The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

Finding and Exploiting Goal Opportunities in Real-time during Plan Execution

Paul Schermerhorn[†], J. Benton[‡], Matthias Scheutz[†], Kartik Talamadupula[‡], and Subbarao Kambhampati[‡]

[†]Cognitive Science Program
Indiana University
Bloomington, IN 47406, USA
{pscherme,mscheutz}@indiana.edu

[‡]Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85281, USA
{j.benton,krt,rao}@asu.edu

Abstract— **Autonomous robots that operate in real-world domains face multiple challenges that make planning and goal selection difficult. Not only must planning and execution occur in real time, newly acquired knowledge can invalidate previous plans, and goals and their utilities can change during plan execution. However, these events can also provide opportunities, if the architecture is designed to react appropriately. We present here an architecture that integrates the SapaReplan planner with the** DIARC **robot architecture, allowing the architecture to react dynamically to changes in the robot's goal structures.**

I. INTRODUCTION

Planning architectures for intelligent autonomous robots that operate in real-world domains and under time pressure must make decisions about which goals to pursue at any given point in time. In particular, they need to address at least six major challenges: (1) the world is partially observable, (2) actions are non-deterministic, (3) the prior knowledge about the planning domain is limited, (4) knowledge acquisition for planning is non-monotonic (i.e., new world states can invalidate previously found plans), (5) planning and plan execution are subject to real-time constraints, (6) goals and goal utilities (including action costs and timeouts) can dynamically change during plan execution. Additional practical challenges include varying action durations and action costs, interleaving of planning and action execution, or identification of world states used for planning across time and contexts.

While the first three challenges are common to many planning applications [1], [2], [3], the second three are typical of robotic scenarios and impose architectural constraints on robotic planning architectures. For example, they all can trigger replanning: (4) if a new percept invalidates a pre-condition of a plan operator to be executed, (5) if delays in executing a plan operator change the expected post-conditions, and (6) if a new goal is introduced during plan execution. Yet, these challenges also present opportunities: (4) new perceptions might lead to opportunities to achieve goals (i.e., new planning paths to goal states that did not exist before), (5) the real-time nature of actions allows for good estimates for action costs and the achievability of possible plan sequences given deadlines, and (6) dynamic adaptations to the robot's goal structures (in light of changes in the environment) might allow it to achieve better task performance compared to static structures. And even though planning algorithms have been used extensively in robotics [4], there are currently no integrated robotic planning architectures with mechanisms for (1) generating counterfactual "what-if" scenarios using background knowledge about the planning domain to detect possible action sequences that can lead to goal opportunities, (2) systematically scheduling perception actions as part of a plan that can be carried out by the robot in parallel to plan actions in order to be able to detect goal opportunities, and (3) dynamic goal management and replanning triggered by perceptions from monitoring processes to exploit detected opportunities, all in light of real-time planning constraints and goal deadlines.

In this paper, we propose such an integrated robotic planning architecture, which thus addresses all six challenges. Specifically, we propose an integration of an extended version of the planner SapaPS [5] called SapaReplan [6] into the robotic DIARC architecture [7], together with additional architectural mechanisms for the scheduling of perceptions, which leads to active monitoring of the robot's environment, and notification of the planner when opportunities are detected, which leads to dynamic adaptations to the robot's goal structures. We start with an intuitive example from a search and rescue domain motivating the proposed architectural mechanisms and then formally present the planning problem and the proposed extensions to our previous planning algorithms. Next, we describe how these new mechanisms were integrated into our existing DIARC architecture and implemented in our ADE robotic middleware [8], [9]. We then demonstrate the utility and functionality of these mechanisms in an evaluation experiment, where we systematically vary environmental conditions and timeouts that dynamically affect goal utilities. The results show that the robot is able to change its plan given its knowledge and time constraints. We conclude with a brief summary and suggestions for future work.

