A Knowledge-Based Service Approach for Human-Centered Robots

M. Omar Faruque Sarker[1], Jung-Min Park[2], ChangHwan Kim[3], and Bum-Jae You[4]

[1]School of Engineering, University of Science & Technology, Daejon, Korea, e-mail : writefaruq@gmail.com
[1,2-4] Center for Cognitive Robotics Research, Korea Institute of Science & Technology, Seoul, Korea
e-mail:(pjm, ckim, ybj)@kist.re.kr

Abstract— **This paper proposes a knowledge-based approach for adopting user's preferences and policies to plan and execute various services by a human-centered robot. To create an image of an obedient assistant in user's mind, we have suggested developing a robot that can customize its services according to user's guidelines. For materializing our idea, we have presented our Knowledge-Based Human-centered (KBH) architecture. Using the deductive database technology, we have also presented some results demonstrating how our robot can take the high-level knowledge of user's guidelines into account and plan and execute an object delivery service accordingly.**

I. Introduction

Rapid development in the field of service robotics indicates that robots are going to become part and parcel of our everyday life. Some key challenges for popularizing service robots are personalization and adaptation of services to various contexts, such as location, time, user etc., prioritizing services based on contexts, increasing overall user friendliness and so on. However, complexity of an intelligent service robot is increasing with time. So, the robot technology developers must consider developing more intelligent and interactive systems for higher adaptation and personalization of robots.

One way to achieve this requirement is to put some degree of autonomy or intelligence into robot behaviors based on the sensed and perceived information from the environment. This reactive strategy has effectively solved many problems related to planning and execution of low level tasks like localization, navigation, dynamic path planning and so on. But in case of human-centered robots, only the sensed and perceived information from robot sensors are not enough for planning and executing a high level service task. To illustrate this problem, let us consider a scenario where a humanoid robot is standing in front of two persons in an office environment and it is asked to move to the nearest person and shake hand with him. This is a trivial problem for most of the robots as it can sense the nearest person by suitable range sensors and approach him easily. However, potential problem arises when the user asked the robot to shake hand with the most senior person first (as a part of usual official norm). How the robot can find the most senior person by its sensors? This problem might not be solved generally by supplying a generic vision algorithm to find older person from skin characteristics since the definition of a “senior” person usually differs from place to place. For example, in an office environment, seniority is defined in terms of service designation and/or service period.

Therefore, it is obvious that human-centered service robots will experience similar kinds of problems in everyday service tasks as robot’s knowledge, sensed and interpreted by it, is not enough to plan/execute a human-centered service. It needs to consult knowledge-bases of user-defined relationships, policies, guidelines etc. specific to its working environment. For our later reference, we have defined the former kind of intelligence as *manufacturer-defined intelligence* and latter one as *user-defined intelligence*.

The impact of user-defined intelligence can be realized by observing real-life cases where we can expect that after purchasing a robot, a user brings it to his home and introduces it with a few things like members of his family or office, locations and other information related to a particular service. After that, certainly, the user will expect personalized services from this robot based on the existing norms and manners of his living environment. These high level manners may include serving senior persons with higher priority, providing urgent services promptly with special care, following individual preferences as much as possible and so on. However, in our knowledge, not a single system considers this kind of user requirements very seriously.

Therefore, we have investigated how a service robot can incorporate high-level knowledge of user preferences and guidelines as well as how it can customize a specific service that will satisfy those guidelines. For this purpose, we have considered a real-life scenario of an office service robot delivering objects like mails, faxes, CDs and confidential documents to the members of our office. We have defined several policies for this object delivery service including policies for confidential delivery, urgent delivery and regular delivery. Using our proposed Knowledge-Based Human-centered (KBH) software architecture, we have implemented our approach in an open source mobile robot simulator Player/Stage [13, 14]. So, the main focus of this paper is how to implement user policies in a knowledge-base of a robot and to demonstrate how this knowledge influences the planning and execution of its service accordingly.

