Mixed-Initiative Planning for Space Exploration Missions

Tatiana Kichkaylo,[†] **Chris van Buskirk,**[‡] **Sameer Singh,**[‡]
Himanshu Neema,[‡] **Michael Orosz,**[†] **and Robert Neches**[†]

[†]USC Information Sciences Institute, Marina del Rey, CA 90292, USA
[‡]Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37203, USA
{*tatiana,mdorosz,rneches*}*@isi.edu,* {*Chris.vanBuskirk,Sameer.Singh,himanshu*}*@vanderbilt.edu*

Abstract

Modern planning and scheduling systems are capable of
dealing with the size and complexity of many real world
problems. However, mission critical planning is still
often done by humans. Even if only a couple of plans
are produced ("Master Plan" and "Plan B"), human
experts evaluate multiple alternatives, think of con-
tingencies, consider the likelihood of failure of various
steps, and account for schedule slack and plan flexibil-
ity. Computers can evaluate thousands of alternative
scenarios, but the solutions they ultimately produce are
often not convincing enough for expert decision makers
to trust human lives or mission critical operations to
computer decisions. Further, automated systems often
require significant changes in the way people operate,
which in high-stakes high-pressure environments leads
to rejection of the system by the users.

In this paper we describe the decision support func-
tionality of the Coordinated Multi-source Maintenance
on Demand (CMMD) system. CMMD is designed to
support the complete life cycle of mission plans for hu-
man space exploration, starting with initial long-term
planning and ending with day-by-day execution of a
detailed schedule. The goal of CMMD is not to re-
place human experts, but to assist them. To do so,
CMMD *explains reasons* for commitments it makes, al-
lows the user to *interactively explore alternatives*, *guide
the search* toward more desirable solutions, and to run
various *queries* (e.g., what courses of action have not
yet been explored with respect to some goal?). We
claim that giving users insight into workings of the
system and *gradually* enhancing existing processes is
crucial for gaining user confidence in produced plans
and ultimately for adoption of the system.

Introduction

Modern automated planning and scheduling algorithms
can support very expressive domain models, and of-
ten scale sufficiently for real-world applications. Most
such algorithms target automated systems, such as au-
tonomous vehicles and other equipment. However, pro-
ducing a good plan is not enough for humans to be able
to approve the plan. Users must be able to see that a
plan is valid and reasonably efficient. Even providing
post hoc explanations *after* the plan is produced is not
helpful because the plan is too complex to understand,
and any human concerns are too late to be addressed.
It is also hard to formalize human intuition that might
be necessary to produce better plans. What is needed is
simple, interactive understanding and control. Humans
must be able to explore alternatives within the plan
expansion process itself, assess them, run various diag-
nostic queries, and guide the system to more desirable
solutions without necessarily codifying plan evaluation
functions.

This paper describes our implementation of those ca-
pabilities in the Coordinated Multi-source Maintenance
of Demand (CMMD) system – a multi-agent decision
support tool for planning and execution of missions for
human space exploration, such as Shuttle flights, In-
ternational Space Station (ISS) increments, and future
lunar missions. Instead of replacing human planners,
CMMD assists them by (i) enforcing previously speci-
fied safety rules and standard procedures, (ii) showing
alternative actions, resource choices, and scheduling op-
tions, and (iii) allowing various diagnostic queries on
the schedule. The system can be tasked to compute a
solution automatically, but the user can intervene at
any point to request alternative solutions and/or to
guide the search process herself. Thus the user can
decide how much control of the planning process she
needs and how much she is willing to trust the auto-
mated system.

The rest of the paper is organized as follows. We first
describe the application domain to motivate examples,
followed by a description of the CMMD system archi-
tecture and its unique decision support capabilities. We
close by discussing related research and future work.

Human space exploration

The application domain of the CMMD effort is centered
around NASA's manned space operations led by the
Johnson Space Center (JSC) in Houston, Texas. The
Center is responsible for planning and day-to-day exe-
cution of the Space Shuttle and the ISS programs. As
the ISS' construction nears completion, JSC's manned
spacecraft center will evolve to play a major role in the
follow-on Exploration Systems program, now in its for-
mulative stages, which has the mission of constructing
a lunar outpost and ultimately establishing a human

presence on Mars, and beyond.

