Fluency project tutorial

Jose F Quesada & Jose Luis Pro

Current state

What do we have now?

- With lekta we can create dialogue systems based upon a certain domain.
- We create lexicons, grammar rules, mind structures and generation strategies in order to get that domain fully implemented.
- But all that things are created *ad-hoc* for that domain.
- If we want to implement another different dialogue system we must start again.
- Without being possible to reuse lexicons or grammar rules.
- It's necessary to have a good level in programming skills and some experience in the implementation of grammars.

Dialogue systems implemented in the exercises

Examples

- Session 04: Exercise 01.
  - In integer calculator exercise we have “english numbers” grammar.
  - But mind structure was an “Expression object” (not reusable).
- Session 04: Exercise 02.
  - In domotics assistant exercise we had grammars and lexicon for actions and devices.
  - And mind structures were basically boolean flags to represent devices context state.

Dialogue systems implemented in the exercises

Examples

- Session 04: Exercise 01.
  - In integer calculator exercise we have “english numbers” grammar.
  - But mind structure was an “Expression object” (not reusable).
- Session 04: Exercise 02.
  - In domotics assistant exercise we had grammars and lexicon for actions and devices.
  - And mind structures were basically boolean flags to represent devices context state.

What do we want?

- **General purpose DM:** We would like a dialogue manager reasoning engine domain-independent.
- **Reusability:** Grammar for some parameters should be reusable. For example, a grammar for english dates or numbers can be used in all imaginable domains with almost no difference between them.
- **Script model:** We must simplify the creation of tasks in a dialogue system by means of script templates, easier to implement and debug.
- **Interface friendly:** If we have a large parameter database it's possible for an inexperienced user (in either programming or linguistic skills) to implement a dialogue system that satisfies his needs.

What is “Fluency”?

Fluency features

- Is a lekta based framework for the easy creation of task-oriented dialogue systems.
- Intended to be domain-independent.
- With generic dialogue manager mind structures and strategies.
- Currently, in a very first production stage.
- Subject to design decisions changes if desired.
- Designed to be translated into any language in a simple and comfortable way.
- Implemented in order to have a GUI designing application that can automatically generate fluency compatible code.

Dialogue act annotation definition

Dialogue act annotation is the activity of marking up stretches of dialogue with information about the dialogue acts performed, [...] focused on marking up their communicative functions.*

- So we must classify and mark up all possible user proferences.
- In order to get its communicative function so we can have different dialogue strategies.
- Bunt taxonomy is so exhaustive and complex.
- We have used at first a simplified taxonomy but it can be expanded when needed.

* *Harry Bunt et al. Towards an ISO standard for dialogue act annotation (2010).*

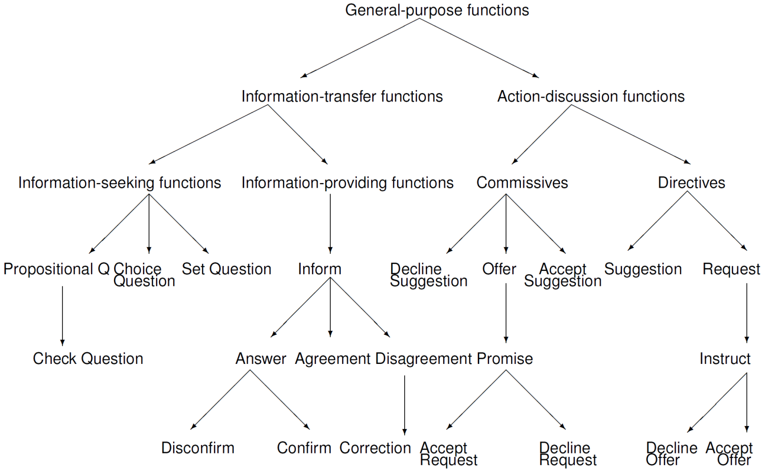
Dialogue act annotation definition

Dialogue act annotation is the activity of marking up stretches of dialogue with information about the dialogue acts performed, [...] focused on marking up their communicative functions.*

- So we must classify and mark up all possible user proferences.
- In order to get its communicative function so we can have different dialogue strategies.
- Bunt taxonomy is so exhaustive and complex.
- We have used at first a simplified taxonomy but it can be expanded when needed.

* *Harry Bunt et al. Towards an ISO standard for dialogue act annotation (2010).*

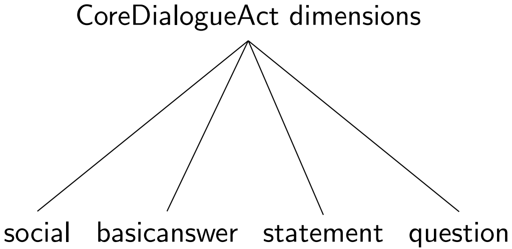
Bunt dialogue act taxonomy



Harry Bunt et al. Towards an ISO standard for dialogue act annotation (2010).

Fluency dialogue act taxonomy

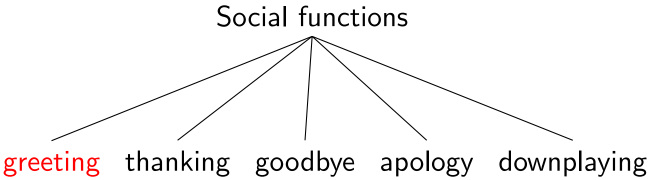
```
classDef:StructureComplex
(
    CoreDialogueAct :
    (
        Dimension,
        Function
    )
)
```



CoreDialogueAct dimensions

social basicanswer statement question

Social dimension



Social functions

greeting thanking goodbye apology downplaying

U: Good morning.

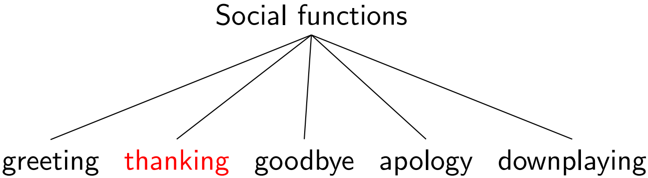
U: Nice to meet you.

U: Hello!

