

TAIPE: Tactical Assistants for Interaction Planning and Execution

Edmund H. Durfee

EECS Department
University of Michigan
Ann Arbor, MI 48109
durfee@umich.edu

Marcus J. Huber, Michael Kurnow, Jaeho Lee

Orincon Corporation
9363 Towne Center Drive
San Diego, CA 92121
{marcush | mkurnow | jaeho}@orincon.com

Abstract

Downsizing the number of operators controlling complex systems can increase the decision-making demands on remaining operators, particularly in crisis situations. An answer to this problem is to offload decision-making tasks from people to computational processes, and to use these processes to focus and expedite human decision making. In this paper, we describe a system comprised of multiple computational agents that has demonstrated an ability to help operators prioritize their tasks better, process their tasks faster, and enlist the aid of other operators more transparently. In developing this system, we have of course encountered challenges, particularly in devising content languages that adequately convey the right information (to be interpreted correctly) across the heterogeneous agents. We here summarize our work that addresses this challenge, and illustrate how our system improves performance for operators in naval situations.

Introduction

Monitoring the operations of systems such as nuclear power stations, chemical plants, and battleships consists of long periods of routine operations requiring little attention, punctuated by dynamic time-critical situations where operators are maximally stressed. Thus, while the worst-case needs require a large staff, on average most of the operators are underutilized. The large discrepancy between worst-case and expected-case needs suggests opportunities for saving on personnel (to improve safety and cost-effectiveness) provided that sufficient decision-making resources can be mobilized during crises.

For example, the Navy has embarked on a program of automating many ship systems so that crew sizes can be reduced. A consequence of this effort has been that the remaining crew members each need to supervise more tasks and oversee a greater variety of ship systems. Especially in high-stress situations, this potentially leads to overtaxing an operator's cognitive abilities as the operator not only has to manage many tasks at once, but also has to move between various complex, non-standard interfaces to legacy ship systems.

Agent-based technology can support an operator in several ways. First, software agents can be tasked with operations that a

human cannot currently manage, including monitoring for particular external situations (such as the presence of an approaching, potentially hostile craft) or internal situations (such as loss of power to the ship's engines), and alerting the operator when particular conditions hold. Second, software agents can help focus the operator's attention by using embedded models of situations to automatically prioritize the currently active alerts for the operator. Third, software agents can internalize models of how the various interfaces to ship systems should be manipulated to bring up the right views of the right information at the right time as an operator is investigating an alert condition. And fourth, software agents can monitor the alert processing loads across operators so as to move critical alerts among operators so that they are addressed by someone in a timely manner.

The Tactical Assistant for Interaction Planning and Execution (TAIPE) provides agent-based technology for these purposes. TAIPE is a multi-agent system, where specialized agents internalize knowledge about particular tasks that the operator needs performed, and the agents interact among themselves and with the operator to offload some number of tasks (depending on the degree to which tasks exceed available human attention) from the operator, freeing the operator's attention for the most critical and complex tasks requiring human reasoning capabilities. TAIPE agents even help coordinate the actions of multiple operators such that important tasks are transferred among operators to ensure that they are completed quickly.

In this paper, we describe TAIPE and its component technologies, and illustrate how the TAIPE agents embedding these technologies collectively support human operators. TAIPE is significant for several reasons. First, it provides a case study in the using agent-based methods to engineer a real application with substantial legacy software that must be "wrapped" to interface with the agent components. Second, TAIPE brings together a heterogeneous collection of agents to formulate, edit, prioritize, and execute interaction plans to support the operator. As a result, we have developed initial insights into developing a content language for heterogeneous planning agents. Third, TAIPE has undergone initial evaluation that empirically demonstrates its advantages to the operator. And fourth, TAIPE is inherently extensible to multiple operators and systems, as has been demonstrated in our prototype.

Our description of TAIPE begins with a brief summary of the ship systems domain, and of the various technologies incorporated in TAIPE. Then, we discuss in more detail the specific agents that we have developed and incorporated in TAIPE, along with how the agents interface to legacy ship automation systems. We subsequently turn to the development of protocols and content languages needed by the heterogeneous agents. Next, we describe the operation of TAIPE, and summarize a preliminary evaluation of its advantages. Finally, we conclude with a discussion of our ongoing improvements to TAIPE.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of ACM. To copy otherwise, or to republish, requires a fee and/or specific permission. Agents '97 Conference Proceedings, copyright 1997 ACM.

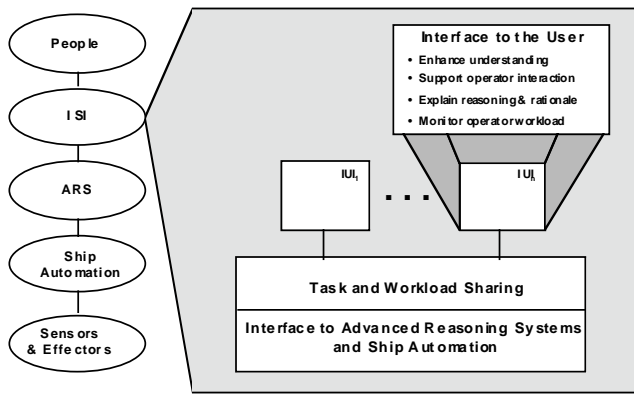


Figure 1: ISI Within a Decentralized System

Application Domain and Component Technologies

The underlying motivation for providing ship system's automation is to allow humans to play a supervisory role over automated ship processes, assigning routine, lower-level tasks to computational systems during normal operations, and offloading increasing numbers of more complex tasks during time-critical operations. Among the advantages of doing this are:

- Fewer people are onboard for operations, reducing the costs and risks to lives.
- Reaction time can be improved, since people monitoring routine situations tend to become bored and unfocused, while computer-based systems do not.
- The active operator processes can expand and decrease in response to the load on the overall ship system.