Manned space operations build upon established and tested rules and procedures. However, the problem is extremely dynamic compared to many planning domains due to the frequent need to accommodate new equipment and one-of-a-kind activities, and addition of new procedures. Sometimes, after careful consideration of alternatives, certain existing rules and constraints may be waved.

In planning ISS operations, the largest unit of time ordinarily dealt with is an *increment*, which is the time period that a specific crew lives aboard the station (currently about 6 months). Increment planning begins 12 to 18 months before launch, initially documenting high-level issues such as crew rotation and training, station construction phases, and required logistics supply flights. Over time, at prescribed intervals, the plan is methodically refined with more detailed tactical versions, each defined over increasingly shorter planning horizons, finally resulting in daily, executable schedules specified to the level of individual actions, such as meals for astronauts. The refinement process is not strictly top down, as high level goals may be added, dropped, or modified as more detailed plans show what is feasible.

A team of experts, called a *discipline*, coordinates each aspect of the increment – from equipment to medical science. Each discipline has its own set of *ground rules and constraints* describing standard procedures and safety restrictions, plus resource requirements that may conflict with other disciplines (e.g., on astronaut's time). The Lead Operations Planner/Flight Director has final authority for resolving any conflicts for the good of the mission, thus requiring disciplines to have a firm grasp on available options and their implications.

This concept of operations has a proven track record, and with JSC's successful legacy, it will likely be adapted incrementally in future Explorations Systems. Thus any new planning technology must be able to integrate with the existing tools and organizational structures. Due to the dynamic nature of the domain, it is very unlikely that an automated system will replace human experts. Rather, what's needed is an interactive planning tool that can switch between manual and automated modes and assist human users by (i) offloading computation intensive tasks, (ii) allowing diagnostic queries to reveal relevant information, (iii) providing intelligent rationale behind commitments, and (iv) facilitating exploration of alternatives.

The system presently used for planning of ISS missions, CPS (Saint 2002), provides a scheduler and allows multiple users to submit specific plan modification requests. However, it does not allow users to actively participate in the planning process. The CMMD system was designed to address these requirements.

CMMD architecture

To enable future space exploration operations, any planning and execution system should support:
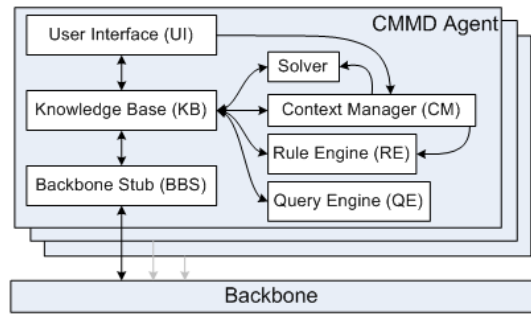

Figure 1: Architecture of CMMD Agent

- An extensible encoding of domain knowledge,
- Multiple time granularities and concern windows (e.g., detailed weekly plan and high-level plan for the next month in the same system),
- Multiple disciplines/roles,
- Distributed, asynchronous operations (consider communications delays to Mars),
- A user-centric approach to the planning process,
- A means of performing diagnostic queries, and
- Multiple ongoing what-iffing sessions.

To satisfy these requirements, the CMMD system is designed as a collection of agents communicating via a virtually centralized Backbone. The agents can be used by various disciplines (human users) or as bridges to external data sources, such as equipment health monitors. The Backbone is responsible for propagating information between the agents, for enforcing visibility restrictions and preferences, and for the transaction-safe persistence of data. In this paper we will focus on the user interactions with a single agent and will not discuss the distributed aspects of the CMMD system.

Agent modules

The architecture of a CMMD agent is shown in Figure 1. The Knowledge Base (KB) contains a working copy of the plan and performs constraint propagation. The data representation for plans is described in the subsequent section. Other modules of the agent can read and modify the data in the KB. The Backbone Stub (BBS) is responsible for communication with the Backbone, including propagation of data updates to and from the agent. The Rules Engine (RE) enforces applicable rules, the Query Engine (QE) locates user-specified subsets of a potentially overwhelming solution space, and the User Interface (UI) module is responsible for communication with the user.

