Flight-booking example

|  |
| --- |
| The first [step-by-step example](http://www.opendial-toolkit.net/user-manual/step-by-step-example) introduced the most basic functions of OpenDial. In this second, more extensive example, we look at some more advanced functionalities of OpenDial in the context of a *slot-filling application*. The objective of the dialogue system in this example is to help the user to book flight tickets between Norwegian airports.  [http://www.opendial-toolkit.net/_/rsrc/1437078540916/user-manual/flight-booking-example/flight-booking.png](http://www.opendial-toolkit.net/user-manual/flight-booking-example/flight-booking.png?attredirects=0)  *Example of interaction with the flight-booking system*  The complete dialogue domain designed for the system is available in the file [domains/examples/example-flightbooking.xml](https://github.com/plison/opendial/blob/master/domains/examples/example-flightbooking.xml). As usual, the XML specification of the dialogue domain can be divided into natural language understanding, dialogue management, and generation. In this dialogue domain, these are specified in separate XML files (referenced to in the top XML file example-flightbooking.xml).  The dialogue state contains various variable related to the various slots to fill (the flight destination and departure, the date of travel, the return date if any, and the number of tickets to book). The dialogue proceeds by moving through a sequence of steps, and the current step is recorded in the variable current\_step. The following steps are:   1. Asking the user for the destination 2. Asking the user for the departure 3. Asking the user for the flight date 4. Asking the user whether she/he wants a return trip 5. If yes, asking for the return date 6. Checking the price of the desired flight and informing the user about this price 7. Asking the user whether he wants to book a ticket 8. Asking for the number of tickets to book for the trip 9. Asking the user to confirm all the provided information on last time 10. If the user confirms, booking all the tickets 11. Asking the user whether she/he wants to book additional tickets.   **1. Natural language understanding**  The models for natural language understanding are expressed in [domains/examples/example-flightbooking\_nlu.xml](https://github.com/plison/opendial/blob/master/domains/examples/example-flightbooking_nlu.xml). The file contains two distinct models: one model to recognise the user dialogue act a\_u from the raw user utterance u\_u, and one model to update the four slots (Destination, Departure, Date, ReturnDate and NbTickets) in accordance with this dialogue act a\_u.  *Dialogue act recognition model*  *对话行为识别模型：*  The first model expresses the mapping between the user dialogue act and its underlying user utterance as a probability distribution P(a\_u | u\_u). The user dialogue act associated with each  用户表达：  utterance is represented in the domain as a *set* of elementary dialogue acts (such as "Confirm" or "Inform(ReturnDate, March 28)"). This representation of the dialogue act as a set of elements enables us to capture the fact that an utterance such as "Yes, 3 tickets" contains both a confirmation and a new information about the number of tickets. To allow the dialogue act to contain more than one element, the set operators of the rule include the attribute exclusive=false (see [Advanced modelling: Manipulating sets of elements](http://www.opendial-toolkit.net/user-manual/advanced-modelling-1#5._Manipulating_sets_of_elements) for details).  The dialogue act recognition model relies on complex conditions such as:    <if var="u\_u" relation="contains" value="(to|from)? {Airport}"/>   <if var="Airport" relation="in" value="[Oslo,Bergen,Trondheim,   Stavanger,Moss,Sandefjord,Tromsø,Bodø,Kristiansand,Ålesund]"/>  The above condition checks for mentions of an airport (among the list of available airports) in the user utterance. Note the reliance on regular expression syntax to capture optional or alternative elements, such as (to|from)? (see [Advanced modelling: String Matching](http://www.opendial-toolkit.net/user-manual/advanced-modelling-1#4._String_matching) for details).  The last rule of the model is a lower-priority rule (priority 2, while the other rules have the default priority 1) that is used to specify a default dialogue act in case no other value is found. If the user utterance is an N-best list {I want to fly to Oslo (0.7) ; I want to fly tomorrow (0.2)}, the resulting dialogue act a\_u will be equal to {[Inform(Airport,Oslo] (0.7) ; [Other] (0.2)}.  *Slot-filling model*  The second model captures the mapping between the user dialogue act a\_u and the domain-specific slots Destination, Departure, Date, ReturnDate and NbTickets. The exact slot to fill depends on the current step in the dialogue: if the current step is to ask the user for the destination and the answer is Oslo, the variable Destination will be filled with the value Oslo. If the current step is about the departure, the same will occur for the slot Departure.  The second rule of the model states that if the last system action was a confirmation request on a particular value {Value} for the slot {Slot} and the user utters a confirmation, then the slot should be set to {Value} (in other words, it will increase the probability of {Value}and decrease the probability of alternative hypotheses). The third rule marks a given slot value as certain after the system has uttered a grounding action. Finally, the fourth rule transfers the current distributions for the slots in the absence of additional information.  