# Knowledge Engineering and Planning for Social Human-Robot Interaction: A Case Study

Ronald P. A. Petrick and Mary Ellen Foster

**Abstract** The core task of automated planning is goal-directed action selection; this task is not unique to the planning community, but is also relevant to numerous other research areas within AI. One such area is interactive systems, where a fundamental component called the interaction manager selects actions in the context of conversing with humans using natural language. Although this has obvious parallels to automated planning, using a planner to address the interaction management task relies on appropriate engineering of the underlying planning domain and planning problem to capture the necessary dynamics of the world, the agents involved, their actions, and their knowledge. In this chapter, we describe work on using domain-independent automated planning for action section in social human-robot interaction, focusing on work from the JAMES (Joint Action for Multimodal Embodied Social Systems) robot bartender project.

## 1 Introduction

At a high level, automated planning can be viewed as a problem of context-dependent action selection: given a set of initial state conditions, action descriptions, and goals, the planner must generate a sequence of actions whose application to the initial state will bring about the goal conditions. However, this view of action selection is not unique to planning. One important area where this problem is also of primary concern is in **interactive systems**, a subfield of natural language dialogue that is focused on implementing tools and applications for interacting with human users.

Ronald P. A. Petrick
Department of Computer Science, Heriot-Watt University, Edinburgh, UK
e-mail: `R.Petrick@hw.ac.uk`

Mary Ellen Foster
School of Computing Science, University of Glasgow, Glasgow, UK
e-mail: `MaryEllen.Foster@glasgow.ac.uk`

A fundamental component in the construction of an interactive system, such as a robot that is able to converse with a human using natural language, is the **interaction manager** (Bui, 2006), whose primary task is to carry out a form of action selection: based on the current state of the interaction and of the world, the interaction manager makes a high-level decision as to which spoken, non-verbal, and task-based actions should be taken next by the system as a whole. Compared with more formal, descriptive accounts of dialogue which aim to model the full generality of language use (Asher and Lascarides, 2003), work on interaction management has concentrated primarily on developing end-to-end systems that operate in specific task settings, and on evaluating them through interaction with human users (Jokinen and McTear, 2009; McTear et al., 2016).

In contrast, the planning community has addressed the problem of high-level action selection through the development of domain-independent planners: systems that employ general-purpose problem-solving techniques that can be applied to a wide range of planning domains and problems, modelled in common representation language such as PDDL (McDermott et al., 1998). Action selection strategies are regularly compared within this common context, especially through events like the International Planning Competitions (ICAPS, 2019), while the representation languages themselves are often studied to better understand their expressiveness and applicability (Rintanen, 2004). Applying planning tools to a complex scenario therefore involves appropriate engineering of the underlying planning domain and planning problem, to capture the necessary dynamics of the world, the agents involved, their actions, and their knowledge, for a suitable choice of planning system and representation language—and then often integrated as part of a larger system.

While the link between automated planning and natural language processing has a long tradition, the planning approach to natural language interaction has for the most part been largely overlooked more recently. In this chapter, we describe work on using domain-independent automated planning for action section in human-robot interaction, using an application from the JAMES (Joint Action for Multimodal Embodied Social Systems)[1] robot bartender project (Petrick and Foster, 2013). We survey recent work in the interactive systems community in the form of toolkits used for constructing interactive dialogue systems. We then describe how we use knowledge engineering techniques to perform similar tasks with an epistemic automated planning system. In the specific context of the JAMES robot system, we show how social states are inferred from low-level sensors, using vision and speech as input modalities; how planning domains and problems are modelled for the bartending scenario; and how an epistemic planner is used to construct plans with task, dialogue, and social actions, as an alternative to other methods of interaction management.

[1] `http://james-project.eu/`

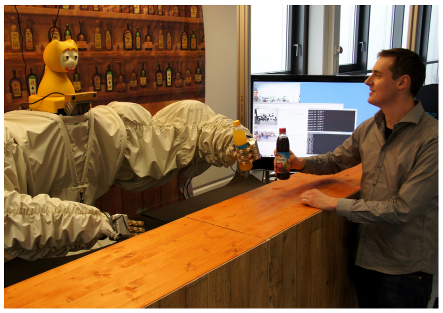
## 2 Interaction Management

Since both interaction management and automated planning deal with goal-directed action selection, in principle interaction management presents an opportunity for showcasing planning tools and demonstrating how different approaches can be applied, benchmarked, and compared. Although early applications of planning in this area can be traced back to the 1980s (Perrault and Allen, 1980; Appelt, 1985; Hovy, 1988; Cohen and Levesque, 1990; Young and Moore, 1994), the planning approach has for the most part been largely overlooked more recently. Instead, interactive systems researchers tend to use purpose-built toolkits for constructing end-to-end dialogue systems. Foster and Petrick (2017) present a survey of such toolkits; we summarise the main features of some of these toolkits below.

