|  |  |  |
| --- | --- | --- |
| [User Guide](http://www.opendial-toolkit.net/user-manual)‎ > ‎  Step-by-step example   |  |  | | --- | --- | | Assume you want to create a simple dialogue system for a robot that can be instructed to move in four directions: left, right, forward, backward. When uncertain, the robot should ask the user to repeat the instruction. Probabilistic rules allow us to design such a dialogue system in a straightforward manner.  We focus in this example on the specification of the dialogue domain to process the user inputs and select the most relevant actions depending on the conversational situation. Of course, a real robot will also need to include distinct modules for the actual speech recognition, speech synthesis and motor control. The integration of such modules is described in the sections [Speech Recognition and Synthesis](http://www.opendial-toolkit.net/user-manual/speech-recognition-and-synthesis) and [External modules](http://www.opendial-toolkit.net/user-manual/external-modules).   |  | | --- | | ***Notational conventions***:  In this example (and in OpenDial in general), we shall label all variables related to the user with the suffix \_u (with u standing for "user") and all variables related to the system \_m  (with m standing for "machine"). The variable u\_u is therefore the default label for the user utterance, while u\_m is the default label for the system utterance. These labels can be changed in the system settings.  Similarly (and as conventionally used in the dialogue management literature), the user dialogue act is denoted a\_u, while the system action is denoted a\_m. |   **General skeleton**  We start by creating a new dialogue domain (Domain > New in the menu bar).  If we go to the domain editor tab, we see an empty domain specification:  <domain>  <!-- the domain specification will go here -->  </domain>  Each dialogue domain is constituted of a set of (rule-structured) *models*. A model is essentially a collection of probabilistic rules together with a trigger variable that indicates when the rules should be applied.  In our case, we want to define a model that is triggered when a new user utterance is observed. We will therefore construct a model with the variable u\_u as trigger. The domain specification becomes:  <domain>   <model trigger="u\_u">  <!-- the rule(s) for this model will go here -->  </model>  </domain>  Inside each model is a collection of probabilistic rules.[[1]](http://www.opendial-toolkit.net/user-manual/step-by-step-example#%5B1%5D) Two distinct types of rules can be encoded:   * *Probability rules* express how some state variables (the "input variables" of the rule) affect the values of some other state variables (the "output variables"). In other words, they encode conditional probability distributions of the form P(**O|I**), where **I**represents the input variables and **O** the output variables. * *Utility rules* express the utility of particular actions (from the system perspective) depending on particular input variables. In other words, they encode utility functions of the form U(**A**|**I**) where **I** represents the input variables and **A** the action variables.   **A first rule**  As we want to map user utterances to specific system actions, our first rule will be a utility rule. We can encode the mapping between user utterances u\_u and system utterances u\_m in the following manner:    <rule>       <case>         <condition>           <if var="u\_u" value="turn left" />         </condition>         <effect util="1">           <set var="u\_m" value="OK, turning left!" />         </effect>       </case>       <case>         <condition>           <if var="u\_u" value="turn right" />         </condition>         <effect util="1">           <set var="u\_m" value="OK, turning right!" />         </effect>       </case>       <case>         <condition>           <if var="u\_u" value="move forward" />         </condition>         <effect util="1">           <set var="u\_m" value="OK, moving forward!" />         </effect>       </case>       <case>         <condition>           <if var="u\_u" value="move backward" />         </condition>         <effect util="1">           <set var="u\_m" value="OK, moving backward!" />         </effect>       </case>   </rule>  As we can see, each rule is composed of an ordered list of case elements. Each case is associated with a specific condition and a set of effects (although in this particular rule, there is only one effect in each case). The rule can be read as such:  **if** user input u\_u is equal to "turn left" **then**  the utility of u\_m="OK, turning left" is set to 1.  **else if** user input u\_u is equal to "turn right" **then**  the utility of u\_m="OK, turning right" is set to 1.  **else if** user input u\_u is equal to "move forward" **then**  the utility of u\_m="OK, moving forward is set to 1.  **else if** user input u\_u is equal to "move backward" **then**  the utility of u\_m="OK, moving backward" is set to 1.  **else**  no utility is set.  We can now run OpenDial, open the domain we have designed, and type for instance "turn left" in the chat window. The system response should be "OK, turning left!" since the system will automatically select the action with highest utility.  [http://www.opendial-toolkit.net/_/rsrc/1437077139972/user-manual/step-by-step-example/opendial_step1.png](http://www.opendial-toolkit.net/user-manual/step-by-step-example/opendial_step1.png?attredirects=0)  We can also click on the state viewer and inspects both the current dialogue state (in the form of a Bayesian network) as well as the intermediate states during the state update.  [http://www.opendial-toolkit.net/_/rsrc/1437077155898/user-manual/step-by-step-example/opendial_step2.png](http://www.opendial-toolkit.net/user-manual/step-by-step-example/opendial_step2.png?attredirects=0)  **Language understanding model**  One shortcoming of the current dialogue domain is its rigid range of possible user inputs. User utterances such as "turn to the left" or "now please move forward" are simply ignored by the system. Although we could in principle directly enumerate all possible inputs in the utility rule, a more principled approach is to write a probability rule that converts the user utterance into a logical representation of the user dialogue act (denoted a\_u), and then let the utility model operate on this logical representation.  We will therefore write a new model with one single probability rule and trigger variable u\_u:   <model trigger="u\_u">   <rule>     <case>      <condition operator="or">       <if var="u\_u" value="turn \* left" relation="contains"/>       <if var="u\_u" value="move \* left" relation="contains"/>       <if var="u\_u" value="go \* left" relation="contains"/>     </condition>     <effect prob="1">       <set var="a\_u" value="Request(Left)" />     </effect>     </case>     <case>     <condition operator="or">       <if var="u\_u" value="turn \* right" relation="contains"/>       <if var="u\_u" value="move \* right" relation="contains"/>      <if var="u\_u" value="go \* right" relation="contains"/>     </condition>     <effect prob="1">       <set var="a\_u" value="Request(Right)" />     </effect>    </case>    <case>     <condition operator="or">       <if var="u\_u" value="move \* forward" relation="contains"/>       <if var="u\_u" value="go \* forward" relation="contains"/>       <if var="u\_u" value="go \* straight" relation="contains"/>     </condition>     <effect prob="1">      <set var="a\_u" value="Request(Forward)" />     </effect>    </case>    <case>     <condition operator="or">       <if var="u\_u" value="move \* backward" relation="contains"/>       <if var="u\_u" value="go \* backward" relation="contains"/>     </condition>     <effect prob="1">       <set var="a\_u" value="Request(Backward)" />     </effect>    </case>    <case>     <effect prob="1">       <set var="a\_u" value="None" />     </effect>   </case>   </rule>  </model>  A few things are worth noting in this model. First, the rule conditions are a bit more complicated. Each condition is encoded as a disjunction of basic conditions, as indicated by the attribute operator=or.  One example of basic condition is      <if var="u\_u" value="turn \* left" relation="contains" />  This condition is satisfied whenever the pattern turn \* left is found inside the string of the user utterance u\_u. The attribute relation="contains" indicates that the condition checks whether the pattern is included as a substring of the full utterance (in other words, it performs partial matching). The \* sign indicates a wildcard and can capture any subsequence.[[2]](http://www.opendial-toolkit.net/user-manual/step-by-step-example#%5B2%5D)  The effects themselves specify how output variables (in this case, the user dialogue act a\_u) must be updated. In this case, all effects are deterministic. We will however later encounter rules that are non-deterministic (i.e. they include several alternative effects with distinct probabilities of occurrence).  **Action-selection & generation models**  Instead of directly hard-coding the system utterances as system actions, it is often more appropriate to factor the system decisions in two steps:   1. selection of a high-level logical representation of the next action (denoted a\_m) 2. selection of the best linguistic realisation for this logical action.   The action-selection model will be quite similar to the utility model described earlier, except it operates on the user dialogue act a\_u and select the high-level action a\_m:  <model trigger="a\_u">     <rule>       <case>         <condition>           <if var="a\_u" value="Request(Left)" />         </condition>         <effect util="1">           <set var="a\_m" value="Move(Left)" />         </effect>       </case>       <case>         <condition>           <if var="a\_u" value="Request(Right)" />         </condition>         <effect util="1">           <set var="a\_m" value="Move(Right)" />         </effect>       </case>       <case>         <condition>           <if var="a\_u" value="Request(Forward)" />         </condition>         <effect util="1">           <set var="a\_m" value="Move(Forward)" />         </effect>       </case>       <case>         <condition>           <if var="a\_u" value="Request(Backward)" />         </condition>         <effect util="1">           <set var="a\_m" value="Move(Backward)" />         </effect>       </case>     </rule> </model>  The generation model can be easily constructed with one single utility rule:  <model trigger="a\_m">   <rule>       <case>       <condition>         <if var="a\_m" value="Move(Left)"/>       </condition>       <effect util="1">         <set var="u\_m" value="Ok, turning left!"