Interruptible Autonomy: Towards Dialog-Based Robot Task Management

Yichao Sun, Brian Coltin and Manuela Veloso
School of Computer Science
Carnegie Mellon Univeristy
5000 Forbes Avenue, Pittsburgh, PA, 15213

Abstract

We have been successfully deploying mobile service robots in an office building to execute user-requested tasks, such as delivering messages, transporting items, escorting people, and enabling telepresence. Users submit task requests to a dedicated website which forms a schedule of tasks for the robots to execute. The robots autonmously navigate in the building to complete their tasks. However, upon observing the many successful task executions, we realized that the robots are *too* autonomous in their determination to execute a planned task, with no mechanism to *interrupt* or redirect the robot through local interaction. In this work, we analyze the challenges of this goal of interruption, and contribute an approach to interrupt the robot anywhere during its execution through spoken dialog. Tasks can then be modified or new tasks can be added through speech, allowing users to manage the robot's schedule. We discuss the response of the robot to human interruptions. We also introduce a finite state machine based on spoken dialog to handle the conversations that might occur during task execution. The goal is for the robot to fulfill humans' requests as much as possible while minimizing the impact to the ongoing and pending tasks. We present examples of our task interruption scheme executing on robots to demonstrate its effectiveness.

Introduction

We have spent the last few years enabling our mobile robots, CoBots, to autonomously localize, navigate, and perform tasks in the environment. We consider these efforts a success in light of the fact that we currently have the CoBot robots moving by themselves in the environment, and they have already navigated more than 200km (Veloso et al. 2012). It was a long path to get to the point where everything is working, with challenges including hardware, perception, sensing, localization, path planning, navigation, task requests, task-based interaction with humans, and symbiotic autonomy enabling the robots to proactively ask for help from humans and the web when needed.

As we reached this achievement, we realized that the robots were, in a way, *too* autonomous. They received and scheduled task requests on a centralized server (Coltin and Veloso 2011), they generated plans to accomplish their tasks, and they executed these tasks as their sole goal. A person in the robot's path could not *interrupt* its execution by asking, "CoBot, can you please take me to the elevator?" CoBot executed its plans and did not allow humans to interrupt it. Also, if CoBot were coming to deliver a package to someone's office and that person met CoBot in the middle of the corridor, she could not stop the robot and say, "CoBot, thanks for the package, I got it!" CoBot would go all the way to the office to finish its delivery task. Tasks could of course be interrupted and managed remotely through an administrative interface, but not by *naturally* interacting with the robot.

With recent advances in speech recognition techniques, a variety of dialog management approaches have been proposed for service robots to interact and receive commands from human users via speech (Zobel et al. 2001; Jayawardena et al. 2010). Many approaches enable plans and control actions to be acquired through dialog context (Schiffer, Ferrein, and Lakemeyer 2012; Rybski et al. 2008; Van Der Zant and Wisspeintner 2007). Likewise, on the CoBots, trusted users were previously able to command CoBot through speech, but these requests could only be input when CoBot was stationary with nothing to do, and were only executed immediately, not added to the robot's schedule of tasks (Kollar, Perera, and Veloso 2013). In this paper, we introduce a dialog-based approach to interrupt the robot, what we call an *interruptible autonomy* approach for the robot to respond and attend to dialog managing its task list. Our goal is to make these interruptions intuitive and natural for untrained passerby, and to interact in a manner that effectively responds to the needs of both passerby and users that already have scheduled tasks planned for execution.

In the remainder of this paper, we first discuss how tasks are described and executed on the CoBots and how interruption is detected and handled. We propose a a interrupt handler that take charge of the spoken dialog triggered interruption and task cancellation. We also discuss the task rescheduling problem if user assigns a new task to CoBot during interruption. Finally we give demonstrative examples of task interruption on the CoBots.

USER:	Go to the meeting room.
COBOT:	I am going to "the meeting room", Room 7405, is that correct?
USER:	Yes.
COBOT:	Going to "the meeting room", Room 7405.

