Social State Recognition and Knowledge-Level Planning for Human-Robot Interaction in a Bartender Domain

Ronald P. A. Petrick
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
rpetrick@inf.ed.ac.uk

Mary Ellen Foster
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh EH14 4AS, Scotland, UK
M.E.Foster@hw.ac.uk

Amy Isard
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
amyi@inf.ed.ac.uk

Abstract

We discuss preliminary work focusing on the problem of combining social interaction with task-based action in a dynamic, multiagent bartending domain, using an embodied robot. We show how the users' spoken input is interpreted, discuss how social states are inferred from the parsed speech together with low-level information from the vision system, and present a planning approach that models task, dialogue, and social actions in a simple bartending scenario. This approach allows us to build interesting plans, which have been evaluated in a real-world study, using a general purpose, off-the-shelf planner, as an alternative to more mainstream methods of interaction management.

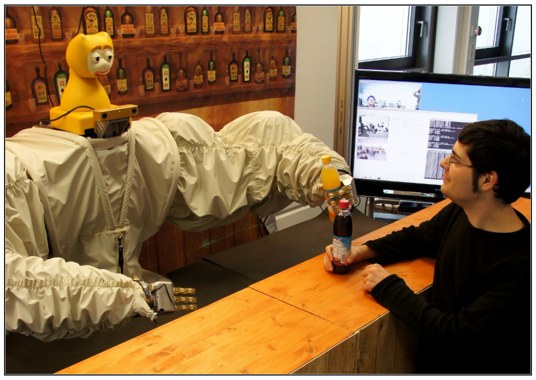

Figure 1: The robot bartender

Introduction

As robots become integrated into daily life, they must increasingly deal with situations in which *socially appropriate interaction* is vital. In such settings, it is not enough for a robot simply to achieve task-based goals; instead, it must also be able to satisfy the social goals and obligations that arise through interactions with people in real-world settings.

Building a robot to meet the goals of social interaction presents several challenges, especially for the reasoning, decision making, and action selection components of such a system. Not only does the robot require the ability to recognise and understand appropriate multimodal social signals—gaze, facial expression, and especially language, the predominant modality for human social interaction—but it must also generate realistic responses using similar modalities.

To address this challenge, we are developing a robot bartender (Figure 1) that is capable of managing interactions with multiple customers in a dynamic setting. For this paper, we focus on a simple drink-ordering scenario, where interactions incorporate a mixture of task-based aspects (e.g., ordering and paying for drinks) and social aspects (e.g., managing multiple simultaneous interactions). Moreover, since the primary interaction modality in this domain is language, users will communicate with the robot bartender via speech and the robot must respond in a similar manner.

One approach to high-level reasoning and action selection is to use general purpose *automated planning* techniques. The ability to reason and plan is essential for an intelligent agent acting in a dynamic and incompletely known world such as the bartending scenario. Automated planning techniques are good at building goal-directed plans of action under many challenging conditions, especially in task-based contexts. Recent work (Steedman and Petrick 2007; Brenner and Kruijff-Korbayová 2008; Benotti 2008; Koller and Petrick 2011) has also investigated the use of automated planning for natural language generation and dialogue—an approach with a long tradition in natural language processing but one that is not the focus of recent, mainstream study.

We focus on four main areas in this paper.

- First, we describe how users' spoken inputs are recognised and interpreted.
- Then, we show how states with task, dialogue, and social features are derived from low-level sensor observations.
- Using these states, we show how plans are generated by modelling the problem as an instance of planning with incomplete information and sensing using a planner called PKS (Petrick and Bacchus 2002; 2004), as an alternative to more mainstream methods of interaction management.
- Finally, we present an embarrassingly simple planning domain that models an initial bartending scenario. This domain has been evaluated in a real-world study, and provides the basis for future work currently underway.

This work forms part of a larger project called JAMES, Joint Action for Multimodal Embodied Social Systems, exploring social interaction with embodied robot systems.[1]

[1]See http://james-project.eu/ for more information.

A customer approaches the bar and looks at the bartender	
ROBOT:	[Looks at Customer 1] How can I help you?
CUSTOMER 1:	A pint of cider, please.
Another customer approaches the bar and looks at the bartender	
ROBOT:	[Looks at Customer 2] One moment, please.
ROBOT:	[Serves Customer 1]
ROBOT:	[Looks at Customer 2]
	Thanks for waiting. How can I help you?
CUSTOMER 2:	I'd like a pint of beer.
ROBOT:	[Serves Customer 2]

Figure 2: The scenario: "Two people walk into a bar".