Our research enables an intelligent service robot to follow the user guidelines like priorities and restrictions for a service. Clearly, our approach of teaching the robot is deductive one. In contrast to the inductive approach where robot learns from accumulated data [11], here it merely infers any executive decision from its own observations. Rather, it always checks the user supplied knowledge for deducing a new decision for planning and executing a service. In this way, we have turned our service robot into an obedient assistant to the human.

Rest of this paper: Section II presents a brief survey of related work. An overview of our KBH architecture is presented in Section III. In Section IV and V, we describe policies for a service scenario and implementation of that scenario with a detail treatment on knowledge-base design, development and testing. Finally, we present KB performance and scalability analysis in Section VI and conclude our paper by our conclusion in Section VII.

II. RELATED WORK

Many active researchers have been focused their attention to design and develop intelligent service robots (ISR) for various human services either from the robot manufacturer's or from application service developer's point of view. This implies that existing research efforts are mainly on hardware and software design, development, integration and deployment in a designated environment. Some examples include: Care-O-bot for home assistant service [1], Pearl for nursing homes [2], IROBI for child education [10], tour-guide Rackham in museum [9], social hospital service robot IHSR [8], and home cleaning robot Roomba [12].

Sakai et al. [6] designed an office service robot and presented some human-robot communication scenarios based on a XML language. But it is not clear how the end-user can customize the different services in different environments. Several other research groups integrated the service robots in real service scenarios and thoroughly reviewed their systems by end-users. Many interesting results have been published in [1-3, 9-10, 12]. In case of Roomba [12], a cleaning robot, users expressed their expectation for seeing a robot always as an intelligent machine that would gain knowledge over time and adapt its behavior accordingly. In case of a tour-guide robot presented in [9], different kinds of persons (e.g., children, adults, elderly etc.) showed different kinds of views while approaching the robot. However, in all cases, there was hardly any facility to modify the behavior of a robot for a particular service based on environmental or contextual knowledge.

Human robot interaction is one of the major research issues in service robotics. In [4], Thrun reviewed major research efforts to improve the interface between human and robot. It seems that most of those efforts have been made to improve the perception and communication capabilities of robot, although the adaptation of robot behaviors or services according to human guidelines is still not taken into account.

III. KNOWLEDGE-BASED HUMAN-CENTERED ARCHITECTURE

Our proposed "Knowledge-Based Human-centered" (KBH) software architecture, as shown in Fig. 1, is organized into three layers: Controller, Reactive-Executive and Deliberative. It has six major modules: *Controller, Reactor, KB Planner, Service-Provider, Memory-Manager* and *HRI-Panel*. Briefly, this architecture has been outlined here.

1) Controller: This module directly controls the robot. Since we assumed that the manufacturer is responsible to manage the technical details and to export the necessary interfaces to the application developers, we haven't elaborated this layer.

