A Planning Model for Agents in Dynamic and Uncertain Real-Time Environments*

Alexander Nareyek
GMD — FIRST
German National Research Center for Information Technology
Research Institute for Computer Architecture and Software Technology
Rudower Chaussee 5
D - 12489 Berlin, Germany
alex@first.gmd.de

Abstract

The management of an agent in a dynamic and uncertain real-time environment is fairly intractable with traditional planning methods. Such methods cannot be computed efficiently, are not capable of replanning, and mostly lack explicit temporal formalisms. The planning model proposed in this paper overcomes these weaknesses by applying iterative repair methods, constraint-programming techniques, and an explicit timeline approach.

For the management of incomplete knowledge, a single-plan approach is proposed in which only one plan possibility is considered. The application of UNKNOWN values, KNOWN values, NOT values, and continuous domain fluents enables to cope with incomplete knowledge.

Introduction

The work originated as part of the EXCALIBUR project. The goal of this project is to develop a generic architecture for autonomously operating agents, like computer-guided characters/mobiles/items, within a complex computer-game environment. These agents have to be able to find the right actions to pursue their given goals, and adapt their behavior to new environments or opponents.

A crucial aspect of an agent is the way its behavior is determined. If there shall be no restriction on reactive actions with predetermined behavior patterns, an underlying planning system is needed. A lot of research has been done on planning, and a wide range of planning systems have been developed, e.g. STRIPS (Fikes & Nilsson 1971), UCPOP (Penberthy & Weld 1992), and PRODIGY (Veloso et al. 1995).

To meet the hard real-time constraints of the computer-game environment (most of the CPU power is used for the game engine and graphics), an interleaving of planning and execution is essential. An agent in sudden danger calls for fast reactions, whereas a longer reasoning time may be taken to establish more elaborate plans. In addition, the highly dynamic environment makes continuous replanning indispensable. Furthermore, agents' interactions require the ability to model temporally complex relations, and the agent's restricted insight requires a way of representing incomplete knowledge.

*This work is supported by the German Research Foundation (DFG), Conitec Datensysteme GmbH, Cross Platform Research Germany (CPR).

Iterative Plan Repair

The search for a plan is done by an iterative plan repair approach. Iterative repair approaches perform a search by iteratively changing an initial state. They provide a solution at anyone time, the quality of the solution being subject to constant improvement. Most of these iterative methods are incomplete, and they may get trapped in local optima or on plateaus. Many methods apply additional techniques to escape from these local minima and plateaus, e.g. restarts, random walks, or tabu lists. If the agent's environment is highly dynamic, the importance of these features declines, the search space changing quickly and less in-depth improvement being possible.

The iterative plan-repair computation is shown in figure 1.

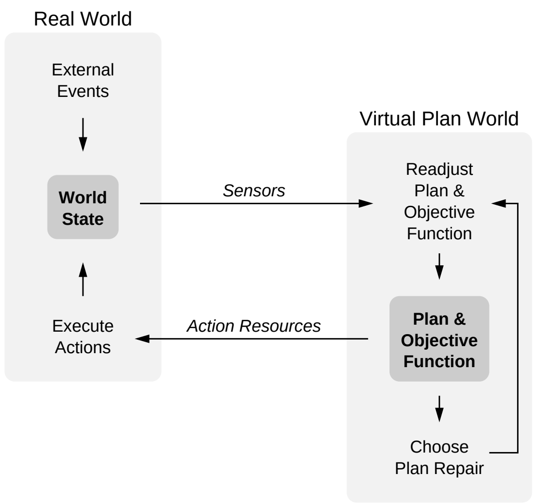

Figure 1: World Dynamics and Plan Improvement

Modeling with Constraints

Constraint-programming techniques focus on the solution of combinatorial search problems, and are thus well suited for providing the framework for the plan search. The basic constraint satisfaction problem is formulated as a set of variables $X = \{X_1, \ldots, X_n\}$, where each variable is associated with a domain $D_1, ..., D_n$, and a set of constraints $C = \{C_1, ..., C_m\}$ over these variables. Constraints are relations between variables, that restrict the possible value assignments.

The EXCALIBUR model applies so-called *global constraints* (Puget & Leconte 1995), instead of using a standard problem encoding with simple primitives such as linear inequalities and the like. A global constraint is a replacement for a set of lower-level constraints, where additional domain knowledge allows the application of specialized data representations and algorithms to guide and accelerate the search.

The global constraints are used to evaluate the current plan. Each constraint calculates a local satisfaction value. These satisfaction values are combined within an objective function, which serves as a measure of the plan's quality. The local satisfaction values not only have to express constraint violations, but also the difference to preferred solutions (goals).