An interaction management toolkit generally incorporates three main features. First, it provides a representational formalism for specifying states and actions. Second, the state/action representation is usually tightly linked to a reasoning strategy that is used to carry out action selection. Finally, most toolkits also include a set of infrastructure building tools designed to support modular system development. While these three features can clearly simplify the task of implementing an individual end-to-end system, the fact that the features are so tightly connected does complicate the task of comparing representational formalisms or reasoning strategies: in general, to carry out such a comparison, there is no alternative but to re-implement the entire system in multiple frameworks (Peltason and Wrede, 2011; Olaso et al., 2016).

Historically, one of the most widely used approaches to dialogue management was the Information State Update (ISU) approach, which is exemplified by the TrindiKit toolkit (Larsson and Traum, 2000). The core of this approach is the use of an information state which represents the state of the dialogue and which is updated by applying update rules following a given update strategy. A similar ISU approach has also been taken in more recent dialogue systems, but using other infrastructures (Johnston et al., 2002; Janarthanam et al., 2015). A more recent approach is exemplified by OpenDial (Lison, 2015), an open-source toolkit designed to support robust dialogue management, using a hybrid framework that combines logical and statistical approaches through probabilistic rules to represent the internal models of the framework. OpenDial also includes a Java-based blackboard architecture where all modules are connected to a central information hub which represents the dialogue state, along with a plugin framework allowing new modules to be integrated.

Many modern interactive systems are built with online toolkits such as the Amazon Alexa Skills Kit (Amazon, 2020) or DialogFlow (Google, 2020)—these toolkits generally use machine learning to learn the correct responses to user actions given sample inputs. One current interactive system which does incorporate aspects of automated planning is the MuMMER social robot Papaioannou et al. (2018), which combines a planner used for action selection with a more traditional dialogue manager. Other approaches (Koller and Stone, 2007; Benotti, 2008; Brenner and Kruijff-Korbayová, 2008) have also explored the use of planning for dialogue and interaction, while recent work on explainable planning (Fox et al., 2017) has also highlighted the links between planning and user interaction.



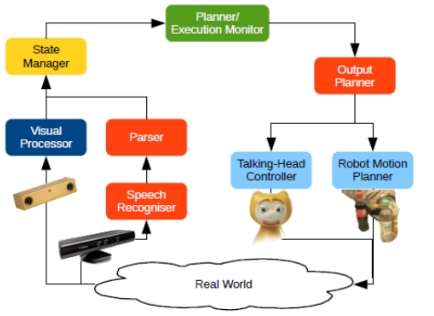


**Fig. 1** The robot bartender and bar setting (left) and the software architecture for the robot (right).

## 3 Task-Based Social Interaction: A Robot Bartender Scenario

The goal of this work is to use domain-independent planning as the high-level decision-making mechanism for action selection in an interactive robot system. In particular, the target domain for this work is a task-based human-robot interaction scenario involving a bartending robot, as shown in Figure 1 (left). In this setting, the robot acts as a bartender that serves customers that approach the bar area seeking attention. The robot hardware itself 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.

A sample bartender interaction is shown in Figure 2. In this example, two people enter the bar area and attempt to order drinks from the robot. During the interaction, a third person approaches the bar and attempts to attract the attention of the bartender. When the third customer appears while the bartender is engaged with the first two customers, the bartender reacts appropriately by telling the third customer to wait, finishing the transaction with the first two people, and then serving the third customer.

Even this simple interaction presents challenges to the robot system tasked with the role of the bartender (see Figure 1, right): the **visual processor** system must track the locations and body postures of the agents; the **speech recogniser** must detect and deal with speech in an open setting and, using the **parser**, transform the input into a logical form representing the speech; the **state manager** must make sense of the social scene using the processed input modalities; the **planner** and **execution monitor** must determine who requires attention and should ensure that they are served correctly, while appropriately responding to unexpected outcomes as they arise; while the **output planner** must select and execute concrete behaviours for each output channel to correctly realise high-level actions, communicating with the **talking-head controller** and **robot motion planner**.