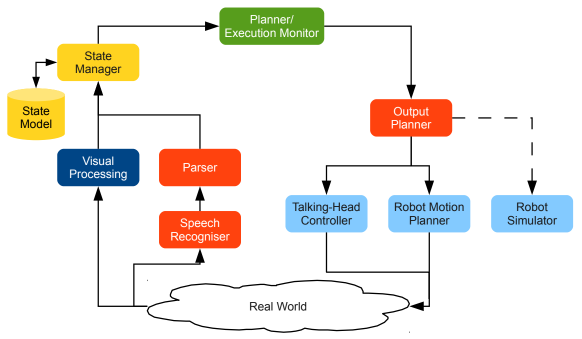

Figure 3: System architecture

Overview of the Scenario and Robot System

In the bartending scenario we discuss in this paper, we support interactions like the one shown in Figure 2: two customers enter the bar area, attract the robot's attention, and order a drink. Even this simple interaction presents challenges: the vision system must accurately track the locations and body postures of the agents; the speech-recognition system must detect and deal with speech in an open setting; the reasoning components must determine that the both customers require attention and should ensure that they are served in the order that they arrived; while the output components must select and execute concrete actions for each output channel that correctly realises high-level plans.

The robot hardware consists of a pair of manipulator arms with grippers, mounted to resemble human arms, along with an animatronic talking head capable of producing facial expressions, rigid head motion, and lip-synchronised synthesised speech. The software architecture (Figure 3) is based on a standard three-layer structure: the low-level components deal with modality-specific, detailed information such as spatial coordinates, speech-recognition hypotheses, and robot arm trajectories; the mid-level components deal with more abstract, cross-modality representations of states and events; while the high level reasons about abstract structures, such as knowledge and action in a logical form.

The low-level input components include a vision system and a linguistic processing system. The vision system tracks the location, facial expressions, gaze behaviour, and body language of all people in the scene in real time (Pateraki, Baltzakis, and Trahanias 2011), while the linguistic processing system combines a speech recogniser with a natural-language parser to create symbolic representations of the speech produced by all users. Low level output components control the animatronic head (which produces lip-synchronised synthesised speech, facial expressions, and gaze behaviour) and the robot manipulator arms (which can point at, pick up, and manipulate objects in the scene).

The primary mid-level input component is the social state manager, which combines information from various low-level input components to estimate the real-time social and communicative state of all users in the scene. On the output side, the main mid-level component is the output planner, which both translates the fleshed-out communicative acts into specific action sequences for the low-level components and coordinates the execution of those sequences.

Finally, the high level includes a knowledge-level planner that generates plans for the robot to achieve its goals, where the plans include a mixture of task actions (e.g., manipulating objects in the world), sensing actions (e.g., using the robot arms to test object properties), and communicative actions (e.g., attracting a user's attention, asking for a drink order). The high-level system also includes a monitor which tracks the execution of the plan steps, detects plan failures, and controls replanning as necessary.

Natural Language Understanding

Natural language plays an important role in input processing, since linguistic interaction is central to social communication. For speech recognition, we use the Microsoft Kinect hardware (Microsoft Corporation 2012) and the associated Microsoft Speech API. This provides a list of intermediate hypotheses for each user utterance and a final best hypothesis, along with a confidence score and an estimate of the source angle. We have created a speech recognition grammar in the SRGS format (Hunt and McGlashan 2004) that covers all expected user utterances in the initial scenario: this constrains the recognition task and allows us to achieve more reliable results. In the current system, we wait until recognition is completed and then pass the top hypothesis to the interpretation system with its confidence score and angle; in the future we also plan to use intermediate hypotheses.

Once the user speech has been recognised, it must be further processed to extract the underlying meaning. To do this, we parse the recognised speech hypothesis using a grammar defined in OpenCCG (White 2006). OpenCCG is an open-source implementation of Combinatory Categorial Grammar (Steedman 2000), a unification-based categorial framework which is both linguistically and computationally attractive. The grammar contains both syntactic and semantic information, and is used both for parsing the linguistic input and for surface realisation of the selected output as described later. As a concrete example, the XML representation of the OpenCCG logical form for the sentence *Please give me a beer* is shown in Figure 4. Once the top speech hypothesis has been parsed, the logical form is then passed on—together with the confidence and angle information—to the state manager for further processing.