The second task of the constraints is to provide suggestions for improvement. Repair methods may consist of changing control variables, like the begin and end time points of tasks, references, etc. But whole actions can also be created, deleted, and objects can be split into several objects, etc. In the current implementation, the repair suggestion of the constraint with the highest inconsistency is chosen.

Plan Structures

The model is inspired by the job-shop-scheduling scenario, which has proved to be a first-class application domain for constraint programming. Consequently the model's components are described in terms of resources, tasks, and constraints.

Actions, Action Tasks, and Action Resources

The execution of an **action** (like EAT PEANUT) can be divided into **action task** subcomponents. Each of these action tasks utilizes an **action resource** for its execution. For instance, the action EAT PEANUT requires action tasks on a MOUTH and a LEFT HAND or RIGHT HAND action resource. Figure 2 visualizes the assignment of action tasks to action resources.

An action task's structure contains a **description**, which specifies execution information for the action resources. Two further entries determine the task's **begin** and **end**. Begin and end are decision variables, but as iterative repair is to be applied, there are always *fixed* values assigned to them.

Each action has a **task constraint** to set the begin and end of the action's tasks in specific relations. For example, the action tasks of the action EAT PEANUT must begin and end at the same time, and the begin and end values must be four seconds apart. The role of an action's **precondition constraint** is explained in the next section.

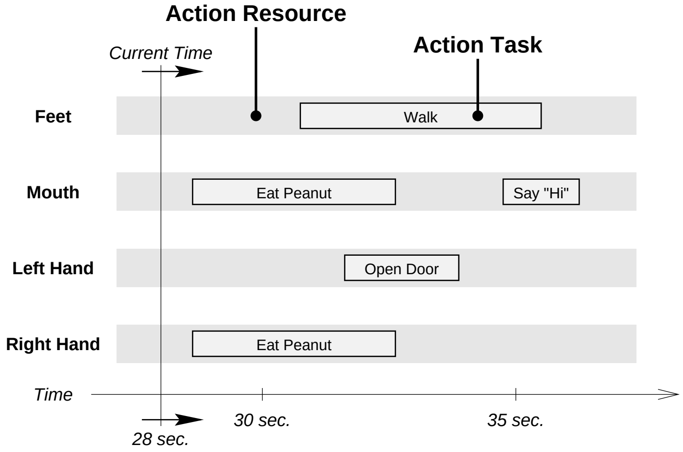

Figure 2: The Assignment of Action Tasks to Action Resources

Within an action resource, an **action resource constraint** must ensure that the resource's action tasks do not overlap, as simultaneous executions of tasks would interfere with each other. For example the agent is not allowed to talk and to eat at the same time.

To clarify the role of the constraints, the current implementation of the task constraint is detailed in the following:

- The task constraint manages a set of task relations. Each task relation is represented by a linear inequality $X_1 \otimes X_2 + c$, X_1 and X_2 being the start or end time points of tasks, $\otimes \in \{<, \leq, =, \geq, >\}$, and a shift constant c.
- For each linear inequality that is unsatisfied, the minimal shift distance required for one of the inequality variables to satisfy the inequality is added to the task constraint's inconsistency.
- If the task constraint is chosen to improve the current state, an inconsistent inequality is selected, and a minimal shift of one of the involved tasks is performed in such a way, that the inequality is fulfilled.

State Tasks and State Resources

Pre- and postconditions of the actions are maintained by constraints between the action tasks and the state resources and state tasks.

A **state resource** is similar to an action resource. It does not manage actively planned actions, but rather the development of a specific property of the environment or the agent itself[1]. For example, an OWN PEANUT state resource with a Boolean assignment for

[1]State resources can be interpreted as *fluents*.

anyone time can provide information about the possession of a peanut (see figure 3).

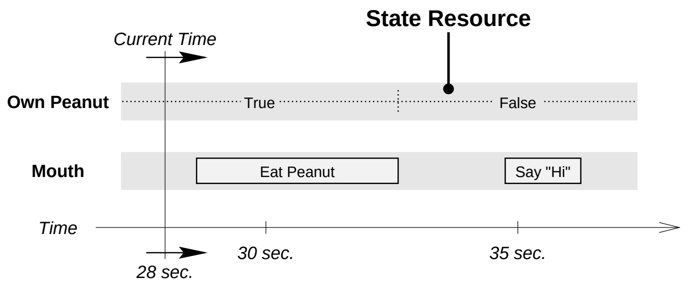

Figure 3: A State Resource

The status of the state resources can restrict the application of actions. To execute the action task EAT PEANUT, it is first necessary to have a peanut. These relations are preserved by the precondition constraints of actions, which access state resources' values.

The effects of actions are more complicated to realize, as multiple actions and events may have synergetic effects. For example, a state resource HUNGER with assignments of natural numbers can be influenced by an improving action EAT PEANUT and a worsening WALK at the same time.

