N93-11937

On the Acquisition and Representation of Procedural Knowledge

T. Saito*, C. Ortiz, and R.B. Loftin**
Software Technology Branch (PT4)
NASA/Johnson Space Center
Houston, Texas

Historically knowledge acquisition has proven to be one of the greatest barriers to the development of intelligent systems. Current practice generally requires lengthy interactions between the expert whose knowledge is to be captured and the knowledge engineer whose responsibility is to acquire and represent the expert's knowledge in a useful form. Although much research has been devoted to the development of methodologies and computer software to aid in the capture and representation of some types of knowledge, little attention has been devoted to procedural knowledge. NASA personnel, on the other hand, frequently perform tasks that are primarily procedural in nature. In this paper we will review previous work in the field of knowledge acquisition and then focus on knowledge acquisition for procedural tasks with special attention devoted to the Navy's VISTA*** tool. We will describe the design and development of a system for the acquisition and representation of procedural knowledge—TARGET (Task Analysis and Rule Generation Tool). TARGET is intended as a tool that permits experts to visually describe procedural tasks and as a common medium for knowledge refinement by the expert and knowledge engineer. The system is designed to represent the acquired knowledge in the form of production rules. Systems such as TARGET have the potential to profoundly reduce the time, difficulties, and costs of developing knowledge-based systems for the performance of procedural tasks.

Introduction

In order to set the stage for a description of "yet another" knowledge acquisition tool a brief review of our evaluation of many existing knowledge acquisition tools is presented below . Following this review a Navy-developed program for task analysis is discussed in some detail. The remainder of the paper is devoted to a description of our own work in creating a knowledge acquisition tool specifically for procedural tasks. In addition to its role in aiding experts and knowledge engineers in the process of knowledge acquisition, this tool also provides mechanisms to support knowledge refinement and its representation in the form of production rules.

Processes and software designed to aid knowledge acquisition can be characterized by the nature of their delivery and implementation methods and styles as well as their ability to extract knowledge. As with other types of software tools and products, knowledge acquisition tools, either on the market or under development, seem to come in many "flavors", "colors," and "shapes". Knowledge acquisition tools operate as front-ends as well as embedded modules within existing expert systems and/or expert system development packages. Delivery platforms range from PCs and Macintosh® computers to RISC and symbolic workstations to IBM® plug-compatible mainframe systems. As the size of the platform grows, so often does the complexity and sophistication of the knowledge acquisition tool or module that it supports. Such commercial PC-based tools as GURU® (mdbs, Inc.), VP-Expert™ (Paperback Software), and Socrates™ (CIM Solutions) possess modest levels of knowledge acquisition features generally oriented toward knowledge representation in the form of rules. On the other hand, mainframe compatible ADS® (Aion Development System from Aion Corp.), Mercury KBE™ (Knowledge Base Enterprise from Artificial Intelligence Technologies) and Knowledge Shaper (Perceptrics, Inc.) all provide more elaborate knowledge acquisition functions and features (e.g., object-oriented representation and management) than the smaller systems. Tools under development such as KART (Knowledge Acquisition Reasoning Tool from IBM) and AKAT (Automated Knowledge Acquisition Tool from Harris Corp.) represent tools that operate within the less exotic workstation environments. However, other institutional knowledge acquisition tools such as Aquinas (Boeing) and KNACK (Carnegie-Mellon) provide sophisticated examples of the future of knowledge acquisition tools. DART (Design Alternative Reasoning Tool - Boeing ATC) is one spinoff tool whose design architecture was derived from its more renowned predecessor, Aquinas.

Operating system environments range from Microsoft DOS® (Disk Operating System) to the Macintosh® OS to UNIX® to IBM's VM® (Virtual Memory) and MVS® (Multi-Virtual System). Some knowledge acquisition tool developers have released their tools under several operating systems. Nextra, a commercial tool marketed by Neuron Data as a front end to the Nexpert Object expert system, functions under UNIX®, VM® (Virtual Machine - IBM), VMS® (Virtual Machine System from DEC®) and Macintosh® operating systems. Other tools, as they evolve, may well follow these same implementation and delivery strategies.

