ICAPS 2005
Monterey, California

ICAPS05

WS3

Workshop on Mixed-Initiative Planning and Scheduling

George Ferguson
University of Rochester, USA

Caroline Hayes
University of Minnesota, USA

Greg Sullivan
BAE Systems Advanced Information Technologies

ICAPS 2005
Monterey, California, USA
June 6-10, 2005

Cover design: L.Castillo@decsai.ugr.es

Workshop on Mixed-Initiative Planning and Scheduling

George Ferguson
University of Rochester, USA

Caroline Hayes
University of Minnesota, USA

Greg Sullivan
BAE Systems Advanced Information Technologies

ICAPS 2005
Monterey, California

Workshop on Mixed-Initiative Planning and Scheduling

Table of contents

ICAPS 2005
Monterey, California

Workshop on Mixed-Initiative Planning and Scheduling

Preface

The goals of mixed initiative planning and scheduling (MIPAS) systems are to combine the strengths of humans and computers so as to produce joint systems that exhibit better performance than either alone. A wide variety of domains, including manufacturing, military logistics, space, and transportation are finding both manual and automated approaches to planning and scheduling inadequate. A manual approach, involving large teams of highly trained experts, does not scale well to increasingly complex scenarios, and remains expensive, time consuming and error-prone. Automated planning and scheduling systems, though long a focus of AI research, are difficult to integrate into human-centric activities, for both technical and psychological reasons. Thus there is increasing interest in mixed-initiative planning and scheduling systems, where humans can apply their expertise, intuition, and exercise an appropriate level of control, while computing resources can be leveraged to assist, learn, verify, generate options, and control resources, using ever more sophisticated algorithms and more powerful hardware. However, few successful MIPAS systems have been fielded to date. Challenges include identifying the parts of a complex task where MIPAS systems might have the greatest added value; developing planning and scheduling algorithms that can simultaneously meet humans' requirements for speed, feasibility and quality; presenting and communicating concepts effectively between human and computer, gaining user acceptance, and assessing impact on overall planning and scheduling performance.

This workshop will address the challenges of mixed-initiative planning and scheduling, discussing past work, current needs and future directions.

Organizers

- *George Ferguson (U. Rochester)*
- *Caroline Hayes (UMN)*
- *Greg Sullivan (BAE Systems)*

Programme Committee

- *John Bresina (NASA ARC)*
- *Mark Burstein (BBN)*
- *George Ferguson (co-chair) (U. Rochester)*
- *Caroline Hayes (co-chair) (UMN)*
- *Karen Myers (SRI)*
- *Dana Nau (U. MD)*
- *Debra Schreckenghost (NASA JSC)*
- *Stephen Smith (CMU)*
- *Greg Sullivan (co-chair) (BAE Systems)*

A Framework for Designing and Evaluating Mixed-Initiative Optimization Systems

Arthur E. Kirkpatrick, Bistra Dilkina and William S. Havens
School of Computing Science
Simon Fraser University
Burnaby, British Columbia
Canada V5A 1S6
{ted, bnd, havens}@cs.sfu.ca

Abstract

Mixed-initiative approaches are being applied in combinatorial optimization systems such as planning and scheduling systems. Mixed-initiative optimization systems are based upon collaboration between the system and the user. Both agents possess unique and complementary abilities which can be jointly applied to intractable optimization problems. Yet current approaches to designing and evaluating these systems remain *ad hoc*. In this short paper, we give a precise definition of a mixed-initiative optimization system. We identify the salient characteristics of combinatorial problems which make them suitable candidates for mixed-initiative reasoning. We provide a framework which informs both the design and evaluation of these systems. Using this framework, we characterize the functional requirements of any mixed-initiative optimization system. These requirements can help to establish suitable evaluation criteria for these systems. We conclude by situating recent work in this area within our framework.

Introduction

Mixed-initiative optimization (MIO) systems are systems in which the user and system collaborate to solve combinatorial optimization problems, such as planning and scheduling (Howe *et al.* 2000; Kramer & Smith 2002; Scott, Lesh, & Klau 2002). The benefits of MIO systems have been broadly claimed in the literature. The arguments are based upon several subordinate claims. Two experts ought to be better than one for solving complex combinatorial optimization problems. The system and the user possess unique expertise, and each complements the other. The division of labour between these experts should reflect their inherently different capabilities. The automated solving methods deal with the combinatorics of optmization problems, while users have different kinds of expertise. Often the user of a MIO system will be a professional in the field of application and consequently will know aspects of the problem not adequately modelled by the system. For example, some important constraints may not be part of the model, or some preferences on solutions may not be coded in the objective function. Furthermore, the user's experience may suggest directions for finding good solutions in the search space. Mixed-initiative designs allow this expertise to be incorporated into the problem solving process.

We share the general enthusiasm for mixed-initiative systems. Yet we are concerned that beneath this large tent is hidden a broad range of systems with potentially quite different properties. Blanket statements about "mixed-initiative systems" may only apply to some fraction of these systems. The methods for evaluating these systems remain *ad hoc* and there is little advice available on such issues as the functional components of an effective MIO system, or the performance evaluation of these systems. A common ground is needed for discussions of design and evaluation.

Such a common ground is particularly important for evaluation. There have been several recent evaluations of mixed-initiative systems. Most of these studies have aimed to demonstrate that mixed-initiative systems can be advantageous. For example, Klau *et al.* (2002a) write,"our goal is to show that *some* people can guide search, not that most people can" (p. 46, emphasis in original). The specific mechanisms by which mixed-initiative systems enhance performance remain poorly-understood, although Scott, Lesh, & Klau (2002) have made a promising start. We locate seven limitations in current ad hoc approaches to evaluation:

- We lack precise terminology to distinguish mixed-initiative systems and applications. This makes it difficult to generalize results from one study to a broader class of situations.
- We have a multiplicity of goals, which may overlap or contradict one another.
- We lack clear metrics of progress.
- We have limited means of systematically organizing results to date and those of the near future.
- The lack of precise terms and clear metrics makes it difficult to state clearly falsifiable hypotheses for our research.
- We have no conventional protocols that researchers can use to evaluate a new system.
- Researchers have to account for too many variables when constructing a study.

The systems are being built, and the need is acknowledged.

However, we lack a bigger picture in which to locate specific studies.

Our overall goal is to suggest more precise definitions for the mixed-initiative optimization systems community. Specifically, we make the following contributions in this paper:

- We define the scope of mixed-initiative optimization (MIO) systems.
- We present a general framework that can inform both the design and evaluation of effective MIO systems and identify the main parameters of the problem space and use them to categorize the functional requirements of a MIO system.
- We situate other research results in MIO within our framework. The framework suggests the range of applicability of previous work and can be used to identify potential confounds.

We end with a description of how this framework might be used to structure future research in MIO systems.

Overview of the Framework

We call our proposal a framework rather than the more demanding terms "model" or "theory". We have in mind an analogy with the framework of a house, which provides a structure to support both the work in progress and the finished product. Parts of a framework may be modified as construction proceeds and parts may be thrown away when they are no longer needed. We suggest the mixed-initiative optimization community can benefit from undertaking development and evaluation in this coordinated way—a common framework for developing tasks and protocols, and for interpreting results.

Within such a framework, researchers can begin detailed analysis, extending questions of evaluation beyond, "Is mixed-initiative optimization possible?", to "When and how does it help, and by how much?", and extending systems design beyond "I think this will be useful", to "Previous research gives this a high likelihood of being useful for these applications".

Evaluation is key to this process, but it is hard to get right and requires great effort. As a practical matter, we must break the evaluation process down into smaller pieces. Given the high dimensionality of the design space, experimental manipulations requires separable research questions. Generalization from evaluations of individual designs requires the ability to locate each design within a larger space. By suggesting the questions to ask about a specific design, the framework provides that generalizability and the interconnection of these results with others. In particular, the community will benefit from standardized evaluation protocols. This allows us to build and extend research systems without doing a full evaluation every time.

Definition of MIO Systems

We begin with a definition of a MIO system. A *mixed-initiative optimization system* is an optimization system featuring both:

- interleaved contributions by the user and the system, together converging on a solution to a single problem.
- asymmetric division of labour such that the contributions made by the computer and the user are distinct.

This definition identifies the characteristics of systems for which a common technology can be developed. It circumscribes our shared field of interest. In particular, it separates the goals of this community from the goals of the user modeling community, which emphasizes different aspects of mixed-initiative systems. We are not arguing that user modeling is incompatible with MIO systems, nor that it is irrelevant to some applications of those systems. We are simply emphasizing that user modeling addresses issues that are independent of the issues common to all mixed-initiative optimization. A given application my need one, the other, or both.

Framework Top Level

At the highest level, our framework describes the context in which the system is used and the system itself. Optimization systems ultimately serve human needs, and thus the context describes the system operators and the social context of their work. It includes such issues as who interacts with the system, what others expect from them, and how they do their job. The context introduces requirements that the resulting system must satisfy. We break the context into two parts, the problem domain and the operator expertise.

For our purposes, the mixed-initiative system can also be considered to have two fundamental parts, the interactive visualization and the solver. We emphasize that this is not intended to be a full description of the architectural options available to designers. Rather, our purpose is to list the high-level system components which will most directly be evaluated. Designers devote most of their attention to the system, and evaluations are likely to compare instantiations of various combinations of them, with little consideration of the context. This is fine—in fact, there is no practical way to cover all possible contexts in a single evaluation. We simply recommend that evaluations explicitly specify the details of their intended context. This will permit readers to determine the range of contexts to which the evaluation results may be generalized.

Properties of Context

The framework's elaboration of social influences on a mixed-initiative system emphasizes that the requirements for that system are strongly shaped by the needs and nature of the organization employing it. Each social component has several properties, and the system requirements are derived from the properties (see Table 1). We will describe each property in turn.

Domains with synchronous collaboration feature teams of individuals working in the same room to solve the problem. The classical (and widespread) example would be several individuals standing at a large whiteboard, discussing, writing, and annotating. Mixed-initiative optimizers for such domains will benefit from having interactive displays that per-

Area	Property	Derived Requirement
Domain	Synchronous collaboration	Simultaneous review and update
	Asynchronous collaboration	Traces left for handoff to others
	Unmodellable aspects	Adding specific constraints and revising solution
	Level of constrainedness	Choice of optimization algorithm
	High dynamism	Rapid solution revision
	Answerability	Explanation
	High stakes	High scrutiny, human approval, explanation
	Task givens and goals	Data model chosen
	Task flow	Activity design
Operator expertise		
Domain-independent	Visual grouping	Display proximity, object similarity
	Conceptual models of other programs	Compatibility with other programs
Domain-specific	Models of objects and relationships	Data model
	Heterogeneity of approaches	Diversity of representations and interaction styles

Table 1: Contextual properties in the mixed-initiative optimization framework.

mit simultaneous review and update of the current solution by multiple users, in the same way that a whiteboard can.

By contrast, asynchronous collaboration features team effort, but with individual team members working at different times. For example, there may be only a single operator, but the work is turned over from one shift to the next. In these cases, the interactive display will not need simultaneous review and update, but will instead benefit from providing a mechanism for the first shift to leave traces of their choices and pointers to ongoing problems.

Domains with unmodellable aspects have problems that are difficult or impossible to completely represent in the model. There are a wide variety of reasons why a problem feature may be missing from the model. Amongst other reasons, the feature may be so specialized that the modellers forget to include it until they see a "solution" that violates it, the feature may not be representable in the modeling formalism, or the feature may be so specific that it would be impractical to represent its many permutations in the model. Indeed, given the considerable mixed-initiative folklore about incomplete models, it might be argued that every domain has unmodellable aspects. In any event, domains with unmodellable aspects will benefit from systems that allow the operator to add specific constraints and call for a revised solution.

The level of constrainedness of the domain imposes requirements on the system. If the domain is highly-constrained, the system should use a more sophisticated optimization algorithm. If the domain instead is only weakly constrained, the choice of solver might be different or almost irrelevant.

If the domain is dynamic, with requests frequently added and withdrawn, the system will be required to generate revised schedules rapidly.

If the operator is answerable to others for the choice of plan, the system should provide features that facilitate explanation. These could take the form of annotations and other tools that highlight specific aspects of the plan.

High stakes domains have outcomes that are considered critical by the participants. For such domains, the system should support high levels of operator scrutiny and a final human approval before the plan is carried out. These domains will likely also have high answerability as well, and so the system should also facilitate explanation.

The task statement imposes givens (initial conditions) and goals (the form of the desired solution) upon a domain. Different sets of givens and goals will often apply at different times for a single domain. For example, an airport gate scheduling system may be used in two different modes. First, the system could prepare a master schedule by allocating planes to gates under the assumption that every flight arrives exactly on time. In this case, the givens are the complete plane list and the goal is a complete schedule. Then, on a specific day, the plan would be revised as notices of delays arrived. This second task has different givens (just the planes whose schedules have changed, together with the original master schedule) and a more specific goal (accommodate the delays with minimal disruption to the original plan). Different tasks may require different data representations in the interactive display.

The final domain property is the task flow. Operators will often perform a task in a sequence of steps. For example, they may review all assignments of low-priority items before all high-priority ones (or vice versa). The operator task flow imposes requirements on the system's activity design, the steps that the system requires the user to perform. For example, a system that presented items to the operator in random order would be extremely frustrating for an operator who wished to review them in priority order.

Operator Expertise

Arguments for the use of mixed-initiative optimization systems often emphasize the unique expertise offered by the user. We suggest that this expertise has both domain-independent and domain-dependent properties. The domain-independent expertise consists of the human perceptual skills. The primary interactive display of most MIOS is visual, capitalizing on human skills of grouping visually-related objects. An effective MIOS should display the problem in a way that allows the operator to draw useful conclusions from object proximity and similarity. Note that even within a given domain, different tasks, with their different givens and goals, might be best served by different displays.

A second form of domain-independent expertise is the operator's experience with the conceptual models of other systems. If the MIOS has a conceptual model that matches that of other software commonly used by the operator population, the operators will find the system congenial. Although this form of expertise is unlikely to have a strong positive impact on the effectiveness of a MIOS, it can have a strongly negative impact. System effectiveness can be substantially reduced if the operators are practiced with a conceptual model that is incompatible with the MIOS they are using.

The domain-specific expertise of the operators will also impose strong requirements on the system design. From both training and experience, operators will think in models of objects and relationships. The structure of these models is highly domain-specific, making it difficult to give specific details. We simply note that the data model presented by the MIOS should be well-matched to the models used by the operators.

The final property in our categorization of expertise is the heterogeneity of solution approaches. An operator may have several ways of solving a problem and different operators within the same community may use different approaches. This is often domain-specific. Some operations communities have strong conventions that all operators are trained to observe, while other communities may be more diverse. We caution designers not to rely too strongly on perceived homogeneity in a community, as even within highly-trained groups there is often subtle variation. In general, an effective MIOS should provide a diversity of views, representations, and interaction styles, to support a diversity of solution approaches.

We have described these contextual properties in detail to emphasize the diversity of contexts in which MIOS might be applied. The effectiveness of a mixed initiative system depends upon how well it is matched to the specifics of the domain and the expertise of the operators. Interpretation of evaluation results must take these factors into account. An otherwise perfectly effective system may have poor performance if it is evaluated on a domain whose requirements are ill-matched to the system's. Results from an evaluation will best generalize to applications whose domains and operator expertise are similar to those of the evaluation.

Properties of Systems

Our model of system properties is deliberately simpler than our model of context. For purposes of summarizing the properties of a system that have the largest effect on evaluation, we break the system down into two parts, the interactive display and the solver (see Table 2).

It is not our intent to list all possible features and properties of mixed-initiative optimization systems. Defining these features and properties is a significant part of the overall research program in these systems. We offer this list as a starting point.

The interactive display has three main properties. The visualization is the visual representation used to display the problem statement and the current solution. An essential outcome of this design is the data model presented to the user.

The second property is the chosen interaction techniques. These will have specific speeds and place certain attentional loads on the operators. An ideal interaction technique will be fast and require so little explicit attention that the operator's reasoning about the problem will not be disrupted. Actual interaction techniques require some compromise of these ideals.

The third property of the interactive display is the detailed visual display design. These choices will determine where and how much the operators can apply their domain-independent expertise to the problem. This includes choices of which aspects of the problem will be represented in close proximity and how data values will be encoded.

The second component of the system highlighted in our framework is the solver. There are many ways of categorizing solvers, and development of new variations is an active area of research. Given that MIOS researchers typically have a strong understanding of the properties of various solvers, we only present here two example properties, showing how they may be connected to the contextual issues described earlier.

Solvers are often categorized as implementing either systematic or local search. Seen in terms of their relationship to the contextual properties described above, the main outcome of this distinction is their suitability for dynamic problem domains. Systematic search, while offering the potential for higher optimization, is less likely to be responsive to shifting requirements. Local search is more likely to apply in these domains.

A second property of a solver is how close its solutions lie to the optimal. Optimality is likely to be of higher value in domains that are capital-intensive , but may not be a possibility for highly dynamic domains, whose volatile constraints make it difficult to even define optimality.

Previous Work

The framework provides a structure for organizing discussion of the research results to date on mixed-initiative optimization. In this section, we review many of these results and locate common threads and unexplored areas in the field.

We start by considering the user modeling community's work on mixed-initiative systems. This literature is concerned with rather different issues than mixed-initiative optimization.

Horvitz (1999) proposed 12 principles for the effective integration of automated reasoning and direct user control. The goal of this integration was an agent that could act as a "benevolent assistant" (p. 160) to the user. The parameters of such an agent are rather different from those of the mixed-initiative optimization systems described in this pa-

Component	Property	Outcomes
Interactive display	Visualization	Data model
	Interaction techniques	Speed, attentional load
	Visual display	Proximity of views, coding of values
Solver	Systematic vs. incremental	Suitability for volatile domains
	Optimality	Quality of solution in highly subscribed domains

Table 2: System properties in the mixed-initiative optimization framework.

per. Assistive agents are expected to have transparent algorithms that perform actions the user also has the resources and representations to perform. The goal is to relieve the user of tedious, repetitive actions. From this perspective, the system has the initiative most of the time and needs to decide when to engage the user based on a user model—a set of beliefs about the abilities, goals and intentions of the user.

Fleming & Cohen (2001) develop guidelines for the design and evaluation of mixed-initiative systems. However, they start with the assumption that the central problem is how the system will take the initiative to request assistance from the user. They propose an approach similar to Horvitz, based on having an explicit model of the user's intentions and abilities.

The work on assistive agents is explicitly excluded by our definition, because such systems do not have an asymmetric division of labour between user and system. In contrast to assistive agents, mixed-initiative optimization systems are designed to produce degrees of optimization that the operator simply could not achieve unaided. Their algorithms are unlikely to be transparent, and their choices may require considerable effort for the operator to understand. The scheduling task is a primary focus of the operator's job and is likely to be her highest priority task. Indeed, the description of the human partner as an "operator" rather than a "user" emphasizes this primacy of the task.

Rich, Sidner, & Lesh (2001) cast human-computer interaction in terms of a collaborative dialogue process between the user and an intelligent interface agent. The authors base their approach to mixed initiative on human collaboration. They argue for an interface agent that engages in a discourse with the user in a similar way that the user would engage with another human. In particular, they argue that an intelligent user interface has to support the following questions:

- Who should/can/will do ____?
- What should I/we do next ____?
- Where am/was I ____?
- When did I/you/we do ____?
- Why did you/we (not) do ____?
- How do/did I/we/you do ____?

Rich et al. propose an intermediate level of software explicitly concerned with managing these questions.

We consider Rich et al.'s questions complementary to our framework, as they are more abstract and at a much higher level. We believe that many of the crucial elements for effective MIO system design lie in its rich, specific context. The questions will best be framed in that context, which may be difficult or impossible if the algorithm that generates the question is insulated from the context.

Howe *et al.* (2000) present a study on mixed initiative scheduling for the Air Force satellite control network. This scheduling problem is oversubscribed—no feasible schedule can satisfy all the requests. They propose a MIO system where the system finds a good but infeasible solution and lets the user negotiate the infeasibilities. A mixed-initiative approach is appropriate for this application because it is hard to express the true objective with a weighted linear sum of criteria, and because the dynamic arrival of emergency requests changes the problem specification as the solver runs. The authors point out the limited number of designs in the research literature mixed-initiative systems. They incorporate in their prototype some of these designs: providing an interactive Gantt chart where the user can interact with a schedule at an abstract, graphical level that hides schedule implementation and optimization details to an appropriate degree; and allowing the user to change the schedule, then call the scheduler to propagate the effects and to optimize, if possible. In terms of our framework, this paper emphasizes the dynamism and highly constrained nature of their domain. Their comments about the limited number of available design ideas and their use of Gantt charts demonstrate the importance of domain-specific models of objects and relationships.

Kramer & Smith (2002) describe the AMC Barrel Allocator, a mixed-initiative resource allocation tool for airlift and air-refueling management. They argue that a dynamic environment is not the only reason for using the mixed-initiative approach. It is also necessary to achieve the transition from manual to fully automated system. Kramer & Smith emphasize that a mixed-initiative optimization system allows a continuum of automation. In deploying their research to production, they found that operators must first gain trust and understanding of the system by inspecting solutions and performing what-if scenarios trust a system before they will accept it in a mission-critical workflow. Kramer and Smith point out that one of the main functional requirements for a mixed-initiative system is to provide explanations for system decisions. In terms of our framework, this paper emphasizes the dynamism, high stakes, and answerability of their problem domain, and argues that the MIO system must be well-matched to the task flow.

Klau *et al.* (2002b) present the HuGS Platform, a toolkit that supports development of human-guided search systems. They discuss four different applications built in the HuGS platform: a graph layout problem that minimizes edge crossings between nodes, a modified version of the travelling salesperson problem, a simplified version of the protein folding problem, and a jobshop application.

They present several motivations for mixed-initiative optimization. First, users need to understand and trust the generated solutions in order to effectively implement, justify and modify them, what we would call the answerability of the system. Second, the problem model usually includes only partially specified constraints and criteria, what we call the unmodellable aspects of the domain. They argue that a mixed-initiative system also leverages on human abilities that outperform the systems: visual perception, learning from experience, and strategic assessment. These strengths range over properties of both domain-dependent and domain-independent expertise.

Each application in the HuGS platform provides visualizations to display the current solution to the user for inspection and modification. Klau et al. argue that the usefulness of the system depends highly on the quality of visualization and recommend visualizations that highlight differences from the previous solution. They suggest an evaluation of the quality of the visualization by running a series of experiments on the same problems for the same time, and using two different visualizations. They propose a visualization quality metric of the number of optimal solutions that users are able to produce. In terms of our framework, these evaluation methods are focused on the interactive display component. Klau et al. end by highlighting some ongoing challenges for the mixed-initiative systems: large-scale problems where the whole solution cannot be viewed at once (again, located within the interactive display component of our framework), and mixed-initiative systems where there is more than one human user (synchronous collaboration).

Scott, Lesh, & Klau (2002) give a lucid outline of the benefits of using a mixed-initiative optimization system. Their research focuses on evaluating a specific aspect of mixed-initiative optimization systems. The authors argue that the design of interactive optimization systems needs input from experiments focused on determining which optimization subtasks are best suited to the strengths of the human and which are most appropriate for the computer. Their study examines several user tasks within a mixed-initiative optimization system for vehicle routing and compares users' performance in these tasks to the performance of the computer on the same tasks. They evaluate the users' contribution on three different subtasks: focusing search through mobilities, finding targets that guide the search towards better solutions, and controlling computational effort by halting the search. Their studies suggest that people are especially effective at managing how computational effort is expended in the optimization process and at focusing short searches. However, the experiments showed that humans were somewhat less effective at visually identifying promising areas of the search space.

In terms of our framework, the work of Scott et al. is motivated by the contextual concerns of answerability and the unmodellable aspects of the domain. Because their experimental participants were not vehicle routing specialists, the evaluation focused on HuGS' support for domain-independent expertise. Their project is a carefully-done, substantial study with strong controls and high validity. However, their paper itself does not specify the context. By providing a context, our framework allows more precise generalization from these results.

Conclusion

The potential benefits of mixed-initiative optimization systems are suggested by informal reasoning from basic principles and has been demonstrated by initial research. Having established its basic feasibility, we can now turn to questions of how much, and under what circumstances, and through which mechanism we can benefit from a MIO system. We have argued that context is rich and diverse, and that the effectiveness of a MIO system is determined by the degree to which it is matched to the requirements of its context. Key MIO system design decisions should be evaluated in terms of the context in which the system will be used, or in terms of requirements that are shared across multiple contexts. Our framework highlights this role of context and provides a more detailed language for describing the relationship between context and system. We hope that these more precise descriptions can support the construction of a more consistent and solid structure of mixed-initiative optimization research.

Acknowledgments

The work in this paper was supported by grants from Precarn, Inc. and the National Sciences and Engineering Council of Canada (NSERC).

References

Fleming, M., and Cohen, R. 2001. A user modeling approach to determining system initiative in mixed-initiative AI systems. In *UM '01: Proceedings of the 8th International Conference on User Modeling 2001*, 54–63. Springer-Verlag.

Horvitz, E. 1999. Principles of mixed-initiative user interfaces. In *CHI'99: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 159–166. ACM.

Howe, A. E.; Whitley, L. D.; Barbulescu, L.; and Watson, J.-P. 2000. Mixed initiative scheduling for the air force satellite control network. In *Proceedings of Second NASA International Workshop on Planning and Scheduling for Space*.

Klau, G. W.; Lesh, N.; Marks, J.; and Mitzenmacher, M. 2002a. Human-guided tabu search. In *AI'02: Proceedings of Eighteenth National conference on Artificial intelligence*, 41–47. American Association for Artificial Intelligence.

Klau, G.; Lesh, N.; Marks, J.; Mitzenmacher, M.; and Schafer, G. 2002b. The HuGS platform: A toolkit for interactive optimization. In *AVI'02: Proceedings of Advanced Visual Interfaces (AVI)*.

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

Workshop on Planning and Scheduling for Space. Houston: The Institute for Advanced Interdisciplinary Research.

Rich, C.; Sidner, C. L.; and Lesh, N. 2001. Collagen: Applying collaborative discourse theory to human-computer interaction. *AI Magazine* 22(4):15–25.

Scott, S. D.; Lesh, N.; and Klau, G. W. 2002. Investigating human-computer optimization. In *CHI '02: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 155–162. ACM.

Mixed-Initiative Issues for a Personalized Time Management Assistant

Pauline M. Berry, Melinda T. Gervasio, Tomás E. Uribe,
Neil Yorke-Smith

Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, California 94025
{berry,gervasio,uribe,nysmith}@ai.sri.com

Abstract

This paper explores the mixed-initiative issues arising in the Personalized Time Manager (PTIME) system. PTIME is a persistent assistant that builds on our previous work on a personalized calendar agent (PCalM) (Berry et al. 2004). In order to persist and be useful, an intelligent agent that includes collaborative human/agent decision processes must learn and adapt to the user's changing needs. PTIME is intended to support a richer dialogue between the user and the system, which should be useful to both. If the system can reliably lean the user's preferences and practices, trust between user and assistant will be established, decreasing the system's reliance on mundane user interaction over time. The enabling technologies include soft constraint satisfaction, multicriteria optimization, a rich process framework, learning, and advice.

Introduction

The human time management problem is intensely personal. Many people—especially busy workers—are reluctant to relinquish control over the management of their own time. Moreover, people have different preferences and practices regarding how they schedule their time, how they negotiate appointments with others, and how much information they are willing to share when doing so. They also have different needs and priorities regarding the reminders they should receive.

We are developing the Personalized Time Manager (PTIME) assistant, with the goal of managing an individual's temporal commitments in a consistent, integrated framework over an extended period of time, while recognizing the differences between individuals and adapting to these differences. The interaction between the human user and the system is central to this goal. To maximize the continued usefulness of this interaction, both the user and the system should benefit from it. The scheduling solutions found by the system should be informative and proactive, and the dialogue should improve the quality of future interactions.

The PTIME project is part of a larger, ambitious automated assistant called CALO. CALO is a cognitive assistant that supports its human user in a variety of ways. For example, project and task management, information collection, organization and presentation and meeting understanding. However, the focus of CALO is its ability to learn and persist. Our hypothesis is that for mixed-initiative systems to succeed in the long term, the dialogue between human and system must evolve over time. To achieve this, we are designing PTIME so that

1. PTIME will unobtrusively learn user preferences, using a combination of passive learning, active learning, and advice-taking;
2. As a result, the user will become more confident of PTIME's ability over time, and will thus let it make more decisions autonomously; and
3. As autonomy increases, PTIME will learn when to involve the user in its decisions.

Background

Tools and standards for representing, displaying, and sharing schedule information have become common. A generally adopted standard for calendar representation is iCalendar (RFC2447).

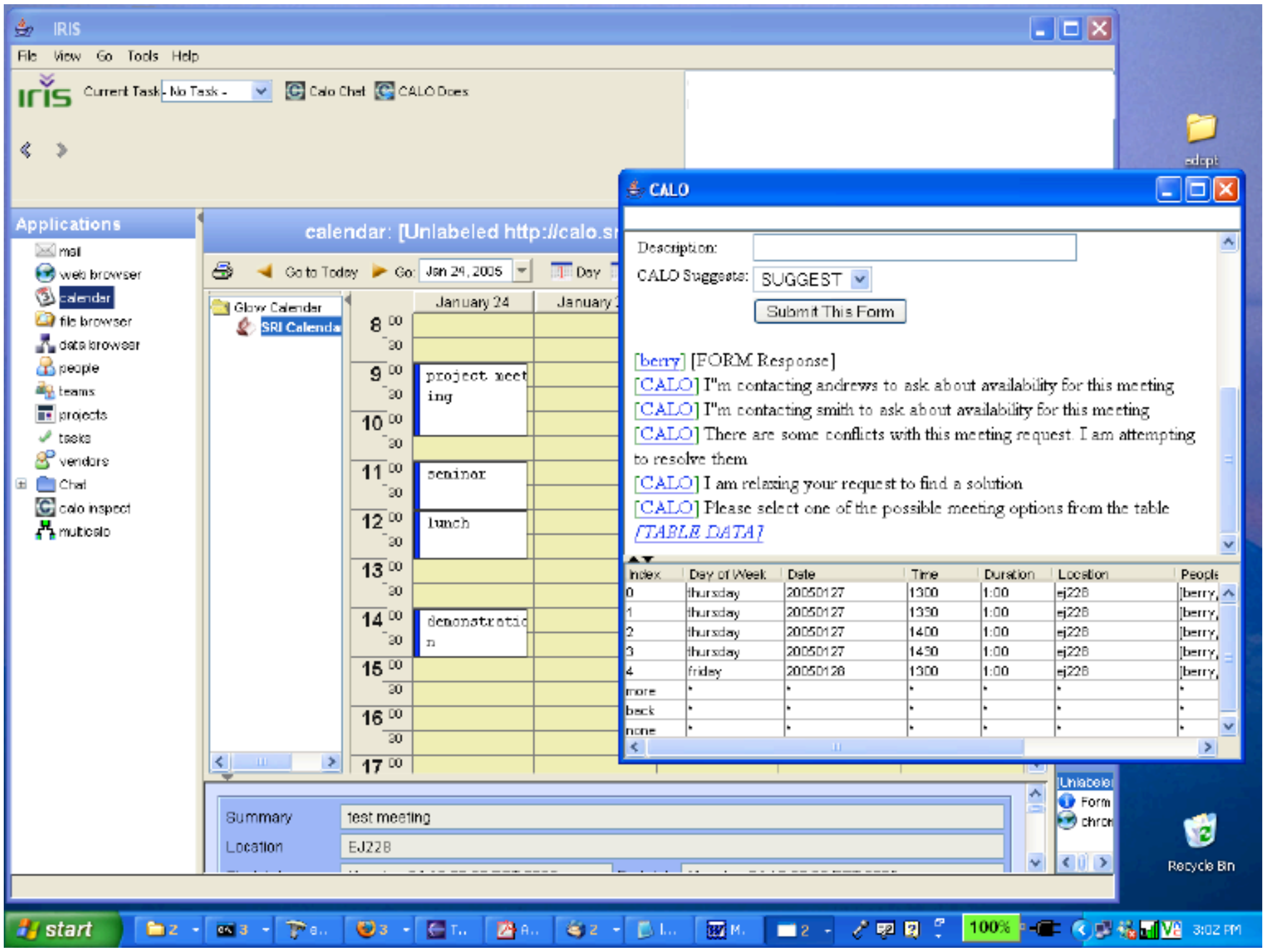
There are also many calendar tools to organize, display, and track commitments. However, most people still spend a considerable amount of time managing the constant changes and adjustments that must be made to their schedules. Desktop tools have dramatically improved the administration of our calendars, but their scheduling capabilities are limited. Automated meeting scheduling assistants have shown promise, but their use tends to be fleeting, since they do not evolve over time. People also use a variety of other tools, such as *to-do* lists, to keep track of workload and deadlines not supported in the typical calendar tools.

