Cognitive Robotics
AAAI Technical Report WS-12-06

What Would You Like to Drink? Recognising and Planning with Social States in a Robot 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

Abstract

A robot coexisting with humans must not only be able to successfully perform physical tasks, but must also be able to interact with humans in a socially appropriate manner. In many social settings, this involves the use of social signals like gaze, facial expression, and language. In this paper 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 discuss how social states are inferred from low-level sensors, using vision and speech as input modalities, 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 with human subjects, using a general purpose, off-the-shelf planner, as an alternative to more mainstream methods of interaction management.

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 (e.g., gaze, facial expression, and language), 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 dealing with multiple customers in a dynamic setting. For the purpose of this paper, we focus on a simple drink-ordering scenario. Interactions in this scenario incorporate a mixture of task-based aspects (e.g., ordering and paying for drinks) and social aspects (e.g., managing multiple simultaneous interactions). Moreover, the primary interaction modality is speech; users 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 *automated planning* techniques. The ability

Figure 1: The robot bartender

to reason and plan is essential for an intelligent agent acting in a dynamic and incompletely known world such as the bartending scenario. General purpose, 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.

While planning offers a possible tool for action selection, real-world domains like the bartender scenario present challenges for plan generation, execution, and monitoring. First, the domain is inherently dynamic and certain aspects of it, such as the initial state, cannot be defined offline (e.g., the number of customers in the bar). Instead, they must be provided to the planner based on observations of the scene sensed by low-level input modalities such as vision and speech. Second, plan generation involves a mix of traditional task-based actions (e.g., handing the customer a drink) and speech acts (e.g., asking a customer for a drink order). Finally, monitoring in such a domain is essential: plans cannot account for all eventualities, so unexpected state changes and plan failures must be detected and dealt with.

The problem of integrating low-level sensor data with symbolic planners introduces representational difficulties that must be overcome. While sensors tend to generate continuous streams of low-level, noisy data, high-level planning

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".

systems typically use representations based on discrete models of objects, properties, and actions, described in logical languages. Furthermore, some parts of the state may not be observable. Thus, a central task necessary for supporting high-level planning in social domains (as in other robot domains) is appropriate *state recognition and management*.

We focus on three main areas in this paper.

- First, 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]

The rest of the paper is organised as follows: first, we present an overview of the bartending scenario and the robot system; we then describe the state manager and show how states are inferred from low-level sensor data; the high-level planner and execution monitor are then presented, along with a description of the planning domain and example plans in the bartending scenario. We conclude by discussing related work, and extensions to our work currently underway.

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 and attempt to 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.

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

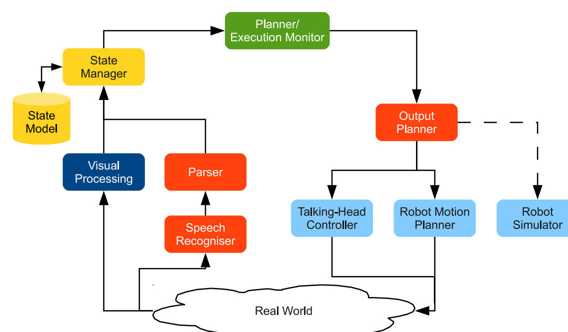

Figure 3: System architecture

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, 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 planner that generates goal-directed plans for the robot. 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 planned actions, detects plan failures, and controls replanning.

In the remainder of this paper, we concentrate on the operation of the mid-level and high-level components.

State Management

The primary role of the state manager is to turn the continuous stream of messages produced by the low-level 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. The resulting state is used in two distinct ways in the system processing. On the one hand, the state manager provides a persistent, queryable interface to the state: for example, it stores the world coordinates of all entities as reported by the vision system so that the robot is able to gaze at a particular agent when needed. On the other hand, it also informs the planner and the execution monitor whenever there is a relevant state change.

