Methodology and notation for the specification of Goals-Guided Interfaces

As stated in the main manuscript, the recommended modeling or specification process of a GGI interface is based on and expands NGOMSL. That is, GGI is based on goals, actions, methods, and selections. Goals are intentions the user has. Actions, operators in NGOMSL, are simple cognitive, perceptive or motor activity. They can be an elementary activity like hitting a key, or a cognitive high-level activity the users can carry out by themselves. Methods are the sequences of steps the user needs to follow to accomplish a goal. And selections play here a similar role than selection rules in NGOMSL.

However, GGI expands the NGOMSL constructors, adding four new descriptors that provide more dynamicity to the models: conditional steps/options, description of effects on the system, cancelability, and selections for the system.

Now, the specification of a method has the following format:

```
Method for: goal [cancellable
    [disable if condition_for_the_system]
    [effect effect_on_the_system]]

1) step_1
2) step_2
...
 n) Return with goal accomplished
    [effect effect_on_the_system]
```

The optional cancellable clause will allow the user, later, during the interaction process, to cancel that method resuming the execution to the previous method. It is actually another available step for the user.

There are 4 types of steps:

```
step_i ::= accomplish | do | decide | goto
    [disable if condition_for_system]
    [effect effect_on_system]
```

These steps, resembling algorithmic procedures, are:

```
accomplish ::= Accomplish: goal
    Asks the user for beginning a new subgoal.

do ::= Do action
    Asks the user for a specific elementary or high-level action.

decide ::= Decide: if condition_for_user then goto step_
    Expresses an elementary mental action in which the user has to decide to jump
to a specific step_ or to continue to the next one. Therefore, it can express a
conditional jump within a method, allowing repetition of steps or fixing previous
errors.

goto ::= Goto step_
    Let us jump to another part of the method.
```
Every *method* can end with a special step, the *Return with goal accomplished*, to make explicit the *goal* has been achieved and the interaction must return to the previous *goal*. For example:

Method for: Furnish the kitchen
  Cancelable
  step 1. Accomplish: Choose furnishing task...
  step 2. Decide: If another furnishing task? then Goto 1
  step 3. Accomplish: Choose common materials.
      disable if common materials are previously chosen
  step 4. Decide: If modify budget? then Goto 1
  step 5. Do: Confirm budget (not allowing more changes)
  step 6. Return with goal accomplished

The *steps* can optionally be followed by a *disable-if* clause and/or an *effect* clause. The *disable if* must be followed by a condition for the system, turning it a *conditional step*. The system, while running, checks internal states enabling or showing to the user, or disabling or hiding from the user, those steps. Each step can be appended with an *effect* clause, followed by the description of that effect on the system (*step with effect*).

Finally, *Selections* have now the format:

```plaintext
[For the system] Selection for: goal
  [Cancelable [disable if condition_for_the_system
    [effect effect_on_the_system]]
    a) option_a
    b) option_b
    ...
  [Return with goal accomplished]
```

where each *option_i* has the next syntax:

```plaintext
If condition [then Accomplish: goal]
  [disable if condition_for_the_system]
  [effect effect_on_the_system]
```

A *selection* can be built using as many mutually excluding *options* as necessary. Each *option* is associated with a condition the user has to check. After they have decided, they must start the associated sub-*goal*, which, again, will be performed by another *method* or another *selection*.

The *cancellable* clause has the same purpose that it has in a *method*: it offers the opportunity to cancel the *goal* and returns to the previous *method*. It can also be followed by a *disable-if* clause and/or an *effect* clause. Example:

```plaintext
Selection for: Choose furnishing task
  Cancelable, disable if #items = 0
  a) If add new item? then Accomplish: Add item...
  b) If move item? then Accomplish: Move item...
      disable if #items = 0
  c) If rotate item? then Accomplish: Rotate item...
      disable if #items = 0
```
Although not very usual, we can define options not implying the starting of new sub-goals, but making a determinate effect on the system. In such cases, the options will not include the accomplish clause, but the effect clause. For example:

Selection for: Choose type of item
Cancelable
a) If the item is an electrical appliance then
   effect item_type = electrical_appliance
b) If the item is a wall furniture then
   effect item_type = wall_furniture
...
Return with goal accomplished

In general, selections are meant for the user to make. Nevertheless, some times, during the interaction process, we want the user to see new added goals, depending on internal states that the system has to check. They are selections for the system. The clause is For the system. Example:

For the system > Selection for: Enter kitchen sizes
a) If kitchen_shape = 2_sides then accomplish: Enter sizes for 2 sides...
   b) If kitchen_shape = 4_sides then accomplish: Enter sizes for 4 sides...
   ...
Return with goal accomplished

In any case, although we recommend and consider this methodology and notation as the most appropriate for the specification of GGI interfaces, other variants could be feasible, as long as they allow expressing the same characteristics, elements and philosophy of this way of interaction.