CoreDialogueAct:

```
[Dimension:   social ]
[ Function:  greeting]
```

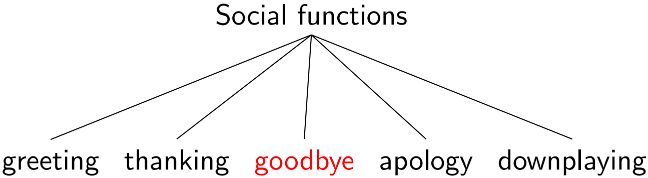
Social dimension



```
U: Thanks!

U: Thank you very much.

U: I'm so thankful!
```

```
CoreDialogueAct:

[Dimension:   social  ]
[Function:   thanking ]
```

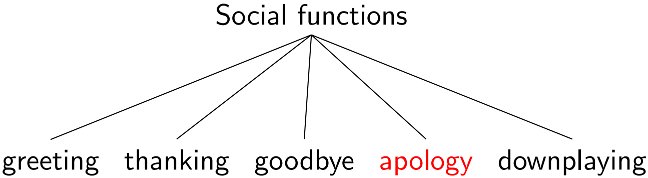
Social dimension



```
U: Have a good day!

U: See you later.

U: Bye!
```

```
CoreDialogueAct:

[Dimension:   social ]
[ Function:   goodbye]
```

Social dimension



U: I'm sorry!

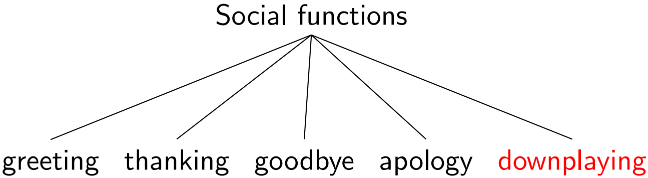
U: Excuse me.

U: My sincere apologies.

CoreDialogueAct:

$$\begin{bmatrix} \texttt{Dimension:} & \texttt{social} \\ \texttt{Function:} & \texttt{apology} \end{bmatrix}$$

Social dimension



Social functions

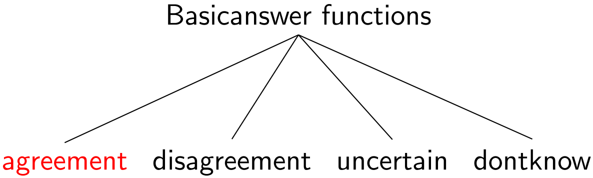
greeting thanking goodbye apology downplaying

```
U: You're welcome!

U: Not at all.

U: Don't mention it.
```

`CoreDialogueAct:`

$$\begin{bmatrix}\texttt{Dimension:} & \texttt{social} \\ \texttt{Function:} & \texttt{downplaying}\end{bmatrix}$$

Basicanswer dimension



U: Yes.

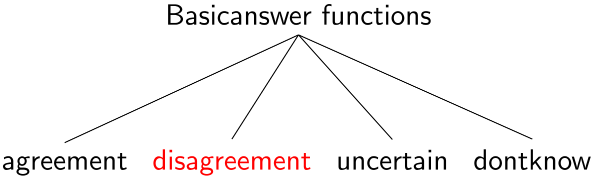
U: Ok.

U: That's right.

CoreDialogueAct:

$$\begin{bmatrix}\texttt{Dimension:} & \texttt{basicanswer}\\ \texttt{Function:} & \texttt{agreement}\end{bmatrix}$$

Basicanswer dimension


Basicanswer functions

agreement disagreement uncertain dontknow

U: No.

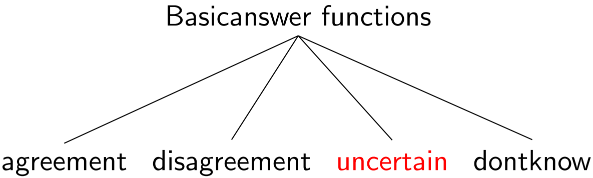
U: No way.

U: I can't agree.

CoreDialogueAct:

$$\begin{bmatrix} \texttt{Dimension:} & \texttt{basicanswer} \\ \texttt{Function:} & \texttt{disagreement} \end{bmatrix}$$

Basicanswer dimension



Basicanswer functions

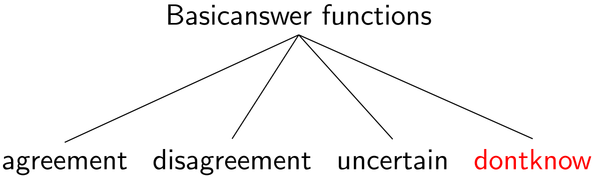
agreement disagreement uncertain dontknow

```
U: Maybe.

U: Perhaps.

U: Probably.
```

`CoreDialogueAct:`

$$\begin{bmatrix}\texttt{Dimension:} & \texttt{basicanswer}\\ \texttt{Function:} & \texttt{uncertain}\end{bmatrix}$$

Basicanswer dimension



Basicanswer functions

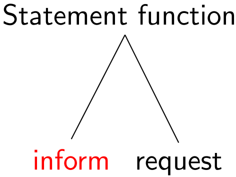
agreement disagreement uncertain dontknow

```
U: I don’t know.

U: I don’t remember.

U: I have no idea.
```

CoreDialogueAct:

$$\begin{bmatrix}\texttt{Dimension:} & \texttt{basicanswer}\\ \texttt{Function:} & \texttt{dontknow}\end{bmatrix}$$

Statement dimension

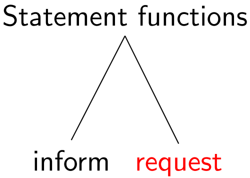


```
U: My name is ...

U: I live in ...

U: I don't want nothing else.
```

CoreDialogueAct:

$$\begin{bmatrix}\texttt{Dimension:} & \texttt{statement}\\ \texttt{Function:} & \texttt{inform}\end{bmatrix}$$

Statement dimension



Statement functions

inform request

U: I want ...

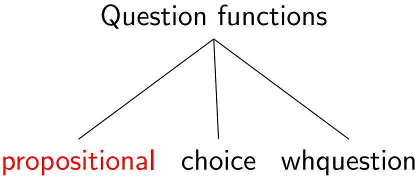
U: I would need ...

