Integrating NLP with Reasoning about Actions for Autonomous Agents Communicating with Humans

Xiaoping Chen, Jiehui Jiang, Jianmin Ji Guoqiang Jin and Feng Wang
Multi-agent Systems Lab,
University of Science and Technology of China, HeFei, China
xpchen@ustc.edu.cn, {jhjiang,jizheng,abxeeled,fenggew}@mail.ustc.edu.cn

***Abstract*—We present a first effort to integrate NLP with ASP for autonomous agents, especially service robots, communicating with humans. We implemented a prototype system and tested it in a simple home environment, which demonstrated the feasibility of our approach under current settings and the possibility that our approach will benefit from future advancement of research on NLP, ASP, and related areas.**

***Keywords*-Human-agent communication, NLP, Planning, ASP.**

I. Introduction

There has been an increasing interest in research and development of various kinds of autonomous agents, in particular, service robots, in recent years [1], [2]. Many of these efforts are paying more and more attention to integrating techniques drawn from areas of AI and Robotics, including vision, navigation, manipulation, machine learning, planning, reasoning, speech recognition and natural language processing, and have made important and promising progress. Behind this trend is the fact that all the related areas have been developing fast and fruitfully. On the other hand, new challenges are expected to be confronted along with the opportunities in the process.

Human-agent dialogue in natural languages is obviously one kind of natural human-agent communication [1]. To the best of our knowledge, however, there is a little work in which natural language processing (NLP) techniques is integrated as a separate module into the software architecture of an agent, the closest to us is [3]. Instead, human-agent dialogue in natural languages is basically handled only by speech recognition technique, with which the information extracted is then fed into motion planning module.

If a substantial NLP module is integrated into an agent, there will be a bid two-fold challenge. On one hand, there are lots of difficulties that cannot be overcome by current NLP techniques. On the other hand, it is challenging to couple general-purpose planning with NLP. Even if the human-agent dialogue is restricted to some extent so that it can be handled with current NLP techniques, there is still a question of how to make planning autonomously by the agent with the information extracted from the NLP module.

This paper presents an effort to attack the challenge mentioned above. We employed NLP techniques to develop a substantial NLP module used for human-agent dialogue in natural languages. We also made use of logic-based planning techniques for task planning, aiming at more scalable and flexible planning capacity. One of features of the proposed approach is that the agent can solve complex tasks much easier comparing to previous work. A prototype system has been implemented and tested in a simple home environment to verify the feasibility of the proposed approach.

We present the basic requirements and a three-layer architecture for autonomous agents communicating with humans in Section 2. Section 3 and 4 report our work on the NLP and the planning modules, respectively. An example is given in Section 5 to show how complex tasks are solved by the prototype system. We give conclusions in Section 6.

II. The three-layer architecture

For ordinary users, using natural languages is the best way to communicate with agents in most cases. Meanwhile, it is undesirable to allow or ask human users to verbally control the agent. At least we should allow users to send following two kinds of messages to an agent in human-agent dialogue:

1) Commands, asking the agent to carry out some tasks.
2) Descriptions, telling the agent something about its environment that cannot be acquired through sensors, such as the name of the user.

Accordingly, an agent should be able to “understand” these messages expressed in natural languages. Therefore, there are three basic requirements for an agent. First of all, the agent must be able to “understand” dialogue in a natural language to some extent. Secondly, the agent must be able to plan its actions autonomously according to the user’s requests and/or information. Thirdly, the agent must be able to execute its plans under uncertain environments autonomously. We propose a three-layer architecture (Figure 1) to meet all the basic requirements. The first layer of the architecture consists of a human-agent dialogue and an NLP module, as well as a dialogue knowledge base. Its main function is to conduct the human-agent dialogue in some natural language and “understand”, through NLP techniques such as syntactic and semantic analysis, what the user says in the dialogue. The second layer is for task planning, with which each task required by the user will be decomposed

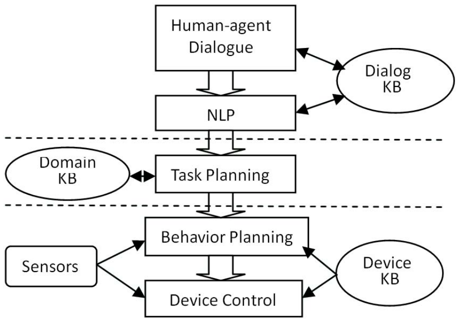

Figure 1. The three-layer architecture

and arranged into an appropriate sequence of subtasks with some classical planning technique. The third player consists of a behavior planning and a device control module, as well as sensors and a device knowledge base. Each subtask in the sequence worked out by the task planning layer will be executed by the third layer.