Table 1: An example of task request through dialog interaction. The user was interacting with CoBot using speech command and assigned a *GoToLocation* task

Task Execution and Interruption

The CoBot robots we developed can execute tasks requested on the web interface and through speech. Users can book a new task for CoBots with detailed specifications of the type of the task, time window, location, and other task-dependent parameters. A more natural way for untrained users to assign a task to a robot is to take advantage of the spoken dialog system. Table 1 shows a simple example of a conversation assigning a task to a CoBot through speech. After a booking request is processed, the task is saved to a database in a generalized format, shown in Table 2. Some of the fields of the task descriptor are optional. For instance, for a *GoToLocation* type task, CoBots only require information about the "Destination", "Start Time", "End Time", "Owner Name" and leave the rest of the fields blank. Due to the limitations of speech dialog interaction, our current dialog manager cannot extract the user's name in an effective and robust way. This part of work is still in progress.

Task Execution on the CoBots

The tasks that the CoBots are capable of carrying out include:

- *GoToLocation*: CoBot navigates to a place in the building.
- *DeliverObject*: CoBot goes to retrieve an object from a specific location and deliver it to the destination. This task is an abstraction of both delivering items and escorting people from one place to another.
- *DeliverMessage*: CoBot goes to a place and displays a message on behalf of the task owner.
- *Telepresence*: CoBot goes to a room and starts the telepresence interface to allow a user to attend a meeting or event (Coltin et al. 2012).
- *MultiObjectDeliver*: CoBot goes to retrieve a set of items from different locations or deliver a set of items to different locations. (Coltin and Veloso 2013)

Tasks are sent to the CoBots from the online scheduling agent that receives users' booking requests. The CoBots execute tasks by breaking them into different segments of actions such as *navigate*, *ask*, *speak*, etc. Each action is an independent module so that the robot can plans for a sequence of actions necessary to complete a task. For example, a *GoToLocation* task consists of a *navigate* action to some location, followed by an *ask* action for task completion confirmation.

Field	**Description**
id	The unique id number of the task.
task_type	Type of the task, e.g., *GoToLocation*, *DeliverObject*, etc.
start_location	Starting location of the task.
destination	Ending location of the task.
start_time	Expected starting time of the task.
end_time	Expected ending time of the task.
object_name	Name of the object to be retrieved/ delivered.
owner_name	Name of the user who booked the task.
owner_email	Email Address of the user who booked the task.

Table 2: The data structure of a task descriptor.

Task Interruption on the CoBots

For a robot to work in a human-inhabited environment, we implemented an obstacle detection module that can search for obstacles within a certain range in front of the CoBot. When a human steps in front of a CoBot, the CoBot would treat him/her as a dynamic obstacle blocking the path, and stop navigation until the path is clear. Once a CoBot is stopped, the passerby can press the "Speak" button on the CoBot's touch screen to begin speech interactions with it.

The dialog between the CoBots and humans can lead to a new task being scheduled to fulfill the users' request. There are several possible actions for users to choose from by giving speech commands:

- **INQUIRY**: Users can ask a CoBot about its current status. The CoBot responds by describing its current task type and the target location it is heading to aloud.
- **NEW_TASK_NOW**: A user can assign a new task for a CoBot to execute immediately. If the CoBot has no currently running task , it should start the new task right away. If the CoBot is running some other tasks now, it should try to move the incoming task to the front of its schedule and start executing it.
- **NEW_TASK_LATER**: User can also choose to schedule the new task after the CoBot's current task finishes, if the currently running task cannot be rescheduled.
- **CANCEL_TASK**: Users can also choose to cancel CoBot's current task. Only the owner of the task or the administrators are authorized to do this. Once a task is cancelled, the owner of the task will receive an email notice of the cancellation.