Various authoring tools have evolved to solve the problems associated with the creation of a specific expert system.[1]. Originally, most knowledge-acquisition-oriented tool designs were directed toward rating or categorizing problems or knowledge. To capture specific knowledge, the developer distinguishes between types of knowledge methods/approaches. Although sharing many of the same goals, the existing methodologies are numerous—ranging from frame modeling to case-based reasoning models to repertory-grid rating structures. The various knowledge types, addressed by these systems, range from semantic/taxonomic to declarative to procedural, affecting the design and performance decisions of researchers and implementers[2]. Knowledge representations, including frames, objects, rules and decision trees, are used to capture and execute

expertise. At this point, most would agree that no one tool accommodates all of the cognitive styles needed to gather the information/knowledge necessary for the creation of an expert system in one contiguous process. It is clear that viable standards have yet to be fully established and accepted.

To further complicate the issue, getting a subject matter expert's (SME) attention, time/commitment, help, and data sources is usually difficult at best. SMEs tend to differ in their communication abilities and styles, willingness to cooperate/availability, and degree of computer literacy, potentially affecting the overall success of the knowledge acquisition process[3]. The strategy of providing the SME with a tool that can be used to document his mission(s) or task(s), on his own and within his schedule, would resolve some of the traditional headaches associated with a knowledge engineer constantly "hovering over" an SME. However, the disadvantages of such a strategy may be the lack of positive reinforcement or external motivation (i.e., SMEs might put off documenting their task/mission unless periodically reminded or encouraged.).

As computer hardware power evolves, more latitude in presentation methods will be available. Visual conception and communication of abstract information will become more common. The strategic fusion of graphical display (bit-map, meta-graphic, etc.) and graphical input device (mouse, light-pen, trackball, etc.) technologies will facilitate visual, as well as textual representation, of knowledge[4]. Drawing tools already allow the user to produce and manipulate complex graphics. The role of these tools can also combine with organizational algorithms to create more intelligent diagrams, flow charts, interactive decision trees, etc. With users becoming more adept at using systems with pictorial modeling capabilities, the mode of knowledge acquisition will also benefit from such advances.

Procedural knowledge acquisition via task analysis is a reasonable candidate for graphical representation modes. Decomposing a complex set of steps that makes up a specific mission or task requires cognitive visualization and the ability to formulate and reformulate the decomposition of those steps or actions. The specific heuristic procedures that most SMEs employ share certain levels of organization and recall[5]. The path in which a procedure evolves starts with specific agendas and goals. The last or final action of reaching or satisfying those actual goals would end the procedure. On the other hand, any actions that would restart a process (loop) would occur before the goal oriented or last action. Decisions may be made during a task that direct the expert along alternative paths which may or may not be taken in other performances of the same task. In cases where the processes offer one or more options to complete a task, the process diverges into as many paths necessary to meet the optional requirements. Each path would contain specific values for technique evaluation or other modes of feedback. These types of complexities lend themselves to representation in a visual form. Below we explore two attempts to provide just such a visual metaphor for knowledge acquisition and representation.

VISTA: The Graphical Predecessor

Of over twenty expert systems assessed by NASA/Johnson Space Center, a graphically-oriented task analysis tool developed by Robert Ahlers of the Naval Training Systems Center (NTSC) showed the most promise for addressing procedural knowledge acquisition. VISTA (Visual Interactive System for Task Analysis) has proven to be a tool an expert can use to easily define and document specified tasks in a "comfortable" and modifiable form.

The NTSC in Orlando, Fl., participating in a governmental tri-service project with the Air Force Human Resources Lab (HRL) and the Army Research Institute (ARI), directed a project that produced a prototype knowledge acquisition tool called KA-1 (Knowledge Acquisition-1). KA-1 was first designed and implemented in Lisp within the Symbolics/Genera 6.x environment. Recognizing the appropriateness of implementing their own knowledge acquisition strategies to acquire task or procedural knowledge, NTSC, after the end of the KA-1 project, started work on its first prototype (named AFEAT—Automated Front-End Analysis Tool). NTSC later redesigned and released an enhanced version, renaming the knowledge acquisition tool VISTA. Both of these applications have been developed on PC platforms using Smalltalk-V.

The NTSC designed VISTA to compose task lists and hierarchies from a graphical representation of a knowledge base. VISTA identifies subsets of tasks meeting specified selection criteria, training objectives, and/or personal performance profiles. VISTA's strategy, permitting the SME to establish task hierarchies and relationships, yields a final report of procedural step data with corresponding conditions and criteria.

The VISTA system is a graphical user interface (GUI) oriented tool that builds box-flow style representations that a user can utilize to document various task levels. VISTA is essentially a qualitative analysis tool directed toward task and procedure decomposition into their component parts, in a largely top-down style. In knowledge acquisition mode, the knowledge engineer and SME could conduct the decomposition process together or the SME could essentially use the tool without direct knowledge engineer support. The system also has a knack for allowing a group of experts to huddle in front of a VISTA screen for consensus verification and modification.

