Plan-Based Social Interaction with a Robot Bartender

Plan-Based Social Interaction with a Robot Bartender

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. 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. The resulting system infers social states from low-level sensors, using vision and speech as input modalities, and uses the knowledge-level PKS planner to construct plans with task, dialogue, and social actions.

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 affective states of its interaction partners. To address this challenge, we are investigating *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.

Key to our approach is the use of high-level planning techniques, which are responsible for action selection and reasoning in the robot system. Specifically, we use the knowledge-level planner PKS (Petrick and Bacchus 2002; 2004), a choice that is motivated by PKS's ability to work with incomplete information and sensing actions: not only must the robot perform physical tasks (e.g., handing a customer a drink), 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, speech will be the main input modality and many of the planner's actions will correspond to speech acts, providing a link to natural language processing—a research field with a long tradition of using planning, but where general-purpose planning techniques are not the focus of mainstream study.

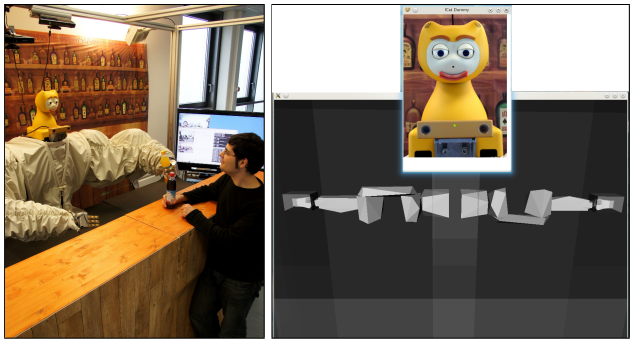

Figure 1: The JAMES robot bartender

While planning offers a tool for action selection, it is only one component in a larger system that operates in a real-world environment. A second, central component in our system is the state manager which mediates between the low-level input sensors and the planner, and which overcomes some of the representational difficulties involved in bridging the gap between continuous, low-level input streams and symbolic, high-level state-based reasoning.

In the rest of the paper we give a technical description of the robot system, with a focus on the role of the planner and how it is integrated in this framework. The application of this work is a simple bartending scenario, which is modelled as a PKS planning domain. More details on this work can be found in (Petrick and Foster 2013). This work forms part of a project called JAMES (Joint Action for Multimodal Embodied Social Systems; see james-project.eu).

Robot System Architecture and Components

The target application for this work is a bartending scenario, using the robot platforms shown in Figure 1. The robot hardware itself (Figure 1) consists of two 6-degrees-of-freedom industrial manipulator arms with grippers, mounted to resemble human 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. For testing and demonstration purposes, the simulated robot shown in Figure 1 is also available.

A sample interaction in a simple bartending scenario is shown in Figure 2. In this example, two customers enter the bar and attempt to order a drink from the bartender. When

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.

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 current transaction, and then serving the second customer.

Even this simple interaction presents challenges which have motivated the design of the overall system: a vision system must track the locations and body postures of the agents; a speech-recognition system must detect and deal with speech in an open setting; reasoning components must determine that both customers require attention and ensure they are served in the correct order; while the output components must select and execute concrete actions for each output channel that correctly realise high-level plans. The software architecture of the robot system is shown in Figure 3, with the main components highlighted below.

Input Processing: One of the primary input channels for the robot is computer vision. The full JAMES vision system tracks the location, facial expressions, gaze behaviour, and body language of all people in the scene in real time, using a set of visual sensors (Baltzakis, Pateraki, and Trahanias 2012); a limited-functionality vision system is also available that can run on a single Kinect for demo and testing purposes. Information from the vision system is constantly published to the state manager multiple times a second.

The other primary input modality in the system is linguistic, combining a speech recogniser with a natural-language parser to create symbolic representations of the speech produced by all users. For speech recognition, we use the Microsoft Kinect and the Microsoft Speech API, with a scenario-specific speech grammar to constrain the recognition task. Recognised speech is then parsed using a grammar implemented in OpenCCG (White 2006); the grammar contains syntactic and semantic information, and is used both for parsing the spoken input and for surface realisation of the selected output (see below). The parsed speech, confidence score, 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. The state representation is based on a set of *fluents*: first-order predicates and functions that denote particular qualities of the world, the robot, and other entities in the domain. A state is a snapshot of all fluent values at a given point in time. Intuitively, states represent a point of intersection between

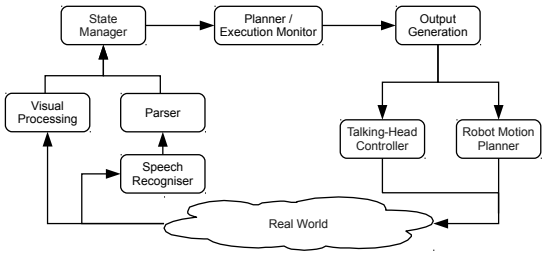

Figure 3: Software architecture of the robot system

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.

In the bartender robot, 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 and its associated confidence score, and another that returns the estimated angle of the sound source. The vision system also senses a large number of properties about the agents and objects in the world, each of which corresponds to a set of individual sensors. 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, 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 the completion of physical actions).

In the current robot bartender system, the state includes information about all agents in the scene: their locations, torso orientations, attentional states, and drink requests if they have made one. The mapping from sensors to states is rule-based. One set of rules infers user social states (e.g., seeking attention) from the low-level sensor data, using guidelines derived from a study of natural bartender interactions (Huth 2011). The state manager also incorporates rules that convert the logical forms produced by the parser into communicative acts (e.g., drink orders), and that use the source angle 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 controls turn-taking.

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 study, the trained classifiers significantly outperformed the hand-coded rules both in cross-validation and when tested with real users (Foster 2013).