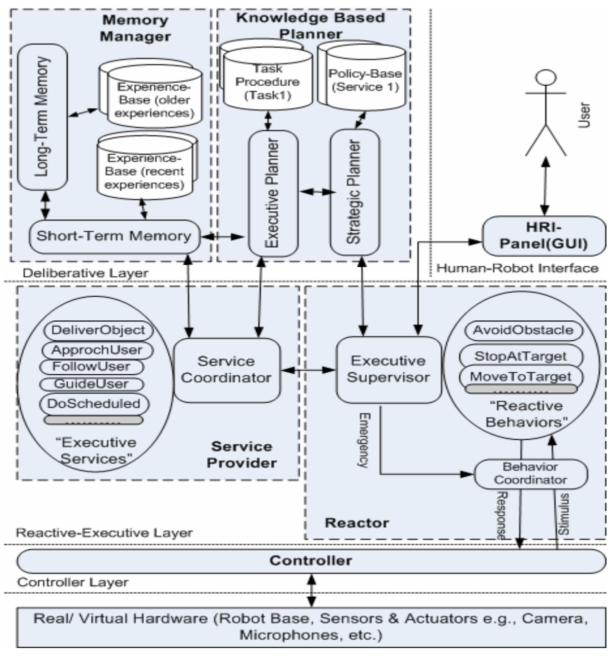

Fig. 1 Knowledge-Based Human-centered (KBH) architecture

2) Reactor: This module utilizes the low level facilities provided by the controller to drive the robot. We have assumed that the robot manufacturer provides us necessary libraries to crate reactive robot behaviors (as shown in right side ellipse in Fig. 1). We have placed *ExecutiveSupervisor (ES)*, the kernel of our architecture, in this layer. This ES is primarily responsible to accept user request for services, to consult planner for a decision and to activate /deactivate the reactive behaviors for various purposes. Occasionally, it uses the *BehaviorCoordinator (BC)* module to send emergency commands to halt the system.

3)KB Planner: The most notable feature of this architecture is two distinct kinds of planners: *Strategic Planner (SP)* that judges and decides whether a service will be started by the robot or not and *Executive Planner (EP)* that explains the *Service Coordinator (SC)* module about how to perform any specific tasks under a service e.g., navigation, path planning etc. They gather necessary information and instructions from relational and deductive databases connected to them.

4) Service Provider: In this module, the *Service Coordinator (SC)* takes care of activating and deactivating its necessary plug-in services *(*as shown in left side ellipse in Fig. 1*)* upon receiving requests from *ES*. *ES* can request to start any user specified service after consulting with the *SP*. *SC* also receives task plans from *EP* and it can request *ES* for activating any behavior to run a service.

5) Memory Manager: This module gives us the provision for remembering the robot's various experiences while planning and executing services as well as other information for navigation and path planning. Temporary working information is placed in *Short-Term Memory* and useful accumulated information is stored in *Long-Term Memory*.

6) HRI-Panel: This GUI based panel is for accepting user commands and displaying robot status to its user.

IV. Design of Service Policies

In this paper, we have proposed a scenario for human-centered service which contains delivery of objects (e.g., mail, fax, CD, confidential document etc.), by a service robot from our office secretary's desk to the different rooms of the staffs (a typical set up is shown in Fig. 2). This kind of scenario has been considered by other researchers focusing mainly on system design and development [3, 5]. However, our focus is to demonstrate how the robot can follow human guidelines by customizing its services. Such guidelines are expected to come from the higher level staff of an office.

So, in this scenario, we have decided that our robot must satisfy the following three conditions:

1. Confidentiality of the delivered objects must be preserved,
2. Urgent objects must be delivered by person seniority and
3. Other regular objects, which are neither confidential nor urgent, should be delivered according to the shortest distance from the service start point to destination point.

To meet these conditions, the robot should follow the working patterns practiced by our secretary. By a close observation, we have found that our secretary usually maintains the confidentiality of objects by visiting one person at a time. She also maintains the priorities of urgent objects according to the seniority of persons and if the object is neither confidential nor urgent, she sorts them out by location and visits the rooms of the staffs by choosing the nearest one first. So, the following categorized higher level policies have been derived.

1) Policies for confidential object delivery:

- In service planning phase (while receiving the user's request for object delivery) if there exist multiple instances of confidential objects for multiple persons, this request must not be processed by the robot.

- In service executing phase, when the robot is going to deliver the confidential object to a person, the person might be unavailable at that moment and the delivery can't be successful. In that case, other non-confidential (even urgent) objects of that mission will not be attempted to deliver at that moment. The robot must return to the service start point to report the failure and re-schedule the service.

2) Policies for urgent object delivery:

- Senior persons (by employment rank and employment start date) will get higher priority to receive the urgent objects.

- If an urgent object is failed to deliver for first time, it might be retried one more time after serving the remaining objects.

3) Policies for regular object delivery:

- The persons located nearer from the service start point should be served first. Also, persons in same location should be served at a time.