/>       </effect>     </case>     <case>       <condition>         <if var="a\_m" value="Move(Right)"/>       </condition>       <effect util="1">         <set var="u\_m" value="Ok, turning right!"/>       </effect>     </case>     <case>       <condition>         <if var="a\_m" value="Move(Forward)"/>       </condition>       <effect util="1">         <set var="u\_m" value="Ok, moving forward!"/>       </effect>     </case>     <case>       <condition>         <if var="a\_m" value="Move(Backward)"/>       </condition>       <effect util="1">         <set var="u\_m" value="Ok, moving backward!"/>       </effect>     </case>   </rule> </model>  **Clarification strategies**  The current domain suffers from a lack of robustness in the face of noise and uncertainty. For instance, if the system observes a user utterance such as "u\_u = move left" with a low probability of 0.1 (this can be tested in the chat window by adding the probability in parentheses at the end of the utterance), it will ignore the fact that this instruction is highly uncertain and select Move(Left) as the next action.  A better approach is to only execute the action when a certain probability threshold has been reached. One can add the following rule to the action-selection model:    <rule>       <case>         <effect util="-0.5">           <set var="a\_m" value="Move(\*)"/>         </effect>       </case>   </rule>  This rule will reduce the utility of all Move(\*) actions by 0.5. In other words, this means that these actions will only be executed if their probability is higher than 0.5.  We can also add another system action AskRepeat to request the user to repeat the utterance when faced with uncertainty:    <rule>       <case>         <effect util="0.2">           <set var="a\_m" value="AskRepeat"/>         </effect>       </case>   </rule>  The generation rule should also be extended with another case:  <!-- ...-->   <case>       <condition>         <if var="a\_m" value="AskRepeat"/>       </condition>       <effect util="1">         <set var="u\_m" value="Sorry, could you repeat?"/>       </effect>     </case>  </rule>  We can test the resulting dialogue domain in the OpenDial user interface and verify the resulting system behaviour:  [http://www.opendial-toolkit.net/_/rsrc/1437077197328/user-manual/step-by-step-example/opendial_step3.png](http://www.opendial-toolkit.net/user-manual/step-by-step-example/opendial_step3.png?attredirects=0)  **Prior distributions**  The AskRepeat action included in the current dialogue domain is not very sophisticated. It simply asks the user to repeat but does not "accumulate" evidence over the turns. Assume for instance the dialogue excerpt:  user:    move forward (0.65) system:  Sorry, could you repeat? user:    move forward (0.65)  With the current dialogue domain, the system will again ask the user to repeat. Ideally, the fact that the top hypothesis in both utterances is "move forward" should provide the system with an increased confidence for the hypothesis "move forward".  We can write a rule that encodes the common sense assumption that the user is likely to repeat her/his utterance when asked to do so:  <!-- Prediction on the next user action --> <model trigger="a\_m">     <rule>       <case>         <condition>           <if var="a\_m" value="AskRepeat" />         </condition>         <effect prob="0.95">           <set var="a\_u^p" value="{a\_u}" />         </effect>       </case>     </rule> </model>  The above rule states that when the system requests the user to repeat the instruction, the next user dialogue act is predicted to be identical to the current one with probability 0.95 (the remaining 0.05 covers the cases where the user decides to say something else).  In order to distinguish such prediction on a future event (in this case, the next dialogue act) from actual observed values, OpenDial relies on the convention that predictive variables are denoted with a superscript ^p. A variable X^p represents therefore a prediction on the variable X to be observed in the future.[[3]](http://www.opendial-toolkit.net/user-manual/step-by-step-example#%5B3%5D)  The reader should also note that the effect value is written as {a\_u}. This value is a *reference*to the current value of  the variable a\_u. The curly brackets { } are important: without them, the effect would simply state that a\_u^p must be set to the string "a\_u" instead of referring to the value denoted by the variable.  Thanks to this rule providing a prior distribution over the next dialogue act, the dialogue system is able to accumulate evidence and select the right action to execute:  [http://www.opendial-toolkit.net/_/rsrc/1437077290575/user-manual/step-by-step-example/opendial_step4.png](http://www.opendial-toolkit.net/user-manual/step-by-step-example/opendial_step4.png?attredirects=0)  **Using universal quantifiers**  Some of the rules in the current dialogue domain exhibit recurring patterns: for instance, the utility rule for the system action states that, if the user dialogue act has the form Request(*some action*), the system can execute the corresponding action Move(*some action*) with utility 1.  