The emphasis in the research community has been on automated meeting scheduling: finding feasible time slots for meetings given a set of requirements on participants, times and locations. Work in this area can be generally divided into *Open* and *Closed* scheduling systems (Ephrati et al. 1994). In *Open* systems, individuals are autonomous, and responsible for creating and maintaining their own calendar and meeting schedules, perhaps selfishly. They can operate in an unbounded environment without constant obligation to one organization. In a *Closed* system, the meeting mechanisms are imposed on each individual, and a consistent and complete global calendar is maintained. Closed systems are more common because preference measures can be normalized across users, participant availability is known at all times, and the problem can be formulated as constraint optimization. Not all closed systems are centralized, and there is interesting work in distributed solutions to the closed scheduling problem (Ephrati et. al.1994, Sandip and Durfee 1998).

Closed systems are rarely adopted because the users seldom live in a truly closed environment, and need to retain more personal control of their calendars. Open scheduling systems pose additional challenges, such as privacy: an individual may not wish to share all, part, or any of his schedule, or may choose not to participate in a meeting, but not divulge this information.

CALO exists in an open, unbounded environment where issues of privacy, authority, cross-organizational scheduling, and availability of participants abound. PTIME is similar in approach to RCAL (Payne et. al. 2002) but extends the notion of collaboration with the user. The scheduling task is viewed as a shared goal of the user and the agent. The collaborative scheduling process is separated from the constraint reasoning algorithms to enable interaction with the user and other PTIME agents. This interaction forms the framework for learning and adjustable autonomy. PTIME considers finding the best solution as a dialogue between user and agent, and treats the underlying scheduling problem as a soft Constraint Satisfaction Problem (CSP). PTIME also addresses the problems of individual preference and scheduling events within the context of the user's workload and deadlines.

Figure 1 is a screenshot of the current PTIME interface, and illustrates the collaborative nature of the dialogue between PTIME and the user.



Description:

CALO Suggests: SUGGEST

Submit This Form

[berry] [FORM Response]
[CALO] I"m contacting andrews to ask about availability for this meeting
[CALO] I"m contacting smith to ask about availability for this meeting
[CALO] There are some conflicts with this meeting request. I am attempting to resolve them
[CALO] I am relaxing your request to find a solution
[CALO] Please select one of the possible meeting options from the table
[TABLE DATA]

Index	Day of Week	Date	Time	Duration	Location	People
0	thursday	20050127	1300	1:00	ej228	[berry,
1	thursday	20050127	1330	1:00	ej228	[berry,
2	thursday	20050127	1400	1:00	ej228	[berry,
3	thursday	20050127	1430	1:00	ej228	[berry,
4	friday	20050128	1300	1:00	ej228	[berry,
more	-	-	-	-	-	-
back	-	-	-	-	-	-
none	-	-	-	-	-	-

Summary: test meeting

Location: EJ228

Figure 1: A screenshot from PTIME.

Architecture

The PTIME architecture, illustrated in Figure 2, includes a number of components that make it personalized and adaptive. Key features of the architecture include:

- A **Process Framework (PTIME-Control),** which captures possible interactions with users and other agents, in the form of structured decision points.
- **Preference Learning (PLIANT)**, which lets the system evolve over time by learning process preferences, scheduling preferences, and, eventually, new processes from the user. Currently, we have developed PLIANT to learn temporal scheduling preference, e.g. *time of day, day or week, fragmentation of schedule*.
- **Advisability (PTIME-Control),** which enables direct instruction by the user at various levels of abstraction. Exploiting the explicit decision points in the process framework lets the user make choices and give advice. Choices may involve selecting an alternative scheduling process, e.g. *negotiate a new time for the meeting* vs. *relax an existing constraint to accept the current time*; or they may involve expressing simple temporal preferences, e.g. *don't schedule meetings just before lunch*.
- **Constraint Reasoning (PTIME-Engine)**, which permits reasoning within a unified plan representation. The representation used by PTIME unifies temporal and non-temporal constraints, soft and hard constraints, and preferences. The constraint reasoner (PTIME-Engine) considers workload issues and task deadlines when scheduling typical calendar events, such as meetings. The PTIME-Engine uses a hybrid solver that manages the application of temporal CSP algorithms, e.g., to handle Simple Temporal Problems (STPs) (Dechter et al. 1991) and Disjunctive Temporal Problems (DTPs) (Stergiou and Koubarakis 1998, Tsamardinos and Pollack 2003), to address complex constraint space and preference handling, and to enable partial constraint satisfaction. The PTIME-Engine can also explore alternative conflict resolution options via relaxation, negotiation, and explanation techniques, (Junker 2004).
- **Personalized Reminder Generation (PTIME-RG),** which reasons intelligently about if, when, and how to alert the user of upcoming events or possible conflicts amongst events. This work builds on the Autominder system (Pollack et al. 2003) and the learning algorithms to create reminders that are context-sensitive and personalized.
- **Adjustable Autonomy (PTIME-Control)**, which modulates control over decision points as the user's preferences and normal practices are learned, and trust between the user and the system is established. The goal is to decrease the system's reliance on user interaction over time.

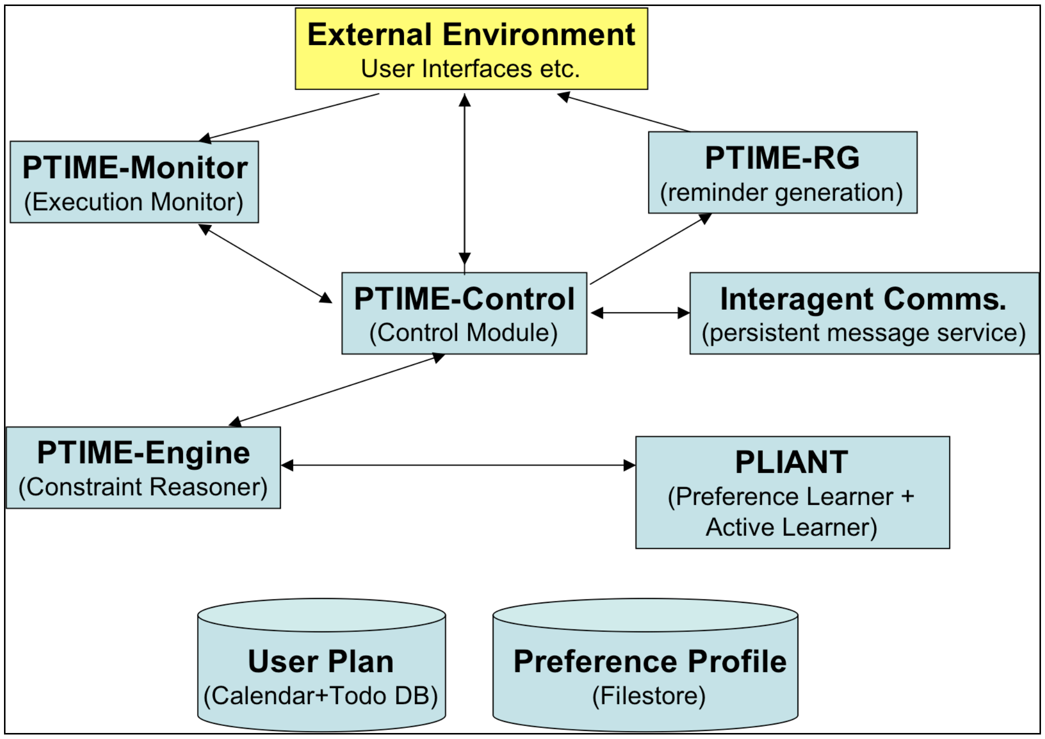


Figure 2: PTIME functional architecture

Persistence and Learning

Central to persistence are the application of learning technology and a framework for advisability. Through continual active learning and advice taking, PTIME constructs a dynamic preference profile containing two types of guidance:

(1) Scheduling: Preferences over *schedules* (when to reserve time and with whom), *relaxations* (which constraints, or constraint sets, are more readily relaxed) and *reminders* (when, how and about which events the user should be alerted).

(2) Process selection and application: preferences over existing process descriptions (e.g., negotiate or relax) and learned processes.

Both types of information can be actively asserted using a policy specification language, building on work on advisability and adjustable autonomy (Myers and Morley 2003). They can also be learned passively by monitoring the user's decisions.

PTIME uses a suite of tools to learn various kinds of preferences. A Support Vector Machine (SVM) module, supplemented with active learning strategies, learns user preferences about schedules in the form of an evaluation function over schedule features (e.g., day of week, start time, fragmentation) (Gervasio et al. 2005). The features were selected to capture the temporal characteristics of a scheduling decision. We are adding features that capture whether or not constraints are satisfied by a candidate schedule; this will let PTIME learn preferences over relaxations in the case of over-constrained schedules as well. We are also exploring the problem of procedural learning, where the performance task is to determine what to do under a particular situation rather than to evaluate the goodness of a candidate schedule. Along similar lines, we are using procedural learning to handle situations that arise after an event is scheduled: for example, if the host cannot make it or if the scheduled venue suddenly becomes unavailable. Finally, PTIME uses reinforcement learning schemes to learn both reminder strategies that are tailored to individual users and strategies for determining the amount of autonomy to take in different situations. By observing the effects of different reminder strategies on a user, PTIME can adjust its reminder strategy to account for personal traits as well as different schedule situations. A similar process occurs with the learning of adjustable autonomy decisions.

In all cases, PTIME learns online (or from the execution traces of the user's actual interactions with PTIME), so it can continually adjust to changing user preferences and situations. Concept shift—the phenomenon of users exhibiting drastic changes in preferences—is a known issue in the calendar domain. We plan to address this problem more directly by designing a learning approach that is sensitive to sharp changes as well as a period of stabilization of user preferences over time.

Mixed-Initiative Research Directions

PTIME has demonstrated its initial calendar management software within the CALO project, and is currently undergoing a test phase, conducted by an external agency, to assess its capability to learn user preferences and therefore retain a high level of usefulness to the user. PTIME development has four principal research goals for 2005, and all relate to its ability to adapt to the user's needs. This section describes our research into hybrid constraint satisfaction, partial constraint satisfaction with preferences, negotiation and advice-taking. The result is a framework for negotiation between agents and with the user. We will also describe our ongoing work to learn preferences and accept advice from the user.

Using Preferences in Scheduling

The constraint problem in PTIME is a combination of three factors: the user's existing schedule, the meeting request, and the interactive collaboration between PTIME and the user. The user may interact with PTIME to explore possible relaxed solutions to the problem, leading to a sequence of related soft Constraint Optimization Problems (COPs) to solve. For example, the user may initially specify a strong preference against meetings on Monday mornings. Later, she may weaken this preference but increase the importance of the specified meeting room.

Critical to the mixed-initiative goals of PTIME is the ability to use the learned knowledge of user preferences within the underlying constraint satisfaction problem. (Berry et al. 2005) describes our approach to constraint satisfaction for PTIME, which involves a combination of disjunctive and finite-domain constraint solvers with preferences. Since it is relevant to this discussion, we now briefly discuss the representation of schedule preferences and relaxations within soft CSPs.

User preferences are mapped into the shape and height of specific preference functions for each of the relevant soft constraints. The shape models how much and in what way the constraint may be relaxed, and the height models the importance of the constraint. This builds on the work by (Peintner and Pollack 2004) and (Bistarelli, Montanari, and Rossi 2001).

For example, suppose a meeting with Bob must occur before a seminar. If $S_M\,(S_S)$ and $E_M\,(E_S)$ are the start and end of the meeting (resp. seminar), the constraint is $c : E_M - S_S \le 0$. Figure 3 shows the shape of the preference functions on c from left to right, if the constraint is hard, a little relaxable, and very relaxable.

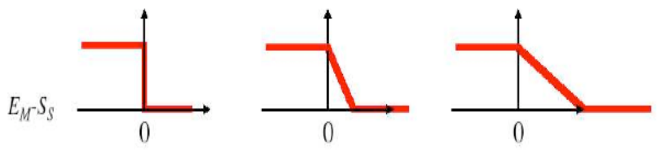


Figure 3: Example Preference Functions

To solve soft CSPs that include such preference functions, we combine existing solvers for the temporal and non-temporal constraint subproblems in a hybrid formulation. Constraint solving in PTIME is implemented in ECLiPSe (Cheadle et al. 2003), and is also compatible with SICStus Prolog; such Constraint Logic Programming systems are well-suited to hybrid solving. We are exploring search techniques that can produce not only a single optimal solution, but also a good set of qualitatively different solutions to present to the user. A good set of solutions has three characteristics: to include the most desirable solution, to give the user qualitatively different choices and to promote future learning.

Negotiation: Process Design for Conflict Resolution

The work on extending the constraint representation and relaxation framework of our CSP is to enable more informative dialogue between the human user and the agent. The motivation behind PTIME is to facilitate a collaborative assistant for time management. Taking note of research in collaboration (Grosz and Kraus 1999) and collaborative interfaces (Babaian, Grosz, and Shieber 2002), we view conflict resolution as a joint task to be undertaken between the human and his agent, or between agents. Currently, the interaction is explicitly captured in the highly reactive process descriptions offered by SPARK-L (Morley 2004) and applied within a framework of advice. We would like to abstract and possibly learn the applicability conditions of the processes within the context of the dialogue.

Figure 4 presents a typical dialogue that might take place between a user and PTIME. To enable this type of dialogue, the processes capture the key decision points. Future research will construct a collaborative framework within which these processes will operate.

Figure 5 illustrates an example process in SPARK-L. Each decision point offers the choice to automate the decision, ask the user for advice or decision, postpone the decision, or take another action. For example, when the goal is:

[do: (select_solution $resultset $result)],

a set of different actions might be intended, including asking the user to select an option or automatically selecting the highest valued one. The choice of action depends on the user's preference (learned or told), the physical context (such as the user's current activity), and the cognitive context. Learning how and when to apply each activity is a highly personalized and evolving problem.

User Helen: "Please schedule a group meeting early next week"

PTIME Agent: *"Your specific request conflicts with your current workload and meeting constraints"*

PTIME Agent: *"May I suggest some possible alternatives"*

1. *Meet Monday at 10am without "Bob"*
2. *Meet Tuesday at 4pm overlapping the seminar*
3. *Meet Monday at 10am warning your report deadline may be in jeopardy*
4. *Meet Tuesday at 11 and reschedule your meeting with the boss*

User Helen: I don't mind overlapping some meetings – show me more possibilities like 2.

PTIME Agent: *"Ok How about"*

1. *Meet Monday at 11:30 running into lunch by 15 minutes*
2. *Meet Tuesday at 9:30 but Bob may have to leave early*

User Helen: "Ok go ahead with 2"

Figure 4. Example user-agent dialogue

```
{defprocedure "schedule"
 cue: [do: (schedule $event_type $constraints $attributes)]
 preconditions (Event_Type "meeting")
 body: [context (and (User $self)
                (Participants $constraints $pset))
       seq:
         [do: (retrieve_availability $pset $constraints)]
         [do: (solve_schedule $constraints $resultset)]
         [do: (select_solution $resultset $result)]
         [select: (= $result [])
              [do: (resolve_conflct $constraints $result)]
          [do: (confirm_meeting $result $attributes)]]
   }
```

Figure 5. Example SPARK-L process

Advice

The PTIME-Controller can take user advice and conform to organizational policies. Advice is defined as an enforceable, well-specified constraint on the performance or application of an action in a given situation. In general advice can be considered to be a type of policy, often personalized. (Sloman 1994) defines two types of policy: authorization and obligation. For our advisable system, we extend this categorization to include preference:

1. *Authorization* defines the actions that the agent is either permitted or forbidden to perform on a target.
2. *Obligation* defines the actions that an agent must perform on a set of targets when an event occurs. Obligation actions are always triggered by events, since the agent must know when to perform the specified actions.
3. *Preference* defines a ranking in the order or selection of an action under certain conditions.

Advice can both apply to the application of strategies, the conditions under which a strategy is applicable, or the instantation of a variable. Advice may be conflicting, can be long-lived, and their relevance may decay over time. Advice can be used to influence the selection of procedures and strategies for problem solving and also to influence adjustable autonomy. The management of advice is an active research focus for the CALO project. The application of advice is central to both PTIME for influencing preference and for controlling adjustable autonomy strategies.

Summary

The concept of a persistent useful interaction motivates the mixed-initiative design of PTIME. It has an extended notion of collaboration with the user, which forms the framework for learning and adjustable autonomy. The time management process is represented using context-sensitive, hierarchical procedures, which provide hooks, via the structured decision points, into the user's decision process at multiple levels of abstraction. These hooks can be used to passively learn the user's preferences or to facilitate the specification of advice from the user. The resulting agent will let the user retain control of decisions when necessary, and relinquish control to the assistant at other times. Meanwhile, the agent will be sensitive to the user's wishes and preferences.

Acknowledgments. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the DARPA or the Department of Interior-National Business Center (DOI-NBC).

References

Berry, P.M., Gervasio, M., Uribe, T., Myers, K., and Nitz, K. (2004). A personalized calendar assistant, *In proceedings of the AAAI Spring Symposium Series*, Stanford University.

Berry, P.M. Gervasio, M., Uribe, T., and Yorke-Smith, N. (2005). *Multi-Criteria Constraint Solving and Relaxation for Personalized Systems*, Technical Report, SRI International.

Babaian, T., Grosz, B. and Shieber, S.M. (2002). A writer's collaborative aid. *In proceedings of the Intelligent User Interfaces Conference*, San Francisco, CA. January 13-16. ACM Press, pp. 7-14.

Bistarelli, S., Montanari, U., and Rossi. F. (2001). Solving and learning soft temporal constraints: Experimental scenario and examples, *In proceedings of the CP'01 Workshop on Modelling and Solving Problems with Soft Constraints.*

Cheadle, A.M., Harvey, W., Sadler, A.J., Schimpf, J., Shen, K. and Wallace, M.G. (2003). *ECLiPSe: An Introduction*, Technical Report IC-Parc-03-1, IC--Parc, Imperial College London.

Dechter, R., Meiri, I., and Pearl. J. (1991). Temporal constraint networks. *Artificial Intelligence*, 49(1– 3):61–95.

Ephrati, E., Zlotkin, G., and Rosenschein, J.S. (1994). A non manipulable meeting scheduling system, *In proceedings of the Thirteenth International Distributed Artificial Intelligence Workshop*, Seattle.

Gervasio, M.T., Moffitt, M.D., Pollack, M.E., Taylor, J. and Uribe, T.E. (2005). *In proceedings of the International Conference in Intelligent User Interfaces (IUI),* San Diego.

Grosz, B. and Kraus, S. (1999). The evolution of SharedPlans. In *Foundations and Theories of Rational Agencies*, A. Rao and M. Wooldridge, eds. pp. 227-262.

Junker, E. (2004). QuickXplain: Preferred Explanations and Relaxations for Over-Constrained Problems. *In proceedings of AAAI-04.*

Morley, D. (2004). *Introduction to SPARK.* Technical Report, Artificial Intelligence Center, SRI International, Menlo Park, CA.

Myers, K. L. and Morley, D. N. (2003). Policy-based Agent Directability. In *Agent Autonomy*, Kluwer Academic Publishers.

Payne, T. R., Singh, R., and Sycara, K. (2002). Rcal: A case study on semantic web agents, *In proceedings of the First International Conference on Autonomous Agents and Multi-agent Systems.*

Peintner B. and Pollack, M.E. (2004). Low-cost addition of preferences to DTPs and TCSPs, *In proceedings of AAAI-04*, pages 723-728.

Pollack, M.E., Brown, L., Colbry, D., McCarthy, C.E., Orosz, C., Peintner, B., Ramakrishnan, S., and Tsamardinos, I. (2003). Autominder: An intelligent cognitive orthotic system for people with memory impairment, *Robotics and Autonomous Systems*, 44:273-282, 2003.

Sandip, S., and Durfee, E.H. (1998). A formal study of distributed meeting scheduling. *Group Decision and Negotiation*, vol. 7, pp. 265-298.

Sloman, M. (1994). Policy driven management for distributed systems. *Plenum Press Journal of Network and Systems Management,* vol.2, no. 4, pp. 333-360.

Stergiou, K., and Koubarakis, M. (1998). Backtracking algorithms for disjunctions of temporal constraints, *In proceedings of AAAI/ IAAI-98*, p.248-253, Madison.

Tsamardinos, I., and Pollack, M.E. (2003). Efficient Solution Techniques for Disjunctive Temporal Reasoning Problems, *Artificial Intelligence*, 151(1-2):43-90.

Weasel: A Mixed-Initiative System to Assist in Military Planning

C. C. Hayes, Capt. A. D. Larson, U. Ravinder

University of Minnesota,
111 Church Street. S. E
Minneapolis, MN 55455
hayes@me.umn.edu

United States Air Force Academy,
2354 Fairchild Drive, Suite 6L101B
USAF Academy CO 80840
adam.larson@usafa.af.mil

NASA Ames Research Center
Mail Stop 262-4
Moffet Field, CA 94035
uravinder@mail.arc.nasa.gov

Abstract

This paper describes a mixed initiative planning system, called Weasel and its evaluation. Weasel was developed to assist military decision makers in the task of enemy course of action generation. The evaluation assesses Weasel's impact on the decision making performance of two potential user groups. When designing Weasel, we aimed to maximize benefits delivered by the software by focusing support functions on key areas in which expert analysts exhibited difficulties. We also aimed to minimize development, training and maintenance costs by designing displays to reflect expert analysts' representations and relying on human problem solving skills where possible. The goals of the evaluation are to 1) assess whether Weasel increases users' problem solving performance, where performance is measured in terms of overall solution quality, 2) identify the most appropriate user group by assessing whether domain intermediates are helped or hindered more than domain experts, and 3) identify possible negative consequences that may occur when Weasel generates a "brittle" solution. The issues explored in Weasel's development and evaluation are common to many mixed initiative systems.

Introduction

This work describes the development and evaluation of Weasel, a mixed-initiative system which assists military planners in exploring possible enemy courses of action (ECOAs). An *enemy course of action* is an arrangement of enemy forces which very abstractly define a very abstract "plan" which may be followed by enemy forces. There is great interest in possible use of decision support tools to assist in military planning; the increased complexity and tempo of modern military operations combined with increased pressure to reduce staff sizes makes it difficult for people to keep up with the demands of operations planning. Mixed initiative systems tend to be more appealing than automated systems in complex, safety critical domains such as this one because of the opportunity to benefit from human judgment. However, before employing mixed initiative systems in decisions with life and death consequences, it is important to first understand both the positive and negative impacts they may have on human problem solving performance.

The design goals behind Weasel were to improve decision making performance in this task for military planners with an intermediate level of experience (i.e. 2 to 5 years training), and possibly for experts (6+ years of training and practice) as well. The evaluation assessed whether use of Weasel changed (improved or decreased) planning performance for two user groups: intermediates and experts, and explored whether there were situations in which use of Weasel might decrement performance.

Mixed-initiative planning and scheduling systems (MIPAS) are computer tools which work jointly with humans to create plans or schedules. MIPAS can be viewed as examples of a larger class of tools called decision support systems (DSSs).

Decision support systems are computer tools which assist human decision makers to make better decisions in any type of task (e.g. medical diagnosis, manufacturing plant layout, product design, etc.) without necessarily making those decisions for them. DSSs may provide support in many ways, for example by providing task-specialized editors, generating whole or partial solutions, or providing solution evaluation and comparison tools. A key philosophical assumption behind DSSs, which is not necessarily shared by all MIPAS, is that the human is the one that should be in control of the decision making process. Weasel is both a DSS and a MIPAS.

High criticality decision making tasks are those in which decisions can result in large costs or catastrophic consequences. Examples of high criticality domains include military planning, search and rescue, and medical diagnosis. They are of interest because they represent areas where problem solving improvements, gained through introduction of DSSs or other means, can yield great value. However, because decisions made in these domains may impact human safety or have political ramifications, it is important to clearly understand how DSS tools impact human decision making before adopting such tools. The possibility of *over-reliance*, i.e. inappropriate trust (Parasuraman, 1997) on a DSS is a major concern in safety critical domains.

In general, DSSs can have *both* positive and negative impacts on human decision making, possibly improving performance in many situations while degrading it in others. In particular, Smith, McCoy and Layton (1997) describe an experiment exploring a situation in which a DSS, The Flight Planning Testbed (FPT) improved users' average problem solving performance in finding fuel-optimal routes for commercial jets, but also occasionally degraded some users' performance when FPT exhibited *brittle* behavior. Brittle behavior occurs when parameter not modeled by the system impact the solution. Brittle behavior results in generation of inappropriate or inadequate suggestions. Unfortunately, in complex, context dependant domains, it can be difficult to predict when brittle behavior may occur. In FTP's case, brittle solutions were fuel-optimal but unnecessarily risky by human-decision makers' standards. However, the researchers also found that this effect appeared to be mitigated if subjects did their own exploration of the problem before seeing the computer's solution(s). They further hypothesized (but did not test) that additional strategies might also mitigate the impact of system brittleness, such as simultaneous presentation of multiple computer generated solution options, and computer critiquing of human generated options,

All DSSs, simulations, or mathematical models will sometimes exhibit brittleness because they are *necessarily* simplifications of the real world's richness. Therefore their solutions will produce some degree of error which may or may not be predictable. Given that some degree of brittleness is inevitable, it is important to consider how brittle behavior may impact decision makers in many tasks, and if it can be mitigated. One of the goals of this work is to assess whether Layton's findings were generally true for other domains; could one expect the similar results in the domain of ECOA planning? Would brittle solutions generated by Weasel also produce a similar performance decrement? If so, could the effect be mitigated by a similar strategy?

The Task Domain: ECOA Generation

Weasel is part of a trio of tools: CoRaven (Jones at al. 1999), Weasel and Fox (Schlabach, Hayes and Goldberg, 1997), which support a range of problem military planning and intelligence activities, shown as ovals in Figure 1. All steps may be conducted in parallel, and all are repeated many times during the course of a battle. The decision makers who engage in this problem solving cycle include both military operation planners and intelligence analysts. The overall goal of this reasoning cycle is to identify what action(s) the friendly forces should take next, based on continual assessments and re-assessments of the current battlefield and enemy situation. Although the direct output of Weasel is a set of possible (and likely) enemy courses of action, it supports the assessment of friendly courses of action by providing a set of foils. Friendly courses of action are assessed based on their performance against multiple enemy courses of action which might occur.

There is no specific starting or ending point to the cycle in Figure 1. However, before the onset of a battle, intelligence analysts often start at oval 1 (Figure 1) "Plan/Schedule Intelligence Collection." They create a plan for gathering key pieces of information pertaining to the enemy such as the type of unit, likely resources, and location of key elements. This information is gathered by scouts, satellites and surveillance devices (step 2), then it is analyzed (step 3) to produce hypotheses and constraints on enemy resources and location.

Weasel assists analysts in step 4 with the systematic generation of enemy courses of action that are consistent with the intelligence conclusions developed in step 3, and the observed rules of behavior for that enemy. ECOAs, developed jointly by Weasel and the analyst, are passed on to the Fox system, which uses a genetic algorithm and war gaming simulator to generate friendly courses of action (FCOAs) that perform well against those ECOAs. This cycle is continually repeated throughout the battle as the situation changes. Plans for friendly actions must continually be reassessed as the battle unfolds.

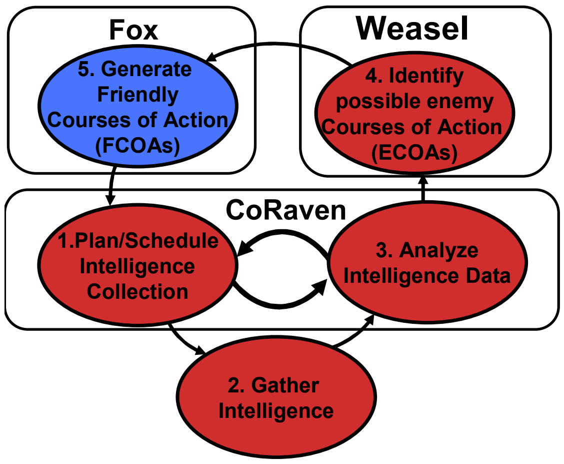

Fox
Weasel
5. Generate Friendly Courses of Action (FCOAs)
4. Identify possible enemy Courses of Action (ECOAs)
CoRaven
1.Plan/Schedule Intelligence Collection
3. Analyze Intelligence Data
2. Gather Intelligence

Figure 1: The intelligence collection and planning cycle.

Weasel

Considerations in the Design of Weasel

Intelligence analysts must consider many ECOAs since there are many actions which an enemy might take. Unfortunately, there are an infinite number of possible ECOAs, so even with extensive computational resources one cannot consider them all. Fortunately, most ECOAs are neither useful nor interesting. Analysts typically focus their search on a few *most likely* and a few *most dangerous* (but possibly unlikely) ECOAs.

Weasel is designed to assist military analysts in thoroughly and systematically considering the *most likely* ECOAs. In this context, the *most likely ECOAs* refers to ECOAs that are consistent with current data and assumptions about most likely enemy position and resources. Terrain constraints and intelligence assumptions provide strong constraints on the search. The likely ECOAs are usually a very small subset of the total possible.

Note that Weasel does not currently assist users with identifying *most dangerous* ECOAs. Analysts must also identify ECOAs that are not necessarily consistent with intelligence assumptions, but which could pose a considerable threat if they were to occur. This is an important task in which analysts would probably welcome assistance. However, outside of brute force exhaustive search and evaluation, we do not currently have an efficient algorithm which could feasibly address this task. Future work may explore approaches to focus dangerous but unlikely ECOAs.

Weasel was developed through cognitive engineering (Smith and Geddes, 2003) and human-centered system development methods. Many design decisions were guided by the objectives to minimize develop, training and maintenance costs, while maximizing benefits to users. In designing a system to meet these objectives, we were very conscious of the fact that ECOA generation is a task at which domain experts already do relatively well, and it is usually performed under time pressure. The majority of computer tools require some overhead to learn and to use, thus, what ever tool we developed had better offer clear benefits to users in areas where they desire assistance. Otherwise there would be little chance they would be willing to take the time required to learn and use them.

With this understanding, we observed analysts performing the ECOA generation in laboratory studies, during which we took "protocol" transcripts (Ericsson and Simon, 1984), and in training exercises such as the Prairie Warrior exercises held in Ft. Leavenworth, KA. Through these observations, we observed that even experts sometimes over looked relevant ECOAs for a variety of reasons. For example, it was often difficult for them to systematically think through all the possible options while simultaneously keeping track of all the current relevant constraints. In other cases, they because fixated on particular assumptions (which were often implicit in their reasoning), forgetting to question them when the context changed.

Based on these findings, we designed Weasel to assist analysts by providing 1) an engine that can systematically enumerate possible ECOAs consistent with a given set of assumptions, and 2) an interface in which they can express and manipulate those constraints. Together these capabilities allow "what-if" scenarios to be described and assessed rapidly. Lastly, we provided an interface displaying hard constraints used by Weasel in order to provide users with insight into (and possibly trust) in the reasoning engine. Making such these constraints observable and explicit may provide an added training benefit to domain novices and intermediates.

Other principles guiding design of Weasel were "computer in the loop" and "minimalist intervention" philosophies. Usually, developers of mixed-initiative technologies view the challenge as bring the "human in the loop." However, for most complex cognitive tasks such as planning, design and medical diagnosis the human *is* in the loop already. Not only are they *in* the loop, humans *are* the loop -- and have been for thousands of years. Thus we feel the challenge should be to bring the "computer in the loop" in a way that is acceptable to humans.

Several design implications follow from a "computer in the loop" philosophy. One is: when in doubt, leave a task and to the human; focus on minimal introduction of computer assistance. A simple computer tool which has been well executed interfaces is more likely to be useful than a more complex one. It will probably also be easier to maintain. We explicitly decided *not* to intervene (at least initially) in tasks such as identifying relevant constraints, or selecting ECOAs for further consideration since these tasks require complex, experience-based judgments which may best be left to humans.

Lastly, Weasel's representations and displays had to fit with analysts' way of thinking about the task. Many of Weasel's representations and displays are based directly on sketches made by analysts on paper or acetate map overlays while doing their work.

Considerations in Weasel's Evaluation

An important part of a human-centered design approach is to evaluate the system early and often. Frequent evaluations provide valuable feedback to system developers as to whether the approach is meeting the design goals so adjustments can be made. There are many properties of mixed-initiative systems that are important to evaluate including ease of use, accuracy of software generated results and overall impact on joint human/computer problem solving performance. The latter is the "bottom line" in many mixed-initiative systems. If users do not derive tangible benefits from the system they won't use it; the computer will be left "out of the loop."