Dealing with Interruption

Figure 1 shows our interruption handling process. It consists of three parts. First is the dialog parser, a dialog management module that interacts with the human user through speech. The dialog parser perceives a user's speech command and converts the speech into text candidates. Using a knowledge base based probabilistic model (Kollar, Perera, and Veloso 2013), we are able to ground the user's speech command into actions the CoBots are capable of. The dialog parser

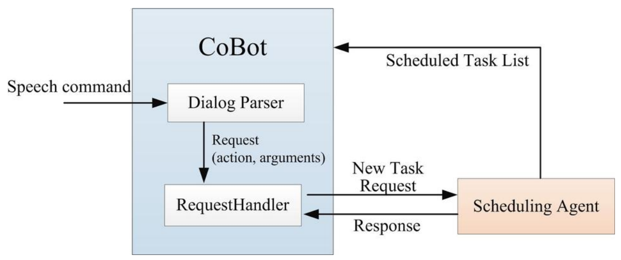


Figure 1: The handling process for spoken dialog-based interruption. The dialog parser and interrupt handler run on CoBot itself and communicate with the scheduling agent, which is on a remote server, through an internet connection.

first extracts linguistic constituents of the speech text, then parses the action and arguments for a task request. If the dialog parser does not understand the user's speech command, it further interacts with the user by prompting possible candidate options the user can choose from. It also updates its knowledge about the mapping relation between the speech text and the actual actions, locations and objects through the results of dialog so that they can be used for future interactions.

Algorithm 1 RequestHandler()

```
loop
  $T_0 \leftarrow GetTaskStatus()$
  $(action, args) \leftarrow DialogParser()$
  if $action$ = NEW_TASK_NOW then
    $T_{new} \leftarrow args.task$
    $T_{new}.start\_time \leftarrow now$
    $TrySchedule(T_{new})$
  else if $action$ = NEW_TASK_LATER then
    $T_{new} \leftarrow args.task$
    $T_{new}.start\_time \leftarrow T_0.end\_time$
    $TrySchedule(T_{new})$
  else if $action$ = CANCEL_TASK then
    if $args.user\_name = T_0.owner\_name$ or $args.user\_name$ is administrator then
      $scheduler.AbortTask(T_0)$
      $Speak$("Task has been cancelled. Anything else I can do for you?")
      $SendCancelNotice(T_0.owner\_email)$
    else
      $Speak$("Sorry, I cannot cancel the task for you.")
    end if
  else if $action$ = INQUIRY then
    $speak$("I am on my way"+ $TaskToSpeech(T_0)$)
  end if
end loop
```

Once an action is extracted from the spoken dialog, the request handler chooses a different behavior to fulfill the user's request. The detailed procedure is shown in Algorithm 1. It first saves the current running task status(T_0), the extracted action($action$) and arguments list($args$) returned by the dialog parser. If $action$ is NEW_TASK_NOW, we put the arguments extracted from spoken dialog into a task descriptor, mark its start_time as now, and then send it to the scheduling agent by calling the $TrySchedule()$ function. If the NEW_TASK_LATER action is executed, we get the estimated end time of the currently running task from T_0 and set it as the start time of the new task and call $TrySchedule()$ in the same way as above. For a CANCEL_TASK action, the handler checks the identity of the current interrupting user. If the user is one of the administrators, or the user is the owner of the running task, the request handler calls the scheduler to cancel the task described by T_0. This makes sense since the owner of the task may discover that he/she may no longer need CoBot's help before the CoBot completes its task. And the administrators should always have the privilege to cancel any tasks CoBots are running. Once a task has been cancelled, a notification is sent to inform the owner of the task through email. If the extracted action is INQUIRY, the CoBot will speak out the content and destination of its currently running task.

The scheduling agent is an independent module running on a remote server, which means the robot can either be executing a task or waiting for new tasks when it is interrupted. The request handler calls the $TrySchedule()$ function to communicate with the scheduling agent, which is described in Algorithm 2. The new task assigned by the user is put in a descriptor named T_{new}, which is then sent to the scheduling agent. The scheduling agent will return a status indicating the booking process is successful or not.