Intelligent System Interface

The Intelligent Systems Interface (ISI) is the interface between a human operator and the ship systems that that operator is controlling (Coury 1995). To be accepted, the ISI must fit within the existing hierarchical command structure of a ship, where responsibilities are divided into several domains:

- Tactical Scene (TS) manages sensor resources to gather, fuse, and interpret data to detect, identify, locate, track, and assess threats for the entire tactical scene.
- Tactical Action (TA) manages weapon resources and uses TS information to plan missions and control engagements during the execution of warfare.
- Platform Readiness (PR) manages mobility, logistics, and platform resources to monitor, diagnose, maintain, and repair the platform and to control casualties and damage.
- Command and Control (CC) manages communications and intelligence resources to plan missions, control cooperative engagements, and resolve conflicts among the other domains.

For each of these domains is associated a system comprised of the appropriate components for working in the domain (sensors, effectors, ship automation components, advanced reasoning systems) accessed through the ISI. The ISI provides an interface to the lower-level resources (advanced reasoning systems and ship automation), provides global (domain-wide) monitoring and

redistribution of processing load, and provides one or more Intelligent User Interfaces (IUIs) to the operators. See Figure 1.¹

Component Technologies

Inside of an Intelligent User Interface (IUI) are numerous functional components for managing the display and dialogue, managing requests for information, managing local tasks, generating explanations, and directing attention and planning user interactions. TAIPE provides a general agent-based architecture to encapsulate and integrate these functions. Specifically, TAIPE has initially focused on encapsulating as agents the specialized components for directing attention and planning user interactions for Tactical Scene operators. Specifically, TAIPE employs the following general architectural and problem-solving technologies to realize a variety of specialized agents:

- Graphical editing tools for specifying interaction plans by actual end users.
- A plan generation system, SIPE-2 (Wilkins 1988) from SRI International, for automatically formulating interaction plans given a description of what the operator needs to understand in a scene.
- A plan execution system, UMPRS (Lee et. al 1994), for controlling the sequencing and execution of interaction plans, as well as for executing plans for operator load management and for alert prioritization.
- An uncertain reasoning system, GISTER (Lowrance 1994) from SRI International, for combining evidence as to the global scene and the specifics of an alert condition to determine prioritization of operator attention direction and associated tasking.
- An advanced reasoning system, IDFS (developed by Orincon) for fusing sensory information and providing alternative views of the tactical scene to the operator.

TAIPE then provides the means by which these agents work together to execute multiple interaction plans in parallel and maintain a consistent view of the world.

Agent Development

The included technologies in principle provide a powerful basis for assisting an operator, but instantiating them into TAIPE has required that they be endowed with specific domain knowledge and that they be incorporated into separate agents whose activities can be interwoven to provide broader collective capabilities. The specific agents that have been constructed from the component technologies, the design rationale behind their development, and their desired interactions with other agents, are described below.

Interaction Plan Controller

The Interaction Plan Controller (IPC) agent, built upon the UMPRS architecture, is the operator's run-time interface to the Advanced Reasoning Systems (ARSs). Each subscribed ARS alerts the IPC to important situations, which the operator can acknowledge. Acknowledgment of an alert triggers the IPC to form an explicit goal to assist the operator in employing one or more ARSs by invoking an interaction plan. Interaction plans provide a specification of context-dependent invocation of ARS

¹ This figure is from [Coury 1995] by Bruce Coury of the JHU Applied Physics Lab.

interactive functionality for the retrieval and display of information via autonomous manipulation of the legacy ARS user interfaces. In other words, an interaction plan embeds knowledge of how to initialize, position, size, and populate display windows to show the operator the right information. During interaction plan execution, the operator maintains control of plan execution through a small number of buttons, menus, and pop-up windows that enable the operator to monitor the goal list, monitor goal and subgoal elaboration, acknowledge individual plan actions, view the executing plan, and abort executing plans.

Let us consider these activities in more detail. When the operator begins working with TAIPE, he or she specifies the aspects of the tactical scene to monitor and process, using graphical tools. The user interface downloads these triggering conditions to the appropriate ARS, or, more properly, to a process associated with an ARS that continuously polls the ARS to check for matches with triggering conditions. This process is called the Context Data Interrogator (CDI). When a CDI finds a match, it sends an alert message to the IPC. Before distracting the user with the new alert, however, the IPC first works with another agent, the Alert Prioritizer (AP) to determine how important the alert is. Consequently, the IPC can add the new alert to a prioritized queue of pending alerts, so that the operator will know at the outset how important an alert is (and might not even be shown relatively unimportant alerts during high-stress periods).