As a plan evolves, changes can come from 3 sources:

1. New information can be added by the local user (via the UI module) or by external users or data sources (as propagated via the Backbone and BBS),

2. Implied changes can be enforced on already available data. This includes constraint propagation and rule application, and

3. The Solver module can perform search for a solution in the space defined by rules and constraints.

These modifications are applicable to a single plan instance. In order to enable exploration of alternative scenarios, CMMD supports simultaneous existence of multiple contexts (described later). The Context Manager (CM) manages the life cycle of these contexts as well as their communication with the Solver, RE, and KB. In addition to decisions made by CM, the user can explicitly initiate context operations for what-iffing.

Data model

The CMMD data model includes several first class entities: variables, constraints, conditions, resources, tokens, rules, options, and contexts. Resources and tokens also have types. Variables and constraints in CMMD form a constraint network. Both real-valued variables (intervals) and discrete variables (sets) are allowed. Conditions are similar to constraints, but condition propagation affects only the value of the condition flag and does not change domains of arguments.

Tokens are used to represent activities in the plan, as well as certain states of resources as a function of time (e.g. see the later *CMG functioning* scenario). Where it aids clarity, we differentiate between *tasks*, which are tokens denoting a collection of sub-activities, *actions* or primitive activities executed by individual resources, and *state* tokens over resource timelines. In addition to the usual start, end, duration, and resource variables, CMMD tokens may also have one *achievement* and zero or more *safety* variables associated with them. These variables take discrete *option* values, where each option represents a possible course of action for achieving a goal or for satisfying a safety constraint. Individual options are created and initialized by the Rule Engine as described below.

Suppose, an EVA task (extra-vehicular activity, or space walk) has just been added to the plan. This task is represented in the plan by a token. Initially, the domains of the start and end variables of this token are set to a wide window during which the activity should occur (e.g., a week). The initial domain of the resource variable for the EVA token will contain resource timelines corresponding to all astronauts present on the station and capable of performing the EVA. Suppose the standard operating procedures contain a safety rule requiring that the airlock be checked at most one day before the EVA. A safety variable will be created by the RE and attached to the EVA token to enforce this requirement. There is also a standard procedure defining that an EVA comprises the following sequence of actions: don suit, egress, work, ingress, doff suit. For simplicity, we omit resource constraints and some of the activities here and drop conditions. In this example, an option representing the proper sequence of these sub-activities will be bound to the EVA token's achievement variable. In some cases, there may be multiple legal procedures for the same high-level activity. In such sit-

```
safety rule id:"10001" {
  metadata(key:"Name", "No exercise after a meal)

  // Bindings and pre-conditions
  trigger:  meal Meal @ Person
  pattern:
    ex Exercise @ Person {
      should Greater(ex.start, meal.start),
      could Greater(meal.end, ex.start, -3h),
      should SetInclusion(meal.timeline, ex.timeline)
    }
  conditions:  = {should True()} // Always applies where binds

  // Effects
  tokens:  {} // no new tokens added
  constraints:  {TemporalArc(ex.start, meal.end, [90min, inf])}
}
```

Figure 2: Example rule

uations, the initial domain of the achievement variable may contain several, alternative options.

The plan domain representation in CMMD is rule based and similar to the concept of hierarchical task networks (HTN) (Erol, Hendler, & Nau 1994). CMMD rules come in two flavors: achievement and safety. Achievement rules correspond to traditional HTN decompositions and specify a *possible* way to achieve a goal. Safety rules prescribe additional, *necessary* conditions. In NASA terms, achievement rules correspond to standard procedures and safety rules to safety constraints or flight rules.[1] Each rule has a trigger token, a pattern possibly binding more tokens, a set of applicability conditions, and a partial network of newly created and/or reused tokens and constraints constituting a refinement to the constraint store. Figure 2 illustrates an example rule that declares one must wait at least 90 minutes after a meal before exercising. The trigger of this rule says the rule needs to be applied to every token of type Meal. The pattern says that all exercise tokens that can start within 3 hours after the meal need to be checked. There are no further conditions. For each such exercise token, a constraint is added forcing the minimum gap between the meal and the exercise. By adding the window condition in the specification of the pattern, the rule prevents creating constraints between every pair of meal and exercise tokens.