II. MOTIVATION

Suppose that a medical support robot, as part of a search and rescue mission, has a *hard goal* to deliver critical drugs needed to treat injured people. While executing a plan to achieve this goal, the robot needs to traverse a corridor that has rooms leading off on either side. The robot also has the knowledge that other wounded humans might be in those rooms, and it has a *soft goal* to report the location of any wounded humans it encounters on the way. The difference between the soft and the hard goal is that the hard goal has

to be met no matter what (hence the planner has to plan accordingly), while the soft goal should be met if possible, but the robot is allowed to fail to meet it. In particular, these soft goals might not even be reachable from the current state of the world. We call this subclass of soft goals "sleeper goals", because there might be a "wake-up" condition where a change in the world state will open up a path to the goal. However, even given such a path, it still may not be worth it to pursue the goal. For instance, if the robot had a time limit to deliver the drugs to the injured people, it may be that it must ignore rooms despite their potential reward because it would cause the robot to miss its deadline. In essence, the agent faces two coupled problems: it must first identify the opportunities that may lead to succeeding at an objective and then decide, given the opportunity, whether it is worth pursuing the goal.

To solve the first problem, we present a framework and methodology that allows problem domain modelers to specify background knowledge regarding potential opportunities by explicitly representing known background facts about sensory data. The background knowledge can give goal definitions, which enables an online planner to reason about the possible reward that pursuing objectives may give. More specifically, the planner generates counterfactual "what-if" scenarios over the background knowledge. Using these counterfactuals, it can identify opportunities that may lead to objectives and simultaneously reason about the reward and cost of these objectives. In our example, the planner would be given the sleeper goal "report location of wounded humans" and the knowledge that "wounded humans are in rooms". Though the sleeper goal has no path to it initially, counterfactual reasoning would allow the planner to see the potential for satisfying the goal of reporting humans once rooms are found.

To handle decisions about whether to pursue goals, we use recent innovations from the automated planning research community. In particular, we employ the techniques found in the planner SapaPS [5], [10] for quickly solving problems that involve actions with cost and *soft* goal objectives with associated reward. Upon updating its internal state, the agent casts its situation into an automated planning problem, where a planner provides counterfactual generation and reasons as to whether an opportunity for reward exists. If one does, the planner can determine whether it is worth it to pursue a goal or ignore it. In the case of discovering rooms, the planner can reason about whether a detour to look for a human in the room is acceptable – and this depends upon a variety of factors, such as the reward of the goal and the deadline to be at the end of the hallway (a hard goal).

III. PROBLEM REPRESENTATION

The planner operates on a partial satisfaction planning model expanded to allow for temporal actions and events. The expanded partial satisfaction planning problem is defined as a tuple $\Pi = \langle O, \omega, T, I, G, E, t_c \rangle$ where T represents a set of predicate templates that can be *grounded* to *typed* objects O into a set of boolean fluents F, ω is a set of operator templates that are *grounded* into a set of actions A and the initial state I is defined by the values of F. The goal set G is defined by a partial set of F where each $g \in G$ can be marked as either *hard* (i.e., must be achieved) or *soft* (i.e., can remain unachieved) and has a reward $u(g)$ that indicates its value of achievement. An action $a \in A$ provides a set of conditions $pre(a)$, an add list $add(a)$ which indicates the facts added by a, a delete list $del(a)$ indicating the facts deleted by a, and a cost $c(a)$ indicating the cost of executing a. Given a state of the world S, applying an action a is defined as $(S \bigcup add(a)) \setminus del(a)$ iff $pre(a) \subseteq S$. We expand on this model by including a temporal component to actions, $dur(a)$ indicating the duration of a (i.e., how long it takes to complete a after starting it). Actions may be executed concurrently but in a limited fashion, following the model of the planner Temporal Graphplan (TGP) [11]. Each goal $g \in G$ may have a deadline $dl(g)$ associated with it. Goals with deadlines must be true at their specified time and, to simplify our description, must *remain* true to be considered achieved. E represents an event queue of changes in state (i.e., a list of facts that change from true to false or vice versa at given times) and t_c is the "current" time stamp to plan from.

