DELIVERABLE D4.2

INITIAL EXTENSIONS FOR KNOWLEDGE-LEVEL PLANNING AND HEURISTIC SEARCH

Ron Petrick (UEDIN)

Beneficiaries: UEDIN (lead)
Workpackage: WP4: Planning and Reasoning
Description: This deliverable reports on the high-level planning and reasoning components included in the initial integrated system (D7.1), including the underlying PKS planner and an associated plan execution monitor. These components make use of the initial representations developed in D4.1, process state messages provided by the WP3 social state recogniser (D3.1), and provide plans understandable by the output planner (D2.1). This deliverable also describes the state of proposed heuristic search extensions for PKS.

Version: Final
Nature: Report (R)
Dissemination level: Public (PU)
Pages: 36
Date: 2012-07-31
1 Executive summary

A central contribution of WP4 (Planning and Reasoning) is to extend general purpose knowledge-level planning and reasoning techniques to the rich state spaces that arise from embodied social communication in the JAMES bartending scenarios. This work package provides the apparatus needed to support plan generation and execution on the JAMES robot platform in the form of a high-level planner and associated plan execution monitor. In the first phase of the project (Month 1–Month 12), this work package focused on integrating a high-level planner with the social state recogniser (WP3) and output planner (WP2) in the initial JAMES system reported in deliverable D7.1. A central problem in this task was the issue of representation in order to facilitate the exchange of information between modules in the system, centred around Task 4.1 (Extended high-level representations for planning and reasoning) and Task 4.2 (Plan generation in social state spaces). The results of this work were reported in deliverable D4.1.

In the second phase of the project (Month 12–Month 18), this work package continued the work begun in Year 1 on Tasks 4.1 and 4.2, and also featured initial work on Task 4.3 (Domain-independent heuristic search for knowledge-level planning). The main technical focus was on improving the integration between the planner and other modules in the mainline JAMES system, while making improvements within the planning module itself to support extensions planned for later phases of the project. The present deliverable presents a snapshot of the state of the planning system, the main JAMES planning domains, and proposed extensions.

The ability to reason and plan is essential for an agent acting in a dynamic and incompletely known world, such as the JAMES bartending scenario. High-level planning capabilities in JAMES are supplied by the PKS planner [3, 4], which UEDIN is extending for use in social domains. PKS is a state-of-the-art conditional planner that constructs plans in the presence of incomplete information. Unlike traditional planners, PKS builds plans at the “knowledge level”, by representing and reasoning about how the planner’s knowledge state changes during plan generation. Actions are specified in a STRIPS-like [2] language in terms of action preconditions (state properties that must be true before an action can be executed) and action effects (the changes the action makes to properties of the state). PKS can build conditional plans with sensing actions, and supports limited numerical reasoning, run-time variables [1], and features like functions that arise in real-world planning scenarios.

Like most AI planners, PKS operates best in discrete, symbolic state spaces described using logical languages. As a result, work that addresses the problem of integrating planning on real-world robot platforms often centres around the problem of representation, and how to abstract the capabilities of a robot and its working environment so that it can be put in a suitable form for use by a goal-directed planner. Integration also requires the ability to communicate information between system components. Thus, the design of a planning system often has to take into consideration external concerns, to ensure proper interoperability with modules that aren’t traditionally considered in pure theoretical planning settings.

Furthermore, a planner must strive for efficiency to overcome the computation challenges arising from operating in real-world environments like the bartending domain. To this end, work on improving the plan generation process employed by the planner, for instance by adapting the use of heuristic search techniques to knowledge-level state spaces, are central to scaling this technology to real-world sized domains.

Two attached papers (see below) provide details of our contributions in the past work period to the above tasks.

Overall, this work package reports a number of significant developments in three main areas:

- A description of the current state of integration between the planner, social state recogniser (WP3), and output planner (WP2) in the JAMES mainline system. In particular, the general purpose planner replaces the use of traditional dialogue/interaction management in the JAMES system.

- An updated version of the mainline planning domain used in the JAMES bartending scenario. The domain description allows plans to be generated for a drink ordering task in single and multiagent settings. The domain also allows for plan recovery due to simple forms of speech recognition failures and over-answering. This deliverable also includes a description of how certain types of uncertainty arising from automatic speech recognition (ASR) errors could be modelled within the planning domain.
• Initial studies into the application of heuristic search techniques to the state spaces used by the PKS planner. This is the first stage of Task 4.3 and lays the groundwork for the implementation of a more efficient plan generation algorithm in later phases of the project.

A number of tasks remain open at the time of this report and constitute ongoing and future work:

• We have not focused on ways of improving planner efficiency by employing domain-dependent knowledge, which is planned for Year 3 of the project (Task 4.4). As preliminary work in preparation for this task, we are exploring connections to the work of WP8 in order to use the results of the human-human studies as a heuristic for guiding plan generation in certain situations.

• As part of Task 4.2 we are extending our existing planning apparatus for ordinary action planning to dialogue planning by reasoning about multiagent knowledge in a manner similar to [7]. Theoretical work on this task is mostly complete and we are proceeding to implement a prototype for initial experimentation.

• In conjunction with FORTISS and WP6, we are experimenting with extensions to the PKS planner that allow modules external to the planner to be used for reasoning about plan-level properties. The particular application of this work is the combination of high-level task planning with low-level robot motion planning.

• Improvements to the planner and plan execution monitor codebases are continuing as part of ongoing system implementation work leading to the intermediate JAMES system expected at Month 24 of the project as part of WP7.

For the latest development version of the PKS software, see http://homepages.inf.ed.ac.uk/rpetrick/software/pks/.

1.1 Attached papers

The following papers are attached to this report:


Abstract: A robot coexisting with humans must not only be able to successfully perform physical tasks, but must also be able to interact with humans in a socially appropriate manner. In many social settings, this involves the use of social signals like gaze, facial expression, and language. In this paper we discuss preliminary work focusing on the problem of combining social interaction with task-based action in a dynamic, multiagent bartending domain, using an embodied robot. We discuss how social states are inferred from low-level sensors, using vision and speech as input modalities, and present a planning approach that models task, dialogue, and social actions in a simple bartending scenario. This approach allows us to build interesting plans, which have been evaluated in a real-world study with human subjects, using a general purpose, off-the-shelf planner, as an alternative to more mainstream methods of interaction management.

This paper combines and extends the results of [5, 6], a draft of which was included as part of deliverable D4.1.