It is the role of **state tasks** to describe an action's effects. For instance, a state task of the action EAT PEANUT is responsible for a decreasing **contribution** of -3 to the state resource HUNGER during the action's execution (see figure 4). Each state resource has a specific **state mapping**, that maps the contributions of the state tasks to values of the state resource's domain. In the case of the HUNGER resource, the synergetic effect is a simple addition of the single gradients. An additional **state resource constraint** maintains internal resource consistencies, e.g. domain restrictions, etc.

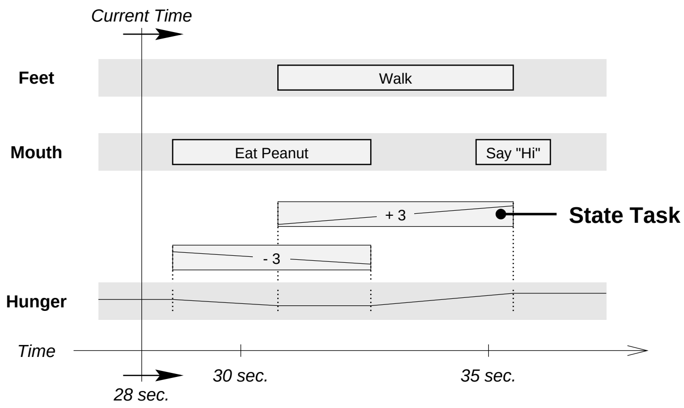

Figure 4: The Mapping Mechanism of State Resources

There can be further effects, which may be caused by synergetic effects within a state resource. Adding water to a bathtub may result in its overflowing and wetting the bathroom. The actions cannot provide state tasks to realize these further effects because an action has only the limited view of its state task contributions. Thus, **dependency** effects of specific state resource states must be expressed in addition. The dependencies are special actions that are beyond the agent's control. Expected external events can also be integrated by these dependencies.

Objects, References, and Sensors

There may be more things to eat in the world than a peanut. Of course it is possible to define an action for every possible object. A neater and more flexible way is to define a general action EAT, whose tasks contain **references** to the affected state resources.

Consider a case where that we have two peanuts, a big and a small one. As long as the world's objects are not all unique, their state resources are indistinguishable. If the EAT action were applied, the big peanut might vanish, whilst the small one would be used to lessen the hunger.

To avoid such equivocations, a group of state resources is combined to form an **object**. The object concept also provides other nice features like inheritance, etc. Figure 5 illustrates the application of the EAT action to a PEANUT.

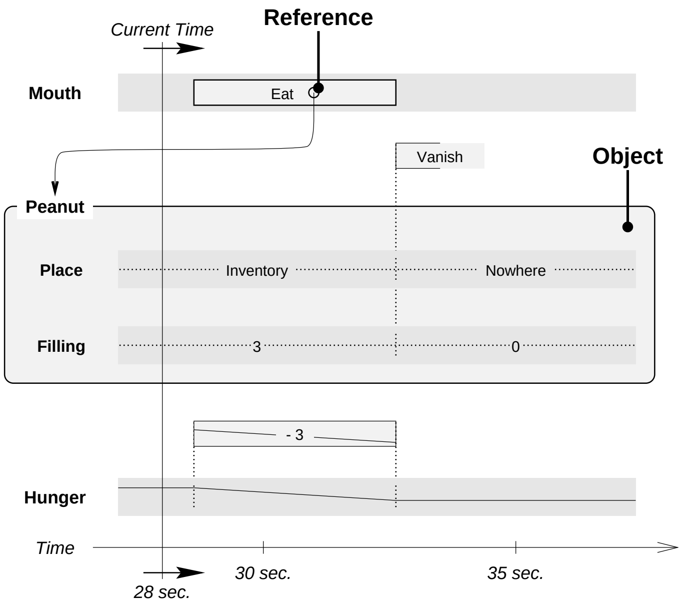

Figure 5: References and Objects

Of course, not only action tasks can use references. State tasks and state resources may also use references to express things like BLOCK A IS ON BLOCK B (see figure 6).

Objects may also have a related **sensor**. The sensors report on actual data, like the current perception of hunger. We assume high-level sensoring that provides ready-structured objects. Sensed objects are related to the virtual objects of the plan.