Since both the NLP and the task planning module employ logical forms as inner representation, their interaction is expected to be smooth. This makes it possible to integrate NLP techniques with logic-based planning techniques in the architecture. On the other hand, the connection between the second and the third layer is also expected to be smooth, since they only need to interact with each other around subtasks.

We have implemented a prototype system of sevice robot for the proposed architecture, where we assumed a small vocabulary with which the speech recognition and the task planning module operate. We used standard NLP techniques for our NLP module and Answer Set Programming as the tool for the task planning module in the prototype system.

III. Natural Language Processing

The process of semantic analyzing is divided into five phases. The first is to call a parser (in our implementation it is the Stanford parser) to get a grammar tree of the input sentence. The second phase involves copying the grammar tree into a similar semantic tree structure. The analysis there after is independent of the parsing process. In the third phase, each node of the analyzing tree is attached with a semantic rule, which takes its basic form as a grammar rule but includes further information for building logic forms. In the fourth phase, each node is labeled with its correct core label, and predicates, entities, and holes are created at leaf nodes, holes are filled at intermediate nodes. In the third phase and the fourth phase, the two times of tree walk are both in a bottom-up type.

Now, a logic form in SDRT [4] style is ready to be extracted. Then in the fifth phase, we transform the logic form into ASP style, discarding irrelevant information. We attach semantic rules to nodes of the analysis tree because each semantic rule brings with it specific operations to be performed on the node. The most important task is to fill holes—when two nodes meet on the tree, information on one node must be able to fulfill some requirement of the other node. Another task which is no less important is to determine which child node's core label would be take on by the parent node.

Other operations would sometimes be appropriate, such as changing the number of holes of a given predicate. This leads to a uniform style of process verbs with different numbers of objects, postponing the determination of the number of objects until all of the objects appear. When transcribing the SDRT logic form into ASP, we would need to merge the argument list of one predicate with some others', using the operation of changing the number of holes, too.

Semantic data consists of atom components. An atom component of semantic data can be a predicate indicator, an entity, or a hole. To represent a predicate indicator, we must record its own label, its print name, and the number of its holes, each hole occupies a place of future argument. formally, it is

$$pred_{l,s,n} \in P \text{ where } l \in L, s \in S, n \geq 0. \tag{1}$$

Here, L is the set of all labels S is the set of all symbols, P the set of predicates, E the set of entities, and H is the set of all holes. For an entity, its own label and its print name are necessary, which can be written as

$$entity_{l,s} \in E \text{ where } l \in L, s \in S. \tag{2}$$

A hole can be viewed as a map from the label of predicate and the number of the place of the hole to the label of the hole filler, which can be either a predicate or an entity. Thus, the label of its predicate, the number of its place, and the label of the filler must be specified.

$$\begin{aligned} &hole_{l_1,k,l_2} \in L \times N \to L, \\ &\text{where } l_1 \in L, 0 \leq k \leq n_{l_1}, l_2 \in L \cup \{\eta\}. \end{aligned} \tag{3}$$

$\eta \in L$ indicates a special label that would never be related with any predicate or entity, and would appear in an unspecified hole. The container of semantic data can be formalized as $M = P \cup E \cup H$, where an M contains the meaning of a sentence. Not only the holes are underspecified, the number of holes is underspecified, too.

Nearly all the predicates and entities in P and E are generated in leaf nodes(or more precisely, preterminal nodes, it's preterminal rules that governs the generation of the predicates and entities). On intermediate nodes, the work to be done is hole filling and the passing of core labels from bottom up.

Each word gets a predicate of its own on the leaf nodes of a analysis tree. Ultimately, function words like prepositions do not get independent meaning. They would be attached to some content words such as nouns or verbs. But this attaching can be postponed to after the logical form is produced. An ASP planner typically do not need large amount of information represented as predicates. Information such as $block(x)$, $block(y)$, $on(x, y)$ would be enough. So we only transcribe relevant predicates into ASP form.

Another thing that must be done to transform to ASP form is to merge verbs and related prepositional phrases. This exhibits the property of understanding on the situation, made possible by the flexibility of the semantic data representation. This involves mapping two predicates to one predicate. To transform $L_1 : go(x), L_2 : to(L_1, y)$ to $go\text{-}to(x, y)$, we have

$$merge(pred_{l_1,go,1} \times pred_{l_2,to,2}) = pred_{l_3,go-to,2} \quad (4)$$

More generally,

$$merge(pred_{l_1,s_1,n_1}, pred_{l_2,s_2,n_2}) = pred_{l_3,s_1s_2,n_1+n_2-1}. \quad (5)$$

It is required that $\exists h_{l_2,k,l_1} \in H$ where $k = 0$.