Abstract: This report describes part of UEDIN’s contribution to the ongoing work of general-purpose, knowledge-level planning in the JAMES project. The focus of this document is a description of the current state of proposed heuristic search extensions for the PKS (Planning with Knowledge and Sensing) planner, and its derivative technologies, which will be used and extended throughout the project as part of WP4 (Task 4.3). In this reporting period (M18), we present an overview of the problem of informed search for knowledge-level planning, and describe our current workplan for extending the PKS planner during the next work period of the project. We also highlight other research directions for knowledge-level planning currently in progress which are related to the heuristic search work.
References


Knowledge-Level Planning for Task-Based Social Interaction

Ronald P. A. Petrick  
School of Informatics, University of Edinburgh, Edinburgh EH8 9AB, Scotland, United Kingdom  
RPETRICK@INF.ED.AC.UK

Mary Ellen Foster  
School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh EH14 4AS, Scotland, United Kingdom  
M.E.FOSTER@HW.AC.UK

Amy Isard  
School of Informatics, University of Edinburgh, Edinburgh EH8 9AB, Scotland, United Kingdom  
AMYI@INF.ED.AC.UK

Abstract

A robot coexisting with humans must not only be able to successfully perform physical tasks, but must also be able to interact with humans in a socially appropriate manner. In many social settings, this involves the use of social signals like gaze, facial expression, and language. In this paper we discuss preliminary work focusing on the problem of combining social interaction with task-based action in a dynamic, multiagent bartending domain, using an embodied robot. We discuss how social states are inferred from low-level sensors, using vision and speech as input modalities, and present a planning approach that models task, dialogue, and social actions in a simple bartending scenario. This approach allows us to build interesting plans, which have been evaluated in a real-world study with human subjects, using a general purpose, off-the-shelf planner, as an alternative to current mainstream methods of interaction management.

1. 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 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. However, the problem of building a robot to meet the goals of social interaction presents several challenges, especially for the reasoning and action selection components of such a system. Not only does the robot require the ability to recognise and understand appropriate multimodal social signals (e.g., gaze, facial expression, and language), but it must also generate realistic responses using similar modalities.

To address this challenge, and help focus our work, we are investigating the sub-problem of task-based social interaction using a bartending scenario as our target domain. In particular, we are developing a robot bartender (Figure 1) that is capable of dealing with multiple customers in a simple drink-ordering scenario. Interactions in this scenario incorporate a mixture of task-based aspects (e.g., ordering and paying for drinks) and social aspects (e.g., managing multiple interactions).
Moreover, the primary interaction modality in this setting is speech; human agents communicate with the robot bartender via speech and the robot must respond in a similar manner.

Our approach to high-level reasoning and action selection in this setting is to use AI planning, specifically, knowledge-level planning (Petrick & Bacchus, 2002; Petrick & Bacchus, 2004) techniques. The ability to reason and plan is essential for a cognitive agent acting in a dynamic and incompletely known world such as the bartending scenario. General-purpose automated planners are good at building goal-directed plans of action under many challenging conditions, especially in task-based contexts. Moreover, recent work (Steedman & Petrick, 2007; Brenner & Kruijff-Korbayová, 2008; Benotti, 2008; Koller & Petrick, 2011) has investigated the use of automated 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. The use of planning for natural language processing is particularly important since plan generation in the bartending domain will require a mix of traditional task-based actions (e.g., handing the customer a drink) and speech acts (e.g., asking a customer for a drink order).

While planning offers a possible 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.

As a result, we focus on three main areas in this paper:
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: “Two people walk into a bar”.

• First, we show how states with task, dialogue, and social features are derived from low-level sensor observations.

• Using these states, we show how plans are generated by modelling the problem as an instance of planning with incomplete information and sensing using a planner called PKS (Petrick & Bacchus, 2002; Petrick & Bacchus, 2004), as an alternative to more mainstream methods of interaction management.

• Finally, we present a planning domain that models a simple bartending scenario. This domain has been evaluated in a real-world study, and provides the basis for future work currently underway.

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

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

2. Overview of the Bartending Scenario and System Architecture

The work in this paper is centred around a bartending scenario which supports interactions similar to the one shown in Figure 2. In this scenario, two customers enter the bar area and attempt to attract the robot’s attention to order a drink. Even this simple interaction presents certain challenges to 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

¹. See http://james-project.eu/ for more information about the JAMES project.
ensure that they are served in the order that they arrived; while the output components must select and execute concrete actions for each output channel that correctly realises high-level plans.

To address these challenges, the system architecture (Figure 3) combines components arising from multiple fields of research (notably natural language processing, social state processing, and automated planning), building on a standard three-layer structure: the low level deals with modality-specific information which is often continuous (e.g., spatial coordinates, speech-recognition hypotheses, and robot arm trajectories); the mid-level works with more general, cross-modality representations of states and events; while the high level reasons about abstract structures in a logical form (e.g., knowledge and action). In the remainder of this section we will outline the operation of the main components in the system.

### 2.1 Robot Hardware and Vision System

The robot system used in the bartender scenario provides the sensors and effectors that interact with the real world. The robot hardware itself consists of two 6-degrees-of-freedom industrial manipulator arms with grippers, mounted to resemble human arms. Sitting atop the main robot torso is an animatronic talking head capable of producing facial expressions, rigid head motion, and lip-synchronised synthesised speech.

One of the primary input modalities used by the robot bartender is 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 et al., 2012; Pateraki et al., 2011; Pateraki et al., 2012). Each agent’s focus of attention is also derived using torso orientation. The partially abstracted information resulting from this process is then made available to modules like the state manager (Section 2.3) for further processing.
Figure 4. XML structures for natural language processing: (a) OpenCCG logical form for the utterance “Please give me a beer”. (b) XML for the serve(a1,beer) action (“serve a beer to agent 1”).

2.2 Natural Language Understanding

A second important input modality in the system is speech, since linguistic interaction is central to social communication. In our case, the linguistic processing system combines 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 hardware (Microsoft Corporation, 2012) 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 $\Pi_u = \{\langle h_1, c_1 \rangle, \langle h_2, c_2 \rangle, \ldots \langle h_n, c_n \rangle\}$, and a final best hypothesis $h^*_u$, along with a confidence score $c^*_u$ and an estimate of the source angle $\theta^*_u$. In addition, we have created a speech recognition grammar in the SRGS format (Hunt & McGlashan, 2004) that covers all expected user utterances in the bartending scenario. The resulting grammar constrains the recognition task, allowing the system to achieve more reliable results. In the current system, the recognition task is completed and the top hypothesis is passed to the interpretation system with its confidence score and angle; in the future, we also plan to use the intermediate hypotheses (see Section 4).

Once the user speech has been recognised, it must be further processed to extract the underlying meaning. To do this, we parse the recognised speech hypothesis using a grammar defined in OpenCCG (White, 2006). OpenCCG is an open-source implementation of Combinatory Categorial Grammar (Steedman, 2000), a unification-based categorial framework which is both linguistically and computationally attractive. The grammar contains both syntactic and semantic information, and is used both for parsing the linguistic input and for surface realisation of the selected output as described later. For instance, the XML representation of the OpenCCG logical form resulting from the sentence Please give me a beer is shown in Figure 4(a). Once the top speech hypothesis has been parsed, the logical form is then passed on—together with the confidence and angle information—to the state manager for further processing.
2.3 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 of the world, the robot, and all entities in the scene, by combining social, dialogue, and task-based properties. The derived state information resulting from this process is available for use by other components in the system (primarily the planner) through a persistent, queryable interface to the state, or a mechanism that informs a component whenever a significant state change occurs.