Operators have a choice as to whether they wish to process an alert. They indicate this desire by acknowledging the alert, which causes a goal to process the alert to be posted on the UMPRS goal list. This causes UMPRS to find appropriate interaction plans for the new goal, as well as for other goals that are still awaiting achievement. Using priority information, the IPC's UMPRS process will select a particular interaction plan to pursue next.

Because goals for multiple alerts could be simultaneously active, the IPC maintains a priority-sorted list, called the pending task list. The IPC manages interruption of low priority plans by higher priority plans by suspending a low priority plan until it eventually becomes the highest priority non-completed plan. The plan is then resumed from its beginning.¹ Of course, the operator can abort an executing plan, or delete any plan or alert from the pending task list or alert list, respectively, at any time.

Once an interaction plan has been selected, execution of the plan involves invocation of ARS actions. Associated with each ARS is a process which knows how to translate requests from the IPC (such as "show highest ranked track hypotheses") into particular manipulations of the operator's display. This process also knows how to task the ARS to evaluate expressions about ARS object characteristics and relationships with other objects.

Alert prioritizer

The Alert Prioritizer (AP) agent interacts with the IPC to both prioritize alerts when they are first sent to the IPC by an ARS, and also to periodically re-prioritize the IPC's existing alerts to rerank them according to the changing situation. The AP agent is also built upon the UMPRS architecture, and utilizes SRI's GISTER evidential analysis application to determine the relative importance of the alerts. Alert priorities are determined based on

¹ UMPRS supports resumption of a plan from the interrupted execution point, but we implemented this plan resumption strategy as a fail-safe default as the interaction plans currently make no guarantees about being able to safely resume execution from the interruption point.

an analysis of the current attributes of the object that is the subject of the alert. The Evidential Reasoner requests the required object attributes from the ARS through its Context Data Interrogator process. The current object attributes that are utilized in the AP agent's GISTER analysis consist of the probabilistic classification of the object in terms of its category and identity (air, surface, subsurface; neutral, friendly, hostile), and its intercept parameters (closest point of approach, time to go to intercept).

Plan editor

TAIPE functionality includes the graphical construction of interaction plans and alert contexts (triggering conditions) via the Interaction Plan Editor (IPE) component. Interaction plans are constructed by graphically composing ARS actions into sequences, subgoals, and conditionalized branches. The operator can also graphically specify the class of situations that the plan is valid for by constructing an enabling context expression to be evaluated by the ARS. Similar graphically generated expressions can be embedded at decision points in the interaction plan so as to create conditional paths of execution. When the plan is complete, the graphical representation of the interaction plan is encoded and stored as a UMPRS Knowledge Area for later use by the Interaction Plan Controller (IPC). Once created and stored, interaction plans can be reviewed, or edited by loading the plan into the IPE.

A subcomponent of the IPE is the graphical context editor, used to define the context requirements for an ARS to generate an alert. The graphical specification is encoded and stored as an alert template. Currently the IPE's graphical context editor provides a sampling of tools used to specify tactical scene objects and spatial constraints on those objects. Spatial constraints may be specified in terms of relative ranges, bearing sectors and intercept parameters. ARS-specific tools are also provided by the IPE graphical context editor. An example of such a tool is the cluster tool that was developed for the Integrated Data Fusion System (IDFS) ARS. This tool is used to specify the constraints on top ranked or alternative hypotheses derived by IDFS. An example of an alert template that could be specified with this tool could be an alternative hypotheses with a probability of 20% that contains a hostile surface track. As additional ARS systems are integrated with TAIPE, it is anticipated that additional ARS-specific graphical context editor tools and representations will be added.

Plan generator

TAIPE supports generative planning capabilities to assist the operator in constructing interaction plans. The generative planner that is currently integrated into TAIPE is SRI's SIPE-2 planning system. The operator invokes the generative planner via a planning request to accomplish a specific mission goal selected from the current list of high-level goals. The auto-generated interaction plan is then translated from the native form of the generative planner (SRI's ACT formalism) into a graphical form and displayed in the Interaction Plan Editor (IPE). Once generated and displayed, the operator can then review and edit the plan as desired. Upon saving the interaction plan, it is encoded into the UMPRS plan representation and stored in the Interaction Plan knowledge base.

ARS interfaces

To connect the intelligent interface components to legacy advanced reasoning systems (ARSs) requires that, for each ARS,

interface processes be developed to encapsulate it. Rather than formulating general-purpose wrappers, our strategy is to provide processes that are tailored to subsets of ARS functionality. Currently, we have identified two such processes. One, the Context Data Interrogator, polls the ARS based on query requests from other agents to identify when particular triggering conditions are met. The other, the Auto-Remote Manipulator, translates action requests into particular actions to be taken by the ARS.