The Rule Engine expands applicable rules into individual *options*, which constitute the domains of safety and achievement variables of appropriate tokens. All possible ways to perform a given task are captured as options in the achievement variable of the corresponding token. Each applicable safety restriction is captured as a separate safety variable of the trigger token, and all ways to satisfy this restriction are collected as options in the domain of this variable. Safety variables also have conditions. For example, if in our above example the exercise token is pushed more than 3 hours away from the meal, the corresponding safety variable becomes disabled.

Figure 3 shows the graphic conventions we use in this paper. The placement of the variable circle on the token rectangle graphically distinguishes between different to-

---

[1]Domain description in CMMD is intentionally kept close to existing nomenclature in order to facilitate adoption.

ken variables (start, end, achievement, etc). The set of safety variables created for a given token is shown in a dashed box connected to the token.
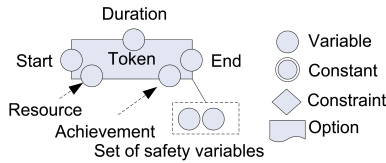

Figure 3: Graphic conventions used in this paper

Maintaining safety and achievement variables in the same way as temporal and resource variables levels the playing field for search algorithms: the solver is free to make decisions in any order, thus fully interleaving action choice (planning) and resource/time assignment (scheduling). When the domain of an achievement or safety variable becomes empty (meaning there is no way to achieve the goal or satisfy the safety requirements given other constraints), the constraint network becomes infeasible, thus inducing backtracking.

The last major concept of the CMMD data model is the *context*. CMMD contexts provide an abstraction for encapsulating speculative computations during search. Our implementation of contexts is functionally equivalent to that of O-Plan (Tate & Dalton 2003). The system actively maintains multiple contexts and exposes them to the end user, thus providing a mixed-initiative what-iffing capability. Due to the focus on explanations and what-iffing, the CMMD agent also maintains *reasons to exist* for various entities. Note that, since tokens may be reused by multiple rules, reasons to exist are not always unique. For example, a check airlock task may be required both to satisfy EVA safety requirements and as part of scheduled maintenance. Removing either reason for the task will not lead to removal of the check airlock task from the plan.

Suppose, `ContextA` is the current context. A new EVA task is added in `ContextA` and applicable rules are evaluated. Now suppose in `ContextA` there already exists a token of type `check_airlock` that may be scheduled close enough to the EVA to be reused. Alternatively, a new `check_airlock` action may be created. Given the available information, no decision whether to reuse the existing `check_airlock` token can be made, so the Rule Engine creates two options (see Figure 4). Only one standard procedure exists for EVAs, producing one option for the achievement variable.
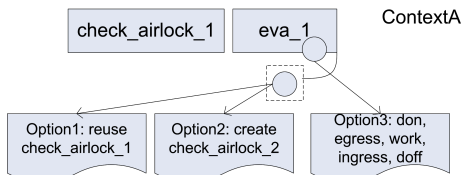

Figure 4: Options created for eva_1

Upon a user request, or due to a decision made by the search algorithm, a new context is derived from `ContextA`. In this new context, `ContextB`, the domain of the safety variable for token `eva_1` is reduced to a singleton value `Option1`. Propagation of this decision causes activation of the option, which in turn results in creation of a new temporal constraint between `check_airlock_1` and `eva_1` (Figure 5). The dashed line in the figure shows the reason to exist.
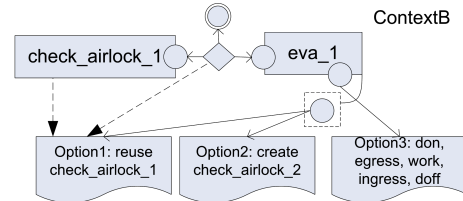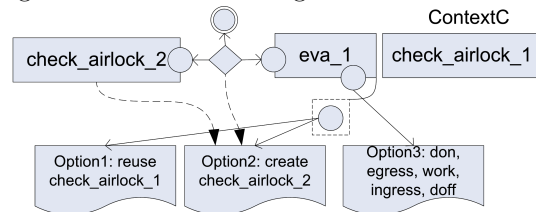

