Document version:	0.1
Last modification:	22/05/12
Author(s):	Filip Jurcicek

UFAL Dialogue state update

The purpose of this document is to describe the principles and implementation of dialogue state update in spoken dialogue systems developed at UFAL, MFF UK, Czech Republic. As a prerequisite, the reader should be familiar with the definition of the UFAL dialogue act scheme as described in UFAL Dialogue act scheme (ufal-dialogue-acts.odt).

Table of Contents

Definition of a dialogue state

In a spoken dialogue system, the purpose of a dialogue state is to track the progress in a dialogue. In its simplest form, the dialogue state can be composed only of variables that represents the goal of the user. A good example of this approach is a tourist information domain where users seeks information about restaurants, bars, and hotels, users can constraint their search by area, price range, stars, and the dialogue system can provide information about the address, the postcode, or the phone number for the selected venue. In this domain, the goal can be composed of the following variables:

venue_type	– defines what type of a venue the user is looking for
area	– desired location of the venue
price_range	– price range of the venue
near	– whether the requested venue should be near another venue
food_type	– type of food served at the venue if the venue type is a restaurant
stars	– number of stars of the venue if the venue type is a hotel

Sometimes these variables are called slots or concepts and their values attributes. In addition to the goal, the dialogue state can store information about what the user requested or wants to confirm, what the system already informed about or what the system confirmed.

The dialogue state update

The UFAL dialogue systems uses the concept of Information State Update, where the state is updated whenever new information is available, either from a user or the system itself or any other partner in the communication. Since the update is defined on the dialogue state, the process of updating the dialogue states is called the dialogue state update. The dialogue state update is best depicted using an influence diagram.

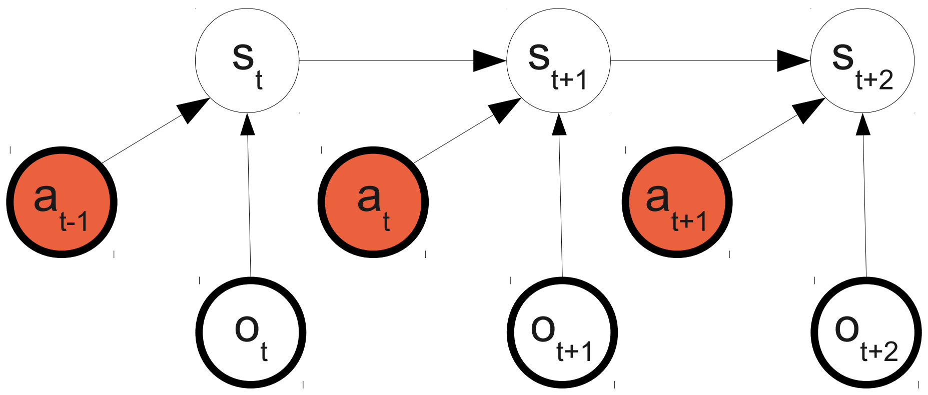

Figure 1: Dialogue state update influence diagram

Shown at Figure 1, the dialogues state $\mathbf{s}_{t+1}$ depends on the previous state $\mathbf{s}_t$, the previous system dialogue act $\mathbf{a}_t$, and the current user dialogue act $\mathbf{o}_{t+1}$. This can be formally written as

$$s_{t+1} \leftarrow s_t + a_t + o_{t+1} \quad .$$

It is assumed that the update is deterministic. Therefore, it must result in one specific dialogue state given the previous state, system dialogue act and the current user dialogue act. Following the example provided in the previous section, an example of the dialogue state update in a tourist information domain can be performed as follows:

Turn 1

Dialogue state:

```
venue_type     = None
area           = None
price_range    = None
near           = None
food_type      = None
stars          = None
```

System says: How may I help you?
System dialogue act: `hello()`

User says: Hi I am looking for a **restaurant**.
User dialogue act: `hello()&inform(venue_type="restaurant")`

Turn 2

Updated dialogue state:

```
venue_type     = restaurant
area           = None
price_range    = None
near           = None
food_type      = None
stars          = None
```

System says: What type of food are you looking for?
System dialogue act: `request(food_type)`

User says: I want a **Chinese** restaurant in a **cheap** price range.
User dialogue act: `inform(venue_type=”restaurant”, food_type=”Chinese”,price_range=”cheap”)`

Turn 3

Updated dialogue state:

```
venue_type     = restaurant
area           = None
price_range    = cheap
near           = None
food_type      = Chinese
stars          = None
```

Note that the value “None” denotes that user did not say anything about the relevant slot and it is a default value for each slot.