State Management

The primary role of the state manager is to turn the continuous stream of messages produced by all of the low-level

```
<lf>
  <node id="w1:situation" pred="give-verb" mood="imp">
    <rel name="ArgOne">
      <node id="w2:animate-being" pred="pron"
        num="sg" pers="1st"/>
    </rel>
    <rel name="ArgTwo">
      <node id="w4:drink" pred="beer"
        det="indef" num="sg"/>
    </rel>
    <rel name="HasProp">
      <node id="w0:politeness" pred="please"/>
    </rel>
  </node>
</lf>
```

Figure 4: OpenCCG logical form for *Please give me a beer*

input and output components into a discrete representation of the world, the robot, and all entities in the scene, combining social, dialogue, and task-based properties. This state is used in two distinct ways in the system processing. On the one hand, this module provides a persistent, queryable interface to the state; on the other hand, it also informs the high-level planner when there are relevant state changes.

The state is made up of a set of *fluents*: first-order predicates and functions that denote particular qualities of the world, robot, and entities. Fluents are defined by the requirements of the scenario (Figure 2): we represent all agents in the scene, their location, torso orientation, and attentional state, along with their drink request if they have made one. In addition, we also store the coordinates of all sensed entities and other properties from the vision system to enable the low-level output components to access them as necessary.

In the current system, the state manager is rule-based. One set of rules infers user social states (e.g., seeking attention) based on the low-level sensor data, using guidelines derived from the study of human-human interactions in the bartender domain (Huth 2011). The state manager also incorporates rules that map from the logical forms produced by the parser into communicative acts (e.g., drink orders), and that use the source localisation from the speech recogniser together with the vision properties to determine which customer is likely to be speaking. A final set of rules determine when new state reports are published, which helps control turn-taking. For example, when the state manager receives a message from the language interpreter including the logical form shown in Figure 4, it first determines which of the agents in the scene is likely to be speaking (based on the interaction history, the source angle, and the vision data), updates that agent's `ordered` and `request` fluents, and then sends the updated state to the high-level reasoning system.

In subsequent versions of the system, the state manager will be enhanced to process more complex messages from the updated input and output components, taking into account the associated confidence scores, and also to deal with the more complex state representations that will be required by the updated high-level reasoning system. To address this, we will draw on recent work in social signal processing (Vinciarelli, Pantic, and Bourlard 2009): we will train supervised learning classifiers on data gathered from humans interacting with both real and artificial bartenders, using methods similar to those employed, for example, by (Kapoor, Burleson, and Picard 2007) and (Bohus and Horvitz 2009).

High-level Planning and Execution Monitoring

The high-level planner is responsible for taking reports from the state manager and producing actions that are sent to the output planner for execution on the robot as speech, head motions, and effector manipulations. All actions in the bartending domain (i.e., task, dialogue, and social) are modelled as part of the same underlying planning domain, using a general purpose planning system, rather than separating task and dialogue acts as is common in many dialogue systems.

We use the PKS planner (Petrick and Bacchus 2002; 2004) for action selection in this work. PKS (Planning with Knowledge and Sensing) is a conditional planner that builds plans in the presence of incomplete information and sensing actions. PKS works at the *knowledge level* by reasoning about how the planner's knowledge changes due to action. PKS is based on a generalisation of STRIPS (Fikes and Nilsson 1971). In STRIPS, the world state is modelled by a single database that is updated due to action. In PKS, the planner's knowledge state, rather than the world state, is represented by five databases, each of which models a particular type of knowledge: K_f stores facts about the world, K_w and K_v store the results of sensing actions that return binary information or general terms, K_x stores limited disjunctive information, and *LCW* stores a type of "local closed world" information (Etzioni, Golden, and Weld 1994). The contents of the databases have a formal interpretation in a modal logic of knowledge, which is restricted to help ensure efficient reasoning. The databases can be inspected using *primitive queries* that ask simple questions about the knowledge state, e.g., whether facts are known ($K(\phi)$) or not known ($-K(\phi)$), whether function values are known ($K_v(t)$), or if the planner "knows whether" certain properties are true or not ($K_w(\phi)$).

