Planning for Social Interaction 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 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 describe an application of planning to task-based social interaction using a robot that must interact with multiple human agents in a simple bartending domain. We show how social states are inferred from low-level sensors, using vision and speech as input modalities, and how we use the knowledge-level PKS planner to construct plans with task, dialogue, and social actions, as an alternative to current mainstream methods of interaction management. The resulting system has been evaluated in a real-world study with human subjects.

Introduction and Motivation

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 its 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. As a result, a robot not only requires the necessary physical skills to perform objective tasks in the world, but also the appropriate *social skills* to understand and respond to the intentions, desires, and affects of the people it interacts with.

However, the problem of building a robot to meet the goals of social interaction presents several challenges for such a system. Not only does the robot require the ability to recognise and interpret appropriate multimodal social signals (e.g., gaze, facial expression, and language), but it must also generate realistic responses using similar modalities. Furthermore, while many interactions may lead to the same goal at the task level, the quality of those interactions may be greatly enhanced by getting the "people skills" right.

To address this challenge, and focus our work, we are investigating the sub-problem of *task-based social interaction* in a bartending domain, by developing a robot bartender (Figure 1) that is capable of dealing with multiple human customers in a drink-ordering scenario. Interactions in this scenario incorporate both task-based aspects (e.g., ordering and delivering drinks) and social aspects (e.g., managing multiple interactions). Moreover, the primary interaction modality in this setting is speech; humans communicate with the robot via speech and the robot must respond in a similar manner. This domain is also appealing since it is an excellent analogue for a range of common interaction and service scenarios in which an assistive robot must detect and respond to the needs of dynamically evolving groups of people in an unstructured, real-world environment.

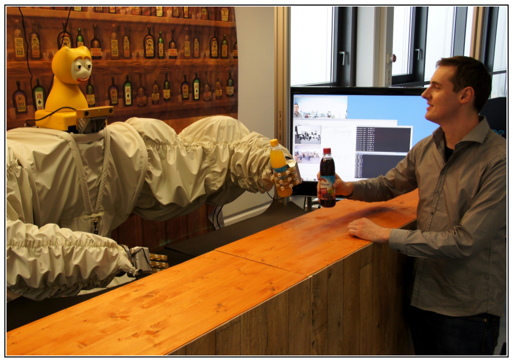

Figure 1: The robot bartender and bar setting.

Since the ability to reason and plan is essential for a cognitive agent operating in a dynamic and incompletely known domain like the bartending scenario, high-level reasoning and action selection is a key component of our robot system. In particular, we use general-purpose planning techniques that can build goal-directed plans under many challenging conditions, especially in task-based contexts. Specifically, we use *knowledge-level planning* and the PKS planner (Petrick and Bacchus 2002; 2004), a choice that is motivated by PKS's ability to work with incomplete information and sensing actions: not only will the robot have to perform physical tasks (e.g., handing a customer a drink), but it will often have to gather information it does not possess from its environment (e.g., asking a customer for a drink order).

Moreover, since interactions will involve human customers, using speech as the main input modality, many of the planner's actions will correspond to speech acts, which introduces a link to natural language processing. Indeed, recent work (Steedman and Petrick 2007; Brenner and Kruijff-Korbayová 2008; Benotti 2008; Koller and Petrick 2011) has investigated the use of planning for natural language generation and dialogue—research fields that have a long tradition of using planning, but where such techniques are no longer the focus of mainstream study.

While planning offers a tool for reasoning and action selection, it is only one component in a larger robot system that must operate in a real-world environment. This introduces some difficulties that must be overcome when the planner interacts with other parts of the system. For instance, the problem of integrating low-level sensor data with symbolic planners introduces representational difficulties that must be addressed: high-level planners typically use representations based on discrete models of objects, properties, and actions, described in logical languages, while many low-level sensors tend to generate continuous streams of low-level, noisy data. Moreover, some aspects of the traditional planning problem, like 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. Furthermore, in an inherently dynamic domain like a busy bar, these sensors may not be able to fully observe the world, or may provide noisy information. Thus, the planner cannot be viewed as simply a black box but must be appropriately situated in the wider cognitive system.