Algorithm 2 TrySchedule(T_{new})

```
$T_0 = \leftarrow GetTaskStatus()$
$response = scheduler.SchedulerTask(T_{new})$
if $response$ =SUCCESS then
  if $T_{new}.start\_time = now$ then
    $Speak(TaskToSpeech(T_{new}))$
  else
    $Speak(TaskToSpeech(T_{new})+$"after I finish my current task.")
  end if
else
  $Speak$("I am sorry I cannot schedule this task.")
end if
```

Task Rescheduling

Once a task is confirmed during a dialog, we have to schedule it into the CoBot's task list and update the currently running task if necessary. If the CoBot is currently occupied with a task, we have to reschedule all of the tasks the CoBot has and see what would be the robot's best choice of action next.

We have developed a centralized scheduling agent to deal with all task booking requests so that it can generate a feasible plan for the CoBots to fulfill all the tasks in a time-efficient manner (Coltin and Veloso 2011). The scheduling agent looks into all the tasks' constraints and solves this scheduling problem using mixed integer programming. The scheduling agent is called every time a new task is given to the CoBots, and returns failure if a sequence of actions that fulfills all the constraints cannot be found. The updated task schedule will be sent to the CoBots right after it is generated.

If the rescheduling happens when a CoBot is executing a task, it is possible that the CoBot will be switched to execute a new task first, and then get back to its previous task after the new one is fulfilled. The sequence of the task to be executed depends on the status of the robot, its destinations, the estimated travel times, and the time window constraints of the tasks in the task list. The interruption process and the scheduler should guarantee that the CoBots will meet the constraints of the tasks already assigned to the CoBots even when the CoBot is interrupted by a new task request.

There are several factors that are vital for the scheduling agent to form a schedule:

1. *Time Constraints*
   The resultant plan should meet all the time constraints of the tasks. Each task has a time window indicating preferred start and end times. The scheduling agent ensures that the execution times of different tasks do not overlap and all tasks are completed as soon as possible by minimizing the total time difference from the start of the window for all tasks.

2. *Preconditions and Postconditions*
   For *DeliverObject* tasks, the robot is expected to fetch an object from a certain location and deliver it to another. If the robot is interrupted during a *DeliverObject* task, it is possible that the CoBot still has the object on it, which indicates the task should not be interrupted by new tasks until the object is delivered. The scheduling agent checks the preconditions and postconditions of all tasks and the state of the CoBot and makes sure the *DeliverObject* task will be fulfilled.

3. *Failure Notification*
   When the CoBots are not able to schedule the new task because it is fully occupied during that period of time, the scheduler returns a failure code to the CoBot so that it can notify the user.

Taking advantage of the web interface, users can book new tasks for robots and specify an expected time window. The scheduling agent generates a proper sequential task list that fulfills all the tasks constraints. For tasks assigned through the dialog interface, we have defined two action types as mentioned, NEW_TASK_NOW and NEW_TASK_LATER. There are three possible outcomes when the request handler sends a new task request to the scheduling agent. First, if the CoBot has spare time slots for incoming tasks, then the scheduler will put the task on the CoBots for execution right away. Second, if the CoBot is currently busy executing a task and the scheduler successfully inserts the new task later into its schedule. The CoBot might run either the new task or an old one first depending on the task specification and constraints. Finally, the third case is that the scheduler may not find a feasible schedule.

