HOW TO ENHANCE GENERICITY IN NATURAL LANGUAGE COMMAND INTERPRETATION USING INTROSPECTION AND ONTOLOGIES?

Laurent MAZUEL, Nicolas SABOURET

LIP6 – Laboratoire Informatique de Paris 6

Corresponding author:

Laurent Mazuel, 104 av du Président Kennedy, 75016 Paris, France
Laurent.Mazuel@lip6.fr

This paper presents a general architecture towards a more generic approach to command interpretation in conversational agents. Our architecture contains generic (in sense of application-independent) natural language (NL) modules that are based on ontologies and agent introspection. We will especially focus on the presentation of the event generator (introspection part) and dialogue manager (application independent part) modules, which rely on a bottom-up approach for matching the user's command with the set of currently possible actions.

Key words: Conversational agent, natural language command, human-computer dialogue strategies, introspection and ontologies.

1. INTRODUCTION

Recent works on Embodied Conversational Agents (ECA) [7] and more generally conversational systems [2] showed that natural language (NL) interaction is one crucial step in the course towards a more natural human-agent interaction. However, the chosen approaches in ECA mostly rely on ad-hoc pattern matching without semantic analysis [1]. The dialogue system community, on the other hand, proposes to use ontologies to improve genericity [9,14]. The main idea behind the use of ontologies is to specify generic algorithms that only depend on the ontology formalism. Thus, applications only depend on the ontology and the specific application problem-solver. Systems like [9], [15] use the ontology to parameterize a generic parser. However, in such systems, the ontology formalism itself is ad-hoc. It strongly depends on the application type and does not allow generic knowledge representation. Moreover, these ontologies describe the application model as well as the application actions. Our claim is that it should be possible to extract the meaning of actions from the code itself. The ontology then is no longer an application descriptor. It only provides the complementary semantic information on relations between the application concepts (which is the initial role of ontologies). Moreover, systems that use generic knowledge representation (e.g. [14]) rely on application-dependent parsers. However, the parser uses the structure of the ontology to understand over-specified or under-specified commands like "switch the light on" (the system will propose the different possible locations to enlighten).

This paper focuses on command interpretation for intelligent agents. We propose a generic NL system based upon a domain ontology and agents capable of introspection. The system extracts the set of possible actions from the agent's code and matches these actions with the user's command using the ontology as a glue. In addition, a score-based dialogue manager (like [11]) deals with misunderstood or indefinite commands.

Our paper is organised as follows. In the second section, we give a general overview of our agent model. The third, fourth and fifth sections present the natural language processing algorithm we use. We first present the parser (section 3), then a general overview and related work (section 4) and finally detail our algorithm for command interpretation using introspection (section 5). We also present our dialogue manager

that deals with clarification in section 6. Section 7 presents the evaluation we made with several users of the system. Section 8 concludes the paper.

2. OVERVIEW OF OUR MODEL

Our aim is to be able to program cognitive agents that can be controlled by natural language commands, and that are capable of reasoning about their own actions, so as to answer questions about their behaviour and their activity. To this purpose, we rely upon a specific language that allows to access to the description of the agent's internal state and actions *at runtime*: the View Design Language (VDL) language.

The VDL model is based on XML tree rewriting: the agent's description is an XML tree whose nodes represent either data or actions. The agent rewrites the tree at every execution step according to these specific elements. This model allows agents to access at runtime the description of their actions and to reason about it for planning, formal question answering [16], behaviour recognition [17], etc. The VDL agent model can be used for web services composition [8], Embodied Conversational Agents [18], social behaviour simulation, etc.

In the VDL model, every agent is provided with a domain ontology written in OWL. This ontology must contain at least all the concepts used by the agent (i.e. the VDL concepts), either as XML tags (except for VDL keywords), attributes names and values or CDATA contents. We note C_{VDL} the set of the VDL concepts and C_{OWL} the set of the OWL concepts, individuals and properties. We define an injective mapping map_{VDL} defined on the set C_{VDL} and taking values in the set C_{OWL}, to match VDL concepts on the ontology.

