Fluency project tutorial

Jose F Quesada & Jose Luis Pro

Current state

What do we have now?

- With Iekta 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).*

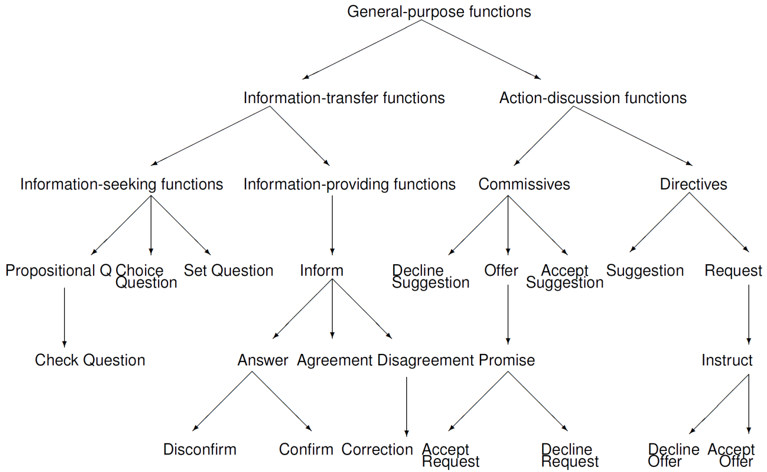
Dialogue act annotation definition

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).*

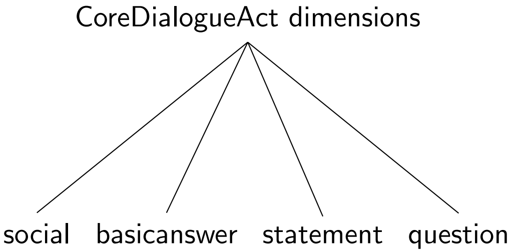
Bunt dialogue act taxonomy



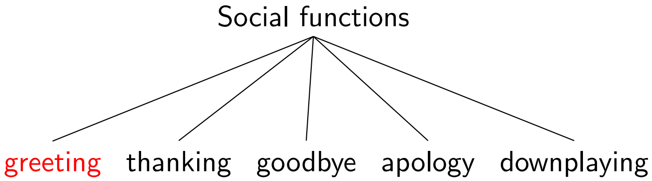
Harry Bunt et al. Towards an ISO standard for dialogue act annotation (2010).

Fluency dialogue act taxonomy

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



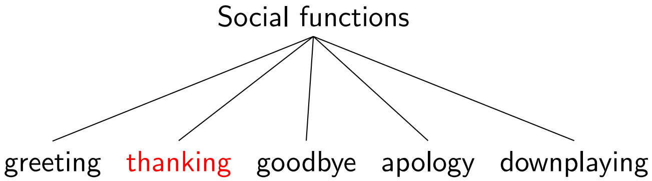
Social dimension



```
U: Good morning.

U: Nice to meet you.

U: Hello!
```

```
CoreDialogueAct:

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

Social dimension



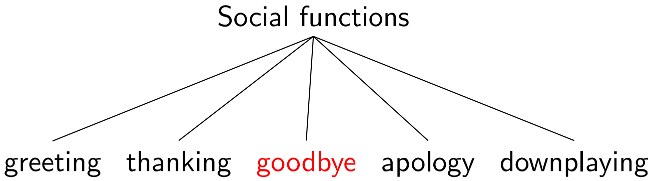
U: Thanks!

U: Thank you very much.

U: I'm so thankful!