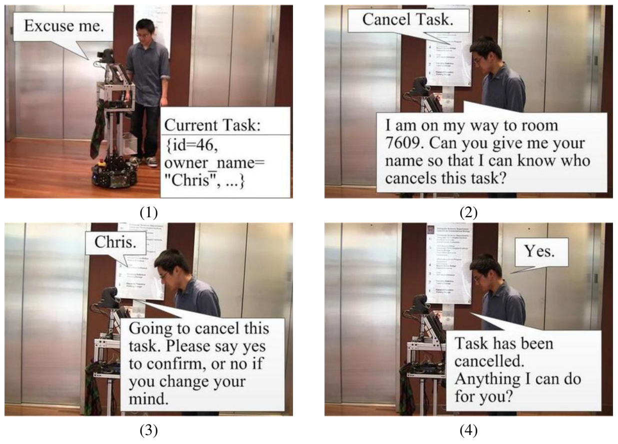
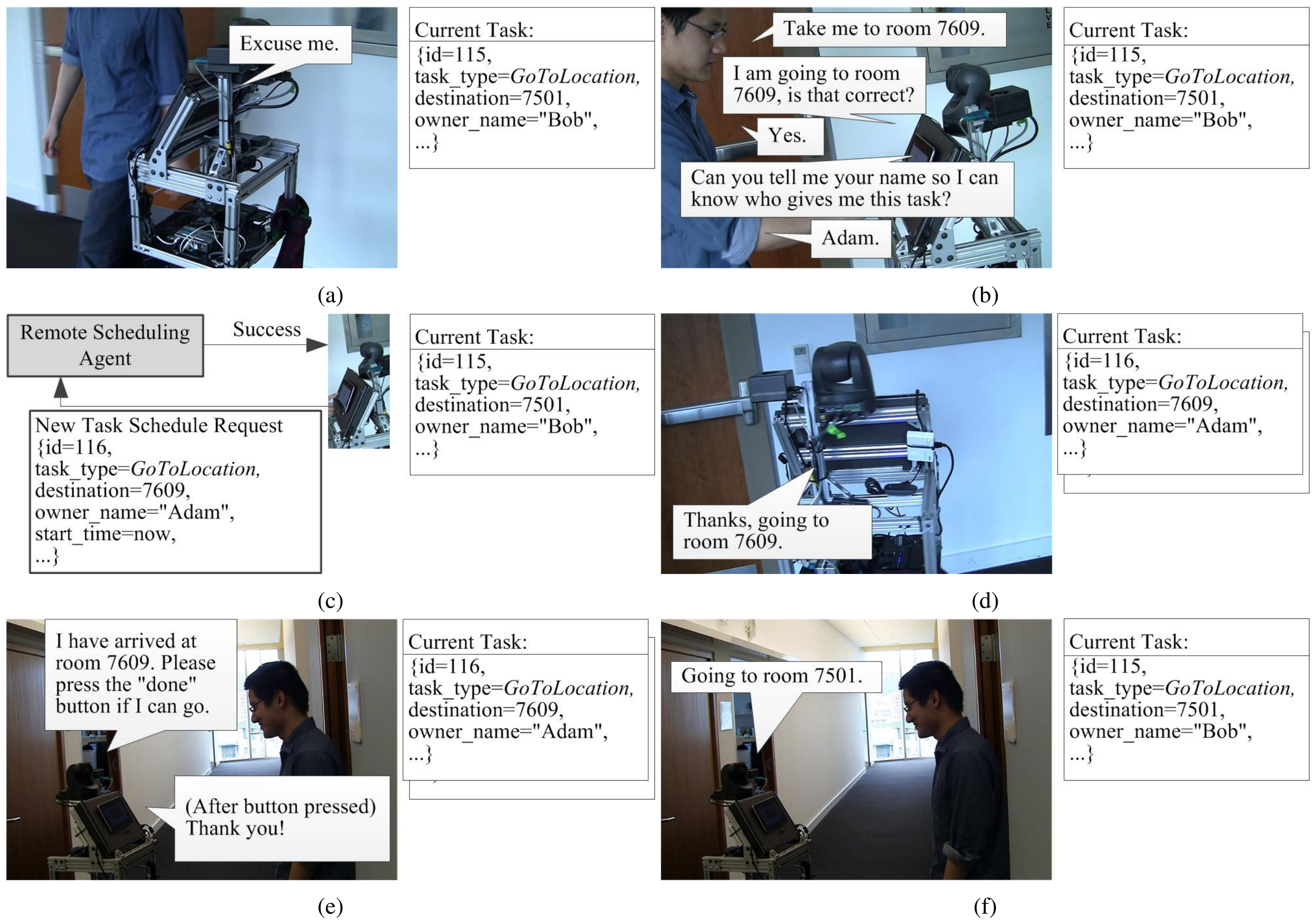