The expressive power of probabilistic rules can be greatly enriched through the use of logical quantifiers. The rule conditions and effects can indeed be partly underspecified and include free variables. In other words, the mapping between conditions and effects specified by the rule is duplicated for every possible assignment of values for these free variables.[[4]](http://www.opendial-toolkit.net/user-manual/step-by-step-example#%5B4%5D)  The utility rule for the selection of the next system action can consequently be reduced to the following:    <rule>       <case>         <condition>           <if var="a\_u" value="Request({X})" />         </condition>         <effect util="1">           <set var="a\_m" value="Move({X})" />         </effect>       </case>   </rule>  The curly brackets are employed to denote the free variable X. At runtime, OpenDial will determine the set of possible assignments (called *groundings*) for the free variables and duplicate the rule for each of these groundings. Note that free variable labels such as X must not conflict with the label of existing state variables.  **Parameters**  The probabilities and utilities of the current domain are all handcrafted. Although this handcrafted approach may work well in specific cases, it remains vulnerable to human errors and inaccuracies. For instance, the utility of the AskRepeat action (currently set to 0.2) or the probability of a user repetition after such a request (set to 0.95) are just informed guesses, and actual interactions may very well deviate from these expected values.  A more principle, data-driven approach is to associate these rules with parameters whose values must be estimated from data. Both probabilities and utilities can be replaced by parameters in OpenDial.  Since OpenDial adopts a Bayesian approach to parameter estimation, each parameter must be associated with a prior distribution over its (usually continuous) range of values. Several types of parametric distributions are available to this end, such as uniform, Gaussian, and Dirichlet distributions.[[5]](http://www.opendial-toolkit.net/user-manual/step-by-step-example#%5B5%5D)  We can create two parameters for our domain:   * one parameter for the utility of the AskRepeat action. A reasonable prior distribution for this parameter is a Gaussian centered on 0. For the purpose of this example, we shall set this distribution to ~ N(0,5). * one parameter for the probability of the user repetition. As we may expect the user to comply with the system request in most cases, we can encode this probability distribution as a Dirichlet distribution ~ Dir(3,1).   These prior parameter distributions are specified at the top of the domain specification:   <parameters>      <variable id="theta\_repeat">       <distrib type="gaussian">         <mean>0</mean>         <variance>5</variance>       </distrib>     </variable>      <variable id="theta\_repeatpredict">       <distrib type="dirichlet">         <alpha>3</alpha>         <alpha>1</alpha>       </distrib>     </variable>   </parameters>  And the last modification is to replace the fixed values in the probabilistic rules by their parameters:     <rule>       <case>         <effect util="theta\_repeat">           <set var="a\_m" value="AskRepeat" />         </effect>       </case>     </rule>  ...        <rule>       <case>         <condition>           <if var="a\_m" value="AskRepeat" />         </condition>         <effect prob="theta\_repeatpredict[0]">           <set var="a\_u^p" value="{a\_u}" />         </effect>       </case>     </rule>  Note that since Dirichlets are multivariate distributions, the parameter of the second rule must index the dimension of the distribution (the first dimension in this case: [0]).  One can inspect the distribution of the two parameters through the state monitor:  [http://www.opendial-toolkit.net/_/rsrc/1437077620879/user-manual/step-by-step-example/opendial_step5.png](http://www.opendial-toolkit.net/user-manual/step-by-step-example/opendial_step5.png?attredirects=0)  The parameters must then be optimised from dialogue data. This will be covered in the section [Parameter estimation](http://www.opendial-toolkit.net/user-manual/parameter-estimation).  **Final domain**  The full XML specification for the dialogue domain can be found at [domains/examples/example-step-by-step\_fixed.xml](https://github.com/plison/opendial/blob/master/domains/examples/example-step-by-step_fixed.xml) (without the unknown parameters) and [domains/examples/example-step-by-step\_params.xml](https://github.com/plison/opendial/blob/master/domains/examples/example-step-by-step_params.xml) (with the unknown parameters).  [1] Probabilistic rules is used as an umbrella term to cover both probability and utility rules.  [2] See the section [String matching](http://www.opendial-toolkit.net/user-manual/advanced-modelling-1#4._String_matching) for more details on the string matching functionalities implemented in OpenDial.  [3] See [Lison (2014)](http://folk.uio.no/plison/pdfs/thesis/thesis-plison2014.pdf), p. 78-79 for details.  [4] See [Lison (2014)](http://folk.uio.no/plison/pdfs/thesis/thesis-plison2014.pdf), p. 67-68 and 74-76 for details.  [5] See [Lison (2014)](http://folk.uio.no/plison/pdfs/thesis/thesis-plison2014.pdf), p. 91-96 for more details. | |