Efficient Knowledge Representation in Intelligent Human-Robot Co-operation

Annamária R. Várkonyi-Kóczy
Institute of Mechatronics and Vehicle Engineering
Obuda University
Budapest, Hungary
Email: varkonyi-koczy@uni-obuda.hu

Balázs Tusor
Integrated Intelligent Systems
Japanese-Hungarian Laboratory
Budapest, Hungary
Email: balazs.tusor@gmail.com

Abstract—With the spreading of intelligent machines, man-machine co-operation has become an important research area. Today, intelligent robots collaborating with humans usually have to be able to store, retrieve, and update information about their environment, interpret and execute commands, offer existing and gain/learn new services. In these processes, the efficient knowledge representation and storage are of key importance. In this paper, a new graph based knowledge storage and representation form is introduced which distinguishes between theoretical knowledge and linkage-possibilities among virtual tools and their real-word embodiments. The proposed modular structure makes easy to store, retrieve, modify, and extend theoretical and practical knowledge, to interpret commands and to associate them with physical means and actions, to adapt to changes, learn and build in new knowledge. These aspects are especially important when the robots are involved in human-robot communication, as well.

I. INTRODUCTION

Today, with the spreading of machine intelligence, smart environments and intelligent robots (robot systems) have become an important research area. These agents and tools are in many sense part of our everyday life aiming to increase humans’ life quality and comfort (see e.g. [1]). These applications may also offer support to disabled persons thus helping them in their full and independent life (see e.g. [2]).

Intelligent robots collaborating with humans usually have to be able to store, retrieve, and update information about their environment, interpret and execute commands, offer existing and gain/learn new services ([3]). The intelligent human-robot co-operation usually involves some kind of communication, as well ([4]).

In these processes, the efficient knowledge representation and storage are of key importance. The size of the databases, the accessibility to the stored knowledge, the possibilities of building in new or refining the possessed information together with the flexibility of the information update have a direct effect on the speed and effectiveness of the cooperation. All these ask for efficient data and knowledge structures.

There can be found different knowledge representation approaches in the literature. Among the known methods there are some which can advantageously be applied in man-machine co-operation, as well. We have to mention e.g. conceptual graphs (CG) (see [5]).

CG has primarily been developed for database interfaces, to make it easier for humans to understand the data and to make inquiries. This approach applies the concepts as basic primitives (depicted as boxes) and the concepts themselves are types which incorporate every instance that shares that type. CG has defined constraints like well-formed-ness and transformations that keep that property.

In [6] a concept graph based knowledge model is proposed to represent concepts, terminologies, methods, and processes in Software Architecture. The knowledge is classified into a hierarchy of 4 levels: fundamental concepts, domain knowledge, process knowledge, and task knowledge.

Fundamental concepts are atomic knowledge units that represent basic ideas of a subject and are described by tuples. Domain knowledge refers to types or constructs of fundamental concepts. It can take the form of a relation, a rule-type, or a structure. Process knowledge corresponds to some operational procedures that represent basic system functionalities. Finally, task knowledge is user-oriented and usually centered around some system goals to achieve.

[7] proposes a graph-based knowledge representation for Geographic Information System (GIS), to represent spatial and non-spatial data (nodes), also including spatial relationships (edges) between spatial objects and use the model for generating a dataset composed of both types of data, so they can apply a data mining technique.

The authors of [8] propose the Feature Event Dependency Graph (FEDG). It focuses on representing the fact level knowledge compressively however without losing any important information. It is efficient in retrieving user concerned knowledge patterns and is especially useful in discovering latent knowledge and in effective reasoning. In FEDG, the knowledge is represented by feature events (nodes) and the weighted context links and dependency links (directed edges) between them.

In this paper, a new graph based knowledge storage and representation form is introduced. Our approach principally belongs to the wider family of conceptual graphs and can be considered as a more specialized version of CG however with differences, thus becoming a novel approach. First of all, it has specifically been developed for control systems, like the iSpace ([9]). While in CG the concepts are themselves...
types, which incorporate every instance that shares that type, in our structure the instances are modeled as separate nodes in the instance domain. We do not define constrains like CG does, etc. However, possibly the main difference between our approach and the conceptual graph approach is that instead of relation nodes we apply specific edge types.