Excuse me.
Current Task: {id=46, owner_name= "Chris", ...}
(1)
Cancel Task.
I am on my way to room 7609. Can you give me your name so that I can know who cancels this task?
(2)
Chris.
Going to cancel this task. Please say yes to confirm, or no if you change your mind.
(3)
Yes.
Task has been cancelled. Anything I can do for you?
(4)

Figure 2: A CoBot robot was interrupted by a passerby when it was executing a *GoToLocation* task. User wanted to cancel the current running task on the CoBot. After confirming the user was the owner of the current task, CoBot successfully cancelled its running task.

Experiments

To illustrate our task interruption functionality, we present several running examples of interruption by a passerby on CoBots.

- **Task Interruption - Cancellation**
  Figure 2 demonstrates task cancellation through the speech. A task can only be cancelled by its owner or the administrator. In this example when a user wanted to cancel CoBot's current task, CoBot first asked about the user's name and checked if he was the owner of the current task. After confirmation the task was successfully cancelled.
- **Task Interruption - Successful Task Booking**
  Figure 3 demonstrates an example of dialog-based task booking during interruption. A CoBot was executing a *GoToLocation* task when a passerby user interrupted the robot through the speech interface. The user "Adam" requested a new *GoToLocation* task with the CoBot, which was then sent to the remote scheduling agent. The start time of the task was now, indicating the task should be executed right away. The scheduler responded with a success message, and the CoBot started executing the new task after it received an updated task schedule. After the CoBot arrived at the destination with the user's confirmation, it executed the previous task from before the interruption.
- **Task Interruption - New Task Postponed**
  Things go differently if the CoBot's new task cannot be scheduled successfully. In the example illustrated in Figure 4 the CoBot was delivering an object to a specific location. The CoBot would only be able to serve the other user after this task is finished. The user first inquired about the CoBot's current task and then tried to interrupt the current task by assigning a new one. The CoBot notified the user that the new task could not be scheduled. After being no-


(a)
Excuse me.
(b)
Take me to room 7609.
I am going to room 7609, is that correct?
Yes.
Can you tell me your name so I can know who gives me this task?
Adam.
(c)
Remote Scheduling Agent
Success
(d)
Thanks, going to room 7609.
(e)
I have arrived at room 7609. Please press the "done" button if I can go.
(After button pressed) Thank you!
(f)
Going to room 7501.

Current Task:
{id=115,
task_type=*GoToLocation,*
destination=7501,
owner_name="Bob",
...}

New Task Schedule Request
{id=116,
task_type=*GoToLocation,*
destination=7609,
owner_name="Adam",
start_time=now,
...}

Current Task:
{id=116,
task_type=*GoToLocation,*
destination=7609,
owner_name="Adam",
...}

Current Task:
{id=115,
task_type=*GoToLocation,*
destination=7501,
owner_name="Bob",
...}

Figure 3: A CoBot robot was interrupted by a passerby when it was executing a *GoToLocation* task. A new task was assigned to the CoBot and was successfully scheduled to be executed right away. CoBot then started executing the new task and resumed the previous one when the new task was finished.

tified, the user asked CoBot to execute the new task after its current task, and the CoBot started running its original task with the new task scheduled as the next one.

Conclusion

In this paper we discussed the task managerment problem for a mobile service robot deployed in a human-inhabited environment through natural interface. We presented a spoken dialog-based approach for robot to interact with passerby and to interfere with the states and knwoledge of the robot. By introducing this scheme, the CoBot we developed can interact with human user any time around the building even if it's is executing a pre-scheduled task or plan. Users can interrupt the CoBots to inquire about the state of the robot, cancel running tasks, and book new tasks. We demonstrated that the task interruption process is intuitive, and user-friendly. Our plans for future work includes implementing vision or voice based person recognition so that the CoBots can recognize and remember their users instead of asking for their names every time when they interact with the CoBots. Different priority levels will also be implemented so that higher priority tasks are guaranteed to have privileges over those with lower priority. We will also work on extending the CoBots functionality during interruption so that it can reason according to its history state, current state, web information and sensing data of the physical world and achieve higher-level autonomous.