Formally, the low-level input components of the system correspond to a set $\Sigma$ of sensors, $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, where each sensor $\sigma_i$ returns an observation $obs(\sigma_i, t)$ about some aspect of the world at a time point $t$. If appropriate, a primary sensor (such as the speech recogniser or a body-pose estimator) may have an associated sensor that indicates the estimated reliability of the observation, capturing the fact that real-world sensors generally produce noisy results. We denote the set of sensor readings occurring at a particular timepoint $t$ as $\Sigma_t = \{obs(\sigma_1, t), obs(\sigma_2, t), \ldots, obs(\sigma_n, t)\}$.

The state representation is based on a set $\Phi$ 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 fluents at $t$ by $\Phi_t$. In this work, fluents are updated in a Markovian fashion, where the value of a fluent is a function of the observations returned by the sensor set along with the set of fluent values from the previous time point, i.e., $f_{i, t} = \Gamma_i(\Sigma_{t-1}, \Phi_{t-1})$. Typically, each fluent depends on a strict subset of the sensor observations, and the mapping is not necessarily one-to-one: any given sensor may map to zero, one, or many fluents, as appropriate. A state $S_t$ is a snapshot of the values of all instantiated fluents at a time $t$ during the interaction, i.e., $S_t = \Phi_t$.

Intuitively, states represent a point of intersection between the low-level data produced by the sensors and the high-level representations used by components like the planner: the set $\Sigma$ of available sensors is defined by the low-level system components, while the set $\Phi$ of required fluents is provided by the high-level reasoning system. States are induced from a set of sensor observations and the corresponding sensor/fluent mappings (i.e., the functions $\Gamma_i$). Implementing the state manager therefore consists primarily of defining the set of mapping functions $\Gamma_i$. In the context of social robotics, this is the problem of social signal processing (Vinciarelli et al., 2009), which is a topic that has received an increasing amount of attention in recent years. The most common technique is to use labelled data to train supervised learning models such as Support Vector Machines (SVMs) to classify the sensor data; this often requires the use of signal processing and feature extraction to convert the sensor data into a form suitable for SVM training. This process is not always straightforward: often, it is not the sensor data in any single frame that determines the value of a state fluent, but rather the patterns found in a sequence of signals, and when determining the value of fluents that combine information from multiple signals, the relevant information may not occur simultaneously, but require temporal cross-modal fusion (hence the inclusion of $\Phi_{t-1}$ as an argument to $\Gamma_i$).

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

In the bartender robot system, we consider each low-level input component to be made up of a set of sensors. The linguistic interpreter corresponds to two sensors: one that observes the parsed content of the recognised speech, and another that returns the estimated angle of the sound source. Both of these sensors also have associated confidence scores, which are represented as additional sensors. 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.

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

The actual state fluents modelled in the state are defined by the particular requirements of the scenario (Figure 2): we represent all agents in the scene, their locations, torso orientations, and attentional states, along with their 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.

In the current system, the state manager is rule-based. One set of rules infers user social states (e.g., seeking attention) based on the low-level sensor data, using guidelines derived from 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 determine when new state reports are published, which helps control turn-taking.

### 2.4 Planning and Execution Monitoring

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

In this work we use the PKS planner (Petrick & Bacchus, 2002; Petrick & Bacchus, 2004) for action selection. PKS (Planning with Knowledge and Sensing) is a conditional planner that constructs plans in the presence of incomplete information and sensing actions. PKS works at the “knowledge-level” by reasoning about how the planner’s knowledge state, rather than the world state, changes due to action. PKS works with a restricted subset of a first-order logical language, and a limited amount of inference, and supports features such as functions and run-time variables (Etzioni et al., 1992). This approach differs from planners that work with possible worlds models.
or belief states. However, as a trade-off, its restricted representation means that certain types of knowledge cannot be directly modelled in PKS.

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

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

$K_w$: This database models the plan-time effects of “binary” sensing actions. $\phi \in K_w$ means that at plan time the planner either “knows $\phi$ or knows $\neg \phi$,” and that at execution time this disjunction will be resolved. The planner can use such information to include conditional branches in a plan, where each branch assumes a particular outcome of the sensing is true.

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

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

(A fifth database that stores “local closed world” information (Etzioni et al., 1994) is not used here.)

PKS’s databases can be inspected through a set of primitive queries that ask simple questions about the planner’s knowledge state: whether facts are (not) known to be true (a query of the form $\lnot K(\phi)$), whether function values are (not) known (a query $\lnot K_v(t)$), or if the planner “knows whether” certain properties are true or not (a query $\lnot K_w(\phi)$). An inference algorithm evaluates primitive queries by checking the contents of the various databases.

An action in PKS is modelled by a set of preconditions that query the agent’s knowledge state, and a set of effects that update the state. Action preconditions are simply a list of primitive queries. Action effects are described by a collection of STRIPS-style “add” and “delete” operations that modify the contents of individual databases. E.g., $\text{add}(K_f, \phi)$ adds $\phi$ to the $K_f$ database, while
del(Kw, φ) removes φ from the Kw database. In the bartending scenario, all actions (i.e., task, dialogue, and social) is modelled as part of the same underlying PKS planning domain, rather than using specialised tools as is common practice in modern interactive dialogue systems. Thus, all high-level action selection is determined by the same general purpose planning mechanism.

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 Kw and Kv 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 plan generation component, PKS is aided by an execution monitor which controls replanning. The monitor takes as input a PKS plan, whose execution it tracks, and a state description 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.

2.5 Output Planning and Language Generation

Output in the robot system is based on processing high-level actions selected by the planner and dividing them into speech, head motion, and arm manipulation behaviours that can be executed in the real world. To do so, we use an XML-based structure which contains specifications for each of the output modalities (Isard & Matheson, 2012). This structure is generated using a rule-based approach, which processes the abstract planned action and splits it into its component subparts. The resulting structure is then passed to the particular output modules for execution.

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

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

For instance, Figure 4(b) shows an XML specification for a high-level action serve(a1, beer) which denotes the abstract behaviour “serve a beer to agent a1”. In this case, the specification results in the robot smiling (an animatronic head facial expression) while handing over a beer (a robot arm manipulation action) and saying to the customer “here is your drink” (speech output).
3. Planning Interactions in the Bartending Domain

3.1 A simple bartending domain

A PKS planning domain for the bartending scenario describes the domain’s properties and actions, denoting particular features of the world, agents, and objects. Domain properties are divided into two types: predicates and functions. In this case, the planning domain properties are based on fluents similar to those 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 bartending domain use domain properties to describe their preconditions and effects. Our domain includes seven high-level actions:

- `greet(?a)`: greet an agent ?a,
- `ask-drink(?a)`: ask agent ?a for a drink order,
- `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 six actions (the actions required for single agent interactions) are given in Figure 5. Actions are described at an abstract level and include a mix of physical, sensory, and speech acts. For instance, `serve` is a standard planning action with a deterministic effect (i.e., it adds definite knowledge to the planner’s Kf database); however, when executed at the robot level it causes the robot to hand over a drink to an agent and confirm the drink order through speech. Actions like `greet`, `ack-order`, and `bye` are modelled in a similar way as `serve` but only map to speech output at the robot level (e.g., “hello”, “okay”, and “good-bye”). The most interesting action
is ask-drink which is modelled as a sensing action in PKS: the function term request is added to
the planner’s $K_v$ database as an effect, indicating that the mapping for this piece of information will
become known at execution time. The not-understand action is used as a directive to the speech
output system to produce an utterance that (hopefully) causes the agent to repeat its last response.
The wait and ack-wait actions are used to control interactions when multiple agents are seeking
the attention of the bartender.