The evaluation aims to address several questions pertaining to users' problem solving performance when using Weasel:

1. Does Weasel actually increase users' average problem solving performance? In this evaluation, performance is measured in terms of overall solution quality,
2. If Weasel does result in a performance change, does it impact performance of domain intermediates more (or less) than domain experts?

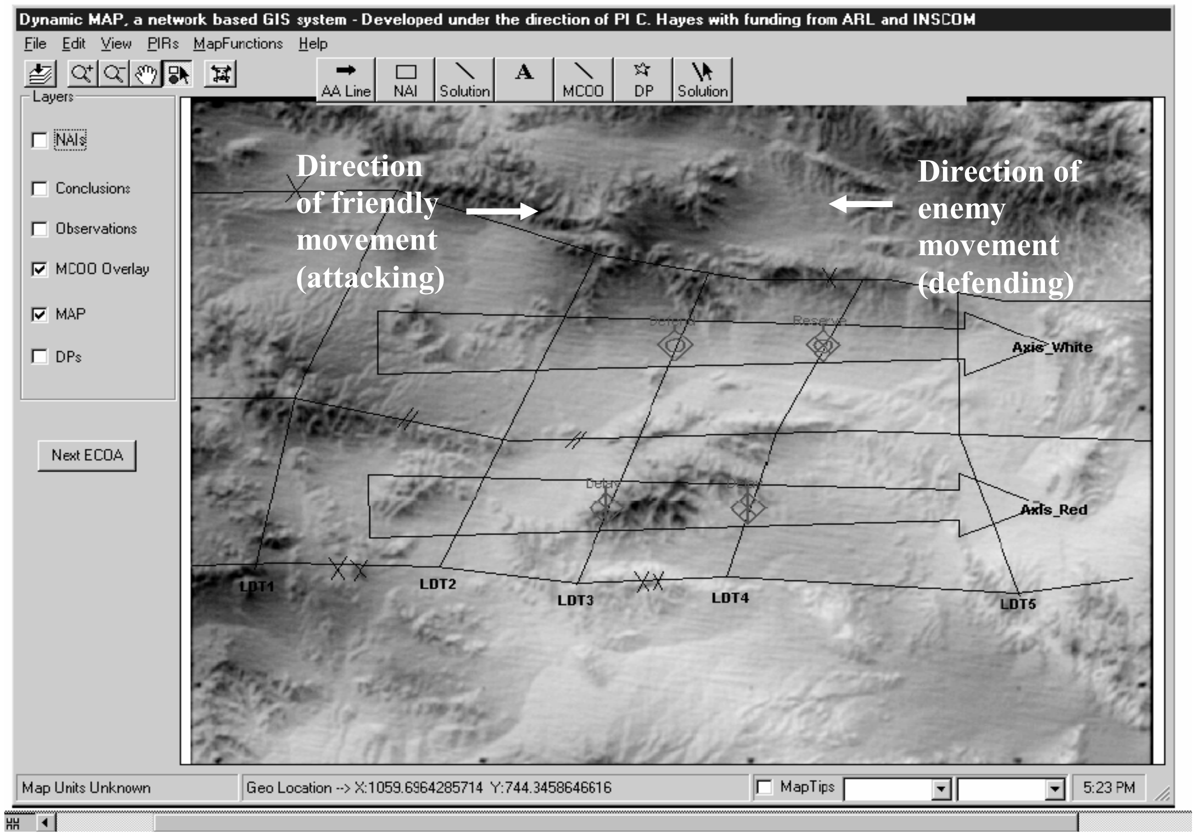

Dynamic MAP, a network based GIS system - Developed under the direction of PI C. Hayes with funding from ARL and INSCOM
File Edit View PIRs MapFunctions Help
AA Line
NAI
Solution
MCOO
DP
Solution
Layers
NAIs
Conclusions
Observations
MCOO Overlay
MAP
DPs
Next ECOA
Direction of friendly movement (attacking)
Direction of enemy movement (defending)
Axis_White
Axis_Red
LDT1
LDT2
LDT3
LDT4
LDT5
Map Units Unknown
Geo Location --> X:1059.6964285714 Y:744.3458646616
MapTips
5:23 PM

Figure 2: An enemy course of action (ECOA) in the context of the terrain.

3. Are there negative consequences that may occur when Weasel generates a "brittle" solution?
4. Can negative impacts of brittle solutions be reduced by presenting computer solutions *after* the human has generated some of their own solutions?

From a development standpoint, these questions will help us to assess whether our basic approach is reasonable, what users groups should be considered as "customers," ways in which the system may sometimes hurt performance, and possible strategies to avoid pitfalls.

ECOA Representations

An example of an ECOA is shown in Figures 2 and 3. Figure 2 shows an ECOA on the terrain; friendly forces are attacking the enemy (but only enemy forces are shown. In this context, an ECOA is an assignment of battlefield locations and fighting roles to enemy units. Battlefield locations are specified in terms of intersections on a grid formed by markers on the map called avenues of approach and lines of defensible terrain. *Avenues of approach* (AAs) are shown in Figure 2 as large horizontal arrows; they represent corridors between mountains and other obstacles through which troops can move. The direction of the AA arrows indicates the direction of attack (and the friendly movement). *Lines of defensible terrain* (LDTs) are shown in Figure 2 as thin vertical lines which are placed across narrow parts of the AAs; the represent areas where defenses tend to be setup and where fire fights occur. The diamonds placed at the intersections of AAs and LDTs represent specific enemy units.

Figure 3 shows an ECOA sketch, which is an abstracted version of the ECOA in Figure 2. All the details of the terrain have been abstracted except for the AAs and LDTs. The labels, "Def" "Del" and "R" represent the roles of the various units: defense, delay and reserve, respectively.

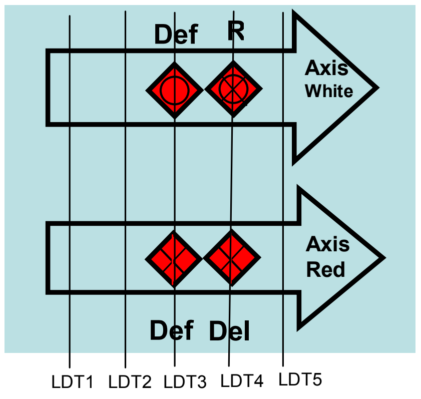


Figure 3: An enemy course of action (ECOA) sketch.

ECOAs can be thought of as the first step in a plan for future enemy actions. Alternatively, one can think of each ECOA as a specific layout of the enemy's chess pieces (i.e. units). However, the board is only partially observable, so many possible board layouts must be considered based on the little you can directly observe or indirectly guess.

System Description

There are several steps by which Weasel generates the most likely enemy COAs, as shown in Figure 4. Each of these steps will be described below.

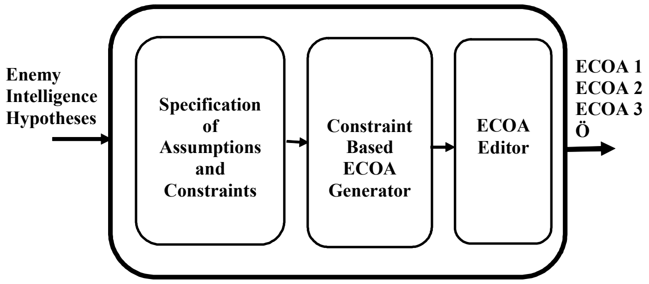

Enemy Intelligence Hypotheses
Specification of Assumptions and Constraints
Constraint Based ECOA Generator
ECOA Editor
ECOA 1
ECOA 2
ECOA 3
Ö

Figure 4: A System diagram of Weasel's components.

Weasel uses several types of intelligence hypotheses as inputs to constrain enemy position. These hypotheses may be generated by the analyst from intelligence reports, or in this case, by a decision support tool called CoRaven. CoRaven uses a belief network to compute the probabilities of various hypotheses from intelligence reports (in the form of SALUTE messages), and then it visually displays its conclusions. Figure 5 shows the display of one type of intelligence hypothesis: depth of the enemy defense. The depth of defense indicates how far west the enemy has penetrated from their starting point, which in this example is near LDT 5. Thus, there are 4 hypotheses under current consideration which are: the enemy has penetrated as far as LDT1, LDT2, LDT3 or LDT4. The color of each LDT indicates the probability of each hypothesis, where black is less than a 5 % probability. As the probability increases, the LDT becomes brighter (whiter).

CoRaven computes these probabilities based on current intelligence reports. Each report is shown as a small symbol on the map in Figure 5. The black symbols indicate places where intelligence observations have been made, and nothing of interest was seen, while the gray (or red in the color version) symbols indicate observations of enemy activity. As new observations are reported, the intensities (i.e. probabilities) of the LDTs shift. In this

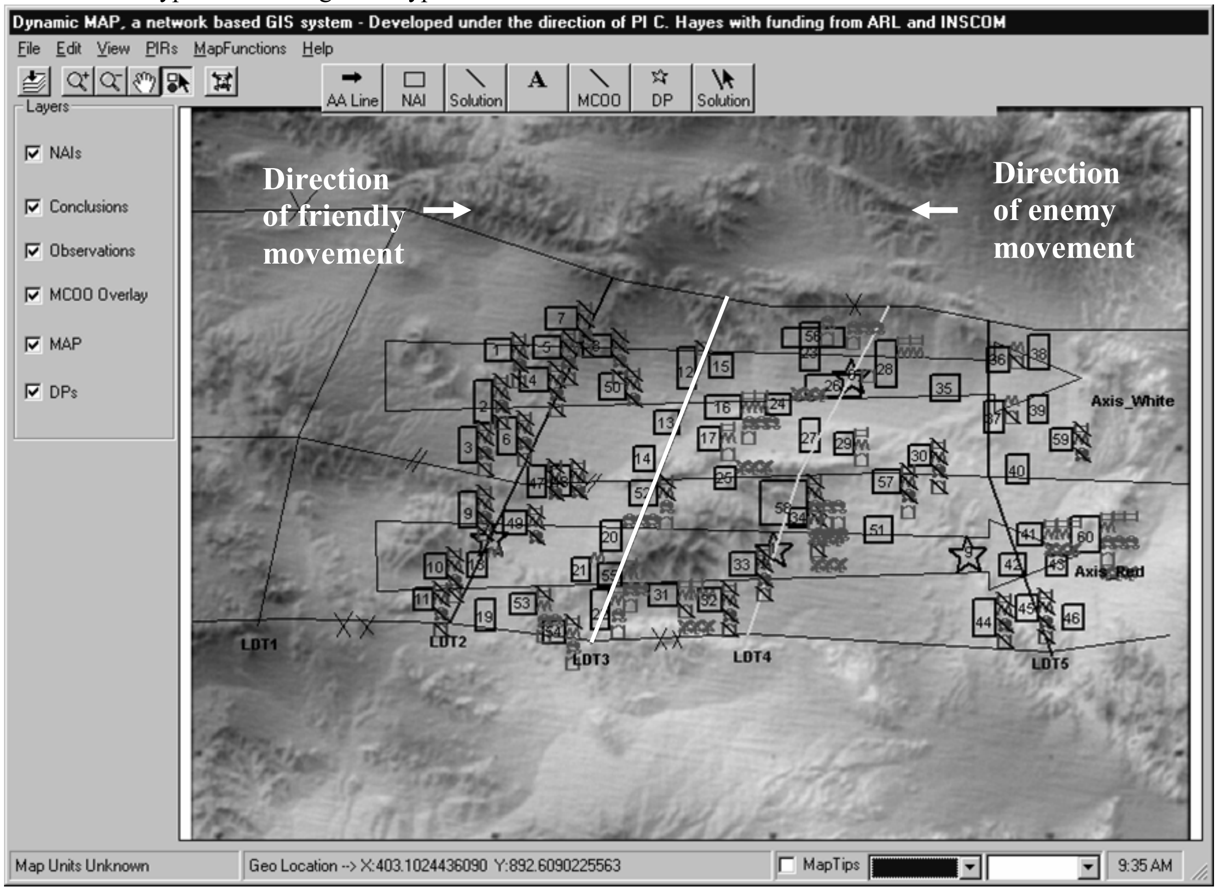
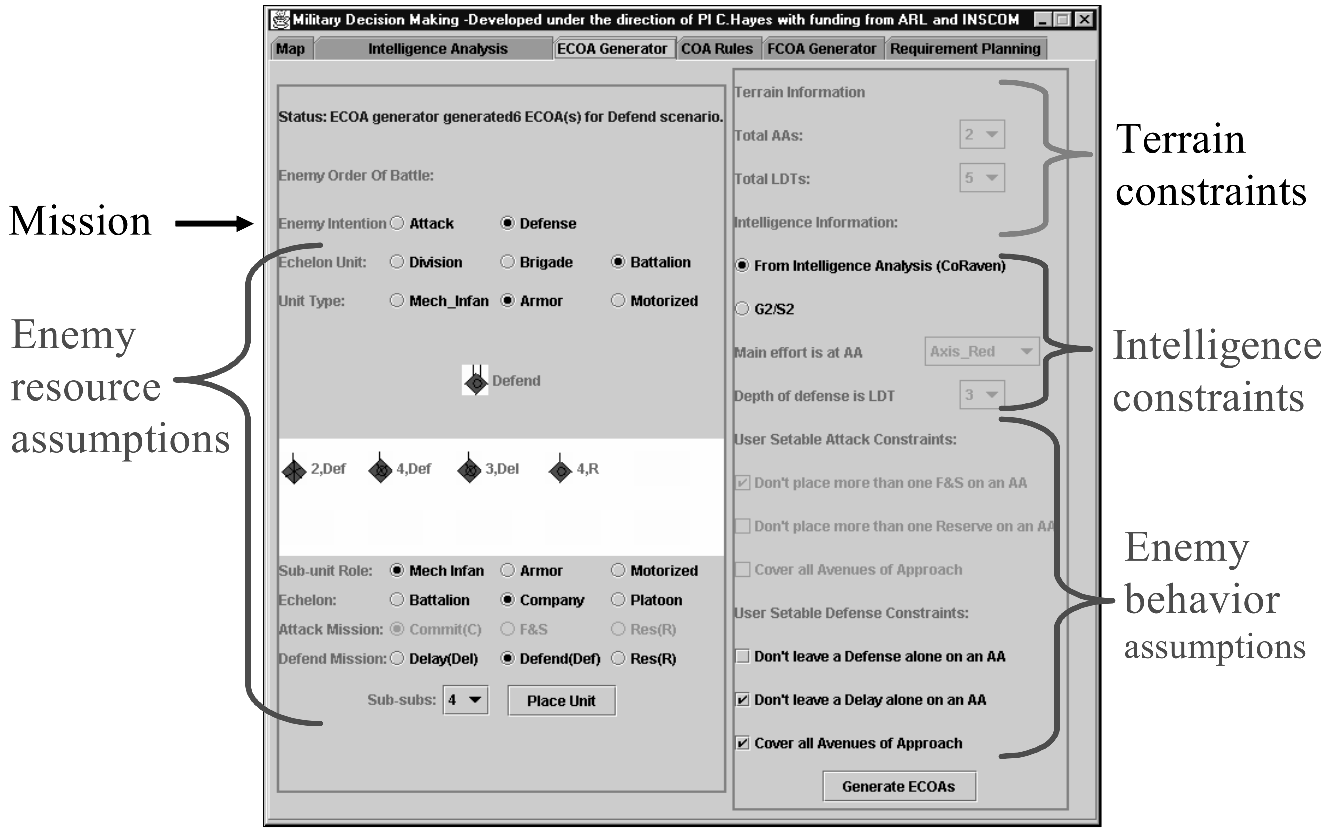

Dynamic MAP, a network based GIS system - Developed under the direction of PI C. Hayes with funding from ARL and INSCOM
File
Edit
View
PIRs
MapFunctions
Help
AA Line
NAI
Solution
A
MCOO
DP
Solution
Layers
NAIs
Conclusions
Observations
MCOO Overlay
MAP
DPs
Direction of friendly movement
Direction of enemy movement
Axis_White
Axis_Red
LDT1
LDT2
LDT3
LDT4
LDT5
Map Units Unknown
Geo Location --> X:403.1024436090 Y:892.6090225563
MapTips
9:35 AM

Figure 5: Partial results from Co-Raven's intelligence analysis.



Status: ECOA generator generated6 ECOA(s) for Defend scenario.

Enemy Order Of Battle:

Enemy Intention: Attack [✓ Defense]

Echelon Unit: Division Brigade [✓ Battalion]

Unit Type: Mech_Infan [✓ Armor] Motorized

Sub-unit Role: [✓ Mech Infan] Armor Motorized

Echelon: Battalion [✓ Company] Platoon

Attack Mission: [✓ Commit(C)] F&S Res(R)

Defend Mission: Delay(Del) [✓ Defend(Def)] Res(R)

Sub-subs: 4

Terrain Information

Total AAs: 2

Total LDTs: 5

Intelligence Information:

[✓ From Intelligence Analysis (CoRaven)] G2/S2

Main effort is at AA: Axis_Red

Depth of defense is LDT: 3

User Setable Attack Constraints:

☑ Don't place more than one F&S on an AA

☐ Don't place more than one Reserve on an AA

☐ Cover all Avenues of Approach

User Setable Defense Constraints:

☐ Don't leave a Defense alone on an AA

☑ Don't leave a Delay alone on an AA

☑ Cover all Avenues of Approach

Figure 6: Weasel's Interface for Specifying Enemy Constraints and Assumptions

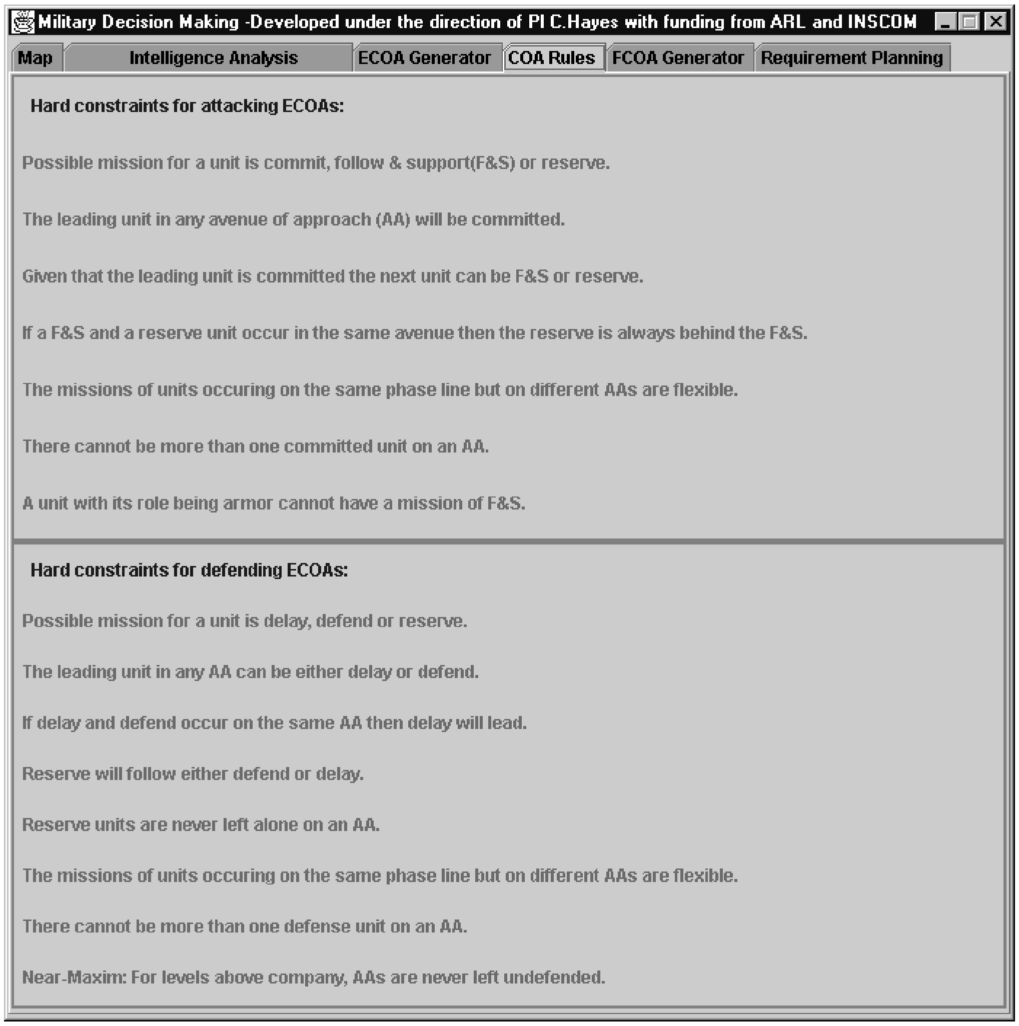
example, the depth of enemy defense is most likely at LDT3, indicated by LDT3's light color. LDT4 is a close second. The analyst can choose how many of these hypotheses to consider. In our example we will focus on the assumption that the enemy has penetrated to LDT3.

Additional intelligence hypotheses (not shown) address the question "Where is the main defense?" In this example, reports cumulatively indicate that the main defense is most probably in the southern AA, Axis Red. This is also given as a constraint to Weasel.

Specification of Assumption and Constraints. Further information which the analyst must specify is: the enemy mission: attack or defend; the size and composition of the enemy forces (battalion, company, platoon, etc) and assumed rules of enemy behavior. Figure 6 shows that, for our example, the analyst has specified the enemy mission as "defense." The enemy unit under consideration is an armor battalion. The analyst further assumes that the battalion is composed of the sub-units shown as red diamonds in the lower left of Figure 6.

Three "soft" rules of enemy behavior are shown in the lower right of Figure 3: "Do not leave a Defense Unit alone in an avenue of approach (AA)," "Do not leave a delay unit alone on an AA," and "Cover all avenues of approach" (with defending units). These are rules which the enemy may or may not follow when planning their COAs. The user can state his or her assumptions about whether or not the enemy will follow these rules by checking (or not checking) the boxes next to the rules. Checking a box turns that rule on. Un-checking it turns the rule off. In the example in Figure 6, the user has chosen assume that the enemy might leave a defense unit alone on an AA, but will not leave a delay unit alone, and will cover all AAs. Weasel's planner uses the checked rules as constraints when constructing enemy COAs.

Additionally, there are "hard" rules of behavior which the enemy will (almost) always follow. These rules are shown in Figure 7. For example, "The leading unit in an AA must be a delay or defense unit." The rules are divided into two sets which apply respectively to enemy offensive (attack), and defensive maneuvers. The rules in Figure 7 are treated as hard constraints because they represent either


Military Decision Making -Developed under the direction of PI C.Hayes with funding from ARL and INSCOM
Map
Intelligence Analysis
ECOA Generator
COA Rules
FCOA Generator
Requirement Planning

Hard constraints for attacking ECOAs:

Possible mission for a unit is commit, follow & support(F&S) or reserve.

The leading unit in any avenue of approach (AA) will be committed.

Given that the leading unit is committed the next unit can be F&S or reserve.

If a F&S and a reserve unit occur in the same avenue then the reserve is always behind the F&S.

The missions of units occuring on the same phase line but on different AAs are flexible.

There cannot be more than one committed unit on an AA.

A unit with its role being armor cannot have a mission of F&S.

Hard constraints for defending ECOAs:

Possible mission for a unit is delay, defend or reserve.

The leading unit in any AA can be either delay or defend.

If delay and defend occur on the same AA then delay will lead.

Reserve will follow either defend or delay.

Reserve units are never left alone on an AA.

The missions of units occuring on the same phase line but on different AAs are flexible.

There cannot be more than one defense unit on an AA.

Near-Maxim: For levels above company, AAs are never left undefended.

Figure 7: Users can view Weasel fixed constraints

definitions which are relatively fixed, or they represent maneuvers that cannot be easily modified without endangering the unit or requiring lengthy preparation on the enemy's part (i.e. training and field exercises). Since few constraints (rules) in any domain can be said to be truly fixed, future work will examine whether to make some of the rules which are currently hard constraints into user settable constraints. Some of these considerations include determining *who* should be allowed to make changes to relatively hard constraints (e.g. domain intermediates, experts or only special system maintainers?) and weighing the utility of adding flexibility (which may be used infrequently) against the possibility that errors will be introduced when users accidentally change relatively hard constraints.

Because these rules considered to be fixed, the user is not permitted to turn them on or off. However, these rules have been made available in Weasel's interface for users to examine should they wish to do so. We feel it is important to make the rules controlling the planner's behavior accessible to the users, and to express them in the users' domain vocabulary, thus de-mystifying the software engine. All too frequently, automated planners and problem solvers are effectively "black boxes" from the users' perspective.

Constraint-Based ECOA generator. Once all constraints and assumptions have been entered, the user can request that Weasel generate all ECOAs consistent with those assumptions. The planner is a simple constraint-based planner that generates all permutations of the resources consistent with the constraints. Although the planner is not complex, it is more systematic about generating all combinations than most humans are, particularly when there are many combinations.

For this example, there are 6 possible ECOAs shown in Figure 8, consistent with the assumptions specified. These ECOAs have been rotated so that they are in the same orientation as they would appear when displayed on the terrain shown in Figure 2.

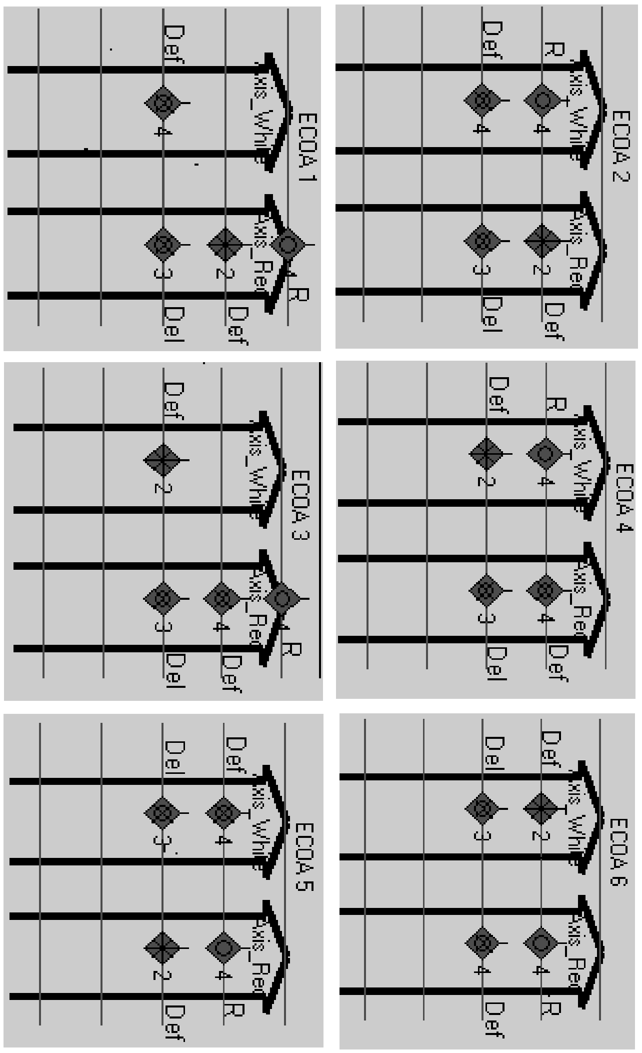


Figure 8: Six ECOAs generated by Weasel which are consistent with the constraints and assumptions in Figure 6.

ECOA Editor. Once ECOAs have been generated, analysts can view them in the plan editor (Figure 8). If he or she is mostly satisfied with the ECOAs but wishes to change a few of their properties, the enemy units can be repositioned by dragging and dropping them. Additionally, specific ECOAs can be selected and viewed in the context of the terrain as shown in Figure 2.

Changing assumptions, repeating the cycle. An important part of the annalist's problem solving is to consider what the enemy might do under a *variety* of different assumptions. For example, what might the enemy do if they decided *not* to leave a defense unit alone on an AA, or *not* to defend all AAs? Weasel's interface allows users to try different "what-if" scenarios defined by sets of assumptions and intelligence constraints, and rapidly see the impact on the likely ECOAs. This is an important function of Weasel's interface because it makes a specific and important task easier.

Next problem solving steps. The analyst's work is not yet complete even after a satisfactory set of ECOAs have been developed. They must select a small set of ECOAs (for computational reasons -- usually between 3 and 6) which they judge to be most relevant or important. This selected set of ECOAs will be used while generating friendly COAs (step 5 in Figure 1) to assess the appropriateness and possible performance of each FCOA considered.

Evaluation Method

Subjects. Eighteen subjects participated in the experiment (9 Air Force and 9 Army subjects). All had between 1 and 21 years of experience in the U.S. armed forces. Five subjects were categorized as experts, and 13 as intermediates; experts were those having at least 6 years of military experience on active duty, in the National Guard or Reserves. Domain novices (those having less than a year experience with the domain) were not used in the evaluation because they lacked sufficient knowledge to perform the task even with Weasel's assistance. The average length of experience of all 18 subjects was 5.03 years.

Scenarios. Subjects were asked to generate ECOAs for 3 different scenarios. Scenario 1 was designed to be difficult, requiring subjects to generate many possible ECOAs. Scenario 2 was designed to be relatively easy, and Scenario 3 was one for which Weasel generated a "brittle" solution set, in that it was incomplete. Solutions in which the enemy protected all possible approaches were not included. Weasel generated eight ECOAs for Scenarios 1, two for Scenario 2, and four for Scenario 3.

Solution Methods. Subjects were asked to generate solutions by three different methods, A, B and C. In Method A, subjects first generated ECOAs by hand, then were shown the ECOAs generated by Weasel and asked to pick between their own solution set and Weasel's. In Method B, subjects again generated solutions first by hand, and then were shown Weasel's solutions. However, this time they could revise their solution set if they so desired. Examples of revisions include copying one of Weasel's ECOAs or incorporating elements of it in one of their own. In Method C, subjects were shown Weasel's solutions first, and then they were asked to generate their own, which could include Weasel's ECOAs, or ECOAs based on them.

Design. All subjects solved all scenarios, and applied all methods. However, to eliminate learning effects, the order in which subjects saw the scenarios and applied the methods was counter-balanced. Given that there are 6 permutations of three items, this suggests a 6x6 experiment requiring 36 subjects. Instead we applied a lattice design

(Montgomery 1991) which reduced the required subjects by half (to 18).

Evaluators. Two evaluators assessed the quality of the ECOA sets generated by the subjects. The evaluators were selected for their expertise in Army battlefield strategy as well as their specific knowledge of current battlefield simulations used in the U.S. Army. One had 9 years U.S. Army experience, and the other 5 years.

Procedure. First, subjects were given familiarization training by the experimenter on a computer workstation. Materials given to subjects included: a scenario instruction page, three pages each describing the scenario, pen, and a one-page list of "required" (hard) constraints used by Weasel to generate ECOAs so that they may understand the computer's behavior.

Next, subjects were given scenario descriptions and asked to generate a set of ECOAs appropriate for each of the three scenarios. An experimenter was present at all times to answer questions. When a subject finished each scenario, they were then asked to provide verbal explanations of their solution choices. Upon completion of all three scenarios subjects, they completed a short questionnaire. Lastly, after all subjects had completed all scenarios, evaluators "scored" all solutions sets (including Weasel's) for each scenario, where best was 10 and worst was 1.

Results

The first steps in analysis were to check 1) the level of agreement between the evaluators and 2) whether there was a significant performance difference between the intermediate and expert subjects when they generated ECOAs by hand, *without* Weasel's assistance. The purpose of the first check was to assess whether evaluators had been chosen appropriately, the assumption being that there is a very low probability that independent evaluators will produce similar quality rankings for many solutions unless they have sufficient experience to assess quality. The purpose of the second check was to assess whether the division between the intermediates and experts was a meaningful one. The correlation for scenario 1 was 0.94, for scenario 2: 0.90 and for scenario 3: 0.99, indicating a high level of agreement between evaluators. In the second check, we compared the average quality ranking given by the evaluators to the intermediate and the expert groups. An ANOVA indicated that the difference between average expert and intermediate quality rankings was very significant, p = 0.001, indicating the experts performed significantly better than intermediates. Once these two issues had been established, we investigated the four questions posed in the introduction:

1. *Did use of Weasel improve the quality of ECOAs generated?* Yes. Overall there was a significant improvement in quality scores when ECOAs generated *without* Weasel's assistance were compared to ECOAs generated *with* Weasel's assistance (p = 0.018). Figure 9 shows the average quality scores received by users without and with Weasel's assistance, as well as the computer's quality scores. As expected, the quality score on the brittle scenario (Scenario 3) was very poor.