In this paper, we describe the structure of our robot bartender application, with a focus on the role of the high-level planning component, and how it is integrated with the rest of the system. In particular, we discuss how states are derived from sensor observations and how generated plans are executed and monitored in the real world. To demonstrate the operation of our system, we present our drink ordering scenarios and give examples of interesting human-robot interactions in this domain, focusing on the plans that are generated using the PKS planner as an alternative to more mainstream methods of interaction management. Finally, we present the results of an evaluation from a real-world study using the bartender system, and discuss future extensions to this work. This work forms part of a project called JAMES—Joint Action for Multimodal Embodied Social Systems—exploring social interaction with robot systems.[1]

Related Work

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 situ-

[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: An example interaction in the bartending scenario.

ations, 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. In this way, the bartender is similar to the system developed by (Bohus and Horvitz 2009), which also handles situated, open-world, multimodal dialogue in scenarios such as a reception desk and a question-answering game.

The link between planning and language is particularly important to this work, an idea with a long tradition in natural language generation and dialogue. 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 specialised approaches such as finite state machines, information states, speech-act theories, or dialogue games (Traum and Allen 1992; Green and Carberry 1994; Matheson, Poesio, and Traum 2000; Asher and Lascarides 2003; Maudet 2004).

Recently, there has been renewed interest in applying planning methods to natural language problems such as sentence planning (Koller and Stone 2007), instruction giving (Koller and Petrick 2011), and accommodation (Benotti 2008). The idea of treating interaction management as planning with incomplete information and sensing has also been revisited (Stone 2000), a view that is 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 also explored this connection (Steedman and Petrick 2007), but fell short of implementing an efficient tool. A related approach (Brenner and Kruijff-Korbayová 2008) manages dialogues by interleaving planning and execution, but fails to solve the problem of deciding when best to commit to plan execution versus plan construction. Thus, many recent planning approaches are promising, but not yet fully mature, and fall outside of mainstream interactive systems research.

Overview of the Robot Bartender

The target application for this work is a bartending scenario, using the robot platform shown in Figure 1. The robot hardware itself consists of two 6-degrees-of-freedom industrial manipulator arms with grippers, mounted to resemble hu-

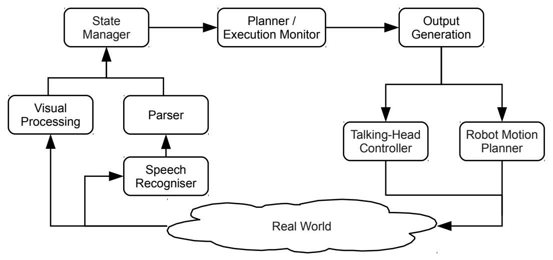

Figure 3: Software architecture of the robot system

man arms. Sitting on the main robot torso is an animatronic talking head capable of producing facial expressions, rigid head motion, and lip-synchronised synthesised speech.

A sample target interaction in the initial version of the bartending scenario is shown in Figure 2. In this example, two customers enter the bar area one at a time and attempt to order a drink from the bartender. Note that when the second customer appears while the bartender is engaged with the first customer, the bartender reacts appropriately by telling the second customer to wait, finishing the transaction with the first customer, and then serving the second customer.

Even this simple interaction presents challenges to all components of the robot system tasked with the role of bartender: the vision system must 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 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 realise high-level plans.

System Architecture and Components

The software architecture of the robot system is shown in Figure 3. We now highlight the main components in the system and the flow of information between these modules.

Input Processing: One of the primary input channels for the robot is computer vision. The vision system tracks the location, facial expressions, gaze behaviour, and body language of all people in the scene in real time. This done by using input from visual sensors to detect and track the faces and hands of agents in the scene, and to extract their 3D position (Baltzakis, Pateraki, and Trahanias 2012). Each agent's focus of attention is also derived using torso orientation. The partially abstracted information is then made available to modules like the state manager for further processing.