Context Data Interrogator

The Context Data Interrogator (CDI) provides the direct interface to the information space of an ARS. The CDI consists of two primary functional components: an interagent interface component and a ARS interface component. There will typically be one CDI for each ARS that is integrated with TAIPE. The interagent interface component is ARS independent, being predominantly comprised of message handling mechanisms to support the agent protocols and languages. The ARS interface component is dependent on the architecture of the specific ARS. For example, in the case of the IDFS ARS, the form of the CDI interface is via connection to a shared memory segment that contains the relevant ARS state information.

The CDI provides autonomous monitoring of the ARS information space for recognition of context matches on object constraints that are associated with assigned alert templates, or needed by the IPC for enabling context or conditional plan elaboration context requirements. Context expressions are sent to the CDI as strings which are parsed by the CDI prior to expression evaluation. The expression grammar utilized by the CDI is ARS-independent and provides for logical relationships of object attributes. The mapping of, and access to, ARS-maintained state data referenced by CDI expressions is controlled by ARS-specific accessors that are incorporated within the CDI.

The CDI also provides object attribute computation and retrieval requests for the alert prioritizer (AP) agent. As described above, the AP agent requests the retrieval of object attributes for alert priority assessment. Some attributes (such as closest point of approach) require derivation, whereas other object attributes (such as classification probabilities) merely require retrieval and message formatting.

Auto Remote Manipulator

The Auto Remote Manipulator (ARM) provides for the direct manipulation of an ARS user interface in accordance with the actions specified in executing interaction plans. Whereas the CDI interfaces to an ARS information space, the ARM interfaces to the user interface component of an ARS. Like the CDI, the ARM consists of two primary functional components: an interagent interface component and an ARS interface component. There will typically be one ARM interface for each ARS that is integrated with TAIPE. The interagent interface component is ARS-independent. The ARM connectivity to an ARS user interface is dependent on the architecture of the specific ARS. For example, in the case of the IDFS ARS, the ARM interface has been incorporated within the IDFS display process.

The ARM receives interaction plan actions from the IPC when a plan is being elaborated during plan execution. The ARM provides for the decomposition of plan actions into sequences of elemental user interface actions which are then automatically applied to the ARS user interface via invocations of indigenous X functionality that is already provided by the ARS native user interface. The specific actions that are sent to the ARM are

encoded in interaction plan knowledge areas at the time the interaction plan is saved to the plan knowledge base.

Task Managers

TAIPE also contains functionality provided by Local Task Manager (LTM) and Global Task Manger (GTM) agents to monitor, assess, and reallocate workload between operators. There is a single UMPRS-based GTM agent and a number of UMPRS-based Local Task Manager (LTM) agents, one LTM for each operator. Workload is assessed for both alerts and tasks in response to periodic GTM queries. The GTM determines when a sufficient workload imbalance between operators occurs and then directs the LTM agents of the appropriate operators to either shed or accept workload.

When a task or alert imbalance is detected, the GTM computes the workload shedding necessary to balance the load. The shedding scheme could involve alerts, tasks, or both. The GTM forecasts the impact of the determined load shedding to determine if the shed load would cause the receiving operator to shed load back. If there will be no "ping-ponging" of workload, the GTM sends a message to the LTM of the overloaded operator to shed alert load or task load, as directed by the GTM. Those alerts and tasks selected are removed from visible displays and sent back to the GTM as a sequence of messages (one per shed task or alert). At this point, the GTM sends the appropriate message to the LTM associated with the underloaded operator, which then prompts its IPC to post the new goals and add new alerts for those received.

Agent Interaction

Even though many (but not all) TAIPE agents employ the UMPRS architecture, their particular tasks are very different, leading to different internal representations and input/output behaviors. This heterogeneity poses challenges when it comes to agent interactions. Central to an agent-based approach is a suite of languages and protocols that agents can use to team their efforts and share information and tasks to accomplish their goals. On the one hand, the languages and protocols should be comprised of high-level constructs that keep the system open to new agent types and tasks being introduced over time. On the other hand, the agents need to communicate about very specific pieces of information in order to correctly and completely transfer relevant information that will be used appropriately by the recipient. Thus, the model of interaction language we use follows the traditional notion of a layering of languages that build off of, and incorporate, each other.

Grounding this in more computational terms, the notion is that a message sent by one agent to another is sent because of some effect that the message is intended to have on the recipient, an effect desired by the sender. We can thus categorize different message types in terms of the well-defined effects of those types. These effects are sometimes called performatives.¹ Sending a message can be seen as invoking a performative at the recipient. For example, an IPC might send a "query" type message to an AP

¹ This is really a misnomer, since a performative is really a particular message type, in which a proposition is established by the very act of speaking (e.g., "I now pronounce you man and wife."). However, in keeping with the current (mis)usage of the term, we will consider a performative as being a message type that invokes a particular response from a hearer.

to find out an alert priority, and expects that message to evoke a good-faith effort on the part of the AP to send a “inform” message indicating the priority.