A model of state management

At a formal level, we can model the operation of the state manager as follows. The low-level system components correspond to a set Σ of sensors, $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, where each sensor σ_i returns an observation $obs(\sigma_i)$ about some aspect of the world. If appropriate, a primary sensor (such as a speech recogniser or a body-pose estimator) may have an associated sensor that indicates the estimated reliability of the observation, capturing the fact that real-world sensors generally produce noisy results.

The state representation is based on a set F of fluents, $F = \{f_1, f_2, \ldots, f_m\}$: first-order predicates and functions that denote particular qualities of the world, robot, and entities. The value $f_{i,t}$ of a fluent f_i at a particular time point t is a function of the observations returned by the sensor set, along with the set of fluent values from the previous time point; i.e., $f_{i,t} = \Gamma_i(\Sigma, F_{t-1})$. Typically, each fluent depends on a subset of the sensor observations, and the mapping is not exclusive: any given sensor may map to zero, one, or many fluents as appropriate. Including the previous state in the function permits fluents with Markov-like properties, where the value is dependent on the immediate interaction history.

A state is then a snapshot F_t of the values of all instantiated fluents at a time t during the interaction. States represent a point of intersection between the low-level data produced by the sensors and the high-level representations needed by the planner and the execution monitor, since states are induced from a set of sensor observations and the corresponding sensor/fluent mappings (i.e., the functions Γ_i).

The set Σ of available sensors is defined by the low-level system components, while the set F of required fluents is provided by the high-level reasoning system. Implementing the state manager therefore consists primarily of defining the set of mapping functions Γ_i. In the context of social robotics, this is the problem of *social signal processing* (Vinciarelli, Pantic, and Bourlard 2009), which is a topic that has received an increasing amount of attention in recent years. The most common technique is to use labelled data to train supervised learning models such as Support Vector Machines (SVMs) to classify the sensor data; this often requires the use of signal processing and feature extraction to convert the sensor data into a form suitable for SVM training.

A crucial issue is modelling the temporal dynamics of user behaviour: often, it is not the sensor data in any single frame that determines the value of a state fluent, but rather the patterns found in a sequence of signals. Also, when determining the value of fluents that combine information from multiple signals, the relevant information may not occur simultaneously, so temporal cross-modal fusion is necessary. It is for these reasons that the previous state F_{t-1} is included as an argument to the Γ_i functions, as mentioned above.

In addition to maintaining and updating the representation of the state, the state manager must also decide when to publish updated state reports to the rest of the system. That is, it must decide which changes to the state are "interesting". This choice is often not clear-cut: while small fluctuations in the sensed location of an individual entity are probably not worth informing the rest of the system about, and the appearance or disappearance of an entity is likely to be interesting, there is a whole space of decisions between these two extremes. In practice, this decision is generally made on an application-specific basis.

State management in the robot bartender

In the robot system, we consider each low-level input component to be made up of a number of sensors, as follows. The linguistic interpreter corresponds to two primary sensors: one that observes the parsed content of the recognised speech, and another that returns the estimated angle of the sound source. Both of these sensors also have associated confidence scores, which are represented as additional sensors in the model. On the other hand, the vision system senses a large number of properties about the agents and objects in the world, including the location, face and torso orientation, and body posture, and therefore corresponds to numerous individual sensors, again with confidence scores.

As well as the input components, the low-level output components are also included in the model as additional sensors. So, for example, the robot arms provide information about the start and end of any manipulation actions as well as indications of success or failure, while the speech synthesiser reports the start and end of all utterances produced by the system. Modelling the output components as additional sensors allows information from these sources to be included in the state (e.g., the success or failure of physical world actions), and also ensures that the state always accurately reflects the current state of turn-taking in the interaction (i.e., whether the robot is currently moving and/or speaking).

The state fluents are defined by the requirements of the scenario (Figure 2): we represent all agents in the scene, and keep track of 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.