The objective of partial satisfaction planning is to find the plan with the best possible *net benefit*. A plan $\mathcal{P}$ is a solution if and only if the execution of the actions in the plan satisfies all hard goals in the problem. We define *net benefit* as the difference between the total cost of actions used in a plan and the total reward for achieving the goals. More formally, our objective is to maximize $\sum_{g \in G'} u(g) - \sum_{a \in \mathcal{P}} c(a)$ where G' is the set of goals achieved by a plan $\mathcal{P}$.

A. Problem Updates

In an online setting, changes to state may come at any moment. In our system, updates can add objects or change any aspect of the grounded problem (i.e., those aspects specified by values of F) except the set of actions A. Therefore, an update is the tuple $v = \langle O, I, G, E, t_c \rangle$. The new objects in O instantiate a new ground problem over the operator and predicate templates, and therefore I, G, and E can specify values for these new fluents and actions.

B. Specifying Goals with Background Knowledge

To successfully reason about and actively seek sleeper goals, we need to express knowledge we have about them. As such, the system provides the ability to relate sensing of particular facts to the goals themselves. The planner receives background knowledge through a description of the properties to be sensed. Though our planner operates over ground actions and fluents, we need to specify *lifted* background knowledge in a form that can be ground with the rest of the problem. In some respects, knowledge about sleeper goals is similar to local closed world rules as defined by Etzioni, et al. [12]. Formally, a background rule is a tuple $\mathcal{B} = \langle S, R, V, \gamma \rangle$. S is a typed variable representing the object to be sensed. R is a first order formula formed as $\forall Q_0, \ldots, Q_n \; \exists \; S | r_0 \wedge \ldots \wedge r_m$ where $Q_0, \ldots, Q_n$ are

typed object variables and $r_0 \wedge \ldots \wedge r_m$ is a conjunctive formula and each r_i indicates information about S. $V \in T$ is an ungrounded predicate indicating that knowledge has been retrieved with regard to the object S and γ is a new goal on S.

For instance, in our example we would want a rule pertaining to humans being in rooms, $\mathcal{B} = \langle h, \forall r \exists h |$ *(in h r)*, *(sensed_for h r)*, *(report h r)*$\rangle$, where h is of type human, r of type room, and $u($*(report h r)*$) = 100$.

IV. PLANNING ARCHITECTURE

For this work, we build on the SapaReplan [6] planning architecture,which is based on the planner SapaPS [10] and allows for problem and goal updates [6]. The architecture consists of two components, a *monitor* which waits for and processes problem and goal updates and a *planner*, which solves for the new updates as they arrive. The *planner* component has been extended to perform counterfactual reasoning over the background knowledge given to it.

A. Processing Updates

In our system, the monitor's role is to process sensory information and to choose goals for the planner to achieve (i.e., it performs *objective selection*). When an *update* arrives, the monitor immediately updates the problem representation and signals the planner. The planner then processes the new updates and generates counterfactuals. Afterwards, the monitor performs a heuristic goal selection process that involves propagating costs on a temporal planning graph structure (one borrowed from the planner SapaPS [5], [10]).

Counterfactuals. Given a current problem definition, a counterfactual is generated by unifying constant objects over the background knowledge and adding a counterfactual (i.e., simulated) object for every unification on the universally quantified variables. These counterfactuals can also be chained together to allow for compound scenarios that involve multiple background rules. In other words, a counterfactual generates the assumption that an object being sensed exists, and that certain facts and goals, as specified by background knowledge, also exist. In the example, for every room that is in the known problem, the planner would assume that a human exists, generating the fact *(in h* r)* and the goal *(report h* r)* where *h** is the new counterfactual object and *r* is a known room. The planner can then plan over these facts, allowing it to assume that opportunities for reward exist. When the fact *(sensed_for h r)* is generated for the room *r*, the counterfactual information is removed from the problem.