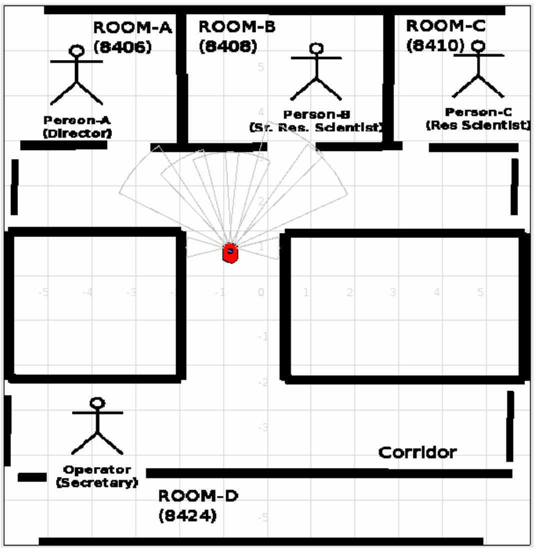

Fig. 2 A typical Stage setup for a robot under object delivery service

V. Service Implementation

In this section, we have described how we have materialized our architecture as well as how our knowledge-base (KB) for an object delivery service is designed, developed and tested.

A. Software Platform

After a brief survey on available open source and commercial mobile robot simulators, we have selected Player/Stage [13, 14] as our development test bed due to the remarkable features of Player/Stage, e.g., well organized client/server framework, diverse programming language support and a large user community. We have used Stage (version 2.0.3) 2-D simulator and Player (version 2.0.3) robot server along with the Java client for Player named Java-Player (version 2.0.1) from [15]. A typical Stage setup for our object delivery service is shown in Fig. 2.

Using Player/Stage as our *Controller* module, we have developed *ExecutiveSupervisor (ES)* using the Java-Player client library. This module has essential behavior functions, e.g., turn and change location, for moving one place to another with avoiding obstacles based on sonar readings (treated as a hybrid behavior consisting of *MoveToTarget* and *AvoidObstacle*). *ES* is interfaced with *KB Planner* modules, developed in XSB (an efficient deductive database from [16]), via a Java bridge program YAJXB obtained from [17]. So whenever a user is requesting an object delivery service to *ES*, it consults with *KB Planner* modules to obtain an appropriate plan conforming user policies and guidelines. This generated plan is passed to *Controller* and other modules for timely execution and status reporting.

B. Knowledge-base design and development

As shown in Fig. 3, we have organized the knowledge-base of our object delivery service hierarchically using the modular programming strategy in KB. Here, each file contains some facts or rules.

The following steps have been undertaken to complete the development of our KB.

1) Finding available facts for making inferences

At the beginning of our design process, we have analyzed the available unconditional facts accessible by the robot. When a user will make a request to the robot to deliver an object, it is natural that she will tell the robot like this: "Please deliver this *mail* to *Dr. Ali*, this is *confidential*" or perhaps in another case she might say: "Please deliver this *fax* to *Dr. Kim*, this is *urgent*". By combining these cases, we have selected five major properties of an object from a run-time request of a user: object category, receiver of the object, object confidentiality, object urgency and an automatic object ID assigned for tracking it.

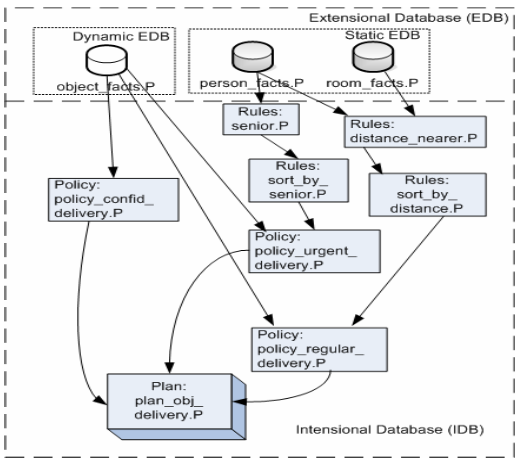
Extensional Database (EDB)
Dynamic EDB
Static EDB
object_facts.P
person_facts.P
room_facts.P
Rules: senior.P
Rules: distance_nearer.P
Rules: sort_by_senior.P
Rules: sort_by_distance.P
Policy: policy_confid_delivery.P
Policy: policy_urgent_delivery.P
Policy: policy_regular_delivery.P
Plan: plan_obj_delivery.P
Intensional Database (IDB)