In subsequent versions of the system, the state management component will be enhanced to support more complex scenarios: this will involve processing more complex messages from the updated input and output components, including taking into account the associated confidence scores, and also dealing with the more complex state representations that will be required by the updated high-level reasoning system. Dealing with this more complex setting will require defining much more complicated mapping functions. To address this, we will make use of supervised learning techniques trained 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).

Planning and Execution Monitoring

The high-level planner is responsible for taking state reports from the state manager and producing actions that are executed on the robot platform as speech, head motions, and effector manipulations. A related component, the execution monitor, is responsible for tracking the execution of planned actions, to ensure the high-level goals of the system are being met. In the case of action failures or significant plan divergences, alternative actions must be planned as necessary.

In this work we use the PKS planner (Petrick and Bacchus 2002; 2004) for action selection, and a new monitor developed for PKS to control plan execution and replanning activities. All actions in the bartending domain (i.e., task, dialogue, and social) are modelled as part of the same underlying planning domain, rather than using specialised tools as is common practice in modern interactive dialogue systems. Thus, all high-level action selection is determined by the same general purpose planning mechanism.

Planning with Knowledge and Sensing (PKS)

PKS (Planning with Knowledge and Sensing) is a conditional planner that constructs plans in the presence of incomplete information and sensing actions. PKS works at the "knowledge-level" by reasoning about how the planner's knowledge state, rather than the world state, changes due to action. PKS works with a restricted subset of a first-order language, and a limited amount of inference, allowing it to support a rich representation with features such as functions and variables. This approach differs from planners that work with possible worlds models or belief states. However, as a trade-off, its restricted representation means that certain types of knowledge cannot be directly modelled in PKS.

PKS is based on a generalisation of STRIPS (Fikes and Nilsson 1971). In STRIPS, the state of the world is modelled by a single database. Actions update this database and, by doing so, update the planner's world model. In PKS, the planner's knowledge state, rather than the world state, is represented by a set of five databases, each of which models a particular type of knowledge. The contents of these databases have a fixed, formal interpretation in a modal logic of knowledge. Actions can modify any of the databases, which has the effect of updating the planner's knowledge state. To ensure efficient inference, PKS restricts the type of knowledge (especially disjunctions) that it can represent:

$\mathbf{K_f}$: This database is like a STRIPS database except that both positive and negative facts are permitted and the closed world assumption is not applied. K_f is used for modelling action effects that change the world. K_f can include any ground literal ℓ, where $\ell \in K_f$ means "the planner knows ℓ." K_f can also contain known function (in)equality mappings.

$\mathbf{K_w}$: This database models the plan-time effects of "binary" sensing actions. $\phi \in K_w$ means that at plan time the planner either "knows ϕ or knows $\neg\phi$," and that at execution time this disjunction will be resolved.

$\mathbf{K_v}$: This database stores information about function values that will become known at execution time. In particular, K_v can model the plan-time effects of sensing actions that return constants. K_v can contain any unnested function term f, where $f \in K_v$ means that at plan time the planner "knows the value of f." At execution time the planner will have definite information about f's value. As a result, PKS is able to use K_v terms as "run-time variables" (Etzioni et al. 1992) or placeholders in its plans, and can also form certain types of conditional branches using such information.

$\mathbf{K_x}$: This database models the planner's "exclusive-or" knowledge of literals, namely that the planner knows "exactly one of a set of literals is true." Entries in K_x have the form $(\ell_1|\ell_2|\ldots|\ell_n)$, where each ℓ_i is a ground literal. Such formulae represent a particular type of disjunctive knowledge that is common in many planning scenarios, namely that "exactly one of the ℓ_i is true."

(A fifth database that stores "local closed world" information (Etzioni, Golden, and Weld 1994) is not used here.)