The other primary input modality in the system is natural language understanding: we combine a speech recogniser with a natural-language parser to create symbolic representations of the speech produced by all users. For speech recognition, the Microsoft Kinect and the associated Microsoft Speech API is used. For a user's utterance u, the system provides a list of intermediate hypotheses and associated confidence scores $\Pi_u = \{\langle h_1, c_1\rangle, \langle h_2, c_2\rangle, \ldots \langle h_n, c_n\rangle\}$, where $\sum_{i=1}^{n} c_i = 1$, a final best hypothesis h_u^* along with a confidence score c_u^*, and an estimate of the source angle, θ_u^*. We have also created a speech recognition grammar that covers all expected user utterances in the bartending scenario. The resulting grammar constrains the recognition task, producing more reliable results. Currently, only the top hypothesis is used.

Once the user's speech has been recognised, it is processed to extract the underlying meaning. The recognised hypotheses are parsed using an OpenCCG grammar (White 2006), which is an open-source implementation of Combinatory Categorial Grammar (Steedman 2000). The grammar contains syntactic and semantic information, and is used both for parsing the linguistic input, and for surface realisation of the selected output (see below). Once the top speech hypothesis has been parsed, the logical form, confidence measure, and source angle are passed to the state manager.

State Management: The primary role of the state manager is to turn the continuous stream of messages produced by the low-level input components into a discrete representation that combines social and task-based properties. Formally, the low-level input components correspond to a set Σ of sensors, $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, where each sensor σ_i returns an observation $obs(\sigma_{i,t})$ about the world at a time point t. We denote the set of all sensor readings at t by $\Sigma_t = \{obs(\sigma_{1,t}), obs(\sigma_{2,t}), \ldots, obs(\sigma_{n,t})\}$. Some sensors (e.g., the speech recogniser) may also attach a confidence measure to an observation, indicating its estimated reliability, and the fact that real-world sensors are often noisy.

The state representation is based on a set Φ of fluents, $\Phi = \{f_1, f_2, \ldots, f_m\}$: first-order predicates and functions that denote particular qualities of the world, the robot, and other entities in the domain. We denote the value of a fluent f_i at a time point t by $f_{i,t}$, and the set of all fluent values at t by Φ_t. Fluents are updated in a Markovian fashion, where the value of a fluent at time point $t+1$ is a function of the current observations and fluent values at time point t, i.e., $f_{i,t+1} = \Gamma_i(\Sigma_t, \Phi_t)$. Typically, the mapping between sensors and fluents is not one-to-one: a sensor may map to zero, one, or many fluents, as appropriate. A state S_t is then a snapshot of all fluent values at time point t, i.e., $S_t = \Phi_t$.

Intuitively, states represent a point of intersection between low-level sensor data and the high-level structures used by components like the planner. Since states are induced from the mapping of sensor observations to fluent values, the challenge of building an effective state manager rests on defining appropriate mapping functions Γ_i. In the context of social robotics, this is the problem of *social signal processing* (Vinciarelli, Pantic, and Bourlard 2009), which has received an increasing amount of attention in recent years. This process is not always straightforward: often, it is not the sensor data at a single time point that determines the value of a fluent, but rather the patterns found in a sequence of signals. Thus, the value of a fluent might combine information from multiple signals and require temporal cross-modal fusion.

In the bartender robot system, we treat each low-level input component as a set of sensors. The linguistic interpreter corresponds to three sensors: two that observe the parsed content of a user's utterance u and its associated confidence score (the pair $\langle h_u^*, c_u^*\rangle$), and another that returns the estimated angle of the sound source (θ_u^*). The vision system also senses a large number of properties about the agents and objects in the world, including the location, face and torso orientation, and body posture, each of which corresponds to a set of individual sensors, again with confidence scores.