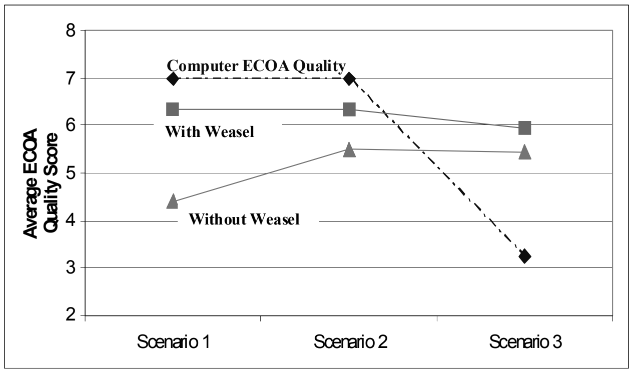


Figure 9: Ave quality scores received by subjects without and with Weasel's assistance (where 10 is best and 1 is worst score).

2. *Did use of Weasel change intermediates' performance more than experts'*? Yes. It improved intermediates' quality scores significantly (p = 0.0002), but did not significantly change experts' quality scores (p = 0.251). Furthermore, differences between experts and intermediates were leveled when both groups used Weasel; there was no significant difference between intermediate and expert quality scores when using Weasel (p = 0.366). This implies that use of Weasel elevates intermediates' ECOA quality to closer to the level of experts.

3. *Did ECOA quality decline when Weasel exhibits brittle behavior?* No. For scenario 3, there was no significant difference in the quality of ECOAs generated without or with Weasel's assistance (p = 0.51). In fact, ECOA quality scores increased on average for all scenarios when users employed Weasel's assistance. However, closer examination of individual subjects performances revels that there is more to the story. When using Weasel's assistance on Scenario 3, *more* subjects' (three out of 18) tended to repeat the mistake made by Weasel on Scenario 3 (i.e. omission of ECOAs that "cover" all avenues of approach). Furthermore, three of the five who made the omission were experts. In contrast, only one subject (an intermediate) made this same mistake when producing solutions manually. This implies that use of Weasel may have "biased" some users towards flawed solution sets when it exhibited brittle behavior, just as FTP biased users towards unnecessarily risky solutions when it exhibited brittle behavior.

4. *Did presentation order change users' the tendency to repeat Weasel's mistakes*? In Smith, McCoy and Layton's study of FPT, they reduced the tendency of users to adapt

the computer's flawed solutions by delaying presentation of the computer's solution until they had explored the problem on their own. However, we did not find a similar effect in this domain. Of the five users who "copied" the computer's mistake on Scenario 3, four generated their own solutions first and only one saw Weasel's solutions first.

Future Work

This work represents a positive start in the right direction. However, we are not going to declare victory yet; there is still much maturation of Weasel that needs to occur (through further development and evaluation) before Weasel can be installed and assessed in the context of a daily work environment. Many issues still need to be investigated and incorporated into system designs. For example, does explicit display of the ECOA generators' fixed constraints allow users to better understand Weasel's behavior, results and limitations? Or do they persist in ascribing highly-nuanced human-like reasoning to the computer, possibly leading to failure to recognize brittle behavior. To what extent does allowing users to manipulate Weasel's soft constraints increase its utility, or decrease its usability? Would allowing users to control more constraints add to Weasel's utility or is there a point where the added complexity of the interface becomes more of a burden than a help to users?

Discussion and Conclusion

We have first examined what we view as the most important "bottom-line" issue: does Weasel improve decision making performance, and if so, for what users? Results show that Weasel results in solution quality gains for users with an intermediate level of domain experience (i.e., 1 ñ 6 years). Based on this result we see potential for use of Weasel in providing practice and training for analysts with an intermediate level of domain experience. However, with supervision from domain experts and with training on how to interpret Weasel's results; users of Weasel, and probably most MIPAS systems, should be trained to regard them as sometimes fallible suggestion generators rather than as oracles, just as they should regard their human counterparts. How successful this training is likely to be is yet another question: will it always be an uphill battle to prevent users from inappropriately regarding computer systems as infallible oracles?

Weasel may also provide benefits to domain experts, for example by reducing the number of times they are "surprised" by unexpected enemy actions. However, further evaluations are needed to determine what, if any benefits domain experts may derive. Lastly, when Weasel exhibited brittle behavior, it still resulted in an average solution quality increase, not a decrease as in the Layton, et al. experiment with FTP. We conclude from this that not all brittle solutions are created equal; the brittle solution examined in the Weasel study was an incomplete solution set. The one examined in the FTP study was a risky point solution. The latter may be a more dangerous form of brittleness than the former.

Decision support systems, of which MIPAS systems are an example, can have both positive and negative impacts on users' performance. The point for readers to take away is that it is that designers and users of MIPAS systems need to be aware that even the best designed system will sometimes exhibit brittle behavior, and both the positive and negative impacts of such systems must be carefully weighed in considering how the system should be used.

References

Ericsson, K. A. & H. A. Simon, "*Protocol Analysis: Verbal Reports as Data*" 1984, MIT Press, Cambridge, MA.

Hayes C.C.; and U. Ravinder, "Weasel: An Automated Planner that Users Can Guide," *IEEE Systems, Man and Cybernetics* (Washington, D.C.; October 5-8, 2003) Paper No. HMS Special P1-5,

Jones, P.M.; C.C. Hayes, D.C. Wilkins, R. Bargar, J. Sniezek, P. Asaro, O. Mengshoel, D. Kessler, M. Lucenti, I. Choi, N. Tu, O. Chernyshenko, M. Liang and J. Schlabach, "CoRAVEN: Modeling and Design of a Multimedia Intelligent Infrastructure for Collaborative Intelligence Analysis," *Federated Laboratories Annual Research Symposium* (Aberdeen, MD; Feb 1999) pp. 3-8.

Montgomery, D. C. (1991). *Design and Analysis of Experiments*. Wiley and Sons, pp. 176-194.

Parasuraman, R. (1997). "Humans and Automation: Use, Misuse, Disuse and Abuse," *Human Factors*, **39**(2): 230-253.

Schlabach, J.L.; C.C. Hayes, and D.E. Goldberg, "FOX-GA: A Genetic Algorithm for Generating and Analyzing Battlefield Courses of Action," *Evolutionary Computation,* **7**(1), pp. 45-68 (1998).

Smith, P. J. and N. D. Geddes, "A Cognitive Systems Engineering Approach to the Design of Decision Support Systems, in "*The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*," Jacko, J. A. and A. Sears (Eds) 2003, Lawrence Erlbaum Associates, Inc, pp. 656 ñ 676.

Smith, P. J., McCoy, C. E., & Layton, C. (1997). "Brittleness in the Design of Cooperative Problem-Solving Systems: The Effects on User Performance." *IEEE Transactions on Systems, Man, and Cybernetics ñ Part A: Systems and Humans*, **27**(3): 360-371.

Delegation Architectures: Playbooks and Policy for Keeping Operators in Charge

Christopher A. Miller

Smart Information Flow Technologies
1272 Raymond Ave
St. Paul, MN 55108 U.S.A.
cmiller@sift.info

Abstract

We argue that as Unmanned Military Vehicles become more intelligent and capable, and as we attempt to control more of them with fewer humans in the loop, we need to move toward a model of delegation of control rather than the direct control (that is, fine grained control with, generally, tight and fast control loops) that characterizes much current practice. We identify and describe five delegation methods that can serve as building blocks from which to compose complex and sensitive delegation systems: delegation through (1) providing *goals*, (2) providing full or partial *plans*, (3) providing *negative constraints*, (4) providing *positive constraints* or *stipulations*, and (5) providing priorities or value statements in the form of a *policy*. We then describe two implemented delegation architectures that illustrate the use of some of these delegation methods: a "playbook" interface for UAV mission planning and a "policy" interface for optimizing the use of battlefield communications resources.

UMV Control as Human-Automation Delegation

While Unmanned Military Vehicles (UMVs—that is, any unmanned vehicle, whether ground, air, sea, undersea or space, used for military purposes) hold the promise of radical change and improvement for a wide range of military applications they also pose a host of challenging problems. Chief among these is how to enable a human operator, who may well be heavily engaged in other tasks of his or her own (such as exploring a building, maintaining radio contact with headquarters or even avoiding fire), to retain sufficient control over the UMV(s) to ensure safe, efficient and productive outcomes. This problem is, of course, magnified when the UMVs may be responsible for the lives of many soldiers or civilians, may be capable of unleashing lethal force on its own, and when a single human may be striving to control groups or even swarms of potentially autonomous and independent actors and may be concurrently engaged in other, high tempo and criticality tasks of his or her own.

Yet this problem is not completely novel. Humans have been striving to retain control and produce efficient outcomes via the behavior of other autonomous agents for millennia. It just so happens that those "agents" have been other humans. Not surprisingly, we have developed many useful methods for accomplishing these goals, each customized to a different domain or context of use. When we have some degree of managerial authority over another human actor and yet will not be directly commanding performance of every aspect of a task, we call the relationship (and the method of commanding task performance) *delegation*. Delegation allows the supervisor to set the agenda either broadly or specifically, but leaves some authority to the subordinate to decide exactly how to achieve the commands supplied by the supervisor. Thus, a delegation relationship between supervisor and subordinate has many requirements:

1. The supervisor retains overall responsibility for the outcome of work undertaken by the supervisor/subordinate team and retains the authority commensurate with that responsibility.
2. The supervisor has the capability to interact very flexibly and at multiple levels with the subordinate. When and if the supervisor wishes to provide detailed instructions, s/he can; when s/he wishes to provide only loose guidelines and leave detailed decision making up to the subordinate, s/he can do that as well—*within the constraints of the capabilities of the subordinate*.
3. To provide useful assistance within the work domain, the subordinate must have substantial knowledge about and capabilities within the domain. The greater these are, the greater the potential for the supervisor to offload tasks (including higher level decision making tasks) on the subordinate.
4. The supervisor must be aware of the subordinate's capabilities and limitations and must either not task the subordinate beyond his/her abilities or

must provide more explicit instructions and oversight when there is doubt about those abilities.

5. There must be a "language" or representation available for the supervisor to task and instruct the subordinate. This language must (a) be easy to use, (b) be adaptable to a variety of time and situational contexts, (c) afford discussing tasks, goals and constraints (as well as world and equipment states) directly (as first order objects), and (d) most importantly, be shared by both the supervisor and the subordinate(s).
6. The act of delegation will itself define a window of control authority within which the subordinate may act. This authority need not be complete (e.g., checking in with the supervisor before proceeding with specific actions or resources may be required), but the greater the authority, the greater the workload reduction on the supervisor.

Items 4 and 6 together imply that the space of control authority delegated to automation is flexible—that the supervisor can choose to delegate more or less "space," and more or less authority within that space (that is, range of control options), to automation. Item 5 implies that the language available for delegation must make the task of delegating feasible and robust—enabling, for example, the provision of detailed instructions on how the supervisor wants a task to be performed or a simple statement of the desired goal outcome.

Types of Delegation

We have developed a variety of architectures within which to support human delegation interactions with automation. Of particular interest as a core enabling technology is the "language" or representation for delegation described in item #5 above. As Klein (1996) points out, without successfully sharing an understanding of the tasks, goals and objectives in a work domain, there can be no successful communication of intent between actors. We believe there are five kinds of delegation actions or delegation methods that should be supported within such a representation, as described in Table 1 below. Note that each method forms a building block, and they can be combined into more effective and flexible composite delegation interactions. Note also that the subordinate has a specific responsibility in response to each method, as articulated below.

In the remainder of this paper, I will described two delegation architectures we are developing. While neither system enables all of the types of delegation described above, and neither is fully implemented yet, collectively they illustrate the five types of delegation and provide a rich and highly flexible set of interactions for human-automation delegation.

Playbook—Delegation of Goals, Plans and Constraints

The first architecture is based on the metaphor of a sports team's playbook. A playbook works because it provides for rapid communication about goals and plans between a supervisor (e.g., a coach) and a group of intelligent actors (the players) who are given the authority to determine how to act within the constraints inherent in the coach's play. Our Playbook architecture supports delegation action types 1-4 in principle and has been implemented in prior prototypes to include action types 2 and 4.

The basic Playbook system architecture is presented in Figure 1. The Playbook 'proper' consists of a User Interface (UI) and a constraint propagation planner known as the Mission Analysis Component (MAC) that communicate with each other and with the operator via a

Table 1. Five types of delegation.

Supervisor's Delegation Action	Subordinate's Responsibility
1. Stipulation of a goal to be achieved—where a goal is a desired (partial) state of the world.	Achieve the goal(s) if possible (via any means available), or report if incapable.
2. Stipulation of a plan to be performed—where a plan is a series of actions, perhaps with sequential or world state dependencies.	Follow the plan if possible (regardless of outcome) or report if incapable.
3. Provide constraints in the form of actions or states to be avoided.	Avoid those states or actions if possible, report if not.
4. Provide "stipulations" in the form of actions or states (i.e., sub-goals) to be achieved.	Achieve those states or perform those actions if possible, report if not.
5. Provide an "optimization function" or "policy" that enables the subordinate to make informed decisions about the desirability of various states and actions	Work to optimize value within the "optimization function" or "policy".

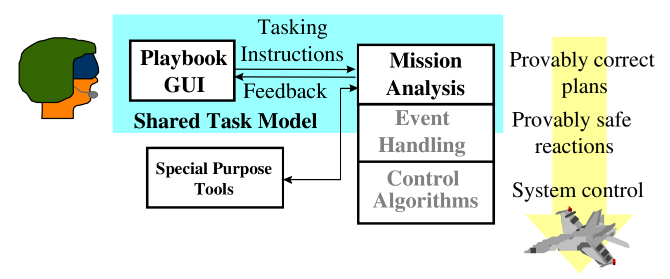


Figure 2. General Playbook Architecture.

Shared Task Model. The operator communicates instructions in the form of desired goals, tasks, partial plans or constraints, via the UI, using the task structures of the shared task model. The MAC is an automated planning system that understands these instructions and (a) evaluates them for feasibility and/or (b) expands them to produce fully executable plans. The MAC may draw on special purpose planning tools (e.g., an optimizing path planner) to perform these functions, wrapping them in its task-sensitive environment. Outside of the tasking interface, but essential to its use, are two additional components. An Event Handling component, itself a reactive planning system capable of making momentary adjustments during execution, takes plans from the Playbook. These instructions are sent to control algorithms that actually effect behaviors.

Operator interaction with the Playbook can be via a variety of user interfaces customized to the needs of the work environment, but operator commands are ultimately interpreted in terms of the Shared Task Model. To date, we have developed prototype playbooks for Unmanned Combat Air Vehicle (UCAV) teams (Miller, Pelican, Goldman, 2000), and Tactical Mobile Robots (Goldman, Haigh, Musliner, Pelican, 2000), and prototypes for the RoboFlag game (Parasuraman, Galster, Squire, Furukawa and Miller, in press) and for real-time interaction with teams of heterogeneous UMVs (Miller, Funk, Goldman and Wu, 2004; Goldman, Miller, Wu, Funk and Meisner, 2005). Below, we provide a description of user interaction with one playbook interface we developed with Honeywell Laboratories to illustrate the general concept.

We developed the playbook illustrated in Figure 2 to enable a human leader to create a full or partial mission

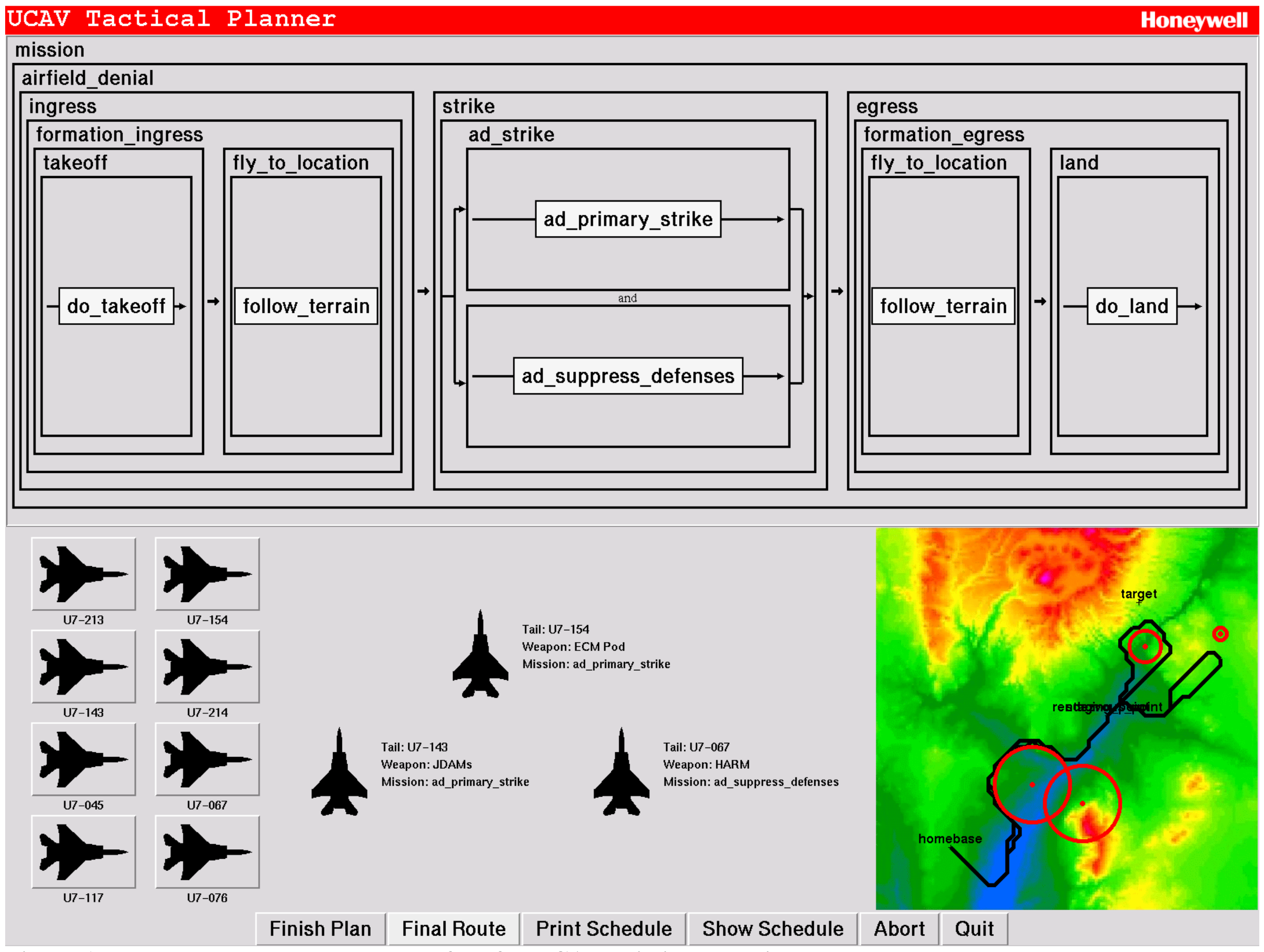


Figure 1. Prototype Playbook User Interface for UCAV Mission Planning.

plan for UCAVs. This initial work was intended as a ground-based tasking interface to be used for a priori mission planning, but current Playbook work is exploring interface modifications to enable real-time and in-flight tasking and task performance monitoring as well.

Figure 2 shows five primary regions of this Playbook UI. The upper half of the screen is a Mission Composition Space that shows the plan composed thus far. In this area, the operator can directly manipulate the tasks and constraints in the plan. The lower left corner of the interface is an Available Resource Space, currently presenting the set of aircraft available for use. The lower right corner contains an interactive Terrain Map of the area of interest, used to facilitate interactions with significant geographic information content. The space between these two lower windows (empty at startup) is a Resource in Use Space—once resources (e.g., UCAVs, munitions, etc.) are selected for use, they will be moved here where they can be interacted with in more detail. Finally, the lower set of control buttons is always present for interaction. This includes options such as "Finish Plan" for handing the partial plan off to the MAC for completion and/or review and "Show Schedule" for obtaining a Gantt chart timeline of the activities planned for each actor, etc.

At startup, the Mission Composition Space presents the three top-level plays (or 'mission types') the system currently knows about: Interdiction, Airfield Denial, and Suppress Enemy Air Defenses (SEAD). The mission leader would interact with the Playbook to, first, declare that the overall mission "play" for the day was, say, "Airfield Denial." In principle, the user could define a new top-level play either by reference to existing play structures or completely from scratch, but this capability has not been implemented yet.

This action is an example of type 2 delegation—providing a specific task for subordinates to perform. But because this is a very high level task in a hierarchical task network, the supervisor has left a great deal of freedom to the subordinates (in this case, the MAC and the UAVs themselves) to determine exactly how a "Airfield Denial" mission is to be performed. If this were the only delegation information the supervisor provided, the subordinates would be obligated to do their best to perform that action (an Airfield Denial mission), but would have a great deal of authority as to how best to accomplish it.

At this point, having been told only that the task for the day is "Airfield Denial," a team of trained pilots would have a very good general picture of the mission they would fly. Similarly, the tasking interface (via the Shared Task Model) knows that a typical airfield denial plan consists of ingress, attack and egress phases and that it may also contain a suppress air defense task before or in parallel with the attack task. But just as a leader instructing a human flight team could not leave the delegation instructions at a simple 'Let's do an Airfield Denial mission today,' so the operator of the tasking interface is required to provide more information. Here, the human must provide four additional items: a target, a homebase, a staging and a rendezvous point. Each of these is a stipulation, or positive constraint, telling the subordinates that whatever specific plan they come up with to accomplish the higher level mission must include these attributes—and thus, they are examples of type 4 delegation interactions. Most of these activities are geographical in nature and users typically find it easier to specify them with reference to a terrain map. Hence, by selecting any of them from the pop up menu, the user enables direct interaction with the Terrain Map to designate an appropriate point. Since the Playbook knows what task and parameter the point is meant to indicate, appropriate semantics are preserved between user and system. As for all plans, the specific aircraft to be used may be selected by the user or left to the MAC. If the user wishes to make the selection, s/he views available aircraft in the Available Resource Space and chooses them by clicking and moving them to the Resources in Use Area.

The mission leader working with a team of human pilots could, if time, mission complexity or degree of trust made it desirable, hand the mission planning task off to the team members at this point. The Playbook operator can do this as well, handing the task to the MAC via the "Finish Plan" button. The leader might wish, however, to provide substantially more detailed delegation instructions. S/he can do this by progressively interacting with the Playbook UI to provide deeper layers of task selection, or to impose more stipulations on the resources to be used, waypoints to be flown, etc. For example, clicking on "Airfield Denial" produces a pop-up menu with options for the user to tell the MAC to "Plan this Task" (that is, develop a plan to accomplish it) or indicate that s/he will 'Choose airfield denial' as a task that s/he will flesh out further. The pop-up menu also contains a context-sensitive list of optional subtasks that the operator can choose to include under this task. This list is generated by the MAC with reference to the existing play structures in the play library, filtered for current feasibility.

After the user chooses 'Airfield Denial' the system knows, via the Shared Task Model, that this task must include an Ingress subtask (as illustrated in Figure 2). The supervisor does not have to tell intelligent subordinates this; it is a part of their shared knowledge of what an 'Airfield Denial' task means—and how it must be performed. To provide detailed instructions about how to perform the Ingress task, however, the user can choose it, producing a "generic" Ingress task template or "play". This is not a default method of doing "Ingress" but a generic, uninstantiated template—corresponding to what a human expert knows about what constitutes an Ingress task and how it can or should be performed. A trained pilot knows that Ingress can be done either in formation or in dispersed mode and, in either case, must involve a "Take Off" subtask followed by one or more "Fly to Location" subtasks. Similarly, the user can select from available options (e.g., formation vs.

dispersed Ingress, altitude constraints on takeoff, etc.) on context-sensitive, MAC-generated menus appropriate to each level of decomposition of the task model. One of our current challenges in creating Playbooks^{TM} for real-time interactions is to enable them to be sensitive to the current state of affairs and of task performance so as to make intelligent assumptions about task performance possible—for example, if the supervisor wishes to command a currently airborne UAV, perhaps in a holding pattern, to perform an 'Airfield Denial' mission, both supervisor and subordinate should know that the Takeoff portion of an Ingress task is no longer necessary and should either be eliminated or be shown as already accomplished.

The user can continue to specify and instantiate tasks down to the "primitive" level where the sub-tasks are behaviors the control algorithms (see Figure 1) on the aircraft can be relied upon to execute. Alternatively, at any point after the initial selection of the top level mission task and its required parameters, the supervisor can hand the partly developed plan over to the MAC for completion and/or review. In extreme cases, a viable "Airfield Denial" plan for multiple aircraft can be created in our prototype with as few as five selections and more sophisticated planning capabilities could readily reduce this number. But potentially more important, the operator (like a human supervisor dealing with intelligent subordinates) can also provide more detailed instructions whenever s/he deems them necessary or useful to mission success and in the way s/he sees fit.

This Playbook illustrates delegation interactions 2 and 4 (plans and stipulations). The subordinates' role in these types of interaction are described in the table above—to perform the plan through any set of sub-methods that adhere to the stipulations provided by the supervisor, or to report that this is infeasible. One of the MAC's roles in the above example is to report when it is incapable of developing a viable plan within the constraints imposed, (e.g., if the user has stipulated distant targets that exceed aircraft fuel supplies). In a real-time delegation system, the MAC will be responsible for continual monitoring of performance to report when world states mean that plan performance is no longer capable of (or likely to) accomplish the user's parent plan (e.g., because of equipment failures, adverse head winds, enemy countermeasures, etc.)

The Playbook architecture is, we believe, also capable of supporting delegation interaction types 1 and 3 (goals and negative constraints) as well. Supporting goal-based delegation interactions would require a slight modification to the shared task representation. Currently, we have used a representation that explicitly includes only hierarchically organized and sequenced tasks (i.e., actions to be performed). Tasks implicitly encode the goals they accomplish, but there are representations (such as Geddes Plan-Goal Graphs—Sewell and Geddes, 1990) that explicitly interleave both plans and goals and a linked hierarchy. Use of such a representation, along with related modifications to the UI and MAC, would enable the supervisor to say, effectively, "Today we're going to achieve a State" (e.g., the destruction of a given airfield) rather than or in addition to, the plan-based representation used above which allows only the issuing of task-based delegation commands (e.g., "Today we're going to fly an airfield-denial mission"). The incorporation of negative constraints into the interaction (delegation interaction method #3), would require a less substantial modification to the Playbook architecture—potentially requiring only a UI addition to enable the supervisor to incorporate negative commands about task types and state parameters (e.g., "do NOT fly through this valley or use this type of munition") and then requiring the MAC to create plans which avoid those negative constraints.

Policy—Delegation via Abstract Policy Statements

The final type of delegation interaction offers the ability to provide priorities between alternate goals and states and to do so more abstractly than the above methods. Sometimes supervisors don't have a single, concrete world state goal in mind, much less a specific plan for accomplishing it. Sometimes supervisors must issue commands well in advance to cover a wide range of largely unanticipatable circumstances. In these cases, the delegation instructions will be less a specific statement of actions to take or world states to be sought or avoided, but rather a general statement of outcomes that would be more or less good or valuable (or, conversely, bad or to be avoided) than others. We refer to the set of such abstract value statements that a supervisor might provide as his or her "policy" for performance in the domain.

We have developed policy-based architectures for two applications: providing commanders' guidance to a resource controller for battlefield network communications (prioritizing communications bandwidth in accordance with the commander's intent—Funk, Miller Johnson and Richardson, 2000), and providing visualization and feedback to dispatchers in upset contexts in commercial aviation (Dorneich, Whitlow, Miller and Allen, 2004). We will describe the first of these below.

A policy statement is an abstract, general, a priori statement of the relative importance or value of a goal state in the domain. In its simplest form, policy provides a method for human operators to mathematically define what constitutes "goodness." Once defined, a policy statement can be treated as a rule and evaluated against a current or hypothetical context—if the rule is true in the context, then the context incurs the "goodness" (or badness) value stipulated by the rule. Alternate contexts (which could be tied to the expected outcomes of alternate decisions) can then be evaluated against each other by examining the set of policy rules that are satisfied or violated and the resulting set of goodness/badness values accrued. A set of individual policy statements can be bundled together, and these policy bundles can be used to flexibly define the priorities that apply in a given situation (priorities can change given different circumstances).

A policy-based delegation system requires at least three components: (1) a representation for specifying the "policy" in terms of the value of various partial world states, (2) a user interface for allowing one or more users to input their policies and, if desired, view results of policy application, (3) a computational framework that allows evaluation of a current situation or hypothetical proposed situation against the expressed policy, and (4) an engine that allows application of the policy either to the control of resource application or to a visualization of sensed data about a current situation or projected data about a future or simulated situation.

In the development of a policy-based delegation system for communications resource usage, we were striving to provide a means for commanders to tell an automated network management system their "policies" for how to prioritize the use of communications bandwidth in order to satisfy the most important requests most fully. Note, however, that "most important" was not a static concept but rather changed across commanders and situations. For this application, we developed a policy representation that allowed commanders to assign, a priori and abstractly, a value to various kinds of communications requests. As communications requests then came in from various field units or operators, they could be matched against the commander's policy statements and a value assigned to each of them. This value was then used by a resource optimizing controller to determine which requests should get network bandwidth with what priority.

This process is conceptually illustrated in Figure 3. Each commander's policy is created as a set of statements (as illustrated at the top of the figure) each of which assigns an importance (or value) function to a defined sub-region in a multidimensional space. In this case, individual policy statements are illustrated abstractly as defined by the dimensions: owner (Wi) who is the originator of an information request, source (Si) which is the location of the information to be transmitted, destination (Di) which is the destination to which the information is to be transmitted (i.e., a specific machine or IP address, which need not be that of the owner), and description of the information content (Ci) to be transmitted, along with an importance assigned to that policy statement. For example, policy statements might be based on a single dimension ('Requests for weather information [Content] get Importance 0.2') or on a combination of dimensions ('Requests owned by the Zone Reconnaissance task [Owner] for weather information [Content] from Satellite 476B [Source] to 3rd Air Calvary Division [Destination] get Importance 0.8). If the policy element regions are allowed to overlap, then they must be sequenced (typically from most to least specific) to indicate the order of precedence.

	Owner	Source	Destination	Content	Importance	
(	{W1}	{S1}	{D1}	{C1}	.9	)
(	{W2}	{S2}	{D2}	{C2}	.8	)
(	{W3}	{S3}	{D3}	{C3}	.5	)
(	*	*	*	*	.1	)

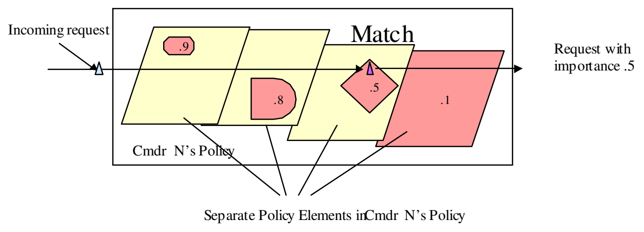

Incoming request
.9
.8
Match
.5
.1
Request with importance .5
Cmdr N's Policy
Separate Policy Elements in Cmdr N's Policy

Figure 3. Representation for a policy for network bandwidth prioritization.

In practice, the commander's policy is then used to assign an importance value to any incoming request for communication resources (illustrated conceptually in the lower portion of Figure 3). Each policy statement defines a region within the multidimensional space defined by the parameters from which policy statements can be built. We've represented that multidimensional space as a simple plane in the figure, and then defined multiple regions within that space with an assigned value for each to represent the valuation contained in each separate policy statement. Each incoming request is matched against the sequenced series of policy statements the commander has made. The first policy element that matches the request determines the importance of that request and informs an automated resource manager about the relative value of satisfying that request.

While conceptually simple, many useful functions can be performed within this framework. First, it is not necessary that importance be construed as an all-or-nothing value as it is depicted in Figure 3. Instead, we have explored more sophisticated representations that allow the requestor to provide a description of how s/he wants the information requested along several dimensions (e.g., freshness, reliability, initiation-time, accuracy, resolution, scope, etc.) Then the resource management system can treat the importance value as a maximum number of value "points" to be awarded for satisfying the request perfectly, while still awarding itself points for partial satisfaction. This permits more sensitive management of resources to be performed.