Note that the values allowed in the slots in the previous example are very simple. The slots can store only one value out of many. More complex values, such as sets or logic expressions, are not currently supported. If sets are really needed then one can defined slot values as elements of a power set defined for the original slot values.

Recording requests, confirmations and selects by users

So far, only the user inform dialogue act was considered when performing dialogue state update. Other dialogue acts which must be considered are request, confirm, and select.

Requests

Typically a user requests a value of a particular slot if the user is offered a venue and the system did not informed about that slot. The fact that user requests details about some properties of a venue is a new information and should be stored in the dialogue state. Once the dialogue system informs about the requested slot, it should be recorded that the system already provided the requested information. Therefore for storing information about requested slots, specialised slots have to be defined called request history slots. A request history slot has three values: 1) None, 2) user-requested, and 3) system-informed. Just for convenience, all request history slots starts with the prefix 'rh_' and are flowed by the name of the relevant slot.

Assuming that a user can request information about all slot from the previous example, the dialogue state update can continue as follows:

Turn 3

Updated dialogue state:

```
venue_type     = restaurant
area           = None
price_range    = cheap
near           = None
food_type      = Chinese
stars          = None

rh_venue_type  = None
rh_area        = None
rh_price_range = None
rh_near        = None
rh_food_type   = None
rh_stars       = None
```

System says: Char Su is a nice cheap Chinese restaurant.
System dialogue act: `inform(name='Char Su', venue_type="restaurant", food_type="Chinese",price_range="cheap")`

User says: Where is it located?
User dialogue act: `request(area)`

Turn 5

Updated dialogue state:

```
venue_type     = restaurant
area           = None
price_range    = cheap
near           = None
food_type      = Chinese
stars          = None
```

```
rh_venue_type  = None
rh_area        = user-requested
rh_price_range = None
rh_near        = None
rh_food_type   = None
rh_stars       = None
```

System says: Ying Su is in the city centre.
System dialogue act: `inform(name='Char Su', area='centre')`

Turn 6

Updated dialogue state:

```
venue_type     = restaurant
area           = None
price_range    = cheap
near           = None
food_type      = Chinese
stars          = None

rh_venue_type  = None
rh_area        = system-informed
rh_price_range = None
rh_near        = None
rh_food_type   = None
rh_stars       = None
```

Note that the system does not have inform immediately or about all requested slots, therefore the request information is stored and dealt with later if necessary. Once the system informs about the requested slot, it is recorded by storing storing system-informed in the request history slot. This ensures that the system will not inform about it multiple times unless it is asked again.

Confirms

When recording information about what user is trying to confirm more information must be stored. For example, if a user tries to confirm whether the venue is expensive then it does not necessary mean that user is looking for an expensive restaurant. The user just wants to make sure that the offered venue is not in the expensive price range and still keep the goal the same. Therefore, a dialogue state has to store not only what slots user wants to confirm but also what value is confirming in a new set of slots. These slots are called confirm history slots. This practically leads to duplication of all domains slots.

The confirm history slots include all values as the original slots plus the system-informed slot value to record that the system responded to the user confirmation dialogue act.

Assuming that a user can confirm information about all slot from the previous example, the dialogue state update can continue as follows:

Turn 5

Updated dialogue state:

```
venue_type     = restaurant
area           = None
price_range    = cheap
near           = None
food_type      = Chinese
stars          = None

rh_venue_type  = None
rh_area        = user-requested
rh_price_range = None
rh_near        = None
rh_food_type   = None
rh_stars       = None

ch_venue_type  = None
ch_area        = None
ch_price_range = None
ch_near        = None
ch_food_type   = None
ch_stars       = None
```

System says:	Ying Su is in the city centre.
System dialogue act:	`inform(name='Char Su', area='centre')`
User says:	Sorry I guess that I missed that. Is it expensive?
User dialogue act:	`apology()&confirm(price_range=expensive)`

Turn 6

Updated dialogue state:

```
venue_type     = restaurant
area           = None
price_range    = cheap
near           = None
food_type      = Chinese
stars          = None

rh_venue_type  = None
rh_area        = system-informed
```

```
rh_price_range = None
rh_near        = None
rh_food_type   = None
rh_stars       = None

ch_venue_type  = None
ch_area        = None
ch_price_range = expensive
ch_near        = None
ch_food_type   = None
ch_stars       = None
```

System says: No, Char Su is in the cheap price range.
System dialogue act: `inform(name='Char Su', price_range='cheap')`

Turn 6

Updated dialogue state:

```
venue_type     = restaurant
area           = None
price_range    = cheap
near           = None
food_type      = Chinese
stars          = None

rh_venue_type  = None
rh_area        = system-informed
rh_price_range = None
rh_near        = None
rh_food_type   = None
rh_stars       = None

ch_venue_type  = None
ch_area        = None
ch_price_range = system-informed
ch_near        = None
ch_food_type   = None
ch_stars       = None
```

Again note that the system does not have inform immediately or about all confirmed slots, therefore the confirm information is stored and dealt with later if necessary. Once the system informs about the confirmed slot, it is recorded to prevent repetition.