The knowledge representation graph proposed in this paper distinguishes between theoretical knowledge and linkage-possibilities among virtual tools and their real-word embodiments. This disassociation of the theoretical knowledge and its embodiment in the real environment results in that a sophisticated modularity can be kept in the knowledge storage and the environmental changes have less effect on the knowledge structures. This way, the redundancy of the representation becomes also lower than that of the traditional structures. The proposed solution makes easy to store, retrieve, modify, and extend theoretical and practical knowledge, to interpret commands and to associate them with physical means and actions, to adapt to changes, learn and build in new knowledge. These aspects are especially important when the agents/robots are involved in human-robot communication as well.

The paper is organized as follows: In Section II knowledge representation is addressed. Section III deals with knowledge harmonized command interpretation and processing while Section IV is devoted to knowledge based hypotheses building. Finally, Section V concludes the paper and presents further possible research directions.

II. KNOWLEDGE REPRESENTATION

The main parts of the graph based knowledge representation are the knowledge base, the hypothesis storage, and one or more dictionaries.

A. Knowledge base

The knowledge base holds the knowledge of the system. It is realized as a graph-based structure, divided to abstract and instance domains. In the abstract domain, the nodes denote the known abstract objects and concepts and the directed edges between them describe their relationship, thus describing the knowledge of the system about the world in general. In the instance domain the nodes represent existing objects that the intelligent machine (robot, iSpace) knows about. The nodes are homogeneous and nameless (for an example, see Fig. 1). Whenever the world of the robot is altered in any way, the changes should appear in the appropriate domains of the knowledge base as well (physical changes (addition or removal of sensors, detectors, agents, etc.) in the instance domain, while property changes (temperature of the room, etc.) in the abstract domain).

Figs. 2 and 3 show structural examples for the realization of the knowledge base. (All the examples are taken from the iSpace application published in [10].) Fig. 2 depicts that tea, coffee, and cappuccino are drinks, beverages are synonymous to drinks, drink machines can make drinks, brewing is (in this case) synonymous to making, and actions done to drinks are heuristically dependent on time and mood. Fig. 3 shows an example how can we define minimum and maximum values for variables together with the used step size (granularity).

The different types of edges used in the proposed knowledge base are the following:

- Ability "edge": It is a three-way connection, where node A is connected to nodes B and C, if A can do C to B. E.g., A = "drink machine", B = "drink", C = "make" means "drink machines can make drinks" (see Fig. 2).


- **Instance edges:** An instance edge (denoted by an arrow with label I) assigns the address of an executive agent to a node, e.g., "window opener". The assigned address is the physical address of the connected window opener device (see Fig. 1). This is a flexible way to bind executive agents to nodes describing them.

- **Meta edges:** node A is connected to node B, if the concept A has a numerical value in B quality. E.g., A = "day" (day of the week), B = "value" and the value is "6" (meaning that "the value of the day is 6" i.e., "it is the 6th day of the week"). With the application of meta edges *environmental* variables can be appointed, which can store the values of sensors or any other numerical values. The range of the value can also be described by creating and setting the values of the meta edges thus connecting nodes *minimum value* and *maximum value* to the appropriate node (see Fig. 3).

- **Heuristics edges:** Node A is connected to node B, if B heuristics can be bound to concept A. B is usually an environmental variable. E.g., actions done to "coffee" are dependent of "time" and "mood" (given that the human user usually likes to drink it at a certain time each day, and is a certain mood when he or she gives out the command). It is denoted by an arrow with label H.

- **Synonym edges:** Node A is connected to node B by a synonym edge (denoted by a symmetrical connection with label S) if the two concepts are synonyms, like in our case "drink" and "beverage". By this, the robot can interpret instructions more flexibly.

- **Association edges:** Association edges (denoted by an arrow with label A) build a connection between two nodes (with dedicated relationship) of the instance domain. E.g., the instance of a particular window opener is associated to a certain instance of window (Fig. 1).