VISTA maintains a fragile balance between ease of use and design complexity/intricacy. Although VISTA does not possess the "bells and whistles" of more sophisticated systems like Aquinas and Protege, it provides enough knowledge modeling (procedural/declarative) support to allow the SME or knowledge engineer to build a fairly elaborate knowledge base without sacrificing the attractiveness of its user interface.

VISTA provides a "windows-icon-mouse-pointing" (christened, WIMP) interface environment based on a grid-marked Work Area in which the user builds task networks. The WIMP approach facilitates the rapid selection and execution of system functions to minimize user keystrokes. The Work Area is lined on three sides with icon and menu selectable functions:

1) System Command Menu Bar (top)
2) Function/Graphics Icons (left side)
3) Message/Explanation Bar (bottom)

Although VISTA provides no bona fide compilation facilities to check the knowledge base for completeness or accuracy, it does utilize some of the Smalltalk inspection and reporting features to help the user confirm the knowledge input into the system. VISTA provides a windowed environment through which decomposition can be organized and recorded. Ultimately the user, knowledge engineer or SME, is responsible for the overall quality checking of the knowledge base before its representation in or transfer to other applications. VISTA's report facilities offer some assistance in this quality checking process. Reports can be generated to provide moderately high-level feedback to the knowledge engineer and SME. VISTA produces the following reports:

- Hierarchical Statistics: counts nodes and subnets at each level
- Task hierarchy: keeps a sequential/hierarchical account of tasks
- Input grammar: maintains various types of component titles in categorized form
- Notecards: keeps notes on conditions, states or other user-supplied details
- Highlighted tasks: accounts for subnet levels

VISTA supports the identification and conceptualization phases of knowledge acquisition with its network approach to knowledge representation. Duties, tasks/subtasks, or steps/substeps within a process can be defined, documented and structured to reflect these relationships to other duties, tasks/subtasks, or steps/substeps.

Given its developmental state, VISTA provides a fairly comprehensive mechanism for generating simple representations at the very first knowledge acquisition session. The next sessions may be used to embellish what has already been elicited, or to create new or modified versions of the knowledge base. The VISTA knowledge acquisition interface gives the user the freedom to generate as complex a hierarchy of knowledge as necessary. However, the disadvantage to such freedom is the ability to create a completely abstract knowledge base with relatively little standards for input. Some guiding controls from the VISTA interface could provide structure to the knowledge acquisition process and greatly enhance the ability of the user to create a "useful" knowledge base.

Although VISTA has a grammar component within its facilities, its ability to correlate domain terms and/or concepts is limited. The open-endedness of the tool design allows key domain definitions to be specifically addressed within a notecard-like management facility and/or defined in a box as a subpart within a process. VISTA does not offer a true lexical facility or natural language interface to accommodate concept definitions.

In addition to the creation of new knowledge base versions, the system also offers the ability to mesh existing VISTA knowledge bases into the current version for incremental enhancements. As new ideas are evolved, the VISTA interface allows the integration of old and new knowledge bases for processing. Task progressions can be devised from scratch or from existing task lists or progressions. A VISTA grammar (task verbs and verb-objects) must be defined in order for new tasks to be created. Creating a grammar from scratch involves building and naming a number of tasks. A significant improvement would support the automatic extraction of a grammar from an existing task description and its integration into a new or different network.

The TARGET Approach

NASA Environment and Needs

The National Aeronautics and Space Administration commits large funding and manpower effort to training new and existing personnel. New recruits are trained to carry out tasks for which they were hired. Existing staff must be trained or retrained to upgrade/update abilities to perform current and new tasks. Significant numbers of training methodologies are utilized involving training manuals, formal classes, instructional computer programs, simulations and on-the-job training. On-the-job training is usually the most effective training mechanism for the more complex tasks requiring substantial autonomy on the part of the task performer. However, this training style is also the most expensive and impractical where trainees significantly outnumber experienced staff.

The effort of educating and training NASA astronauts, flight controllers, and other ground support personnel has generally required extensive on-the-job experience in order for individuals to acquire the knowledge and skills necessary for acceptable performance and/or certification. Current flight schedules, combined with the loss of experienced personnel to retirement/transfer, have significantly reduced the ability of traditional training techniques to produce an adequate number of trained personnel[6]. Recently, it has been shown that workstation-based, intelligent computer-aided training (ICAT) systems can deliver intensive, personalized training to large numbers of trainees, independent of integrated simulations[7]. Such systems can noticeably reduce the amount of training time needed to achieve acceptable levels of performance. After over four years of experience in building such systems, the developers have concluded that the greatest barrier to the large-scale, efficient production of ICAT systems is the extent and difficulty of the knowledge acquisition process. For some time an effort has been underway to create a software tool to aid in the capture of mission support procedural knowledge both to preserve the existing corporate

knowledge base and to assist in the development of decision support expert systems and ICAT systems.