Second, it is rarely the case that a single commander or supervisor is the only one who may have an interest in dictating policy about how subordinates behave. Rather, each commander must allocate his/her resources in accordance with the policies of those above. We support this requirement (Figure 4) by modeling policies that exist at nodes in a command hierarchy. As requests come in, they are matched against the commander's policy that governs them (command node N1.2.2 in Figure 4), but must then also be matched against his/her commander's policy (i.e., the command node in charge of node N1.2.2 in

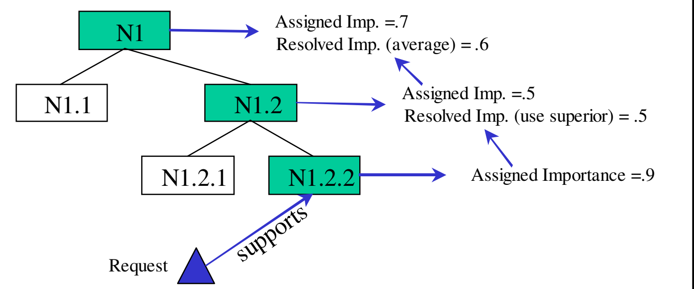


Figure 4. Applying and resolving policy across echelons in a command hierarchy.

Figure 4—node N1.2)—and so on, up the chain of command (i.e., node N.1 in Figure 4). We allow each commander to stipulate how this matching policy element should be resolved with the subordinate commander's matching policy element: as a ceiling or floor value, or linear or weighted combination of the values. Even when a single well-defined chain of command does not exist the policies of different "interest groups" may be represented with relative weights on the importance values that each would assign to a potential outcome. We have used this approach (Dorneich, et al., 2004) in representing the many, varied interests which impact the decisions of a commercial airline's dispatch operators (e.g., crew scheduling, maintenance scheduling, marketing, passengers, finance, etc.).

Folding this policy-based form of delegation interaction (method 5) into an overall architecture that includes the other methods is not as difficult as it might first appear. While we have not yet developed a system that accomplishes this, the way forward is clear. Policy is simply an assignment of value or priority to the goal states and tasks in the other delegation interaction types. Priorities for resource usage and the desirability of various outcomes stem, after all, from a superior's goals and plans for subordinates (whether human or machine). If, for example, I task a given unit under my command to perform an Airfield Denial task, and I know that their task is the most important of all concurrent tasks to me, then I have effectively said that giving them the resources they require to perform that task (specific aircraft, munitions, fuel, communications bandwidth, etc.) represents the highest value to me. In other words, delegation interactions that provide specific goals, plans, stipulations and constraints to subordinates carry with them specific policy implications. Whenever a commander can provide more specific delegation instructions, this will generally get him/her closer to the results desired from his/her subordinates, but this will not always be the case. Hence, the ability to stipulate more abstract policies should probably be preserved in a complete delegation system as a means of covering unexpected and unfamiliar situations.

Conclusions and Future Work

While the work described above represents a general framework for delegation interactions suitable for human interaction with smart automation of various kinds and, perhaps uniquely, suitable for the tasking of multiple UMVs, our work has thus far progressed only to the proof of concept stage. As noted above, we have currently implemented only portions of the various methods of delegation that a fully flexibly delegation interface might benefit from, and have done so in disparate systems. Furthermore, our proof of concept implementations have not yet afforded us the opportunity to do rigorous human in the loop evaluations to demonstrate improved performance, if any.

These situations are changing, however. We are currently engaged in exploration of human interaction with Playbook -like interfaces (Parasuraman, et al., in press) and are performing work on a Playbook interface for real-time interactions with heterogeneous UMV assets by operators who may be concurrently involved in other critical tasks (under a DARPA-IXO SBIR grant-- cf. Miller, et al., 2004; Goldman, et al., 2005). One of the goals of this work will be to develop task libraries and task construction tools and interface concepts to move the delegation interface work along toward implementation and utility.

Of course, anyone who has worked with a poorly trained, or simply mismatched, subordinate is well aware that it is possible for delegation to cause more work than it saves. Our challenge, and that of others who adopt a delegation framework for human interaction with complex and largely autonomous automation, will be to ensure that this does not happen--through judicious use of technology and substantial usability analysis and testing. On the positive side, however, we benefit from the knowledge that delegation approaches to interaction with intelligent yet subordinate actors have worked repeatedly throughout history and, particularly, the history of warfare. As automation in the form of UMVs increasingly takes its place as one of those actors we want to be intelligent, capable and effective yet remain subordinate, we will increasingly need methods for enabling it to interact with us in the ways that we trust and are familiar with. Since delegation is the primary method that fits that bill, it only makes sense to pursue delegation approaches to human interaction with automation.

Acknowledgements

An earlier version of this paper was presented at the NATO Research and Technology Organization Special Workshop on Uninhabited Military Vehicles, June 10-13, 2003; Leiden, Holland.

Many people have contributed to the ideas presented above. A partial list includes Robert Goldman, Harry Funk, Michael Pelican, Dave Musliner, Karen Haigh, Michael Dorneich, Stephen Whitlow, John Allen, and Jim Richardson.

References

Dorneich, M.C., Whitlow, S. D., Miller, C. A., Allen, J. A. 2004. A Superior Tool for Airline Operations. *Ergonomics in Design, 12*(2). 18-23.

Funk, H., Miller, C., Johnson, C. and Richardson, J. 2000. Applying Intent-Sensitive Policy to Automated Resource Allocation: Command, Communication and Most Importantly, Control. In *Proceedings of the Conference on Human Interaction with Complex Systems*. Urbana-Champaign, Ill. May.

Goldman, R.P., K. Haigh, D. Musliner, & M. Pelican. 2000. MACBeth; A Multi-Agent, Constraint-based Planner. In *Notes of the AAAI Workshop on Constraints and AI Planning*, Austin, TX, pp. 1-7.

Goldman, R., Miller, C., Wu, P., Funk, H. and Meisner, J. 2005. Optimizing to Satisfice: Using Optimization to Guide Users. In *Proceedings of the American Helicopter Society's International Specialists Meeting on Unmanned Aerial Vehicles*. January 18-20; Chandler, AZ.

Klein, G. 1998. *Sources of Power: How People Make Decisions*. Cambridge, MA; MIT Press.

Miller, C., Pelican, M. and Goldman, R. 2000. "Tasking" Interfaces for Flexible Interaction with Automation: Keeping the Operator in Control. In *Proceedings of the Conference on Human Interaction with Complex Systems*. Urbana-Champaign, Ill. May.

Miller, C., Funk, H., Goldman, R. and Wu, P. 2004. A "Playbook" for Variable Autonomy Control of Multiple, Heterogeneous Unmanned Air Vehicles. In *Proceedings of the 4th Conference on Human Performance, Situation Awareness and Automation.* Daytona Beach, FL; March 22-25.

Parasuraman, R., Galster, S., Squire, P., Furukawa, H. and Miller, C. In press. A Flexible Delegation-Type Interface Enhances System Performance in Human Supervision of Multiple Robots: Empirical Studies with RoboFlag. Accepted for inclusion in J. Adams, guest ed. *IEEE Systems, Man and Cybernetics—Part A, Special Issue on Human-Robot Interactions*.

Sewell, D. and Geddes, N. 1990. A plan and goal based method for computer-human system design. In D. Diaper, ed. *Human-Computer Interaction—INTERACT '90*. Elsevier Science; North-Holland. pp. 283-288.

A Mixed-Initiative Approach to Human-Robot Interaction in Rescue Scenarios

Alberto Finzi
Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, 00198 Roma Italy
finzi@dis.uniroma1.it

Andrea Orlandini
Dipartimento di Informatica e Automazione
Università degli Studi di Roma TRE
Via della Vasca Navale 79, 00146 Roma Italy
orlandin@dia.uniroma3.it

Abstract

In this paper we present a mixed-initiative planning approach to human-robot interaction in a rescue domain. We deploy a model-based executive monitoring system to coordinate the operator's interventions and the concurrent activities of a rescue rover. We show that this approach can enhance both operator situation awareness and human-robot interaction for the execution and control of the diverse activities needed in rescue missions. We have implemented this control architecture on a robotic system (DORO) and tested it in rescue arenas comparing its performances in different settings.

Introduction

Urban search and rescue (USAR) deals with response capabilities for facing urban emergencies, and it involves the location and rescue of people trapped because of a structural collapse. Starting in 2000, the National Institute of Standard Technology (NIST) together with the Japan National Special Project for Earthquake Disaster Mitigation in Urban Areas (Tadokoro *et al.* 2000; Tadokoro 2000; Maxwell *et al.* 2004; Jacoff, Messina, & Evans 2001) has initiated the USAR robot competitions. NIST, in particular, features future standards of robotics infrastructures, pioneering robotics participation to rescue missions. RoboCup Rescue contests are a test-bed of the technology development of NIST project, and are becoming a central international event for rescue robots, and a real challenge for the robotics community. Rescue robots uphold human operators exploring dangerous and hazardous environments and searching for survivors.

A crucial aspect of rescue environments, discussed in (Burke *et al.* 2004) and (Murphy 2004) concerns the operator situation awareness and human-robot interaction. In (Murphy 2004) the difficulties in forming a mental model of the "robot eye" are endorsed, pointing out the role of the team. Differently from real tests, like the one in Miami (see (Burke *et al.* 2004)), during rescue competitions the operator is forced to be alone while coordinating the robot activities, since any additional team member supporting the operator would penalize the mission. The operator can follow the robot activities only through the robot perception of the environment, and its internal states. In this sense, the overall control framework has to capture the operator attention towards "what is important" so as to make the correct choices: follow a path, enter a covert way, turn around an unvisited corner, check whether a visible victim is really reachable, according to some specific knowledge acquired during the exploration. In this setting, a fully manual control over a robot rescue is not effective (Bruemmer *et al.* 2003): the operator attention has to be focused over a wide range of activities, losing concentration on the real rescue mission objective, i.e. locating victims. Moreover, a significant level of training is needed to teleoperate a rescue rover. On the other hand, fully autonomous control systems are not feasible in a rescue domain where too many capabilities are needed. Therefore, the integration of autonomous and teleoperated activities is a central issue in rescue scenarios and has been widely investigated (Kiesler & Hinds 2004; Yanco & Drury 2002; Drury, Scholtz, & Yanco 2003; Michael Baker & Yanco 2004; Yanco & Drury 2002).

In this work we describe a mixed-initiative planning approach (Ai-Chang *et al.* 2004; Myers *et al.* 2003; Allen & Ferguson 2002; Burstein & McDermott 1996) to Human-Robot Interaction (HRI) in a rescue domain and illustrate the main functionalities of a rescue robot system[1]. We deploy a model-based executive monitoring system to interface the operators' activities and the concurrent functional processes in a rescue rover. In this setting, the user's and the robot's activities are coordinated by a continuos reactive planning process which has to (i) check the execution status with respect to a declarative model of the system; (ii) provide proactive activity while mediating among conflicting initiatives. In particular, we show that this approach can enhance both the operator situation awareness and human-robot interaction for the execution and control of the diverse activities needed during a complex mission such as the rescue one.

The advantage of this approach can be appreciated considering the HRI awareness discussed in (Drury, Scholtz, & Yanco 2003):

- robot-human interaction: given a declarative model of the robot activities, the monitoring system can be "self-aware" about the current situation, at different levels of

[1]Doro is the third award winner in Lisbon contest (2004)

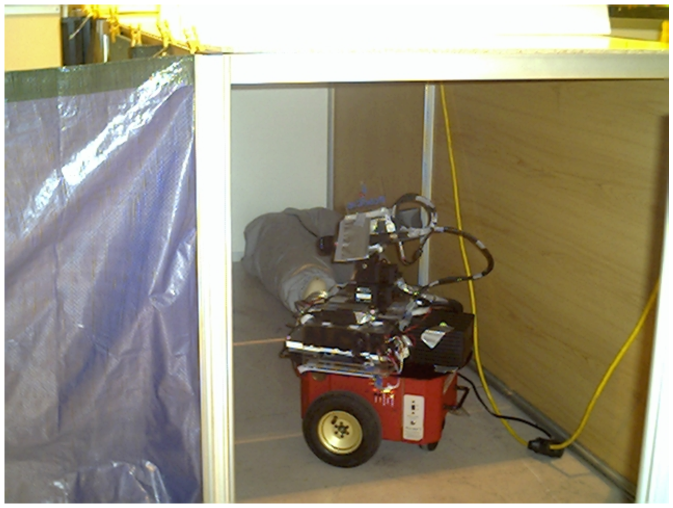


Figure 1: The mobile robot DORO, in a yellow arena.

abstraction; in this way, complex and not nominal interactions among activities can be detected and displayed to the operator;

- human-robot interaction: the operator can take advantage of basic functionalities like mapping, localization, learning vantage points for good observation, victim detection, and victim localization; these functionalities purposely draw his attention toward the current state of exploration, while he interacts with a mixed initiative reactive planner (Ai-Chang *et al.* 2004).

Finally, the humans' overall mission can take advantage of the model, that keeps track of the robot/operator execution history, goals, and subgoals. Indeed, the proposed control system provides the operator with a better perception of the mission status.

Rescue Scenario

NIST has developed physical test scenarios for rescue competitions. There are three NIST arenas, called yellow, orange, and red, of varying degrees of difficulty. A yellow arena represents an indoor flat environment with minor structural damage (e.g. overturned furniture), an orange arena is multilevel and has more rubble (e.g. bricks), a red one represents a very damaged unstructered environment: multilevel, large holes, rubber tubing etc. The arenas are accessible only by mobile robots controlled by one or more operators from a separated place. The main task is to locate as many victims as possible in the whole arena.

Urban search and rescue arena competitions are very hard test-beds for robots and their architectures. In fact, the operator-robot has to coordinate several activities: exploring and mapping the environment, avoiding obstacles (bumping is severely penalized), localizing itself, searching for victims, correctly locating them on the map, identifying them through a numbered tag, and finally describing their own status and conditions.

For each mission there is a time limit of 20 minutes, to simulate the time pressure in a real rescue environment. In this contest human-robot interaction has a direct impact on the effectiveness of the rescue team performance.

We consider the NIST yellow arena as the test-bed for our control architecture. It is mounted on our robotic platform (DORO) whose main modules are: *Map*, managing the algorithm of map construction and localization; *Navigation*, guiding the robot through the arena with exploration behaviour and obstacle's avoidance procedures; *Vision*, used in order to automatically locate victims around the arena.

In this context, (Murphy 2004) propose a high level sequence tasks cycle as a reference for the rescue system behaviour: Localize, Observe general surroundings, look specially for Victims, Report (LOVR). Our interpretation of the cycle corresponds to the following tasks sequence: map construction, visual observation, vision process execution and victim's presence report.

Human Robot Interaction and Mixed Initiative Planning in Rescue Arenas

There have been several efforts to establish the essential aspects of human-robot interaction, given the current findings and state of the art concerning robot autonomy and its modal-abilities towards humans and environments (see e.g.(Dautenhahn & Werry 2000; Kiesler & Hinds 2004; Burke *et al.* 2004; Sidner & Dzikovska 2002; Lang *et al.* 2003) and the already cited (Murphy 2004; Michael Baker & Yanco 2004; Yanco & Drury 2002; Drury, Scholtz, & Yanco 2003), specifically related to the rescue environment. It is therefore crucial to model the interaction in terms of a suitable interplay between supervised autonomy (the operator is part of the loop, and decides navigation strategies according to an autonomously drawn map, and autonomous localization, where obstacle avoidance is guaranteed by the robot sensory system) and full autonomy (e.g. visual information is not reliable because of darkness or smoke etc., and the operator has to lean upon the robot exploration choices).

In order to allow the tight interaction described above, we designed a control system where the HRI is fully based on a mixed-initiative planning activity. The planning process is to continuously coordinate, integrate, and monitor the operator interventions and decisions with respect to the ongoing functional activities, taking into account the overall mission goals and constraints. More precisely, we developed an interactive control system which combines the following features:

- **Model-based control.** The control system is endowed with declarative models of the controllable activities, where causal and temporal relations are explicitly represented (Muscettola *et al.* 2002; Williams *et al.* 2003; Muscettola *et al.* 1998). In this way, hard and soft constraints can be directly encoded and monitored. Furthermore, formal methods and reasoning engines can be deployed either off-line and on-line, to check for consistency, monitor the executions, perform planning or diagnosis. In a mixed-initiative setting the aim of a model-based system is twofold: on the one hand the operator activities are explicitly modeled and supervised by the control system; on the other hand, the model-based monitoring activity exports a view of the system that is intuitive and readable by humans, hence the operator can further supervise the robot status in a suitable human robot interface.

- **Reactive executive monitoring.** Given this model, a reactive planning engine can monitor both the system's low-level status and the operator's interventions by continuously performing sense-plan-act cycles. At each cycle the reactive planner has to: (i) monitor the consistency of the robot and operator activities (w.r.t. the model) managing failures; (ii) generate the robot's activities up to a planning horizon. The short-range planning activity can also balance reactivity and goal-oriented behaviour: short-term goals/tasks and external/internal events can be combined while the planner tries to solve conflicts. In this way, the human operator can interact with the control system through the planner in a mixed initiative manner.
- **Flexible interval planning.** At each execution cycle a flexible temporal plan is generated. Given the domain uncertainty and dynamics, time and resources cannot be rigidly scheduled. On the contrary, it is necessary to account for flexible behaviours, allowing one to manage dynamic change of time and resource allocation at execution time. For this reason the start and end time of each scheduled activity is not fixed, but the values span a temporal interval.
- **High-level agent programming.** The high-level agent programming paradigm allows one to integrate procedural programming and reasoning mechanisms in a uniform way. In this approach, the domain's first principles are explicitly represented in a declarative relational model, while control knowledge is encoded by abstract and partial procedures. Both the system's and the operator's procedural operations can be expressed by high-level partial programs which can be completed and adapted to the execution context by a program interpreter endowed with inference engines.

Control Architecture

In this section, we describe the control system we have defined to incorporate the design principles introduced above. Following the approach in (Muscettola *et al.* 2002; Williams *et al.* 2003; Volpe *et al.* 2001; Finzi, Ingrand, & Muscettola 2004) we introduce a control system where decision processes (including declarative activities and operator's interventions) are tightly coupled with functional processes through a model-based executive engine. Figure 2 illustrates the overall control architecture designed for DORO. The physical layer devices are controlled by three functional modules associated to the main robots activities (mapping and localization, visual processing, and navigation). The *state manager* and *task dispatcher* in the figure are designed to manage communication between the executive and functional layers.
The *state manager* gets from each single module its current status so that any module can query the state manager about

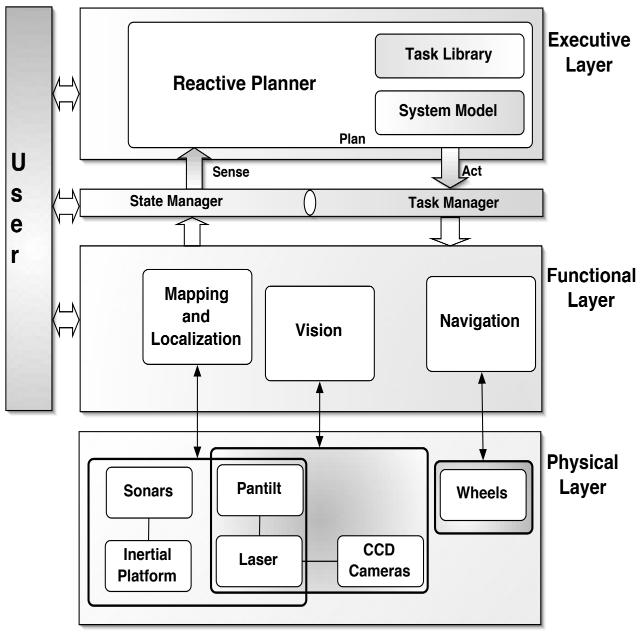


Figure 2: Control architecture

the status of any another module. The state manager updates its information every 200 msec., the task dispatcher sends tasks activation signals to the modules (e.g. map_start) upon receiving requests from the planner or the human operator. The overall computational cycle works as follows: the planner gets the modules status querying the state manager. Once the state manager provides the execution context, the planner produces a plan of actions (planning phase about 0.5 sec.) and yields the first set of commands to the task dispatcher. In the execution phase (about 0.5 sec.), each module reads the signals and starts its task modifying its state. At the next cycle start, the planner reads the updated status through the state manager and can check whether the tasks were correctly delivered. If the status is not updated as expected, a failure is detected, the current plan is aborted and a suitable recovery procedure is called.

Functional Modules. As mentioned above, the functional layer is endowed with three main modules: *Mapping and Localization*, *Navigation*, and *Vision*. These modules provide different tasks that can be activated or stopped according to the *start* or *end* actions communicated by the task dispatcher.

A functional module is a reactive component that changes its internal status with respect to the action received from the task dispatcher. Nevertheless, it can also provide some proactiveness, by suggesting the planner/operator an action to be executed. For instance, the *Slam* module assumes a particular mode in order to communicate to the system that a map's construction cycle is ended, and then the control system can decide an action to stop the mapping phase. Morover, some modules can directly interact among them by communicating some low-level information bypassing the state manager (and the executive layer), e.g. *Slam* devises to *Navigation* the coordinates of the nearest unexplored point during the exploration phases.

User interaction. The human operator can interact with the control loop both during the plan and the act phase. In the planning phase, the operator can interact with the control system by: (i) posting some goals which are to be integrated in the partial plan already generated; (ii) modifying the generated plan through the user interface; (iii) on-line changing some planning parameters, like the planning horizon, the lenght of the planning cycle, etc.. In the executive phase, the user can directly control some functional modules (e.g., deciding where the rover is to go, or when some activities are to stop). In this case, the human actions are assimilated to exogenous events the monitoring system is to manage and check. Finally, the operator's actions can be accessed by the state manager, and, analogously to the functional modules, can be monitored by the model-based control system.

Model-Based Monitoring

The role of a model-based monitoring system is to enhance both the system safeness and the operator situation awareness. Given a declarative representation of the system causal and temporal properties, the flexible executive control is provided by a reactive planning engine which harmonizes the operator activity (commands, tasks, etc.) with the mission goals and the reactive activity of the functional modules. Since the execution state of the robot is continuously compared with a declarative model of the system, all the main parallel activities are integrated into a global view and subtle resources and time constraints violations can be detected. In this case the planner can also start or suggest recovery procedures the operator can modify, neglect, or respect. Such features are implemented by deploying *high-level agent programming* in Temporal Concurrent Golog (Reiter 2001; Pirri & Reiter 2000; Finzi & Pirri 2004) which provides both a declarative language (i.e. Temporal Concurrent Situation Calculus (Pinto & Reiter 1995; Reiter 1996; Pirri & Reiter 2000)) to represent the system properties and the planning engine to generate control sequences.

Temporal Concurrent Situation Calculus. The Situation Calculus (SC) (McCarthy 1963) is a sorted first-order language representing dynamic domains by means of *actions*, *situations*, i.e. sequences of actions, and *fluents*, i.e. situation dependent properties. Temporal Concurrent Situation Calculus (TCSC) extends the SC with time and concurrent actions. In this framework, concurrent durative processes (Pinto & Reiter 1995; Reiter 1996; Pirri & Reiter 2000) can be represented by fluent properties started and ended by durationless actions. For example, the process $going(p_1, p_2)$ is started by the action $startGo(p_1, t)$ and it is ended by $endGo(p_2, t')$.

Declarative Model in TCSC. The main processes and states of DORO are explicitly represented by a declarative dynamic-temporal model specified in the Temporal Concurrent Situation Calculus (TCSC) . This model represents cause-effect relationships and temporal constraints among the activities: the system is modeled as a set of *components* whose state changes over time. Each component (including the operator's operations) is a concurrent thread, describing its history over time as a sequence of states and activities. For example, in the rescue domain some components are: *pant-tilt*, *slam*, *navigation*, *visualPerception*, etc.

Each of these is associated with a set of processes, for instance some of those are the following: *SLAM* can perform $slmMap$ to map the environment and $slmScan$ to acquire laser measures; *visualPerception* can use $visProcess(x)$ to process an image x. *navigation* can explore a new area ($nvWand$) or reach a target point x ($nvGoTo$); *pan-tilt* can deploy $ptPoint(x)$ (moving toward x) and $ptScan(x)$ (scanning x). The history of states for a component over a period of time is a *timeline*. Figure 3 illustrates a possible evolution of $navigation$, $slam$, and $pan\text{-}tilt$ up to a planning horizon.

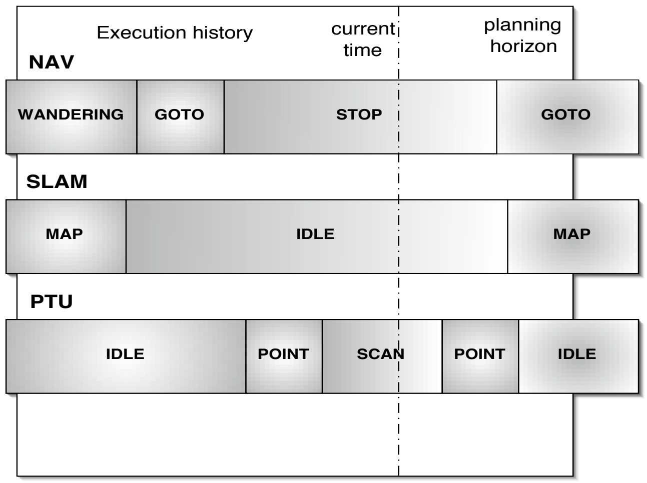


Figure 3: Timelines evolution

Hard time constraints among activities can be defined by a temporal model using Allen-like temporal relations, e.g.: $ptPoint(x)$ *precedes* $ptScan(x)$, $ptScan(x)$ *during* $nvStop$, etc..

Temporal Concurrent Golog. Golog is a situation calculus-based programming language which allows one to define procedural scripts composed of primitive actions explicitly represented in a SC action theory. This hybrid framework integrates procedural programming and reasoning about the domain properties. Golog programs are defined by means of standard (and not so-standard) Algol-like control constructs: (i) action sequence: $p_1; p_2$, (ii) test: $\phi?$, (iii) nondeterministic action choice $p_1|p_2$, (iv) conditionals, while loops, and procedure calls. Temporal Concurrent Golog (TCGolog) is the Golog version suitable for durative and parallel actions, it is based on TCSC and allows parallel action execution: $a\|b$. An example of a TCGolog procedure is:

$$\begin{array}{l}\mathbf{proc}(observe(x),\\ \ \mathbf{while}\ (nvStop \wedge \neg obs(x))\ \mathbf{do}\ \pi(t_1, start(t_1)? :\\ \quad [\mathbf{if}\ (ptIdle(0))\ \mathbf{do}\ \pi(t_2, startPoint(x,t_1) : (t_2 - t_1 {<} 3)?)|\\ \quad\ \mathbf{if}\ (ptIdle(x))\ \mathbf{do}\ \pi(t_3, startScan(x,t_3) : (t_3 - t_1 {<} 5)?))).\end{array}$$

Here the nondeterministic choice between $startPoint$ and $startScan$ is left to the Golog interpreter which has to decide depending on the execution context. Note that, time constraints can be encoded within the procedure itself. In this case the procedure definition leaves few nondetermistic choices to the interpreter. More generally, a Golog script can range from a completely defined procedural program to an abstract general purpose planning algorithm like the following:

$$\begin{array}{l}\mathbf{proc}(plan(n),\\ \qquad true? \mid \pi(a, (primitive\_action(a))? \ : \ a) \ : \ plan(n-1))\end{array}$$

The semantics of a Golog program δ is a situation calculus formula $Do(\delta, s, s')$ meaning that s' is a possible situation reached by δ once executed from the situation s. For example, the meaning of the $a|b$ execution is captured by the logical definition $Do(a|b, s, s') \doteq Do(a, s, s') \vee Do(a, s, s')$.

Flexible behaviours. Our monitoring system is based on a library of Temporal Concurrent Golog scripts representing a set of flexible behaviour fragments. Each of them is associated to a task and can be selected if it is compatible with the execution context. For example a possible behaviour fragment can be written as follows:

$$\begin{array}{l}\mathbf{proc}(explore(d),\\ \ [\pi(t_1, startMap(t_1))\|\pi(t_2, startWand(t_2) :\\ \ \ \pi(t_3, endWand(t_3) : \pi(x, startGoto(x,t_3)) : (t_3 - t_2 {<} d)?))].\end{array}$$

This Golog script is associated with the exploration task, it starts both mapping and wandering activities; the wandering phase has a timeout d, after this the rover has to go somewhere. The timeout d will be provided by the calling process that can be either another Golog procedure or a decision of the operator.

Reactive Planner/Interpreter As illustrated before, for each execution cycle, once the status is updated (sensing phase), the Golog interpreter (planning phase) is called to extend the current control sequence up to the planning horizon. When some task ends or fails, new tasks are selected from the task library and compiled into flexible temporal plans filling the timelines.

Under nominal control, the robot's activities are scheduled according to a closed-loop similar to the LOVR (*Localize, Observe general surroundings, look specially for Victims, Report*) sequence in (Murphy 2004). Some of these activities can require the operator initiative that is always allowed.

Failure detection and management Any system malfunctioning or bad behaviour can be detected by the reactive planner (i.e. the Golog interpreter) when world inconsistencies have to be handled. In this case, after an idle cycle a recovery task has to be selected and compiled w.r.t the new execution status. For each component we have classified a set of relevant failures and appropriate flexible (high-level) recovery behaviours. For example, in the visual model, if the scanning processes fails because of a timeout, in the recovery task the pan-tilt unit must be reset taking into account the constraints imposed by the current system status. This can be defined by a very abstract Golog procedure, e.g.

$$\begin{array}{l}\mathbf{proc}(planToPtuInit,\\ \quad \pi(t, time(t)? : plan(2) : \pi(t_1, PtIdle(0) :\\ \qquad time(t_1)? : (t_1 - t < 3)?))).\end{array}$$

In this case, the Golog interpreter is to find a way to compile this procedure getting the pan-tilt idle in less than two steps and three seconds. The planner/Golog interpreter can fail in its plan generation task raising a *planner timeout*. Since the reactive planner is the engine of our control architecture, this failure is critical. We identified three classes of recoveries depending on the priority level of the execution. If the priority is high, a safe mode has to be immediately reached by means of fast reactive procedures (e.g. $goToStandBy$). In medium priority, some extra time for planning can be obtained by interleaving planning and execution: a greedy action is executed so that the interpreter can use the next timeslot to end its work. In the case of low priority, the failure is handled by replanning: a new task is selected and compiled. In medium and low level priority the operator can be explicitly involved in the decision process in a synchronous way. During a high-priority recovery (i.e. $goToStandBy$) the autonomous control is to manage the emergency, unless the operator wants to take care of it disabling the monitoring system.

Mixed-Initiative Planning

The control architecture introduced before allows us to define some hybrid operative modalities lying between autonomous and teleoperated modes and presenting some capabilities that are crucial in a collaborative planning setting. In particular, following (Allen & Ferguson 2002), our system permits *incremental planning*, *plan stability*, and it is also *open to innovation*.

The high-level agent programming paradigm, associated with the short-range planning/interpretation activity, permits an *incremental* generation of plans. In this way, the user attention can be focused on small parts of the problem and the operator can assess local possible decisions, without losing the overall problem constraints.

Plan stability is guaranteed by flexible behaviours and plan recovery procedures, which can harmonize the modification of plans, due to the operator's interventions or exogenous events. Minimal changes to plans lead to short replanning phases minimizing misalignments.

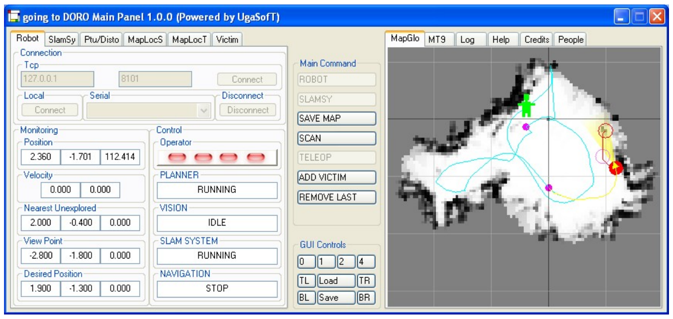
Concerning the *open to innovation* issue, the model-based monitoring activity allows one to build novel plans, under human direction, and to validate and reason about them.

Depending on the operator-system interaction these features are emphasized or obscured. We distinguish among three different mixed-initiative operational modalities.

- **Planning-based interaction.** In this setting, the planning system generates cyclic LOVR sequences and the operator follows this sequence with few modifications, e.g. extending or reducing process durations. Here task dispatching is handled in an automated way and the operator can supervise the decisions consistency minimizing the interventions. The human-operator can also act as an executor and manually control some functional activities scheduled by the planner. For example, he can decide to suspend automated navigations tools and take the control of mobile activities, in this way he can decide to explore an interesting location or escape from difficult environments. In this kind of interaction the operator initiative minimally interferes with the planning activity and *plan stability* is emphasized.
- **Cooperation-based interaction.** In this modality, the operator modifies the control sequence produced by the planner by skipping some tasks or inserting new actions. The operator's interventions can determine a misalignment between the monitoring system expectations (i.e. the control plan) and the state of the system; this is captured at beginning of the next execution cycle when the state monitor provides the current state of the modules. In order to recover the monitor-system adherence, the planner has to start some recovery operations which are presented to the operator. Obviously, these activities are to be executed in real-time by verifying the satisfiability of the underlaying temporal and causal constraints.