- **Fuzzy edges:** Fuzzy edges are similar to meta edges, except that they handle fuzzy values instead of numerical values. E.g., A = "Saturday", B = "day" and the assigned value is given by the membership function that can be seen in Fig. 4, for input value x. (x = 0...6, assuming the days of the week start with Sunday (with index 0).

In Fig. 1 an example is shown how can we represent our knowledge about the existing windows and their openers. A window can be opened and closed by its opener. The actions depend heuristically on the temperature of the room and how used up is the air in the room. Let us consider that there are two windows and window opener devices in the room (denoted by the side of the room where they are located: east or south), and the openers are associated with the appropriate windows they can do actions to, thus it is trivial which device can operate on which window. The user can instruct the robot to open or close one or both windows. Each node is identified by the dictionaries.

To be able to represent instances of objects which do not exist at the moment but can be made by devices operated by the robot (e.g., coffee) the void nodes are introduced. A void node is the instance of the concept of the product. There is a void node for each product and they are associated with the instance (which is the drink machine, in the case of coffee) that can create the product.

**B. Hypothesis Storage**

The robot is able to build hypotheses about the environment and the habits of its user. The hypotheses are stored in the *hypothesis storage*. A hypothesis consists of (pointers to) the action and object nodes. It can also have optional numerical values, similarly to the command the hypothesis is based on. A hypothesis also has at least one *trigger*, which consists of a *justification* value that denotes how reassured the system is in the trigger of the hypothesis; and at least one *condition*. A *condition* is derived from an environmental variable: it consists of a condition node (the node of the environmental variable), value (the value stored in the meta edge connecting the environmental variable to node "value"), *sensitivity* (which can be derived similarly to value using node "granularity" and the appropriate meta edge), and *affirmativeness*, which is a Boolean value. If its value is false then the condition is used as if it would be negated. (Thus the trigger will be triggered only if the condition is not fulfilled.)

**C. Dictionaries**

For identification, dictionaries are used (each consisting both abstract and the instance domains), which assign analogous words to the nodes. Here, as example, only one (English) dictionary is used, although the concept is designed for the usage of multiple dictionaries. Further, non-verbatim dictionaries (e.g., based on hand signs, see [11]) can also be included in the system. In general, the dictionaries are "translated" (traced back) to an equivalent, non-language-dependent dictionary during the preprocessing. This way the knowledge base can be kept independent of natural languages.
The most basic form of dictionaries is an ordered list containing the words (and expressions) and the reference to their corresponding nodes. The construction of the dictionaries and the knowledge base happens simultaneously; whenever a new node is created, a new entry is added to the dictionaries if the concept of the node is described by a word or expression that is not included in the dictionary yet.

The system creates and manages hypotheses based on the interpreted commands (see Subsection IV).

III. KNOWLEDGE HARMONIZED COMMAND INTERPRETATION AND COMMAND PROCESSING

In order to achieve advanced command processing, the system has to use grammar rules. These contain rules that determine how the words and sentences can be built with regards to the language. The modules of the Command Processor analyze the given command (with regards to its language and grammar) and instruct the appropriate executive agents to carry it out. These modules are the Command Parser Module (CPM), the Command Interpreter Module CIM), and the Instructor Module (IM).

A. Command Parsing

The CPM first determines the type of the command, and then parses it. Latter step depends on the type of the command. The input of the CPM is the pre-processed command (which is converted into the form of a whole sentence by the CPP modules) and its output is the parsed command. Two types of commands are defined: instructions and prohibitions. (For their structure, see Fig. 5.)

Instructions are simple commands. They are given by the human user to achieve change in the environment or by the robot itself due to the Autonomous Action Planning, which is the result of learning. The first part of instructions describes the action that is needed to be executed, possibly followed by an optional "all" word, which means that the action is needed to be executed to all available objects (e.g., "open all windows"). The second part describes the object of the action. After that there can be put optional numerical values.