From a high-level planning perspective, the task of applying planning in this scenario centres around the knowledge engineering task of accurately modelling the states, actions, and goals that reflect the types of activities the robot is expected

| **Interaction** | | **Action type** |
|---|---|---|
| *Two people, A and B, each individually approach and look at the robot bartender* | | |
| ROBOT: | [Looks at Person A] How can I help you? | Sensing action |
| PERSON A: | A pint of cider, please. | |
| *Person C approaches the bar and tries to attract the attention of the bartender* | | |
| ROBOT: | [Looks at Person C] One moment, please. | Social action |
| ROBOT: | [Serves Person A] | Physical action |
| ROBOT: | [Looks at Person B] What will you have? | Sensing action |
| PERSON B: | A glass of red wine. | |
| ROBOT: | [Serves Person B] | Physical action |
| ROBOT: | [Looks at Person C] Thanks for waiting. | Social action |
| | How can I help you? | Sensing action |
| PERSON C: | I'd like a pint of beer. | |
| ROBOT: | [Serves Person C] | Physical action |

**Fig. 2** An example interaction in the robot bartending scenario.

to perform. Considering the sample interaction, this includes a mixture of physical actions in the underlying task domain (e.g., serving the actual drinks), sensing actions that acquire new information (e.g., asking a customer for a drink order), and social actions that help facilitate the interactive context (e.g., thanking a customer). As a result, we also require a suitably expressive representation that enables such actions to co-exist within a planning domain. This task is further complicated by the fact that the planner is a single component situated in a much larger architecture, with the representation of states, actions, and goals having connections to the input and output modalities processed by other system components.

## 4 Modelling Social Human-Robot Interaction for Planning

In this section, we describe how planning techniques are applied to the problem of social human-robot interaction in the robot bartender scenario by considering how states, actions, and goals are modelled. We begin by presenting an overview of the particular planner used in this work, the epistemic PKS planner; we then describe how states are inferred from the low-level sensor data and how those states are translated into the representations used by PKS in the context of user interaction.

### 4.1 Planning with Knowledge and Sensing

The high-level planner is responsible for selecting robot actions to respond appropriately in the current scenario state. Since the activities of the robot include a mix of physical, dialogue, and social behaviours, the representation language of the planner must be able to support such action models. In this work, we use PKS (Planning with Knowledge and Sensing) (Petrick and Bacchus, 2002, 2004), a contingent planner that works with incomplete information and sensing actions. PKS is an **epistemic planner** that 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 update the planner's knowledge state. To ensure efficient inference, PKS restricts the type of knowledge it can represent:

**Kf**: This database is like a STRIPS database except that both positive and negative facts are permitted and the closed world assumption is not applied. Kf is used to model action effects that change the world and can include any ground literal or function (in)equality mapping $\ell$, where $\ell \in \texttt{Kf}$ means "the planner knows $\ell$."

**Kw**: This database models the plan-time effects of sensing actions that have one of two possible outcomes. $\phi \in \texttt{Kw}$ means that at plan time the planner either "knows $\phi$ or knows $\neg\phi$," and that at run time this disjunction will be resolved. PKS uses such information to build contingent branches in a plan, where each branch assumes one of the possible outcomes is true.

**Kv**: This database stores information about function values that will become known at execution time. Kv can model the plan-time effects of sensing actions that return a range of possible constants, where any unnested function term $f \in \texttt{Kv}$ 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 Kv 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.

**Kx**: This database models the planner's "exclusive-or" knowledge. Entries in Kx have the form $(\ell_1 \,|\, \ell_2 \,|\, \ldots \,|\, \ell_n)$, where each $\ell_i$ is a ground literal. Such formulae represent a type of disjunctive knowledge common in planning domains, namely that "exactly one of the $\ell_i$ is true."

(A fifth database modelling local closed world (LCW) information, is not used.) Questions about the knowledge state are answered using a set of **primitive queries**:

| | |
|---|---|
| **K($\phi$)**: | is $\phi$ known to be true? |
| **Kw($\phi$)**: | does the planner know whether $\phi$ is true or not? |
| **Kv($t$)**: | does the planner know the value of $t$? |

The negation of the queries is also permitted. An inference procedure evaluates the queries by checking the database contents and applying a set of reasoning rules.

**Table 1** Excerpt of a social state identified by the state manager

| **Property** | **Value** | **Confidence** |
|---|---|---|
| `seeksAttention(A1)` | `true` | 0.75 |
| `seeksAttention(A2)` | `false` | 0.45 |
| `lastSpeaker()` | `A1` | 1.00 |
| `lastEvent()` | `userSpeech(A1)` | 1.00 |
| `drinkOrder(A1)` | `green lemonade` | 0.68 |
| | `blue lemonade` | 0.32 |
| `lastAct(A1)` | `greet` | 0.25 |

Actions in PKS are modelled by a set of **preconditions** that query the knowledge state and a set of **effects** that update the knowledge 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., `add`(`Kf`, $\phi$) adds $\phi$ to the `Kf` database, while `del`(`Kw`, $\phi$) removes $\phi$ from `Kw`.

PKS builds plans by reasoning about actions in a forward-chaining manner: if the preconditions of a chosen action are satisfied by the knowledge state then the action's effects are applied 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 `Kw` and `Kv` knowledge. Planning continues along each branch until it satisfies the **goal** conditions, also specified as a list of primitive queries.