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

### 3.2 Example interactions in the bartending domain

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

$$\forall K(a : \text{agent}) K(\text{seeksAttn}(a)) \Rightarrow K(\text{transEnd}(a)).$$

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

#### 3.2.1 Ordering a drink

In our first example, we consider the case where there is a single agent $a_1$. No specific drinks are
defined and no other state information is supplied, except that the robot is not interacting with any
agent (i.e., inTrans = nil $\in K_f$). The appearance of $a_1$ seeking attention is reported to PKS in
an initial state report, which has the effect of adding a new constant named $a_1$ of type agent to the
planner’s domain description, and adding a new fact seeksAttn($a_1$) 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($a_1$).  [Greet agent $a_1$]
- ask-drink($a_1$).  [Ask $a_1$ for drink order]
- ack-order($a_1$).  [Acknowledge $a_1$’s drink order]
- serve($a_1$, request($a_1$)).  [Give the drink to $a_1$]
- bye($a_1$).  [End the transaction]

Initially, the planner can choose greet($a_1$) since inTrans = nil $\in K_f$ and seeksAttn($a_1$) $\in K_f$,
and the other preconditions are trivially satisfied (i.e., none of greeted($a_1$), ordered($a_1$),
action greet(?a : agent)
preconds: K(inTrans = nil) & -K(greeted(?a)) &
K(seeksAttn(?a)) & -K(ordered(?a)) &
-K(otherAttnReq) & -K(badASR(?a))
effects: add(Kf,greeted(?a)),
add(Kf,inTrans = ?a)

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

action ack-order(?a : agent)
preconds: K(inTrans = ?a) & K(ordered(?a)) &
-K(ackOrder(?a)) & -K(otherAttnReq) &
-K(badASR(?a))
effects: add(Kf,ackOrder(?a))

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

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

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

Figure 5. PKS actions in a single agent interaction
otherAttnReq, or badASR(a1) are in $K_f$). After `greet(a1)`, the planner is in a state where $\text{inTrans} = a1 \in K_f$ and $\text{greeted}(a1) \in K_f$. The `ask-drink(a1)` action can now be chosen, updating PKS’s knowledge state so that $\text{ordered}(a1) \in K_f$ and $\text{request}(a1) \in K_v$. The next action considered by the planner is `ack-order(a1)`, in particular since $\text{ackOrder}(a1) \notin K_f$. As a result $\text{ackOrder}(a1)$ is added to $K_f$. Consider the `serve(a1, request(a1))` action. Since $\text{inTrans} = a1$ remains in $K_f$, the first precondition of the action is satisfied. Since $\text{ordered}(a1) \in K_f$, the second precondition, $K(\text{ordered}(a1))$, holds. Also, since $\text{request}(a1) \in K_v$, the third precondition $Kv(\text{request}(a1))$ holds (i.e., the value of $\text{request}(a1)$ is known). The fifth precondition is satisfied by the effects of the `ack-order(a1)` action. The remaining two preconditions are trivially satisfied. Thus, $\text{request}(a1)$ acts as a run-time variable whose definite value (i.e., $a1$’s drink order) will become known after action execution. The action updates $K_f$ so that $\text{serve}(a1) \in K_f$, leaving $K_v$ unchanged. Finally, `bye(a1)` is added to the plan resulting in $\text{inTrans} = \text{nil} \in K_f$ and $\text{transEnd}(a1) \in K_f$, satisfying the goal.

### 3.2.2 Ordering a drink with restricted drink choices

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

Consider a second example, where there is again a single agent $a1$ seeking attention but 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 new constants of type `drink`, i.e., `juice`, `water`, and `beer`. Information about the possible drinks is also put into PKS’s $K_x$ database as the formula:

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

In terms of PKS’s knowledge, this restricts the set of possible mappings for $\text{request}(a1)$ (i.e., $a1$ requested one of a juice, a water, or a beer). 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 drink order]
- `branch(request(a1))` [Form branching plan]
  - $K(\text{request}(a1) = \text{juice})$: [If order is juice]
    - `serve(a1, juice)` [Serve juice to $a1$]
  - $K(\text{request}(a1) = \text{water})$: [If order is water]
    - `serve(a1, water)` [Serve water to $a1$]
  - $K(\text{request}(a1) = \text{beer})$: [If order is beer]
    - `serve(a1, beer)` [Serve beer to $a1$]
- `bye(a1)`. [End the transaction]
In this case, a conditional plan is built. After the drink is ordered, the possible values for \texttt{request(a1)} are tested by creating a plan branch for each possible mapping. Each branch considers a state where the planner has definite knowledge of one mapping. E.g., in the first branch \texttt{request(a1)} = \texttt{juice} is assumed to be in the $K_f$ database; in the second branch \texttt{request(a1)} = \texttt{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 \texttt{request(a1)}, combined with $K_c$ knowledge provided by the \texttt{ask-drink} action.) Along each branch, an appropriate \texttt{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?”).