We can exhibit two kinds of behaviour for an autonomous agent provided with interaction capabilities [10]:

* The reactive behaviour is used when the agent performs operations in response to a command (a typical example is a start/stop operation).

* The proactive behaviour is the capability for the agent to run independently of any command.

In this paper, as we focus on the human-machine interaction, we will only work on reactions. In VDL, reactions are triggered by external events, i.e. XML nodes sent to the agent at runtime for command. They are the formal representation of commands. External events correspond to the content of “request” ACL messages, whereas reactions describe how such messages (sent by other agents or by the user) must be processed. The aim of the NLP system presented in this paper is to build VDL events from a user's command.

Message processing in MAS protocols can be decomposed in two stages. In the first stage, a parser checks the message's syntax (eventually, the message could be rejected). It ensures that the reaction will be able to process the event and it switches the event to the correct reaction. In the second stage, the reaction processes the event itself according to the agent's internal state and reaction's definition (i.e. behaviour). It must extract relevant information (i.e. parameters expected by the reaction) from the event and then perform modifications. However, these modifications will be performed only if the current agent's context (internal state) allows it.

In VDL, as in most action representation models, actions are defined as a triple <N,P,E> where N is the action name, P is the set of preconditions of the action and E its effects. The parser and context verification must be implemented within the agent using preconditions. Based on the previous definitions, we characterise four kinds of preconditions for a reaction *r* in *R*, the set of agent reactions:

* $P_e(r)$ is the set of event preconditions. They are used to ensure that a given action is triggered by a given class of events. Their interpretation relies on subsumption for checking the structure of the received event.

* $P_s(r)$ is the set of structure preconditions set. It is used to check the message's syntax and to ensure that the action will be able to process the event. Preconditions in $P_s(r)$ do not depend on the agent's internal state, but only on the received event.

* $P_c(r)$ is the set of context preconditions. Such preconditions only depend on the agent's internal state. For example, a (simulated) robot cannot move when it runs out of energy.

* $P_{cs}(r)$ is the set of contextual structure preconditions, i.e. preconditions that depend both on events (selected by $P_e(r)$) and on the agent's internal state. For example, a robot cannot catch an object when this object is out of reach.

We note $P_e = \bigcup_{r \in R} \{P_e(r)\}$. For all $e \in P_e$, we note $R_e(e) = \{r \in R | e \in P_e(r)\}$ the set of reactions that process the event *e*.

3. NLP TOOLCHAIN AND GLOBAL ARCHITECTURE

This section gives a general overview of the NLP tool chain and the global system's architecture. In our project (see figure 1), the lexical module is based on the default Open NLP tokenizer, tagger and chunker. Additionally we make use of a home-made lemmatizer. As anticipated in [13], the use of a grammar based syntactic parser is not relevant for NL commands. In fact, users often command the system with keywords rather than well-structured sentences (e.g. “drop object low” or “take blue”). Thus, we represent the content of sentences as bags of words, after having removed stop-words identified by their tags.