PKS's databases can be inspected through a set of *primitive queries* that ask simple questions about the planner's knowledge state, namely whether facts are (not) known to be true (a query of the form $[\neg]K(\phi)$), whether function values are (not) known (a query $[\neg]K_v(t)$), or if the planner "knows whether" certain properties are true or not (a query $[\neg]K_w(\phi)$). An inference algorithm evaluates primitive queries by checking the contents of the databases, taking into consideration knowledge from different databases.

An action in PKS is modelled by a set of *preconditions* that query the agent's knowledge state, and a set of *effects* that update the state. Action preconditions are simply a list of primitive queries. Action effects are described by a collection of STRIPS-style "add" and "delete" operations that modify the contents of individual databases. E.g., $add(K_f, \phi)$ adds ϕ to K_f, and $del(K_w, \phi)$ removes ϕ from K_w.

PKS constructs plans by reasoning about actions in a simple forward-chaining manner: if the preconditions of an action are satisfied by the planner's knowledge state, then the action's effects are applied to the state to produce a new knowledge state. Planning then continues from the resulting 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.

PKS is aided by an execution monitor which controls replanning. The monitor takes as input a PKS plan, whose execution it tracks, and a state description denoting the sensed state, in this case provided by the state manager. The task

of the monitor is to assess how close an expected, planned state is to a sensed state in order to determine whether a plan should continue to be executed. To do this, it tries to ensure that a state still permits the next (n) action(s) in the plan to be executed, by testing an action's preconditions against sensed properties. In the case of a mismatch, the planner is directed to build a new plan, using the sensed state as its initial state.

A simple bartending domain

A PKS planning domain for the bartending scenario describes the domain's properties and actions, denoting particular features of the world, agents, and objects. Domain properties are divided into two types: predicates and functions. In this case, the planning domain properties are based on similar fluents defined in the state manager. In particular, predicates in the planning domain include:

- `seeksAttn(?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,
- `otherAttnReq`: other agents are seeking attention,
- `badASR(?a)`: agent `?a` was not understood, and
- `transEnd(?a)`: the transaction with `?a` has ended.

Two functions are also defined:

- `inTrans = ?a`: the robot is interacting with `?a`, and
- `request(?a) = ?d`: agent `?a` has requested drink `?d`.

We use a typed version of the domain with two types: `agent` and `drink`. All predicate arguments accept constants of type `agent`, while `inTrans` maps to type `agent`, and `request` takes an argument of type `agent` and maps to type `drink`.

Actions in the bartending domain use domain properties to describe their preconditions and effects. Our domain includes 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 4. Actions are described at an abstract level and include a mix of physical, sensory, and speech acts. For instance, `serve` is a standard planning action with a deterministic effect (i.e., it adds definite knowledge to the planner's K_f database); however, when executed at the robot level 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 as `serve` but only map to speech output at the robot level (e.g., "hello" and "good-bye"). The most interesting action is `ask-drink` which is modelled as a sensing action in PKS: the function term `request` is added to the planner's K_v database as an effect, indicating that the mapping for this piece of information will become known at execution time.

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

action bye(?a : agent)
    preconds: K(inTrans = ?a) & K(served(?a)) &
              -K(otherAttnReq) & -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 4: PKS actions in a single agent interaction

The `not-understand` action is used as a directive to the speech output system to produce an utterance that (hopefully) causes the agent to repeat its last response. The `wait` and `ack-wait` actions are used to control interactions when multiple agents are seeking the attention of the bartender.

Most of the domain properties act as state markers for the actions, to help guide the interaction through a type of standard "script" (i.e., an interaction begins with `greet` and ends with `bye`). These properties map to their counterparts provided by the state manager. However, since dialogue is inherently noisy there are still opportunities for things to go wrong, and for plans in this domain to exhibit interesting behaviour, even though the domain model is quite simple.