  This modality enables maximal flexibility for the planner's and operator's initiatives. Indeed, they can dialogue and work in a concurrent way contributing to the mission completion (*incremental planning*): while the operator tries to modify the plan in order to make it more effective (i.e. the system is *open to innovation*), the monitoring system can validate the operator's choices. Moreover, in the case of safety constraints violations, it warns the user and/or suggests suitable corrections.
- **Operator-based interaction.** This modality is similar to teleoperation, the system activities are directly managed by the operator (some minor autonomy can always be deployed when the operator attention is to be focused on some particular task, e.g. looking for victims). The operator-based interaction is reached when the operators' interventions are very frequent, hence the planner keeps replanning and cannot support the user with a meaningful proactive activity. In this operative scenario, the planner just follows the operators' choices playing in the role of a consistency checker. The monitoring system can notify the user only about safety problems and, in this case, recovery procedures can be suggested (*incremental planning* can be used only to generate non-critical planning procedures).

Each of these modalities is implicitly determined by the way the operator interacts with the system. Indeed, in a mixed-initiative setting, if the operator is idle, the monitor works in the planner-based mode. Instead, the operator's interventions can disturb such a status bringing the system toward the operator-based interaction. However, the operator can always directly set the latter interaction mode by setting to zero the planning horizon and disabling the planner proactive activity. Note that for each mixed-initiative mode, the monitoring system continuously checks the activities performed, including human-operator actions, and when necessary it replans or provides suggestions to the operator.



Connection — Tcp: 127.0.0.1 · 8101

Monitoring
Position: 2.360 · -1.701 · 112.414
Velocity: 0.000 · 0.000
Nearest Unexplored: 2.000 · -0.400 · 0.000
View Point: -2.800 · -1.800 · 0.000
Desired Position: 1.900 · -1.300 · 0.000

Control
PLANNER: RUNNING
VISION: IDLE
SLAM SYSTEM: RUNNING
NAVIGATION: STOP

Figure 4: DORO graphical interface showing the current global map, the victims detected and localized, the path-history: in blue the whole history, and in yellow the most recent one.

Mixed-initiative approach at work

The architecture discussed in this article is implemented on our robotic platform (DORO) and here we present some tests performed in a yellow rescue arenas.

Robotic Platfrom. The hardware platform for DORO is a two wheeled differential drive Pioneer from ActivMedia with an on-board laptop hosts navigation, map building, reactive planning routines and the on-board sensors control processing. An additional PC, for remote control, is also used for image processing. The two PCs running Windows XP are linked with an Ethernet wireless LAN (802.11a) to enable remote control and monitoring of the mobile robot. Two color cameras are mounted on top of the robot on a pant-tilt head. A laser range finder DISTO pro is mounted on the pan-tilt between the two cameras.

Robot Software. The robot motion control (speed and heading) and sonar readings are provided by a serial connection to the Pioneer controller using the Aria API facilities. Video streaming and single frames are acquired through the Image Acquisition Toolbox from Matlab (TM). Inertial data and laser measurements are acquired through dedicated C++ modules that manage the low level serial connections.

Experiences in our domestic arenas. We tested the control architecture and the effectiveness of the mixed-initiative approach in our domestic arenas comparing three possible settings: (i) *fully teleoperated*: navigation, slam, and vision disabled; (ii) *mixed-initiative control*: the monitoring system was enabled and the operator could supervise the rover status and take the control whenever this was needed; (iii) *autonomous control*.

During mixed-initiative control tests, we considered also the percentage of time spent by the operator in *operator-based* mode (see *operator* in the table below). We deployed these three settings on yellow arenas considering increasing surface areas, namely, 20 m^2, 30 m^2, 40 m^2 (see *surface* in the table below), associated with increasingly complex topologies. For each test, there were 4 victims to be discovered. We limited the exploration time to 10 minutes. We performed 10 tests for each modality. Different operators were involved in the experiments in order to avoid an operator visiting the same arena configuration twice.

For each test class we considered: (i) the percentage of the exlored arena surface; (ii) the number of visited and inspected topological environments (rooms, corridors, etc.) w.r.t. the total number; (iii) the overall number of encountered obstacles (i.e. arena bumps); (iv) the number of detected victims; (v) the operator activity (percentage w.r.t. the mission duration). The results are summarized in the Table 1 reporting the average values of each field.

	Fully Teleop			**Supervised**			**Autonomous**		
Surface (m^2)	20	30	40	20	30	40	20	30	40
Explored (%)	85	78	82	85	82	79	49	80	75
Visited env.	5/6	7/9	7/9	6/6	8/9	7/9	3/6	7/9	6/9
Bumps (tot.)	11	7	9	3	2	2	2	1	2
Victims (x/4)	3.0	2.1	2.2	2.5	2.6	2.1	1.3	1.4	1.2
Operator (%)	100	100	100	10	15	15	0	0	0

Table 1: Experimental results for the three operational modalities.

Following the analysis schema in (Scholtz *et al.* 2004) here we discuss the following points: *global navigation*, *local navigation and obstacle encountered*, *vehicle state*, *victim identification*.

Concerning *global navigation*, the performance of the mixed-initiative setting are quite stable while the autonomous system performs poorly in small arenas because narrow environments challenge the navigation system which is to find how to escape from them. In greater and more complex arenas the functional navigation processes (path planner, nearest unexplored point system, etc.) start to be effective while the fully teleoperated behaviour degrades: the operator gets disoriented and often happens that already visited locations and victims are considered as new ones, while we never experienced this in the mixed-initiative and autonomous modes. The effectiveness of the control system for *local navigation* and *vehicle state* awareness can be read on the *bumps* row; indeed the bumps are significantly reduced enabling the monitoring system. In particular, we experienced the recovery procedures effectiveness in warning the operator about the vehicle attitude. E.g. a typical source of bumping in teleoperation is the following: the visual scanning process is interrupted (timeout) and the operator decides to go on in one direction forgetting the pan-tilt in a non-idle position. Enabling the monitor, a recovery procedure interacts with the operator suggesting to reset the pan-tilt position. The victim identification effectiveness can be assessed considering the founded victims in the autonomous mode; considering that visual processing was deployed without any supervision, these results seem quite good (we experienced some rare false-positive).

Our experimental results show that the system performances are enhanced with the presence of an operator supervising the mission. It seems that the autonomous activities are safely performed, but the operator can choose more effective solutions in critical situations. For instance, the number of visited environments in supervised mode (see Table 1) is greater than that one in the autonomous mode, while the victims detected are approximately the same. Furthermore, the number of bumps in teleoperated mode is greater than in both supervised and autonomous settings, and this can be explained by the cognitive workload on the operator during the teleoperation. Thus, we can trade off high performances and low risks by exploiting both human supervision and machine control .

Conclusion

Human-robot interaction and situation awareness are crucial issues in a rescue environment. In this context a suitable interplay between supervised autonomy and full autonomy is needed. For this purpose, we designed a control system where the HRI is fully based on a mixed-initiative planning activity which is to continuously coordinate, integrate, and monitor the operator interventions and decisions with respect to the concurrent functional activities. Our approach integrates model-based executive control, flexible interval planning and high level agent programming.

This control architecture allows us to define some hybrid operative modalities lying between *teleoperated mode* and *autonomous mode* and presenting some capabilities that are crucial in a collaborative planning setting.

We implemented our architecture on our robotic platform (DORO) and tested it in a NIST yellow arena. The comparison between three possible settings (*fully teleoperated*, *mixed-initiative control*, *autonomous control*) produce encouranging results.

References

Ai-Chang, M.; Bresina, J.; Charest, L.; Chase, A.; Hsu, J.-J.; Jonsson, A.; Kanefsky, B.; Morris, P.; Rajan, K.; Yglesias, J.; Chafin, B.; Dias, W.; and Maldague, P. 2004. Mapgen: mixed-initiative planning and scheduling for the mars exploration rover mission. *Intelligent Systems, IEEE* 19(1):8– 12.

Allen, J., and Ferguson, G. 2002. Human-machine collaborative planning. In *Proceedings of the 3rd international NASA Workshop on Planning and Scheduling for Space*.

Bruemmer, D. J.; Boring, R. L.; Few, D. A.; Marble, J. L.; and Walton, M. C. 2003. "i call shotgun!": An evaluation of mixed-initiative control for novice users of a search and rescue robot. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*.

Burke, J.; Murphy, R.; Coovert, M.; ; and Riddle, D. 2004. Moonlight in miami: A field study of human-robot interaction in the context of an urban search and rescue disaster response training exercise. *Special Issue of Human-Computer Interaction* 19(1,2):21–38.

Burstein, M., and McDermott, D. 1996. Issues in the development of human-computer mixed-initiative planning. *Cognitive Technology* 285–303. Elsevier.

Dautenhahn, K., and Werry, I. 2000. Issues of robot-human interaction dynamics in the rehabilitation of children with autism.

Drury, J. L.; Scholtz, J.; and Yanco, H. A. 2003. Awareness in human-robot interaction. In *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*.

Finzi, A., and Pirri, F. 2004. Flexible interval planning in concurrent temporal golog. In *Working notes of the 4th International Cognitive Robotics Workshop*.

Finzi, A.; Ingrand, F.; and Muscettola, N. 2004. Model-based executive control through reactive planning for autonomous rovers. In *Proceedings IROS-2004*, 879–884.

Jacoff, A.; Messina, E.; and Evans, J. 2001. A reference test course for urban search and rescue robots. In *FLAIRS Conference 2001*, 499–503.

Kiesler, S., and Hinds, P. 2004. Introduction to the special issue on human-robot interaction. *Special Issue of Human-Computer Interaction* 19(1,2):1–8.

Lang, S.; Kleinehagenbrock, M.; Hohenner, S.; Fritsch, J.; Fink, G. A.; and Sagerer, G. 2003. Providing the basis for human-robot-interaction: a multi-modal attention system for a mobile robot. In *Proceedings of the 5th international conference on Multimodal interfaces*, 28–35. ACM Press.

Maxwell, B. A.; Smart, W. D.; Jacoff, A.; Casper, J.; Weiss, B.; Scholtz, J.; Yanco, H. A.; Micire, M.; Stroupe, A. W.; Stormont, D. P.; and Lauwers, T. 2004. 2003 aaai robot competition and exhibition. *AI Magazine* 25(2):68–80.

McCarthy, J. 1963. Situations, actions and causal laws. Technical report, Stanford University. Reprinted in Semantic Information Processing (M. Minsky ed.), MIT Press, Cambridge, Mass., 1968, pp. 410-417.

Michael Baker, Robert Casey, B. K., and Yanco, H. A. 2004. Improved interfaces for human-robot interaction in urban search and rescue. In *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*. "To appear".

Murphy, R. 2004. Human-robot interaction in rescue robotics. *IEEE Transactions on Systems, Man and Cybernetics, Part C* 34(2):138–153.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence* 103(1-2):5–47.

Muscettola, N.; Dorais, G. A.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. Idea: Planning at the core of autonomous reactive agents. In *Proc. of NASA Workshop on Planning and Scheduling for Space*.

Myers, K. L.; Jarvis, P. A.; Tyson, W. M.; and Wolverton, M. J. 2003. A mixed-initiative framework for robust plan sketching. In *Proceedings of the 2003 International Conference on Automated Planning and Scheduling*.

Pinto, J., and Reiter, R. 1995. Reasoning about time in the situation calculus. *Annals of Mathematics and Artificial Intelligence* 14(2-4):251–268.

Pirri, F., and Reiter, R. 2000. Planning with natural actions in the situation calculus. *Logic-based artificial intelligence* 213–231.

Reiter, R. 1996. Natural actions, concurrency and continuous time in the situation calculus. In *Proceedings of KR'96*, 2–13.

Reiter, R. 2001. *Knowledge in action : logical foundations for specifying and implementing dynamical systems*. MIT Press.

Scholtz, J.; Young, J.; Drury, J.; and Yanco, H. 2004. Evaluation of human-robot interaction awareness in search and rescue. In *Proceedings of the 2004 International Conference on Robotics and Automation*.

Sidner, C., and Dzikovska, M. 2002. Human-robot interaction: Engagement between humans and robots for hosting activities. In *The Fourth IEEE International Conference on Multi-modal Interfaces*, 123–128.

Tadokoro, S.; Kitano, H.; Takahashi, T.; Noda, I.; Matsubara, H.; Shinjoh, A.; Koto, T.; Takeuchi, I.; Takahashi, H.; Matsuno, F.; Hatayama, M.; Nobe, J.; and Shimada, S. 2000. The robocup-rescue project: A robotic approach to the disaster mitigation problem. In *ICRA-2000*, 4089–95.

Tadokoro, S. 2000. Robocuprescue robot league. In *RoboCup-2002*, 482–484.

Volpe, R.; Nesnas, I.; Estlin, T.; Mutz, D.; Petras, R.; and Das, H. 2001. The claraty architecture for robotic autonomy. In *IEEE 2001 Aerospace Conference*.

Williams, B.; Ingham, M.; Chung, S.; Elliott, P.; Hofbaur, M.; and Sullivan, G. 2003. Model-based programming of fault-aware systems. *AI Magazine*.

Yanco, H., and Drury, J. 2002. A taxonomy for human-robot interaction. In *Proc. AAAI Fall Symposium on Human-Robot Interaction*, 111–119.

Metatheoretic Plan Summarization and Comparison

Karen L. Myers
AI Center, SRI International
333 Ravenswood Ave.
Menlo Park, California 94025
myers@ai.sri.com

Abstract

We describe a domain-independent framework for plan summarization and comparison that can help a human understand both the key elements of an individual plan and important differences among plans. Our approach is grounded in the use of a *domain metatheory*, which is an abstract characterization of a planning domain that specifies important semantic properties of templates, planning variables, and instances. The metatheory provides a semantic framework for guiding the choice and description of concepts used in summarizing and comparing plans, thus enabling results that are grounded in semantically significant concepts rather than syntactic constructs whose meaning or import is unclear. We define three specific capabilities grounded in the metatheoretic approach: (a) summarization of an individual plan, (b) comparison of pairs of plans, and (c) analysis of a collection of plans. Use of these capabilities within a rich application domain shows their value in facilitating the understandability of complex plans by a user.

Introduction

AI planning technology is being applied in increasingly more challenging application domains, resulting in the generation of plans with rich sophistication and complexity. In these complex domains, it is generally the case that a wide range of solutions is possible; part of the challenge for a human decision maker is to analyze the relative merits of various candidates before deciding on a final option. Given these advances, the development of tools that can help users understand complex plans and tradeoffs among them presents an important technological challenge.

In this paper, we describe an approach to plan summarization and comparison that is designed to help a human understand both the key elements of an individual plan and important differences among alternative plans. Our approach is grounded in the use of a *domain metatheory*. The domain metatheory is an abstract characterization of a planning domain that specifies important semantic properties of templates, planning variables, and instances. The abstraction provides the means to describe and compare plans in high-level, semantically meaningful terms.

Previous work on plan summarization and explanation has been grounded in methods that are tightly linked to either the syntactic characteristics of a plan's structure or the underlying reasoning processes used to generate it. Such approaches suffer from the problem that these structures and processes match the system's conceptualization of the domain rather than that of the user. As such, their outputs have limited explanatory value.

The concept of the domain metatheory was introduced originally to provide a language that would enable a user to *advise* a planning system, without requiring detailed knowledge of its internal workings [Myers 1996]. Advice, which describes high-level characteristics of desired solutions, is operationalized into structures and mechanisms that guide an automated planning system at runtime. Subsequently, the metatheory was also used as the basis for generating *qualitatively different* plans, by using structure within the metatheory to direct a planning system toward solutions with distinct semantic traits [Myers & Lee 1999].

A key insight underlying the work reported here is that the metatheory can be used as the basis for identifying and communicating important explanatory information about a plan. In particular, the metatheory provides a semantic framework for guiding the choice of concepts used in summarizing and comparing plans. The resultant comparisons and summaries are thus grounded in semantically significant concepts rather than syntactic constructs whose meaning or import are unclear.

Within our metatheoretic framework, we define techniques for (a) summarization of an individual plan, (b) comparison of pairs of plans, and (c) analysis of a collection of plans. These techniques look for regularities or interesting exceptions relative to key aspects of the domain metatheory. For example, a metatheory *role* corresponds to an important actor or object within a plan. In comparing two plans, one interesting dimension to consider is whether the plans fill key roles in different ways. Two plans may be similar in structure but one uses a cheap and abundant resource while the other relies on an expensive and more exotic resource.

Our approach embodies the spirit of *reconstructive explanation* [Wick and Thompson 1992], whereby an explanation is produced not by the system's own internal knowledge, but by a separate store of explanatory knowledge designed specifically with the user in mind. We believe that this style of approach is critical to ensuring that the results are of value to a user, rather than driven by the syntactic structure of the plan.

The plan summarization and comparison methods have been implemented within the PASSAT mixed-initiative planning framework [Myers et al. 2002]. To assess their effectiveness in facilitating user understandability of complex plans, we applied the methods to a test suite from an extensive special operations domain. This usage shows that our techniques can help a user understand subtle aspects of individual plans, important differences among plans, and the structure of the overall solution space.

Domain Metatheory

Our plan summarization and comparison work assumes a *hierarchical task network* (HTN) paradigm for representing plans, similar to that described in [Erol et al. 1994]. An HTN domain theory consists of four basic types of element: *individuals* corresponding to real or abstract objects in the domain, *relations* that describe characteristics of the world, *tasks* to be achieved, and *templates* that describe available means for achieving tasks. (Templates are alternatively referred to as *methods* or *operators* in the literature.) We assume a type hierarchy for terms within the domain model. Thus, each individual has an associated type *Type(v)*, and there is a unique most-specific supertype *MinSupertype(V)* defined for any set of individuals *V*.

A *domain metatheory* defines semantic properties for domain theory elements that abstract from the syntactic details of the domain knowledge. The metatheory for plan summarization and comparison is similar to that introduced for the work on advisable planning. To support summarization and comparison, however, we introduce a few extensions and refinements that provide a somewhat richer and more structured framework. The main metatheoretic concepts that we use are *template features*, *task features*, and *roles*.

Template Features

A *template feature* designates a characteristic of a template that distinguishes it from other templates that could be applied to the same task. For example, among templates that could be applied to a transportation task, there may be an air-based template that is `fast` but `expensive` with a land-based alternative that is `slow` but `cheap`. Although the two templates are functionally equivalent in that they accomplish the same task, they differ significantly in their approaches. Template features provide the means to distinguish among such functionally equivalent alternatives by capturing these characteristics explicitly.

We model template features in terms of a *feature category* (e.g., `COST`) and a *feature value* (e.g., `expensive`). Feature values are drawn from a predefined set that constitutes the *domain* of the feature category. For this work, we require that the domain for a template feature be totally ordered (that need not be true in general).

We say that a template feature *f* with value *v occurs in plan P* iff there is some template *T* applied to a task *t* in *P* such that *T* has the feature *f* with value *v*. In general, a plan may have multiple occurrences of a given template feature that cut across templates used to accomplish a range of tasks. Different occurrences may have different values associated with them; duplication of values is also possible. The term *TemplFeatureInsts(f,P)* denotes the collection of values (including duplicates) for occurrences of template feature *f* in plan *P*.

The value of template features for plan summarization and comparison is that they provide the means to identify, abstract, and contrast important evaluational properties of different strategies, such as speed or cost. In particular, template features can be used as a kind of ‘quick and dirty’ proxy for deeper, more significant evaluations of a plan.

Task Features

Task features capture important semantic attributes of a task. As with template features, task features are modeled in terms of a feature category and feature value. Here, we focus on task features that designate *types* of activities, and restrict categories to have the domain `[false true]`. For example, there may be several types of reconnaissance task: satellite reconnaissance, ground reconnaissance, and aircraft reconnaissance. Each of these tasks can be assigned the feature `RECON` with value `true`, thus providing a mechanism for abstracting over that set of tasks. (A similar sort of grouping could be achieved through the use of a class hierarchy for tasks.)

We say that a plan *P has a task feature f* iff some task *t* in *P* has the feature *f* with value `true`. The term *TaskFeatures(P)* denotes the set of task features for *P*.

Roles

A *role* describes a capacity in which an individual is used within a template or task; it maps to a template or task variable. For instance, a template for transporting materials may contain variables `location.1` and `location.2`, with the former corresponding to the `START` role and the latter the `DESTINATION` role for the move. Roles provide a semantic basis for describing the use of individuals within templates and tasks that abstracts from the details of specific variable names. Roles also provide the means to reference a collection of semantically linked variables that span different templates and tasks (i.e., `START` roles may occur in multiple templates and tasks).

We say that a *role r with fill v occurs in plan P* iff either:

- there is some task $t(a_1, \ldots a_n)$ in *P* such that *t* has the declared role *r* for its i^{th} argument, and $a_i = v$, [Task Role] or
- some template *T* with role *r* declared for local variable x_i is applied to a task $t(a_1, \ldots a_n)$ in *P*, and x_i is bound to *v* [Template Role]

The term *Roles(P)* denotes the set of roles that occur in plan *P*, while *RoleFills(r,P)* denotes the collection of values (including duplicates) that occur as fills for role *r* in plan *P*.

Experimental Framework

We evaluated the effectiveness of our plan summarization and comparison techniques on a suite of nine test plans drawn from a *special operations forces* (SOF) domain. (This domain was created as part of an earlier project focused on mixed-initiative planning technology.) The SOF domain constitutes a sizable and rich test environment for evaluating our work on plan summarization and comparison: the base-level domain contains 65 predicates modeling key world properties, more than 100 tasks, and more than 50 templates spanning a hierarchy of five abstraction layers.

The original SOF domain included a limited metatheory designed to showcase advice-taking within the PASSAT system [Myers et al. 2002]. For this work, we extended the domain to include a fairly comprehensive metatheory with 13 template features, 12 task features, and more than 75 roles. The task features (see Figure 3) use the domain `[false true]`; the template features (see Figure 4) use the domain `[low medium high]`.

The test plans address the high-level task of extracting a set of hostages held by a guerilla team in an urban environment. More specifically, this task requires rescuing a set of hostages being kept at Mogadishu-Town-Hall using forces based at Riyadh Airport, and then evacuating the hostages to a safe haven at Riyadh Stadium.

The SOF domain includes a number of templates that reflect different strategies for rescuing the hostages. Variations among solutions result from three sources. The first is whether the plan contains certain types of strategic and tactical activities; depending on a given situation, the commander may or may not decide to include such activities within the plan. For example, while it is not necessary to create diversions to distract the guerillas, doing so may be desirable in some circumstances. The second relates to the selection of resources to be used. In some cases, for example, it may be appropriate to use satellites to gather intelligence information while in others it may be preferable to rely on ground forces. The third relates to decisions about key parameters within a plan, such as where to establish a forward base or the drop point for inserting the assault team.

Figure 1 summarizes the nine test plans used in our evaluation. These plans were created by the developer of the SOF domain knowledge, through a combination of manual and semiautomated methods within PASSAT.[1] The plan developer was asked to create a core set of plans reflecting a representative set of strategic alternatives that a SOF commander might consider. Additionally, he was asked to create variants of the core plans by making a few key strategic changes that might correspond to handling contingencies in different ways. Given that variants of this type are commonly made in practice, we were interested in determining how well our plan comparison techniques would be able to recognize the differences among them.

[1] The plan developer was not involved with the research on plan summarization and comparison described in this paper. As such, the plans provide an objective test suite for evaluating the reported work.

Plan Identifier	Description
tiny-plan-a	Very simple plan without security or support
tiny-plan-b	Variant on *tiny-plan-a* that uses a different type of rescue force
small-plan-a	Basic solution that includes reconnaissance and combat search and rescue
small-plan-b	Variant on *small-plan-a* that uses the same high-level strategy but differs in the lower-level realization of parts of it
medium-plan-a	Broadly similar to the small plans but involves refueling
medium-plan-b	Broadly similar to the small plans but with suppression of enemy air defenses (SEAD) activities
large-plan-a	Extensive plan with significant reconnaissance and support activities as well as a diversion from the main assault
large-plan-b	Variant on *large-plan-a* that provides increased fire support and SEAD
large-plan-c	Variant on *large-plan-a* with a different style of diversion

Figure 1. Summary of Test Plans

Plan Summarization

The roles and features of the metatheory provide a semantic basis for summarizing key properties of a plan. In particular, a description of how a plan fills its roles and the features that it possesses can provide valuable insight into the structure, strengths, and weaknesses of a plan.

Task Features

Task features provide a succinct summary of key activity types within a plan. In particular, such a summary can inform the user that a given plan does or does not contain critical activities such as reconnaissance or fire support.

Template Features

Template features provide a different perspective on a plan, as they designate plan characteristics that have more of an evaluational nature (e.g., cost, speed). Template features can be applied in multiple contexts within a plan, with different occurrences yielding different values. This variation reflects the fact that, for example, a given plan may use an inexpensive reconnaissance operation but an expensive rescue strategy. To enable plan-level summarization of the property represented by a template feature, we introduce the concept of *template feature value* for a feature *f* and plan *P*, denoted by *TemplFeatureValue(f,P)*. This value is defined to be the average of the values for all occurrences of *f* within *P*.

Definition 1 [Template Feature Value for a Plan] The *template feature value* for feature *f* and plan *P* is defined by *TemplFeatureValue(f,P)=Avg (TemplFeatureInsts(f,P))*.

The use of a qualitative domain for template features (as in the SOF application) introduces a complication in computing *TemplFeatureValue(f,P)*, as it is necessary to support qualitative averaging. To this end, we require for each qualitative feature *f* a surjective, order-preserving mapping θ_f from a designated interval of the reals *Interval*(*f*) to the domain of the feature *f*: θ_f: *Interval*(*f*) → *Domain(f)*. Variation in the 'closeness' of values in *Domain(f)* can be achieved by appropriate definitions of θ_f.[2] With this mapping, we define the average of a set *V* of qualitative template feature values as follows:

$$Avg(V) = \theta_f\left(\frac{\sum_{v \in V} \theta_f^{-1}(v)}{|V|}\right)$$

Roles

A description of how roles are filled within a plan can provide a concise summary of what resources are used and how, as well as key parameters to a plan (e.g., the choice of location for a forward base). Furthermore, it is possible to search for patterns in the filling of roles. So, for example, it may be useful to know that only satellites are used as reconnaissance assets, or that all transport of troops is through the use of helicopters of a particular type. We refer to such patterns as *uniformities* in the filling of roles. Here, we define two specific types of uniformity for role fills, oriented around *values* and *types*.

Definition 2 [Value Uniformity in Role Fills] A plan *P uniformly fills a role r with value c* iff |*RoleFills(r,P)*| > 1 and *v* ∈*RoleFills(r,P)* implies that *v =c*.

Type uniformity depends on the declaration of a type *Type(r)* for a given role *r*, which indicates that all fills for role *r* must be of that type. Type uniformity becomes interesting when some proper subtype of *Type(r)* generalizes all fills for a given role. For example, it can be useful to note that only satellites are used for reconnaissance within a given plan, although other types of assets (e.g., ground forces) are possible.

Definition 3 [Type Uniformity in Role Fills] A plan *P uniformly fills a role r with a type T* iff |*RoleFills(r,P)*| > 1, *T* is a proper subtype of *Type(r)*, and every fill value *v*∈*RoleFills(r,P)* is of type *T*.

Value and type uniformity for roles constitute generic, domain-independent mechanisms for generalizing a collection of role fills. For a given domain, it may be appropriate to introduce domain-specific generalization mechanisms. For example, in domains where locations play a significant role, it might be useful to generalize based on geographic proximity, or co-location within some designated geographic area (e.g., all air assets are pulled from bases in the same region).

```
* (Rescue-Hostage Mogadishu-Town-Hall Riyadh-Airport Riyadh-Stadium)
   * (Rescue-And-Recover Riyadh-Airport Mogadishu-Town-Hall Riyadh-Stadium)
      * (Recon Mogadishu-Town-Hall)
         * (Infiltrate Green-Oda-2 Ankara-Airport Mogadishu-Town-Hall)
            * (Produce-Landing-Plan Mh-60-G-Pave-Hawk-2)
            * (Produce-Air-Movement-Plan Mh-60-G-Pave-Hawk-2)
            * (Produce-Loading-Plan Green-Oda-2)
            * (Produce-Aircraft-Bump-Plan Green-Oda-2)
            * (Load Green-Oda-2 Mh-60-G-Pave-Hawk-2)
            * (Fly Mh-60-G-Pave-Hawk-2 Ankara-Airport Mogadishu-Stadium)
            * (Drop Green-Oda-2 Mh-60-G-Pave-Hawk-2 Mogadishu-Town-Hall)
            * (Depart Mh-60-G-Pave-Hawk-2 Mogadishu-Stadium)
         * (Establish-Observation-Post Green-Oda-2 Mogadishu-Town-Hall)
         * (Exfiltrate Green-Oda-2 Mogadishu-Town-Hall Mogadishu-Building4)
            * (Fly Uh-60a-2 Mogadishu-Town-Hall Mogadishu-Building3)
            * (Load Green-Oda-2 Uh-60a-2)
            * (Depart Uh-60a-2 Mogadishu-Building3)
         * (Provide-Fire-Support Mogadishu-Town-Hall)
            * (Take-Off Ch-53e-Super-Stallion-1 Addis-Ababa-Airport)
            * (Fly Ch-53e-Super-Stallion-1 Addis-Ababa-Airport Mogadishu-Town-Ha
            * (Place-On-Station-Fire-Support Ch-53e-Super-Stallion-1 Mogadishu-T
            * (Fly Ch-53e-Super-Stallion-1 Mogadishu-Town-Hall Addis-Ababa-Airpo
            * (Land-At Ch-53e-Super-Stallion-1 Addis-Ababa-Airport)
         * (Provide-Csar-Coverage Csar-Team-2 Mogadishu-Town-Hall)
            * (Prepare Csar-C1-A)
            * (Take-Off Csar-C1-A Baidoa-Stadium)
            * (Fly Csar-C1-A Baidoa-Stadium Mogadishu-Town-Hall)
            * (On-Station Csar-C1-A Mogadishu-Town-Hall)
            * (Provide-Fire-Support Mogadishu-Town-Hall)
               * (Take-Off Ah-100-1 Balikesir-Stadium)
               * (Fly Ah-100-1 Balikesir-Stadium Mogadishu-Town-Hall)
               * (Place-On-Station-Fire-Support Ah-100-1 Mogadishu-Town-Hall)
               * (Fly Ah-100-1 Mogadishu-Town-Hall Balikesir-Stadium)
               * (Land-At Ah-100-1 Balikesir-Stadium)
            * (Provide-Sead Sead-1 Ad-Dammam-Stadium Mogadishu-Town-Hall)
               * (Prepare Sead-1)
               * (Take-Off Sead-1 Ad-Dammam-Stadium)
               * (Fly-To Sead-1 Mogadishu-Town-Hall)
      * (Infiltrate Orange-Oda-1 Riyadh-Airport Mogadishu-Town-Hall)
         * (Fly-Commercial Aa7864 Orange-Oda-1 Riyadh-Airport Mogadishu-Town-Hal
      * (Storm Orange-Oda-1 Mogadishu-Town-Hall)
      * (Exfiltrate Orange-Oda-1 Mogadishu-Town-Hall Riyadh-Stadium)
         * (Fly-Commercial Aa201 Orange-Oda-1 Mogadishu-Town-Hall Riyadh-Stadium
      * (Provide-Fire-Support Mogadishu-Town-Hall) [Fire-Support-Naval]
         * (Station Yorktown Mogadishu-Town-Hall)
      * (Provide-Csar-Coverage Csar-Team-2 Mogadishu-Town-Hall)
         * (Prepare Uh-60l-1)
         * (Take-Off Uh-60l-1 Bihac-Stadium)
         * (Fly Uh-60l-1 Bihac-Stadium Mogadishu-Town-Hall)
         * (On-Station Uh-60l-1 Mogadishu-Town-Hall)
```

Figure 2. Task Decomposition View of Plan *medium-plan-b*

Sample Plan Summary

To illustrate the value of metatheory-based plan summarization, consider the summary of the test plan *medium-plan-b* shown in Figure 2. The figure presents a task decomposition view of the plan that highlights its hierarchical structure; for simplicity, temporal sequencing information among activities has been omitted.