### 3.2.3 Ordering drinks with multiple agents

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

```plaintext
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, \texttt{a1}’s drink order is taken and processed, followed by \texttt{a2}’s order. The \texttt{wait} and \texttt{ack-wait} actions (which aren’t required in the single agent case) are used to defer a transaction with \texttt{a2} and resume it when the transaction with \texttt{a1} has finished. (The \texttt{otherAttnReq} property, which is a derived property defined in terms of \texttt{seeksAttn}, ensures that other agents seeking attention are told to wait before an agent is served.)

One drawback with our current domain encoding is that agents who have been asked to wait are not necessarily served in the order they are deferred. From a task achievement point of view, plans generated with this limitation will still achieve the goal of serving drinks to all agents seeking attention. However, from a social interaction point of view they potentially fail to be appropriate (depending on local pub culture), since some agents may be served before other agents that have been waiting for longer periods of time. Such a scenario is certainly possible in our bartending domain, where the appearance of a new agent is dynamically reported to the planner by the state manager, possibly triggering a replanning operation: the newly built plan might “preempt” an already waiting agent for a newly-arrived agent as the next customer for the bartender to serve. Since
socially appropriate interactions are central to the goals of this work, we are addressing this issue by modifying our domain description to introduce ordering constraints on waiting agents.

3.2.4 When things go wrong: handling low-confidence ASR and overanswering

Once a plan has been built, it is sent for execution by the robot, one action at a time. Each high-level action is divided into speech, head motion, and manipulation behaviours using a simple rule-based system before they are executed in the real world. After execution has started, PKS's execution monitor is used to assess plan correctness by comparing subsequent state reports from the state manager against states predicted by the planner. In the case of disagreement, for instance due to unexpected outcomes like action failure, the planner is invoked to construct a new plan using the sensed state as its new initial state. This method is particularly useful for responding to unexpected responses by agents interacting with the bartender.

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

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

Thus, replanning produces a loop that repeats an action in an attempt to obtain the information the planner requires in order to continue executing the previous plan.

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

ack-order(a1). [Acknowledge a1’s drink order]
serve(a1, request(a1)). [Give the drink to a1]
bye(a1). [End the transaction]

4. Extending the approach to planning interactions

While the basic planning approach handles a number of realistic interactions in the bartending scenario we are also in the process of extending our domain, and the underlying planning approach, to model more complex scenarios. We discuss two of these extensions here.
First, we are seeking to improve coordination between the high-level planner and the natural language understanding module to improve the parsing of expected responses from the user. Since the planner knows what action is being executed, and what information is expected in response in an ideal situation, it can supply “hints” to the natural language input processing module. One way this information can be used is to select a restricted grammar for parsing. For instance, if a \texttt{ask-drink(a1)} action is being executed, the planner knows that an ideal response is a drink request. A restricted grammar might reduce the set of possible responses to the available drink options, plus a set of related drink queries.

Second, we are improving our ability to plan in the presence of uncertainty due to automatic speech recognition (ASR). Currently, any speech input hypotheses other than the top hypothesis is discarded by the natural language processing module. As a result, potentially high-likelihood alternatives to the top hypothesis are ignored by the high-level decision making components of the system, raising the possibility of misunderstanding due to incomplete processing of user utterances. As an alternative approach, we will pass an \textit{n}-best list of processed hypotheses to the state manager for inclusion as part of the state representation. This information will be recorded as a set of alternative fluent interpretations for the utterance in question. In practical terms, the size of the \textit{n}-best list will be determined by the number of entries that account for a significant probability mass in terms of the hypotheses’ associated confidence measures. At the planning level, such information will be represented in PKS’s \textit{K}_k database, which models the planner’s “exclusive or” information. Once such information is recorded in the planner’s knowledge state, PKS can make direct use of this information during planning, introducing extra actions into the plan as necessary to disambiguate between \textit{K}_k alternatives. To aid in this process, we may extend the domain to include additional questions that the bartender can ask to help clarify uncertain beliefs (without necessarily asking the user to simply repeat the last utterance, except possibly as a last resort).

Finally, we are also extending the type of reasoning the planner performs when working which multiagent knowledge, along the lines of (Steedman & Petrick, 2007). In particular, the current version of PKS does not easily work with information about other agents’ beliefs; rather, such information must be encoded using the standard (single agent) tools available to PKS (i.e., its databases and inference algorithm). Instead, a more sophisticated form of planning can be achieved by introducing operators into the planner for directly representing and reasoning with multiagent knowledge conditions. We are extending PKS to generate plans using such operators.\textsuperscript{2}

5. Discussion and Related Work

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

\textsuperscript{2} For the latest development version of PKS, see http://homepages.inf.ed.ac.uk/rpetrick/software/pks/.
The general focus of this work fits into the active research area of *social robotics*: “the study of robots that interact and communicate with themselves, with humans, and with their environment, within the social and cultural structure attached to their roles.” (Ge & Matarić, 2009) Most current social robots play the role of a companion, often in long-term relationships with the user, e.g., (Breazeal, 2005; Dautenhahn, 2007; Castellano et al., 2010). In such contexts, the primary goal for the robot is to engage in social interaction for its own sake, and build relationships with the user: the robot is primarily an interactive partner, and task behaviour is secondary to this overall goal.

We build on this recent work, but address a different style of interaction, which is distinctive in two main ways. First, while existing projects generally consider social interaction as the primary goal, the robot bartender supports social communication in the context of a cooperative, task-based interaction. Second, while most social robotics systems deal primarily with one-on-one interactive situations, the robot bartender must deal with dynamic, multi-party scenarios: people will be constantly entering and leaving the scene, so the robot must constantly choose appropriate social behaviour while interacting with a series of new partners.

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

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

6. Conclusions and Future Work

In this paper we discussed initial work aimed at combining social interaction with task-based action in a dynamic, multiagent bartending domain, using an embodied robot. Action selection uses the off-the-shelf PKS planner, combined with a social state manager and plan monitor. Although this work is preliminary, it has resulted in a working system that has been evaluated with human users.
In subsequent versions of the system, state management will be enhanced to support more complex scenarios: this will involve processing more complex messages from the updated input and output components, including taking into account the associated confidence scores, and also dealing with the more complex state representations that will be required by the updated high-level reasoning system. Dealing with this more complex setting will require defining much more complicated mapping functions. To address this, we will make use of supervised learning techniques trained on data gathered from humans interacting with both real and artificial bartenders, using methods similar to those employed, for example, by (Kapoor et al., 2007) and (Bohus & Horvitz, 2009).

We are currently extending the planning work to more complex bartending scenarios, including agents that can ask questions about drinks, a bartender that can query agents for more information, agents that can order multiple drinks, and situations where the bartender or an agent may terminate an interaction early. We believe a general-purpose planning approach offers a potential solution to the problem of action selection in task-based interactive systems, as an alternative to more specialised approaches, such as those used in many mainstream natural language dialogue systems.