Objective Selection. The monitor performs objective selection using a temporal planning graph technique taken from the planner SapaPS [5], [10]. Objective selection relies on the reachability and cost analysis provided by a temporal planning graph (TPG). The problem is relaxed by removing the delete lists of actions. After reachability of a goal has been determined, a relaxed plan is found from the goal set. Each goal is analyzed by finding the difference between the reward for the goal and its estimated cost in the relaxed plan (i.e., the actions used to reach it). If the goal is soft and the number falls below a threshold (in our case, 0), then the goal is removed from consideration. Though we perform up-front objective selection, the planner's search process is aware of whether goals are hard or soft and may eventually discover that the achievement of a pre-selected goal provides a poor quality plan or prevents the achievement of a hard goal.

B. Planner

SapaReplan is based on SapaPS [5], [10], which uses a forward chaining weighted-A*, decision epoch search for handling temporal actions and partial satisfaction planning problems. Additionally, it allows for problem updates and can be run in an online environment. The planner uses a heuristic similar to the objective selection heuristic, where a temporal planning graph is used to extract a relaxed version of the problem. Goals can then be pruned from the heuristic value at each state.

V. PLANNER INTEGRATION

The SapaReplan planner is integrated into the DIARC robotic architecture for human-robot interaction[1] and implemented in the ADE robot development infrastructure [9].[2] In ADE, each functional component is implemented as an *ADE server* that registers its functionality with an *ADE registry* (essentially a white pages component used to help locate resources, enforce security policies, and provide fault tolerance and error recovery to the system). When an ADE server requires the services of another component, it requests a reference to that component from the registry, which verifies that it has permission to access the component and provides the information needed for the two components to communicate directly. The "client" ADE server can then access the (possibly remote) component's services via a JAVA RMI interface.

The SapaReplan planner was integrated into the ADE infrastructure by creating an ADE server that wraps the existing planner. This new component does not manage action execution; that functionality exists already in the ADE *Goal Manager*, a goal-based action selection and management system that uses procedural knowledge in the form of *action scripts* to achieve goals [7]. These scripts can be simple or complex, composed of (potentially recursive) calls to other scripts. When a new goal is instantiated, the goal manager retrieves a script that can achieve the goal and starts an *Action Interpreter* to execute the script (possibly in parallel with other action interpreters servicing other goals). Simple script elements are carried out directly in an action manager and typically result in the invocation of remote methods in other ADE servers (e.g., to initiate movement in a robot server).

[1] DIARC combines higher-level cognitive tasks, such as natural language understanding, with lower-level tasks, such as navigation, perceptual processing, and speech production [13]. DIARC has served as a research platform for several human subject experiments (e.g., [7], [13]).

[2] ADE combines support for the development of complex agent architectures with the infrastructure of a multi-agent system that allows for the distribution of architectural components over multiple computational hosts [9].

Note that there is no problem-solving built into the goal manager or action interpreter, so if there is no script available that achieves the specified goal, or actions are missing in a complex script (e.g., to deal with a failure condition), then the action interpreter simply fails, causing a failure to achieve the goal. The integration of the planner thus fulfills a dual role: it provides DIARC with the problem-solving capabilities of a standard planner (for finding action sequences to achieve goals for which no prior procedural knowledge exists) and it allows DIARC to use the specific opportunistic planning based on counterfactual reasoning and perceptual monitoring that is presented in this paper.

The new SapaReplan planner server starts the SapaReplan problem update monitor, specifies the planning domain, and (when applicable) presents the planner with an initial problem description. The set of percept types that are of interest to the planner are sent to the ADE *Action Manager* (via the "attend" mechanism, described below), and the planner server enters its main execution loop, in which it retrieves new plans from the planner and sends new percepts and goal status updates to the planner.