Interagent Protocol and Performatives

The notion of using message typing and discourse protocols as a means of flagging the intended processing that should be done on received messages is nothing new. Examples range from the venerable Contract Net (Smith 1980) to recent multiagent languages such as COOL (Barbenceau and Fox, 1995). While several speech-act-based languages have been proposed over the years, we have adopted and adapted KQML (Finin et al 1994). Within the Intelligent Systems Interface (ISI), a protocol such as KQML can be adopted as a standard means that agents can use to encode their communications in an unambiguous manner. This protocol is mapped to a network transport layer which actually carries out the transmission of the message. Initially, facilities for managing and connecting agents are less likely to be very useful, given that the ISI will, for the near term at least, be a closed system, such that the population of agents will only vary in controlled ways, and such that agents can directly know which other agents are capable of accomplishing tasks that they need done. Thus, KQML’s capabilities for advertising and brokering will not be exploited in the near term.

Planning Content Language

Once agents can successfully send messages among themselves, and express the intent of the message (what the sender wanted done with it), the next issue to address is the content of the messages. That is, while protocols support communication, without content they are useless. A fundamental need within the ISI, therefore, is to define a content language for plans. Plans can be for what operators should do, for what ARSs should do, for what the ship should do, and so on. Plans need to be generated, elaborated, visualized, analyzed, executed, revised, logged, and so on. In short, many different agents might need to be able to collectively form, manipulate, and execute plans.

A planning content language needs to satisfy all of the constituencies that would use the plan. While limited work has been done on plan representations that admit to having specialized agents act as demons to fill in particular features (Kambhampati et al, 1991), the problem that we face is that different combinations of agents will need to generate, modify, and execute different *overlapping* combinations of plan features. So, in exchanging a plan, the agents need to be able to find the information they need so as to take the actions that they are expected to take in modifying or executing the plan. They also need to know how to change the plan in ways that will be interpreted correctly by other agents and lead to desirable effects.

To date, there are few standards for specifying plans for computer-based agents. Some conventions certainly exist (such as the “STRIPS operator” format (Fikes & Nilsson 1971)), but these lack the expressiveness needed by plan generation and execution systems that explicitly consider features such as resource needs and real-time deadlines. One effort for formulating a more general description of a plan has been undertaken by SRI, in the development of their Cypress system (Wilkins et al 1995). In a nutshell, Cypress combined existing systems for plan generation and for plan execution. These existing systems were initially written to be stand-alone; Cypress needed to define a language that the two systems could use to exchange plans, despite the fact that what each system did with plans was very different.

The Act Formalism as a Plan Content Language

The outcome of this endeavor has included a formalism for expressing plans that satisfies the needs of Cypress’s two planning systems, and appears to capture the needs of both generative planning (as embodied by SIPE-2 (Wilkins 1988)) and plan execution (as embodied by PRS-CL (Wilkins et al 1995)). In their formalism, an ACT is composed of the following fields: name, cue, precondition, setting, resources, properties, plot, and comment.

Each of these fields has content expressed in some form. For Cypress, several of the fields only needed to be understandable to one or the other of the underlying planning systems. For example, some of the details dealing with action consequences or resources consumed within the structure of an ACT’s plot are important to the plan generation system but need not be understood by the plan execution system. Other fields need to be accessed and understood by both systems. For these, a common syntax and semantics is needed.

The strategy taken in ACT is the following. First, the content language for describing the environment (states of the world in which Cypress is acting) is in terms of specifying a relation (predicate) followed by zero or more arguments. ACT makes no commitment to the particular predicates, constants, and variables being manipulated, nor to the primitive actions that are available to a system using the plan formalism. This *domain-dependent* information must be defined for the particular application to which the planners are being applied. This is done in the ACT framework, through the use of declarations which define and classify the entities in the world that the plans are manipulating, the methods for doing the manipulations, the outcomes of the manipulations, and so on, captured in a domain-specific *ontology*. See (Wilkins & Myers 1995) for more details.

The ACT Formalism as a Standard Planning Content Language

The ACT formalism has provided an excellent starting framework for defining a planning content language. The fields of an ACT appear to capture important semantic features of a plan that are generally of concern to a variety of planning systems. Thus, the basic structure of ACTs can provide the foundation for an inter-agent plan content language. The efficacy of the language has already been established as part of the Cypress system, and in modified form within TAIPE.

However, because TAIPE involves more agents than a specific plan generator and plan executor, an important fundamental contribution of TAIPE has been to identify limitations of ACT, and to suggest improvements and extensions to ACT to make it more broadly useful as a general plan interlingua, including:

Broadly Readable Format. ACT, as initially written, is intended to be graphically represented to a user. Through the efforts of SRI, Orincon, and Michigan, a version of the ACT specification has been developed to provide a transferable, text-based, advertised representation that can be parsed appropriately by any of the agents conversing in the interlingua. Using this language, Michigan, with SRI and Orincon, has developed a translator that can convert this specification into the internal representation used by C-based UMPRS (Lee et al 1994), highlighting greater interlingual characteristics than the original ACT formalism.

Explicit Elaboration of Implicitly Declared Information.