U: Would you mind if ...?

CoreDialogueAct:

$$\begin{bmatrix}\texttt{Dimension:} & \texttt{statement}\\ \texttt{Function:} & \texttt{request}\end{bmatrix}$$

Question dimension



S: Do you agree with the date of the appointment?

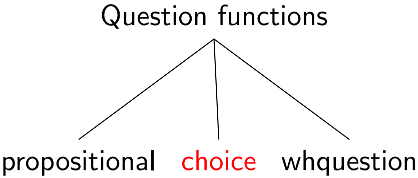
U: Do you like basketball?

U: Do you I am right?

CoreDialogueAct:

$$\begin{bmatrix} \texttt{Dimension:} & \texttt{question} \\ \texttt{Function:} & \texttt{propositional} \end{bmatrix}$$

Question dimension



S: What do you want to do first, make a bank transfer or locate the nearest ATM?

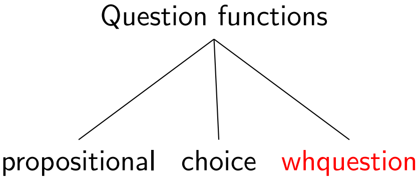
S: Do you prefer coffee or tea?

S: When do you want the appointment, in the morning or in the afternoon?

`CoreDialogueAct:`

$$\begin{bmatrix}\texttt{Dimension:} & \texttt{question}\\ \texttt{Function:} & \texttt{choice}\end{bmatrix}$$

Question dimension



Question functions

propositional choice whquestion

U: What time is it?

U: How much is the doctor appointment?

U: Where is the nearest ATM?

CoreDialogueAct:

$$\begin{bmatrix}\texttt{Dimension:} & \texttt{question} \\ \texttt{Function:} & \texttt{whquestion}\end{bmatrix}$$

TaskDialogueAct structure

- Dialogue act annotation is used to know the communicative function of user proferences.
- This doesn't depend on domain.
- But it lacks of semantic information. For example: In `statement-request` pair we wish to know what user wants to do and the object of his desire.
- This kind of information may depend on domain.

```
classDef:StructureComplex (
   TaskDialogueAct : ( Action, Scope ) )

classDef:StructureBatch (  Action : ( ActionDomain ) )

classDef:ElementLiteral (  ActionDomain )
classDef:ElementLiteral (  Scope )
```

TaskDialogueAct structure

- Dialogue act annotation is used to know the communicative function of user proferences.
- This doesn't depend on domain.
- But it lacks of semantic information. For example: In `statement-request` pair we wish to know what user wants to do and the object of his desire.
- This kind of information may depend on domain.

```
classDef:StructureComplex (
   TaskDialogueAct : ( Action , Scope ) )

classDef:StructureBatch (  Action : ( ActionDomain ) )

classDef:ElementLiteral (  ActionDomain )
classDef:ElementLiteral (  Scope )
```

Domains implemented

So we have implemented a couple of domains to do some testings:

Medical appointment

- Task 1: `ActionDomain = 'book'. Scope = 'appointment'`

Banking management

- Task 1: `ActionDomain = 'consult'. Scope = 'bankaccount'`
- Task 2: `ActionDomain = 'locate'. Scope = 'atm'`
- Task 3: `ActionDomain = 'execute'. Scope = 'transfer'`

Domains implemented

So we have implemented a couple of domains to do some testings:

Medical appointment

- Task 1: `ActionDomain = ‘book’. Scope = ‘appointment’`

Banking management

- Task 1: `ActionDomain = ‘consult’. Scope = ‘bankaccount’`
- Task 2: `ActionDomain = ‘locate’. Scope = ‘atm’`
- Task 3: `ActionDomain = ‘execute’. Scope = ‘transfer’`

Verb lemmas

To detect actions in understanding stage, we associate some verbs lemmas to a certain action:

ActionDomain = 'book'

book, establish, have, **make**, get, schedule, ask, set up, ...

```
U: I want to get an appointment.
U: I would like to make medical appointment.
```

ActionDomain = 'execute'

make, move, execute, perform, do, accomplish, fulfill, effectuate, carry out, complete, ...

```
U: I want to perform a bank transfer.
U: I would like to make a transfer.
```

Please note that a verb lemma can be associated with more than one `ActionDomain` (ambiguities everywhere!).

Verb lemmas

To detect actions in understanding stage, we associate some verbs lemmas to a certain action:

ActionDomain = ‘book’

book, establish, have, **make**, get, schedule, ask, set up, ...

```
U: I want to get an appointment.
U: I would like to make medical appointment.
```

ActionDomain = ‘execute’

make, move, execute, perform, do, accomplish, fulfill, effectuate, carry out, complete, ...

```
U: I want to perform a bank transfer.
U: I would like to make a transfer.
```

Please note that a verb lemma can be associated with more than one `ActionDomain` (ambiguities everywhere!).

Verb lemmas

To detect actions in understanding stage, we associate some verbs lemmas to a certain action:

ActionDomain = ‘book’

book, establish, have, **make**, get, schedule, ask, set up, ...

```
U: I want to get an appointment.
U: I would like to make medical appointment.
```

ActionDomain = ‘execute’

make, move, execute, perform, do, accomplish, fulfill, effectuate, carry out, complete, ...

```
U: I want to perform a bank transfer.
U: I would like to make a transfer.
```

Please note that a verb lemma can be associated with more than one `ActionDomain` (ambiguities everywhere!).

Parameters

A ‘parameter’ is some kind of useful information to complete a task. For example a ‘datetime’ or an ‘accountnumber’.

```
classDef:StructureComplex
(
    Parameter :
    (
        ParameterCategory, // 'terminal', 'and', 'or', ...
        ParameterType,     // 'datetime', 'accountnumber', ...
        ParameterValue,    // Similar to math expressions
        ParameterOperand1,
        ParameterOperand2
    )
)
```

Example of ParameterCategory

U: I want an appointment for tomorrow or the day after tomorrow.

U: My telephone numbers are 1234 and 5678.