## 4.2 State Management

For PKS to operate successfully in the context of the larger robot system, it requires a discrete representation of the world, the robot, and all entities in the scene, integrating social, interaction-based, and task-based properties. Converting the continuous, low-level sensor information into the discrete states is the job of the state manager.

The social state is represented as a list of properties and their values, where every relation in the state has an associated **confidence** value, represented as a number between 0 and 1. In addition, every relation in the state can potentially have **multiple values**, with each possible value having its own confidence. Table 1 shows a sample social state using this representation, including multiple possible values for the `drinkOrder(A1)` relation. Social state properties fall into two main categories: properties that are directly transferred from the input sensors such as `headPos` (which tracks the 3D position of each customer's head), as well as derived properties such as `lastSpeaker` and `seeksAttention` which are computed by the state manager based on the input data.

For speech, the speech recogniser produces an $n$-best list of recognition hypotheses, each with an estimated confidence score, along with an estimate of the sound source angle and the angle confidence. The recognised hypotheses are parsed to extract the syntactic and semantic information using a grammar implemented in

OpenCCG (White, 2006), while the source angle is used together with the location information from vision to estimate which of the customers in the scene is most likely to have been speaking (`lastSpeaker`). If a possible speaker is found, the semantic information from speech is used to update `lastAct`. In the case that the customer says something regarding their drink order, we also update the value of `drinkOrder`, using a generic belief tracking procedure proposed by Wang and Lemon (2013), which maintains beliefs over user goals based on a small number of domain-independent rules using basic probability operations. This enables us to maintain a dynamically-updated list of the possible drink orders made by each customer, with an associated confidence value for each.

Information from the robot bartender's vision system (Pateraki et al., 2013) provides a continuous estimate of the location, gaze behaviour, and body language of all people in the scene in real time. Every feature reported by the vision system includes an estimated confidence value, which is incorporated into the state and also used for further processing. The information from the vision system contributes to the processing of speech as outlined above; it is also used to estimate which customer(s) are currently seeking attention (`seeksAttention`). `seeksAttention` is one of the most important properties required for the bartender scenario, and we have experimented with several methods of estimating it, including a rule based on the observation of customers in a real bar (Loth et al., 2013) and a set of classifiers trained on annotated robot bartender interactions (Foster et al., 2017).

## 4.3 Representing Properties, Actions, Objects, and Goals

The properties, actions, and goals that make up the planning domain definition are built on the state properties defined by the state manager but exist at a higher level of representation local to the planning system. All of the robot's high-level actions in the bartending scenario (physical, dialogue, and social) are modelled as part of the same planning domain, rather than using specialised tools for certain aspects of the problem (e.g., separating task and dialogue) as is common practice in many modern interactive systems. As a result, the planning domain representation must capture the dynamics of the task, the world, the agents, and the available objects.

Planning domain properties in the bartender scenario are shown in Table 2 (left). These properties are defined at a high level of abstraction and in many cases are based on the properties defined by the state manager. For instance, the planning property `seeksAttn(?a)` corresponds to the state manager property `seeksAttention`, while a property like `badASR(?a)` is extracted from the confidence values of other properties maintained by the state manager. Other planning properties like `greeted(?a)` or `served(?a)` do not have a direct analogue in the state manager but are instead derived from a set of properties being tracked at that level.

Actions in the bartending domain are also described at a high level of abstraction, and are inspired by studies of human customers ordering drinks from real bartenders in real bars (Loth et al., 2013). A list of the available actions is given in Table 2

**Table 2** Example of properties and action in the robot bartender domain.

| **Properties** | |
|---|---|
| `seeksAttn(?a)` | `?a` is seeking attention |
| `greeted(?a)` | `?a` has been greeted |
| `ordered(?a)` | `?a` has ordered |
| `ackOrder(?a)` | `?a`'s order has been acknowledged |
| `served(?a)` | `?a` has been served |
| `otherAttnReq` | other agents are seeking attention |
| `badASR(?a)` | `?a` was not understood |
| `transEnd(?a)` | the transaction with `?a` has ended |
| `inTrans=?a` | the robot is interacting with `?a` |
| `request(?a)=?d` | `?a` has requested drink `?d` |

| **Actions** | |
|---|---|
| `greet(?a)` | greet `?a` |
| `ask-drink(?a)` | ask `?a` for a drink order |
| `ack-order(?a)` | acknowledge `?a`'s order |
| `serve(?a,?d)` | serve drink `?d` to `?a` |
| `wait(?a)` | tell `?a` to wait |
| `ack-wait(?a)` | thank `?a` for waiting |
| `inform(?a,?d)` | tell `?a` about drink `?d` |
| `bye(?a)` | end interaction with `?a` |
| `not-understood(?a)` | alert that `?a` was not understood |