The remainder of this paper details the design and implementation of a knowledge acquisition system tailored to the acquisition and representation of procedural knowledge associated with the performance of complex tasks. The goal of this effort has been the production of a system with an easy-to-learn and "comfortable" user interface that provides powerful mechanisms for the visual expression of procedural knowledge. The ultimate goal of this work is the expression of acquired knowledge in the form of production rules to facilitate the use of the acquired knowledge in expert systems for mission support and training.

TARGET and Design Strategy

Attempting to strike that delicate balance between nonprogrammer usability, design sophistication, and hardware universality, the Task Analysis Rule GEnerating Tool (TARGET) is designed to provide a knowledge acquisition environment for users of commonly-available computer systems (IBM® PCs and Apple© Macintoshes®). The forte of TARGET is the gathering of task or procedural knowledge to be expressed and analyzed graphically as well as contextually. TARGET provides users the ability to graphically decompose a task or mission using a box-flow presentation/manipulation style within a windowed environment.

TARGET is designed to let the SME to start documenting their job or task with minimal training in its use and no absolute need for knowledge engineer intervention. If the SME is not able to find time to work on the knowledge acquisition process alone, TARGET does allow the knowledge engineer and SME to work together in iterative sessions. TARGET is tailored to accommodate a wide range of users, from the novice to the expert. With TARGET, users can develop a discrete representation of tasks and their subtasks within their domains[8]. The system then manages the information entered and represents the knowledge in a "top-down" reporting format that can then be used for rule induction and generation.

Action Descriptions within TARGET

Within the NASA/Johnson Space Center environment, CLIPS (C-Language Integrated Production System) is widely used as an expert system development and delivery vehicle. In order to support the development of intelligent computer-aided training (ICAT) systems, TARGET will implement its rule representations using the rule types and structures originally developed for ICAT systems.

Within the ICAT metaphor the overall mission or task is decomposed into sets of tasks/subtasks that are termed actions. For most effective use, actions are expressed, within reason, at the lowest possible level. At any point in an ICAT training session, the expert expects the trainee to perform a valid action. Each valid action, as defined by the expert, is represented as a CLIPS fact in the following pattern:

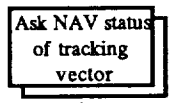
(message-E-to-I <step number> <action type> <argument> <argument> ...)

An action itself comprises at least two <argument> fields that define one single action decomposed into a hierarchical structure of two or more subactions of the form (<action> <argument>). Each <argument> may itself be an <action> at the next lower level. For example, (arg1 arg2 ... argn) can be decomposed into at least two levels: (arg1 arg2) and (arg2 . . . argn). The first pair, (arg1 arg2), is an (<action> <argument>) pair at the top level where the action arg1 has one argument, arg2. In turn, (arg2 ... argn) is another (<action> <argument>) pair at the second level where arg2 has one or more arguments, depending on the value of n. The structure for each action may be different and the number of arguments that belong with each action is variable. The expert is free to decompose the actions and arguments into hierarchies that fit his or her specific domain.

TARGET supports three action types: required, optional and flexible (as defined in the ICAT architecture). Required actions are necessary task(s) and/or subtask(s) performed at specific points in a mission or task. These actions are then asserted to the factlist by the expert module. The following pattern reflects the expected action at a given point in time:

(message-E-to-I <step number> require <argument> <argument> ...)

For example, Figure 1 shows the TARGET visual representation of the action: "Ask the Navigation Officer to give the status (good or bad) of the current state vector (describing a specific orbiting vehicle) obtained from ground- or space-based tracking stations." Following this representation is the CLIPS fact produced to represent the same knowledge.



(message-E-to-I 150 require request nav nav-status)
Fig. 1 Required Task

The ICAT architecture prohibits the performance of further actions if the current action cannot be achieved. In cases where more than one alternative precedes a required action, the alternatives may not all be equivalent. Although they accomplish the same goal, one method may be more suitable than another in a given context. TARGET distinguishes between these alternatives by labeling the "less desirable" as an "other" required action, as opposed to the previous "require" action. In the ICAT architecture, only one "require" action, at any point, exists, with the rest, if any, being "other" actions. When more than one alternative exists for a required action (and none are more suitable than the others), all alternatives are labeled as "other" actions. Other actions are represented by TARGET as:

(message-E-to-I <step number> other <argument> <argument> ...)