Parameters

A ‘parameter’ is some kind of useful information to complete a task. For example a ‘datetime’ or an ‘accountnumber’.

```
classDef:StructureComplex
(
    Parameter :
    (
        ParameterCategory, // 'terminal', 'and', 'or', ...
        ParameterType,     // 'datetime', 'accountnumber', ...
        ParameterValue,    // Similar to math expressions
        ParameterOperand1,
        ParameterOperand2
    )
)
```

Example of ParameterCategory

```
U: I want an appointment for tomorrow or the day after tomorrow.

U: My telephone numbers are 1234 and 5678.
```

Parameters

This information is provided by the user and may be compulsory (red) or optative (green)

Examples

- BookAppointment task:
  - medicalspeciality
  - countryplace
  - phonenumber
  - peselnumber (Ok, it's a polish medical appointment!)
  - datetime
- ConsultBankaccount task:
  - accountnumber
- LocateAtm task:
  - countryplace
- ExecuteTransfer task:
  - accountnumber
  - moneyamount

Parameters classification

- Parameteres can be classified depending upon its domain.
- If it's domain-independent we say that the parameter belongs to "kernel" domain.
- But take into account that we can move a parameter from its domain to kernel domain in order to make it **reusable**.

Kernel domain implemented parameters

- countryplace
- datetime
- letter
- moneyamount
- number
- ordinal
- phonenumber
- signchunk

Parameters classification

- Parameteres can be classified depending upon its domain.
- If it's domain-independent we say that the parameter belongs to "kernel" domain.
- But take into account that we can move a parameter from its domain to kernel domain in order to make it **reusable**.

Kernel domain implemented parameters

- `countryplace`
- `datetime`
- `letter`
- `moneyamount`
- `number`
- `ordinal`
- `phonenumber`
- `signchunk`

Parameters classification

Medical appointment domain implemented parameters

- `medicalspeciality`
- `peselnumber`

Banking management domain implemented parameters

- `accountnumber`

Parameters example: datetime

```
classDef:StructureComplex (
      DateTime: (
      BaseDate,
      OffsetDate,
      MinDate,
      MaxDate,
      GeneralTime
   )
)

classDef:StructureComplex (
      GeneralTime: (
      BaseTime,
      OffsetTime,
      MinTime,
      MaxTime
   )
)
```

Parameters example: datetime

U: Starting next thursday until 3pm to the day after 25 of august from noon to a quarter to nine in the afternoon.

```
(DateTime:
   (MinDate:(GeneralTime:(MaxTime:(BaseTime:(Hour:15))),
             OffsetDate :(DirectionOfTime:'forward',
                          Date           :(DayInWeek:4),
                          DayInWeekOffset:1)),
    MaxDate:(GeneralTime:(MinTime:(BaseTime:(Hour:12)),
                          MaxTime:(BaseTime:(Hour:20,
                                                    Minute:45))),
             OffsetDate :(DirectionOfTime:'forward',
                          Date           :(Day:1)),
             BaseDate   :(Day  :25,
                          Month:8))))))
```

Parameters example: datetime

U: Starting next thursday until 3pm to the day after 25 of august from noon to a quarter to nine in the afternoon.

```
(DateTime:
   (MinDate:(GeneralTime:(MaxTime:(BaseTime:(Hour:15))),
             OffsetDate :(DirectionOfTime:'forward',
                          Date           :(DayInWeek:4),
                          DayInWeekOffset:1)),
    MaxDate:(GeneralTime:(MinTime:(BaseTime:(Hour:12)),
                          MaxTime:(BaseTime:(Hour:20,
                                             Minute:45))),
             OffsetDate :(DirectionOfTime:'forward',
                          Date           :(Day:1)),
             BaseDate   :(Day  :25,
                          Month:8))))))
```

Parameters example: countryplace

```
classDef:StructureComplex (
      CountryPlace : (
      CountryName,
      CountryZone,
      CountryRegion,
      CountryProvince,
      CountryTown
   )
)

classDef:ElementLiteral (
   CountryName,
   CountryZone,
   CountryRegion,
   CountryProvince,
   CountryTown
)
```

Parameters example: `countryplace`

- We have used **NUTS** (Nomenclature of Territorial Units for Statistics) and **LAU** (Local Administrative Unit), two standards developed by European Union.
- We have, in the lexicon, all cities and towns of countries belonging to EU (except UK whose format file is different as usual!).
- This lexicon is expressed in the local language so we have Sevilla, but not Seville. We have Warszawa but not Warsaw.

Some examples

- France: 39096 entries.
- Germany: 11167 entries.
- Bulgary: 10532 entries.
- Spain: 8837 entries.
- Italy: 8161 entries.
- Poland: 2478 entries.

Parameters example: `countryplace`

- We have used **NUTS** (Nomenclature of Territorial Units for Statistics) and **LAU** (Local Administrative Unit), two standards developed by European Union.
- We have, in the lexicon, all cities and towns of countries belonging to EU (except UK whose format file is different as usual!).
- This lexicon is expressed in the local language so we have Sevilla, but not Seville. We have Warszawa but not Warsaw.

Some examples

- France: 39096 entries.
- Germany: 11167 entries.
- Bulgary: 10532 entries.
- Spain: 8837 entries.
- Italy: 8161 entries.
- Poland: 2478 entries.

DialogueAct structure

So we define this dialogue act type in Fluency:

```
classDef:StructureComplex
(
    DialogueAct :
    (
        CoreDialogueAct,
        TaskDialogueAct,
        Parameters
    )
)

classDef:StructureBatch
(
    Parameters :
    (
        Parameter
    )
)
```

Dialogue act annotation full example

U: I want to book an appointment for tomorrow to the dentist.

```
(DialogueAct:
   (CoreDialogueAct:(Dimension:'statement',
                     Function :'request')),
   (TaskDialogueAct:(Action:{(ActionDomain:'book')}
                     Scope:'appointment')),
   (Parameters:     {(Parameter:
                      (ParameterValue:
                       (DateTime:(OffsetDate:(DirectionOfTime:'forward',
                                              Date           :(Day:1))))),
                       ParameterCategory:'terminal',
                       ParameterType    :'datetime'),
                     (Parameter:
                      (ParameterValue:
                       (MedicalSpeciality:(SpecialityName:'Orthodontics'),
                       ParameterCategory:'terminal',
                       ParameterType    :'medicalspeciality')})
```

Dialogue act annotation full example

U: I want to book an appointment for tomorrow to the dentist.

