Joint handling of Rational and Behavioral reactions in Assistant Conversational Agents

Jean-Paul Sansonnet and **François Bouchet** [1]

Abstract. We describe here a framework dedicated to studies and experimentations upon the nature of the relationships between the rational reasoning process of an artificial agent and its psychological counterpart, namely its behavioral reasoning process. This study is focused on the domain of Assistant Conversational Agents which are software tools providing various kinds of assistance to people of the general public interacting with computer based applications or services. In this context, we show on some examples how the agents must exhibit both rational reasoning about the system functioning and a human-like believable dialogical interaction with the users.

1 INTRODUCTION

Assistant Conversational Agents (ACA) are a combination of assistant agents (rational agents used to assist novice computer users) and conversational agents (virtual characters, generally with a personality, interacting multimodaly with users, particularly through natural language). We use the word 'assistant' as a generic term to embrace a broad range of *conversational situations*, where the agent's role varies from an underling presenter only displaying informational content to a bossy coach or teacher actively monitoring user's actions, through more help or companion oriented ones for entertainment or social activities. In those situations, 3 actors (User U, Agent A, System S) are in bilateral interaction through 3 interfaces (Graphical GUI, Conversational CUI, Control CCI), as shown on Figure 1.

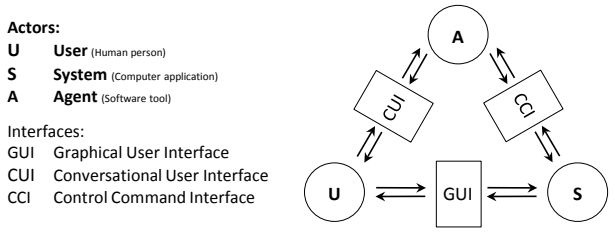

Figure 1. A typical Assistant Conversational Agent architecture

Depending on the role played by the agent, its interaction with the user is expected to be different on the behavioral side, which can influence the rational side. To deal with the acceptability issues, psychological factors must be taken into account to produce more believable human-like interactions, which requires beforehand to be able to study the relationships between rational and behavioral reasoning. Hence, we propose in section 2.1 a general architecture of a framework (called R&B) dedicated to the support of rational and behavioral reasoning in the ACA context, describe some of the languages used in section 2.2, before concluding with two short case studies in section 3.

[1] LIMSI-CNRS, BP 133 91403 Orsay cedex, France. {jps,bouchet}@limsi.fr

2 THE R&B FRAMEWORK

2.1 General architecture

The R&B framework (*cf.* Figure 2), relies on two main principles:
Principle of separation: rational and behavioral heuristics are designed separately and also executed concurrently on two separate engines (R and B), and can be experimented together or not.
Principle of genericity: a R&B case study is associated to a couple <Policy, Situation> where the policy is a particular instance of the generic architecture and the situation characterizes the roles of the actors. Each case study instantiates a particular strategy of heuristic management in the engines (R and B), and they interoperate through a shared communication space (W).
The languages *FRL* and *MQL* are used to represent respectively a formal representation of the natural language requests (input or output) used by the dialogue engine (D) and the access functions to elements of the model (M) described in *MDL*. Those languages (formally described in [1]) act as a layer upon which each case study can implement its particular strategies, while W deals with Query objects Q_i (in *QDL*) wrapping *FRL* and *MQL* requests, ensuring the communication between the engines: an engine can access the queries put by other engines in W, both for consultation or alteration.

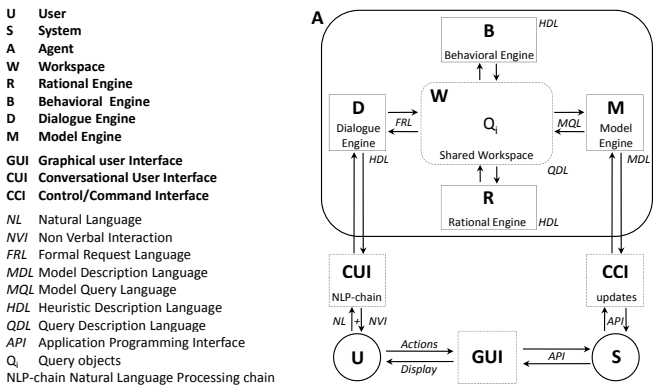

Figure 2. General architecture of a R&B agent

In the case of a strict help agent, the user U generally interacts with the system S through the GUI, ignoring A. But when "things go wrong", the user has the possibility to put a question in Natural Language, along the following workflow:

$$\mathtt{U} \overset{NL}{\Rightarrow} \mathtt{CUI} \overset{NL+}{\rightarrow} \mathtt{D} \underset{[FRL]}{\overset{QDL}{\longrightarrow}} \left(\mathtt{B|R} \underset{[MQL]}{\overset{QDL}{\longrightarrow}} \mathtt{M} \underset{[MQL]}{\overset{QDL}{\longrightarrow}} \mathtt{B|R} \right)^{*} \underset{[FRL]}{\overset{QDL}{\longrightarrow}} \mathtt{D} \overset{NL}{\rightarrow} \mathtt{CUI} \overset{NL}{\Rightarrow} \mathtt{U}$$

with optional system updates: $\left(\mathtt{M} \overset{\text{API}}{\longrightarrow} \mathtt{CCI} \Rightarrow \mathtt{S} \right)^{*}$ where $\underset{[X]}{\overset{QDL}{\longrightarrow}}$ means that a request written in X is wrapped into a *QDL* query.

2.2 Description languages

Query Description Language (*QDL*)
A query is an element of W that *wraps* a request written in *FRL* or *MQL* and provides extra attributes. It has the following structure:

$$Q_i = [\text{value}[\{r|\{r_i\}], \text{history}[\{\texttt{D}, \texttt{R}, \ldots\}], \text{to}[\texttt{M}], \text{status}[+]]$$
$$= Q_i.\text{value}^{Q_i.\text{status}}_{Q_i.\text{history}|Q_i.\text{to}} = \{r_1, \ldots, r_n\}^{+}_{\{\texttt{D},\texttt{R},\ldots\}|\texttt{M}} \text{ (shortened)}$$

Where:
– i $\in \mathbb{N}^+$ is the absolute identifier of the query in the session
– value contains a sequence of *FRL* | *MQL* requests or a single one
– history $\in \{\texttt{D}, \texttt{R}, \texttt{B}, \texttt{M}\}^*$ is the stack of engines that handled Q_i
– to $\in \{\texttt{D}, \texttt{R}, \texttt{B}, \texttt{M}\}$ is the next engine meant to retrieve Q_i
– status $\in [\emptyset, -, +]$ shows the success of the latest handling of Q_i
Note that although the query can be given a destination (field 'to'), it doesn't prevent other engines to see it in W and possibly to alter it.

Heuristic Description Language (*HDL*)
A heuristic defines a rational or a behavioral reaction to a class of formal requests in *QDL*, and its general form is:
H : id[QDL pattern]:–{GuardedScript$_1$, …, GuardedScript$_n$}
Where:

GuardedScript	$\equiv$	{Guard$_1$ → Script$_1$, …, Guard$_n$ → Script$_n$}
Guard$_i$	$\equiv$	Logical expr | $\emptyset$ ($\emptyset$ = True)
Script$_i$	$\equiv$	Instruction | {Instruction$_1$, …, Instruction$_n$}
Instruction$_i$	$\equiv$	Basic operation | Query call | GuardedScript
Query call	$\equiv$	Q[Query id, {*FRL* req | *MQL* req}]

Note that instructions can recursively be guarded scripts.

A set of heuristics can be defined and associated with any of the four engines (D, M, R and B). Their execution is performed by the Heuristic Scheduler (HS) which ensures their coroutining and:
– within a heuristic H, it decides when to execute (guard→script),
– within a R&B case study defined as <HS-Policy, Situation>, it decides when engines and heuristics should take a turn.

As guards in heuristics can overlap, several *execution policies* can be selected (*e.g.* first-hit-exit, execute-all, random-choice), and as a guard can remain active ($true$) after the execution of its script, several *repetition policies* can also be selected (*e.g.* execute-once, loop). Moreover, since several heuristics (in the same engine or in different ones) can match queries in W, again several *heuristic policies* (*e.g.* behavior-first, rational-first, alternate M and B) and *query policies* (*e.g.* FIFO based, random choice) are possible.
The principle of genericity compels the scheduler to be parameterizable, but we'll only consider here only the following policies:
– *Within heuristics:* all instructions with active guards ($true$) are executed; when several guards are simulteaneously active, a random choice is performed; instructions are only executed once; a heuristic is terminated when all its instructions are executed (which may never happen – hence, W is cleared after each request handling).
– *Between heuristics:* all heuristics that match a query object in W are launched (*i.e.* coroutined with the already launched ones). When a heuristic is terminated, it can be launched again (but no reentrance is available). When several heuristics (even associated with different engines) are eligible, a random choice is performed, thus resulting in various R&B interleaved executions.