On the other hand, to make the entities appear directly in the ASP formulae instead of the labels of the entities, the module works to replace the labels with the print names of the entities.

IV. Task Planning

We follow the perspective proposed in [5] to use Answer Set Programming (ASP [6]) for the design and implementation of deliberative agents, which captures reasoning, planning and acting in a changing environment and has been used to tackle a variety of problems, including: diagnosis, planning, modeling and rescheduling of the propulsion system of the NASA Space Shuttle, multi-agent systems and Semantic Web and web-related technologies.

ASP is a form of declarative logic programming, whose syntax is similar to standard Prolog with stable model semantics, a model-based semantics for logic programs with negation as failure. Specifically, an answer set logic program is a finite set of rules of the form:

$$H \leftarrow p_1, \ldots, p_k, not\, q_1, \ldots not\, q_m, \quad (6)$$

where p_i, $1 \leq i \leq k$, and q_j, $1 \leq j \leq m$, are literals, and H is either empty or an literal. A *literal* is a formula of the form a or $\neg a$, where a is an atom. If H is empty, then this rule is also called a *constraint*. A rule consisting of only H is called a *fact*. Note that, the negative literal $\neg a$ represents classical negation, which is different from not, negation as failure.

A logic program with variables is viewed as shorthand for the set of all ground instances of its rules. The computational result of an ASP program is a set of answer sets [6].

The computation of a program is carried out by two components, a *grounder* which removes the variables from the program by instantiations and an *answer set solver* which computes answer sets of the propositional program.

Following the proposal given in [5], the initial state, domain knowledge of the environment and descriptions of robot's actions are declaratively expressed in ASP and stored in the knowledge base of the robot. For example, the robot's ability of 'move' and the corresponding predicate $location$ are expressed as follows:

$$1\,\{occurs(A, T) : action\_of(A)\}\,1 \leftarrow time(T), T < lasttime.$$
$$holds(location(Z, X), T + 1) \leftarrow occurs(move(X), T),$$
$$\quad not\, abnormal(wheel, T), agent(Z), number(X),$$
$$\quad time(T), T < lasttime.$$
$$\leftarrow occurs(move(X), T), holds(location(Z, X), T),$$
$$\quad agent(Z), number(X), time(T), T < lasttime.$$
$$holds(location(A, X), T + 1) \leftarrow holds(location(A, X), T),$$
$$\quad not\, \neg holds(location(A, X), T + 1), object(A),$$
$$\quad number(X), time(T), T < lasttime.$$

The first rule states that, at any time $T < lasttime$, one and only one action occurs. The second rule states the effect of the action $move$, if the agent moves to one place X at time T, then her location is X at time $T + 1$. The following constraint states an inexecutable condition for $move$, the agent cannot move to the place where she is. The last rule is the inertia rule for $location$, which concerns the frame problem.

For task planning, we represent commands and descriptions on the base of ASP rules and treat users' commands as statements for goal states, which are translated to sets of constraints.

In the end, the task planning problem becomes a logic-based planning problem, which is then reduced to the problem of finding an answer set of the ASP program. If the goal is achievable, then a sequence of actions (one action at each step) is contained in one answer set of the program, which stands for a plan to achieve the goal. We use Lparse to ground the program, and clasp, one of the most efficient ASP solvers, to compute answer sets in our system.

Task planning is related to both natural language processing and behavior planning. User's commands and provided information can be accessed from NLP module in the form of facts. With the help of the part of the program which translates commands to the corresponding goal states, these facts can be directly added to the KB of the robot, thus an answer set of the whole program is a solution to fulfilling the command with the help of information provided by the user. The sequence of actions (subtasks) computed from the ASP program is then passed to the third layer.

V. A Case Study

In this section we describe a case study on complex tasks. We present it with just a simpler instance for the following user command:

> The green bottle is on the cupboard. Give Jim the red bottle and put the green bottle on the table.

The first sentence is a description of the environment, the second is a command for a complex task, consisting of more than one simple tasks.