Fig. 3 KB Inference tree for planning an object delivery service

If we cross check it with our confidential object delivery policy, we can see that the collected facts are enough to make a decision to satisfy this policy (i.e., the underlying logic of inference is based on the count the number of persons who are going to receive confidential objects). A sample object facts file has been shown here.

```
%% object_facts.P : Run-time list of objects to be
% delivered, (extracted from client side XML-RPC
% request, supplied by the user)
:- export object/5.
% Object(Id, Category, ReceiverStaffID,
% isConfidential, isUrgent)
object(1,'Mail',ali,1,0).
object(2,'Fax',kim,0,1).
object(3,'Report',park,0,0).
```

As we can see, each XSB program module exports the predicates with a certain number of arity. Here, *object* predicate has been exported with an arity of 5. Later, we will see that other program modules will use this predicate by an *import* command. We can also import any built-in predicate from the libraries shipped with XSB.

Now, we have to consider about additional database facts which should be given to the robot in advance. So, we can examine our urgent object delivery policy which says that robot has to deliver an urgent object based on the seniority of a person (by employment rank and employment start date). To satisfy this policy, we have decided to keep some attributes of every employee in the database of persons, such as a unique staff ID, full name of the person, employment rank or designation, employment start date, room number and status of attendance in office. As the robot has to move from one room to next room, we have created another database containing room facts, the available rooms for the object delivery service.

```
% persons_facts.P: persons DB (with 6 attributes:
% StaffID,Name,Designation, EmploymetStartDate,
% OfficeLocation, AttendenceStatus)
:- export person/6.
person(ali,'A. Ali','Director',
200404,8406,'Present').
person(kim, 'B. Kim', 'Sr. Res. Scientist',
200407, 8408, 'Present').
person(lee, 'C. Lee', 'Sr. Res. Scientist',
200403, 8408, 'Present').
person(park, 'X. Park', 'Res. Scientist',
199103, 8410, 'Present').
person(smith, 'Y. Smith', 'Res. Scientist',
199903, 8410, 'Present').
```

Again, we have cross checked these files to ensure that this much information is enough for making inferences to satisfy our policies for urgent and regular object delivery cases(i.e., to find the most senior person or the nearest room from the service start point of the robot etc).

2) Constructing the inference tree by writing deductive rules

As it is seen in Fig. 1, *Strategic Planner (SP)* will evaluate our KB modules and it will generate a strategic plan for *ES* using an inference tree. So, we have incrementally written our logic program modules by deductive rules with a final goal for building the inference tree as shown in Fig. 3. The input of this tree will be the person facts and room facts extracted from our relational database and the run time object facts. The output of this tree should be the decision whether user's request is valid or not and if valid what is the correct order of delivering objects following the user specified policies. Here we have presented some examples of our KB modules following the syntax of XSB's native language, Hi-Log [16].

```
%% policy_confid_delivery.P
:- import length/2 from basics.
:- import object/5 from object_facts.
:- export confid_rcvr/1, confid_rcvr_set/1,
confid_delivery/1.
% Find the disinct list of persons who will
receive %confidential documents
confid_rcvr(Rcvr) :-
object(_,_,Rcvr,IsConfid,_), IsConfid =:= 1.
confid_rcvr_set(List) :-
setof(Rcvr, Rcvr ^ confid_rcvr(Rcvr), List).
confid_rcvr_set([]).
% policy.confidentiality: only one person is
%allowed for confidential delivery
confid_delivery(Answer) :-
confid_rcvr_set(List), length(List, Len), Len
=< 1, Answer = 'Valid'.
confid_delivery(Answer) :-
confid_rcvr_set(List), length(List, Len), Len >
1, Answer = 'Invalid'.
```

As we can see in Fig. 3, our inference tree for deriving a plan for the object delivery is rooted in three facts files. From object facts, we have derived our decision to conform the confidentiality policy in file policy_confid_delivery.P. So if we input two files: object_facts.P and po icy_confid_delivery.P to XSB engine and post a query `?confid_delivery(Ans).`
For valid request, we can see that it says:
`Ans = Valid.`