Figure 5: Reuse of existing check_airlock_1 token

Figure 6: Creation of new check_airlock token

To explore the other alternative, `ContextC` is derived from `ContextA`. Thus, `ContextB` and `ContextC` are sibling alternatives. In `ContextC`, the value of the safety variable for token `eva_1` is set to `Option2`. As the result, a new token `check_airlock_2` is created and constrained to precede `eva_1`. Note that in this context `check_airlock_1` has no ordering constraints with `eva_1` (Figure 6).

Token `check_airlock_1` was most probably added to the plan to satisfy a safety constraint of some other activity, such as docking of a transport spacecraft or scheduled maintenance, and therefore has temporal and resource constraints to some other tokens in the plan. Although `ContextB`, which reuses this token, has fewer actions, it has less flexibility, because it indirectly forces constraints between the EVA task and the original task related to the `check_airlock_1` token. One or both of `ContextB` and `ContextC` may turn out to be infeasible due to temporal or resource constraints. Even if both are feasible, however, they likely have different downstream effects on the plan at large. Thus, it may be beneficial to explore the consequences of both courses of action.

Decision support functionality

Context operations

The typical user interaction with a single CMMD agent proceeds as follows. The user loads an existing plan or creates a new, empty plan. This plan forms the root context, which defines available resources and high-level goals. The user can then perform operations on the context tree or within any leaf-level context.
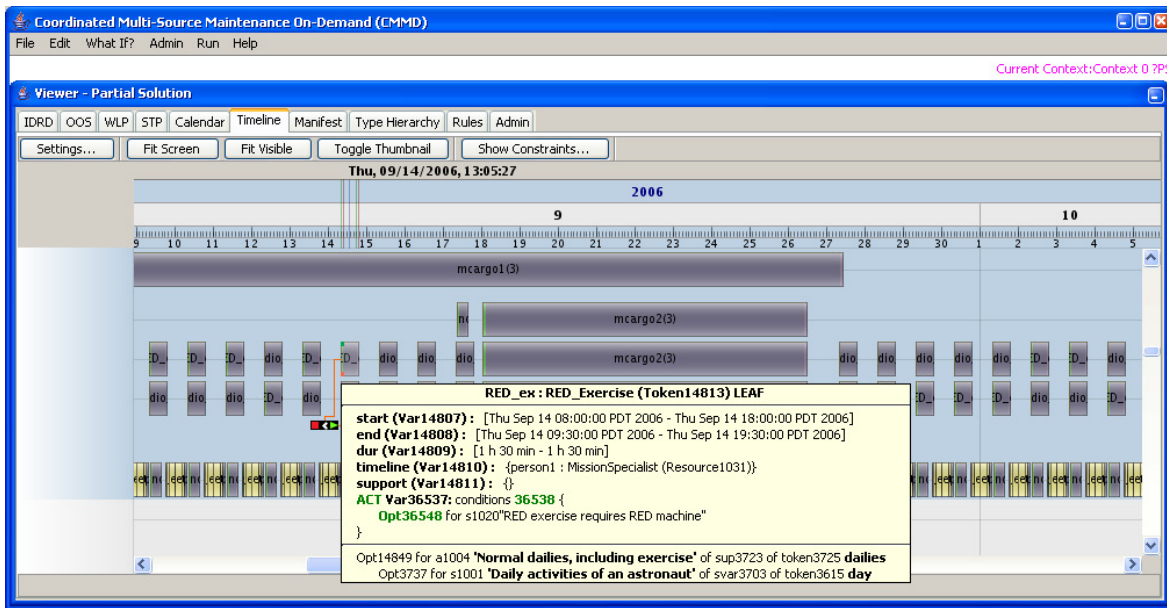
Figure 7: Part of an astronaut timeline in the 5-week plan after rules are enforced. The popup shows a resistive exercise device (RED) action. The action has been added to the schedule as part of the normal daily wake-time activities (achievement rule `a1004`) for every astronaut on board (safety rule `s1001`). The exercise task also triggered a safety rule `s1020`, which caused reservation of the RED machine for the duration of the exercise.