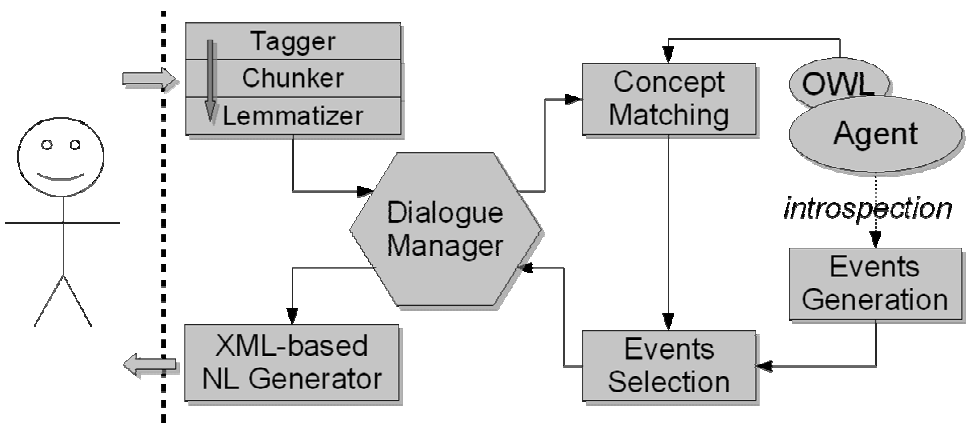

Figure 1. Global Architecture.

The semantic analysis is the core of our model. Our aim is to use ontologies for concepts matching. In the current stage, we simply use synonymy for linking the user's command with VDL concepts. For $(x, y) \in C^2_{OWL}$, we define the distance between *x* and *y* with respect to synonymy as:

$$dist_{syn}(x,y) = \begin{cases} 0 & \text{if} \quad (x = y) \ \vee \ (\ owl_{sameas}(x,y) \ ) \\ 1 & \text{else} \end{cases} \tag{1}$$

with $owl_{sameas}(x, y)$ being the transitive and reflexive relation for concept synonymy in OWL.

Our semantic analysis method relies on the hypothesis of semantic connectivity [19]: every concept that appears within a relevant command must be defined in the ontology. This hypothesis is enriched with our $dist_{syn}(x, y)$ operator:

Every concept that appears within a relevant command is either directly associated with a VDL concept or is in a OWL sameAs relation with an agent's concept.

More precisely, if we note *S* the bag of words that represent the user sentence, we can build the set *C* of known concepts as follows:

$$C = \left\{ c \in C_{VDL} \middle| s \in S.dist_{syn}(map_{VDL}(c), s) = 0 \right\} \quad (2)$$

C contains the set of all VDL concepts that appear within the user's command. Similarly, we build the set *U* of not-understood concepts:

$$U = \left\{ s \in S \middle| \forall c \in C_{VDL}.dist_{syn}(map_{VDL}(c), s) = 1 \right\} \quad (3)$$

U contains the set of command words that do not appear within the agent's description. Note that the construction of *C* and *U* is only a preliminary step for the algorithm described in section 5.

The last part of our chain is an English NL generator that transforms any VDL node into an English sentence, by appending the translation of concepts obtained by a depth-first search of the node. Generally, this recursive algorithm prints the node's tag, its attributes as "attribute is value", its content (if any) and then all its sub-elements. For instance:

<take position="out"> <shape>square</shape> </take>

will become "take position is out shape is square". Moreover, we use specific rules for VDL keywords.

The resulting outputs are awful from a syntactical point of view, but it is sufficient for users to understand the system's proposal or explanations. However, it is possible to improve it significantly using an XML-based NL generator such as [4] that does not require any template.

The next two sections show how the set *C* of known concepts is used to build VDL events using an introspection NL algorithm.

4. GENERAL OVERVIEW OF THE INTROSPECTION ALGORITHM

Our approach is based on Allen's bottom-up approach [3]. The classical bottom-up approach makes use of an early defined list of competences and tries to match the natural language command onto one of the possible formal commands (e.g. [15], [9]). However, the competences list has to forecast all possible dialogues (even problem cases) and their translation into formal commands (possibly with parameters). To avoid this issue, we propose to adopt a constructive bottom-up approach based on preconditions analysis. Our approach uses contextual information (obtained from the agent's code at runtime) to determine which events can be processed by the agent in the current state. This issue has been widely studied for software validation (e.g. [5]) and showed interesting results for testbeds generation.