Through a process described in section 3, the command representing a complex task is translated into two sets of logical forms:

$$\{\, green(x),\, bottle(x),\, on(x,y),\, cupboard(y) \,\},$$
$$\{\, give(v,y,x),\, Jim(y),\, bottle(x),\, red(x),\, put(v,z,t),$$
$$bottle(z),\, green(z),\, table(t),\, \_robot(v) \,\}.$$

The task planning module transforms the above sets of formulas to corresponding rules in ASP,

$$\begin{aligned} on(A,B) &\leftarrow bottle(A), green(A), object(A), object(B),\\ &\quad cupboard(B).\\ give(V,B,A) &\leftarrow agent(V), bottle(A), red(A), object(A),\\ &\quad jim(B), object(B).\\ put(V,A,B) &\leftarrow agent(V), bottle(A), green(A), object(A),\\ &\quad table(B), object(B).\end{aligned}$$

The predicate $on(A,B)$ is explained as follows:

$$location(A,X,0) \leftarrow on(A,B), location(B,X,0),\\ number(X).$$

The above rule states that, if object A is on the object B, then A is at the location of B. From the information "The green bottle is on the cupboard", the task planning module derives that, the *green bottle* is at the location of cupboard. The predicates give(V,B,A) and put(V, A,B) are explained with the same principle.

In the domain knowledge base, object 12 is the *red bottle*, which is on the teapoy and is at the location 6; object 17 is the *green bottle*, we have told the robot that it is on the cupboard and the location is 3; finally, *Jim*'s location is 10 and the location of the *table* is 5. For the above example, we get six different answers, two of them are:

$$\langle\, move(3),\, catchl(17),\, move(6),\, catchr(12),$$
$$move(10),\, putdownl(17),\, move(5),\, putdownr(12) \,\rangle,$$
$$\langle\, move(3),\, catchl(17),\, move(10),\, putdownl(17),$$
$$move(6),\, catchl(12),\, move(5),\, putdownl(12) \,\rangle,$$

where $move(X)$ means the robot moves to the location X, $catchl(A)$, the robot catches the object A and put it on the left plate, symmetrically, $catchr(A)$ means catching the object A and put it on the right plate. Similarly, $putdownl(A)$ means putting down the object A which is on the left plate, and $putdownr(A)$ means putting down the object A which is on the right plate. For instance, the first plan states that, first the robot moves to the location 3 (*cupboard*), catches 17 (*green bottle*) and put it on the left plate, then it moves to the location 6 (*teapoy*), catches 12 (*red bottle*) and put it on the right plate, later the robot moves to 10 (*Jim*) and puts 17 down from the left plate, last moves to 5 (*table*) and puts 12 down from the right plate.

The robot can get the distance between two locations from the domain knowledge base, then she chooses the plan with the shortest path and pass it to behavior planning layer for execution. Our robot has two plates to hold patable objects and thus has sufficient hardware support to this kind of complex tasks.

VI. Conclusions

We present a first effort to integrate NLP with ASP for autonomous agents, especially service robots, communicating with humans.It is expected to gain better ability of human-agent communication, comparing to the current human-agent dialogue mainly through speech recognition techniques, and have more scalable and powerful planning/reasoning capacity. We implemented a prototype system, which demonstrated the feasibility of the proposed approach under current settings and also provided a platform for further research. We hope that our approach will benefit from future advancement of research on NLP, ASP, and the related topics such as SAT.

Acknowledgement

This work is supported by the Natural Science Foundations of China under grant 60745002 and the National Hi-Tech Project of China under grant 2008AA01Z150. We thank Mary-Anne Williams, Fengzhen Lin, Enrico Pagello, Shlomo Zilberrstein, Yan Zhang, and Dongmo Zhang for helpful discussions about this work. We are also grateful for the anonymous reviewers for their constructive comments.

References

[1] M. Quigley and A. Y. Ng, "STAIR: hardware and software architecture," in *AAAI 2007 Robotics Workshop*, 2007.

[2] P. Doherty, J. Kvarnström, and F. Heintz, "A temproal Logic-Based planning and execution monitoring framework for unmanned aircraft systems," *Journal of Automated Agents and Multi-Agent Systems*, 2008.

[3] P. Drews and P. Fromm, "A natural language processing approach for mobile service robot control," in *Proceedings of twenty-third International Conference of the Industrial-Electronics-Society*, 1997.

[4] N. Asher and A. Lascarides, *Logics of Conversation*. Cambridge University Press, 2003.

[5] M. Gelfond, "Answer set programming and the design of deliberative agents," in *Proceedings of Twentieth International Conference on Logic Programming*, 2004, pp. 19–26.

[6] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.