Note that the slot filling model describes the distributions P(slots|a\_u). An alternative modelling approach would be to model the inverse distribution P(a\_u|slots), as often done in the "user action model" of traditional POMDP approaches. However, the latter approach requires an explicit prior distribution over all possible values for the slots, which would be hard to express in our domain since the number of possible dates for the flights is theoretically infinite.  **2. Dialogue management**  Dialogue management is split into three models, as shown in [domains/examples/example-flightbooking\_dm.xml](https://github.com/plison/opendial/blob/master/domains/examples/example-flightbooking_dm.xml):   * an action selection model to find the best system action to execute given the last user dialogue act, values of the slot to fill, and current step in the interaction. * a transition model that specifies how the selected action modifies the current dialogue state (and especially the current step) * a prediction model for the prior distributions on the destinations, departures and next user dialogue act.   *Action selection model*  This model specifies the utilities of various system actions. At each step during the dialogue, the system has 4 alternative choices:   * Grounding the most likely answer for the current slot and moving on to the next step * Uttering a confirmation request on a particular value for the slot * Asking the user to repeat * Doing nothing.   The initial utility of grounding a particular value X for the slot S is set to -4.5, plus 5 if the slot S is indeed equal to that value X. In other words, this means that in order to ground a given answer, the probability of the slot value needs to be at least 0.9 (since -4.5 + 5\*0.9 = 0), otherwise the grounding action will have a negative utility.  The initial utility of confirmation request is set to -0.3 plus 0.5 if the slot is indeed equal to the value for which the confirmation is requested. In other words, the probability of the slot value needs to be at least 0.6, otherwise the confirmation will have a negative utility.  Finally, asking the user to repeat can be done in all cases, provided that the user utterance is not equal to the None value.  *Transition & prediction models*  The transition model specifies how the selection of a particular system action affects the current dialogue state, in particular the current step in the dialogue. For instance, if the destination slot is being grounded, the system can move to the next slot, namely the departure.  The prediction model expresses the prior likelihoods of departures and destinations, as well the the prior likelihoods of the next user responses given the context. These predictions allows the system to increase the probability of user dialogue acts that are likely given the context, and decrease the probability of unlikely responses. In this example, the prior distributions are fixed by hand, but they could also be estimated from data (see the corresponding section on [Parameter Estimation](http://www.opendial-toolkit.net/user-manual/parameter-estimation)).  **3. Natural language generation**  Finally, the generation models in [domains/examples/example-flightbooking\_nlg.xml](https://github.com/plison/opendial/blob/master/domains/examples/example-flightbooking_nlg.xml)specify the mappings between the current dialogue state and actual system responses. The system can produce new system utterance either when the current step is changed, or when a new system action is selected.  Note that many generation rules rely on templates to fill in certain parts of the generated responses. For instance, the following rule case describes that if the current dialogue step is LastConfirm and the trip to book is a single one-way ticket, the system response can be constructed based on the values in the variables Departure, Destination, Date and TotalCost:  <case>   <condition>     <if var="current\_step" value="LastConfirm" />     <if var="ReturnDate" value="NoReturn" />     <if var="NbTickets" value="1"/>   </condition>   <effect util="1">     <set var="u\_m" value="You are ordering one one-way ticket from  {Departure} to {Destination} on {Date} for a total cost of  {TotalCost} EUR. Shall I confirm your order?" />   </effect> </case>  (see [Advanced modelling: References](http://www.opendial-toolkit.net/user-manual/advanced-modelling-1#1._References) for more details on such references).  It is also worth noting that the generated utterance can be tailored to the context. For instance, the communicative action a\_m = AskRepeat has distinct realisations depending on the current dialogue step and on whether an AskRepeat action has already been executed in the previous turn.  **4. External module**  The flight-booking system needs to access an external database to check the prices of particular routes and perform the ticket booking. The module[src/opendial/modules/examples/FlightBookingExample.java](https://github.com/plison/opendial/blob/master/src/opendial/modules/examples/FlightBookingExample.java) provides a simple, faked example of such a connection. The module monitors the dialogue state and is triggered upon the update of system actions a\_m. When the system action is equal to FindOffer, the module checks the filled values for the destination, departure and dates, and returns the associated price by adding a new action MakeOffer(price) in the dialogue state.  Similarly, if the system action is Book, the system can check the filled values for the destination, departure, dates and number of tickets, and perform the booking. |