Certain low-level output components are also treated as sensors. For example, the robot arms provide information about the start and end of manipulation actions, together with their success or failure, while the speech synthesiser reports the start and end of all system utterances. Modelling output components as sensors allows information from these sources to be included in the derived state, ensuring the current state of interaction is accurately reflected (e.g., the state of turn taking or whether physical actions succeeded).

The actual fluents modelled in the state are defined by the particular requirements of the scenario: we represent all agents in the scene, their locations, torso orientations, attentional states, and drink requests 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.

The initial 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 a 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 determines when new state reports are published, which helps control turn-taking.

For example, (Huth 2011) found that bar customers used two main signals to show that they wanted attention from the bartender: (1) standing close to the bar, and (2) turning to look at the bartender. We therefore defined the following rule which determines whether an agent a is seeking attention based on those two low-level vision features:

$$\mathit{seeksAttn}(a) = \mathit{loc}(a).z < 30 \wedge abs(\mathit{torsoOri}(a)) < 10°.$$

In other words, a customer is considered to be seeking the bartender's attention if they are less than 30cm from the bar and facing approximately forwards.

To deal with the more complex states required in future versions of the bartender system, we are currently exploring the use of supervised learning classifiers trained on multimodal corpora. In an initial cross-validation study, the trained classifiers significantly outperformed the hand-coded rule (Foster 2013), and the best-performing trained classifier will shortly be evaluated in a user study.

Planning and Execution Monitoring: The high-level planner is responsible for taking state reports from the state manager and choosing actions to be executed on the robot. Plans are generated using PKS (Planning with Knowledge and Sensing) (Petrick and Bacchus 2002; 2004), a conditional planner that works with incomplete information and sensing actions. PKS operates at the *knowledge level* and reasons about how its knowledge state, rather than the world state, changes due to action. To do this, PKS works with a restricted first-order language with limited inference. While features such as functions and run-time variables are supported, these restrictions mean that some types of knowledge (e.g., general disjunctive information) cannot be modelled.

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 is represented by a set of five databases, each of which models a particular type of knowledge, and can be understood in terms of a modal logic of knowledge. Actions can modify any of the databases, which updates the planner's knowledge state. To ensure efficient inference, PKS restricts the type of knowledge (especially disjunctions) 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 to model action effects that change the world. K_f can include any ground literal or function (in)equality mapping ℓ, where $\ell \in K_f$ means "the planner knows ℓ."

$\mathbf{K_w}$: 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 run time this disjunction will be resolved. PKS uses such information to include conditional branches in a plan, where each branch assumes one of the possible outcomes is true.

$\mathbf{K_v}$: 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 can use K_v terms as run-time variables in its plans, and can build conditional plan branches when the set of possible mappings for a function is restricted.

$\mathbf{K_x}$: This database models the planner's "exclusive-or" knowledge. 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 type of disjunctive knowledge common in planning domains, namely that "exactly one of the ℓ_i is true."

(A fifth database called *LCW* is not used in this work.)

Simple questions about PKS's knowledge state can be answered using a set of *primitive queries*: whether facts are (not) known to be true (a query $[\neg]K(\phi)$), 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 procedure evaluates the queries by checking the contents of the various databases.

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

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.

In addition to the main planner, 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 provided by the state manager, denoting the sensed state. 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 action (or set of actions) in the plan to be executed, by testing an action's preconditions against the current set of (sensed) state properties. In the case of a mismatch, the planner is directed to build a new plan, using the sensed state as its initial state.

Output Generation: Output in the system is based on dividing actions selected by the planner into speech, head motions, and arm manipulation behaviours that can be executed by the robot. To do so, we use a structure containing specifications for each of the output modalities (Isard and Matheson 2012), based on a rule-based approach which splits each planned action into its component subparts. The resulting structure is then passed to the multimodal output generator, which sends specific commands to each output channel.

On the linguistic side, we use OpenCCG to generate the robot language output, using the same OpenCCG grammar used for input. The language output 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 are turned into speech by the robot's animatronic head. In addition to speech, the robot system also expresses itself through facial expressions, gaze, and arm manipulation actions. The animatronic head can currently express a number of pre-assigned expressions, while the robot arm can perform tasks like grasping objects to hand over a drink to a customer; motion planning and robot control make use of the Robotics Library (Rickert 2011).