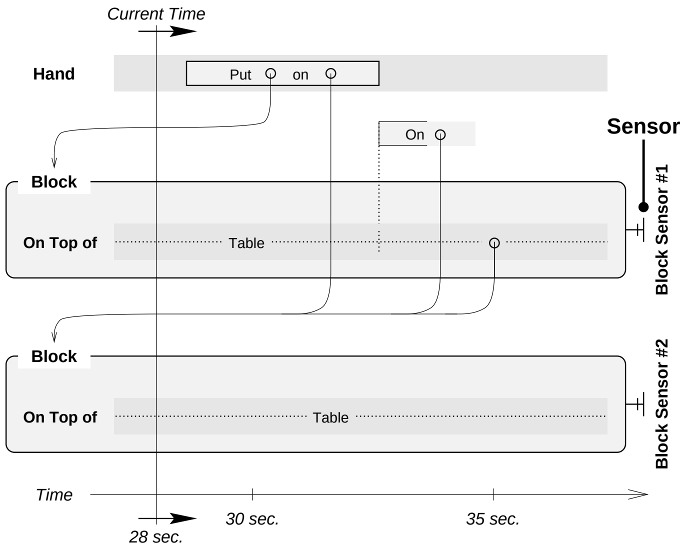

Figure 6: A Blocks World Example

Since the agent has a limited view of the dynamic environment, it cannot fully rely on its knowledge of objects. Hence, each object has a **confidence** value which expresses the amount of trust that the virtual plan object has a real counterpart. The assignment of a sensor to an object usually results in a very high confidence value. A sensor must always be assigned to an object.

Incomplete Knowledge

An agent's incomplete knowledge of his environment complicates the planning process enormously because multiple possibilities have to be considered. This incompleteness of knowledge relates to the environment's state as well as to the effects of actions. This is much too expensive for real-time planning systems, and even worse for temporal planning, where a great deal more information has to be handled. A dynamic environment like a multi-agent system increases the complexity even more.

The following subsections give an overview of different approaches for coping with the plan possibilities, discuss how to model missing information, how to eliminate uncertainty by information gathering mechanisms, and how to extend the expressiveness to model even situations, where the transition from lack of knowledge to knowledge is fuzzy.

Considering Only One Plan Possibility

If a decision relies on an unknown property, every possible property value may yield another plan. If the property value becomes available somewhere within the plan, the plan has to branch for each important refinement. Planners that construct such branching plans are called *contingency planners*. Examples are WARPLAN-C (Warren 1976), CNLP (Peot & Smith 1992), Plinth (Goldman & Boddy 1994), and Cassandra (Pryor & Collins 1996).

An extension of this approach is *probabilistic planning*, where special probability distributions are considered as well. For example, it is likely that a vending machine neither has the requested product nor wants to return the money. Work in this area includes synthetic projection (Drummond & Bresina 1990), Markov decision process approaches (Dean et al. 1995), and the BURIDAN probabilistic planning (Kushmerick, Hanks & Weld 1995) with its contingent extension (Draper, Hanks & Weld 1994).

The consideration of multiple possibilities can be useful in terms of reliability. Although the planners (especially the probabilistic ones) do not always search the whole search space, application to more complex problem domains, temporal planning, and dynamic environments would largely overtax storage and computation power. Strong real-time requirements such as those for EXCALIBUR's agents are out of the question.

A solution is to consider only *one possible plan*. The choice of a plan alternative does not have to be statically fixed. The decision can be based on pessimism or optimism, as well as on estimated probabilities. The preference can be dependent on the specific situation, and can be influenced by learning mechanisms, too. In the case of a failed prediction, the plan has to be changed. As long as no critical errors have been made, the iterative plan repair can automatically adapt the plan. This consideration of a single plan possibility is close to the operator-parameterization approach of the Cypress system (Wilkins et al. 1995).

A fixed plan still allows incomplete knowledge regarding the states. Only the *actions are fixed*, while some *states can have a variety of possible assignments*. For example, climbing onto a telephone book is independent of the language in which it is written. Opting for a special language would merely mislead further action decisions, as no distinction between assumptions and knowledge can be made.

Lack of Information

The state resources can represent the lack of information by the addition of an UNKNOWN value to their domain (which gets the default state). For example, in the agent's absence, other agents might open or close a door without the agent's noticing. Whether the door is closed or not can only be known if the door is within the agent's field of vision. Thus, the state resource DOOR with a domain of OPEN and CLOSED gets an additional UNKNOWN value, which is triggered by actions, that cause the agent's absence.

The action of passing the door requires that door is open. In a pessimistic approach, the precondition constraint of the passing action has a bad satisfaction (inconsistency) if the door is in an UNKNOWN state because failure could endanger later commitments. The inclusion of a prophylactic OPEN DOOR action averts this threat (see figure 7).