As can be seen, the plan is sufficiently complex that its key strategic elements are not readily apparent. Rather, some form of analysis tool is required to understand the plan. Figure 3 summarizes the task features within this plan while Figure 4 summarizes the template features and their normalized values. Figure 5 summarizes key role fills for the plan.

The summary of task features in Figure 3 makes it easy to identify the key strategic elements of the plan. The features `RESCUE-AND-RECOVER` and `RESCUE` derive from the fact that the plan describes a rescue-and-recover operation; these features are common to every plan in the test suite. At a lower level, we can see that this particular solution includes components for combat search and rescue support (`CSAR-SUPPORT`), fire support (`FIRE-SUPPORT`), reconnaissance (`RECON`), and suppression of

[2] For the SOF metatheory, every template feature has the domain [low medium high], the interval [0,1], and the mapping function θ: [0,1] → [low medium high] where θ^1 is distributed linearly across [0,1]: low maps to 0, medium to 0.5 and high to 1.

enemy air defenses (SEAD). These components are optional, as not every solution contains them.

Task Feature	Value
CSAR-SUPPORT	true
DIVERSION	false
EVACUATION	false
FIRE-SUPPORT	true
PARACHUTE	false
RECON	true
REFUELING	false
RESCUE	true
RESCUE-AND-RECOVER	true
SEAD	true
SECURITY	false
SUPPORT	true

Figure 3. Task Features for Plan *medium-plan-b*

Template Feature	Plan Value
BLUE-CASUALTY-RISK	medium
COLLATERAL-DAMAGE	low
COORDINATION-COMPLEXITY	medium
COVERTNESS	medium
DURABILITY	low
FORCE-FATIGUE	medium
FORCE-FOOTPRINT	medium
FORCE-INTEGRITY	medium
INFORMATION-QUALITY	high
LANDING-ZONE-PREP	low
ROBUSTNESS	medium
SPEED	medium
VULNERABILITY-GROUND-FIRE	high

Figure 4. Template Features for Plan *medium-plan-b*

Role	Fill Values
ASSAULT-FORCE	green-oda-2 orange-oda-1
FORCE	orange-oda-1 (2) green-oda-2 (2)
OBSERVATION-FORCE	green-oda-2

Roles related to Strategic Decisions about Locations

Figure 5. Force Usage Roles in *medium-plan-b*

Value-based Role Uniformity

Role	Value	Count
CSAR-LOCATION	mogadishu-town-hall	2
FIRE-SUPPORT-LOCATION	mogadishu-town-hall	5
FORWARD-POINT	riyadh-airport	3
INFIL-DESTINATION	mogadishu-town-hall	2
SEAD-AIRCRAFT	sead-1	2

Type-based Role Uniformity

Role	Role Type	Fill Types	Min. SuperType
EXFIL-ASSET	Asset	Commercial-flight Helicopter	Air-asset
INFIL-ASSET	Asset	Commercial-flight Helicopter	Air-asset
TRANSPORT-ASSET	Asset	Sead-aircraft Helicopter	Air-asset

Figure 6. Role Uniformities in Plan *medium-plan-b*

The template features in Figure 4 summarize key evaluational qualities of the plan. Desirable qualities include the fact that the expected quality of information underlying the plan is high, while expected collateral damage is low. On the negative side, there is high vulnerability to ground fire.

More than 30 roles occur in the plan *medium-plan-b*, some of which have multiple fills. Typically, a user would not choose to view all roles and their fills at once. Rather, at a given point in time he would be interested in knowing about a subset of these roles as he focuses on certain aspects of the plan. So, for example, a user interested in understanding the high-level strategy of a plan may concentrate on a subset of roles related to key strategic decisions, while a user interested in asset usage may concentrate on roles related to resource utilization.

Figure 5 displays the role fills related to force usage for the plan *medium-plan-b*. For fill values that occurred more than once for a given role, the number of occurrences is noted in parentheses. This summary makes it easy to see that only Green and Orange teams are used in the plan; both are used in assault roles while the Green team is also used in a reconnaissance capacity as an observation force.[3]

Figure 6 summarizes value-based and type-based role uniformities for the plan *medium-plan-b*. For value-based uniformity, the summary indicates the role, the fill value, and the number of occurrences. For type-based uniformity, the summary indicates the role, its type, the types of the fill values, and the most specific type that generalizes the fill values. The information on type-based uniformity is particularly useful here as it highlights the exclusive use of air assets for many key functional roles within the plan.

Plan Comparison

Our approach to comparing plans is grounded in two techniques: *feature differencing* and *role differencing*. These techniques can be useful both in terms of identifying subtle variations in similar plans, and understanding larger differences in more varied plans.

Feature Differencing

As noted above, features correspond to high-level semantic characteristics of tasks (for task features) and strategic or evaluational qualities (for template features).

Task features provide a semantic summary of key activities within a plan. Task feature differencing, which involves a comparison of task features within two plans, provides a snapshot of how the two plans differ in their key task types. This type of capability can enable a user to see easily that, for example, one plan contains reconnaissance capabilities while another does not.

[3] The color in a force name is significant: colors denote units with specific skills and capabilities. For the sake of brevity, we omit detailed descriptions of the qualities associated with the various force colors.

Template feature differencing compares the normalized template feature values for two different plans in order to identify significant variations. This form of differencing makes it easy to see, for example, that one plan trades risk for increased complexity relative to another plan.

Role Differencing

Role differencing looks at variabilities in how two plans fill their roles. This type of comparison can shed insight on key differences in strategic decisions (e.g., *Where are the hostages to assemble?*) and resource usage (e.g., *What types of reconnaissance asset are used?*).

Figure 7 presents a categorization of the ways in which the fill values for a given role in two plans can differ. There, V_1 and V_2 designate sets of fill values for a role from which duplicates have been removed. It is assumed that $V_1 \neq V_2$ and that both V_1 and V_2 are nonempty. The first three entries cover situations where V_1 and V_2 are disjoint; the last two cover situations where V_1 and V_2 overlap.

The category *different single valued,* although just a special case of *disjoint types*, is useful for identifying differences in key strategic decisions for a plan. For example, for the role `ASSAULT-FORCE`, the plan *tiny-plan-a* uses `orange-oda-2` while the plan *tiny-plan-b* uses `green-oda-1`. This difference is important, as noted above, because orange and green forces have significantly different core capabilities. The category *different single valued* is especially useful when the role appears exactly once within each of the two plans being compared; such a role often designates some critical parameter choice.

The category *disjoint types* requires both that the most specific supertype of the role-fill values in the two plans be different, and that neither be a subtype of the other. As an example, the plan *small-plan-a* uses only helicopters of type `CSAR-HELICOPTER-CLASS-1` for combat search and rescue while the plan *large-plan-b* uses helicopters of type `CSAR-HELICOPTER-CLASS-2`. The category *disjoint multivalued* defines an even weaker condition, requiring only that the fill values for the two plans be different.

For overlapping values, the strongest condition is *restricted subtype*, which indicates that the most specific supertype of one collection of values is a subtype of the most specific supertype of the other collection. For example, the plan *large-plan-b* uses only assault forces of type `SOF-UNIT` while the plan *large-plan-c* uses a more general set of forces (of type `FORCE-COMPOSITION`); in contrast, the plan *large-plan-b* uses a range of watercraft to fill the role `WATER-ASSET` while the plan *large-plan-c* uses only values of type `BOAT`. *Restricted subset* weakens the *restricted subtype* condition to require only that one collection of values be a subset of the other.

Role differencing can provide insights into fundamental differences between plans, as illustrated in the next section. However, there are limitations to its usefulness.

First, the significance of role differences may be difficult to gauge in isolation. So, while the decision to use force `Green-ODA-1` rather than `Orange-ODA-2` to fill the `ASSAULT-FORCE` role is significant, as those two units have markedly different capabilities, the difference between the forces `Green-ODA-1` and `Green-ODA-2` is insignificant as they have the same fundamental capabilities. This problem can be addressed by introducing a notion of 'semantic distance' between individuals to help identify differences that are significant.

Second, the utility of role differencing can decrease as plan size grows due to increased numbers of occurrences of a role that are not closely related. (For example, it is possible to create larger SOF plans by introducing multiple assault prongs involving forces inserted at different drop locations; doing so leads to duplication of roles used in very different contexts.) Thus, while unrestricted role differencing can be useful in small- to medium-sized plans, larger plans would benefit from some scheme to contextualize role fills to certain portions of the plan.

Disjoint: $V_1 \cap V_2 = \{\}$

Different single valued: $V_1 \cap V_2 = \{\} \wedge |V_1|=|V_2|=1$

Disjoint types: $\text{MinSupertype}(V_1) \neq \text{MinSupertype}(V_2)$
$\wedge\ \text{MinSupertype}(V_1) \not\subset \text{MinSupertype}(V_2)$
$\wedge\ \text{MinSupertype}(V_2) \not\subset \text{MinSupertype}(V_1)$

Disjoint multivalued: $V_1 \cap V_2 = \{\} \wedge (|V_1|>1 \vee |V_2|>1)$

Overlapping: $V_1 \cap V_2 \neq \{\}$

Restricted subtype: $\text{MinSupertype}(V_1) \subset \text{MinSupertype}(V_2)$
$\vee\ \text{MinSupertype}(V_2) \subset \text{MinSupertype}(V_1)$

Restricted subset: $V_1 \subset V_2$

Figure 7. Categories of Role-fill Differences

Sample Plan Comparison

Figure 8 displays the results of applying our metatheoretic plan comparison techniques to the test plans *medium-plan-a* and *medium-plan-b*.

In looking at the results of task feature differencing, two fundamental differences emerge: *medium-plan-a* contains refueling activities and *medium-plan-b* does not, while *medium-plan-b* contains SEAD (suppression of enemy air defense) activities and *medium-plan-a* does not.

For template feature differencing, there is some variation among expected values for key evaluation criteria. Given the use of a fairly coarse-grained set of qualitative values for template feature domains in the SOF metatheory, the scope for variability is limited. A more fine-grained set of values would enable more precise comparisons.

Role differencing highlights some interesting variations in the use of resources between the two plans. Both plans include reconnaissance operations, but *medium-plan-a* relies on a satellite (`satellite-1`) while *medium-plan-b* makes use of a ground force (`green-oda-2`) as the asset used to perform the reconnaissance (see the table *Different Single Valued*). This distinction is important because the nature and quality of the intelligence that can be obtained with these two assets is markedly different. Different types of infiltration, exfiltration, fire support and transport

Task Feature Differencing:

Task Features in *medium-plan-a* but not in *medium-plan-b*: `REFUELING`

Task Features in *medium-plan-b* but not in *medium-plan-a*: `SEAD`

Template Feature Differencing:

Template Feature	*medium-plan-a*	*medium-plan-b*
DURABILITY	medium	low
FORCE-FATIGUE	high	medium
FORCE-INTEGRITY	high	medium
LANDING-ZONE-PREP	medium	low

Role Differencing:

Different Single Valued

Role	*medium-plan-a*	*medium-plan-b*
RECON-ASSET	satellite-1	green-oda-2

Disjoint Multivalued

Role	Values for *medium-plan-a*	Values for *medium-plan-b*
ASSET	csar-c2-b tanker-1	uh-60l-1 yorktown sead-1 csar-c1-a
EXFIL-ASSET	uh-60l-1	aa201 uh-60a-2
FIRE-SUPPORT-ASSET	av-8b-harrier-ii-a	yorktown ah-100-1 ch-53e-super-stallion-1
INFIL-ASSET	uh-60l-2	aa7864 mh-60-g-pave-hawk-2
TRANSPORT-ASSET	tanker-1 av-8b-harrier-ii-a uh-60l-1 uh-60l-2	sead-1 uh-60a-2 mh-60-g-pave-hawk-2

Restricted Subtype

Role	Type for *medium-plan-a*	Type for *medium-plan-b*
ASSAULT-FORCE	ORANGE-UNIT	SOF-UNIT
INFIL-POINT	BUILDING	POINT-LOCATION
INFIL-TEAM	ORANGE-UNIT	SOF-UNIT
LANDING-LOCATION	AIRPORT	POINT-LOCATION

Restricted Subset

Role	Values for *medium-plan-a*	Values for *medium-plan-b*
EXFIL-POINT	mogadishu-town-hall	mogadishu-town-hall mogadishu-building4
INFIL-START	riyadh-airport	riyadh-airport ankara-airport

Figure 8. Comparison of *medium-plan-a* and *medium-plan-b*

assets are used, each with their individual strengths and weaknesses (see the table *Disjoint Multivalued*).

The tables *Restricted Subtype* and *Restricted Subset* show that plan *medium-plan-a* is much less diverse than plan *medium-plan-b*, since it uses more restricted sets of entities to fill a number of key roles (i.e., `ORANGE-UNIT` is a subtype of `SOF-UNIT`, `BUILDING` and `AIRPORT` are subtypes of `POINT-LOCATION`).

Overall, a user looking at the style of comparison in Figure 8 could quickly grasp the fundamental differences in strategy and resource usage between the two plans. Detailed examination of the plans themselves shows that there are additional differences in terms of unimportant low-level activities used to accomplish higher-level tasks and resource allocation. However, the metatheoretic comparison hides these nonessential differences.

Plan Space Analysis

We define two capabilities grounded in the domain metatheory for reasoning about a collection of plans: *identifying unique characteristics of a plan*, and *identifying maximally different plans*.

Identifying Unique Characteristics of a Plan

The metatheoretic differencing capabilities defined in the previous section can be used to identify three useful distinguishing characteristics of a plan *P* relative to a set *S* of candidate solutions.

1. *Unique task features:*
   - *P* has a task feature not found in any other *P'* in *S*
   - *P* lacks a task feature found in all *P* in *S*
2. *Unique normalized template features: P* has a normalized template feature value that differs from the value for all other solutions in *S.* This situation is especially interesting when all other plans share a common value for that template feature; in that case, the template feature for plan *P* is called *exceptional*.
3. *Differing role fills:* There is a role common to all plans for which some fill value in *P* does not occur as a fill value in other solutions in *S*.

Figure 9 summarizes the unique task features and normalized template features within our suite of test plans; they occurred in the plans *small-plan-b* and *medium-plan-a*. (We have not yet implemented the ability to look for differing role fills.)

The plan *small-plan-b* differs from all others in the test suite on the normalized value for the template feature `BLUE-CASUALTY-RISK`. In particular, its value for that feature is `low` while the other plans have value `medium`.

Plan: *small-plan-b*
  Has Exceptional Template Feature Values:
    `BLUE-CASUALTY-RISK: low;` all others `medium`

Plan: *medium-plan-a*
  Has Unique Task Features: `REFUELING`
  Has Exceptional Template Feature Values:
    `LANDING-ZONE-PREP: medium;` all others `low`
  Has Unique Template Feature Values:
    `DURABILITY: medium`
    `FORCE-FATIGUE: high`

Figure 9. Unique Features in the Test Suite

The plan *medium-plan-a* has several unique characteristics relative to the other plans in the test suite. First, it is the only plan with the task feature `REFUELING`; hence, no other plans in the test suite include refueling operations. Second, while the plan *medium-plan-a* has the normalized value `medium` for the template feature `LANDING-ZONE-PREP`, all other plans have the value `low`. Finally, the plan *medium-plan-a* differs from the other plans in the values for template features `DURABILITY` and `FORCE-FATIGUE`; in those cases, however, there is no common value for the remaining plans in the test suite.

Maximally Different Plans

For many applications, a human planner will want to explore a range of plans that embody qualitatively different solutions [Tate et al. 1998; Myers & Lee, 1999]. Such exploration can be useful both in terms of helping the user understand fundamental tradeoffs that are inherent to the domain, and identifying 'out of the box' solutions that he may not normally consider.

Our metatheoretic differencing techniques can be used to identify plans that are semantically far apart from each other, and hence are likely to have significant qualitative differences. To that end, we define a concept of *distance* between plans that builds on the concepts of *task feature*, *template feature*, and *role distance* between plans.

Task Feature Plan Distance

Task feature distance is a normalized form of Hamming distance for the task features within the plans. In particular, it is defined to be the ratio of the number of task features that appear in one but not both plans to the number of features that appear in either plan.

Definition 4 [Task Feature Plan Distance] The *task feature distance* between plans P_1 and P_2, denoted by *TaskFeatureDist(P₁, P₂)*, is defined by

$$TaskFeatureDist(P_1,P_2) = \frac{|TaskFeatures(P_1) - TaskFeatures(P_2)| + |TaskFeatures(P_2) - TaskFeatures(P_1)|}{|TaskFeatures(P_1) \cup TaskFeatures(P_2)|}$$

Template Feature Plan Distance

Template feature distance for a pair of plans is defined to be the average distance between the values of those features that are common to both plans, normalized with respect to the range of possible values for the features. Let *TemplateFeatures(P)* denote the set of template features that occur in plan *P*, and *FDist*(f,P_1,P_2) the distance between values for template feature *f* in plans P_1 and P_2.

Definition 5 [Template Feature Plan Distance] The *template feature distance* between plans P_1 and P_2, denoted by *TemplFeatureDist*(P_1, P_2), is defined by

$$TemplFeatureDist(P_1,P_2) = \frac{\sum_{f \in F^{\cap}} FDist(f,P_1,P_2)}{|F^{\cap}|}$$

$$F^{\cap} = TemplateFeatures(P_1) \cap TemplateFeatures(P_2)$$

For quantitative feature values, *FDist*(f,P_1, P_2) is defined as

$$\frac{|TemplFeatureValue(f,P_1) - TemplFeatureValue(f,P_2)|}{|Domain(f)|}$$

For qualitative template feature values, the normalizations and differencing required to calculate *FDist*(f,P_1,P_2) should be done within a single application of the mapping θ_f^{-1} from the qualitative values to *Interval*(*f*) (i.e., rather than mapping once to compute each *TemplFeatureValue*(f,P_i) and then again to difference them). This is necessary to minimize the discretization error from applying θ_f to map back to *Domain(f)*. Let V_i = *TemplFeatureInsts*(f,P_i); the qualitative version of *FDist*(f,P_1,P_2) is defined to be

$$\frac{(|V_2| \times \sum_{v \in V_1} \theta_f^{-1}(v)) - (|V_1| \times \sum_{v \in V_2} \theta_f^{-1}(v))}{|V_1| \times |V_2|}$$

Role Plan Distance

Role distance for a pair of plans is defined in terms of how distant the sets of fill values are for the roles that the two plans share. Our measure for the distance between sets of role fill values is defined to be the ratio of values that appear in one but not both sets to the total number of fill values (another normalized form of Hamming distance). We note that when possible, it may be appropriate to employ more specialized definitions that take into account the semantics of the underlying values. Such a definition could, for instance, reflect the fact that two airplanes of the same type are 'closer' than an airplane and a helicopter.

Definition 6 [Role Plan Distance] The *role distance* between plans P_1 and P_2, denoted by *RoleDist*(P_1, P_2), is defined as follows.

$$RoleDist(P_1,P_2) = \frac{\sum_{r \in R^{\cap}} RoleFillDist(r, RoleFills(r,P_1), RoleFills(r,P_2))}{|R^{\cap}|}$$

$$R^{\cap} = Roles(P_1) \cap Roles(P_2)$$

$$RoleFillDist(r,V1,V2) = \frac{|V1 - V2| + |V2 - V1|}{|V1 \cup V2|}$$

Metatheoretic Plan Distance

Using the above definitions, we define the *metatheoretic distance* between two plans as follows.

Definition 7 [Metatheoretic Plan Distance] The *metatheoretic distance* between plans P_1 and P_2, denoted by *PlanDistance*(P_1, P_2), is defined as follows, where $w_1 + w_2 + w_3 = 1$.

$$\begin{aligned} PlanDistance(P_1, P_2) = {} & w_1 \times TaskFeatureDist(P_1, P_2) \\ & + w_2 \times TemplFeatureDist(P_1, P_2) \\ & + w_3 \times RoleDist(P_1, P_2) \end{aligned}$$

The definition of metatheoretic plan distance assumes a set of weights, w_i, that can be used to adjust the relative importance of task features, template features, and roles in the distance specification. Because these three components address different aspects of an overall plan, different users may be interested in biasing the plan distance calculation to stress the relative importance of these three components.

Similarly, the definitions for template feature, task feature, and role distance can be modified in a straightforward manner to support weights that enable varying degrees of emphasis for individual features and roles. Such weights could be defined either for an entire domain or customized by an individual user (on a situation-by-situation basis, if so desired).

Plan Distances for the Test Suite

The motivation for defining the concept of plan distance was to support a user in identifying semantically distinct plans. The results in Figure 10 show that for the SOF domain, our definition is effective. The figure displays the distances for plans in our test suite, using an equal distribution of weights for the task feature, template feature, and role distances (i.e., w_1=w_2=w_3 =1/3).

The figure shows that the 'closest' plans correspond to core plans and their variants. In particular, the shortest plan distances found are between the two tiny plans (.08), between the various large plans (.08, .09, .15), and between the two small plans (.15). The distance between the two medium plans is appreciably higher (.31); as noted in Figure 1 and made apparent in Figure 8, these two plans are not simple variants of each other but rather contain key strategic differences. In addition, the plans that are farthest apart (the tiny vs large plans) are indeed the plans with the greatest meaningful variations among them. These results thus provide a preliminary validation of the effectiveness of the metatheoretic methods for capturing meaningful similarities and differences among plans.

Plan Dist	Template Feature, Task Feature, Role Dist	Plan1	Plan2
.08	.03 .11 .08	*large-plan-a*	*large-plan-b*
.09	.00 .00 .28	*large-plan-a*	*large-plan-c*
.12	.00 .00 .35	*tiny-plan-a*	*tiny-plan-b*
.15	.03 .11 .30	*large-plan-b*	*large-plan-c*
.15	.15 .00 .31	*small-plan-a*	*small-plan-b*
.17	.12 .14 .26	*small-plan-a*	*medium-plan-a*
.21	.00 .14 .49	*small-plan-a*	*medium-plan-b*
.26	.27 .14 .37	*small-plan-b*	*medium-plan-a*
.29	.15 .14 .56	*small-plan-b*	*medium-plan-b*
.31	.14 .25 .53	*medium-plan-a*	*medium-plan-b*
.34	.12 .25 .67	*small-plan-a*	*large-plan-a*
.35	.07 .22 .75	*medium-plan-b*	*large-plan-b*
.35	.21 .67 .18	*tiny-plan-b*	*small-plan-b*
.36	.12 .25 .72	*small-plan-a*	*large-plan-c*
.37	.08 .33 .70	*small-plan-a*	*large-plan-b*
.40	.27 .25 .67	*small-plan-b*	*large-plan-c*
.40	.18 .33 .69	*medium-plan-a*	*large-plan-a*
.40	.27 .25 .68	*small-plan-b*	*large-plan-a*
.41	.18 .33 .71	*medium-plan-a*	*large-plan-c*
.41	.14 .40 .69	*medium-plan-a*	*large-plan-b*
.41	.11 .33 .79	*medium-plan-b*	*large-plan-a*
.42	.23 .33 .69	*small-plan-b*	*large-plan-b*
.42	.11 .33 .81	*medium-plan-b*	*large-plan-c*
.44	.21 .67 .45	*tiny-plan-a*	*small-plan-b*
.47	.29 .67 .46	*tiny-plan-a*	*small-plan-a*
.47	.29 .67 .46	*tiny-plan-b*	*small-plan-a*
.51	.29 .71 .51	*tiny-plan-b*	*medium-plan-b*
.51	.29 .71 .53	*tiny-plan-a*	*medium-plan-b*
.52	.42 .71 .42	*tiny-plan-a*	*medium-plan-a*
.52	.42 .71 .42	*tiny-plan-b*	*medium-plan-a*
.56	.33 .75 .58	*tiny-plan-a*	*large-plan-c*
.56	.33 .75 .60	*tiny-plan-a*	*large-plan-a*
.56	.33 .75 .60	*tiny-plan-b*	*large-plan-a*
.57	.33 .75 .62	*tiny-plan-b*	*large-plan-c*
.58	.37 .78 .60	*tiny-plan-a*	*large-plan-b*
.58	.37 .78 .60	*tiny-plan-b*	*large-plan-b*

Figure 10. Plan Distances for the SOF Test Suite

Discussion

To date, research on general-purpose plan summarization and comparison methods has focused on approaches that analyze plan structures and planning processes directly. For example, [Mellish & Evans 1989] generate a textual description of a plan that references every plan element, without regard to its relative importance, thus making it difficult to understand the essence of large plans. [Young 1999] improves on that work by rating the importance of an action in a plan by counting the number of its incoming causal links; only actions with certain numbers of links are included in the plan summary.

Such syntactic approaches do not necessarily shed light on the semantic content of a plan. In particular, it is possible to have plans with significant variations in syntactic structure that are semantically similar; as well, plans with similar syntactic structure may have semantic differences that are extremely significant.

One key benefit of our metatheoretic approach to plan summarization and comparison is its emphasis on semantic rather than syntactic characteristics of plans. Thus, our comparison of metatheoretic properties grounds the results in concepts that are significant from a semantic perspective, rather than concepts that are important to an automated system when generating a plan.

Our approach also supports customization to domains, individual users, or specific contexts. This can be achieved by selecting the sets of features and roles that are of interest to the user (for plan summarization and plan comparison) and by appropriate adjustment of weights (for analyzing a solution space).

Our plan summarization and comparison methods are *domain-independent*, making them applicable to a broad range of problems. In particular, we avoid domain-specific algorithms or bodies of knowledge that would limit the applicability of the method. One problem with general-purpose methods is that their generality often comes at the cost of depth. This tradeoff applies to our approach, in that more precise quantitative analysis tools could be developed for an individual domain that provide deeper summarization and comparison capabilities.

Our methods for plan comparison and summarization are not intended to eliminate the need for more discriminating tools. Rather, we envision the metatheoretic approach being valuable in the early stages of planning, both in terms of enabling a user to quickly understand the main features of a plan, and to perform an inexpensive analysis of what differentiates alternative candidate plans. After developing some preliminary understanding of the plan space, a user may then wish to perform more expensive and time-consuming quantitative analyses to assess plans in detail.

The existence of a well-designed domain metatheory is critical for the successful application of our plan summarization and comparison methods. As noted elsewhere [Myers, 2000], the design of the metatheory should be a by-product of a principled approach to modeling a planning domain. Still, it remains a bit of an art to design a metatheory appropriately.

The explanatory capability of our methods when applied to larger plans could be improved by introducing a capability for contextualization that could localize application of the summarization and comparison techniques to meaningful subportions of a plan. This localization could enable more interesting regularities or trends within plans to be identified. The hierarchical structure of HTN plans provides an obvious way to generate candidate contexts, namely, subplans appearing below a given task node. Within that framework, however, identifying the most appropriate contexts for a given situation remains an interesting challenge.

Conclusions

AI planning tools must provide effective explanation capabilities in order for them to gain acceptance for real applications. To date, there has been relatively little effort devoted to developing such capabilities. Furthermore, the work that has been done has focused on syntactic elements of plans and planning processes, despite the fact that such syntactic characteristics may not correspond to important semantic features.

This paper defines an approach to plan summarization and comparison that builds on the notion of a domain metatheory. The approach has the benefit of framing summaries and comparisons in terms of high-level semantic concepts, rather than low-level syntactic details of plan structures and derivation processes. We defined a set of techniques that instantiate this approach and evaluated them within the context of a rich special operations planning domain. The evaluation showed that the techniques are effective in helping a user understand subtle aspects of individual plans, importance differences among plans, and the structure of the overall solution space.

Acknowledgments This work was supported by NASA grant NCC 2-1267. The author thanks the following people for their contributions to this work: Mabry Tyson for his assistance in making modifications to PASSAT to support the metatheoretic summarization and comparison capabilities, Peter Jarvis for developing the suite of SOF test plans used to evaluate the plan summarization and comparison techniques, Michael Wolverton for discussions on explanation technology and feedback on the ideas.

References

Erol, K., Hendler, J., and Nau, D. 1994. Semantics for Hierarchical Task-Network Planning. Technical Report CS-TR-3239, Computer Science Department, University of Maryland.

Mellish, C. and Evans, R. 1989. Natural Language Generation from Plans. Computational Linguistics 15(4).

Myers, K. 1996. Strategic Advice for Hierarchical Planners. In Proc. of the 5th Intl. Conf. on Principles of Knowledge Representation and Reasoning, Cambridge, Massachusetts.

Myers, K. L. and Lee, T. J. 1999. Generating Qualitatively Different Plans through Metatheoretic Biases. In Proc. of AAAI-99, AAAI Press, Menlo Park, CA.

Myers, K. L., Tyson, W. M., Wolverton, M. J., Jarvis, P. A., Lee, T. J., and desJardins, M. 2002. PASSAT: A User-centric Planning Framework. In Proc. of the 3rd Intl. NASA Workshop on Planning and Scheduling for Space, Houston, TX.

Tate, A., Dalton, J., and Levine, J. 1998. Generation of Multiple Qualitatively Different Plan Options. In Proc. of Artificial Intelligence Planning Systems.

Wick, M. R. and Thompson, W. B. Expert System Explanation, Artificial Intelligence, 54(1-2), 1992.

Young, R. M. 1999.Using Grice's Maxim of Quantity to Select the Content of Plan Descriptions. Artificial Intelligence, 115(2).

Mixed-Initiative Planning in MAPGEN: Capabilities and Shortcomings

John L. Bresina, Ari K. Jónsson*, Paul H. Morris, Kanna Rajan

NASA Ames Research Center
Mail Stop 269-2
Moffett Field, CA 94035
{bresina,jonsson,pmorris,kanna}@email.arc.nasa.gov

Abstract

MAPGEN (Mixed-initiative Activity Plan GENerator) is a mixed-initiative system that employs automated constraint-based planning, scheduling, and temporal reasoning to assist the Mars Exploration Rover mission operations staff in generating the daily activity plans. This paper describes the mixed-initiative capabilities of MAPGEN, identifies shortcomings with the deployed system, and discusses ongoing work to address some of these shortcomings.

Introduction

In January 2004, NASA landed rovers on the surface of Mars at two widely separated sites. Their mission: to explore the geology of Mars, especially looking for evidence of past water. At the time of writing, signs of past water presence have been discovered at both sites, and although well past their design lifetime, both rovers are still healthy, and the mission is continuing.

Operating the Mars Exploration Rovers is a challenging, time-pressured task. Each day, the operations team must generate a new plan describing the rover activities for the next day. These plans must abide by resource limitations, safety rules, and temporal constraints. The objective is to achieve as much science as possible, choosing from a set of observation requests that oversubscribe rover resources. In order to accomplish this objective, given the short amount of planning time available, the MAPGEN (Mixed-initiative Activity Plan GENerator) system was made a mission-critical part of the ground operations system.

In this paper, we report on the mixed-initiative capabilities of the MAPGEN system, outline some of the shortcomings that we observed during the deployment effort or during mission operations, and then briefly describe more recent research that is attempting to address some of these shortcomings. We first present some background material on the MER mission and then summarize the characteristics of the MAPGEN system.

* Research Institute for Advanced Computer Science - USRA

Background

The MER rovers (see Figure1), Spirit and Opportunity, are solar-powered (with a storage battery) and incorporate a capable sensor and instrument payload. Panoramic cameras (Pancam), navigation cameras (Navcam), and a miniature thermal emissions spectrometer (MiniTES), are mounted on the mast that rises above the chassis. Hazard cameras (Hazcams) are mounted on the front and rear of the rover. A microscopic imager (MI), a Mössbauer spectrometer (MB), an alpha particle X-ray spectrometer (APXS), and a rock abrasion tool (RAT), are mounted on the robotic arm.

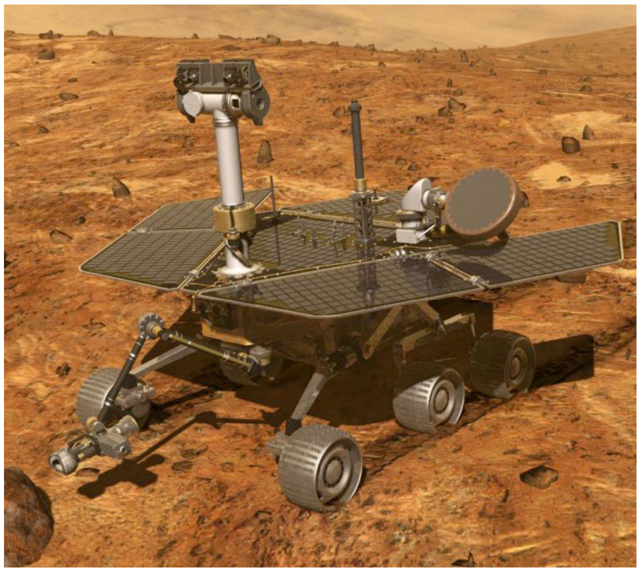


Figure 1: MER Rover

An onboard computer governs the operation of subsystems and provides data handling, system state tracking, limited obstacle avoidance, and so forth. Because of its large power draw and the rover's limited energy supply, the computer is used judiciously.