Actions in PKS are modelled by sets of *preconditions* that query the agent's knowledge state, and *effects* that update the state. Action preconditions are simply a list of primitive queries. Action effects are described by STRIPS-style "add" and "delete" operations that modify the contents of individual databases, updating the planner's knowledge state. E.g., $add(K_f, \phi)$ adds ϕ to K_f, and $del(K_w, \phi)$ removes ϕ from K_w. Actions are often parameterised; e.g., `serve(?a,?d)` might indicate an action to serve a drink `?d` to an agent `?a`.

PKS constructs plans by reasoning about actions in a forward-chaining manner: if the preconditions of an action are satisfied in the planner's knowledge state, then the action's effects are applied to produce a new knowledge state. PKS can also build plans with branches, by considering the possible outcomes of its K_w and K_v knowledge. Planning continues along each branch until it satisfies the *goal* conditions, also specified as a list of primitive queries.

An associated execution monitor controls replanning in PKS. The monitor takes as input a plan and a description of the sensed state provided by the state manager. The monitor assesses how close an expected, planned state is to a sensed state in order to determine whether a plan should continue to be executed. In the case of a mismatch, the planner is directed to build a new plan, starting from the sensed state.

```
<output>
  <gesture-list>
    <gesture type="Smile"/>
  </gesture-list>
  <action-list>
    <action type="give">
      <object id="obj2" name="juice" type="drink"/>
      <person id="a1"/>
    </action>
  </action-list>
  <speech-list>
    <speech type="inform" politeness="4">
      <person id="a1"/>
      <pred type="hand-over">
        <object type="drink" name="juice" id="obj2"/>
      </pred>
    </speech>
  </speech-list>
</output>
```

Figure 5: XML for the `serve(a1,juice)` action

Output Planning and Language Generation

Output in the robot system is based on processing high-level actions selected by the planner and dividing them into speech, head motion, and arm manipulation behaviours that can be executed in the real world. To do so, we use an XML format which contains specifications for all of the output modalities, which is generated using a rule-based approach, and then passed on to particular output modules.

On the linguistic side, we use OpenCCG to generate the robot language output, with the same OpenCCG grammar used for input, since it also contains the language necessary for speech output. The language output in the XML description is specified in terms of communicative acts based on Rhetorical Structure Theory (RST) (Mann and Thompson 1988). A generation module then translates the RST structure into OpenCCG logical forms, which are sent to the OpenCCG realiser which outputs text strings that can be turned into speech by the robot's animatronic head.

In addition to speech, the robot system also expresses itself through facial expressions, gaze behaviour, and robot manipulation actions. The presentation planner coordinates the output across the various multimodal channels to ensure that it is coordinated both temporally and spatially. The animatronic head can currently express a number of pre-assigned expressions, and the robot arm can perform tasks like grasping objects to hand over a drink to a customer.

For instance, Figure 5 shows an XML specification for a high-level action `serve(a1,juice)` (i.e., "serve juice to agent `a1`"). In this case, the specification results in the robot smiling (an animatronic head facial expression) while handing over a juice (a robot arm manipulation action) and saying to the customer "here is your drink" (speech output).

Planning Interactions for the Robot Bartender

We now consider some example plans we can generate for controlling the robot bartender, using a planning domain which describes the properties and actions in the bartending scenario. Domain properties are divided into two types, predicates and functions, which correspond to fluents from the state manager. For instance, predicates in the planning domain include: `seeksAttention(?a)` (agent `?a` seeks attention), `greeted(?a)` (agent `?a` has been greeted), `ordered(?a)` (agent `?a` has ordered), `served(?a)` (agent `?a` has been served), `otherAttentionReq` (other agents are seeking attention), `badASR(?a)` (agent `?a` was not understood), and `transEnd(?a)` (the transaction with `?a` has ended). Functions include `inTrans = ?a` (the robot is interacting with agent `?a`), and `request(?a) = ?d` (agent `?a` has requested drink `?d`). Predicate arguments are typed so that `?a` accepts a binding of type `agent`, and `?d` accepts a binding of type `drink`. The function `inTrans` maps to type `agent` and `request` maps to type `drink`. As we'll see, many of these properties act as state markers, to help guide plan generation in building realistic interactions.

```
action greet(?a : agent)
    preconds: K(inTrans = nil) & -K(greeted(?a)) &
              K(seeksAttention(?a)) & -K(ordered(?a)) &
              -K(otherAttentionReq) & -K(badASR(?a))
    effects:  add(Kf,greeted(?a)),
              add(Kf,inTrans = ?a)

action ask-drink(?a : agent)
    preconds: K(inTrans = ?a) & -K(ordered(?a))
              -K(otherAttentionReq) & -K(badASR(?a)) &
    effects:  add(Kf,ordered(?a)),
              add(Kv,request(?a))

action serve(?a : agent, ?d : drink)
    preconds: K(inTrans = ?a) & K(ordered(?a)) &
              Kv(request(?a)) & K(request(?a) = ?d)
              -K(otherAttentionReq) & -K(badASR(?a)) &
    effects:  add(Kf,served(?a))

action bye(?a : agent)
    preconds: K(inTrans = ?a) & K(served(?a)) &
              -K(otherAttentionReq) & -K(badASR(?a))
    effects:  add(Kf,transEnd(?a)),
              add(Kf,inTrans = nil)

action not-understand(?a : agent)
    preconds: K(inTrans = ?a) & K(badASR(?a))
    effects:  del(Kf,badASR(?a))
```

