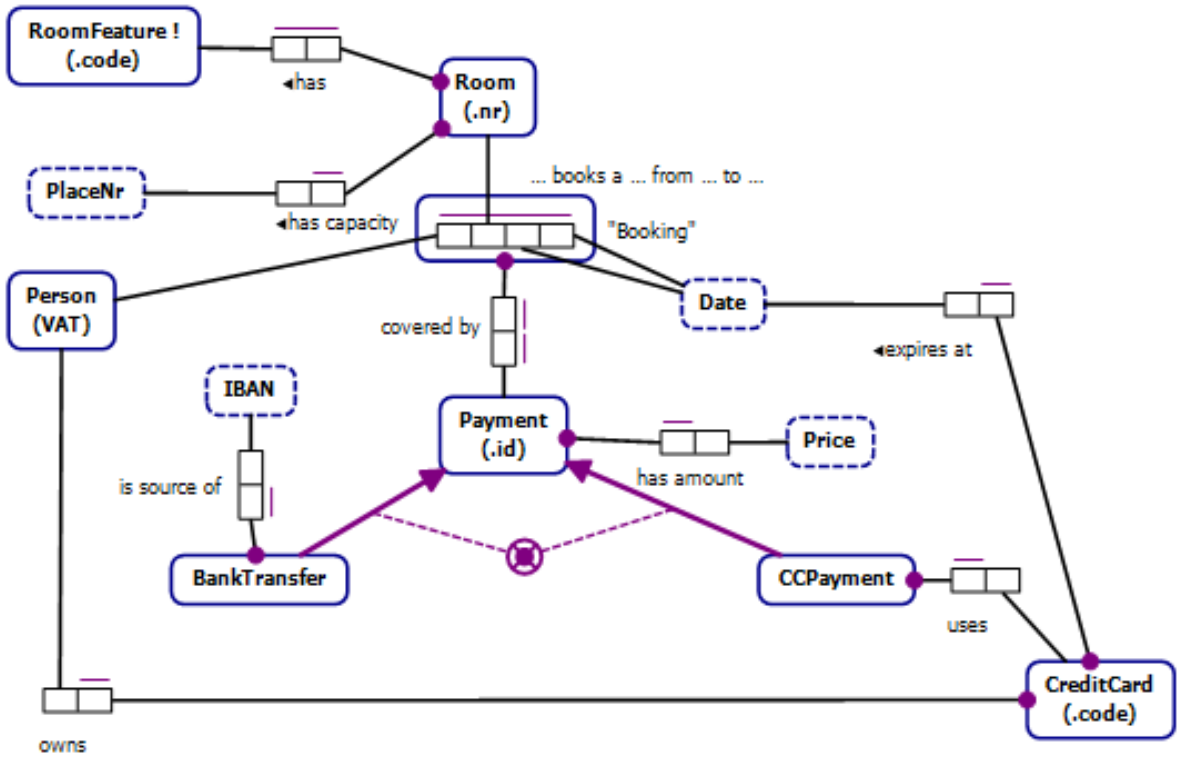
1 Hotel Process

Consider the following ORM schema.



We want to model a process that describes the booking of a room. In particular, we want to model the private process of the hotel's website, dealing only with credit card payments.

The process starts when a request message from a customer is received. When such a message is received, the hotel starts a cyclic subprocess whose aim is to "pre-book" a room. The subprocess starts with a task that *asks booking info* to the user, and then waits an incoming message with such an information. It then executes a task to *check the availability* of the room requested by the user. If there is an available room, it is *pre-booked* by the hotel; otherwise, the flow goes back to the beginning of the subprocess, asking again for booking information.

During the execution of the "pre-book" subprocess, the user can always send a specific message to stop the booking. In this case, the subprocess terminates, and so does the parent process.