Prohibitions are commands that are given by the human user to alter the behavior of the system by bounding one or more commands that were already learned, thus achieving change in the hypotheses. Their structure is basically the same as the structure of instructions except that they start with "DO NOT" and have additional (at least one) text parameters, which usually represent fuzzy variables. E.g., in case of the prohibition command "DO NOT make coffee on Saturday"; "Saturday" is defined by the fuzzy membership function shown in Fig. 4 (the input value can be retrieved from the meta edge connecting nodes value and day). Since the granularity of Saturday is 0.2, early Sunday morning and late Friday night count as "almost Saturday" (this is why a fuzzy set is used for the definition of the day instead of a crisp function).

This offers an efficient way to handle situations, where certain instructions are regularly given except some particular cases. And further, the definition of the occasions can hardly be defined sharply. The example presented here is the alarm setting. The person asks the robot to set the alarm to 7:00 a.m. every day. The forbidding command is formulated by the human after he/she is awakened at 7:00 on Saturday morning, as well. As "Saturday" is defined as a fuzzy notion, the robot will not set the alarm to 7:00 on the following late Friday evenings, Saturdays and early Sunday mornings.

Note, that this does not exclude that the system learns and uses parallel with the previous command another alarming hypothesis, like "set the alarm to 9:00 on Saturdays". The latter will be handled as different command, because of the differing numeric parameters.

The membership functions of fuzzy edges are defined by the person (so-called knowledge manager) who is in charge of creating (setting the a priori knowledge, e.g. heuristics edges) and managing the knowledge base of the system.

Since the command is in a pre-defined format (which is based on the strictly defined word order of the English language), the algorithm of the parsing phase is quite trivial. To mention an example, in case of instructions the first word is always the action and the second one is either the "all" word or (after the removal of the occurring articles, like "the") a noun that gives the object of the command.

Fig. 6 shows an illustration for the advanced command analysis. The system first parses the sentence to separate words, then analyses each part. From that, it produces a graph, where the bigger circles are references to the appropriate concepts of the knowledge base, while the smaller ones connected to them are features of those words (e.g. a pronoun belonging to the noun, is it plural or singular, etc.). Lastly, the squares denote the words-part of speech. This way the commands that the human users can give become more flexible, the users do not need to stick to a rigid order of words if it is unnatural in their native language. (The method used for building the graph, as well as the structure of the grammar rules database is still under development.)

B. Command Interpretation

The function of the CIM is to determine which executive agent is able to carry out a given command. Its input is the pre-parsed command and its output is the address or reference of an executive agent.
The CIM only processes instruction commands since prohibition type commands are not needed to be executed. The algorithm searches for three key nodes, in order: the object node, the action node and the executive node.

These three nodes are needed to be connected via an ability edge (in such way that represents the following: the concept of the executive node can do the concept of the action node to the concept of the object node).

The object node is a node that either corresponds to the object of the command (through the dictionary) or can be reached from the corresponding node through inheritance and/or synonym edges with a constraint that the node is needed to have at least one instance (a node in the instance domain, which is bound to the object node with an instance edge).

The action node is a node that either is corresponding to the action of the command, or can be reached from the corresponding node through inheritance and/or synonym edges, similarly to the object node, with the difference that the action node does not need to have any instance.

The executive node is a node that can be found the same way as the object nodes, with the difference that its instance is needed to be associated with the instance of the object node.

Thus, the algorithm effectively does the following: First, it finds out what concept the object of the command is, then what action is needed to be done to the object, and finally, what concept can do that action to the object. If the algorithm cannot locate the three previously defined nodes, then it stops: the instruction cannot be carried out. If it finds three nodes that satisfy all of the constrains defined above, it returns the address or reference of the executive agent that is stored in the instance node of the drink machine agent stored in the instance node.

C. Instruction and Execution

The task of the Instructor Module is to instruct the executive agents of the robot/robot system chosen by the interpreter to execute the task to provide the desired service for the user.

In advanced man-machine co-operation the communication is usually bi-directional, i.e. the robot (agent) has to be able to construct questions or give information in a way that the human user/partner can easily understand. In case of verbatim communication, this can be solved by the usage of voice synthesizers (see e.g. [12]) or by simply writing the message out to a screen. Either way, the usage of grammar rules is necessary, since communicating with the user in his/her native language is the most convenient for the person. To achieve this, an advanced, language dependent grammar rule representation and processing method is needed.