Our system builds the list of possible events from the agent's point of view, without concern about whether any of those matches the user's command. Similarly, using constraints relaxation on context preconditions and contextual structure preconditions ($P_c(r)$ and $P_{cs}(r)$), it builds the lists of "currently impossible" events, i.e. events that are not acceptable by the agent in its current state but that would be accepted in a different state.

The next section presents the algorithm that computes the possible and the impossible events set and selects relevant events. Section 6 presents the Dialogue Manager (DM) that deals with the sets of possible and impossible events to generate better feedback to user.

5. EVENTS GENERATION AND SELECTION

The event generation algorithm is responsible for building a set of potential event. It is the core of our NL command processing system. It allows the system to use the agent's actions description (extracted from the agent's code itself) so as to build the set of events that can be carried out by the agent. This avoids the use of priorly defined static competences lists.

Our algorithm builds both the set of possible events *E* and the set of "currently impossible" events *F*. We use event preconditions (P_e) to provide the initial skeleton of the event. Since P_e filters external events

using subsumption, all events build by addition of sub-elements to a skeleton $e \in P_e$ will be accepted and reciprocally, all accepted events must be build from a skeleton. We note Ψ the (infinite) set of all possible VDL nodes. To compute *E*, we remove from P_e events that cannot be processed currently:

$$P_{e+c} = \left\{ e \in P_e \middle| \forall p \in \bigcup_{r \in R_r(e)} P_c(r),\ eval(p,e) = \mathrm{T} \right\} \tag{4}$$

with $eval : \Psi^2 \rightarrow \{\mathrm{True}, \mathrm{False}\}$ (we will note only T and F in place of True and False in future equation), the precondition evaluation function: $\forall p \in P_x(r), eval(p,e) = \mathrm{T}$ if and only if the precondition *p* is valid with respect to the event *e* and the current agent's state. P_{e+c} is the set of event skeletons that are accepted by the agent with respect to constraint preconditions (P_c). Note that $P_{e+c} \subseteq P_e$.

Now we use structure and contextual structure preconditions (P_s and P_{cs}) as a set of constraints on the events to refine event skeletons into actual events. For all $e \in P_e$, we note $refine(e,r) \in \Psi$ the event obtained from the skeleton *e* and the set of preconditions $P_s(r) \bigcup P_{cs}(r)$ of the action $r \in R_e(e)$ using our test-bed generation based algorithm. The complete algorithm for *refine* is too long to be presented here. It strongly relies on the VDL model's operational semantics. It is based on a recursive interpretation of VDL terms with different rules for each VDL keyword.

This leads to the set of syntactically correct events:

$$E' = \left\{ refine(e,r), \forall e \in P_{e+c}, \forall r \in R_e(e) \middle| \forall p \in P_s(r) \bigcup P_{cs}(r), eval(p, refine(e,r)) = \mathrm{T} \right\} \tag{5}$$

$$F' = \left\{ refine(e,r), \forall e \in P_{e+c}, \forall r \in R_e(e) \middle| \forall p \in P_s(r), eval(p, refine(e,r)) = \mathrm{T} \right\} \bigcap \bar{E}' \tag{6}$$

E' is the set of possible events: all events in *E'* will be accepted by the agent and all accepted events belong to *E'*. Conversely, *F'* is the set of "currently impossible" events, i.e. events that are not acceptable by the agent in its current state but that would be accepted in a different state.

Once possible and currently impossible events have been generated, the selection algorithm tries to select the most appropriate one with respect to the user's command. The general idea is to compute the probability of every event in $E' \bigcup F'$ and to determine the maximum probability event in *E'* and *F'*.