As mentioned earlier, contexts represent alternative solution paths (either partial plans or point solutions). A context can be *branched* (spawn a child) or *deleted*. Changes from a child context can be *promoted* to its parent, thus replacing the contents of the parent.

At any time there is a single context chosen as the *current context*. The user can modify the state of the current context by adding tokens and constraints and by manually reducing domains of variables. The user can also invoke various algorithms on the current context, e.g. request the system to automatically prune the domains of variables through *propagation*, to *enforce rules* or to *search* for a solution. Alternatively, the user can ask the Context Manager (CM) to automatically derive a solution by interleaving rule firing and search. This operation produces a context tree, which includes a solution compliant with all known rules and constraints, plus a tree of unexplored alternatives. The CM loop enforces that generated sibling contexts are unique by adding special *no-good* constraints.

Explanations and what-iffing

Consider the following scenario. JSC wants to plan for a 5-week period on the ISS involving two unmanned cargo shipments, a Shuttle mission, repair of two major sub-systems on the exterior of the station, and a solar experiment that must take place at a specified date and time. The initial plan contains only the high-level tasks, such as `dock_shuttle` and `run_solar_experiment`.

The user can let the system run the CM loop until a solution is found. Alternatively, she may decide to take a more active part in the search and exploration of alternatives. In this latter case, the next logical step is to instruct the system to apply all relevant rules. As the result, required actions, such as sleep and meals for the crew and airlock operations required for docking, are enforced and all viable ways to perform tasks are collected (Figure 7).

At this stage, the user can use CMMD's interactive features to explore the set of enforced safety constraints (and their effect on temporal windows and possible resource assignments) and applicable procedures. In situations where there is only one way to achieve a goal or satisfy a safety constraint, it will be enforced immediately, and required actions and constraints will be added to the plan. However, the Rule Engine does not make a choice when multiple options are available.

Suppose, a safety rule for solar experiments requires the station to be properly positioned before the experiment starts. This safety rule fires immediately, and a `turn` task is added to the plan. This is an *abstract* task, hence it cannot be directly performed. Instead, some procedure should be followed to achieve the desired outcome. Until a particular procedure is chosen, it is impossible to say how long turning will take or when it should start. All that is known is the turn goal should be satisfied before the solar experiment starts. There are two possible ways to turn the station: using Control Moment Gyroscope (CMG) or using thrusters. If both options seem viable, the Rule Engine does not make a choice about which option to use – this is the job of the search algorithm or the user.

If we let the search algorithm make the choice, it will go with the first viable alternative – use the working

CMG to turn the station. This option would succeed if the CMG is *functioning* at the time of the turn, which depends on the CMG being repaired before, which in turn depends on the robotic arm being repaired, which depends on the Shuttle arriving in time with the spare parts. The user can obtain all these causal dependencies using the CMMD user interface.

Suppose the user wants to explore the situation when the CMG repairs are unsuccessful. One way to model this is to declare that the CMG is not functioning. The user creates a branch of the current context and by adding a new constraint, limits the duration of the state token `functioning` on the CMG resource timeline.

This modification, when propagated, renders the use of CMG for turning the station infeasible. Only one option is left – to use the thrusters – and the next time rules are enforced necessary tasks and constraints between them are added to the system. The user can then compare this new plan with the plan that relied on having a functioning CMG and decide whether committing to the more conservative solution is a good idea.

Exploring alternatives

In the earlier example, the user created new contexts to explore alternative ways to turn the station. When the Context Manager runs in autonomous mode, interleaving rule enforcement and making choices, it also creates multiple context branches. In particular, the CM periodically creates *checkpoints* by adding two branches of the current context: the working branch where the choices are made and an alternative branch where the same choices form a nogood. During an automated search, the CM can use the alternative branches for backtracking. Even if the autonomous loop finds a solution, the user can use the preserved alternative branches of the context tree to explore other alternatives.