```
CoreDialogueAct:

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

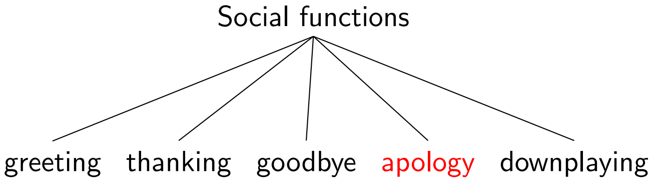
Social dimension



```
U: Have a good day!

U: See you later.

U: Bye!
```

```
CoreDialogueAct:

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

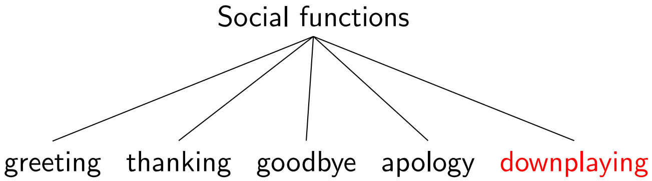
Social dimension



```
U: I'm sorry!

U: Excuse me.

U: My sincere apologies.
```

```
CoreDialogueAct:

[Dimension:   social ]
[ Function:  apology]
```

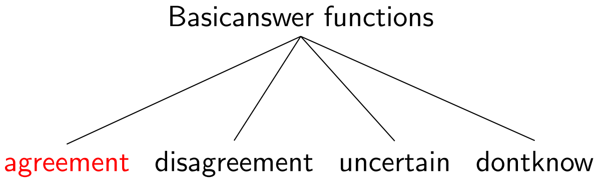
Social dimension



```
U: You're welcome!

U: Not at all.

U: Don't mention it.
```

```
CoreDialogueAct:

[Dimension:     social    ]
[ Function:   downplaying ]
```

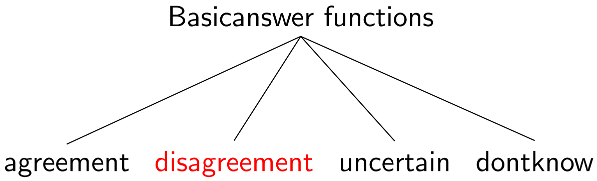
Basicanswer dimension



```
U: Yes.

U: Ok.

U: That's right.
```

```
CoreDialogueAct:

[Dimension:   basicanswer]
[Function:     agreement ]
```

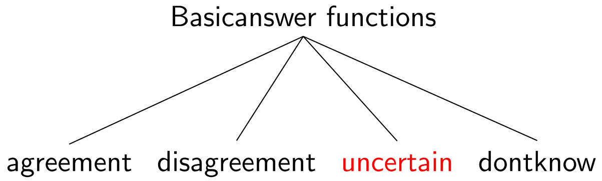
Basicanswer dimension



```
U: No.

U: No way.

U: I can't agree.
```

```
CoreDialogueAct:

[Dimension:   basicanswer ]
[Function:   disagreement ]
```

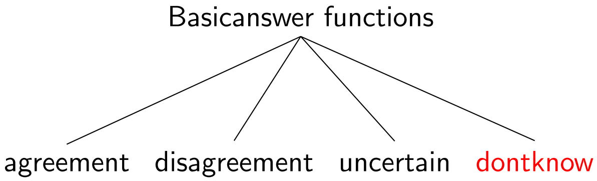
Basicanswer dimension



```
U: Maybe.

U: Perhaps.

U: Probably.
```

```
CoreDialogueAct:

[Dimension:   basicanswer]
[Function:     uncertain ]
```

Basicanswer dimension



U: I don’t know.

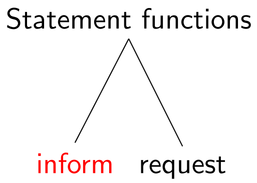
U: I don’t remember.

U: I have no idea.

CoreDialogueAct:

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

Statement dimension

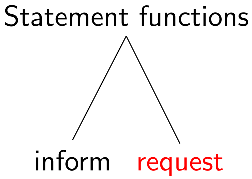


```
U: My name is ...