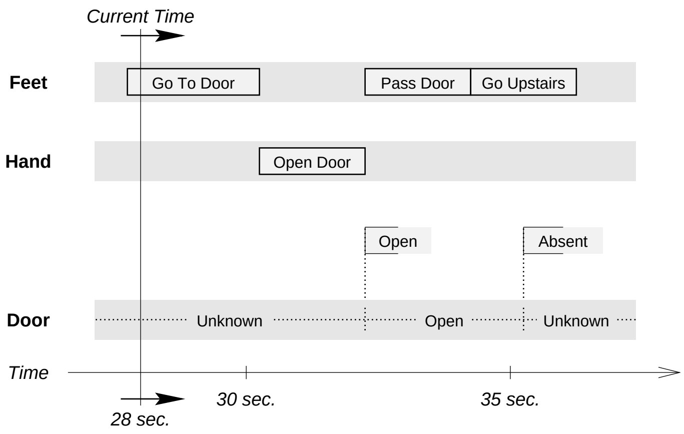

Figure 7: The Use of UNKNOWN Values

On the other hand, the satisfaction could be driven by experience. If the agent has learned that the door is usually open, the precondition constraint of the passing action might settle for the UNKNOWN state of the door.

Information Gathering

Classical planners normally try to find a plan to satisfy goal states if they are not already satisfied. But in an incomplete environment they cannot decide whether an unknown state is already satisfied or not. An unknown state like COLOR(DOOR, BLUE) could only be satisfied by painting the door blue. If the door is already blue, this action would be unnecessary, and a lack of blue paint would even entail an inconsistency. The ability to plan sensory actions too was realized in various STRIPS-based approaches, such as IPEM (Ambros-Ingerson & Steel 1988), UWL (Etzioni et al. 1992) and XII (Golden, Etzioni & Weld 1994), Sage (Knoblock 1995), and Occam (Kwok & Weld 1996).

Run-Time Variables The concepts are mostly based on run-time variables, which are initialized by sensing actions. In the EXCALIBUR model, the fact of knowing a state can easily be expressed by the inclusion of KNOWN values for state resources. These values are triggered by actions, which include the sensing of the state resource's property. The state resources' mapping mechanisms must protect already specified states from the more general KNOWN reassignment, and allow a switch to the KNOWN state only from UNKNOWN states. This process does not differ from the normal state processing and does not require a special sensory treatment. In the example in figure 8, the crossing of a bridge in an unknown state is considered to be unsatisfiable enough to include an additional TEST BRIDGE action.

The problem of *redundant sensing*, which is addressed in (Golden, Etzioni & Weld 1994) is not present here, as there is only a single plan and the state resources' information is accessible over the whole time period.

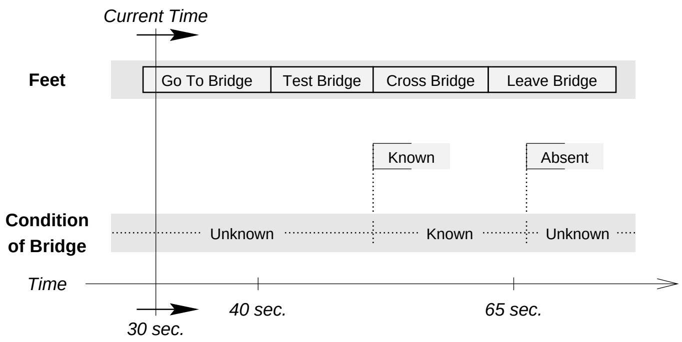

Figure 8: The Use of KNOWN Values

Hands-Off Goals *Hands-off goals*, as in (Etzioni et al. 1992), that forbid the change of states are not necessary either. The formulation of goals like passing the blue door are a problem only for planners, who cannot express that the door has to be blue at the moment of the goal formulation. These planners have to introduce such hands-off goals to prevent the planner from passing another door after painting it. Temporal planners can express these goals much more adequately because they can quantify the preconditions temporally (see figure 9). Moreover, in dynamic multi-agent domains, such hands-off goals do not help, as external actions might change the states.

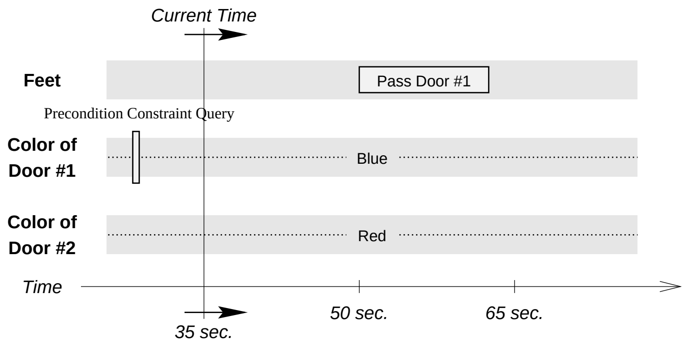

Figure 9: Temporal Quantification

Partial Knowledge

So far, only state resources with two-valued domains and clear transitions between knowledge and no knowledge have been considered. The following sections refine this approach.