For every node $n \in \Psi$ and for any concept $c \in C$, we note $contains(n,c) = \exists x \in sub(n) | c \in (tag(x) \bigcup attributes(x) \bigcup content(x))$, with *sub(n)* the set of all direct and indirect sub-elements of $n \in \Psi$. In other words, *contains(n,c)* is true iff *c* appears anywhere within node *n*. The probability *p(e)* of an event $e \in E' \bigcup F'$ is:

$$p(e) = \frac{\left| \{ c \in C | contains(e,c) \} \right|}{|C|} \tag{7}$$

We can build the set *E* of maximum-probability possible events and the set *F* of maximum-probability "currently impossible" events (*X*=*E* or *X*=*F*):

$$X = \begin{cases} \varnothing & if \max\{p(x), x \in X'\} = 0 \\ \{ e \in X', p(e) = \max\{p(x), x \in X'\} \} & otherwise \end{cases} \tag{8}$$

Moreover, $\forall e \in F$, we note *np(e)* the set of invalid preconditions that make this event impossible to process:

$$np(e) = \{ p \in P_c(e) \cup P_s(e) \cup P_{cs}(e), eval(p) = \mathrm{F} \} \tag{9}$$

Note that, by construction, all events in E (resp. F) have the same probability: $\forall e \in E, p(e) = p_e$ and $\forall e \in F, p(e) = p_f$.

6. USER'S FEEDBACK: THE DIALOGUE MANAGER

The dialogue manager (DM) is responsible for both command acknowledgement and management of misunderstood or imprecise commands. The DM will produce different answers depending on the different contextual situations.

We make use of two thresholds in [0.0,1.0] (as p_e and p_f). $p_{\min}$ is the minimum value for an event to be considered as possibly understood command and $p_{\max}$ is the further limit beyond which the event is considered as a correct representation of the user's command. They correspond respectively to the "tell me" and "do it" thresholds for Patty Maes in [11]. She proposed empirically to use a margin for accepting events: $p_{\min} = 0.3$ and $p_{\max} = 0.8$.

The answer given by the DM depends on the position of p_e and p_f with respect to $p_{\min}$ and $p_{\max}$:

1. If $p_e \geq p_{\max}$, the command is considered as correctly understood by the system. The DM either sends the event to the agent (when $|E| = 1$ |) or informs the user about an ambiguity (when $|E| > 1$). For instance:
   User: Take something red
   System: I can either take object square red, take object triangle red.
2. If $(p_{\min} < p_e < p_{\max}) \wedge (p_e < p_f)$, the best understood event is not possible ($p_e < p_f$) but something close was understood which is still possible ($p_{\min} < p_e$). The DM asks the user for a reformulation. It displays both failed preconditions of impossible events ($\bigcup_{e \in F} np(e)$) and the list of possible events E.
   User: Put it on the upper left cell (with upper left cell already occupied)
   System: I can't because: the content of upper left cell is not empty. Therefore, i can either: - drop object in the upper middle, in the upper right, in the center left or in the lower left
3. If $(p_{\min} < p_e < p_{\max}) \wedge (p_f < p_e)$, the impossible events can be ignored, but still the system is not sure about the user's command ($p_e < p_{\max}$). It asks for a confirmation by displaying events $e \in E$.
   User: Take the blue or red triangle form (while there is no blue triangle)
   System: Do you mean "take object triangle red"?
4. If $(p_e < p_{\min}) \wedge (p_{\max} < p_f)$, the system correctly understood an impossible command. It tells the user that this command is not possible by giving the list of failed preconditions $np(e), e \in F$.
   User: Take the blue red figure (with something already in hand)
   System: I can't because : the content of hand is not empty.
5. If $(p_e < p_{\min}) \wedge (p_{\min} < p_f)$, the system might have understood something but this command cannot be performed. The DM asks the user for confirmation.
6. If $(p_e < p_{\min}) \wedge (p_f < p_{\min})$, the system didn't understand the command and tells it to the user.

7. EVALUATION

Our experiment was conducted using a simple agent called Jojo inspired from Winograd's block's world [20]. This agent has two possible actions: to take an object or to drop it into a given position in a "grid". An object is characterised by its shape (shape $\in \{square, triangle, circle\}$), its colour (colour $\in \{red, green, blue, white\}$) and its size (size $\in \{tiny, small, medium, big\}$). A position is a couple in $\{upper, center, lower\} \times \{right, middle, left\}$. Examples in the dialogue manager's algorithm come from this experiment's corpus.