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

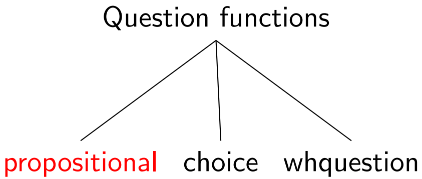


```
U: I want ...

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?

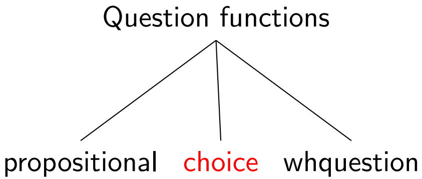
U: Do you like basketball?

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?

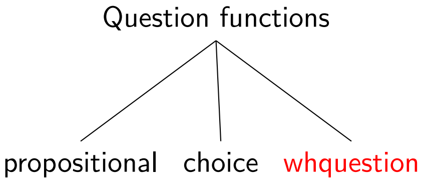
S: Do you prefer coffee or tea?

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



```
U: What time is it?

U: How much is the doctor
appointment?

U: Where is the nearest ATM?
```

```
CoreDialogueAct:

[Dimension:   question ]
[ Function:  whquestion]
```

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

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.`

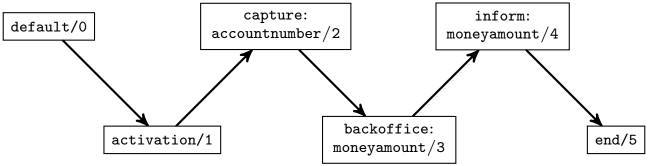
```
(DialogueAct:
   (CoreDialogueAct:(Dimension:'statement',
                    Function :'request')),
   (TaskDialogueAct:(Action:{(ActionDomain:'book')}
                    Scope:'appointment')),
   (Parameters:     {(Parameter:
                      (ParameterValue:
                       (DateTime:(OffsetDate:(DirectionOfTime:'forward',
                                              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.

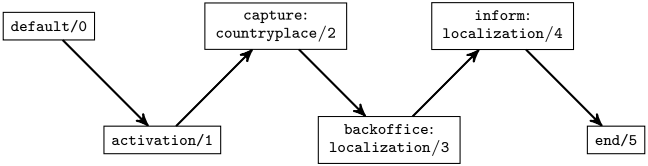
Script model: ConsultBalance script

```
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
```



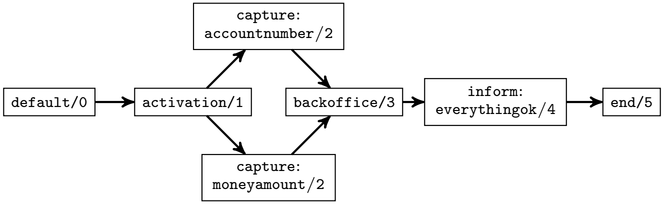
Script model: LocateAtm script

```
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
```



Script model: MakeTransfer script

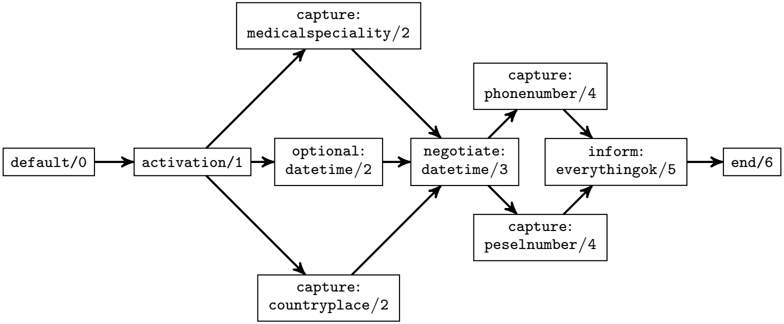
```
Descriptor: 'MakeTransfer'

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
```

Key design points

- **Understanding:** Fluency must treat with some natural language richness such as:
  - Ambiguity.
  - Polysemy.
  - Idiomatic constructions.
  - Negation.
  - Anaphora (use of an expression whose interpretation depends upon another precedent expression).
- **Context:** Establish a dialogue level with the user avoiding the fact that the enquiries could be atomic.
- **Integration:** With information systems or external API's.
- **Multilingualism.**
- **Multi-channel acces:** Written or spoken.
- **Business oriented:** The system itself can act as an information provider in order to improve business strategies.