ACTs allow a terse representation of planned procedures by relying heavily on the ACT interpreter having global domain knowledge that spans many of the ACTs. For example, it is assumed that agents communicating ACTs are working with a rich, common domain ontology, with sufficient and consistent knowledge about the objects in the domain and their classifications, the primitive actions available in the domain, and so on, so that commonly declared knowledge need not accompany every communicated ACT in order to make sense of it. However, implicit embedded knowledge in an interpreter has its drawbacks when significantly different systems are exchanging plans. An example of where this has arisen in the interactions between SIPE and UMPRS in TAIPE illustrates this point. An underlying assumption about the relationship between a generative planner and a plan executor, as embodied in Cypress, is that generative planning elaborates a plan down to a particular level of detail, and then can pass this off to the executor, which can perform further elaborations depending on dynamic context, to actually carry out the plan. While this model satisfies the needs of the Cypress system, it is not a general model of interaction. UMPRS does not assume that the "handoff" occurs at a particular level, and expects the passed plan to be explicit as to whether a particular step represents a subgoal to "ACHIEVE" (triggering UMPRS to retrieve appropriate subplans) or a particular action to "EXECUTE" (triggering UMPRS to invoke a particular function). The distinction can be very important, defining the latitude that the plan generator entrusts to the plan executor (Ephrati and Rosenschein). In developing an ACT translator for UMPRS, therefore, we have had to encode directly executable function in special (degenerate) procedures to unify the invocation style. Extensions to ACT still need to be done, to give it a clean, explicit semantics (Lee & Durfee 1994).

Extended Meta-Predicate Suite. In accomplishing the enrichment of ACTs to support other types of inter-planner interactions, it is important to define a set of meta-predicates that is sufficiently concise to be understandable, while also being sufficiently broad to capture the significant semantic differences in the operations of various interpreters of an ACT. The current constructs of ACT were motivated by the needs of combining generative and reactive planners at a particular level of plan detail. Meanwhile, the constructs in UMPRS were tailored for peer execution systems that might need to communicate about and make commitments to particular courses of action. More generally, the purposes of interaction among agents that talk about plans can be quite widely varied, and so it is probable that the interlingua will continue to grow. Our starting point for extending the ACT meta-predicates is based on a characterization of what meta-predicates need to tell the interpreter, including the relative importance of a goal, the temporal constraints on achieving a goal, the temporal extent of a goal, and so on.

Modular Control Structures. To clarify semantics and support analysis, our extensions are also working toward introducing more modular control structures into ACTs. The ACT plot topology can be fairly unstructured, with plot nodes pointing to other plot nodes pointing to other plot nodes.... While this yields quite a bit of flexibility, it also can potentially make larger constructs in a plot difficult to extract or modularize. The more strict use of structuring constructs has been advocated in UMPRS, and in the subsequent development of Structured Circuit Semantics (Lee & Durfee 1994).

It is important to reiterate that ACT is intended to be broadly applicable, and so are these improvements. That is, while motivated in the context of TAIPE, these improvements are really addressing the needs of a plan content language that must provide

clear, analyzable, and operational semantics in highly heterogeneous agent-based systems.

Task Management Content Language

Though much less developed, we have also embarked on defining a content language for describing and passing tasks. For the purposes of load balancing among heterogeneous agents, the content language must walk a tightrope between allowing a task to be specified in detail (so that it is done precisely as defined) and allowing agents with their own idiosyncratic capabilities to exploit them. Moreover, to keep overhead down, it is better to effect a more permanent transfer of multiple tasks rather than incur the costs of sending a continuous stream of single tasks. Thus, a task management content language must start by identifying what a task might be:

- A fully specified combination of a goal to pursue, a method to pursue it, and a designation of all of the materials/resources to be used by the method.
- A partial specification that defines a class of activities. Given a goal, let the agent figure out how it wants to accomplish the goal. In this case, the partial specification stands for a disjunctive (do-any) task class -- the agent need only accomplish one of the tasks defined by the partial specifications.
- A goal template that defines an even broader set of activities. Given a template, generate and accomplish all goals that match the template. This is an instance of a conjunctive (do-all) task class.

In TAIPE, therefore, a task description language could include: goal description or a partial description template), context description (variable bindings which could involve objects or pointers to objects on Advanced Reasoning Systems (ARSs)), and method description (ACT to achieve the goal by, or ACTs *not* to use).

In the ISI, there is not yet a well-formulated taxonomy of tasks, and chances are that such a taxonomy will evolve as new types of operators and automation systems are incorporated. Currently, the task management content language is comprised of REQUESTs from the GTM to LTMs for information about load and for shedding/accepting tasks, and INFORMs from the LTMs to the GTM about load and tasks to shed. But this language will evolve, and the identification of the different task classifications above will play an important role in identifying the degree to which task-passing resolves temporary load imbalances versus accomplishing more permanent organizational redesign (Durfee & Montgomery 1991).