In case of implementing our policy for delivering urgent objects, we have derived the sorted list of receivers by seniority rule (using employment rank and employment start date) in module senior.P and then we have used a quick sort algorithm to find a list of persons arranged in the order of decreasing seniority. Some default clauses are attached to arbitrate some exceptions, e.g., we set the first come first serve rule when two persons have exactly same employment ranks and same employment start dates. Then, we have constructed our policy_urgent_delivery.P module.

```
%% policy_urgent_delivery.P:produces a list of
%%% receivers of urgent objects by seniority.
:- import object/5 from object_facts.
:- import sort_by_senior/2 from sort_by_senior.
:- export urgent_delivery_list/1.
% Make a list of rcvr. who will rcv. urgent obj.
urgent_rcvr(Rcvr) :-
object(_,_,Rcvr,_,IsUrgent), IsUrgent =:= 1.
urgent_rcvr_set(List):-
setof(Rcvr, Rcvr^urgent_rcvr(Rcvr), List).
% Sort by seniority
urgent_delivery_list(SortedList):-
urgent_rcvr_set(RawList),
sort_by_senior(RawList, SortedList).
urgent_delivery_list([]).
```

Similarly, in our regular object delivery case, we have made a sorted list of receivers based on the nearer distance using three XSB modules: distance_nearer.P and sort_by_ditance.P and policy_regular_delivery.P module (same as policy_urgent_delivery.P). Finally, we have assembled our all policy modules to generate a plan in module plan_obj_deliver.P. The underlying idea behind this combining process is: the confidential delivery policy is checked first and if it succeeds then this module will make a list and add the receiver who will receive the confidential documents; otherwise it will make an empty list and incrementally add urgent and regular lists to it. Some built-in and custom predicates are used to produce an appropriate list of persons and locations.

C. Test and Discussion on KB Implementation

The results obtained by running our KB modules in XSB with several data sets give us satisfactory performance of our KB rules. Here, we have described a few cases with sample data sets and results. For brevity *export* commands and some optional lines are removed from these examples.

a) Case-1a: In this case, the user has requested our robot to deliver multiple confidential objects to multiple receivers.

```
%Object(Id,Category,Receiver,isConfid,isUrgent)
object(1,'Mail',kim,1,0).
object(2,'Fax',ali,1,0).
object(3,'Report',lee,1,1).
object(4,'Notes',kim,0,0).
```

The result is saved in a file and is shown here:

```
Error in your delivery request !
Too many confidential receivers: [kim,lee,ali]
Please select only one receiver for your request.
```

b) Case-1b: The input fact file and output result for multiple confidential objects to one receiver are shown here.

```
%Object(Id,Category,Receiver,isConfid,isUrgent)
object(1,'Mail',kim,1,0).
object(2,'Notes',kim,1,0).
```

```
I am going to process your request with this plan-
Persons: [kim]
Locations: [8408]
```

c) Case-2: Now, the user has requested multiple urgent objects to multiple persons.

```
%Object(Id,Category,Receiver,isConfid,isUrgent)
object(1,'Mail',kim,0,1).
object(2,'Notes',ali,0,1).
object(3,'Mail',lee,0,1).
object(4,'Notes',smith,0,1).
object(5,'CD',park,0,1).
```

So, KB engine gives this answer according to seniority rule.

```
I am going to process your request with this plan-
Persons: [ali,lee,kim,park,smith]
Locations: [8406,8408,8410]
```

d) Case-3:
Here, all kinds of objects are mixed in the user's request.

```
%Object(Id,Category,Receiver,isConfid,isUrgent)
object(1,'Notes',smith,0,0).
object(2,'Notes',ali,0,1).
object(3,'Mail',kim,0,1).
object(4,'Fax',lee,1,0).
object(5,'Fax',park,0,0).
object(6,'Mail',lee,1,0).
```

The corresponding answer is shown below.

```
I am going to process your request with this plan-
Persons: [lee,ali,kim,park,smith]
Locations: [8408,8406,8410]
```

The above plans have been carried out by our Player/Stage robot in an environment shown in Fig. 2. Here, we have observed that our service robot can properly utilize its knowledge of certain user relationships and policies to derive a proper plan for its object delivery service. In case of the traditional service planning, a robot can not achieve this kind of plan by using simple and unrelated location and/or person information. However, using relationship information from a knowledge-base it can easily deduce an executive service plan that satisfies user policies. If the robot is working in a human-living environment, this kind of behavior of robot is highly expected by the end-users.