Unordered Domains Action-resource domains in the EXCALIBUR agent model may not only be two-valued, like the DOOR with OPEN and CLOSED values. For example, there may be an additional LOCKED value. In a situation of incomplete knowledge, each domain value must have a knowledge level of

- UNKNOWN: The information as to whether the value corresponds to the state is not available;

- KNOWN: The information as to whether the value corresponds to the state will be available in the future; or
- NOT: The domain value definitely does not correspond to the state.

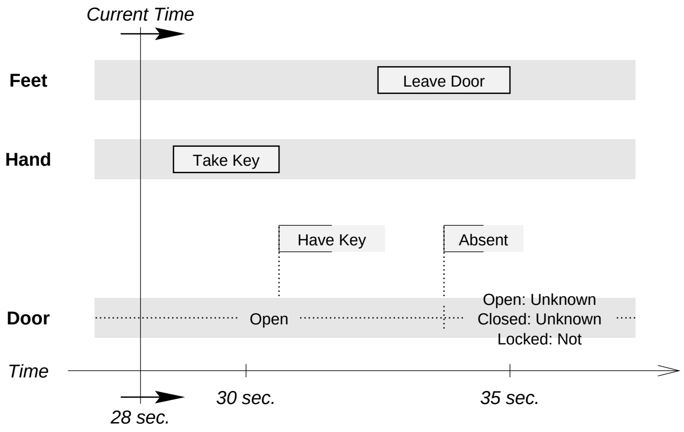

Figure 10: Incomplete Knowledge of State Values

Because of the XOR-relation of the domain values, some propagations can be made within a state resource. If only one domain value has a knowledge level of UNKNOWN, then this value gets the knowledge level KNOWN. If only one domain value is not NOT, then this value must be the state resource's state.

Ordered Domains In contrast to the value sets of the previous section, the elements of a state resource's domain can also be in a specific ordering relation. For example, the agent wants to fill a bucket with water, but he does not know how much water is in the bucket to start with. The amount of water in the bucket can be modeled by an integer state resource.

Of course, it is possible to apply the UNKNOWN-KNOWN-NOT representation from the previous section to each domain value. But this would be a rather costly approach. It is more efficient to subsume consecutive values of the same knowledge level by intervals[2]. Figure 11 shows an example.

It is even more efficient to consider only the convex (vertical) hull of intervals (trapezoids) of the same knowledge level. Actually, this method is not precise, because already excluded intermediate values may not be accessible any more. Consequently, the intervals (trapezoids) of different knowledge levels might overlap. For example, in the 65th second of figure 11, the convex hull of the NOT knowledge level includes the KNOWN values.

Probabilities Probabilities for prospective states of the state resources are not a basic part of the model,

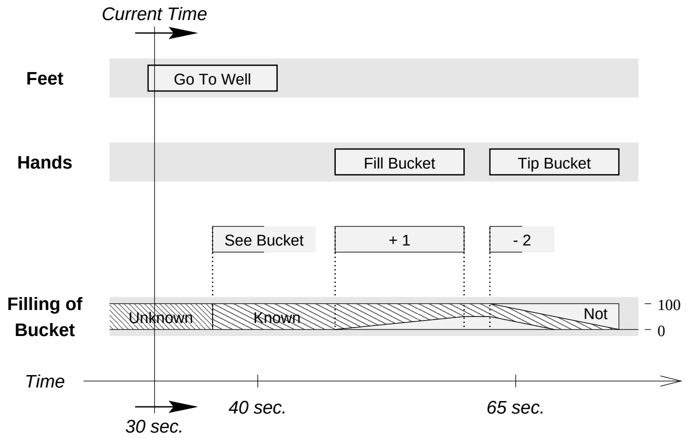

Figure 11: Value Subsumption

because this is not generally needed and would waste a lot of system resources. In some cases, however, the use of probabilities can dramatically affect the plan result, especially if a state probability can be changed by special actions. For example, if the agent is searching for a key, it is not easy to express progress within the search process without probabilities. Whether the agent searches one drawer or two drawers must make a difference.

The probabilities can be realized by continuous domain state resources. Figure 12 shows a possible modeling of the key search. Further actions' precondition constraints (like that of TAKE KEY) can use the probability value to compute their satisfaction. The probability value can also be adopted to influence further dependencies.