Goal Prioritization Content Language

A final use of interagent languages in the extended TAIPE system allows communication between an agent that is maintaining a set of goals (alerts to process) for the operator, and an agent that can perform probabilistic reasoning to assess the importance (priority) of processing a particular alert. Again, for this language, the messages are couched in terms of KQML-like speech acts. The content of the messages is specific to the particular tasks of formulating and revising alert priorities. The messages are summarized below.

```
QUERY PRIORITY-EVAL <alert instance data>
INFORM PRIORITY-EVAL <alert instance data>
REQUEST REFRESH-PRIORITIES
```

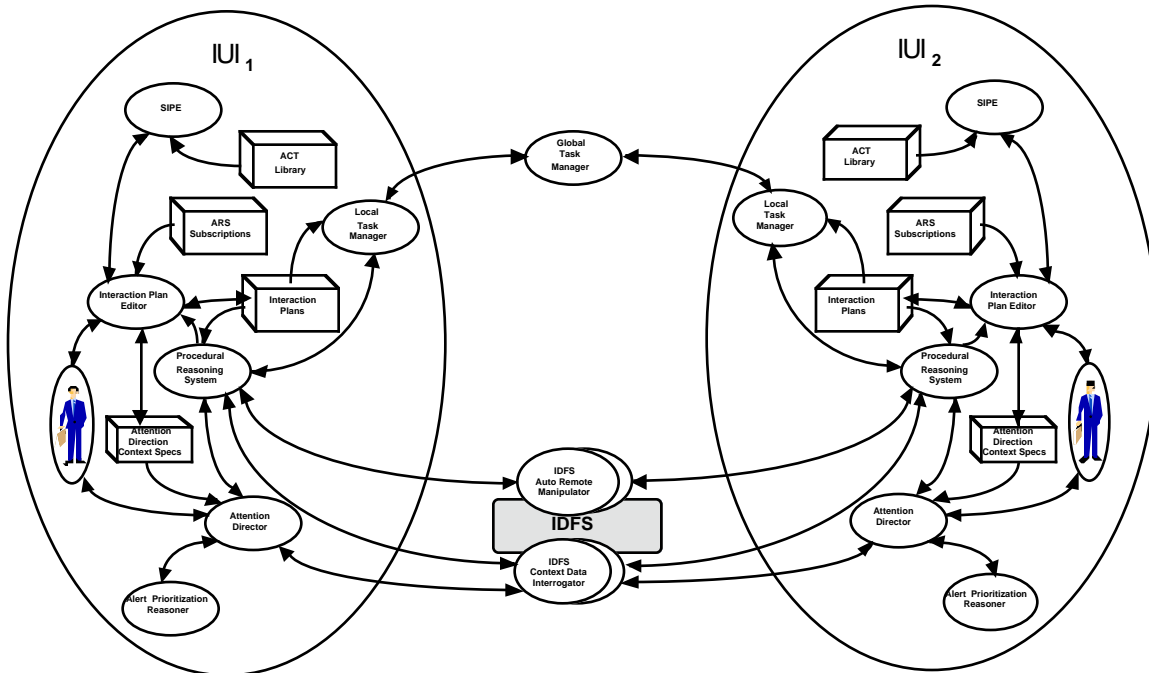


Figure 2: TAIPE Implementation

TAIPE Implementation, Performance, and Evaluation

TAIPE, as it is currently implemented, is graphically depicted in Figure 2. The operator in an off-line (or on-line when needed) mode uses the IPE to store alert templates and interaction plans in the appropriate libraries. When the operator begins using TAIPE to assist in the tactical scene assessment, he or she begins by selecting from the defined alert templates those that the operator wants to process. The templates are sent to the ARS's CDI. As external events occur, the CDI matches alerts and sends back messages to the IPC. The IPC requests priority computation from the AP, and uses the returned value to update the portion of the operator's screen showing pending alerts (color coded as red, yellow, or green). As the operator acknowledges alerts, they are added to IPC's pending task queue, and the IPC initiates an interaction plan for the most important task. The operator steps through the plan (or can request the IPC to step through without waiting for operator permission), which invokes ARS functionality and configures information displays to support the understanding of (and response to) the alert by the operator.

At any given time, higher priority alerts might appear and be acknowledged, which can change the current interaction plan so that the operator rapidly develops awareness of more critical features of the environment. Moreover, the ongoing task management processes are checking to see whether one operator has more (and more important) tasks than the other, and initiates movement of tasks to rectify unbalanced situations. From the viewpoint of an operator, tasks (or alerts) disappear or appear over time.

Our preliminary evaluation of TAIPE utilization for IDFS tasks shows a great reduction in operator workload compared to manual manipulation of IDFS. The IDFS graphical user interface is

currently composed of seventeen distinct displays that incorporate 131 possible control actions and 178 display filter settings. Comparison of task performance workload with and without utilizing TAIPE shows that for one typical task the operator was required to perform 49 relatively complex manual manipulations (such as sizing and placing windows, pulling down menus, etc.) compared to 7 very simple manipulations (single mouse clicks) when assisted by TAIPE. In another task, the operator experience a reduction from 19 complex to five simple manual operations.¹ This reduction of manual operations comes as the result of two TAIPE features, ARM actions that accomplish multiple equivalent manual manipulations, and the ability for an operator to specify automated sequences of ARM actions within interaction plans.

The simple metrics described above only measure some of the many aspects of an operator's physical workload, however, and do not take cognitive workload into account at all. Cognitive load is much harder to measure, and represents an important aspect of an operator's perception of total workload. While we currently have no metrics for cognitive workload, working with TAIPE perceptively reduces cognitive workload as well by performing attention direction through automated alertment and prioritization of alerts, which greatly reduces the need for the operator to continually analyze the changing scene simply to identify important aspects of the scene.