3 EXAMPLES OF HEURISTICS

Let's assume the user puts the question "What is your age?", which has for consequence the addition into the workspace W of the query:
Q_1 = {ASK$_u$[agent.age]}$^{\emptyset}_{\{\texttt{D}\}|R}$

A possible *rational* heuristic handling questions about attributes is:

```
1 :  H_R1 : ask-agent-attribute[{ASK_u[agent.x_]}^-_|_] :–{
2 :            → Q[i, GET[x_]],
3 :     Q_i^+  → Q[j, TELL_a[agent.x_, Q_i.value]],
4 :     Q_i^-  → Q[j, {UNKNOWN_a[agent.x_], TELL_a[Q_i.value]}]
5 :     Q_i^¬∅ → Q+_this
6 :  }
```

1 : x_ is a pattern variable matching any symbol like age, gender…
2 : The empty guard prompts the script to be executed immediately. In Q_1, x_ being 'age', a query Q_i is created to get this value from M.
3 : If the request in Q_i has been successful (Q_i.status == +), *FRL* request TELL$_a$[agent.x_, Q_i.value] is wrapped into a new query Q_j, and Q_i.value contains a *MQL* request OK[retrieved-value].
4 : If the request in Q_i has been unsuccessful (Q_i.status == −), a *FRL* answer in two parts is wrapped into a new query Q_j and Q_i.value contains FAIL[report].
5 : Once Q_i has been handled, the current query Q_{this} is declared to have been successfully handled as well.
In any case, the request in Q_j is then retrieved by D to be sent to U.

Now let's assume the user expresses his/her dissatisfaction regarding the agent previous reaction(s), generating the addition into W of:
Q_2 = {DISLIKE$_u$[agent}$^{\emptyset}_{\{\texttt{D}\}|R}$
Dealing with an emotional reaction can't be rational, and one of the possible *behavioral* reactions could be given by a heuristic like:

```
1 :  H_B2 : dislike-agent[{DISLIKE_u[agent]}^-_|_] :–{
2 :   → {   Q[i, MAP[energy, λ x.x * 0.9]],
3 :         Q[j, MAP[confidence, λ x.x * 0.9]],
4 :         Q[k, MAP[cooperation, λ x.x * 0.9]]   }
5 :    Q_i^+ ∧ Q_i.value < −0.5 → Q[l, TELL_a[energy, "tired"]]
6 :    Q_j^+ ∧ Q_j.value < −0.5
↪ :         → Q[l, TELL_a[confidence, "depressed"]]
7 :  }
```

2-4 : Executes a sequence of queries to change the agent's mind state
6 : If the agent's energy is very low, a query with a *FRL* request to say "I feel tired" is generated.
7 : A second *FRL* request can be added to that query (or created).

Other attempts to add psychological aspects to agent architectures have been undertaken (like [3]), and [2] has even shown that the order in which heuristics are applied can impact the agent's *perceived* personality. The proposed R&B framework, relying on the state of the art for rationality (decision trees here, but conventional plans in the current Mathematica implementation[2]) and psychology (*cf.* the mind model in [1]) provides a useful testbed for flexible (thanks to the shared workspace and parameterizable scheduling policies) and controllable experiments (thanks to the principle of separation). Further work will have to enhance this approach by defining software tools for heuristics manipulation and developing larger case studies.

REFERENCES

[1] François Bouchet and Jean-Paul Sansonnet, 'A framework for modeling the relationships between the rational and behavioral reactions of assisting conversational agents', in *Proc. of the 7th European Workshop on Multi-Agent Systems (EUMAS'09)*, Agia Napa, Cyprus, (2009).
[2] Mehdi Dastani and Leendert van der Torre, 'A classification of cognitive agents', in *Proceedings of Cogsci02*, pp. 256—261, (2002).
[3] Emma Norling and Franck E. Ritter, 'Towards supporting psychologically plausible variability in Agent-Based human modelling', in *Proc. of AAMAS'04*, (2004).

[2] http://www.limsi.fr/~jps/research/rnb/rnb.htm