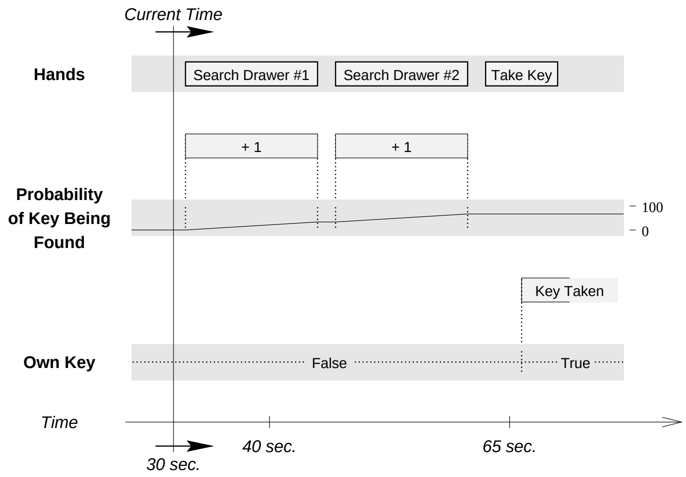

Figure 12: State Probabilities

Related Work

Most existing planning systems use a STRIPS-like representation, which is based on the Situation Calculus

[2]An appropriate representation for taking into account the dimension of time are trapezoids (provided that changes are only linear).

(McCarthy & Hayes 1969). The Situation Calculus works on a branching time-point structure, where complete world descriptions (sets of states) are linked by actions. In contrast to this, the EXCALIBUR model belongs to the group of explicit timeline approaches. In there, actions and states are related to a timeline. Their most prominent representative is Allen's interval temporal logic (Allen & Ferguson 1994). The Situation Calculus' branching structure allows to reason about multiple possible futures in a very direct way, whereas the focus of explicit timeline approaches is on handling complex temporal relations.

There are only a few planners that use an explicit notion of time, e.g. *parc*PLAN (Lever & Richards 1994), ZENO (Penberthy & Weld 1994), and *Descartes* (Joslin 1996). All these systems have problems in expressing state transitions (internal fluent constraints). In most cases, only temporal overlaps of contradictory propositions are forbidden. *parc*PLAN allows the exclusion of simple property constellations as well.

The representation problem is even bigger for standard constraint-solving tools for planning and scheduling, such as OZONE (Smith, Lassila & Becker 1996) or CHIP (Simonis 1995). They do not feature general state concepts and focus on handling capacity resources. The HSTS representation (Muscettola 1994) addresses states and state transitions much better, but the representation is on a very low level. EXCALIBUR's specialized high-level constraints make it possible to include constraint-specific search control and representation knowledge.

Nearly all planning systems search for a plan by refinement. Refinement is a stepwise narrowing process. In each step, a subset of states (or plans) is chosen for further investigation until a solution is found. In the case of an inconsistency, backtracking is performed over the refinement decisions.

The use of iterative repair search is very rare. Besides EXCALIBUR, there are only a few iterative repair approaches, e.g. Satplan (Kautz & Selman 1996) and ASP (Bonet, Loerincs & Geffner 1997).

While long-term refinement search produces better results than iterative repair strategies, the quality improvement gradient of iterative repair is much better for a short reasoning time. Larger problems usually result in the significant superiority of iterative repair as well (Wallace & Freuder 1996; Gent et al. 1997). Furthermore, a dynamic environment promotes iterative repair search, because the iterative search is little affected by modifications of the search space. This is not true of refinement methods, which have to update their memory of the already explored search space and may have to redo the whole search.

The iterative repair approach inherently supports a partial constraint satisfaction and anytime availability as well. This makes fast reactions possible, which has long been the domain of popular Alife agent systems like subsumption architectures (Brooks 1986). Such systems apply a reactive approach with predetermined behavior patterns, and are not capable of longer-term reasoning. Many hybrid agent systems like the 3T robot architecture (Bonasso et al. 1997) or the New Millenium Remote Agent (Pell et al. 1996) perform reactions only in a very limited sense, applying a traditional offline deliberative planner for higher-level planning.

Conclusion

The EXCALIBUR agent model satisfies the requirements for a dynamic real-time multi-agent system. The explicit temporal approach makes it possible to handle temporally complex relations, and the representation of UNKNOWN, KNOWN, and NOT knowledge levels enables to deal with incomplete knowledge. Even a limited use of probability computation is possible. High-level global constraints allow a declarative problem specification and the application of specialized satisfaction algorithms and representations. A constant improvement/replanning by iterative repair optimization allows efficient real-time creation and maintenance of the agent's plan.

Future work includes the refinement of specialized satisfaction mechanisms, the elaboration of interagent coordination, and the incorporation of learning methods. Further and more detailed information about the EXCALIBUR project is available at:

`http://www.first.gmd.de/concorde/EXCALIBURhome.html`

References

Allen, J. F., and Ferguson, G. 1994. Actions and Events in Interval Temporal Logic. *Journal of Logic and Computation* 4(5): 531–579.

Ambros-Ingerson, J. A., and Steel, S. 1988. Integrating Planning, Execution and Monitoring. In Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88), 83–88.

Bonasso, R. P.; Firby, R. J.; Gat, E.; Kortenkamp, D.; Miller, D. P.; and Slack, M. G. 1997. Experiences with an Architecture for Intelligent, Reactive Agents. *Journal of Experimental and Theoretical Artificial Intelligence* 9(1).