A special "attend" primitive has been defined in the goal manager to allow servers (such as the planner server) to specify which percepts are of interest. This will "focus attention" on those types of percepts by causing the instantiation in the goal manager of monitoring processes that communicate with other ADE servers (e.g., the vision server to detect targets of interest that are visually perceivable, the laser rangefinder server to detect doorways, which are detected in the range finder profile, etc.). In the case of the SapaReplan planner server, the percept types of interest are those that could prompt opportunistic replanning (e.g., detection of a new doorway might trigger a new plan to explore the room). A wide variety of percept types are available from various ADE servers; a subset of those most relevant to the present study are:

- *(robot orientation ?heading velocity ?vel location ?loc)* provides the current orientation, velocity, and location of the robot
- *(landmark ?name type ?t heading ?dir distance ?dist)* provides information about a perceived landmark with the given label $?name$ of type $?t$ (e.g., chair, table, etc.), its direction relative to the robot's heading in degrees and its distance in meters
- *(box ?name color ?value heading ?dir distance ?dist)* provides information about a perceived box, including its label, color, direction relative to the robot's heading in degrees, and distance in meters
- *(doorway ?name heading ?dir distance ?dist)* provides information about a perceived doorway, including its direction relative to the robot's heading in degrees and distance in meters

When the monitoring process of the goal manager detects a percept in its attend list, it constructs a plan update and sends it to the planner via the planner server's update method.

Updates from the goal manager can trigger the planner to replan to take advantage of detected opportunities. Plans are generated in the form of ADE action scripts, which are directly executable by an action interpreter in the goal manager. Some examples of ADE scripts (both simple and complex) available to the planner for creating plans are:

- *(look-for ?t)* scans the room for percepts of type *?t* while turning 360 degrees
- *(move-to ?location)* moves to the location specified (e.g., as indicated by a landmark, e.g., *(move-to chair1)*)
- *(turn-to ?location)* turns to face the location specified (e.g., *(turn-to box3)*)
- *(move-through ?doorway)* moves through the specified doorway
- *(report ?object ?c1 ...)* reports the given characteristics *?c1*, etc., (e.g., location, color) of the *?object*

The new plan/script is passed to the goal manager, which oversees its execution. When a plan completes, its post-conditions are sent to the planner server as goal status updates. If a newly encountered percept triggers replanning, the previously executing plan is discarded and the new plan takes its place.

Hence, the SapaReplan planner server can provide problem-solving capabilities to architectures constructed in the ADE infrastructure. When no pre-defined script is available to achieve a goal, the planner can create the script. This newly-generated script can then potentially be saved and reused in similar future contexts.

TABLE I

RESULTS OF TRIAL RUNS FOR VARIOUS TIME LIMITS

Time Limit	Room1	Report	Room2	Report	Room3	Report	Hard goal
30	-	-	-	-	-	-	Failure
60	Pass	-	Pass	-	Pass	-	Success
90	Enter	Yes	Pass	-	Pass	-	Success
120	Enter	Yes	Enter	No	Pass	-	Success
160	Enter	Yes	Enter	No	Enter	No	Success

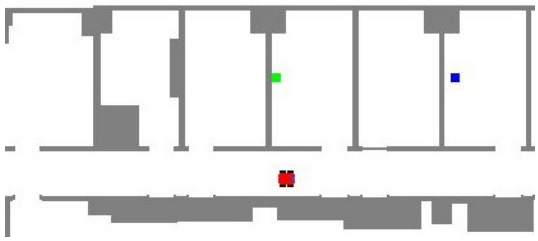

Fig. 1. Partial map of the evaluation environment.

VI. EXPERIMENTS AND RESULTS

We evaluated the planner integration using a version of the search and rescue scenario described in Section II, where we use colored boxes to represent humans, with green boxes representing wounded humans and boxes of other colors representing non-wounded humans.[3] The task takes place in a long hallway with several doors on either side, some of

[3]Note that while it is an important (open) problem to detect humans and determine whether they are injured, this problem is not relevant to our efforts here to verify the functionality of the proposed algorithms for detecting opportunities.

which are open (see Figure 1). In this setup, there are only two rooms containing colored boxes ("humans"), only one of which is green (representing a "wounded human"). There is one *hard task goal*: to arrive at the end of the hallway within the allotted time. The net utility of achieving this goal is 1000 units, while failing to achieve the goal in time is considered catastrophic. In addition, there is *one soft task goal*: to search rooms and report the presence of green boxes ("wounded humans"). This goal has a net utility of 500 units, but its completion is optional, so the task is not considered a failure if the soft goal is not achieved.