The multimodal behaviours are coordinated across the various output channels to ensure that they are synchronised both temporally and spatially. For instance, a high-level action to serve an agent a drink is transformed into a specification that results in the robot smiling (an animatronic head facial expression) while physically handing over the ordered drink (a robot arm manipulation action) and saying to the customer "here is your drink" (speech output).

```
action greet(?a : agent)
    preconds:  K(inTrans = nil) $\wedge$ $\neg$K(greeted(?a)) $\wedge$
               K(seeksAttn(?a)) $\wedge$ $\neg$K(ordered(?a)) $\wedge$
               $\neg$K(otherAttnReq) $\wedge$ $\neg$K(badASR(?a))
    effects:   add($K_f$, greeted(?a)),
               add($K_f$, inTrans = ?a)

action ask-drink(?a : agent)
    preconds:  K(inTrans = ?a) $\wedge$ $\neg$K(ordered(?a)) $\wedge$
               $\neg$K(otherAttnReq) $\wedge$ $\neg$K(badASR(?a))
    effects:   add($K_f$, ordered(?a)),
               add($K_v$, request(?a))

action ack-order(?a : agent)
    preconds:  K(inTrans = ?a) $\wedge$ K(ordered(?a)) $\wedge$
               $\neg$K(otherAttnReq) $\wedge$ $\neg$K(badASR(?a)) $\wedge$
               $\neg$K(ackOrder(?a))
    effects:   add($K_f$, ackOrder(?a))

action serve(?a : agent, ?d : drink)
    preconds:  K(inTrans = ?a) $\wedge$ K(ordered(?a)) $\wedge$
               $K_v$(request(?a)) $\wedge$ K(request(?a) = ?d) $\wedge$
               $\neg$K(otherAttnReq) $\wedge$ $\neg$K(badASR(?a)) $\wedge$
               K(ackOrder(?a))
    effects:   add($K_f$, served(?a))

action bye(?a : agent)
    preconds:  K(inTrans = ?a) $\wedge$ K(served(?a)) $\wedge$
               $\neg$K(otherAttnReq) $\wedge$ $\neg$K(badASR(?a))
    effects:   add($K_f$, transEnd(?a)),
               add($K_f$, inTrans = nil)
```

Figure 4: PKS actions in a single agent interaction

System Integration

Like most interactive multimodal systems, the robot bartender is made up of a number of distributed, heterogeneous software components, developed by different research groups and drawing on diverse research paradigms, each with individual hardware and software requirements. These components must all communicate with one another to support interactions with customers in the bartender scenario. In particular, the planner must be situated in this system and make use of the same interfaces as other components.

For inter-module communication in the robot bartender, we use the Ice object middleware (Henning 2004), which provides platform- and language-independent communication among the modules and supports direct module-to-module communication as well as publish-subscribe messaging. This framework has been recently used in multimodal interactive systems such as (Green et al. 2010; Dzikovska et al. 2011), and is also the underlying communication layer for the Orca robot communication framework (Makarenko, Brooks, and Kaupp 2006).

On the planning side, adapting the off-the-shelf PKS planner for use with Ice involved creating a communication-level API to common planning features, and re-engineering the backend planning code into a suitable library that supported this interface. Common operations like planner configuration, domain definition, and plan construction were abstracted into a class definition that allowed a PKS planner instance to be created as a C++ object. The interface to this library was built into a simple server which provided a transparent network interface to its functions over Ice.

Robot Actions in the Bartending Domain

All of the robot's high-level actions in the bartending scenario (task and social) are modelled as part of a simple PKS planning domain, rather than using specialised tools as is common practice in many modern dialogue systems. As a result, our planning domain must capture the necessary properties and actions of the world, agents, and objects.