IV. Knowledge Based Hypotheses Building

An intelligent robot also has to be able to learn human reactions (e.g. instructions) together with the circumstances in which they appear and automatically initiate (execute) actions if the conditions become similar to the learned situation. A part of the robot’s intelligence, the so called Autonomous Action Planner (AAP) shall be responsible for learning via hypotheses and for decision making whether or not to take actions according to what the system has learned. It can be built of Hypothesis Trainer (HTM) and a Hypothesis Trigger Checker (HTCM) parts.

A. Hypothesis Training

The task of the HTM is to determine which executive agent can execute the command. Its input is the parsed command and has no outputs. The algorithm of the hypothesis training works, as follows:

If the command is an instruction, then it searches for a hypothesis that has the same action, subject, and numerical parameters. (There can be only one hypothesis like that, i.e. it is sufficient to get the first found). If there is none found, then a new hypothesis is created using the parameters of the command and a new trigger and new conditions are added with the (environmental variable) nodes connected to the subject node with heuristic edges. There will be as many conditions as many heuristic edges are connected to the subject node. The value and granularity of each condition is derived from the current value and granularity of the environmental variable.

If there already is such a hypothesis, then the algorithm checks its triggers. If there is a trigger with conditions triggered by the current values of the environmental variables, the algorithm increases the justification of that trigger and end the algorithm. If there is not any trigger like that, then a new trigger is added using heuristics just like it is explained above.

If the command is a prohibition then the algorithm searches for a corresponding hypothesis with triggers. If it does not have any triggers, it adds a new trigger to it. If there is one then adds a new condition to all its triggers using the negated fuzzy membership function of the prohibition.
The task of the Hypothesis Trigger Checking (HTCM) is to frequently check the conditions of the hypotheses in the hypothesis storage. If one is triggered, the HTCM sends the command of the hypothesis to the CIM, which instructs the appropriate executive agents to carry out the command.

For the hypothesis building, consider the following example: The human gives certain commands to the intelligent robot at certain times (e.g., "open the curtains" after waking up at 7:00, "make coffee" at 7:33 and 12:00, etc.). The robot makes and manages hypotheses based on these commands. The justification threshold of the triggers of hypotheses is set e.g. to 2, thus the system is only able to give out the command of the triggered hypotheses if the command has been detected at least 2 times under the same circumstances.

Fig. 7 shows an example for a learned hypothesis based on the instruction "make coffee". The command has been given at 7:33 and 12:00 (the parameters after "value" are: the value and granularity of the environmental variable and the affirmativeness of the condition). (In this example, the mood of the user was found frustrated (2) and neutral (3)).

Continuing the example, consider that the man’s weekday and weekend schedules are different, though the robot does not know about this on the first time, so it makes coffee at 7:33 on Saturday, as well. In reaction, the user gives a prohibition: "DO NOT make coffee on Saturday". Thus, the hypothesis based on command "make coffee" is modified by the complement of fuzzy membership function Saturday as is shown in Fig. 8, meaning that the robot will not make any coffee if the value of the environmental variable "day" equals to 6. (If later the user decides that he wants to drink coffee after all on Saturday, the robot will add a brand new trigger to the hypothesis.)

V. CONCLUSIONS

Human-robot co-operation asks for efficient knowledge storage and representation models which have a direct effect on the accessibility to the knowledge and also on the flexibility and adaptivity of the application and update of the knowledge.

In this paper, a new graph based structure is proposed which principally belongs to the wider family of conceptual graphs. The novelty of the presented structure lies in its specific edge types and that it stores and handles theoretical (abstract) and practical (real-word) knowledge separately. As result, the proposed model makes easy to store, retrieve, modify, and extend theoretical and practical knowledge, to interpret commands and to associate them with physical means and actions, to adapt to changes, learn and build in new knowledge which aspects are especially important when the agents/robots are involved in human-robot communication as well.

Our future research will address building in user dependent causality, active bi-directional interaction between the robot (robot system) and the user, and improved dictionaries, command preprocessing, command parsing, and command interpretation forms.

ACKNOWLEDGMENT

This work was sponsored by the Hungarian National Scientific Fund (OTKA 78576).

REFERENCES