For all evaluation runs, the robot begins nearly halfway down the hallway, just over 15 meters away from the location it is supposed to reach – the end of the hallway – to meet its hard goal. The robot has initially no information about the environment other than that of the hallway which it needs to traverse, hence it does not know of the presence of any of doorways/rooms along the way. Rather, that information is presented to the planner as it is acquired perceptually along the way. The integrated system was tested with various values for the time limit on the hard goal, ranging from 30 seconds to 160 seconds. The results are summarized in Table I. Traversing the hallway takes approximately 50 seconds, so the planner does not even produce a plan when the time limit is set to 30 seconds, given that it knows about the length of the hallway ahead of time. Entering, scanning, and exiting a room takes approximately 35 seconds, so even with a time limit of 60 seconds, the planner does not replan to search any of the rooms encountered on the way to the hard goal, as it can estimate the action costs for these operations. It is only when the time limit of the hard goal is increased to 90 seconds that there is sufficient time to search the first room, after which the robot reports the presence of the green box. Increasing the time limit further allows the planner to produce plans to investigate additional rooms encountered on the way; in none of the remaining cases, however, is there a "wounded person", i.e,, a green box, to report (even though there is another room that contains a blue box representing a healthy human). Notice that the architecture could perform the intended actions (i.e., either entering an room and reporting any findings, or skipping the room) when the goal deadlines are dynamically adjusted during plan execution (by virtue of triggering replanning and allowing the planner to initiate perceptual monitoring via the "attend" primitive).

In sum, the evaluation confirms that the integrated architecture is able to tackle challenges 4 through 6 mentioned in the Introduction to its advantage. Specifically, by being able to detect opportunities using active *knowledge acquisition* (of perception obtained via scheduling monitoring), *real-time constraints* (to estimate the duration and completion times of action sequences), and *dynamic changes to goal structures* (based on updates to the problem representation), the robot is able to meet (optional) soft goals and thus significantly improve overall task performance (which would otherwise not be possible).

VII. RELATED WORK

While many hybrid control architecture have been proposed in the course of the last two decades (e.g., *AuRA* [14], *Atlantis* [15], the *SFX* [16], *3T* [17], *Saphira* [18], *Task Control Architecture* [19], and others), their design has remained a challenge when fast-working, highly responsive reactive layers are necessary (e.g., [20], [21]), largely because the two layers are so different in terms of their operational principles. Reactive layers, for example, typically keep state to a minimum, if state is kept at all [22]. Rather, sensory information is continuously processed and actions are immediately generated in response to sensory input. In contrast, higher architectural layers store information and use it to perform complex processing. Moreover, reactive layers often use continuous sensor-motor mappings, whereas action representations used by deliberative layers are typically discrete. In short, the challenge is get two layers that operate on different time scales, typically use different representations, perform operations of different complexity within different time spans, etc. to work together. The integrated architecture proposed in this paper demonstrates how higher-level planners can interface with lower-level action execution and monitoring components in systematic ways, through standard communication channels like notification of planner through the action execution component as well as real-time monitoring of plan execution through the *monitor*.

Most of the work on real-time, continuous planning depends on the existence of a known, direct path to a hard, defined goal. For instance, the work by Ruml, Do and Fromherz [23] defines an online planning problem and algorithm that depends upon up-front knowledge of the problem topology. Additionally, a good deal of work has been done with NASA domains and problems, including that by Chien et al. [24] and Pell et al. [25], but these also require a topology for goal achievement.

There has additionally been work on directly handling *goal arrival*. In particular, methods have been developed for planning with online goal arrival by Benton, et al. [26] and Sapena and Onaindía [27]. Hubbe et al. [28] deals with the same types of problems when goal arrival distribution is known.

