with the name "average rating" which is a *derived attribute* calculated with the following formula (only if at least one rating is available):

$$average\ rating = \frac{total\ rating}{number\ of\ ratings}$$

In SQL:
```
SELECT *, ("total" / "# of ratings") AS "average rating"
FROM Trips
```
This solution carry the risk of dividing by zero. This can be mitigated in a number of ways, the simplest now is to filter the results to keep only the rated elements (see next task).
```
SELECT *, ("total" / "# of ratings") AS "average rating"
FROM Trips
WHERE "# of ratings" > 0
```
An even more elegant solution would be to produce blank cells (N/A) where we divide by zero, but this is out of the scope of System Modelling.

c. Emily and Francis are looking for a passage together. Emily would like a car with a ranking above 4 if possible. Francis considers cars with air conditioning only. Which operation will provide the potential choices?
Solution
We need to use the *filtering* operation which keeps only the records (rows) that match certain criteria. In this case the two criteria are: "average rating $>$ 4" and "A/C $==$ yes".
In SQL (again keeping the rated elements only):
```
SELECT *, ("total" / "# of ratings") AS "average rating"
FROM Trips
WHERE "average rating" > 4 AND "A/C" = "yes" AND "# of ratings" > 0
```

3 Type Modelling

We would like to design a database scheme for our service. In order to do this it's important to distinguish the different types in the system and to look for validation rules.

a. What could be the basic element and connection types according to the text above? Draw a type graph!

Solution

Notice that the arcs labelled with "driver" always goes from nodes representing trip fragments to nodes representing people. It's possible that we have already found two node types and an arc type.

Choosing the node types and arc types are usually straightforward, but the name and direction of arcs can be chosen in many ways. For example, instead of the

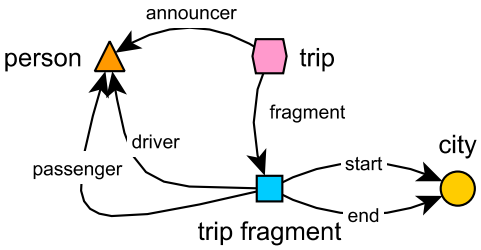
$$(\text{trip fragment}) - [\text{driver}] \rightarrow (\text{person})$$

naming and direction we can use

$$(\text{person}) - [\text{drives}] \rightarrow (\text{trip fragment})$$

or we can even mix them. It's recommended to invest some time into coming up with a consistent naming and directing convention for cases like this.

Based on this a possible solution could be the following:



(Note: the graph doesn't contain the cars of people, we ignore that in this type graph).

b. What could be the classification of the elements described in the table based on their available attributes and relations?

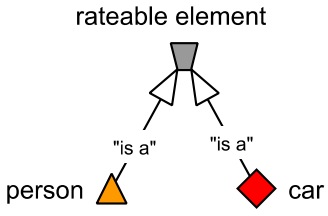
Solution

We can observe that some attributes (e.g. "plate" or "DL number") are defined only for some records. We can come to the same conclusion by inspecting the types of relations in the graph. The two groups (one has "plate" values, the other has "DL number" values) can be grouped by using the value of the "category" attribute. Notice that a rateable element (a row in the table) is fundamentally defined by being either a car or a person and this won't change even later. So we can classify these values ("car" and "person") as two types.

c. Define a type hierarchy based on the previous sub-task!

Solution

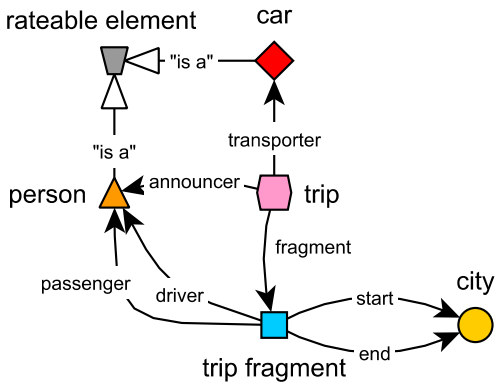
Notice that the "rateable element" is a common generalization of cars and people since we manage them in a similar way regarding some aspects (like rate values). So the "person" and "car" can be considered subtypes of "rateable element".