1. Flexibility in expressions

We can express an idea in various forms (suppose here the day after tomorrow is Wednesday)

```
U: I would like an appointment for the dentist the
day after tomorrow.

U: I need to visit an orthodontics doctor on
Wednesday.
```

2. Naturalness and idiomatic constructions

```
U: Yes, hello?  Well, fine, I need an appointment ehh
for the dentist, please.

U: Good morning, miss, I want to make a transfer to
my son.

U: All right, see you!

U: Bye, have a nice day.
```

3. Fluidity

U: I want an appointment tomorrow early in the morning for the dentist.

U: I want to fly to Krakow next month.

4. Repetition of information

Request repetition:

U: I want an appointment, yes ehh I want it for the dentist.

Parameter repetition:

U: I want an appointment for the dentist, yes, that's right, for the dentist.

Scope repetition:

U: I need an appointment, an appointment tomorrow for the dentist.

Action repetition:

U: I want to cancel or revoke my bank account.

5. Spelling correction

```
U: I watn an appoitnment for tonorrow.
S: Ok, I see that you wish to book an appointment for
a doctor tomorrow.  Where are you calling from?
```

6. Order-free parameters

```
U: I need to book an appointment for tomorrow morning
for the dentist.  I'm calling from Sevilla.

U: Hi, I'm Jose calling from Sevilla, for the dentist
tomorrow morning I need an appointment booked.
```

7. Incomplete requests

```
U: For the dentist tomorrow, please.
S: Do you wish to make a medical appointment?
U: Yes, that's right!
S: You said tomorrow.  So you need a Orthodontics
doctor.  From which city do you call?
```

8. Incomplete optional parameters

For example, `datetime` in this example

```
U: An appointment for the dentist in Sevilla.
S: Ok, I see that you wish to book an appointment for
the dentist in Sevilla.  I will search for a suitable
appointment for you as soon as possible.  What about
today at 16 o´clock?
```

9. Incomplete compulsory parameters

For example, `countryplace` in this example

```
U: An appointment for the dentist.
S: So you want to make a medical appointment for the
dentist.  Could you please tell me where are you
calling from?
U: I call from Sevilla.
```

10. Multiple grounded request

System has no doubt about what user wants ⇒ *Weak Disambiguator*

A weak disambiguator asks user for the order in the requests but maintains all of them.

U: I want to make an appointment and execute a bank transfer.
S: What do you want to do first, make a bank transfer or book an appointment?

U: My account number is 1234567890 a total amount of 120 euros and for the dentist tomorrow.
S: What do you want to do first, book an appointment or make a bank transfer?

11. Single or multiple ambigous request

System is not sure about user needs ⇒ *Strong Disambiguator* or *Explicit Disambiguator*

A strong disambiguator asks user between several options, discarding the ones not chosen.

U: My account number is 1234567890.
S: With the information that you provided, I think you can choose between these alternatives, make a bank transfer or consult your balance. What do you prefer?

An explicit disambiguator asks user to confirm a system assumption.

U: Consult.
S: Please, could you be so kind to confirm that you want to consult the balance of your savings?

12. Infering user needs with incomplete information

U: My account number is 1234567890 a total amount of 120 euros for the dentist tomorrow in Sevilla.

It's possible that ALL scripts could be activated with above sentence, but if we read it from a human point of view, it's natural to discard some alternatives while keeping some of them:

S: What do you want to do first, book an appointment or make a bank transfer?

13. Parameter negotiation

U: I want an appointment for the dentist tomorrow in Sevilla.
S: Ok, I see that you wish to book an appointment for an Orthodontics doctor tomorrow. So you are in Sevilla. What about tomorrow at 13 o'clock?
U: Could it be earlier, please?
S: Ok, I see, earlier. And at 12 o'clock?
U: No, no, impossible.
S: And what about 9 o'clock?
U: Oh, please, a little bit later!
S: What about tomorrow at 11 o'clock?
U: Yes, that's perfect!
S: Ok, agreed! What is your phone number, please?