Ontology

The structure of a dialogue state can be defined by some form of ontology. In its simplest form, the ontology can be defined as list of slots and the possible values for the slots.

More complex ontologies can define dependencies between the slots themselves. Some times, not all variable are applicable at all states. For example, the food_type variable is applicable only when the requested venue is a restaurant. Similarly, the variable stars is applicable only when the requested venue is a hotel. Therefore each slot variable can be stores a generic value 'N/A' (not applicable) when the context (the rest of the state) suggest that the content of the slot is irrelevant.

In addition, the ontology can define what the user or the system can talk or ask about.

The ontology for a specific domain in a UFAL dialogue system defines:

- all slots in the domain
- all values for the slots in the domain
- the dependencies between the slots
- what slots can a user request
- what slots can a user confirm
- what slots can a system request
- what slots can a system confirm
- on what slots can a system use the select dialogue act
- what slots can a system inform about

The syntax for the ontology ontology is defined as Python script and using Python syntax.

1. Slots and their values are defined as a dictionary. E.g.

```
slots = {'venue_type':  ['restaurant', 'bar', 'hotel'],
         'area':        ['adenbrooks', 'arbury', ],
         'price_range': ['free', 'cheap', 'moderate', 'expensive'],
         'near':        ['the bakers', 'trinity college'],
         'food_type':   ['chinese', 'indian', 'english'],
         'stars':       ['1', '2', '3', '4', '5']
        }
```

2. Slot attributes stores what operation a user or a system can do the slots: e.g. inform, request, confirm, or select.

```
slots = {'venue_type': ['user_requests',
                   'user_confirms',
                   'system_informs',
                   'system_requests',
                   'system_confirms',
                   'system_selects',
                  ],
   ...
   'near':        ['user_requests',
                   'user_confirms',
                   'system_informs'
                   'system_confirms',
                   'system_selects'],
   ...
  }
```

In the example above, the user and the system can freely talk and ask about the slot venue_type. It is expected that you user knows what type of venue is looking for and that system can freely asks abut it. In the case of the near slot, the situation is more complicated. It is assumed that a user would not be happy if the systems explicitly asked for the nearness of the venue to some specific place. Therefore, the system is not allowed to request this slot (note that the 'system_requests' is missing). Nevertheless, the system can inform about this slot if it decides that it is appropriate and it can confirm or ask the user to select values in the slot, for example once a user started to talk about the nearness of the requested venue.

3. When designing dependencies between slots, it is difficult to do it in a general way that would fit all possible ways of implementing dialogue state update and its approximations. There for the UFAL ontology defines only a very simple dependencies.

a) it is assumed each slot depend on the value in the slot in the previous turn. For example, if user did not say anything about the slot then the slot value are copied from previous turn to the next. This dependency is assumed to be implemented in the dialogue state update algorithm and it does not have any supporting syntax in the ontology.

b) the UFAL ontology applicability dependencies. As already described the food_type slot is applicable only when the requested venue is a restaurant.

The applicability rules are defined as list of tuples where each tuple represent one applicability rule. The applicability tuple has four elements: 1) parent slot name, 2) value in the parent slot, 3) child slot name and 4) applicability. The applicability element has two values i) 'A' – applicable and ii) 'N/A' - not applicable. For example, the tuple ('venue_type','restaurant','food_type', 'A') defines that slot food_type is applicable when the value of the slot venue_type is equal to restaurant. Another example, the tuple ('venue_type','restaurant','stars', 'N/A') defines that slot food_type is applicable when the value of the slot venue_type is equal to restaurant. Some times is more efficient to define what slots are applicable and sometimes what slots are not applicable. It is assumed that if 'A' values are specified for applicability then the default value for not specified slots is 'N/A' and the opposite when 'N/A' values are provided.

```
applicability = [('venue_type','restaurant','food_type', 'A'),
                 ('venue_type','restaurant','stars', 'N/A'),
                 ('venue_type','bar','stars', 'N/A')
]
```

Note that the last two rules could be more efficiently expressed by ('venue_type','hotel','stars', 'A')

Applicability of the dialogue state variables
Name =none