Acknowledgement

We thank Joydeep Biswas, Tom Kollar, Vittorio Perera, and Robin Soetens for their contributions to CoBot's autonomy and speech understanding, upon which our work builds.

References

Coltin, B., and Veloso, M. 2011. Dynamic user task scheduling for mobile robots. In *Workshop on Automated Action Planning for autonomous Mobile Robots at the Twenty-Fifth Conference on Artificial Interlligence(AAAI 2011).*

Coltin, B., and Veloso, M. 2013. Towards replanning for mobile service robots with shared information. In *Autonomous Robots and Multirobot Systems (ARMS).*

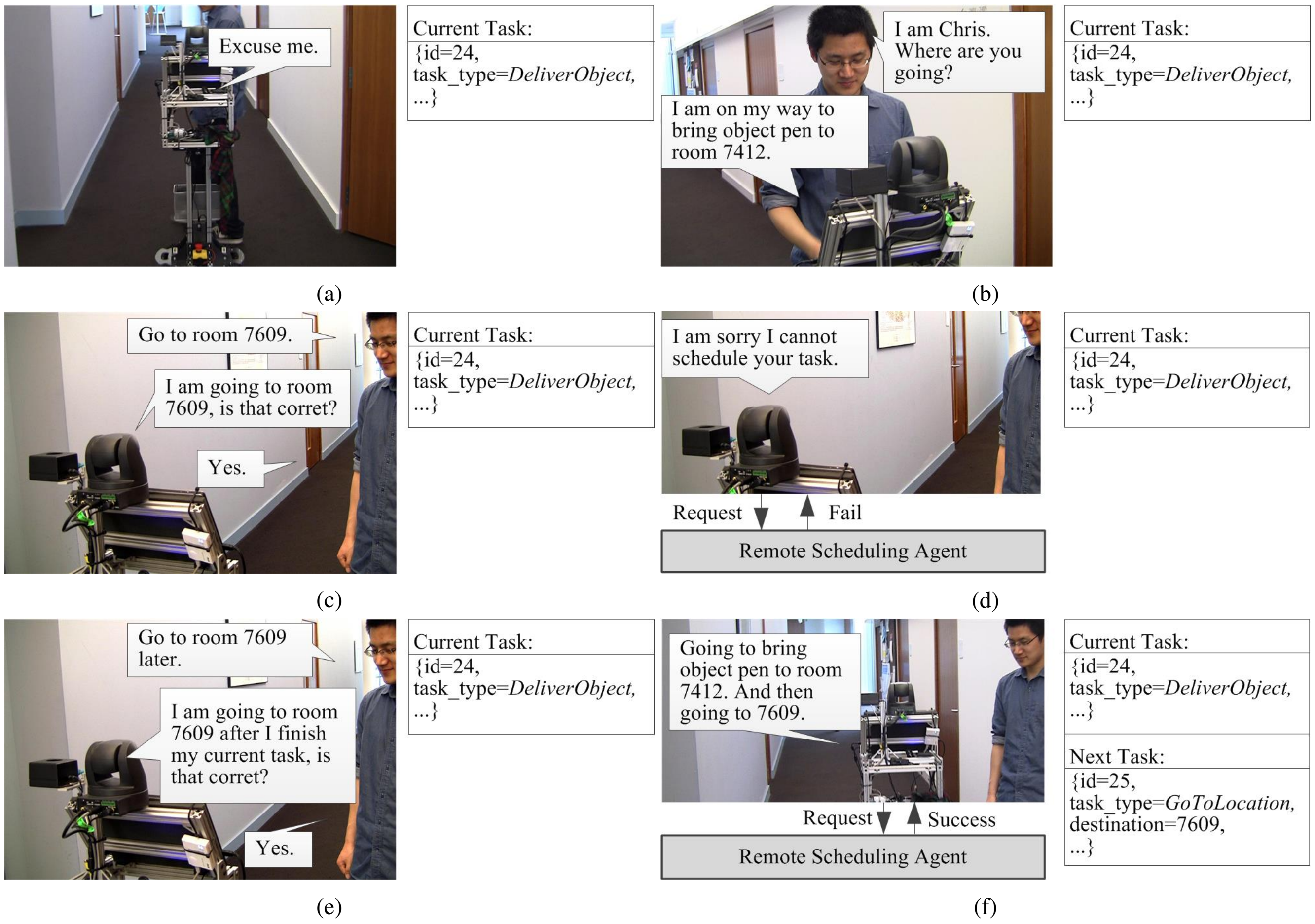

Excuse me.
Current Task: {id=24, task_type=*DeliverObject*, ...}
(a)
I am Chris. Where are you going?
I am on my way to bring object pen to room 7412.
Current Task: {id=24, task_type=*DeliverObject*, ...}
(b)
Go to room 7609.
I am going to room 7609, is that corret?
Yes.
Current Task: {id=24, task_type=*DeliverObject*, ...}
(c)
I am sorry I cannot schedule your task.
Request
Fail
Remote Scheduling Agent
Current Task: {id=24, task_type=*DeliverObject*, ...}
(d)
Go to room 7609 later.
I am going to room 7609 after I finish my current task, is that corret?
Yes.
Current Task: {id=24, task_type=*DeliverObject*, ...}
(e)
Going to bring object pen to room 7412. And then going to 7609.
Request
Success
Remote Scheduling Agent
Current Task: {id=24, task_type=*DeliverObject*, ...}
Next Task: {id=25, task_type=*GoToLocation*, destination=7609, ...}
(f)

Figure 4: A CoBot robot was interrupted by a passerby when it was executing a *DeliverObject* task. The user wished the CoBot can run a new task right away, but the CoBot cannot do that. Then user asked the CoBot execute the new task once the previous one was finished.