Domain properties are based on fluents defined in the state manager. In particular, predicates in the domain include:

- *seeksAttn(?a)*: agent *?a* seeks attention,
- *greeted(?a)*: agent *?a* has been greeted,
- *ordered(?a)*: agent *?a* has ordered,
- *ackOrder(?a)*: agent *?a*'s order has been acknowledged,
- *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 domain are defined using the above fluents. In particular, our domain includes eight high-level actions:

- *greet(?a)*: greet an agent *?a*,
- *ask-drink(?a)*: ask agent *?a* for a drink order,
- *ack-order(?a)*: acknowledge agent *?a*'s 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, and
- *wait(?a)*: tell agent *?a* to wait, and
- *ack-wait(?a)*: thank agent *?a* for waiting.

Definitions for the first five 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 PKS'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*, *ack-order*, and *bye* are modelled in a similar way, but only map to speech output at run time (e.g., "hello", "okay", and "goodbye"). The most interesting action is *ask-drink* which is modelled as a sensing action: the function term *request* is added to the planner's K_v database as an effect, indicating that this piece of information will become known at execution time. 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 control interactions when multiple agents are seeking the attention of the bartender.

Finally, we also require a description of the domain's initial state and goal before we can build plans. 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 agent.The planner's goal is simply to serve each agent seeking attention, i.e.,

$$\textit{forallK(?a : agent)}\ K(\textit{seeksAttn(?a)}) \Rightarrow K(\textit{transEnd(?a)}).$$

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

Planning Interactions for Social Behaviour

When agents are detected in the scene by the vision system, and reported to the planner through the state manager, the planner responds by generating a plan to direct the robot's behaviour. Even though our domain definition is simple, it nevertheless results in some interesting (and appropriate) behaviour in many common interaction scenarios.

Ordering a Drink: We first 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 state manager informs PKS that a new agent *a1* has appeared and is seeking attention; this 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]
ack-order(a1),	[Acknowledge *a1*'s drink order]
serve(a1, request(a1)),	[Give the drink to *a1*]
bye(a1).	[End the transaction]

Initially, the planner can choose *greet(a1)* 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)* $\in K_v$. The next action considered by the planner is *ack-order(a1)*, since *ackOrder(a1)* $\notin K_f$. As a result *ackOrder(a1)* is added to K_f. 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, $K(\mathit{ordered(a1)})$, holds. Also, since *request(a1)* $\in K_v$, the third precondition $K_v(\mathit{request(a1)})$ holds (i.e., the value of *request(a1)* is known). The fourth precondition, $K(\mathit{request(a1)} = \mathit{request(a1)})$ 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. Thus, *request(a1)* acts as a run-time variable whose definite value (i.e., *a1*'s drink order) will become known at run time. The fifth precondition is satisfied by the effects of *ack-order(a1)*. The remaining two preconditions are also trivially satisfied. 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 Drink Choices: The above plan relies on PKS's ability to use 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, again with a single agent *a1* seeking attention, where PKS is also told there are three possible drinks that can be ordered: juice, water, and beer. In this case, the drinks are represented as constants of type *drink*, i.e., *juice*, *water*, and *beer*. Information about the drink options is also put into PKS's K_x database as the formula:

$$\mathit{request(a1) = juice} \mid \mathit{request(a1) = water} \mid \mathit{request(a1) = beer}$$

which restricts the 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]
ack-order(a1),	[Acknowledge *a1*'s 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 with branches for each possible mapping of *request(a1)*. 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 using additional information gathering actions (i.e., "would you like ice in your water?").