Acknowledgements

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

References


Abstract
This report describes part of UEDIN’s contribution to the ongoing work of general-purpose, knowledge-level planning in the JAMES project. The focus of this document is a description of the current state of proposed heuristic search extensions for the PKS (Planning with Knowledge and Sensing) planner, and its derivative technologies, which will be used and extended throughout the project as part of WP4 (Task 4.3). In this reporting period (M18), we present an overview of the problem of informed search for knowledge-level planning, and describe our current workplan for extending the PKS planner during the next work period of the project. We also highlight other research directions for knowledge-level planning currently in progress which are related to the heuristic search work.

1. Introduction
In this document we describe the state of heuristic search extensions proposed for the knowledge-level planning work by UEDIN in the JAMES project. This work forms part of WP4 (Planning and Reasoning) and, in particular, Task 4.3 (Domain-independent heuristic search for knowledge-level planning). The present report primarily presents the results of initial exploratory studies into informed search methods and how they could be adapted to knowledge-level planning. This work is also closely related to other tasks in WP4, namely Task 4.2 (Plan generation in social state spaces) and Task 4.4 (Domain-dependent heuristics for planning in social spaces; not yet started), and the project-wide integration work and demonstrations of WP7 (System Integration and Evaluation).

High-level planning capabilities in the JAMES project are supplied by the PKS planner (Petrick & Bacchus, 2002; Petrick & Bacchus, 2004), which UEDIN is extending for use in socially interactive domains as part of WP4. PKS is a state-of-the-art knowledge-level planner that constructs plans in the presence of incomplete information. Unlike traditional planners, PKS builds plans at the “knowledge level”, by representing and reasoning about how the planner’s knowledge state changes during plan generation. Actions are specified in a STRIPS-like (Fikes & Nilsson, 1971) manner in terms of action preconditions (state properties that must be true before an action can be executed) and action effects (the changes the action makes to properties of the state). PKS is able to construct conditional plans with sensing actions, and supports numerical reasoning, run-time variables (Etzioni et al., 1992), and features like functions that arise in real-world planning scenarios.

Like most AI planners, PKS operates best in discrete, symbolic state spaces described using logical languages. As a result, work that addresses the problem of integrating planning on real-
world robot platforms often centres around the problem of representation, and how to abstract the capabilities of a robot and its working environment so that it can be put in a suitable form for use by a goal-directed planner. Integration also requires the ability to communicate information between system components. Thus, the design of a planning system often has to take into consideration external concerns, to ensure proper interoperability with modules that aren’t traditionally considered in pure theoretical planning settings.

PKS was successfully used on a previous EU-funded FP6 project called PACO-PLUS. On that project, integration work successfully established a link between PKS and a humanoid robot platform called ARMAR from the Karlsruhe Institute of Technology (KIT) in Germany. In JAMES, as part of WP4, we are building on these past successes to extend such robot-planner integration even further. In particular, since we will address more complex challenges in the social domains we consider in JAMES, the high-level planning capabilities must be extended on a number of levels. First, representations must be extended to improve our ability to model real-world problems at the planning level (Task 4.1). This will particularly be important for the dialogue planning work which is required in our social domains. However, changes to the representation alone are not enough. The plan generation methods themselves (Task 4.2) must be improved to take advantage of the new features of an extended representation language. Furthermore, as the complexity of the planning problem increases, we must work harder to find methods to overcome the difficulties arising from increased domain size leading to increased planning times (Task 4.3 and Task 4.4). Methods such as informed search control (Task 4.3) seek to address these concerns by improving the efficiency of the plan generation process itself.

In the remainder of this document we will focus on this final concern, and present a brief overview of our plans for adapting informed (heuristic) search techniques to knowledge-level planning in JAMES, and how these plans will potentially affect other extensions and integration work involving PKS in the future.

2. Background to knowledge-level planning and informed search

The ability to reason and plan is essential for an intelligent agent acting in a dynamic and incompletely known world—such as the robot scenarios in JAMES. Achieving goals under such conditions often requires complex forward deliberation that cannot easily be achieved by simply reacting to a situation without considering the long term consequences of a course of action.

The problem of planning has been extensively studied in artificial intelligence. One of the most influential approaches has been STRIPS (Fikes & Nilsson, 1971), a representation language that reduces the problem of specifying actions to that of describing an action’s preconditions (i.e., the qualification problem) apart from an action’s effects. A solution to the frame problem (McCarthy & Hayes, 1969) is also captured in STRIPS as a persistence condition on properties unaffected by an action. Although STRIPS traditionally makes certain unrealistic assumptions about the nature of a planning domain, namely that actions are deterministic and world states are completely known, it nevertheless forms the core of PDDL (McDermott et al., 1998), the standard language of many modern planners and the language of the International Planning Competition (ICAPS, 2008). The success of this representation has led to the development of many modern STRIPS-based planners
that use techniques like heuristic search (Hoffmann & Nebel, 2001) or compilation (Palacios & Geffner, 2007) to scale to large problem instances.

While pure STRIPS-based planners are not directly relevant to JAMES, due to their representational limitations, alternate attempts to solve more general planning problems often use variants of STRIPS as their underlying representations (e.g., Cimatti & Roveri, 2000; Bonet & Geffner, 2001)). In JAMES, we use an approach based on knowledge-level planning, an instance of the general problem of planning with incomplete information and sensing (e.g., Peot & Smith, 1992)—i.e., planning with incomplete states and observational actions that return information about the world state. In contrast to more traditional approaches which build plans by reasoning about how actions affect the state of the world, the knowledge-level approach builds plans at a more abstract level, by directly representing and reasoning about the incompleteness of the planner’s knowledge and how that knowledge changes during planning. By abstracting the type of reasoning used during plan generation, this approach has the potential to generate quite complex plans very efficiently. This approach also has links to the knowledge representation and reasoning community and those logical accounts that restrict epistemic expressiveness for tractable reasoning (e.g., Funge, 1998; Demolombe & Pozos Parra, 2000; Son & Baral, 2001; Liu & Levesque, 2005; Vassos & Levesque, 2007)), as an alternative to more traditional accounts of knowledge and action (e.g., Moore, 1985; Scherl & Levesque, 2003)). A similar idea to this is also explored in the EU FP7 CogX project (ICT-215181).

One of the most advanced knowledge-level planners is PKS (“Planning with Knowledge and Sensing”) (Petrick & Bacchus, 2002; Petrick & Bacchus, 2004), a contingent planner that constructs plans with incomplete information and sensing. Unlike most planners, PKS uses an extended STRIPS representation but restricts the types of knowledge it can represent in exchange for more efficient reasoning. PKS is particularly adept at modelling knowledge-level changes resulting from sensing actions, which arise in many challenging planning scenarios. PKS also supports many features needed for real-world planning, such as the representation of functional information, numerical reasoning, and run-time variables (Etzioni et al., 1992). In addition to JAMES, PKS has been successfully used in real-world environments such as the robot kitchen domain in the FP6 PACO-PLUS project, and the PACO-PLUS follow-on project called Xperience (Grant No. 270273).