Perhaps most closely related to this work is that of Etzioni et al. [12], who developed a method of defining *local closed world* assertions, essentially allowing a planner to determine what it knows and does not know (thereby allowing it to seek out this knowledge). Their work, however, assumes all goals are hard and must be achieved.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we presented a novel integrated robotic planning architecture for opportunistic planning and plan execution in the light of limited prior knowledge about the planning domain, non-monotonic changes to the planner's knowledge through dynamic knowledge acquisition and changes of goals and goal utilities during plan execution. The architecture contains mechanisms for generating counterfactual "what-if" scenarios using background knowledge about

the planning domain to detect to possible action sequences that can lead to the goal opportunities. Based on the outcome of these reasoning processes, it can systematically schedule perception actions as part of a plan that can be carried out by the robot in parallel to plan actions in order to be able to detect goal opportunities. Whenever an opportunity is detected, goal structures and deadlines can be re-assessed dynamically to be able to exploit the detected opportunities if they can lead to higher net payoffs. The new integrated architecture was then implemented and tested on a robot to demonstrate its functionality and utility in dynamically changing real-world domains.

For future work, we are investigating methods of reasoning about the cost of failure. The planner should be able to reason about the possibility that a sensing action will be unsuccessful at enabling a path to the goal (i.e., the sensing action completes with no object found), allowing it to determine whether the cost of failure outweighs the potential benefits of success. Additionally, background knowledge is currently limited to specifying information about single sensed objects. Expanding this representation to allow for more objects would provide a better way of chaining counterfactuals together. For example, predicates that involve multiple sensed objects could be specified such as "there exists a door connected to a room" where both the door and room need to be sensed for (the current system can only chain such counterfactual scenarios together in a very limited fashion). We are also planning to extend the planner's pre- and postconditions to accommodate temporal specifications in a temporal logic, which will allow the robot to accept natural language instructions by humans that can be automatically converted into formal goal specifications and passed on to the planner for consideration [29]. Finally, we will also conduct more extensive and systematic evaluation experiments both in simulation and on robots to verify that the proposed architecture will scale up.

IX. ACKNOWLEDGMENTS

This work was in part funded by ONR MURI grant #N00014-07-1-1049. Thanks also to W. Cushing and Minh B. Do.

REFERENCES

[1] L. Kaelbling, M. Littman, and A. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[2] A. Cimatti, M. Roveri, and P. Traverso, "Strong planning in non-deterministic domains via model checking," in *Proceedings of the 4th International Conference on Artificial Intelligence Planning System (AIPS98)*, 1998, pp. 36–43.

[3] S. Kambhampati, "Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain theories," *Proceedings of AAAI07*, 2007.

[4] C. Boutilier, T. Dean, and S. Hanks, "Decision-Theoretic Planning: Structural Assumptions and Computational Leverage," *Journal of Artificial Intelligence Research*, vol. 11, p. 194, 1999.

[5] M. Do and S. Kambhampati, "Partial Satisfaction (Over-Subscription) Planning as Heuristic Search," *Proceedings of KBCS-04*, 2004.

[6] W. Cushing, J. Benton, and S. Kambhampati, "Replanning as a deliberative re-selection of objectives," Arizona State University, Tech. Rep., 2008.

[7] M. Scheutz, P. Schermerhorn, J. Kramer, and D. Anderson, "First steps toward natural human-like HRI," *Autonomous Robots*, vol. 22, no. 4, pp. 411–423, May 2007.

[8] J. Kramer and M. Scheutz, "Robotic development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, no. 2, pp. 101–132, 2007.

[9] M. Scheutz, "ADE - steps towards a distributed development and runtime environment for complex robotic agent architectures," *Applied Artificial Intelligence*, vol. 20, no. 4-5, pp. 275–304, 2006.

[10] J. Benton, M. Do, and S. Kambhampati, "Anytime heuristic search for partial satisfaction planning," *Artificial Intelligence Journal*, 2009.