When the "pre-book" subprocess correctly terminates (i.e., the room has been pre-booked), a second "payment" cyclic subprocess is entered. The subprocess starts with a task that *shows the room's cost* to the customer, followed by a second task that *asks for credit card details*. Once the details are received through a message from the customer, the hotel simply forwards the details to its trusted bank. The hotel then waits for an answer from the bank. Two different situations may then arise:

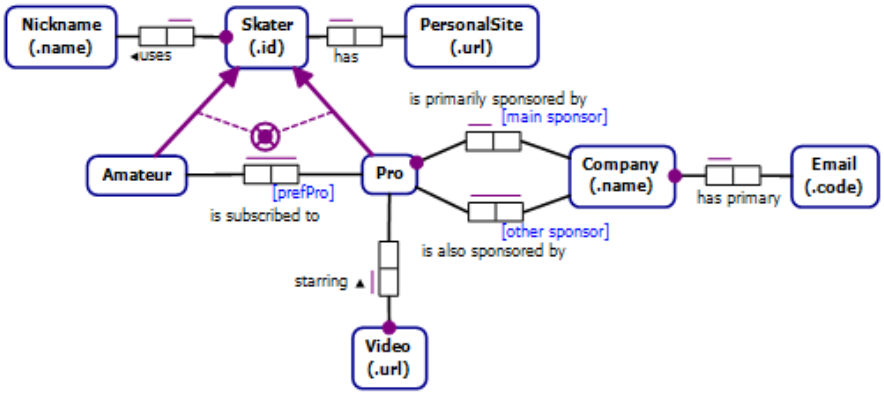
1. An acknowledgement is received from the bank; in that case, the hotel executes a *confirm booking* task, and the subprocess correctly terminates.
2. An error message is received from the bank; in this case, the hotel executes a *show error* task and then goes back to the task in which the credit card details are asked to the customer, or triggers a "failure" error if three attempts have been already made.

If the "payment" subprocess terminates with the "failure" error, the parent process terminates as well. If instead the "payment" subprocess is terminated correctly, the entire process terminates by sending a final message to the customer, containing the payment receipt.

Model the booking process splitting it into different diagrams: one for the main process, and two for the "pre-booking" and "payment" subprocesses.

2 Skaters Process

Consider the following ORM schema.



We focus on the process schema that is used by the web site to register a new professional skateboarder to the web site. The main goal of the process is to register the skateboarder as a "pro" only if his/her primary sponsor certifies that he/she is really a pro.

The process starts when an external user sends a registration request message to the web site, with the intention of registering as a "pro". We assume that the request contains all relevant information of the user, including the name of his/her primary sponsor and also his/her personal e-mail.

The process then executes a *check user info* task in order to verify wether the provided information is correct and can be processed. If not, the process ends by sending a message alerting that the information is wrong. If everything is ok, a *verify matching company* task is executed to check whether the company communicated by the user exists in the web site's database. If not, the process terminates by sending a message alerting that the company does not exist. If instead there exists a matching company, a `verify pro` sub-process is executed. The aim of the sub-process is to interact with the company and certify whether the user is really a pro.

From the external point of view, the `verify pro` sub-process can be abnormally terminated when one between the following two critical events occurs:

1. 12 days have elapsed and the subprocess is still running. In this case, the process executes a *flag company* task to remember that there has been a communication problem with the company. The process then terminates sending a message to the user, alerting that the registration process cannot be completed.
2. An error event is generated by the subprocess, attesting that the user has been refused by the company ("not pro" error). In this case, a *flag user* task is executed, to remember the refusal for future use. The process then terminates informing the user of the refusal via message.

If instead the subprocess is correctly completed, the last phase of the registration is executed. In particular, two tasks are executed in parallel: *register user* to finalize the registration, and *create template page* to create a template web page for the user. When both tasks are completed, the process sends two messages (one to the user and another to the company) to inform them that the registration has been successfully completed, and terminates.

The `verify pro` sub-process starts with the execution of task *ask pro confirmation*. The tasks sends to the company's e-mail all the information related to the user. The sub-process then waits for one of the following incoming events:

1. An accept message, representing a positive answer received from the company. In this case, the subprocess terminates normally.
2. A refuse message, representing a negative answer received from the company. In this case, the sub-process terminates abnormally, in particular triggering the "not pro" error.
3. A deadline of 3 days elapses without an answer from the company. In this case, the sub-process goes back to the *ask pro confirmation* task, attempting again to contact the company.

Model the behavior described above in BPMN, focusing on the private process of the web site (i.e., without showing the company and user pools explicitly). When modeling the main process, depict the `verify pro` subprocess as a collapsed sub-process. Then expand the content of the `verify pro` sub-process separately.