(right) with the PKS encoding for a selection of actions shown in Figure 4.3. The available list includes a mix of physical, dialogue, and social actions to reflect some of the behaviours that arise in typical interactions (e.g., as in Figure 2). For instance, `serve` is a standard planning action with a deterministic effect (i.e., it adds definite knowledge to PKS's `Kf` database so the planner comes to know particular facts like the customer has been served); 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 "good-bye"). The `inform` action is used to supply information about specific drinks in response to a customer query. The most interesting action is `ask-drink` which is modelled as a sensing or knowledge-producing action: the function term `request` is added to the planner's `Kv` database as an effect, indicating that this piece of information will become known at execution time. In other words, the planner will come to know the value of the drink the customer requested. 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.

The planning domain model also includes a list of the objects (drinks) and agents (customers) in the bar. This information is not hard-coded but is instead provided to the planner dynamically by the state manager, based on real-time observations provided by the input sensors, and defined as part of the planning problem's initial state (denoted in PKS syntax using two defined types, `drink` and `agent`). Changes in the object or agent list, when identified by the state manager, are also sent to the planner, causing it to update its domain model. Initially, the `inTrans` function is initially set to `nil` to indicate that the robot isn't interacting with any customers. The planner's goal is simply to serve each agent seeking attention, i.e.,

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

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

```
action greet(?a : agent)              action bye(?a : agent)
    preconds: K(inTrans = nil)            preconds: K(inTrans = ?a)
              !K(greeted(?a))                       K(served(?a))
              K(seekAttn(?a))                       !K(otherAttnReq)
              !K(ordered(?a))                       !K(badASR(?a))
              !K(otherAttnReq)                effects:  add(Kf,inTrans = nil)
              !K(badASR(?a))
    effects:  add(Kf,greeted(?a))
              add(Kf,inTrans = ?a)

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

**Fig. 3** Example encoding of PKS actions in the robot bartender domain.

## 5 Planning for Social Human-Robot Interaction

Using the planning model defined above, plans can now be generated to respond to many common interactive situations that arise in the bartender domain. This process is triggered by the appearance of agents (customers) in the scene which are reported to be seeking attention by the state manager. The planner responds by attempting to generate a plan to achieve the goal of serving all agents in the bar. Here we consider the generated behaviour in a number of common scenarios.

### 5.1 Ordering a Drink

The simplest interactive situation in the bartender domain is the case where a single agent `A1` is seeking attention in the bar, represented by the state manager adding a a new fact `seeksAttn(A1)` to the initial `Kf` database. Initially, the robot is not interacting with any agent (`inTrans = nil` ∈ `Kf`). In response, the planner 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))` | Serve `A1` the drink they requested |
| `bye(A1)` | End the transaction |

Initially, the planner can choose `greet(A1)` since no transaction is taking place (`inTrans = nil` ∈ `Kf`) and `A1` is seeking attention (`seeksAttn(A1)` ∈ `Kf`). The other preconditions of `greet(A1)` are trivially satisfied (i.e., none of `greeted(A1)`, `ordered(A1)`, `otherAttnReq`, or `badASR(A1)` are in `Kf`). After greeting `A1`, the `ask-drink(A1)` action is then chosen, updating the planner's knowledge state so that `ordered(A1)` ∈ `Kf` and `request(A1)` ∈ `Kv`, i.e., the planner knows that `A1` has ordered and knows the value of the drink that was requested. The `ack-order(A1)` is then selected to acknowledge the drink order to the customer. The most interesting action in the plan is `serve(A1,request(A1))` which, intuitively, has the effect of "serving `A1` the drink that `A1` requested". This follows as a consequence of the planner knowing the value of `request(A1)`, which is recorded in the planner's `Kv` database. Thus, `request(A1)` acts as a run-time variable whose definite value (`A1`'s actual drink order) will become known at run time. Finally, after serving the drink the `bye(A1)` action can be selected, resulting in `inTrans = nil` ∈ `Kf` and `transEnd(A1)` ∈ `Kf`, thereby ending the transaction and satisfying the goal.

## 5.2 Ordering Drinks with Multiple Agents

The planning domain model also enables more than one agent to be served if multiple customers are reported as seeking attention. For instance, in the case of two agents, `A1` and `A2`, the following plan might be built:

| | |
|---|---|
| `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(A2)` | 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) act as social actions that are used to defer a transaction with `A2` until `A1`'s transaction has finished. (The `otherAttnReq` property, whose value depends on `seeksAttn`, ensures that other agents seeking attention are told to wait before an agent is served.)

Larger number of customers result in plans with the same general structure. E.g., a plan for three agents, `A1`, `A2`, and `A3`, would look like the following:

```
wait(A2)          Tell agent A2 to wait
wait(A3)          Tell agent A3 to wait
greet(A1)         Greet agent A1
...               Transact with A1
bye(A1)           End A1's transaction
ack-wait(A2)      Thank A2 for waiting
...               Transact with A2
bye(A2)           End A2's transaction
ack-wait(A3)      Thank A3 for waiting
...               Transact with A3
bye(A3)           End A3's transaction
```

Similarly, if a new customer appears it is dynamically reported to the planner, possibly triggering a replanning operation: the newly built plan might result in the extension of an existing plan (which might reflect a transaction currently in progress) to include actions for interacting with the new agent if they are seeking attention. However, it is important to note that we are not just stitching together single-agent plans to account for the number of agents in the scenario. Instead, the planner generates a plan appropriate to the social context in response to the state information reported to it by the state manager.

## 5.3 Ordering a Drink with Restricted Drink Choices

From a planning point of view, the above plans rely on the planner's ability to reason about particular types of knowledge (e.g., functions like `request(A1)`) which act as variables in parameterised plans. However, an alternative type of plan can also be built in the case that the possible set of drinks is explicitly restricted. For instance, consider a single agent `A1` seeking attention, where the planner also told there are three possible drinks that can be ordered: juice, water, and beer. This information is represented in the planner as a type of "exclusive or" knowledge in the `Kx` database:

```
request(A1) = juice | request(A1) = water | request(A1) = beer
```

The planner can now build a plan of the following form to serve the customer:

```
greet(A1)                     Greet agent A1
ask-drink(A1)                 Ask A1 for drink order
ack-order(A1)                 Acknowledge A1's order
branch(request(A1))           Form conditional plan
   K(request(A1) = juice):    If juice was requested
      ...
      serve(A1,juice)         Serve juice to A1
   K(request(A1) = water):    If water was requested
      ...
      serve(A1,water)         Serve water to A1
   K(request(A1) = beer):     If beer was requested
      ...
      serve(A1,beer)          Serve beer to A1
bye(A1)                       End the transaction
```

In this case, a contingent 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 true; in the second branch `request(A1) = water` is true; and so on. Planning continues along each branch under the given assumption. (We note that this type of branching is only possible here because the planner had initial `Kx` knowledge that restricted `request(A1)`, combined with `Kv` knowledge provided by the `ask-drink` action.) Along each branch, an appropriate `serve` action is added to deliver the appropriate drink. The places in the plan indicated by "`...`" indicate places where drink-specific interactions (subdialogues) could be inserted. For instance, each branch may require different actions to serve a drink, such as putting the drink in a special glass, or requesting additional information from the customer (i.e., "would you like ice in your water?").

## 6 Plan Execution, Monitoring, and Recovery

Once a plan is built, it is executed by the robot one action at a time. A plan execution monitor tracks the plan, comparing the expected plan states against sensed states provided by the state manager, 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 and that effects needed by actions or goals later in the plan have been achieved as expected. In the case of a mismatch, the planner is directed to build a new plan, using the sensed state as a new initial state.

The execution of individual actions is handled by dividing each high-level planned action into specific output modalities—speech, head motions, and arm manipulation behaviour—that can be executed by the robot. This mapping is specified by a simple rule-based structure containing specifications of each output (Isard and Matheson, 2012). The resulting structure is then passed to the multimodal output planner, which mediates execution to each output channel. Language output is specified in terms of communicative acts based on Rhetorical Structure Theory (RST) (Mann and Thompson, 1988), using a generation module that translates RST into speech by the robot's animatronic head. The robot also expresses itself through facial expressions, gaze, and arm motions. The animatronic head can express a number of predefined expressions, while the robot arm can perform tasks like grasping objects (e.g., to hand over a drink to a customer). Multimodal behaviour is coordinated across the various output channels to ensure they are synchronised temporally and spatially. For instance, an action `serve(?a,?d)` to serve an agent a drink might be transformed into multimodal outputs that result in the robot smiling at `?a` (an animatronic head facial expression) while physically handing over drink `?d` (a robot arm manipulation action) and saying to the customer "here is your drink" (speech output).

If the plan execution monitor detects a situation where a plan has failed, for instance, due to unexpected outcomes like action failure, the planner is invoked to construct a new plan. 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 an agent `A1`'s response to `ask-drink(A1)` was not understood due to low-confidence automatic speech recognition, the state report sent to the planner will have no value for `request(A1)`, and `badASR(A1)` will be set to true. This situation will be detected by the plan execution monitor and the planner will be directed to build a new plan. One possible 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 plan:

| | |
|---|---|
| `ask-drink(A1)` | Ask `A1` for drink order |
| `???` | `A1` was not understood |
| `[Replan]` | Replan |
| `not-understood(A1)` | Alert `A1` it was not understood |
| `ask-drink(A1)` | Ask `A1` again for drink order |
| ... | |

Another consequence of this approach is that certain types of overanswering can be handled through plan 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:

| | |
|---|---|
| `greet(A1)` | Greet `A1` |
| `???` | `A1` says "I'd like a beer" |
| `[Replan]` | Replan |
| `ack-order(A1)` | Acknowledge `A1`'s drink order |
| `serve(A1,request(A1))` | Serve `A1` their drink |
| ... | |

In this case, the state manager would include `request(A1) = beer` in its state report, along with `ordered(A1)`. The execution monitor would detect that the preconditions of `ask-drink(A1)` aren't met and direct the planner to replan. A new plan could then omit `ask-drink` and proceed to acknowledge and serve the requested drink.

## 7 Discussion and Conclusions

This chapter has described how automated planning can be applied to the problem of social human-robot interaction in the JAMES robot bartending domain, as an alternative to mainstream approaches to interaction management. In particular, we have shown how the planning representation is engineered from social states induced from different input modalities, and how plans are built incorporating a mix of task, dialogue, and social actions, with execution involving various output modalities on the robot. The use of the epistemic PKS planner has also provided certain benefits during the work, such as enabling sensing actions to be used to model certain types of dialogue actions, generating parameterised high-level plans, and considering subdialogues with contingent branches.

The planning approach has also presented certain technical advantages. For instance, the JAMES robot system has been evaluated through a series of user studies aimed at exploring socially-appropriate interaction in the bartender scenario, where participants successfully ordered and received drinks from the bartender (Foster et al., 2012). An interesting variant of the study compared the full planning domain described above with a domain that dealt with task-based actions only. From a representation point of view, this was done by simply removing the social actions from the domain model. Results showed that the social version led to more efficient dialogues (Giuliani et al., 2013). Another variant of multiple customer drink ordering also considered different ordering strategies when agents arrive in groups (e.g., interacting with a group representative versus transacting with all agents in a single group before moving to another group). Again, the changes required to support planning in this new setting resulted from modifications to the domain model: in this case adding a new property to track agents in groups, and introducing another type of drink ordering action to accommodate multiple agents ordering drinks in a group.