[11] D. Smith and D. Weld, "Temporal planning with mutual exclusion reasoning," in *International Joint Conference on Artificial Intelligence*, vol. 16, 1999, pp. 326–337.

[12] O. Etzioni, K. Golden, and D. Weld, "Sound and efficient closed-world reasoning for planning," *Artificial Intelligence*, vol. 89, no. 1-2, pp. 113–148, 1997.

[13] T. Brick and M. Scheutz, "Incremental natural language processing for HRI," in *Proceedings of the Second ACM IEEE International Conference on Human-Robot Interaction*, Washington D.C., March 2007, pp. 263–270.

[14] R. C. Arkin and T. R. Balch, "AuRA: principles and practice in review," *JETAI*, vol. 9, no. 2-3, pp. 175–189, 1997. [Online]. Available: citeseer.nj.nec.com/arkin97aura.html

[15] E. Gat, "Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots," *SIGART Bulletin 2*, pp. 70–74, 1991.

[16] R. Murphy and R. Arkin, "SFX: An architecture for action-oriented sensor fusion," in *Proceedings of IROS 1992*, 1992, pp. 1079–1086.

[17] R. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack, "Experiences with an architecture for intelligent, reactive agents," *JETAI*, vol. 9, no. 2/3, pp. 237–256, Apr 1997.

[18] K. Konolige, K. L. Myers, E. H. Ruspini, and A. Saffiotti, "The Saphira architecture: A design for autonomy," *Journal of experimental & theoretical artificial intelligence: JETAI*, vol. 9, no. 1, pp. 215–235, 1997. [Online]. Available: citeseer.nj.nec.com/konolige97saphira.html

[19] R. Simmons, R. Goodwin, K. Haigh, S. Koenig, and J. Sullivan, "A layered architecture for office delivery robots," in *First International Conference on Autonomous Agents*, February 1997, pp. 235 – 242.

[20] R. Jensen and M. Veloso, "Interleaving deliberative and reactive planning in dynamic multi-agent domains," in *Proceedings of the AAAI Fall Symposium on Integrated Planning for Autonomous Agent Architectures*. AAAI Press, oct 1998. [Online]. Available: http://citeseer.nj.nec.com/jensen98interleaving.html

[21] P. Maes, "Situated agents can have goals," in *Designing Autonomous Agents*, P. Maes, Ed. MIT Press, 1990, pp. 49–70.

[22] E. Gat, "On three layer architectures," in *Artificial Intelligence and Mobile Robots*, D. Kortenkamp, R. P. Bonnasso, and R. Murphey, Eds. AAAI Press, 1998.

[23] W. Ruml, M. Do, and M. Fromherz, "On-line planning and scheduling for high-speed manufacturing," in *International Conference on Automated Planning and Scheduling*, 2005.

[24] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, "Using iterative repair to improve responsiveness of planning and scheduling," in *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 2000, pp. 300–307.

[25] B. Pell, E. Gat, R. Keesing, N. Muscettola, and B. Smith, "Robust periodic planning and execution for autonomous spacecraft," in *International Joint Conference on Artificial Intelligence*, vol. 15, 1997, pp. 1234–1239.

[26] J. Benton, M. Do, and W. Ruml, "A simple testbed for on-line planning," in *Proceedings of the ICAPS Workshop on Moving Planning and Scheduling Systems into the Real World*, 2007.

[27] O. Sapena and E. Onaindıa, "An architecture to integrate planning and execution in dynamic environments," *Proceedings of PLANSIG*, vol. 3, pp. 184–193, 2003.

[28] A. Hubbe, W. Ruml, S. Yoon, J. Benton, and M. Do, "On-line Anticipatory Planning," *ICAPS Workshop on a Reality Check for Planning and Scheduling Under Uncertainty*, 2008.

[29] J. Dzifcak, M. Scheutz, C. Baral, and P. Schermerhorn, "What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution," in *Proceedings of the 2009 International Conference on Robotics and Automation*, Kobe, Japan, forthcoming.