The rovers are equipped with extensive communication facilities, including a High Gain Antenna and Low Gain

Antenna for Direct-To-Earth transmission and reception, as well as an UHF antenna for communicating with satellites orbiting Mars. Communication opportunities are determined by each rover's landing site and the Deep Space Network schedule or orbital schedules for the satellites.

For this mission, the communication cycle was designed so that both rovers could be commanded every sol (i.e., Mars mean solar day, which is 24 hours, 39 minutes, and 35.2 seconds). The time for ground-based mission operations is severely limited by the desire to wait until up-to-date information is available but nevertheless finish in time to get the command load to the rover. During the nominal mission, this left 19.5 hours for ground operations. In this process, the engineering and science data from the previous sol are analyzed to determine the status of the rover and its surroundings. Based on this, and on a strategic longer-term plan, the scientists determine a set of scientific objectives for the next sol. At this stage only rough resource guidance is available. Hence, the scientists are encouraged to oversubscribe to ensure that the rover's resources will be fully utilized in the final plan.

In the next step in the commanding process, the science observation requests are merged with the engineering requirements (e.g., testing the thermal profile of a particular actuator heater) and a detailed plan and schedule of activities is constructed for the upcoming sol. The plan must obey all applicable *flight rules*, which specify how to safely operate the rover and its instrument suite and remain within specified resource limitations. It is in this step that the Tactical Activity Planner (TAP) employs MAPGEN.

Once approved, the activity plan is used as the basis to create sequences of low-level commands, which coordinate onboard execution. This sequence structure is then validated, packaged, and communicated to the rover. This completes the commanding cycle.

MAPGEN System Summary

Traditionally, spacecraft operations' planning is done manually; utilizing software tools primarily for simulating plan executions and identifying flight rule violations. The time criticality and complexity of MER operations, combined with advances in planning and scheduling technology, provided an opportunity for deploying automated planning and scheduling techniques to the Mars rover ground-operations problem.

As an integral part of a large mission operations system, MAPGEN's capabilities have evolved over time with the rest of the ground data system. The current user features are the end result of a journey through the design space, guided by feedback from the users in the course of many tests and subject to the changing landscape of the overall operations system. We can summarize the primary features as follows:

- **Plan editing:** Both activities and constraints can be modified, via direct manipulation, form editing, or menu items.

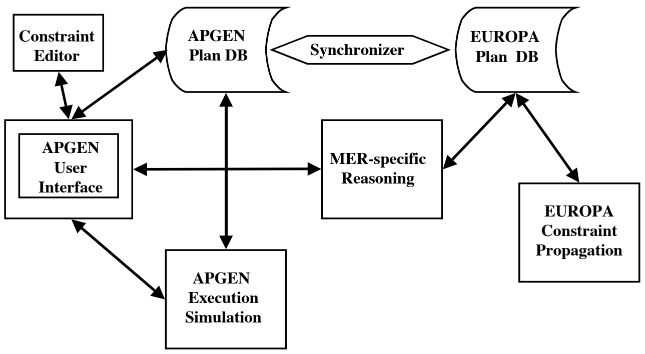


Figure 2: MAPGEN Architecture

- **Plan completion:** The selected subset of activities can be completed, in the sense that all subgoals are achieved and any necessary support activities are added to the plan.
- **Active constraints:** During plan editing, the formal constraints and rules are *actively* enforced. Thus, when one activity is moved or modified, other activities are modified as needed to ensure the constraints are still satisfied.

The MAPGEN system has five primary components, some of which were pre-existing software modules (see Figure 2). One of the requirements for infusing this technology into the mission was the use of an existing interactive plan editor from JPL, called APGEN (Maldague, et al., 1998), as the front end of MAPGEN. The core of the plan representation and reasoning capabilities in MAPGEN is a constraint-based planning framework called EUROPA (Extendable Uniform Remote Operations Planning Architecture), developed at NASA Ames Research Center (Jónsson, et al., 1999; Frank and Jónsson, 2003).

The new functionality in the MAPGEN system involves the interface between these two subsystems, support for extensions to the APGEN graphical user interface to provide the mixed-initiative capabilities, and more sophisticated plan search mechanisms that support goal rejection, priorities, and timeouts. The APGEN and EUROPA databases, which remain separate, are kept synchronized; changes may be initiated by either database.

Finally, we considered it expedient to develop an external tool, called the Constraint Editor, to enter and edit daily science constraints, since this is not conveniently supported by the current APGEN graphical user interface.

We next further describe the EUROPA, APGEN, and Constraint Editor components.

EUROPA

In constraint-based planning (Frank and Jónsson, 2003), actions and states are described as holding over intervals of time. Each state is defined by a predicate and a set of parameters, as in traditional planning paradigms. Actions, which are durative, are also represented by parameterized predicates. The temporal extent of an action or state is specified in terms of start and end times. For example, specifying that the panorama camera heater needs to be on for 25 minutes, starting at 8:00, could be written as:

```
  holds(8:00,8:25,pan_cam_htr(on,0:25))
```

However, in constraint-based plans, each time and parameter value is represented by variables, connected by constraints. Consequently, the statement would be:

```
  holds(s,e,pan_cam_htr(state,dur))
  s=8:00, e=8:25, state=on, dur=0:25
```

Constraint reasoning plays a major role in the constraint-based planning paradigm. Any partial plan, which is a set of activities connected by constraints, gives rise to a constraint network. Constraint-based inference can provide additional information about plans, reduce the number of choices to make and identify dead-end plans early. Achieving arc consistency is one commonly used example of applicable constraint reasoning methods.

Typically, the temporal variables and associated constraints give rise to a simple temporal network (STN), or can be reduced to one by decision choices that enforce the mutual exclusion constraints. For STNs, it is possible to make the network arc consistent and to determine consistency in low-order polynomial time, using the Bellman-Ford algorithm (Dechter, Meiri, and Pearl, 1991; Cormen, Leiserson, and Rivest, 1990).

In constraint-based planning, explicit temporal constraints fall into three categories: *model* constraints, *problem-specific* constraints, and *expedient* constraints. The model constraints encompass definitional constraints and mutual-exclusion flight rules. In MER, for example, the expansion of activities into sub-activities gives rise to temporal relations between the parent and its children.

The problem-specific constraints comprise "on the fly" relations between specific activities in a planning problem. In MER, these constraints, often called "daily constraints", related elements of scientific observations in order to capture the scientists' intent. As an example, several measurements of atmospheric opacity may be required to be at least 30 minutes apart. These constraints are entered using the Constraint Editor tool, described below.

The expedient constraints are those resulting from arbitrary decisions made to guarantee compliance with higher-level constraints that cannot be directly expressed in an STN. For example, a flight rule might specify that two activities are mutually exclusive (such as moving the arm while the rover is moving). This is really a disjunctive constraint, but satisfying it will involve placing the activities in some arbitrary order. Expedient constraints are typically added during search in automated planning.

APGEN

APGEN (Activity Plan GENerator) is an institutional tool at JPL and has been used in a number of spacecraft missions. It has a large number of features, but the core capabilities can be summarized with three components:

- **Activity plan database**: A set of activities, each at a specific time. This database has no notion of constraints between activities, but does support context-free activity expansion.
- **Resource calculations**: A method for calculating, using forward simulation, resource states that range from simple Boolean states to complex numerical resources.
- **Graphical user interface**: An interface for viewing and editing plans and activities.

To deploy APGEN for a particular mission, the mission-specific information is stored in an adaptation, which can be viewed as a procedural domain model. It defines a set of activity and state types and then defines a way to calculate resource states from a given set of activities. In addition, it defines a set of "constraints" on legal combinations of resources. The constraints and resource calculations are only useful for passively identifying problems with a plan; APGEN does not have the capability to reason with this information in order to help fix the identified problems.

Constraint Editor

The APGEN plan-editing interface has no notion of variables and constraints in the traditional AI sense. This raised the issue of how to get the daily constraints into the reasoning component of MAPGEN. These daily constraints were needed to coordinate the activities in scientific observations, and these could vary in unforeseen ways. For example, it might be specified that two specific measurements should be taken within 10 minutes of each other. This required an ability to enter and modify temporal constraints dynamically.

To resolve this, an external, temporal-constraint editing tool, called the Constraint Editor, was developed as an augmentation to the APGEN interface. In this tool, users can view activities and existing temporal constraints, and then add, delete, or edit constraints.

Mixed-Initiative Planning in MAPGEN

In this section, we first motivate the need for a mixed-initiative approach to activity planning and then describe the capabilities in MAPGEN that supported this approach.

In traditional automatic planning, the operator loads in the goals and initial conditions, pushes a button, and waits for a complete plan. Due to the need to bring human expertise in mission planning and science operations to bear on solving this complex operational problem, this approach was deemed unacceptable; consequently, we adopted a mixed-initiative approach for this application.

There were many aspects of the need for human involvement. Mission operations rely on a number of checkpoints and acceptance gates to ensure safety. For activity plans, the critical gate was the activity plan approval meeting where the fully constructed plan would be presented by the Tactical Activity Planner (TAP),

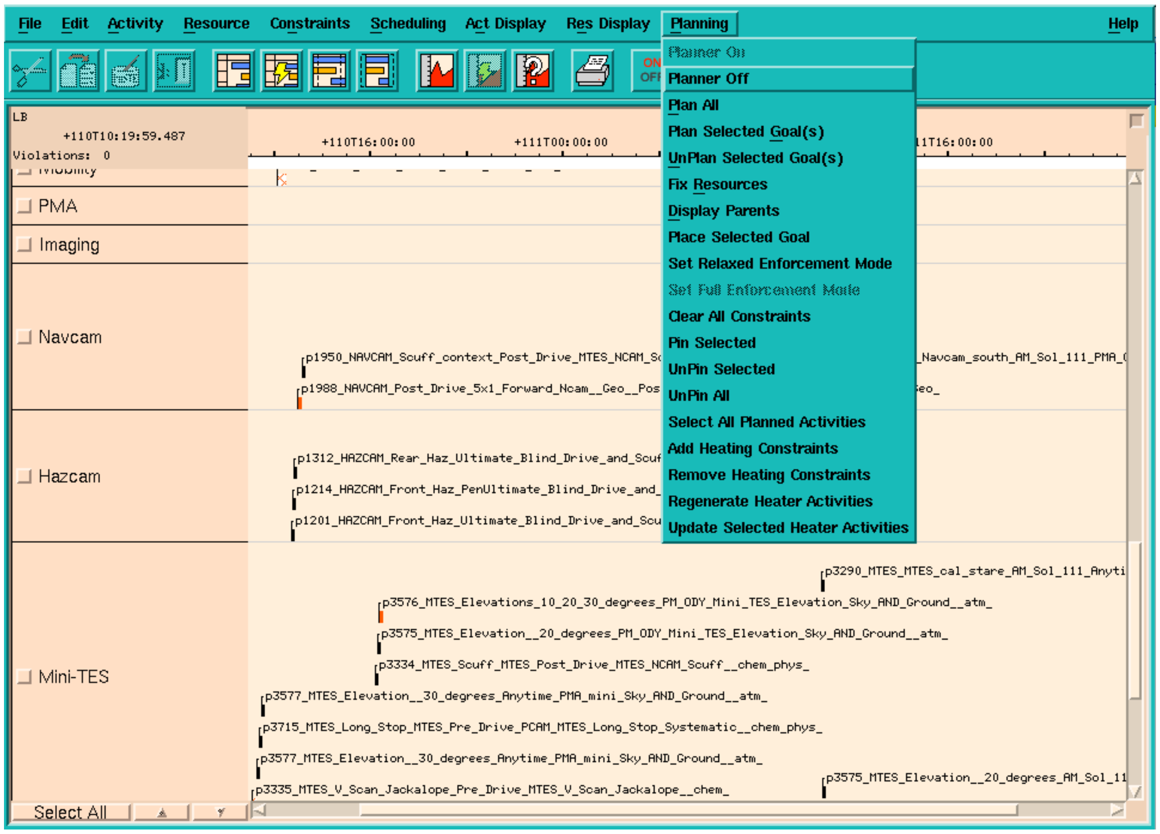

File
Edit
Activity
Resource
Constraints
Scheduling
Act Display
Res Display
Planning
Help
Planner On
Planner Off
Plan All
Plan Selected Goal(s)
UnPlan Selected Goal(s)
Fix Resources
Display Parents
Place Selected Goal
Set Relaxed Enforcement Mode
Set Full Enforcement Mode
Clear All Constraints
Pin Selected
UnPin Selected
UnPin All
Select All Planned Activities
Add Heating Constraints
Remove Heating Constraints
Regenerate Heater Activities
Update Selected Heater Activities
LB
+110T10:19:59.487
Violations: 0
+110T16:00:00
+111T00:00:00
11T16:00:00
PMA
Imaging
Navcam
Hazcam
Mini-TES
Select All
p1950_NAVCAM_Scuff_context_Post_Drive_MTES_NCAM_Sc
p1988_NAVCAM_Post_Drive_5x1_Forward_Ncam__Geo__Pos
_Navcam_south_AM_Sol_111_PMA_
p1312_HAZCAM_Rear_Haz_Ultimate_Blind_Drive_and_Scuf
p1214_HAZCAM_Front_Haz_PenUltimate_Blind_Drive_and_
p1201_HAZCAM_Front_Haz_Ultimate_Blind_Drive_and_Scu
p3290_MTES_MTES_cal_stare_AM_Sol_111_Anyti
p3576_MTES_Elevations_10_20_30_degrees_PM_ODY_Mini_TES_Elevation_Sky_AND_Ground__atm_
p3575_MTES_Elevation__20_degrees_PM_ODY_Mini_TES_Elevation_Sky_AND_Ground__atm_
p3334_MTES_Scuff_MTES_Post_Drive_MTES_NCAM_Scuff__chem_phys_
p3577_MTES_Elevation__30_degrees_Anytime_PMA_mini_Sky_AND_Ground__atm_
p3715_MTES_Long_Stop_MTES_Pre_Drive_PCAM_MTES_Long_Stop_Systematic__chem_phys_
p3577_MTES_Elevation__30_degrees_Anytime_PMA_mini_Sky_AND_Ground__atm_
p3575_MTES_Elevation__20_degrees_AM_Sol_11
p3335_MTES_V_Scan_Jackalope_Pre_Drive_MTES_V_Scan_Jackalope__chem_

Figure 3: MAPGEN with planner menu

critiqued by both scientists and mission specialists, and, hopefully, accepted, possibly with minor modifications. As a result, the TAPs had to be able to understand, defend, and sign-off on the validity of the plan. Initial user tests indicated that a plan constructed automatically in its entirety was too difficult to analyze by the human operator, especially given the inherent time pressures. The TAPs, therefore, prefer to incrementally construct a plan in small, understandable chunks.

Another major concern was the infeasibility of formally encoding and effectively utilizing all the knowledge that characterizes plan quality. One aspect of plan quality involves a rich set of science preferences, including everything from preferences on absolute and relative scheduling of activities to preferences on which combinations of science observation cuts and changes are least painful in the face of strict resource limitations. A second, and more complex, aspect of quality is concerned with global characteristics of a plan, such as acceptable profiles of resource usage, and the estimated complexity of turning a plan into a command sequence structure.

The role of mixed-initiative planning in MAPGEN is very much in the spirit of the original notion of such planning (Burstein and McDermott, 1996); the purpose is to support collaboration between a human user and an automated system to build a high quality activity plan. However, it is worth noting that, unlike some variations of mixed-initiative planning, MAPGEN does not actively solicit user assistance during planning. The primary role of the operator is to direct and focus the plan construction process and to provide qualitative evaluation of plans. The system makes automated planning capabilities available to the user and performs potentially tedious tasks, such as maintaining constraints. The intended interaction between user and system is that the system handles constraint enforcement constantly in the background, while automated plan construction is user invoked.

Interactive plan modification

One of the core issues in mixed-initiative planning is the introduction of external decision-making and plan editing into a carefully designed automated search engine. The intrusion of user choices complicates commonly used approaches such as backtracking search and propagation-based checking of consistency. The EUROPA planning framework used in MAPGEN supports non-chronological backtracking, but it cannot propagate information in plans that have constraint violations. To support arbitrary changes by users, MAPGEN included a plan modification strategy that would adjust plans to eliminate inconsistencies.

Mixed-initiative planning systems must respond and return control quickly to the user. For an automated planning operation, which involves a cascading decision process, MAPGEN relaxes completeness in favor of responsiveness. This has to be done carefully to maximize chances of finding near-optimal solutions within limited time. We developed a backtracking algorithm that noted the difficulty of planning activities, and when the effort to plan an activity exceeded an allowance determined by its priority, the activity was rejected from the plan.

In constraint-based planning, partial plans have an underlying simple temporal constraint network (Dechter, Meiri, and Pearl, 1991). The consistency of STNs can be determined by checking for arc consistency. Furthermore, each value in an arc-consistent temporal variable domain appears in at least one legal solution for the temporal network. The set of such values defines a temporal interval that can be represented by its bounds.

Consider a plan where all decisions have been made, except for grounding temporal variables appearing only in simple temporal constraints. Finding a fixed solution is then an easy matter of choosing a value for any variable within its legal bounds, re-enforcing arc consistency, choosing a value for another variable, and so on.

It is not necessary to immediately ground the variables; plans with temporal variables left ungrounded are called *flexible plans*. In MAPGEN, we utilize the fact that the underlying plans are flexible to support a common way for users to modify plans, namely to change the placement of activities in time. As long as the activity is moved only within the flexibility range defined by the domain in the underlying arc-consistent flexible plan, the result is necessarily another consistent instantiation. This observation gave rise to the notion of a *constrained move*.

During a constrained move, the system actively restricts the movements of an activity to stay within the permitted range. Then, once the user places the activity, any dependent activity is updated as necessary to yield a new valid plan instance.

Note, however, that the consistency enforcement takes into account *all* the constraints that determine the flexible plan. This includes expedient constraints resulting from decisions about how to order mutually exclusive activities. Since these decisions are maintained, the ordinary constrained move has the effect of "pushing" the excluded activities ahead of it. However, sometimes the TAP wants to *reorder* mutually excluded activities. To support this, we provided a variation, called a *super-move*, that temporarily relaxes expedient constraints until the move is completed.

Adjustable automation

MAPGEN users wanted an adjustable spectrum of automated planning services (see Figure 3). The system offers a fully automated "plan everything" operation, a selective "plan this and everything related to it" operation, and a fine-grained "plan this and try to put it here" operation. Users can also un-plan activities and store them in a "hopper," which holds requested activities that are not yet in the plan.

The *plan all* operation leaves it entirely up to the automated search to find a plan that achieves as much science as possible. This functionality is most like what traditional automated planning methods do. This capability functioned well and yielded near-optimal plans in terms of the number of science observations in the plan. However, the plans tended not to have an intuitive structure and, therefore, made it difficult for the TAP to explain the plan structure during the approval meeting. Additionally, they were often sub-optimal with respect to preferences and other solution quality criteria that were not encoded in the domain model or the priorities. Consequently, it was rarely used.

Instead, the users often applied a more incremental operation, called *plan selected goals*. With this operation, the user could select a set of observation requests not in the plan and request that these be inserted into the partial plan already in place, such that all constraints were satisfied. While repeated application of this led to a result similar to the full planning variation, users found this more intuitive, in part because it allowed them to fine-tune and understand the incremental plans as they were built. Furthermore, this made it possible for the users to have a complete plan ready at just about any time.

The user could exercise even more control over the planning process via the *place selected goals* operation, which was applicable only to individual activities. This operation allowed the user to select an activity in the hopper and then choose an approximate temporal placement for it in the plan. The planning algorithm would then treat the user-chosen time as heuristic guidance and search for a plan where the selected activity was as close to the desired time as possible.

Minimizing perturbation

The key to making the automated services feel natural and unobtrusive is for them to respect the existing plan as much as possible. This is accomplished by combining an effective form of temporal placement preference with a heuristic bias. For changes in the temporal placement of activities, the system exploits the underlying temporal flexibility of EUROPA plans. As each plan represents a family, the system chooses an instance to display that is as close as possible to what the user had prior to the changes being made.

The method we developed is based on minimizing the departure from a *reference* schedule, which need not be consistent. The reference schedule provides a general method for expressing unary temporal preferences. Its primary use in MAPGEN is to support a minimum perturbation framework where changes to the previous plan are minimized when a planner-supported operation is invoked, such as a constrained move. This is accomplished by continually updating the reference schedule to reflect the evolving plan. This means that changes made by the user to reflect preferences or eliminate problems are respected and maintained unless they violate constraints or are revised by the user.

When it came to making activity placement choices, i.e., expedient ordering-decisions, the heuristic guidance used was based on minimizing deviation from the reference schedule. The motivation behind this was twofold. One was that it would be intuitive to the user, as this approach would attempt to preserve the temporal placement of activities. The other motivation was that it would allow users to "sketch out" a plan in the hopper and then ask the system to complete the plan. For more details on this method, see (Bresina, et al., 2003).

Addressing MAPGEN's Shortcomings

During the multi-year deployment effort, there were a number of capabilities on our task agenda that never made it to the top of the stack; we also encountered issues that require significant research before being ready for mission deployment. During mission operations, we observed a number of shortcomings, and often we were not able to address them at that time due to the restrictions of the change control process or due to the complexity of the issue. In this section, we focus on the shortcomings in MAPGEN's mixed-initiative approach and describe some of the new research we are carrying out to address them.

Explanations

The clearest lesson we have learned from our observations is the need for the automated reasoning component to provide better explanations of its behavior. Especially important are explanations of why the planner could not achieve something, such as inserting an activity in the plan at a particular time, or moving an activity beyond the enforced limit. Such a facility would have greatly helped during training, in addition to increasing the TAPs' effectiveness during operations. The system did have a form of explanation of inconsistency by presenting a minimal *nogood*. While the TAPs found it to be useful when editing constraints, only the developers used this facility in the context of constructing and modifying plans, and this was done for the purpose of debugging the system. The reason is that, in this context, the explanation typically involved complex chains of activities and constraints that could not easily be grasped. For example, during MER, nogoods encountered during planning could involve hundreds of constraints.

There are several contexts in which inconsistencies can arise during planning. First, when an activity is considered for insertion, it may be inconsistent with the current plan even before any location is examined. Second, it may be inconsistent with the *specific* location chosen in a Place Selected operation. Third, it may be inconsistent with *each one* of the possible locations identified during a Plan Selected operation. The first context gives rise to a nogood directly. In the second context, a nogood can be extracted by temporarily placing the activity in the infeasible location. In the third context, it may be possible to resolve the individual nogoods arising from each location to form a compound nogood. Note that these cases may arise before or during the search. We have focused our efforts thus far on the first context; we expect similar considerations to apply in the other contexts.

The lengthy nogoods are partly an artifact of the mixed-initiative planning process. When MAPGEN attempts to insert an additional activity into the evolving plan, it first brings in (i.e., starts enforcing) the constraints associated with that activity. Since the existing plan was formulated without those constraints, it is often the case that they are inconsistent with previous ordering decisions made to prevent forbidden overlaps (due to mutual exclusion restrictions). Furthermore, the ordering decisions may involve mutual exclusions between low-level activities that are part of activity expansions. Because of this, the constraint engine must keep track of interactions between activity expansion constraints and planner decision constraints, as well as daily constraints. The duration of a high-level activity is also determined by its activity expansion constraints, so if this is a factor in an inconsistency, the raw nogood will include the entire expansion of the high-level activity. Thus, the raw nogoods during planning can be very large.

It is obviously impractical to expect a time-pressured TAP to read, let alone grasp the significance of, a nogood involving hundreds of constraints. However, we believe that the essential content of the nogood can be summarized in a concise form. To this end, we have been investigating methods of *compressing* nogoods.

The first compression step rolls up expansions that are only needed because they determine a higher-level duration that is involved in the inconsistency. While this step helps, the explanations can still be quite long, often involving chains of duration and daily-constraint pairs. We can distinguish between these constraints, which should be known to the TAP, and the "hidden" constraints that come from planner ordering decisions. The second compression step rolls up the duration/daily sequences into a single chunk. Based on MER examples, these two steps typically compress the nogood by a factor of ten.

A remaining issue is that sub-chains of the nogood that pass through planner ordering decisions can wander somewhat randomly through large portions of the plan. The intermediate wandering is not very meaningful in terms of understanding the inconsistency, so a further step could involve rolling such a segment into a single statement about planner placement of the bookend activities in the segment.

These compression steps carry the risk that one of the components of the compressed summary will itself be mystifying. To counter this, it would also be useful to allow components of the summary to be re-expanded on demand. Thus, the nogood would be organized into a hierarchical structure that is more easily grasped.

In general, an inconsistent network may involve more than one inconsistency. The approach used in the constraint editor is to first present one (the first one found by the temporal reasoning algorithm), have the user resolve that, then present another one if the network is still inconsistent, and so on. This may not be the best approach within the planning context.

Considering the entire set of nogoods, it may be possible to select the one nogood that yields the "best explanation", i.e., an explanation that is easiest to understand and leads to the easiest resolution of the associated inconsistency. Another approach is to focus on constraints common to multiple nogoods, such that the user could resolve more than one inconsistency with one constraint retraction. A prerequisite for either of these approaches is a suitable algorithm for enumerating all the temporal nogoods. At this point, it is not clear how practical it is to compute such an enumeration, since theoretically the number of nogoods may be exponential in the size of the network.

Temporal preferences

A second important issue is that the user does not have sufficient means to control the planning process and to influence the types of solutions generated. In MAPGEN, the user's only language for specifying their desires is to create a set of absolute (hard) temporal constraints, which represent what is necessary for the observation requests to be scientifically useful. These constraints can specify ordering among the activities and observations (along with temporal distances required) and can specify that an activity or observation has to be scheduled within a particular time window. For example, the scientist can specify that three atmospheric imaging activities have to be a minimum of thirty minutes apart and a maximum of six hours apart. However, the scientist cannot specify that they *prefer* the largest possible spacing between the three activities. Likewise, they cannot specify that a particular spectrometer reading must occur between 10:00 and 15:00 but it is preferred to be as near to 12:00 as possible. It is clear that both absolute constraints and temporal preferences are needed to generate a high-quality science activity plan.

MAPGEN did have a limited capability for expressing start time preferences via the reference schedule of the minimal-perturbation approach. The operator could also establish more complex preferences by an iterative process of relaxing or tightening hard constraints, but this is too time-consuming and too primitive of an approach.

We are currently investigating a number of alternative, automated approaches to incorporating temporal preferences into MAPGEN. We have extended the Constraint Editor to allow specification of temporal preferences on an activity's start or end time, as well as on distances between start/end time points of two activities.

There are three key issues involved in utilizing temporal preferences in mixed-initiative planning. The first is the common problem of combining local preferences into a global evaluation function. The second issue is finding a globally optimal instantiation of a given flexible plan. The third key issue is searching for a flexible plan that yields a globally preferred instantiation.

Let us first consider the second issue. To effectively solve constraint problems that have local temporal preferences, it is necessary to be able to order the space of assignments to times based on some notion of global preference. Globally optimal solutions can be produced via operations that compose and order partial solutions. Different concepts of composition and comparison result in different characterizations of global optimality. Past work (Khatib, et al., 2001; Khatib, et al., 2003, Morris, et al., 2004) has presented tractable solution methods (under certain assumptions about the preference functions) for four notions of global preference: weakest link, Pareto, utilitarian, and stratified egalitarian. These four notions are examples of general solutions to the first issue, namely, how to combine local preferences into an overall comparison of solutions.

We are incorporating these preference-optimization methods into MAPGEN and plan to employ them for a number of purposes. One use is to apply the optimization, as a post-process, to the family of solutions represented by a flexible MAPGEN plan in order to display the most-preferred solution to the user. These methods can also be employed, as a pre-process, to compute the reference schedule as a globally optimal solution to the specified temporal preferences. The minimal-perturbation method would then try to stay close to this globally optimal reference. We also intend to investigate other heuristic methods that include consideration of the preferences when making search decisions; thus, addressing the third issue.

Other shortcomings

The need for explanations and handling of temporal preferences were the most obvious shortcomings that needed to be addressed. Consequently, work is already underway to address those. However, a number of other issues have been identified.

In addition to temporal preferences, users may have preferences regarding the global characteristics of the solution, such as plan structure preferences or resource usage preferences. Many constraints can have absolute validity limits and a preference on the legal values. For example, the limits on the energy usage may be determined by minimum battery levels, but it is preferred that the battery be left charged above a certain level at the end of the plan. As with temporal preferences, the main issues are how to combine local preferences into global evaluations functions and how to then control the search towards preferred plans.

In MAPGEN, the underlying plan is always kept consistent. This allows propagation to take place at any time, which in turn enables active constraint enforcement, constrained moves, and other propagation-based capabilities. However, the users sometimes desire to "temporarily" work with plans that violate rules or constraints. One possible approach for allowing violations is to isolate the inconsistent parts of the plan; a second approach is to allow constraints and rules to be disabled and re-enabled. The latter approach was in fact designed for the MAPGEN tool, but we never got a chance to implement it. Future work will explore possible approaches and techniques for this.

The users also want to advise the planner on how it makes decisions at a high level and on how the planner's search is done. Users have noted that they would like to specify limits on what the automated reasoning process can change in order to enforce constraints and rules. For example, users may want a portion of the plan to remain unchanged, either in terms of a subinterval of the plan's time span or a subset of the plan's activities.

It would also be useful for the system to answer questions from the user regarding trade-offs, for example, by answering the following types of queries:

- What needs to be unplanned (in priority order) to enable additional time for arm instrument use, or to allow for driving further?
- For a given panorama that does not fit as a whole, which parts of it can be fit into the current plan?
- In order to fit in another imaging activity, what needs to be unplanned or shortened?

Another technique for supporting trade-off analyses is to help the user better understand the space of possible solutions by presenting qualitatively different solutions. We are extending some previous work on advisable planners (Myers, 1996; Myers, et al., 2003) to apply within the context of our constraint-based planning technology in order to help address these issues.

Acknowledgements:

The authors would like to acknowledge the entire MAPGEN team, which includes: Mitch Ai-Chang, Len Charest, Brian Chafin, Adam Chase, Kim Farrell, Jennifer Hsu, Bob Kanefsky, Adans Ko, Pierre Maldague, Richard Springer, and Jeffrey Yglesias. We would also like to acknowledge our research collaborators on mixed-initiative planning, Karen Myers and Michael Wolverton from SRI.

References

Bresina, J., Jónsson, A., Morris, P., and Rajan, K.. Constraint Maintenance with Preferences and Underlying Flexible Solution. *CP-2003 Workshop on Change and Uncertainty*, Kinsale, Ireland, 2003.

Burstein, M., and McDermott, D., Issues in the development of human-computer mixed-initiative planning. In *Cognitive Technology*, B. Gorayska, and J. L. Mey, editors, pages 285-303. Elsevier, 1996.

Cormen, T., Leiserson, C., and Rivest, R.. *Introduction to Algorithms*. MIT press, Cambridge, MA, 1990.

Dechter, R., Meiri, I., and Pearl, J., Temporal constraint networks. *Artificial Intelligence*, 49:61– 95, May 1991.

Frank, J., and Jonsson, A., Constraint-Based Interval and Attribute Planning, *Journal of Constraints* Special Issue on Constraints and Planning, 2003.

Khatib, L., Morris, P., Morris, R., Rossi, F. Temporal reasoning about preferences. *Seventeenth International Joint Conference on AI*, Seattle, WA, 2001.

Khatib, L., Morris, P., Morris, R., Venable, B. Tractable pareto optimization of temporal preferences. *Eighteenth International Joint Conference on AI*, Acapulco, Mexico, 2003.

Maldague, P., Ko, A., Page, D., and Starbird, T., APGEN: A multi-mission semi-automated planning tool. *First International NASA Workshop on Planning and Scheduling*, Oxnard, CA, 1998.

Morris, P., Morris, R., Khatib, Ramakrishnan, and Bachmann. Strategies for Global Optimization of Temporal Preferences. *Tenth International Conference on Principles and Practices of Constraint Programming* (CP-2004), Toronto, Canada, 2004.

Myers, K.L. Advisable Planning Systems. In *Advanced Planning Technology: Technological Achievements of the ARPA/Rome Laboratory Planning Initiative*, ed. A. Tate, 206-209. AAAI Press, 1996.

Myers, K. L., Jarvis, P. Tyson, W. M., and Wolverton, M. J. 2003. "A Mixed-initiative Framework for Robust Plan Sketching". In *Proceedings of the 13th International Conference on Automated Planning and Scheduling*, Trento, Italy.