Figure 6: PKS actions in a single agent interaction

Actions use domain properties to describe their preconditions and effects. Our domain has seven high-level actions: `greet(?a)` (greet an agent `?a`), `ask-drink(?a)` (ask agent `?a` for a drink order), `serve(?a,?d)` (serve drink `?d` to agent `?a`), `bye(?a)` (end an interaction with agent `?a`), `not-understand(?a)` (alert agent `?a` that its utterance was not understood), `wait(?a)` (tell agent `?a` to wait), and `ack-wait(?a)` (thank agent `?a` for waiting). Definitions for the first five actions (the actions required for single agent interactions) are given in Figure 6.

Actions are described at an abstract level and include a mix of task, sensory, and speech acts. For instance, `serve` is an ordinary planning action with a deterministic effect (i.e., it adds definite knowledge to the planner's K_f database); however, when executed it causes the robot to hand over a drink to an agent and confirm the drink order through speech. Actions like `greet` and `bye` are modelled in a similar way but only map to speech output at the robot level (i.e., "hello" and "good-bye"). The `ask-drink` action is modelled as a sensing action in PKS: `request` is added to the planner's K_v database as an effect, indicating that `request`'s mapping (i.e., an agent's drink order) will become known at execution time. The `not-understand` action is used to inform an agent that it was not understood. The `wait` and `ack-wait` actions are used to control interactions when multiple agents are seeking the attention of the bartender.

The initial state, which includes a list of the drinks and agents in the bar, is not hard-coded. Instead, this information is supplied to the planner by the state manager. The `inTrans` function is also initially set to `nil` to indicate that the robot isn't interacting with any agents. The goal is simply to serve each agent seeking attention: `forallK(?a : agent) K(seeksAttention(?a)) => K(transEnd(?a))`. This goal is viewed as a rolling target and reassessed each time PKS receives a state report.

Ordering a drink: First, we consider the case where a single agent `a1` is seeking attention (denoted by storing `seeksAttention(a1)` in PKS's K_f database), and no drink restrictions are specified. PKS builds the following plan to achieve the goal:

```
greet(a1),                  [Greet agent a1]
ask-drink(a1),              [Ask a1 for drink order]
serve(a1,request(a1)),      [Give the drink to a1]
bye(a1).                    [End the transaction]
```

In this case, a simple linear plan is built by reasoning about the planner's knowledge state. The `greet(a1)` action is chosen first since `inTrans = nil` and `seeksAttention(a1)` are known, but `greeted(a1)`, `badASR(a1)`, and `otherAttentionReq` are not known. After `greet(a1)`, the planner is in a state where `inTrans = a1` and `greeted(a1)` are known, so the `ask-drink(a1)` action is chosen. As an effect of this action, the planner comes to know the value of `request(a1)` which acts as a placeholder whose value (i.e., `a1`'s drink order) will only be known after execution. Finally, adding `bye(a1)` to the plan causes `transEnd(a1)` to become true, satisfying the goal.