Example plans in the bartending domain

We now consider some example plans we can generate in the above domain. However, in order to do so we require a description of the domain's initial state and goal, in addition to the above action definitions. The initial state, which includes a list of the objects (drinks) and agents (customers) in the bar, is not hard-coded in the domain description. Instead, this information is supplied to the planner by the state manager. Changes in the object or agent list are also sent to the planner, causing it to update its domain model. The `inTrans` function is initially set to `nil` to indicate that the robot isn't interacting with any agents. The planner's goal is simply to serve each agent seeking attention, represented as:

```
forallK(?a : agent)
     K(seeksAttn(?a)) => K(transEnd(?a)).
```

This goal is viewed as a rolling target which is reassessed each time PKS receives a state report from the state manager.

Ordering a drink: In our first example, we consider the case where there is a single agent `a1`. No specific drinks are defined and no other state information is supplied, except that the robot is not interacting with any agent (i.e., `inTrans = nil` $\in K_f$). The appearance of `a1` seeking attention is reported to the planner in an initial state report, which has the effect of adding a new constant named `a1` of type agent to the planner's domain description, and adding a new fact `seeksAttn(a1)` to the initial K_f database. Using this initial state and the above actions, PKS can build 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]

Initially, the planner can choose the `greet(a1)` action since `inTrans = nil` $\in K_f$ and `seeksAttn(a1)` $\in K_f$, and the other preconditions are trivially satisfied (i.e., none of `greeted(a1)`, `ordered(a1)`, `otherAttnReq`, or `badASR(a1)` are in K_f). After `greet(a1)`, the planner is in a state where `inTrans = a1` $\in K_f$ and `greeted(a1)` $\in K_f$. The `ask-drink(a1)` action can now be chosen, updating PKS's knowledge state so that `ordered(a1)` $\in K_f$ and `request(a1)` $\in K_v$. Consider the `serve(a1,request(a1))` action. Since `inTrans = a1` remains in K_f, the first precondition of the action is satisfied. Since `ordered(a1)` $\in K_f$, the second precondition, `K(ordered(a1))`, holds. Also, since `request(a1)` $\in K_v$, the third precondition `Kv(request(a1))` holds (i.e., the value of `request(a1)` is known). The fourth precondition, `K(request(a1)=request(a1))` is trivially satisfied since both sides of the equality are syntactically equal; this also has the effect of binding `request(a1)` to `serve`'s second parameter. The remaining two preconditions are trivially satisfied. Thus, `request(a1)` acts as a run-time variable whose definite value (i.e., `a1`'s drink order) will become known after action execution. The action updates K_f so that `served(a1)` $\in K_f$, leaving K_v unchanged. Finally, `bye(a1)` is added to the plan resulting in `inTrans = nil` $\in K_f$ and `transEnd(a1)` $\in K_f$, satisfying the goal.

Ordering a drink with restricted choice: The above plan relies on PKS's ability to use known function terms as run-time variables in parameterised plans. However, doing so requires additional reasoning, potentially slowing down plan generation in domains where many such properties must be considered. Furthermore, it does not restrict the possible mappings for `request`, except that it must be a drink.

Consider a second example, where there is again a single agent `a1` seeking attention but the planner is also told there are three possible drinks that can be ordered: juice, water, and beer. In this case, the drinks are represented as new PKS constants of type `drink`, i.e., `juice`, `water`, and `beer`. Information about the possible drinks is also put into PKS's initial K_x database as the formula `(request(a1)=juice|request(a1)=water|request(a1)=beer)`. In terms of PKS's knowledge, this restricts 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 the drink is ordered, the possible values for `request(a1)` are tested by creating a plan branch for each possible mapping. Each branch considers a state where the planner has definite knowledge of one mapping. E.g., in the first branch `request(a1)=juice` is assumed to be in the K_f database; in the second branch `request(a1)=water` is in K_f; and so on. Planning continues in each branch under each assumption. (We note that this type of branching was only possible because the planner had initial K_x knowledge that restricted `request(a1)`, combined with K_v knowledge provided by the `ask-drink` action.) Along each branch, an appropriate `serve` action is added to deliver the appropriate 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 has been built, it is sent for execution by the robot, one action at a time. Each high-level action is divided into speech, head motion, and manipulation behaviours using a simple rule-based system before they are executed in the real world. After execution has started, PKS's execution monitor is used to assess plan correctness by comparing subsequent state reports from the state manager against states predicted by the planner. In the case of disagreement, for instance due to unexpected outcomes like action failure, the planner is invoked to construct a new plan using the sensed state as its new initial state. This method is particularly useful for responding to unexpected responses by agents interacting with the bartender.