For example, suppose instead of manually exploring scenarios with a working or broken CMG the user instructs the CMMD agent to find a point solution, i.e. a total assignment for all discrete and numeric variables. If the solver selects using the CMG for turning the station, this would lead it to a valid solution, so the solver will not automatically explore other alternatives. However, after the search finishes, the context tree contains an unexplored context with "use CMG to turn the station" listed as a no-good. The user can then instruct the CMMD agent to find a solution for this context.

The CM implements several strategies for branching and interleaving calls to the Solver and the RE. In addition, the user can explicitly create CM scripts by specifying the number of steps or termination conditions for each invocation of the modules. For example, such a script can specify that the Solver should first assign singleton values to all achievement and safety variables one variable at a time, and the RE should apply all relevant rules after each such assignment. After that, the Solver should make decisions about 10 variables at a time. Note that scheduling decisions can make new safety rules applicable.

The number of contexts produced by autonomous CM mode can be very large. The scripting feature can also be used for navigating the context tree. For example, the user can search for all contexts where the achievement variable of the turn task is first bound to a singleton, or for contexts where the use of CMG is removed from the list of options for this variable, or for contexts where the arrival time of a cargo ship is limited to a window of at most 5 days. The user might then desire to inspect said contexts separately or to display them side by side for comparison.

Related work

In this paper we discussed interactive features of CMMD: providing explanations, exploring alternatives, and controlling the search process. We review related work with respect to these features.

The ability to discover reasons behind system's choices can be used to debug constraint based reasoning engines (Daley *et al.* 2005). Beyond debugging, our focus is the integration of a planning tool into existing human processes. We seek to address the adoption hurdle problem, by a) slowly gaining users' trust via on-demand decision rationales and b) allowing end-users to refine the correct behavior of the system by incrementally building up and maintaining the declarative, corporate knowledge base that drives the decision support process itself.

(Smith *et al.* 2004) generate *explanations* for inconsistencies in simple temporal networks (STN) that suggest a possible relaxation of constraints, which make the STN feasible. In (Bresina & Morris 2006), explanations pinpoint inconsistent constraints added automatically by the search engine to encode arbitrary ordering decisions in the STN for any activities that are defined as disjoint. CMMD's solver component has a similar ability to discover conflict sets in inconsistent constraint networks; we call these *violations*. By contrast, our explanations describe why the problem was formulated as it is and why decisions were made.

The second discussed feature is the ability to explore alternatives. The Barrel Allocator system described in (Kramer & Smith 2002; Becker & Smith 2000) provides mixed-initiative "what-if" capabilities for exploring solution spaces in the domain of airlift transport operations. The planning aspects of this work, however, are limited to potential optimizations of resource usage by combining unrelated transport missions that are nearby in space and time.[2] In contrast, since CMMD allows end-users to think in terms of high-level goals, and there are frequently multiple, functionally equivalent operational plans that will accomplish individual goals, CMMD handles a great deal of reasoning about the planning aspects of scheduling problems. Finally, their system implements what-if exploration via lin-

---

[2]The challenge here is similar to the inefficiencies of returning to one's home base with an empty trailer in the over-the-road trucking domain.

ear undo/redo operations, rather than the more general branching mechanism of our contexts.

Another mixed-initiative, constraint-based scheduling application is NASA's MAPGEN, which derives daily activity plans for the Mars Exploration Rovers. This system does interleave planning operators into its scheduling process to some extent, but according to (Bresina *et al.* 2005) abstract tasks generally have static expansions. Context-dependent alternatives over suitable planning operators were rare and typically added manually. Nor does MAPGEN appear to support multiple, simultaneous what-if branches.

We discuss search control in the next section.

Discussion and future work

Real-world deployment of planning/scheduling systems requires gaining user trust and integrating new systems with existing processes. The user should be able to switch from the preexisting approach (manual or based on another product) gradually, at her own pace.