Ordering a drink with restricted choice: The previous plan relies on placeholder terms in parameterised plans, which require additional reasoning to build, potentially slowing down plan generation. Consider a second example with a single agent `a1`, where the planner is also told there are three possible drinks that can be ordered: juice, water, and beer. The drinks are added to the domain description as new constants of type `drink`: `juice`, `water`, and `beer`. This information is also recorded in the planner's databases as a restriction on the set of possible mappings for `request(a1)`. PKS can now build a plan of the form:

```
greet(a1),                  [Greet agent a1]
ask-drink(a1),              [Ask a1 for drink order]
branch(request(a1))         [Form branching plan]
   K(request(a1)=juice):    [If order is juice]
      serve(a1,juice)       [Serve juice to a1]
   K(request(a1)=water):    [If order is water]
      serve(a1,water)       [Serve water to a1]
   K(request(a1)=beer):     [If order is beer]
      serve(a1,beer)        [Serve beer to a1]
bye(a1).                    [End the transaction]
```

In this case, a conditional plan is built. After a drink is ordered, the possible values for `request(a1)` are tested by creating a plan branch for each possible mapping: in the first branch `request(a1)=juice` is assumed to be true; in the second branch `request(a1)=water` is true; and so on. Planning continues in each branch, and an appropriate `serve` action is added to deliver the corresponding drink. In more complex domains (currently under development), each branch may require different actions to serve a drink, such as putting the drink in a special glass or interacting further with the agent (i.e., "would you like ice in your water?").

When things go wrong: Once a plan is built, it is sent to the output planner, one action at a time, for execution on the robot. During execution, PKS's execution monitor assesses plan correctness by comparing sensed state reports from the state manager against states predicted by the planner. In the case of disagreement, for instance due to action failure, the planner is invoked to construct a new plan from the sensed state. This method is particularly useful for handling unexpected responses by agents interacting with the robot.

For example, if agent `a1`'s response to `ask-drink(a1)` was not understood due to low-confidence speech recognition, the state report sent to PKS will have no value for `request(a1)`, and `badASR(a1)` will also appear. The monitor will detect this and direct PKS to build a new plan. One result is a modified version of the old plan that first informs the agent they were not understood before repeating the `ask-drink` action and continuing the old plan:

```
not-understand(a1),    [Alert a1 it was not understood]
ask-drink(a1),         [Ask a1 again for drink order]
...continue with remainder of old plan...
```

Another useful consequence of this approach is that certain types of over-answering can be handled by the monitor through replanning. For instance, a `greet(a1)` action by the bartender might cause `a1` to respond with an utterance that includes a drink order. In this case, the state manager would include an appropriate `request(a1)` mapping in the state description, along with `ordered(a1)`. The monitor can detect that the preconditions for `ask-drink(a1)` aren't met and direct PKS to replan. A new plan would then omit `ask-drink` and instead serve the requested drink.

Ordering drinks with multiple agents: Our simple planning domain also enables more than one agent to be served if the state manager reports multiple customers are seeking attention. For instance, say that there are two agents, `a1` and `a2` (as in Figure 2). One possible plan that might be built is:

```
wait(a2),                   [Tell agent a2 to wait]
greet(a1),                  [Greet agent a1]
ask-drink(a1),              [Ask a1 for drink order]
serve(a1,request(a1)),      [Give the drink to a1]
bye(a1),                    [End a1's transaction]
ack-wait(a2),               [Thank a2 for waiting]
ask-drink(a2),              [Ask a2 for drink order]
serve(a2,request(a2)),      [Give the drink to a2]
bye(a2).                    [End a2's transaction]
```

In this plan, the first agent orders a drink and is served, followed by the second agent. The `wait` and `ack-wait` actions (which aren't required in the single agent case) are used to defer a transaction with agent a2 and resume it when the transaction with a1 has finished. (The otherAttentionReq state property, which is a derived property defined in terms of seeksAttention, ensures that other agents seeking attention are told to wait before an agent is served.)

One drawback with our current domain encoding is that agents who have been asked to wait are not necessarily served in the order they are deferred. While plans in this domain might still achieve the goal of serving drinks to all agents, from a social interaction point of view they potentially fail to be appropriate (depending on local pub culture), since some agents may be served before other agents that have been waiting for longer periods of time. Since socially appropriate interactions are central to the goals of this work, we are addressing this issue by modifying our domain description to introduce an ordering on waiting agents.

Discussion and Related Work

We have carried out a user evaluation in which 31 participants interacted with the bartender in a range of social situations, resulting in a wide range of objective and subjective measures. Overall, most customers were successful in obtaining a drink from the bartender in all scenarios, and the robot dealt appropriately with multiple simultaneous customers and with unexpected situations including over-answering and input-processing failure. The factors that had the greatest impact on subjective user satisfaction were task success and dialogue efficiency. More details of the user study are presented in (Foster et al. 2012).