TAIPE's workload reallocation mechanisms have not been exercised to such an extent that we can make any conclusions about their efficacy in improving multi-operator performance by balancing overall workload. The interagent protocol and simple balancing algorithms implemented so far have shown themselves to balance workload reliably and effectively. We have great

¹ These examples were based on typical IDFS use, and we expect them to be typical of the improvements TAIPE will provide. However, more extensive test are needed to confirm this.

confidence that significant benefits will be demonstrated when we develop and collect multi-operator performance metrics.

Conclusion

TAIPE represents state-of-the-art technology for operator assistance in the application domain of tactical scene assessment in naval operations. As we have outlined, TAIPE encompasses a variety of useful tools, including tools for focusing operator attention, reducing an operator's need for remembering how to work with a particular analysis system (ARS), and moving processing load among operators. Our initial implementation of TAIPE has allowed us to begin evaluating the utility of these capabilities, and our demonstrations have become a centerpiece in defining the role of technology to support reduced crewsizes in the Navy.

At a more scientific level, our work on TAIPE has forced us to address fundamental research issues in the development and deployment of multiple agents that collectively satisfy operator needs. Agent heterogeneity in TAIPE is what allows it to succeed so well, and yet has proven a challenge during development, raising critical questions in the definition of interface processes for legacy systems and content languages for agents to use. Our strategy of providing multiple interfaces to a legacy system, rather than a single wrapper process, has proven effective, although it requires more study. Meanwhile, our work in defining agents that use plans in different ways, and in defining a language that they can all use in communicating about their plans, has uncovered basic tradeoffs between the advantages of explicit semantics for controlling an interpreter and the disadvantages in terms of message size (and effort on the part of people/agents that use the language to spell everything out). Our observations about plan content languages are seemingly also going to hold true for languages whose content includes tasks or beliefs. And our examination of the task content language points to a simple, general-purpose strategy for effecting long-term load-balancing decisions through the exchange of task templates.

Our ongoing work involves the development of these languages and, concurrently, the extension of TAIPE to support more operators, more ARSs, and more operator domains (e.g. platform readiness and tactical action).

Acknowledgments

We would like to acknowledge the invaluable participation in this effort of David Wilkins and John Lowrance of SRI International in the development of the alert prioritizer, plan generator, and interlingua. Brad Clement and Pradeep Pappachan have also contributed to TAIPE development. This work has been supported, in part, by DARPA under contract N66001-93-D-0058.

References

Barbenceau, M. and Fox, M. S., 1995. "COOL: A language describing coordination in multi-agent systems." In Proceedings of the First International Conf. on Multi-Agent Systems, pages 17-24, June.

Coury, B. (editor) 1995. "Intelligent Systems Interface (IS) Functional Specification Document (Draft)." Produced for the ARPA/MSTO Ship Systems Automation Program, April 1995.

Durfee, E. H. and Montgomery, T. A., 1991. Coordination as distributed search in a hierarchical behavior space." *IEEE Trans. on Sys., Man, and Cyber.* 21(6):1363-1378.

Ephrati, E., and Rosenschein, J. S., 1992. "Constrained Intelligent Action: Planning under the influence of a master agent." In *Proceedings of the Tenth National Conference on AI*, pages 263-268, July.

Fikes, R. E. and Nilsson, N. J. 1971. "STRIPS: A new approach to the application of theorem proving to problem solving." *Artificial Intelligence*, 2(3-4):189-208.

Finin, T., Weber, J., Wiederhold, G., Genesereth, M. Fritzon, R., McKay, D., McGuire, J., Pelavin, R., Shapiro, S., and Beck, C., 1994. "Specification of the KQML Agent-Communication Language (DRAFT)." URL: <http://www.cs.umbc.edu/kqml/papers/kqmlspec.ps>.

Kambhampati, S., Cutkosky, M., Tenenbaum, M., Lee, S. H., 1991. "Combining Specialized Reasoners and General Purpose Planners: A case study." In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 199-205, July.

Lee, J. and Durfee, E. H., 1994. "Structured Circuit Semantics for Reactive Plan Execution Systems." In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1232-1237, July .

Lee, J., Huber, M. J., Durfee, E. H., and Kenny, P. G., 1994. "UM-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications." In *Proceedings of the AIAA/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space*, pages 842-849, March.

Lowrance, J. D., 1994. Evidential Reasoning with Gister-CL: A Manual. Artificial Intelligence Center, SRI International, 333 Ravenswood Avenue, Menlo Park, CA, February.

Smith, R. G., 1980. "The Contract Net Protocol: High-level communication and control in a distributed problem solver." *IEEE Trans. on Computers*, 29(12):1104-1113, December.

Wilkins, D. E., Myers, K. L., Lowrance, J. D., and Wesley, L. P., 1995. "Planning and reacting in uncertain and dynamic environments." *Journal of Experimental and Theoretical AI*, 7(1):197-227.

Wilkins, D. E. and Myers, K. L., 1995. "A common knowledge representation for plan generation and reactive execution." *Journal of Logic and Computation*, 5(6):731-761.

Wilkins, D. E., 1988. Practical Planning: Extending the Classical AI Planning Paradigm. Morgan Kaufmann, San Mateo.