Ordering Drinks with Multiple Agents: Our 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]
ack-order(a1),	[Acknowledge *a1*'s 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]
ack-order(a1),	[Acknowledge *a2*'s drink order]
serve(a2, request(a2)),	[Give the drink to *a2*]
bye(a2).	[End *a2*'s transaction]

Thus, *a1*'s drink order is taken and processed, followed by *a2*'s order. The *wait* and *ack-wait* actions (which aren't needed in the single agent plan) are used to defer a transaction with *a2* until *a1*'s transaction 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 of our domain encoding is that agents who are asked to wait are not necessarily served in the order they are deferred. From a task achievement point of view, such plans still succeed in their goal of serving drinks to all agents. 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 others that have been waiting for longer periods of time. This situation can arise in our domain since the appearance of a new agent is dynamically reported to the planner, possibly triggering a replanning operation: the newly built plan might preempt a waiting agent for a newly-arrived agent as the next customer for the bartender to serve. Since socially appropriate interaction is central to this work, we are modifying our domain to include ordering constraints on waiting agents.

Low-Confidence Utterances and Overanswering: Once a plan is built, it is executed by the robot, one action at a time. Each action is divided into head, speech, and arm behaviours based on a simple set of rules, before it is executed in the real world. At run time, PKS's execution monitor assesses 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 robot.

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 automatic 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 in order to continue executing the previous plan.

Another consequence of this approach is that certain types of overanswering can be handled through execution monitoring and replanning. For instance, a *greet(a1)* action by the robot 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 its state report, along with *ordered(a1)*. The monitor would detect that the preconditions of *ask-drink(a1)* aren't met and direct PKS to replan. A new plan could then omit *ask-drink* and proceed to acknowledge and serve the requested drink.

Evaluation

We have tested our basic bartending domain in a user evaluation in which 31 naïve subjects enacted variations on the drink-ordering scenario shown in Figure 2 and then answered a short questionnaire. Overall, most customers (95%) successfully ordered a drink from the bartender, and the robot dealt appropriately with multiple simultaneous customers and with unexpected situations of the form described above (i.e., overanswering and input-processing failure). The factors that had the greatest influence on subjective user satisfaction were task success and dialogue efficiency: that is, the users gave the system higher scores when a drink was served (a) successfully and (b) quickly. Full details of this user study are presented in (Foster et al. 2012).

In terms of traditional planning measures, plan generation time was typically quite short (usually less than 0.1s), which did not negatively impact the reaction time of the overall system. The quality of most plans was initially quite high (as in the example scenarios described above); however, input processing failures and unexpected responses from human users in some situations resulted in replanning, leading to longer overall plans. In particular, most drink-ordering transactions included at least one attempted user turn that fell below the ASR confidence threshold, while the number of system turns in a complete transaction ranged from 3 (a maximally efficient transaction, including successful overanswering from the customer) to 22 (indicating severe input-processing problems), with a mean of 5.8.

Discussion and Future Work

While our planning domain models many realistic situations, we are also extending our approach to manage more complex scenarios that arise in dialogue-based interaction.

First, we are improving our ability to plan under uncertainty due to automatic speech recognition (ASR). Currently, any speech input hypotheses other than the top hypothesis (h_u^*, c_u^*) is discarded by the natural language processing module. As a result, potentially high-likelihood alternatives to the top hypothesis are unavailable to the planner, raising the possibility of less effective (or incorrect) action choices by the planner during plan construction. Instead, we are exploring the idea of passing an n-best list of processed hypotheses to the state manager for inclusion in the state representation, as a set of alternative interpretations for an agent's utterance. In practical terms, the n-best can be determined by the list of top entries that account for a significant probability mass in terms of the hypotheses' associated confidence measures, i.e., $\{\langle h_1, c_1\rangle, \langle h_2, c_2\rangle, \ldots \langle h_n, c_n\rangle\}$, such that $\sum_{i=1}^{n} c_i > \theta$, where θ is some threshold. Using this list, the state manager can then derive a set of interpretations, $\{\phi_1, \phi_2, \ldots, \phi_n\}$, where each ϕ_i is a conjunction of fluents. (In our domain, each ϕ_i is usually a single fluent.)