The CMMD project aims to create a planner/scheduler to assist a human expert in dynamic, mission critical domains such as space exploration. To achieve this goal, CMMD provides user interface functions for obtaining reasons for the system's decisions and allows the user to interactively explore alternative options and to guide the system toward a more desirable solution. In CMMD, we don't seek to build an automated system that necessarily knows more about the problem domain than an expert. Rather, we expect it to primarily assist its human masters, who will "only truly know what they want, when they can see what they could get".

Complex domains such as space exploration are simply too rich to be correctly and completely modeled. Active involvement and frequent feedback from expert users offer the only workable solution for addressing this formidable issue. To achieve the necessary flexibility, CMMD relies on extensible libraries of domain rules and procedures layered upon efficient, domain-agnostic, deductive inference procedures. The initial implementation of our rulebase language is arguably too unwieldy for average subject matter experts. In follow-on efforts, we expect to refine the syntax and semantics of the language. Additionally, it is likely that providing simplified wrappers (at the expense of expressivity) and/or graphical languages for the generalized rules facility is beneficial for common use cases.

To facilitate adoption, an intelligent system should provide continuous spectrum of control, from almost fully manual to fully autonomous. Even when operating in fully autonomous mode, the system's ability to elucidate reasons for its commitments (or backtrack-inducing failures) is important for gaining user acceptance (in the case of correct answers) and for identifying limitations in the system's knowledge of the problem domain (for incorrect and sub-optimal answers, in the face of a previously unseen situation). Integration with a simulation environment for evaluation of plan sensitivity could also help further increase user confidence.

In domains where multiple solutions are possible, it is useful to give the user interactive control over the search process. The interactive nature of CMMD allows incremental exploration and the opportunity to redefine preferences on the fly. We believe that any planner, which must cope with evolving problem domains, should also support run-time configurable meta-reasoning heuristics. The CM scripting feature already allows control over such heuristics, but there is room for extensions. (Myers & Morley 2003) offer some related perspectives on how quite powerful facilities for user-defined guidance might be implemented. Effective libraries of such meta-control rules would also facilitate the inter-agent negotiations of a distributed problem solving system such as CMMD.

CMMD's interactive *explanation, exploration, and control* features aim to increase user confidence in produced plans and thus to facilitate the system's adoption. CMMD's architecture provides a good base for the extensions outlined above.

Acknowledgments

References

Becker, M., and Smith, S. 2000. Mixed-initiative resource management: The AMC Barrel Allocator. In *Proc. of International Conference on Automated Planning and Scheduling (ICAPS)*.

Bresina, J. L., and Morris, P. H. 2006. Explanations and recommendations for temporal inconsistencies. In *5th International Workshop on Planning and Scheduling for Space*.

Bresina, J. L.; Jónsson, A. K.; Morris, P. H.; and Rajan, K. 2005. Activity planning for the Mars exploration rovers. In *Proc. of International Conference on Automated Planning and Scheduling (ICAPS)*.

Daley, P.; Frank, J.; Iatauro, M.; McGann, C.; and Taylor, W. 2005. PlanWorks: A debugging environment for constraint-based planning systems. In *1st International Knowledge Engineering Competition*.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. Semantics for hierarchical task-network planning. Technical Report CS-TR-3239, University of Maryland.

Kramer, L., and Smith, S. 2002. Optimizing for change: Mixed-initiative resource allocation with the AMC Barrel Allocator. In *Proc. of the 3rd International NASA Workshop on Planning and Scheduling for Space*.

Myers, K. L., and Morley, D. N. 2003. Policy-based agent directability. In Hexmoor, H.; Falcone, R.; and Castelfranchi, C., eds., *Agent Autonomy*. Kluwer Academic Publishers.

Saint, R. 2002. Lessons learned in developing an international planning software system. In *Proc. of SpaceOps 2002*.

Smith, S.; Cortellessa, G.; Hildum, D.; and Ohler, C. 2004. Using a scheduling domain ontology to compute user-oriented explanations. In *Planning, Scheduling, and Constraint Satisfaction: From Theory to Practice*. IOS Press.

Tate, A., and Dalton, J. 2003. O-Plan: a Common Lisp planning web service. In *Proc. of the International Lisp Conference*.