Coltin, B.; Biswas, J.; Pomerleau, D.; and Veloso, M. 2012. Effective semi-autonomous telepresence. In *RoboCup 2011: Robot Soccer World Cup XV*, volume 7416 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 365–376.

Jayawardena, C.; Kuo, I. H.; Unger, U.; Igic, A.; Wong, R.; Watson, C.; Stafford, R.; Broadbent, E.; Tiwari, P.; Warren, J.; Sohn, J.; and MacDonald, B. 2010. Deployment of a service robot to help older people. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5990–5995.

Kollar, T.; Perera, V.; and Veloso, M. 2013. Learning environmental knowledge from task-based human-robot dialog. In *International Conference on Robotics and Automation*.

Rybski, P.; Stolarz, J.; Yoon, K.; and Veloso, M. 2008. Using dialog and human observations to dictate tasks to a learning robot assistant. *Journal of Intelligent Service Robots, Special Issue on Multidisciplinary Collaboration for Socially Assistive Robotics* 1(2):159–167.

Schiffer, S.; Ferrein, A.; and Lakemeyer, G. 2012. Caesar: an intelligent domestic service robot. *Intelligent Service Robotics* 5(4):259–273.

Van Der Zant, T., and Wisspeintner, T. 2007. Robocup@home: Creating and benchmarking tomorrows service robot applications. *Robotic Soccer* 521–528.

Veloso, M.; Biswas, J.; Coltin, B.; Rosenthal, S.; Kollar, T.; Mericli, C.; Samadi, M.; Brandao, S.; and Ventura, R. 2012. Cobots: Collaborative robots servicing multi-floor buildings. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5446–5447.

Zobel, M.; Denzler, J.; Heigl, B.; Nth, E.; Paulus, D.; Schmidt, J.; and Stemmer, G. 2001. Mobsy: Integration of vision and dialogue in service robots. In Schiele, B., and Sagerer, G., eds., *Computer Vision Systems*, volume 2095 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 50–62.