For example, if the planner receives a report that `a1`'s response to `ask-drink(a1)` was not understood, for instance 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. This will be detected by the monitor and PKS will be directed to build a new plan. One result is a modified version of the original plan that first informs `a1` 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...	

Thus, replanning produces a loop that repeats an action in an attempt to obtain the information the planner requires.

Another useful consequence of this approach is that certain types of over-answering by the interacting agent can be handled by the execution monitor through replanning. For

instance, a `greet(a1)` action by the bartender might cause the customer 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 would detect that the preconditions for `ask-drink(a1)` aren't met and would direct PKS to replan. A new plan could then omit `ask-drink` and instead proceed to serve the requested drink to the agent.

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]
```

Thus, `a1` orders a drink and is served, followed by `a2`. The `wait` and `ack-wait` actions (which aren't required in the single agent case) are used to defer a transaction with `a2` and resume it when the transaction with `a1` has finished. (The `otherAttnReq` property, which is a derived property defined in terms of `seeksAttn`, 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. From a task achievement point of view, plans in this domain might still achieve the goal of serving drinks to all agents seeking attention. However, 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.

Such a scenario is certainly possible in our bartending domain, where the appearance of a new agent is dynamically reported to the planner by the state manager, possibly triggering a replanning operation: the newly built plan might "preempt" an already waiting agent for a newly-arrived agent as the next customer for the bartender to serve. 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 active research 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) Most current social robots play the role of a companion, often in a long-term relationship with the user, e.g., (Breazeal 2005; Dautenhahn 2007; Castellano et al. 2010). In such a context, the primary goal for the robot is to engage in social interaction for its own sake, and to build a relationship with the user: the robot is primarily an interactive partner, and any task-based behaviour is secondary to this overall goal.

We build on this recent work, but address a different style of interaction, which is distinctive in two main ways. First, while existing projects generally consider social interaction as the primary goal, the robot bartender supports social communication in the context of a cooperative, task-based interaction. Second, while most social robotics systems deal primarily with one-on-one interactive situations, the robot bartender must deal with dynamic, multi-party 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.

Natural language generation and dialogue also have long traditions of using planning. 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).

There has also been a renewed interest 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 actions (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 explored this connection (Steedman and Petrick 2007), but fell short of implementing a tool to leverage this relationship for efficient dialogue planning. A related approach (Brenner and Kruijff-Korbayová 2008) managed dialogues by interleaving planning and execution, but failed to solve the consequent problem of deciding when best to commit to plan execution versus plan construction. Thus, many planning approaches are promising, but not yet fully mature, and fall outside the mainstream of recent 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 off-the-shelf PKS planner, combined with a social state manager and plan monitor. Although this work is preliminary, it has resulted in a working system that has been evaluated with human users. We are currently 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 the bartender or an agent may terminate an interaction early. We believe a general-purpose planning approach offers a potential solution to the problem of action selection in task-based interactive systems, as an alternative to more specialised approaches, such as those used in many mainstream natural language dialogue systems.

Acknowledgements

The authors would like to thank their colleagues from the JAMES consortium 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 Amy Isard 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 Cohen, P.; Morgan, J.; and Pollack, M., eds., *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.; Hanks, S.; Weld, D.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In *Proceedings of KR-1992*, 115–125.

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.

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.

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.

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.

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.

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.