The general focus of this work fits into the area of *social robotics*: "the study of robots that interact and communicate with themselves, with humans, and with their environment, within the social and cultural structure attached to their roles" (Ge and Matarić 2009). While we build on recent work in this area, we address a style of interaction that is distinct in two ways. First, many existing projects consider social interaction as the primary goal (Breazeal 2005; Dautenhahn 2007; Castellano et al. 2010), while the robot bartender supports social communication in the context of a cooperative, task-based interaction. Second, while most social robotics systems deal with one-on-one interactive situations, the robot bartender must deal with dynamic, multiparty scenarios: people will be constantly entering and leaving the scene, so the robot must constantly choose appropriate social behaviour while interacting with a series of new partners. In this way, the bartender is similar to the system developed by Bohus and Horvitz, which also handles situated, open-world, multimodal dialogue in scenarios such as a reception desk and a question-answering game.

The use of general purpose planning techniques is central to our work, an idea that has a long tradition in natural language generation and dialogue research. Early approaches to generation as planning (Perrault and Allen 1980; Appelt 1985; Young and Moore 1994) focused primarily on high-level structures, such as speech acts and discourse relations, but suffered due to the inefficiency of the planners available at the time. As a result, recent mainstream research has tended to segregate task planning from discourse and dialogue planning, capturing the latter with more specialised approaches such as finite state machines, information state approaches, speech-act theories, dialogue games, or theories of textual coherence (Traum and Allen 1992; Green and Carberry 1994; Matheson, Poesio, and Traum 2000; Beun 2001; Asher and Lascarides 2003; Maudet 2004).

However, there has been renewed interest recently in applying modern planning techniques to problems in generation, such as sentence planning (Koller and Stone 2007), instruction giving (Koller and Petrick 2011), and accommodation (Benotti 2008). The idea of using planning for interaction management has also being revisited, by viewing the problem as an instance of planning with incomplete information and sensing (Stone 2000). This view is also implicit in early BDI-based approaches, e.g., (Litman and Allen 1987; Bratman, Israel, and Pollack 1988; Cohen and Levesque 1990; Grosz and Sidner 1990). Initial work using PKS has explored this connection (Steedman and Petrick 2007), but fell short of implementing a tool to leverage the relationship for efficient dialogue planning. A related approach (Brenner and Kruijff-Korbayová 2008) manages dialogues by interleaving planning and execution, but fails to solve the consequent problem of deciding when best to commit to plan execution versus plan construction. Thus, while recent planning approaches are promising, many are not yet fully mature, and fall outside the mainstream of current natural language dialogue research.

Conclusions and Future Work

In this paper we have discussed initial work aimed at combining social interaction with task-based action in a dynamic, multiagent bartending domain, using an embodied robot. Action selection uses the general purpose, off-the-shelf PKS planner, combined with a social state manager and plan monitor, supported by a vision and linguistic processing system. Although this work is preliminary, we have produced a system that has been evaluated with human users. We are currently in the process of extending this work to more complex scenarios in the bartending domain, including agents that can ask questions about drinks, a bartender that can query agents for more information, agents that can order multiple drinks, and situations where an agent may terminate an interaction early. We believe that general purpose automated planners offer potential solutions to the problem of action selection in task-based interactive systems, as an alternative to more specialised methods, such as those used in many mainstream natural language dialogue systems.

Acknowledgements

The authors thank their JAMES colleagues who helped implement the bartender system described in this paper: Andre Gaschler and Manuel Giuliani from fortiss Gmbh, Maria Pateraki from the Foundation for Research and Technology–Hellas, and Richard Tobin from the University of Edinburgh. The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007–2013) under grant agreement No. 270435.

References

Appelt, D. 1985. *Planning English Sentences*. Cambridge, England: Cambridge University Press.

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

Benotti, L. 2008. Accommodation through tacit sensing. In *Proceedings of LONDIAL-2008*, 75–82.

Beun, R.-J. 2001. On the generation of coherent dialogue. *Pragmatics and Cognition* 9:37–68.

Bohus, D., and Horvitz, E. 2009. Dialog in the open world: platform and applications. In *Proceedings of the 2009 International Conference on Multimodal Interfaces (ICMI-MLMI 2009)*, 31–38.

Bratman, M.; Israel, D.; and Pollack, M. 1988. Plans and resource-bounded practical reasoning. *Computational Intelligence* 4:349–355.

Breazeal, C. 2005. Socially intelligent robots. *interactions* 12(2):19–22.