14. Inmediate correction.

```
U: I want to make a 120 euros bank transfer, please,
oh, no, no, I'm sorry, the money amount should be 150
euros.
S: So you want to transfer some money.  The money
amount is 150.00 EUR. Could you please tell me your
account number?
```

15. Implicit confirmation

U: An appointment, please.
S: So you want to make a medical appointment. Where are you calling from?
U: From Sevilla.

User implicitly confirms that he wants an appointment for the doctor.

S: In Sevilla, ok. What medical speciality fits you?
U: For pediatricts, please.

User implicitly confirms that he calls from Sevilla.

16. Explicit confirmation

`S: Ok, so you want to make a 120 euros transfer from your account number 1234567890. Are you sure you want to continue?`

17. Later correction

```
U: An appointment, please.
S: So you want to make a medical appointment.  Where
are you calling from?
U: From Sevilla.
S: In Sevilla, ok.  What medical speciality fits you?
U: No, no, I call from Granada.
```

```
User corrects system.
```

```
S: So you are in granada.  Sorry, I did not
understand you.  Please let me know for which medical
speciality you need the appointment.
```

18. Grounded correction

S: Where are you calling from?
U: From Sevilla.
S: I understood that you are calling from Sevilla. Ok, I am looking for a date and time that would fit you. Wait a moment, please. What about today at 20 o´clock?
U: Yes, perfect.
S: Ok, agreed! What is your phone number, please?
U: I am in Granada.
S: Ok, we were talking about an appointment in Sevilla. Is that correct?
U: No.
S: Ok, I understood that you call from Granada. I will look for a date that would fit you. What about today at 16 o´clock?

19. Semantic expectatives

S: What's your phone number?
U: 123123123
S: So your phone number is 123123123.

S: What's your account number?
U: 1231231231
S: I understood that the account number is 1231231231.

20. User parameter segmentation intrasententially

```
S: What's your phone number?
U: 686 57 58 85.
S: So your phone number is 686575885.
```

21. User parameter segmentation intersententially

```
S: What's your phone number?
U: 686.
S: 686, ok, go on.
U: 57.
S: 57, yes.
U: 59.
S: 59, all right.
U: no, no!
S: Ok, we have 686 57, please continue.
U: 58.
S: 58, right.
U: 85.
S: OK, let me confirm your phone number, is it
686575885?
```

22. Subdialogues

U: I want to make a transfer.
S: Ok, I see that you wish to make a transfer. And how much money do you want to transfer?
U: But I need first to consult my balance.
S: So you want to consult the balance of your savings. What is your account number, please?
U: 1231231231.
S: I understood that the account number is 1231231231. The total money is 2078.67 EUR. Now, let me continue with the bank transfer. I understood that the account number is 1231231231. Could you please tell me the money amount?
U: 120 euros.

23. Parameter validity detection

```
U: My account number is 1234 1234 1234.
S: I understood that the account number is
123412341234.  The account number is not valid.
Remember that a correct account number must have 10
digits exactly.  Could you please tell me your
account number?

U: I want an appointment for the dentist on 31st of
november
S: Ok, I see that you wish to book an appointment for
the dentist.  You said the date 31st November.  But
that date is incorrect.  From which city do you call?
```

24. Early inconsistency detection

```
U: I want an appointment for a doctor of infectious
diseases in Sevilla for tomorrow.
S: I'm sorry, but we don't have any doctor of
infectious diseases near you at this moment.  Would
you like that I look for a dentist far away?
```

25. Overinformation

U: I want an appointment for 25th November for the dentist. I call from Sevilla, and my phone number is 123123123.
S: Ok, I see that you wish to book an appointment for the doctor. I understood that you are calling from sevilla. So you need a Orthodontics doctor. You said the date Friday, 25th November. I will search for a suitable appointment for you. Wait a moment, please. What about Friday, 25th November at 19 o´clock?
U: Yes, perfect!
S: Ok, agreed! So your phone number is 123123123. What is your identification number, please?