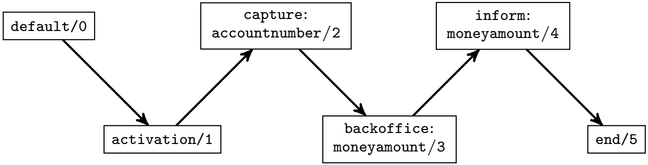
```
(DialogueAct:
   (CoreDialogueAct:(Dimension:'statement',
                     Function :'request')),
   (TaskDialogueAct:(Action:{(ActionDomain:'book')}
                     Scope:'appointment')),
   (Parameters:     {(Parameter:
                      (ParameterValue:
                       (DateTime:(OffsetDate:(DirectionOfTime:'forward',
                                              Date           :(Day:1))))),
                       ParameterCategory:'terminal',
                       ParameterType    :'datetime'),
                     (Parameter:
                      (ParameterValue:
                       (MedicalSpeciality:(SpecialityName:'Orthodontics'),
                       ParameterCategory:'terminal',
                       ParameterType    :'medicalspeciality')})
```

Script model

- Every task in all domains are modelled like scripts.
- A script consist of four parts:
  - **Descriptor:** Literal identificator to distingush this script among others.
  - **Trigger:** Information that user must provide in order to activate the script.
  - **InfoItems:** Place where the information is stored when script is activated. Can be provided by the user or the system itself.
  - **Phases:** Subtasks needed to be performed in order to execute bigger one.
    - Every phase has a priority level (0 the highest priority).
    - Election mode: If all phases of level *n* are finished we select a *n+1* level phase randomly.
- All scripts and states are stored in mindboard structures.

Script model: ConsultBalance script

```
Descriptor: 'ConsultBalance'

Trigger.CoreDialogueAct: ('statement', 'request')
Trigger.ActionDomain: 'consult'
Trigger.Scope: 'bankaccount'
Trigger.Parameter: 'accountnumber'

InfoItem.Parameter1: 'accountnumber' // Provided by the user
InfoItem.Parameter2: 'moneyamount' // Provided by the system
```



default/0

activation/1

capture: accountnumber/2

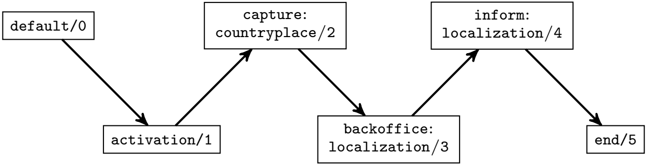
backoffice: moneyamount/3

inform: moneyamount/4

end/5

Script model: LocateAtm script

```
Descriptor: 'LocateAtm'

Trigger.CoreDialogueAct: ('statement', 'request')
Trigger.ActionDomain: 'locate'
Trigger.Scope: 'atm'
Trigger.Parameter: 'countryplace'

InfoItem.Parameter1: 'countryplace' // Provided by the user
InfoItem.Parameter2: 'localization' // Provided by the system
```



default/0

activation/1

capture: countryplace/2

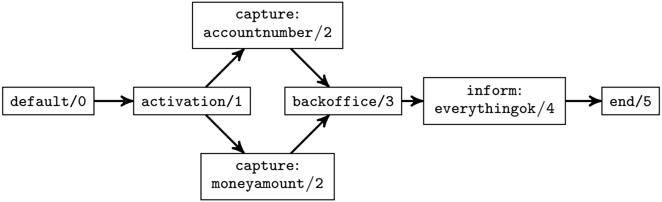
backoffice: localization/3

inform: localization/4

end/5

Script model: MakeTransfer script

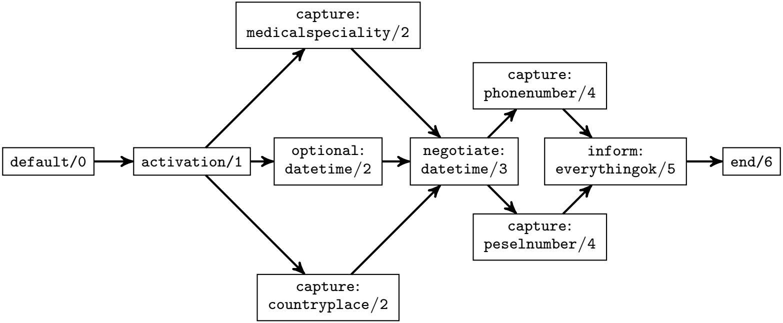
```
Descriptor: 'MakeTransfer'

Trigger.CoreDialogueAct: ('statement', 'request')
Trigger.ActionDomain: 'execute'
Trigger.Scope: 'transfer'
Trigger.Parameter1: 'accountnumber'
Trigger.Parameter2: 'moneyamount'

InfoItem.Parameter1: 'accountnumber' // Provided by the user
InfoItem.Parameter2: 'moneyamount'   // Provided by the user
```



Script model: BookAppointment script

```
Descriptor: 'BookAppointment'

Trigger.CoreDialogueAct: ('statement', 'request')
Trigger.ActionDomain: 'book'
Trigger.Scope: 'appointment'
Trigger.Parameter1: 'medicalspeciality'
Trigger.Parameter2: 'countryplace'
Trigger.Parameter3: 'datetime'
Trigger.Parameter4: 'phonenumber'
Trigger.Parameter5: 'peselnumber'

InfoItem.Parameter1: 'medicalspeciality'
InfoItem.Parameter2: 'countryplace'
InfoItem.Parameter3: 'datetime'
InfoItem.Parameter4: 'phonenumber'
InfoItem.Parameter5: 'peselnumber'
```

Script model: BookAppointment script



Fluency DM phases

Every dialogue system turn execute this loop until last phase is reached:

1. Start talking.
2. Digest expectatives.
3. Digest search scripts.
4. Activate scripts.
5. Select current script.
6. Review states.
7. Select current node.
8. Process talking.
    1. Execute node (go to 5).
    2. Wait node (go to 9).
9. Close talking (go to 1 after user turn).