In this work, we have followed the declarative programming paradigm where it is said that the programmer does not need to explicitly specify the computer how to solve the problem except describing the problems declaratively. This is partially true. The correct order of execution of rules has also to be specified. In our opinion, declarative way of solving problems (related to developing knowledge-based systems) is more flexible, modular and efficient than the imperative one.

VI. KB Performance And Scalability Analysis

In the deductive database literature, such as in [7] it is common to distinguish a set of facts as the extensional database (EDB) and to refer to the set of rules as the intensional database (IDB). As shown in Fig. 3, our KB planner contains EDB (e.g., object facts, person facts, room facts etc.) and IDB (remaining part of KB). EDB has static part like person facts and dynamic part like object facts. Usually the sizes of both static and dynamic EDB have an impact on planning time. In order to verify the feasibility of our proposed approach in a real robot environment, we have performed a series of simulations for analyzing the impact of varying the dynamic EDB (object facts) size, for a given static EDB (person facts) size on planning time spent in KB planner modules. This gives us a general idea how the planning time increased if the number of persons of an office environment is changed and the number of delivery objects is also increased. The results have been plotted in Fig. 4.

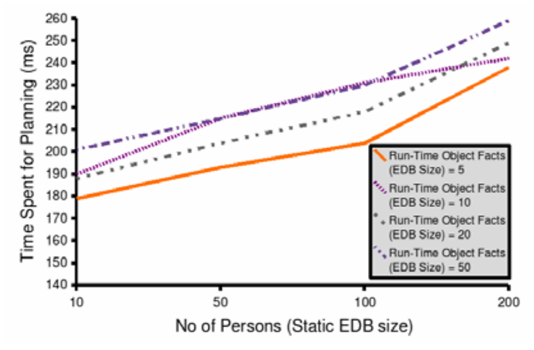

Fig. 4 Effect of varying EDB size on KB planning time

From this graph, we can see that if the number of persons (static EDB size) is increased planning time is also increased proportionally, but increasing the number of delivery objects (dynamic EDB size) has less impact on overall planning time. And for a typical case of an office environment with 100 persons and 20 delivery objects our planner took less than 250 milliseconds to generate the schedule of delivery.

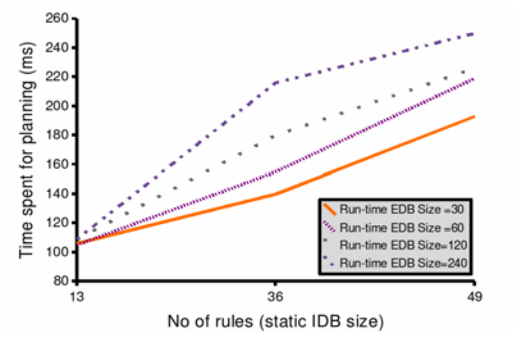

Fig. 5 Effect of varying IDB size on KB planning time

Similarly, we have plotted the linear relationship between the increasing number of IDB rules and time spent in planning in planner modules in Fig. 5. For 4 sets of predetermined set of EDB we have measured the time for planning with three cases: 1) using only confidential object delivery policy (13 rules), 2) using both confidential and urgent object delivery policies (36 rules) and 3) using all three policies (49 rules). The EDB size is dynamically adjusted at run-time, e.g., in our first trial we have used EDB size = 30, (10 object facts, 17 person facts and 3 room facts). In each of three later trials, we have doubled EDB size, i.e., we have used EDB size 60 (7 object facts, 50 person facts and 3 room facts), 120 (17 object facts, 100 person facts and 3 room facts) and 240 (37 object facts, 200 person facts and 3 room facts). From these trials, we have found that for a given EDB size, planning time increases almost linearly with the increasing number of rules.