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A Robust and Fast Action Selection Mechanism for Planning. In Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97), 714–719.

Brooks, R. A. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation* RA-2 (1): 14–23.

Dean, Th.; Kaelbling, L. P.; Kirman, J.; and Nicholson, A. 1995. Plannung under Time Constraints in Stochastic Domains. *Artificial Intelligence* 76: 35–74.

Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic Planning with Information Gathering and Contingent Execution. In Proceedings of the Second International Conference on AI Planning Systems (AIPS-94), 31–36.

Drummond, M. E., and Bresina, J. L. 1990. Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction. In Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90), 138–144.

Etzioni, O.; Hanks, S.; Weld, D.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An Approach to Planning with Incomplete Information. In Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92).

Fikes, R. E., and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 5(2): 189–208.

Gent, I. P.; MacIntyre, E.; Prosser, P.; and Walsh, T. 1997. The Scaling of Search Cost. In Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97), 315–320.

Golden, K., Etzioni, O., and Weld, D. 1994. Omnipotence Without Omniscience: Efficient Sensor Management for Planning. In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), 1048–1054.

Goldman, R. P., and Boddy, M. S. 1994. Conditional Linear Planning. In Proceedings of the Second International Conference on AI Planning Systems (AIPS-94), 80–85.

Kautz, H., and Selman, B. 1996. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96), 1194–1201.

Knoblock, C. A. 1995. Planning, Executing, Sensing, and Replanning for Information Gathering. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95), 1686–1693.

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An Algorithm for Probabilistic Planning. *Artificial Intelligence* 76: 239–286.

Kwok, Ch. T., and Weld, D. S. 1996. Planning to Gather Information. In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96), 32–39.

Lever, J., and Richards, B. 1994. *parc*Plan: a Planning Architecture with Parallel Actions, Resources and Constraints. In Proceedings of the Ninth International Symposium on Methodologies for Intelligent Systems (ISMIS-94), 213–223.

McCarthy, J., and Hayes, P. J. 1969. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In Meltzer, B., and Mitchie, D. (eds.), *Machine Intelligence 4*, Edinburgh University Press.

Joslin, D. 1996. Passive and Active Decision Postponement in Plan Generation. PhD thesis, University of Pittsburgh, Pittsburgh, PA.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M., and Fox, M. S. (eds.), *Intelligent Scheduling*, Morgan Kaufmann, 169–212.

Penberthy, J. S., and Weld, D. S. 1992. UCPOP: A Sound, Complete, Partial Order Planner for ADL. In Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92), 102–114.

Penberthy, J. S., and Weld, D. S. 1994. Temporal Planning with Continuous Change. In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94), 1010–1015.

Pell, B.; Bernard, D. E.; Chien, S. A.; Gat, E.; Muscettola, N.; Nayak, P. P.; Wagner, M. D.; and Williams, B. C. 1996. A Remote Agent Prototype for Spacecraft Autonomy. In Proceedings of the SPIE Conference on Optical Science, Engineering, and Instrumentation.

Peot, M., and Smith, D. 1992. Conditional Nonlinear Planning. In Proceedings of the First International Conference on AI Planning Systems, 189–197.

Pryor, L., and Collins, G. 1996. Planning for Contingencies: A Decision-based Approach. *Journal of Artificial Intelligence Research* 4: 287–339.

Puget, J.-F., and Leconte, M. 1995. Beyond the Glass Box: Constraints as Objects. In Proceedings of the 1995 International Logic Programming Symposium (ILPS'95), 513–527.

Simonis, H. 1995. The CHIP System and Its Applications. In Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP95).

Smith, S. F.; Lassila, O.; and Becker, M. 1996. Configurable, Mixed-Initiative Systems for Planning and Scheduling. In Tate, A. (ed.), *Advanced Planning Technology: Technological Achievements of the ARPA/Rome Laboratory Planning Initiative*, AAAI Press, Menlo Park (CA).

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating Planning and Learning: The PRODIGY Architecture. *Journal of Experimental and Theoretical Artificial Intelligence* 7(1).

Wallace, R. J., and Freuder, E. C. 1996. Anytime Algorithms for Constraint Satisfaction and SAT problems. *SIGART Bulletin* 7(2).

Warren, D. H. D. 1976. Generating Conditional Plans and Programs. In Proceedings of the Summer Conference on Artificial Intelligence and Simulation on Behavior, 344–354.

Wilkins, D. E.; Myers, K. L.; Lowrance, J. D.; and Wesley, L. P. 1995. Planning and Reacting in Uncertain and Dynamic Environments. *Journal of Experimental and Theoretical AI* 7(1): 197–227.