Fluency DM phases

1. Start talking

- Erase output mindboard structures.

2. Digest expectatives

- Here we convert some parameters to the types of expected parameters.
- For example, if we are expecting a phone number and user says a number, we can transform on into the other.

3. Digest search scripts

- We analyze user proferences and create some triggering schemes.
- We give a scoring to every scheme to see its relevance.

Fluency DM phases

1. Start talking

- Erase output mindboard structures.

2. Digest expectatives

- Here we convert some parameters to the types of expected parameters.
- For example, if we are expecting a phone number and user says a number, we can transform on into the other.

3. Digest search scripts

- We analyze user proferences and create some triggering schemes.
- We give a scoring to every scheme to see its relevance.

Fluency DM phases

1. Start talking

- Erase output mindboard structures.

2. Digest expectatives

- Here we convert some parameters to the types of expected parameters.
- For example, if we are expecting a phone number and user says a number, we can transform on into the other.

3. Digest search scripts

- We analyze user proferences and create some triggering schemes.
- We give a scoring to every scheme to see its relevance.

Fluency DM phases

4. Activate scripts

- Depending on triggering schemes from previous phase we select what scripts must be activated.
- This is not trivial:
  - What happens if comes a parameter from an active script, but not the current?
  - What happens if comes a parameter from a non-active script?
  - What happens if we have several triggered scripts with the same scoring?

5. Select current script

- Previous phase ends sorting the activated scripts stack so we select, as current script, the one placed in the top of the stack.
- If we have a recently activated script it's the moment to recover some mid-term memory slots.

Fluency DM phases

4. Activate scripts

- Depending on triggering schemes from previous phase we select what scripts must be activated.
- This is not trivial:
  - What happens if comes a parameter from an active script, but not the current?
  - What happens if comes a parameter from a non-active script?
  - What happens if we have several triggered scripts with the same scoring?

5. Select current script

- Previous phase ends sorting the activated scripts stack so we select, as current script, the one placed in the top of the stack.
- If we have a recently activated script it's the moment to recover some mid-term memory slots.

Fluency DM phases

6. Review states

- For every info item in the current script we review its state.
- For example, some of these states are:
  - **empty**: The info item has no value.
  - **proposed**: System has recently proposed the value of this item to the user.
  - **checking**: We must check if the value of this item is valid and consistent.
  - **captured**: We have recently captured this info item value from the user.
  - **echoed**: This info item has been "echoed" to the user (implicit confirmation).
  - **grounded**: User seems to agree with this info item.
  - **recovered**: The value of this info item has been recently recovered from mid-term memory.

Fluency DM phases

7. Select current node

- Here we select the next node to be executed in current script.
- Let n the lowest priority level in script with not finished nodes.
- Select one of the not finished nodes with that priority level ramdonly.

8. Process talking

- If the selected node is an "**execution**" node, execute that node and go back to select current node stage.
- If the selected node is a "**wait**" node, we must pass the dialogue turn to user.

9. Close talking

- Erase input mindboard structures.

Fluency DM phases

7. Select current node

- Here we select the next node to be executed in current script.
- Let n the lowest priority level in script with not finished nodes.
- Select one of the not finished nodes with that priority level ramdonly.

8. Process talking

- If the selected node is an "**execution**" node, execute that node and go back to select current node stage.
- If the selected node is a "**wait**" node, we must pass the dialogue turn to user.

9. Close talking

- Erase input mindboard structures.

Fluency DM phases

7. Select current node

- Here we select the next node to be executed in current script.
- Let n the lowest priority level in script with not finished nodes.
- Select one of the not finished nodes with that priority level ramdonly.

8. Process talking

- If the selected node is an "**execution**" node, execute that node and go back to select current node stage.
- If the selected node is a "**wait**" node, we must pass the dialogue turn to user.

9. Close talking

- Erase input mindboard structures.

Main folder

```
Fluency
 ├─ Doc Some docs we have been generating
 ├─ Kernel Generic domain
 ├─ Domains Any other domain
 │   ├─ Alter The union of next two domains
 │   ├─ BankingManagement
 │   └─ MedicalAppointments
 ├─ AlterFluency.lkt Main project file
 └─ EnglishAlterFluency.slk File for interpreter
```

Translation tip

There must be a .slk file for every language and every domain.

Main folder

```
Fluency
├── Doc Some docs we have been generating
├── Kernel Generic domain
├── Domains Any other domain
│   ├── Alter The union of next two domains
│   ├── BankingManagement
│   └── MedicalAppointments
├── AlterFluency.lkt Main project file
└── EnglishAlterFluency.slk File for interpreter
```

Translation tip

There must be a `.slk` file for every language and every domain.

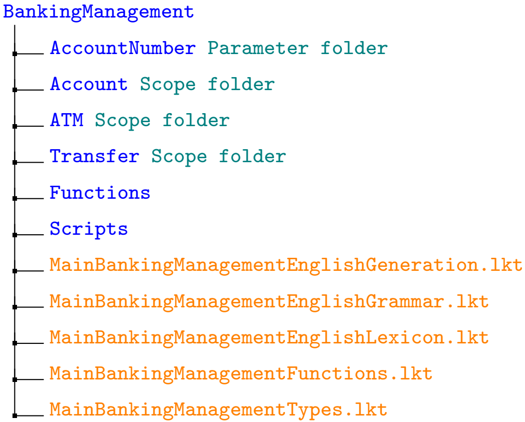
Domain folder example: Banking Management

- BankingManagement
  - AccountNumber Parameter folder
  - Account Scope folder
  - ATM Scope folder
  - Transfer Scope folder
  - Functions
  - Scripts
  - MainBankingManagementEnglishGeneration.lkt
  - MainBankingManagementEnglishGrammar.lkt
  - MainBankingManagementEnglishLexicon.lkt
  - MainBankingManagementFunctions.lkt
  - MainBankingManagementTypes.lkt

Translation tip

There must be three “index” files (grammar, lexicon and NLG) for every language.