It would seem evident to distinguish drivers (has a car and drivers licence) from passengers (joins trips and pays for fuel). However, this is a wrong type classification since this can change over time (e.g. Francis can acquire a drivers licence in time and buy a car; Bob may choose to join others as a passenger next time), furthermore it's possible that the driver isn't the owner of

the car (thus the owner is a passenger). Usually the word *type* is used for such cases when the classification of an element doesn't change over time, thus it's not recommended to base a type system on who is the driver and who is a passenger. Although in some cases of trip fragments there are some *roles* such as who are sitting in the car, who is the owner of the car and who drives. The lesson is that types and roles are not the same.

d. (*Bonus task*) Draw a metamodel based on the type graph, type hierarchy and domain of attributes. With what kind of (well-formedness) constraints can we extend the metamodel?
Solution
Since the type graph is a metamodel of some sort, the complete metamodel can be constructed by combining the solutions of subtasks a. and c..

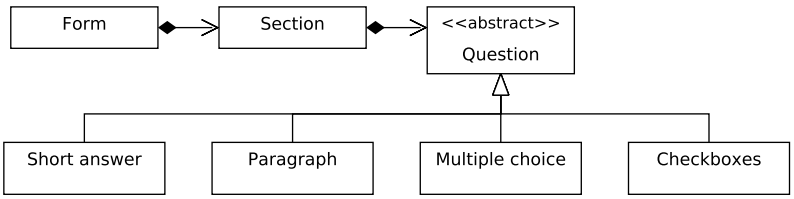


Possible well-formedness constraints:
- Every trip must have at least one fragment.
- The fragments of a trip must be connected to each other, i.e. the third fragment must end where the fourth fragment starts.

We defined the above constraints using informal natural language, although there exists tools in which we can define *formal* constraints using the model elements.

4 Form

Create a form based on the following metamodel through which the passengers can provide feedback about the driver after a trip. For time efficiency reasons most of the feedbacks are gathered with the help of yes/no and multiple choice questions. Passengers can summarize their experience in the form of a short textual opinion.



a. What information do we need to identify the driver?
Solution
A driver can be identified by either the ID card number or by the drivers licence number, but both raise data protection (privacy) questions. A general solution is to generate a trip identification number which is used to store trip related data (start, end, driver, passengers, etc.) in a (private) database. This way the passenger only has to provide the identification number of the trip for which he or she wants to provide feedback.

b. Gather some questions, group them according to their types, then create a model of the finished form.
Solution
Possible questions (without being exhaustive):
- Trip identifier (Short answer)
- Rating of service (from 1 to 5) (Multiple choice)

- Which of the followings need to be improved: leaving/arriving on time, frequency of stops, comfort of seats, facilities of the car (Checkboxes)
- Textual feedback, other experiences (Paragraph)

c. Did we use top-down or bottom-up design?
Solution
In this case we used bottom-up design: starting from the questions through designing the sections we created the form. The metamodel was created using top-down design: starting from the form through the sections we arrived at the questions then we gathered the possible types of questions.

d. If a passenger rates the driver less than 3 in the scale from 1 to 5 then we would definitely like a textual opinion. How can we express this in the model (and in which model)?
Solution
We can use well-formedness constraints in the metamodell against which the instance model can be validated. The informal constraint would be something like: ”if the rating is less than 3 then the textual feedback cannot be empty”.

Bonus Assignment: Implementation

a. Create a data structure (in your favourite programming language) that can represent the contents of our transportation service!
Solution
Initial idea: let’s create a C structure with the attributes of ”trip” and ”trip fragment” and the pointers between them:

```
1    typedef struct {
2    char* user_name;
3    //...
4    BOOL smokes;
5    } Person;
6
7    typedef struct {
8    char* start_city;
9    char* end_city;
10   time_t date;
11   //...
12   Person* array_of_passengers;
13   } Trip_fragment;
14
15   typedef struct {
16   Trip_fragment* fragments;
17   //...
18   } Trip;
19
```

Side note: handling the two types of rateable elements is an interesting problem and design decision. Should we use one `struct` that will contain a lot of `NULL` attribute, or should we use some more elegant method, like class inheritance in C++?

b. Extend the program with a procedure (method) that can enumerate the possible destinations from a given city based on published trips (without changing cars)!

c. Create a smarter version of the previous method that leaves out the passages (on demand) where there would be a smoking passenger during at least one fragment of the trip!