27: Mid-term memory

```
U: I want to locate my nearest ATM.
S: Ok, let me locate an atm for you.  Where are you
calling from?
U: Sevilla.
S: I understood that you are calling from sevilla.
Well done!  I will present in your screen the
location of your nearest ATM in a minute.  Could you
please tell me how can I help you?
U: Yes, I want an appointment for the dentist
tomorrow.
S: So you want to make a Orthodontics appointment for
tomorrow.  I remember that you are in Sevilla.  What
about tomorrow at 10 o´clock?
```

28: Non-collaborative user

```
S: Good morning.  Your Personal Assistant Service speaking.
Could you please tell me how can I help you?
U: I want an appointment in Sevilla.
S: Ok, I see that you wish to book an appointment for the doctor.
I understood that you are calling from Sevilla.  What kind of
medical speciality do you want?
U: blah
S: I did not understand that.  Please let me know for which
medical speciality you need the appointment.
U: blah
S: I need to know what type of doctor or speciality you require
in order to continue.  Please repeat again.
U: blah
S: Ok, I think that I am having too many problems trying to
understand you, so let me start from the beginning...  Please let
me know what do you need to do.
```

29: Cancelling task with user farewell

```
S: Good morning.  Your Personal Assistant Service speaking.
Could you please tell me how can I help you?
U: Bye!
S: It has been a pleasure serving you using this Personal
Assistant Service.  Hoping to serve you again.  Have a good
morning.

S: Good morning.  Your Personal Assistant Service speaking.
Could you please tell me how can I help you?
U: I want to book an appopintment.
S: Ok, I see that you wish to book an appointment for the doctor.
I would like to know your preferred town, please.
U: Sevilla.
S: So you are in sevilla.  What kind of medical speciality do you
want?
U: See you later.
S: Ok, confirm that you want to cancel this task, please.
```

30: Cancelling task with appropiate commands

```
S: Hello.  Your Personal Assistant Service speaking.
Could you please tell me how can I help you?
U: I want an appointment in Sevilla.
S: So you want to make a medical appointment.  I
understood that you are calling from Sevilla.  What
kind of medical speciality do you want?
U: Cancel this, please.
S: Are you sure that you want to cancel this task?
```

31: Formatting Mask for alphanumeric parameters

```
S: What is your account number, please?
U: ES20.
S: ES20.  Right, continue.
U: 7700.
S: 7700.  Please, continue.
U: 0024.
S: 0024.  Yes, what else?
U: 0031.
S: 0031.  All right, anything else?
U: 0257.
S: 0257.  Yes, what else?
U: 5766.
S: So your account number is ES20 7700 0024 0031 0257
5766.  Is it correct?
```

32: Multiple valid masks for parameters

```
S: So you want to consult the balance of your
savings.  What is your account number, please?
U: 2077.
S: 2077.  Ok, go on.
U: 0024.
S: 0024.  All right, anything else?
U: 00.
S: 00.  Right, continue.
U: 3102575766.
S: Ok, now I will repeat you account number:  2077
0024 00 3102575766.  Correct?
```

33: Digit check integrity validation

```
S: What is your PESEL number, please?
U: 020708.
S: 020708.  Please, continue.
U: 03628.
S: I catched 020708 03628.  Correct?

S: What is your PESEL number, please?
U: 021708.
S: 021708.  Ok, go on.
U: 03628.
S: The pesel number is not valid.  Please try again!
```

34: Backoffice malfunction messages

```
S: Ok, I see that you wish to make a transfer.  The
money amount is 150.00 EUR. What is your account
number, please?
U: 12345678901234567890.
S: The money amount is 150.00 EUR. Ok, we are having
some troubles trying to connect to our servers.
Please try again in a few minutes.  How may I help
you?
```

35: Ordinal detection for multiple choice question

```
U: I want to consult my balance, make a medical
appointment and locate my nearest ATM.
S: What do you want to do first, consult your
balance, locate an ATM or book an appointment?
U: The first one, please!

U: Second one.

U: The last option.
```