Domain folder example: Banking Management

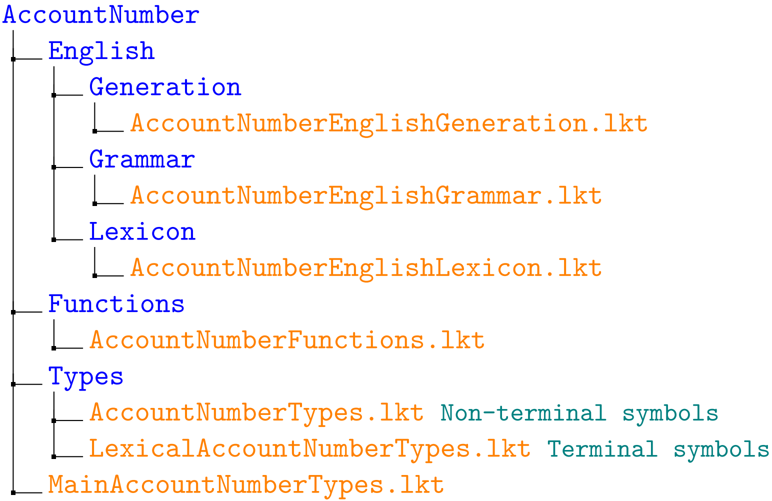


BankingManagement
- AccountNumber Parameter folder
- Account Scope folder
- ATM Scope folder
- Transfer Scope folder
- Functions
- Scripts
- MainBankingManagementEnglishGeneration.lkt
- MainBankingManagementEnglishGrammar.lkt
- MainBankingManagementEnglishLexicon.lkt
- MainBankingManagementFunctions.lkt
- MainBankingManagementTypes.lkt

Translation tip

There must be three “index” files (grammar, lexicon and NLG) for every language.

Parameter folder example: Account Number



```
AccountNumber
├── English
│   ├── Generation
│   │   └── AccountNumberEnglishGeneration.lkt
│   ├── Grammar
│   │   └── AccountNumberEnglishGrammar.lkt
│   └── Lexicon
│       └── AccountNumberEnglishLexicon.lkt
├── Functions
│   └── AccountNumberFunctions.lkt
├── Types
│   ├── AccountNumberTypes.lkt Non-terminal symbols
│   └── LexicalAccountNumberTypes.lkt Terminal symbols
└── MainAccountNumberTypes.lkt
```

Parameter folder example: Account Number

Translation tip

There must be a folder for the three parts of the parameter (grammar, lexicon and NLG) for every language.

Parameter functions: `AccountNumberFunctions.lkt`

- Possibilities of grouping for this parameter.
    - `canMergeWithSequentialAccountNumber`.
    - `canMergeWithOrAccountNumber`.
    - `canMergeWithAndAccountNumber`.
- Valid formats for this parameters: `getParameterFormatsAccountNumber`.
- Additional checks not related with formats: `checkAccountNumberValidity`.
- Conversions from other parameters
    - `convertNumberToAccountNumber`.
    - `convertSignChunkToAccountNumber`.
    - ...

Parameter folder example: Account Number

Translation tip

There must be a folder for the three parts of the parameter (grammar, lexicon and NLG) for every language.

Parameter functions: `AccountNumberFunctions.lkt`

- Possibilities of grouping for this parameter.
  - `canMergeWithSequentialAccountNumber`.
  - `canMergeWithOrAccountNumber`.
  - `canMergeWithAndAccountNumber`.
- Valid formats for this parameters: `getParameterFormatsAccountNumber`.
- Additional checks not related with formats: `checkAccountNumberValidity`.
- Conversions from other parameters
  - `convertNumberToAccountNumber`.
  - `convertSignChunkToAccountNumber`.
  - ...

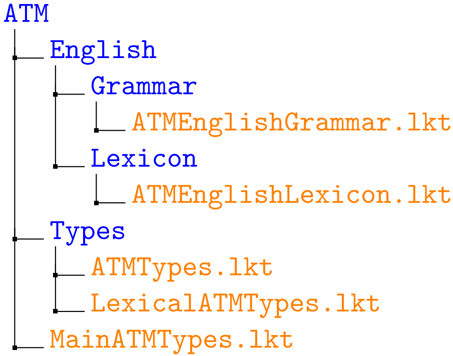
Format example

```
ParameterFormats getParameterFormatsAccountNumber() {
   // Spanish account number
   ParameterFormats ret;
   ParameterFormat format;

   format <- '####################';
   BatchInsertEnd(ret, format);

   format <- 'es######################';
   BatchInsertEnd(ret, format);

   return ret;
}

/* Possible mask formats
#    Any valid number
^    Any letter
@    Any letter or number
*    Anything */
```

checkValidity example: PESEL number

```
// https://en.wikipedia.org/wiki/PESEL

bool checkPeselNumberValidity( string pesel )
{
   int A <- ShapeToInt(LiteralPositionValue(pesel,1));
   int B <- ShapeToInt(LiteralPositionValue(pesel,2));
   int C <- ShapeToInt(LiteralPositionValue(pesel,3));
   int D <- ShapeToInt(LiteralPositionValue(pesel,4));
   int E <- ShapeToInt(LiteralPositionValue(pesel,5));
   int F <- ShapeToInt(LiteralPositionValue(pesel,6));
   int G <- ShapeToInt(LiteralPositionValue(pesel,7));
   int H <- ShapeToInt(LiteralPositionValue(pesel,8));
   int I <- ShapeToInt(LiteralPositionValue(pesel,9));
   int J <- ShapeToInt(LiteralPositionValue(pesel,10));
   int K <- ShapeToInt(LiteralPositionValue(pesel,11));

   int weighted <- 1*A + 3*B + 7*C + 9*D + 1*E + 3*F + 7*G + 9*H + 1*I + 3*J;
   int remainder <- Modulo(weighted, 10);
   int complement <- Modulo(10 - remainder, 10);

   if(complement != K)
      return False;

   return True;
}
```

Scope folder example: ATM

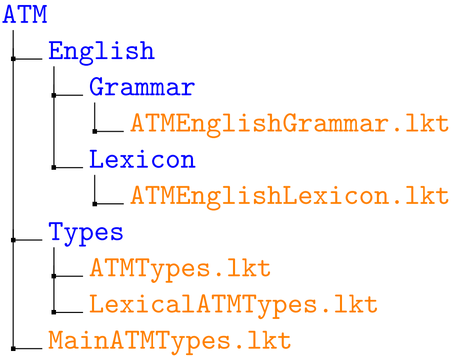


```
ATM
    English
        Grammar
            ATMEnglishGrammar.lkt
        Lexicon
            ATMEnglishLexicon.lkt
    Types
        ATMTypes.lkt
        LexicalATMTypes.lkt
    MainATMTypes.lkt
```

Translation tip

Again `English` folder structure must be replicated in other languages.