Brenner, M., and Kruijff-Korbayová, I. 2008. A continual multiagent planning approach to situated dialogue. In *Proceedings of LONDIAL-2008*, 67–74.

Castellano, G.; Leite, I.; Pereira, A.; Martinho, C.; Paiva, A.; and McOwan, P. W. 2010. Affect recognition for interactive companions: challenges and design in real world scenarios. *Journal on Multimodal User Interfaces* 3(1):89–98.

Cohen, P., and Levesque, H. 1990. Rational interaction as the basis for communication. In *Intentions in Communication*. MIT Press. 221–255.

Dautenhahn, K. 2007. Socially intelligent robots: dimensions of human-robot interaction. *Philosophical Transactions of the Royal Society B: Biological Sciences* 362(1480):679–704.

Etzioni, O.; Golden, K.; and Weld, D. 1994. Tractable closed world reasoning with updates. In *Proceedings of KR-1994*, 178–189.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Foster, M. E.; Gaschler, A.; Giuliani, M.; Isard, A.; Pateraki, M.; and Petrick, R. P. A. 2012. "Two people walk into a bar": Dynamic multi-party social interaction with a robot agent. In submission.

Ge, S. S., and Matarić, M. J. 2009. Preface. *International Journal of Social Robotics* 1(1):1–2.

Green, N., and Carberry, S. 1994. A hybrid reasoning model for indirect answers. In *Proceedings of ACL-94*, 58–65. ACL.

Grosz, B., and Sidner, C. 1990. Plans for discourse. In *Intentions in Communication*. MIT Press. 417–444.

Hunt, A., and McGlashan, S. 2004. Speech recognition grammar specification version 1.0. W3C recommendation, W3C. `http://www.w3.org/TR/2004/REC-speech-grammar-20040316/`.

Huth, K. 2011. Wie man ein Bier bestellt. Master's thesis, Universität Bielefeld.

Kapoor, A.; Burleson, W.; and Picard, R. W. 2007. Automatic prediction of frustration. *International Journal of Human-Computer Studies* 65(8):724–736.

Koller, A., and Petrick, R. P. A. 2011. Experiences with planning for natural language generation. *Computational Intelligence* 27(1):23–40.

Koller, A., and Stone, M. 2007. Sentence generation as planning. In *Proceedings of the ACL*, 336–343.

Litman, D., and Allen, J. 1987. A plan recognition model for subdialogues in conversation. *Cognitive Science* 11:163–200.

Mann, W. C., and Thompson, S. A. 1988. Rhetorical structure theory: Toward a functional theory of text organization. *Text* 8(3):243–281.

Matheson, C.; Poesio, M.; and Traum, D. 2000. Modeling grounding and discourse obligations using update rules. In *Proceedings of NAACL 2000*.

Maudet, N. 2004. Negotiating language games. *Autonomous Agents and Multi-Agent Systems* 7:229–233.

Microsoft Corporation. 2012. Kinect for XBox 360. `http://www.xbox.com/kinect`.

Pateraki, M.; Baltzakis, H.; and Trahanias, P. 2011. Visual tracking of hands, faces and facial features as a basis for human-robot communication. In *Proceedings of the Workshop on Visual Tracking and Omnidirectional Vision, held within the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Perrault, C. R., and Allen, J. F. 1980. A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics* 6(3–4):167–182.

Petrick, R. P. A., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of AIPS-2002*, 212–221.

Petrick, R. P. A., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proc. of ICAPS-2004*, 2–11.

Steedman, M., and Petrick, R. P. A. 2007. Planning dialog actions. In *Proceedings of SIGdial 2007*, 265–272.

Steedman, M. 2000. *The Syntactic Process*. MIT Press.

Stone, M. 2000. Towards a computational account of knowledge, action and inference in instructions. *Journal of Language and Computation* 1:231–246.

Traum, D., and Allen, J. 1992. A speech acts approach to grounding in conversation. In *Proceedings of ICSLP-92*, 137–140.

Vinciarelli, A.; Pantic, M.; and Bourlard, H. 2009. Social signal processing: Survey of an emerging domain. *Image and Vision Computing* 27(12):1743–1759.

White, M. 2006. Efficient realization of coordinate structures in Combinatory Categorial Grammar. *Research on Language and Computation* 4(1):39–75.

Young, R. M., and Moore, J. D. 1994. DPOCL: a principled approach to discourse planning. In *Proceedings of the 7th International Workshop on Natural Language Generation*, 13–20.