These simulations were performed dedicatedly in a Pentium IV machine with 256MB memory. From these observations, it is seen that planning time is feasible to implement it in a real robot in real-time.

VII. Conclusion

In this paper, we have proposed a knowledge-based service approach for a human-centered robot that can appropriately adopt high-level human guidelines or policies into its service planning and execution. This deductive scheme of learning enables a robot to be portrayed as an obedient assistant in a user's mind. Using our Knowledge-Based Human-centered (KBH) architecture, we have designed and developed a knowledge-base for implementing an object delivery service. The results obtained by testing the knowledge-base in XSB deductive database engine along with Player/Stage simulator have been reported.

Acknowledgement

This research has been conducted as a part of the Network-based Humanoid Project funded by the Ministry of Information and Communication (MIC) of the Republic of Korea.

References

[1] B. Graf, M. Hans, R.D. Schraft, “Care-O-bot II—Development of a next generation robotic home assistant”, *Autonomous Robots,* vol. 16, no. 2, March, 2004, pages 193-205.

[2] P. Joelle, et al., “Towards robotic assistants in nursing homes: Challenges and results”, *Robotics and Autonomous Systems*, vol. 42, no. 3-4, March, 2003, pages 271-281.

[3] K. Severinson, et al., “Social and collaborative aspects of interaction with a service robot”, *Robotics and Autonomous Systems*, vol. 42, no. 3-4, March, 2003, pages 223-234.

[4] S. Thrun, “Toward a framework for human-robot interaction”, *Human-Computer Interaction,* vol. 19, no. 1-2, 2004, pages 9-24.

[5] T. Taipalus and K. Kosuge , “Development of service robot for fetching objects in home environment”, in *Proceedings of the IEEE Int’l Symposium on Computational Intelligence in Robotics and Automation, CIRA,* 2005, page(s): 451- 456.

[6] K. Sakai, et al., “Developing a service robot with communication abilities”, in *Proceedings of the IEEE Int’l Workshop on Robot and Human Interactive Communication, ROMAN,* 2005, pages 91-96.

[7] E. Bertino, et al., “Deductive Object Databases” In *Proceedings of the 8th European Conference on Object-Oriented Programming* (July 04 - 08, 1994). M. Tokoro and R. Pareschi, Eds. Lecture Notes In Computer Science, vol. 821. Springer-Verlag, London, 213-235.

[8] M.Y. Shieh, et al., “Design of an intelligent hospital service robot and its applications”, in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics,* 2004, pages 4377- 4382.

[9] A. Clodic, et al., “Supervision and interaction: Analysis of an autonomous tour-guide robot deployment”, in *Proceedings of the 12th Int’al Conference on Advanced Robotics, ICAR,* 2005, pages 725-732.

[10] J. Han, et al., “The educational use of home robots for children”, in *Proceedings of the IEEE International Workshop on Robot and Human Interactive Communication, ROMAN,* 2005, pages 378- 383.

[11] M. Ehrenmann, et al., “Programming service tasks in household environments by human demonstration”, in *Proceedings of the IEEE International Workshop on Robot and Human Interactive Communication, ROMAN,* 2002, pages 460- 467.

[12] Jodi Forlizzi and Carl DiSalvo, “Service robots in the domestic environment: a study of the roomba vacuum in the home”, in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction,* 2006, pages258 - 265.

[13] Brian Gerkey, Richard T. Vaughan and Andrew Howard. "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems" Proceedings of the 11th International Conference on Advanced Robotics, pages 317-323, Coimbra, Portugal, June 2003.

[14] http://playerstage.sourceforge.net

[15] http://java-player.sourceforge.net/

[16] http://xsb.sourceforge.net

[17] http://infolab.stanford.edu/~stefan/rdf/yajxb/