Scope folder example: ATM



```
ATM
  English
    Grammar
      ATMEnglishGrammar.lkt
    Lexicon
      ATMEnglishLexicon.lkt
  Types
    ATMTypes.lkt
    LexicalATMTypes.lkt
  MainATMTypes.lkt
```

Translation tip

Again English folder structure must be replicated in other languages.

Scripts folder example: Banking Management

- Scripts
  - English
    - BankingManagementEnglishGeneration.lkt
  - BankingManagementScripts.lkt Functions generating the scripts of this domain

Translation tip

File in English folder has some natural language generation rules related with the scripts.

Scripts folder example: Banking Management

```
Scripts
 └─ English
 │   └─ BankingManagementEnglishGeneration.lkt
 └─ BankingManagementScripts.lkt Functions
    generating the scripts of this domain
```

Translation tip

File in `English` folder has some natural language generation rules related with the scripts.

Functions folder example: Banking Management

```
Functions
├─ BankingManagementBackOffice.lkt Back office
│  callback functions
└─ BankingManagementFunctions.lkt Functions related
   with current domain
```

Functions folder example: Banking Management

Translation tip

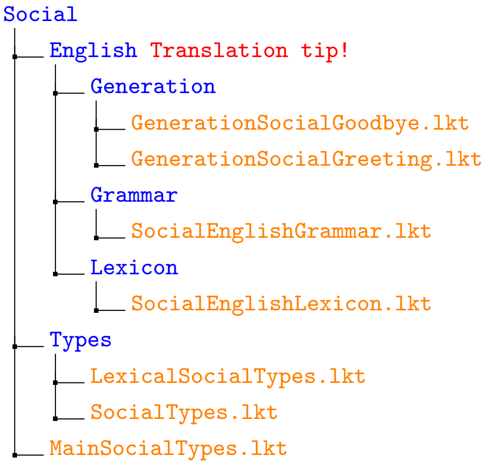
Among other functions, we have to link lemmas with actions:

```
string getActionDomainFromLemmaBankingManagement(string lemma)
{
   switch (lemma)
   {
      // Action consult (scope: 'bankaccount')
      case 'consult'   { return 'consult'; }
      case 'check'     { return 'consult'; }

      // Action locate (scope: 'atm')
      case 'locate'    { return 'locate'; }
      case 'look for'  { return 'locate'; }
      case 'search'    { return 'locate'; }
      case 'find'      { return 'locate'; }

      // Action execute (scope: 'transfer')
      case 'make'      { return 'execute'; }
      case 'perform'   { return 'execute'; }
      case 'fulfill'   { return 'execute'; }
      case 'complete'  { return 'execute'; }
   }

   return 'unknown';
}
```

Kernel folder

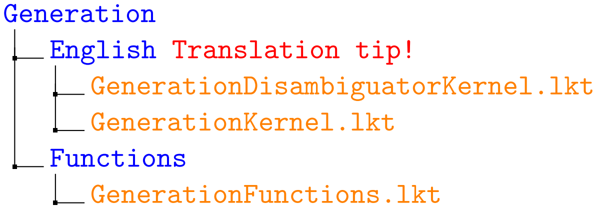
```
Kernel
 ├── CountryPlace Parameter folder
 ├── ...
 ├── Social Core dialogue act folder
 ├── ...
 ├── Generation Generation rules for kernel scripts
 ├── Generic Generic natural language features
 ├── Constants Some useful lekta constants
 ├── DialogueManager DM implementation
 ├── Macros Lekta macros for verbs and nouns inflections
 ├── MainEnglishGeneration.lkt
 ├── MainEnglishGrammar.lkt
 ├── MainEnglishLexicon.lkt
 ├── MainFunctions.lkt
 └── MainTypes.lkt
```

Core dialogue act folder example: Social



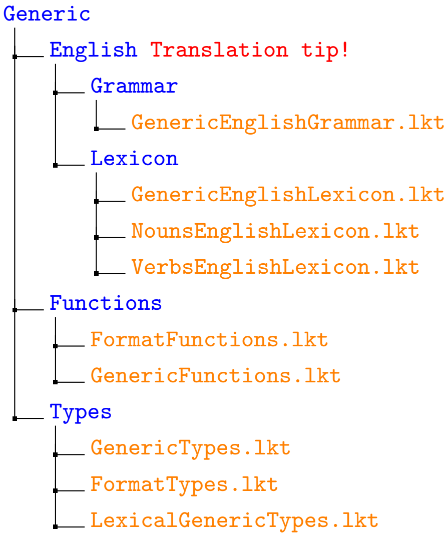
- Social
  - English Translation tip!
    - Generation
      - GenerationSocialGoodbye.lkt
      - GenerationSocialGreeting.lkt
    - Grammar
      - SocialEnglishGrammar.lkt
    - Lexicon
      - SocialEnglishLexicon.lkt
  - Types
    - LexicalSocialTypes.lkt
    - SocialTypes.lkt
  - MainSocialTypes.lkt

Generation folder



```
Generation
  English Translation tip!
    GenerationDisambiguatorKernel.lkt
    GenerationKernel.lkt
  Functions
    GenerationFunctions.lkt
```

Generic folder



- Generic
  - English Translation tip!
    - Grammar
      - GenericEnglishGrammar.lkt
    - Lexicon
      - GenericEnglishLexicon.lkt
      - NounsEnglishLexicon.lkt
      - VerbsEnglishLexicon.lkt
  - Functions
    - FormatFunctions.lkt
    - GenericFunctions.lkt
  - Types
    - GenericTypes.lkt
    - FormatTypes.lkt
    - LexicalGenericTypes.lkt