More generally, interactive systems also offer several opportunities for the automated planning community to showcase their tools and techniques. For instance, interaction problems could form the the basis for new challenge domains in planning, and the standard planning representation languages offer an approach to modelling problems that break the tight link between representation and reasoning that is often found in interaction toolkits. There are lessons that the planning community can also learn from the interactive systems community. For example, the issue of user evaluation is at the heart of interactive systems research, with a focus on (non-expert) users interacting with the developed tools. The fact that interactive systems are also inherently application driven means that planning must be situated in the context of larger, more complex systems, requiring a degree of maturity and robustness in development that often goes beyond lab settings, but which could facilitate the wider adoption of planning approaches in such settings. Our ongoing research aims to address some of these issues by adapting our planning techniques to other types of service robots and scenarios that involve interacting with humans in public spaces.

## 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 (JAMES, `http://james-project.eu/`).

## References

Amazon (2020) Alexa Skills Kit Official Site. `https://developer.amazon.com/en-GB/alexa/alexa-skills-kit`, accessed: 2020-02-09

Appelt D (1985) Planning English Sentences. Cambridge University Press

Asher N, Lascarides A (2003) Logics of Conversation. Cambridge University Press

Benotti L (2008) Accommodation through tacit sensing. In: Proceedings of LONDIAL 2008, London, United Kingdom, pp 75–82

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

Bui TH (2006) Multimodal dialogue management - state of the art. Tech. Rep. 06–01, University of Twente (UT), Enschede, The Netherlands

Cohen P, Levesque H (1990) Rational interaction as the basis for communication. In: Intentions in Communication, MIT Press, Cambridge, MA, pp 221–255

Fikes RE, Nilsson NJ (1971) STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2:189–208

Foster ME, Petrick RPA (2017) Separating representation, reasoning, and implementation for interaction management: Lessons from automated planning. In: Dialogues with Social Robots: Enablements, Analyses, and Evaluation, Springer Singapore, Singapore, pp 93–107, DOI 10.1007/978-981-10-2585-3_7

Foster ME, Gaschler A, Giuliani M, Isard A, Pateraki M, Petrick RPA (2012) Two people walk into a bar: Dynamic multi-party social interaction with a robot agent. In: Proceedings of ICMI 2012, pp 3–10, DOI 10.1145/2388676.2388680

Foster ME, Gaschler A, Giuliani M (2017) Automatically classifying user engagement for dynamic multi-party human–robot interaction. International Journal of Social Robotics 9(5):659–674, DOI 10.1007/s12369-017-0414-y

Fox M, Long D, Magazzeni D (2017) Explainable planning. In: Proceedings of the IJCAI Workshop on Explainable AI

Giuliani M, Petrick RPA, Foster ME, Gaschler A, Isard A, Pateraki M, Sigalas M (2013) Comparing task-based and socially intelligent behaviour in a robot bartender. In: Proceedings of ICMI 2013, DOI 10.1145/2522848.2522869

Google (2020) DialogFlow. `https://dialogflow.com/`, accessed: 2020-02-09

Hovy E (1988) Generating natural language under pragmatic constraints. Lawrence Erlbaum Associates, Hillsdale, NJ, USA

ICAPS (2019) ICAPS Competitions. `http://www.icaps-conference.org/index.php/Main/Competitions`, accessed: 2019-08-01

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

Janarthanam S, Hastie H, Deshmukh A, Aylett R, Foster ME (2015) A reusable interaction management module: Use case for empathic robotic tutoring. In: Proceedings of goDIAL 2015, Gothenburg, Sweden

Johnston M, Bangalore S, Vasireddy G, Stent A, Ehlen P, Walker M, Whittaker S, Maloor P (2002) MATCH: An architecture for multimodal dialogue systems. In: Proceedings of ACL 2002, Philadelphia, Pennsylvania, USA, pp 376–383

Jokinen K, McTear M (2009) Spoken dialogue systems. Synthesis Lectures on Human Language Technologies 2(1):1–151

Koller A, Stone M (2007) Sentence generation as planning. In: Proceedings of ACL 2007, Prague, Czech Republic, pp 336–343

Larsson S, Traum DR (2000) Information state and dialogue management in the TRINDI dialogue move engine toolkit. Natural Language Engineering 6(3&4):323–340, DOI 10.1017/S1351324900002539

Lison P (2015) A hybrid approach to dialogue management based on probabilistic rules. Computer Speech & Language DOI 10.1016/j.csl.2015.01.001

Loth S, Huth K, De Ruiter JP (2013) Automatic detection of service initiation signals used in bars. Frontiers in Psychology 4(557), DOI 10.3389/fpsyg.2013.00557

Mann WC, Thompson SA (1988) Rhetorical structure theory: Toward a functional theory of text organization. Text 8(3):243–281

McDermott D, Ghallab M, Howe A, Knoblock C, Ram A, Veloso M, Weld D, Wilkins D (1998) PDDL – The Planning Domain Definition Language (Version 1.2). Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control

McTear M, Callejas Z, Griol D (2016) The Conversational Interface. Springer International Publishing, DOI 10.1007/978-3-319-32967-3

Olaso JM, Milhorat P, Himmelsbach J, Boudy J, Chollet G, Schlögl S, Torres MIT (2016) A multi-lingual evaluation of the vAssist spoken dialog system: Comparing Disco and RavenClaw. In: Proceedings of IWSDS 2016, Saariselkä, Finland

Papaioannou I, Dondrup C, Lemon O (2018) Human-robot interaction requires more than slot filling - multi-threaded dialogue for collaborative tasks and social conversation. In: Proceedings of the FAIM/ISCA Workshop on Artificial Intelligence for Multimodal Human Robot Interaction, pp 61–64

Pateraki M, Sigalas M, Chliveros G, Trahanias P (2013) Visual human-robot communication in social settings. In: Proceedings of ICRA Workshop on Semantics, Identification and Control of Robot-Human-Environment Interaction

Peltason J, Wrede B (2011) The curious robot as a case-study for comparing dialog systems. AI Magazine 32(4):85–99, DOI 10.1609/aimag.v32i4.2382

Perrault CR, Allen JF (1980) A plan-based analysis of indirect speech acts. American Journal of Computational Linguistics 6(3–4):167–182

Petrick RPA, Bacchus F (2002) A knowledge-based approach to planning with incomplete information and sensing. In: Proceedings of AIPS 2002, pp 212–221

Petrick RPA, Bacchus F (2004) Extending the knowledge-based approach to planning with incomplete information and sensing. In: Proceedings of ICAPS 2004, pp 2–11

Petrick RPA, Foster ME (2013) Planning for social interaction in a robot bartender domain. In: Proceedings of ICAPS 2013, Rome, Italy

Rintanen J (2004) Complexity of planning with partial observability. In: Proceedings of ICAPS 2004, pp 345–354

Wang Z, Lemon O (2013) A simple and generic belief tracking mechanism for the dialog state tracking challenge: On the believability of observed information. In: Proceedings of SIGDial 2013

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

Young RM, Moore JD (1994) DPOCL: a principled approach to discourse planning. In: Proceedings of INLG 2004, Kennebunkport, Maine, USA, pp 13–20