Reasoning about sensing actions allows planners to control certain types of indefinite information that arise in the world. However, related approaches to planning under uncertainty also attempt to manage other types of nondeterminism, including actions with noisy effects or probabilistic outcomes. Currently, one of the most successful techniques is to employ rapid replanning methods that make use of advances in heuristic search (Hoffmann & Nebel, 2001), a technique we are extending to PKS in JAMES. Planners that employ these ideas, like FF-Replan (Yoon et al., 2007), have been successfully used in domains such as those in the probabilistic track of the International Planning Competition (ICAPS, 2008).

In JAMES, planning will also be used for the purpose of natural language dialogue, and the demands of that problem will influence the structure and requirements of a suitable planning system. The tasks of natural language generation and reasoning about dialogue have long traditions of using planning approaches. Early approaches to generation as planning (e.g., Perrault & Allen, 1980; Appelt, 1985; Hovy, 1988; Young & Moore, 1994)) focused primarily on high-level structures,
such as speech acts and discourse relations, but suffered due to the inefficiency of the planners available at the time. As a result, recent mainstream research has tended to segregate task planning from discourse and dialogue planning, capturing the latter with more specialised approaches such as finite state machines, information state approaches, speech-act theories, dialogue games, or theories of textual coherence (Lambert & Carberry, 1991; Traum & Allen, 1992; Green & Carberry, 1994; Young & Moore, 1994; Chu-Carroll & Carberry, 1995; Matheson et al., 2000; Beun, 2001; Asher & Lascarides, 2003; Maudet, 2004).

There has also been a renewed interest in applying modern planning techniques to problems in NLG, such as sentence planning (Koller & Stone, 2007), instruction giving (Koller & Petrick, 2008), and accommodation (Benotti, 2008). The idea of viewing interaction management as a planning problem has also been revisited, for instance by identifying the problem of planning conversational moves as an instance of the general problem of planning with incomplete information and sensing actions (Stone, 2000). This view is also implicit in early “beliefs, desires and intentions” (BDI)-based approaches, e.g., (Litman & Allen, 1987; Bratman et al., 1988; Cohen & Levesque, 1990; Grosz & Sidner, 1990). Thus, certain types of communicative actions (e.g., speech acts like “asking” and “telling”) are treated as ordinary sensing actions that return otherwise unknown information to the agent. Initial work using the knowledge-level PKS planner explored this connection (Steedman & Petrick, 2007), but fell short of implementing a robust tool that could leverage this relationship for efficient dialogue planning. A related approach from the FP6 CoSy project (Brenner & Kruijf-Korbayová, 2008) also attempted to manage dialogues by interleaving planning and execution, but failed to solve the consequent problem of deciding when best to commit to plan execution versus plan construction. Thus, many planning approaches are promising, but not yet fully mature, and fall outside the mainstream of most recent NLG and dialogue work.

The common thread in all these approaches is that they seek to overcome the computational challenges inherent in complex domains. The most successful approach arising from the planning community has been the use of heuristic or informed search techniques for this task.

3. Informed search strategies

Most modern planning systems rely on the process of search to find a sequence of actions that transforms an initial state into a state satisfying the goal of the planning domain. However, a search through a large state space of the kind that typically arises in many robot domains is problematic: the search process may take a large amount of time to complete, if it completes at all, due to the resource overhead required to explore large state spaces. As a result, most modern planners employ some form of informed search or heuristic search (either directly or indirectly) in an attempt to gain leverage on the structure of a given search problem, in order to improve plan generation times.

Surveying the recent literature on informed search in the planning community shows that there are two main strategies employed by existing planning systems:

1. **Problem relaxation:** In problem relaxation, rather than solving the original problem in the original search space, the problem space is abstracted in some way, typically by considering a subset or generalisation of the original planning space. The relaxed problem, which is typically a simpler problem, is solved and then used as an estimate for guiding the search in
the original problem. In this case, solving the relaxed problem must be done in an efficient manner so that the planner has time to apply its results to the original problem with a net gain in overall planning time.

2. **Compilation methods:** The idea of compilation usually refers to the process whereby the original problem domain is transformed into an alternate, usually simpler problem instance, and an existing (potentially more efficient) tool is then used to solve the simpler problem. The solution to the compiled problem is then applied to the original problem, sometimes through a process that modifies the generated plan in some predefined way. Such methods may lead to exact compilations where a problem can be provably reduced to a simpler problem instance, or approximate compilations where the reduced problem is not provably “exact”, but nevertheless produces a solution that can be used to solve the original problem.

(A third approach also exists, which is namely a hybrid approach that combines the two approaches. For instance, as part of the problem relaxation approach a compilation method might be employed to simplify the existing problem first before relaxing it at the search space level.)

Examples of both approaches exist in the planning literature and have been shown to be successful in practice. The most famous planner based on problem relaxation is FastForward (FF) (Hoffmann & Nebel, 2001) which uses a relaxation technique that ignores the “delete lists” in a STRIPS action description (i.e., the properties of the world that become false when an action is applied). In this case, FF generates over-specified states during its search which provide a lower bound on the number of steps a plan requires before a domain property could possibly become true. This estimate, which is generated as part of a structure called a planning graph (Blum & Furst, 1997), is then used to inform the original search problem. In many of the standard planning benchmarks, such as those from the International Planning Competition, this heuristic has been shown to be quite effective and had lead to impressive performance on these domains. The success of this approach has also influenced a number of subsequent planners which have attempted to adapt the FF approach for improved performance. The task of designing new relaxation heuristics for search remains an active area of research within the planning community.

The most influential work on compilation techniques within the planning community is that of (Palacios & Geffner, 2007), which investigate the problem of converting a conformant planning problem (a problem with incomplete information but no sensing actions) into a classical problem (a problem with complete information) that can then be solved using an off-the-shelf planner like FF. This is done by converting the original problem (described in terms of world-level properties) to its knowledge level counterpart, by introducing new fluents into the domain model. When viewing the problem at the knowledge level, a closed-world view of the fluents can be used, allowing a planner like FF to be employed. Once a solution to the compiled problem is found, it is translated back to the original problem, in some cases by expanding or removing certain actions in the process. Compilation approaches are promising because they allow existing tools to be leveraged to solve subproblems. However, usually the compilation imposes restrictions on the types of problems that such techniques can be used on. For those domains for which compilation techniques do not apply, approximate solutions might be possible, or else other techniques must be adopted.
4. Informed search strategies for PKS

There are two immediate research questions for the knowledge-level planning component on JAMES concerning informed search: (1) which technique should we use, and (2) what are the requirements for adapting that technique to a planner like PKS? First, we observe that neither technique offers an immediate solution that can simply be “lifted” into a planner like PKS. First, the state spaces that PKS uses are different from those of planners like FF, and the relaxation technique of ignoring delete lists does not immediately transfer to the knowledge-level situation. Second, the representation language used by PKS is substantially more expressive than those used in the planners that are considered by Palacios and Geffner. As a result, the actual compilation algorithm presented in (Palacios & Geffner, 2007), and other similar work, does not immediately translate to PKS.