None of the eight subjects for this experiment had ever used the system before. They were given no information on the system's NLP capabilities. The aim was to reach a given particular state (see figure 2). No time limitation was given, but the subject could stop the experience at any time. After performing the task, the subject completed a questionnaire. The questionnaire asks for the subject's appreciations on the system's NLP capabilities.

The subject's evaluation of the system outlines the lack of semantic interpretation of commands, which make the system unable to understand complex command like "take the smallest triangle" or "drop it in place of the red form". This result was expected since we only use the *owl:sameAs* relation for "semantic" relations! Once the subjects had acquired the agent's vocabulary (i.e. the transitive closure of *owl:sameAs* in the ontology with respect to the agent's VDL concepts), no interpretation error occurs and the users are rather satisfied with the system. The feedback provided by the dialogue manager especially appears to be clear enough about what the agent expects from the user. From a mixed-initiative planning perspective, since the system always proposes a list of possible events when it cannot exactly find one with full probability of acceptability, the user knows exactly what the system expects. Moreover, users receive explanations about impossible commands.

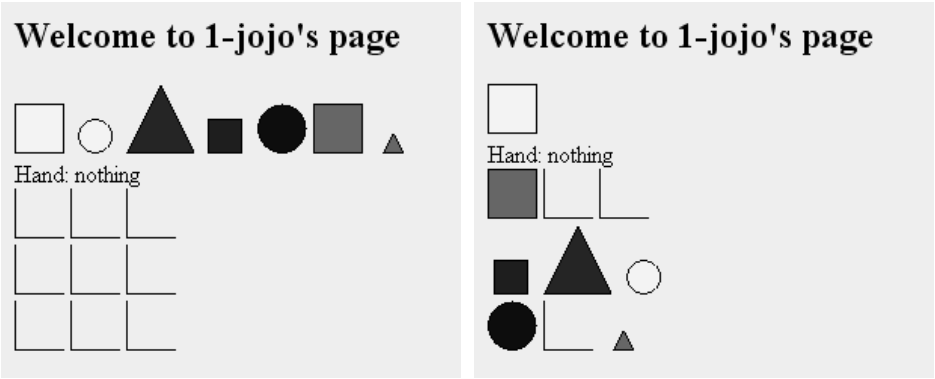

Figure 2. Operation example.

Note that this evaluation is a small part of a bigger evaluation made to compare three interpretation algorithms [12]. This section gives the evaluation results of the best of the three algorithms, "the bottom-up with feedback" presented here.

8. CONCLUSIONS AND PERSPECTIVES

In this paper, we proposed a general NLP architecture for command interpretation based on the idea that generic algorithms can be parameterized by the agent's code and a domain ontology. Our system relies on a constructive bottom-up approach based upon the action preconditions. Even if we use the VDL language for programming our agents, the approach is language-independent and can be easily adapted to others introspection-capable models.

We conducted an evaluation of our system that shows that the feedback provided by the DM allows the user to align on the agent's ontology. Our system tells the user why a given command cannot be performed and shows the system's expectations. Users have the feeling that the system is "more clever" with our

constructive bottom-up DM than with classical approaches. This evaluation also shows that the limitation of our system resides in the minimal semantic analysis on the ontology (synonymy). To overcome this issue, we propose to use advanced semantic distance measures (as given by [6] for instance) for associating the human command concepts with the agent's concepts in the ontology. The first work we have already done in this direction outlined the importance of the ontology definition. Using too general ontologies such as WordNet leads to misinterpretation at the semantic interpretation level. On the contrary, domain ontologies allows the programmer to define specific relations between agent concepts.