At the planning level, such disjunctive state information can be represented in PKS's K_x database as an "exclusive or" formula of the form $(\phi_1|\phi_2|\ldots|\phi_n)$. Once such information is available in planner's knowledge state, it can be directly used during plan construction. In practice, such knowledge often has the effect of introducing additional actions into a plan, to disambiguate between K_x alternatives. To aid in this process, we are adding new domain actions which correspond to information-gathering questions that the bartender can ask to help clarify uncertain beliefs (without asking an agent to simply repeat an utterance, which is often interpreted by humans as a poor dialogue move).

Second, we are also extending PKS's ability to work with certain types of multiagent knowledge that arise in dialogue scenarios (Steedman and Petrick 2007). For instance, *common ground* information (i.e., the beliefs about the conversational state that arise during an interaction) is often used by mainstream dialogue systems but is not present in our domain. Currently, PKS cannot easily model other agents' beliefs; rather, such information must be encoded using the standard (single agent) tools available in PKS. We are adding new operators to PKS that enable it to directly represent and reason with such knowledge during planning. This is particularly important in dialogue contexts, since more effective dialogue moves can often be made by taking into consideration what other agents believe about an interaction.

Conclusions

We have presented an application of knowledge-level planning to task-based social interaction in a robot bartender domain. Actions are selected by the PKS planner, using states derived from low-level visual and speech inputs. Plans are executed on a robot platform to control an animatronic head, speech synthesiser, and a pair of manipulator arms.

While our current planning approach works well for our simple bartending scenario, we are currently extending this domain to support more complex interactions, including agents that can ask questions about drinks, a bartender that can query agents for more information, and groups of agents that can order multiple drinks. All of these extensions will require a more active role for the planner. However, based on the results of our initial evaluation, we believe that general-purpose planning continues to offer a promising tool for action selection in task-based interactive systems, as an alternative to more specialised approaches, such as those used in many interactive dialogue systems.

Acknowledgements

The authors thank their JAMES colleagues who helped implement the bartender system: Andre Gaschler, Manuel Giuliani, Amy Isard, Maria Pateraki, and Richard Tobin. This research has received funding from the European Union's 7th Framework Programme under grant No. 270435.

References

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

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

Baltzakis, H.; Pateraki, M.; and Trahanias, P. 2012. Visual tracking of hands, faces and facial features of multiple persons. *Machine Vision and Applications* 1–17.

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

Bohus, D., and Horvitz, E. 2009. Dialog in the open world: platform and applications. In *Proceedings of ICMI-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 SemDial-2008 (LONDIAL)*, 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.

Dzikovska, M.; Isard, A.; Bell, P.; Moore, J.; Steinhauser, N.; and Campbell, G. 2011. BEETLE II: an adaptable tutorial dialogue system. In *Proceedings of SIGDIAL 2011*, 338–340.

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 *Proceedings of ICMI-2012*.

Foster, M. E. 2013. Automatically classifying customer engagement for a robot bartender. Submitted.

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-1994*, 58–65.

Green, S.; Chase, G.; Chen, X.; and Billinghurst, M. 2010. Evaluating the Augmented Reality Human-Robot Collaboration System. *International Journal of Intelligent Systems Technologies and Applications* 8(1-4).

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

Henning, M. 2004. A new approach to object-oriented middleware. *IEEE Internet Computing* 8(1):66–75.

Huth, K. 2011. Wie man ein Bier bestellt. MA thesis, Fakultät für Linguistik und Literaturwissenschaft, Universität Bielefeld.

Isard, A., and Matheson, C. 2012. Rhetorical structure for natural language generation in dialogue. In *Proceedings of SemDial-2012 (SeineDial)*, 161–162.

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 ACL-2007*, 336–343.

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

Makarenko, A.; Brooks, A.; and Kaupp, T. 2006. Orca: Components for robotics. In *Proceedings of the IROS 2006 Robotic Standardization Workshop*.

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.

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 *Proceedings of ICAPS-2004*, 2–11.

Rickert, M. 2011. *Efficient Motion Planning for Intuitive Task Execution in Modular Manipulation Systems*. Dissertation, Technische Universität München.

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-1992*, 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 International Workshop on Natural Language Generation*, 13–20.