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 representation 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. To ensure efficient inference, PKS restricts the type of knowledge it can represent to a set of four databases:

$\mathbf{K_f}$: 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 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 build conditional branches into a plan.

$\mathbf{K_v}$: This database stores functions whose values will become known at run time. In particular, K_v can model the plan-time effects of sensing actions that return terms. K_v can contain any unnested function, where $f \in K_v$ means that at plan time the planner "knows the value of f."

$\mathbf{K_x}$: 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 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 a list of simple questions about PKS's knowledge state (e.g., a query $K(\phi)$ asks if ϕ is known). Effects are described by a set of STRIPS-style "add" and "delete" operations that modify the contents of individual databases. E.g., *add*(K_f, ϕ) adds ϕ to the K_f database, while *del*(K_w, ϕ) removes ϕ from K_w. PKS constructs plans by reasoning about actions in a simple forward-chaining manner, and can build plans with branches be considering the possible outcomes of its K_w and K_v knowledge. *Goals* are specified in a form similar to action preconditions.

PKS is also aided by an execution monitor which controls replanning. The monitor takes as input a PKS plan, and a description of the sensed state provided by the state manager. The monitor must assess how close an expected, planned state is to a sensed state in order to determine whether the current 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.

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 *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)*)

Figure 4: Example PKS actions in the bartender domain

OpenCCG is used to generate speech output for the robot, using the same grammar that is used to parse the input. The output description is specified in terms of high-level communicative acts, which are translated into logical forms and sent to the OpenCCG realiser. The realiser then outputs text strings that are turned into speech by the robot's animatronic head. In addition to speech, the robot also expresses itself through facial expressions, gaze, and arm manipulation actions. The animatronic head can produce a number of expressions and can gaze at customers or objects, while the robot arm can perform tasks like grasping to hand over a drink to a customer; motion planning and robot control make use of the Robotics Library (Rickert 2011).

System Integration: Like most interactive multimodal systems, the robot bartender is made up of a number of distributed, heterogeneous software components, drawing on diverse research paradigms, each with individual hardware and software requirements. These components must all communicate with one another to support interactions in the bartender scenario. The planner must also be situated in this system and use 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. On the planning side, adapting the off-the-shelf PKS planner for use with Ice is achieved by creating a communication-level API to common planning features, and re-engineering the backend planner 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.

Planning Interactions for Social Behaviour

The robot's available high-level actions are modelled as part of a PKS planning domain, rather than using specialised tools as is common in many dialogue systems. For instance,

the basic bartender domain consists of the following actions, available to the robot for interacting with human customers:

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)	inform agent *?a* was not understood,
wait(?a)	tell agent *?a* to wait, and
ack-wait(?a)	thank agent *?a* for waiting.

Actions model high-level robot behaviours that include a mix of physical, sensory, and speech acts. Examples of two PKS actions in the bartender domain are shown in Figure 4.

Information about human agents is not hard-coded in the domain but is detected by the vision system and passed to the planner by the state manager through its state updates. Similarly, changes to the agent list are also sent to the planner in state reports, causing it to update its domain model. The goal is simply to serve each agent seeking attention. This goal is viewed as a rolling target which is reassessed each time a state report is received by the planner. For instance, if two agents (*a1* and *a2*) are seeking attention, PKS can build the following plan (similar to the interaction in Figure 2):

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]

Here, *a1*'s drink order is taken and processed, followed by *a2*'s order. The *ask-drink* action is a sensing action that returns information about the term *request* (an agent's drink order), which is then used as a run-time variable in the *serve* action. The *wait* and *ack-wait* actions are used to defer a transaction with *a2* until *a1*'s transaction has finished.

Once a plan is built, it is executed by converting each action into its head, speech, and arm behaviours, based on a simple set of rules. Execution is monitored for plan correctness by comparing states from the state manager against states predicted by the planner. In the case of divergence, the planner is directed to construct a new plan using the sensed state as its new initial state. For example, if *a1*'s response to *ask-drink(a1)* was not understood, the execution monitor will direct PKS to build a new plan. One result is a modified plan that first informs *a1* they were not understood before repeating the *ask-drink* action and continuing the old plan.

Another consequence of execution monitoring is that certain types of overanswering can be detected and handled through 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 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.

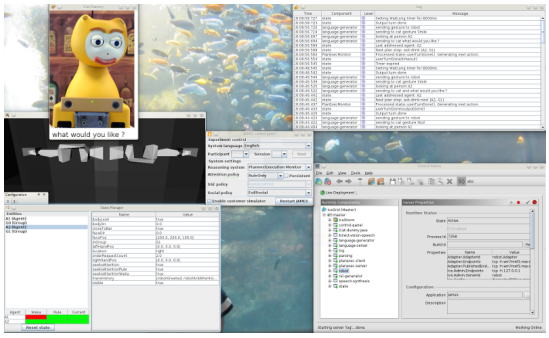

Figure 5: The JAMES software interface

The complete bartender system uses the physical or simulated robot to process interactions similar to those shown above. Users interact with the system using speech, while the main system interface (Figure 5) displays the reasoning and execution status of the core components, including planning and state management.

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 number 270435.

References

Baltzakis, H.; Pateraki, M.; and Trahanias, P. 2012. Visual tracking of hands, faces and facial features of multiple persons. *Machine Vision and Applications* 23(6):1141–1157.

Foster, M. E. 2013. Evaluating engagement classifiers for a robot bartender. In submission.

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.

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.

Petrick, R. P. A., and Foster, M. E. 2013. Planning for social interaction in a robot bartender domain. In *Proceedings of ICAPS 2013, Special Track on Novel Applications*.

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

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