REFERENCES

1. ABRILIAN S., BUISINE S., RENDU C., MARTIN J.-C., *Specifying Cooperation between Modalities in Lifelike Animated Agents*. In Working notes of the International Workshop on Lifelike Animated Agents: Tools, Functions, and Applications, pp. 3–8, 2002.
2. ALLEN J., BYRON D., DZIKOVSKA M., FERGUSSON G., GALESCU L., STENT A., *Towards Conversational Human-Computer Interaction*. AI Magazine, 2001.
3. ALLEN J., MILLER B. W., RINGGER E. K., SIKORSKI T., *A robust system for natural spoken dialogue*. In ACL, pp. 62–70, 1996.
4. BATEMAN J. A., *Enabling technology for multilingual natural language generation: the kpml development environment*. Nat. Lang. Eng., **Vol.3(1)**, pp. 15–55, 1997.
5. BOTELLA B., TAILLIBERT P., GOTLIEB A., *Utilisation des contraintes pour la génération automatique de cas de test structurels*, Test de logiciel, RSTI-TSI, **Vol.21**, pp. 1163–1187, 2002.
6. BUDANITSKY A., HIRST G., *Evaluating wordnet-based measures of semantic distance*. Computational Linguistics, **Vol.32(1)**, pp. 13–47, 2006.
7. CASSEL J., SULLIVAN J., PREVOST S., CHURCHILL E., *Embodied Conversational Agents*. MIT Press, 2000.
8. CHARIF-DJEBBAR Y., SABOURET N., *An Agent Interaction Protocol for Ambient Intelligence*. In Proc. of the 2nd International Conference on Intelligent Environments (IE'06), pp. 275–284, 2006.
9. DZIKOVSKA M. O., ALLEN J. F., SWIFT M. D.. *Integrating linguistic and domain knowledge for spoken dialogue systems in multiple domains*. In Proc. of IJCAI-03 Workshop on Knowledge and Reasoning in Practical Dialogue Systems, 2003.
10. FERBER J., *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
11. MAES P., *Agents that reduce workload and information overload*. Communications of the ACM, **Vol. 37(7)**, pp. 30–40, 1994.
12. MAZUEL L., SABOURET N., *Generic command interpretation algorithms for conversational agents*. In Proc. Intelligent Agent Technology (IAT'06), pp. 146-153, 2006.
13. MILWARD D., *Distributing representation for robust interpretation of dialogue utterances*. In ACL, pp. 133–141, 2000.
14. MILWARD D., BEVERIDGE M., *Ontology-based dialogue systems*. In Proc. 3rd Workshop on Knowledge and reasoning in practical dialogue systems (IJCAI03), pp. 9–18, 2003.
15. PARAISO E. C., BARTHES J.-P. A., TACLA C. A., *A speech architecture for personal assistants in a knowledge management context*. In ECAI, pp. 971–972, 2004.
16. SABOURET N., SANSONNET J.-P., *Automated Answers to Questions about a Running Process*. In Proc. CommonSense 2001, pp. 217–227, 2001.
17. SABOURET N., SANSONNET J., *Learning Collective Behavior from Local Interactions*. In B. Dunin-Keplicz and E. Nawarecki, editors, From Theory to Practice in Multi-Agent Systems, Proc. CEEMAS 2001, **Vol.2296** of LNAILNCS, pp. 273–282, Springer-Verlag, 2002.
18. SABOURET N., GOLSENNE M., MARTIN J.-C., *VDL+LEA: complémentarité entre interaction multi-modale et interaction multi-agents*. In Proc. 1st Workshop sur les Agents Conversationnels Animés (WACA), pp. 13–22, 2005.
19. SADEK D., BRETIER P., PANAGET E., *Artimis: Natural dialogue meets rational agency*. In IJCAI 97, pp. 1030–1035, 1997.
20. WINOGRAD T., *Understanding Natural Language*. New York Press. 1972.