However, the situation is not all bad news. First, while the FF relaxation technique does not directly apply to PKS, relaxation techniques that are applicable should be possible. The challenge in this case is identifying an approach that solves an abstraction of the PKS planning problem quickly and accurately, and that can be used to inform a heuristic search process. This remains a challenging but essential area of research for the knowledge-level planning agenda at UEDIN.

Second, while the technique of (Palacios & Geffner, 2007) also doesn’t immediately translate to PKS, a comparable approach used in earlier work, which does consider a subset of the PKS representation, is applicable (Petrick & Levesque, 2002; Petrick, 2006). This approach uses ideas similar to the work Palacios and Geffner but is motivated by the logical language of the situation calculus, used in the knowledge representation community. As a result, this approach provides a much needed logical theory, which is not clearly present in (Palacios & Geffner, 2007).

As a result, we are adopting a two-pronged strategy for PKS. First, we are exploring relaxation techniques that can be used directly with the state spaces that arise in the PKS planner. This approach consists of a substantial implementation stage (which is currently underway) to restructure the codebase of the PKS planner which implements search, to take advantage of informed heuristic estimates. The code restructuring will also provide a testbed whereby the PKS planning algorithm can be used with various informed search techniques, to evaluate which are more effective in the JAMES experimental domains.

Second, we are also extending the work of (Petrick & Levesque, 2002; Petrick, 2006) to provide a theoretical understanding of how compilation can be used in PKS. Part of this work will out of necessity revisit the work of (Palacios & Geffner, 2007) and re-interpret it in terms of the work of Petrick and Levesque. We believe that many of the techniques presented in (Petrick & Levesque, 2002) can be adapted to a more practical planning setting, but also that some of the insights of (Palacios & Geffner, 2007) have a role in improving the existing theoretical models in the situation calculus. We also believe that the revised compilation technique can then be used in conjunction with a PKS-specific relaxation method, combining the two approaches. However, more research is needed to investigate what form such a technique might take.

5. Example: compiling open world PKS to closed world FF

As an example of our initial thinking on the use of compilation techniques, we apply the approach of (Petrick & Levesque, 2002; Petrick, 2006) to compile a simple PKS planning domain into a closed
world form that is solvable using the off-the-shelf FF planner. An example is given using a standard planning benchmark problem, the Bomb-in-the-Toilet domain.

**Original PKS Bomb-in-the-Toilet domain**

Here we give an encoding of the original Bomb-in-the-Toilet domain in PKS syntax:

<table>
<thead>
<tr>
<th>Action</th>
<th>Precondition</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>dunk(p, t)</td>
<td>K(package(p))</td>
<td>add(K_f, disarmed(p))</td>
</tr>
<tr>
<td>K(toilet(t))</td>
<td></td>
<td>add(K_f, clogged(t))</td>
</tr>
<tr>
<td>K(~clogged(t))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flush(t)</td>
<td>K(toilet(t))</td>
<td>add(K_f, ~clogged(t))</td>
</tr>
</tbody>
</table>

This domain consists of two actions: dunk, which dunks a particular package into a toilet, and flush, which flushes a toilet. Dunking a package into a toilet disarms a package but clogs the toilet. Flushing a toilet unclogs the toilet. Given a set of p packages and t toilets, the goal of the domain is to ensure that all packages are disarmed. No information is provided as to which packages contain bombs so a successful plan involves reasoning that all packages must be dunked.

**Compiled Bomb-in-the-Toilet domain**

We now show the encoding of the compiled Bomb-in-the-Toilet domain using the techniques of (Petrick & Levesque, 2002; Petrick, 2006). Note that for each fluent P in the original PKS domain, a pair of knowledge fluents, KP and K~P are introduced into the compiled version of the problem. In this case, the set of knowledge fluents forms a type of “closed world” problem which can be solved using classical planning techniques. In other words, the knowledge fluents can be treated as ordinary fluents in a standard planning domain without explicit reasoning about incomplete information:

<table>
<thead>
<tr>
<th>Action</th>
<th>Precondition</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>dunk(p, t)</td>
<td>K(package(p))</td>
<td><del>K</del>disarmed(p)</td>
</tr>
<tr>
<td>K(toilet(t))</td>
<td></td>
<td><del>K</del>clogged(t)</td>
</tr>
<tr>
<td>K~clogged(t)</td>
<td></td>
<td>Kdisarmed(p)</td>
</tr>
<tr>
<td>flush(t)</td>
<td>K(toilet(t))</td>
<td>~Kclogged(t)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K~clogged(t)</td>
</tr>
</tbody>
</table>

**Run times**

Finally, we give some early results on solving the compiled version of the PKS domain using the FF planner. In this case the running time is significantly faster using the optimised FF approach, based on heuristic search. Moreover, the quality of the plans has also improved (i.e., the plans are shorter) mostly due to extraneous flush actions that have been removed.
6. Proposed tasks towards heuristic search in PKS

In order to breakdown the task into more manageable parts, we identify three core areas of work that are necessary for implementing a heuristic search strategy on PKS: literature studies of existing methods, development of an adapted theory for knowledge-level spaces, and the eventual implementation and evaluation of these methods in PKS.

<table>
<thead>
<tr>
<th>Task</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Studies</strong></td>
<td></td>
</tr>
<tr>
<td>Initial informed search studies</td>
<td>Complete</td>
</tr>
<tr>
<td>Intermediate studies</td>
<td>Underway</td>
</tr>
<tr>
<td><strong>Theory development</strong></td>
<td></td>
</tr>
<tr>
<td>Initial heuristic search strategies</td>
<td>Underway</td>
</tr>
<tr>
<td>Initial compilation techniques</td>
<td>Underway</td>
</tr>
<tr>
<td>Intermediate search and compilation techniques</td>
<td>Not yet started</td>
</tr>
<tr>
<td>Integration with domain dependent techniques</td>
<td>Not yet started</td>
</tr>
<tr>
<td><strong>Implementation and evaluation</strong></td>
<td></td>
</tr>
<tr>
<td>Extension of existing search framework</td>
<td>Underway</td>
</tr>
<tr>
<td>Implementation of initial/intermediate search techniques</td>
<td>Starting soon</td>
</tr>
<tr>
<td>Evaluation of extended search methods</td>
<td>Ongoing</td>
</tr>
</tbody>
</table>

We expect to produce an initial version of the heuristic search module in time for the Year 2 system evaluation (currently scheduled for early 2013), however, its inclusion in the actual evaluation system will depend on its robustness. In the case that we fail to have a working module by that time, a series of offline experiments will be performed in time for the Year 2 project review.
7. Implications for related research activities

Due to the central nature of search to the plan generation process, the addition of heuristic search in PKS has the potential to affect other research activities involving the planner as part of WP4. We conclude this document by highlighting some of these activities and outline the expected impact of heuristic search on these tasks.

- **Extended representations for planning and reasoning (Task 4.1)**: Any addition to the planner's representation language has the potential to change the planner's search space, meaning the techniques used to search over this space must potentially change. The initial work on heuristic search will focus on interoperability with the basic PKS representation language, and domains modelled in this language.

- **Plan generation in social state spaces (Task 4.2)**: One major planned extension to PKS involves modelling multiagent knowledge for the purpose of dialogue planning. This extension will involve additions to the basic PKS representation language to allow agent-based knowledge assertions to be modelled. Due to time constraints, our current plans do not include extending heuristic search to the new state spaces that will arise from this language. Time permitting, an initial theoretical study into the form of such heuristics may be undertaken.

- **Domain-dependent heuristics for planning in social spaces (Task 4.4)**: Our current plans for Task 4.4, which will not begin until Year 3 of the project, involve the addition of control knowledge in the form of domain constraints, expressed in PKS's current representation language. As a result, no changes should be necessary on the heuristic search side to accommodate the work of this task.

- **System integration and evaluation (WP7)**: The addition of heuristic search techniques in the planner will not change the planner’s interface to other modules in the wider JAMES system. As a result, integration activities will be unaffected. However, this work will potentially have an impact on system evaluation activities. To properly evaluate the effectiveness of the heuristic search techniques in practice, an evaluation of the new version of the planner compared with the original version, is desirable. While we plan to perform a series of offline experiments in JAMES-style domains, a small study with real human users using the mainline JAMES system would also provide a useful result.
References