Optional actions are those recommended by the SME as good action or technique preferences in a given context. However, the trainee is not prohibited from advancing in the training session if the action is disregarded. Optional actions should not be confused with situations where more than one required action exists at a given time. Optional actions are represented by the following pattern:

(message-E-to-I <step number> optional <argument> <argument> ...)

Figure 2 shows the TARGET visual representation of an optional task.

Fig. 2 Optional Task

Since optional actions do not halt the training session within the ICAT architecture, even if they are ignored by the trainee, <step number> could be any future step beginning from the current context. For example, if the current step is 20 and the expert asserts (message-E-to-I 50 optional ...), that particular optional action remains valid from step 20 until step 50. Validity checks for certain values can be displayed in the user interface. For example, a fact of the form

(message-E-to-I 260 optional check-display vector-comp MET)

indicates that the expert recommends that the Mission Elapsed Time field within the Vector Comparison Table display be checked at any time from the current step through step number 260.

TARGET will internally manage flexible actions, i.e., actions that must be completed by predefined times (similar to a required action but operable over a span of time or steps). Flexible actions operate identically to optional actions inside specified range of steps. If the deadline arrives and the flexible action still has not been completed, it will be changed into a required action. Flexible actions take the following pattern:

(message-E-to-I <step number> flexible <argument> <argument> ...)

TARGET, following the current ICAT architecture, is designed to deal with a series of tasks/subtasks that are performed procedurally or in steps. Steps are defined as the progression from one required action to the next. Steps are represented by numerals and their values increase with the progression of the task.

Task-Action Concepts (Building Blocks)

TARGET employs a free-form flow charting concept. SME or knowledge engineers can explain procedural processes by the use of various icons arranged in the TARGET work area. The tasks can then be linked together using directed arcs to show procedural flow. The task icons are separated into five categories.

The first category consists of actions that are required to complete a process. The required actions are denoted by using a blue (or white, for monochrome monitors) rectangular box on the screen. On the other hand, optional tasks are represented using grey rectangles. The use of color for representing various forms of tasks has been reduced to just two, grey for optional tasks and blue (white) for all others. The shape of the task box is the most important key to determining its function. By using shapes and not colors, TARGET, although designed for color systems, also works reasonably well on monochrome systems.

Fig. 3 Required and Optional Tasks

Hexagonal-shaped structures are used to show processes where a decision is needed. The connections leaving the decision structures are labeled to show what action is to be taken. For example, a decision structure may ask if a process has been completed. This decision may be linked to two other tasks (see Figure 4). The first arc leaving the decision is labeled "YES" and the other is labeled "NO". The arc labels represent the only possible answers allowed by the decision structure.

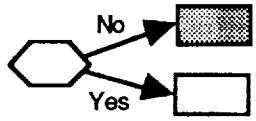


Fig. 4 Decision Task

The fourth basic structure that TARGET provides is a control structure. Control structures are used as a "goto" or looping mechanism. Controls can only jump to other tasks that are on the current layer (see discussion of layers below) and may not jump directly to other controls. Control tasks are denoted by using an ellipse to distinguish them from the other forms of tasks.

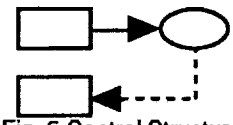


Fig. 5 Control Structure

The fifth task type is called a Discretionary or D-Path (Figure 6). D-Paths, defined as alternate paths when taken, will not affect the final outcome of the procedure. D-Paths are used when decision structures are too strong, but a two or more options exist that can be explored. Eating breakfast is an example of a D-Path. One could eat a large breakfast, a small breakfast or skip breakfast entirely and would not impact the process of "getting ready to go to work".

D-Paths are defined to start when two or more tasks branch off from a single required or optional task. (Decision structures are the only other task structures

that allow for multiple branching.) Tasks within a given D-Path chain must be optional or required tasks and cannot be connected to more than one task at a time. This rule insures that no branching occurs from outside or within the D-path itself. All D-Paths will end at a single common (optional or required) task. The final specification for D-Path structures regards the preference value for taking one choice over another.

A numerical value is given to each D-path and is termed that path's preference value. The preference value is a number that ranks each path choice from highest to lowest preference. The path with the highest numerical value is considered to be the "best" path by the SME. Other paths represent processes that may be performed but are not regarded as optimal by the SME.

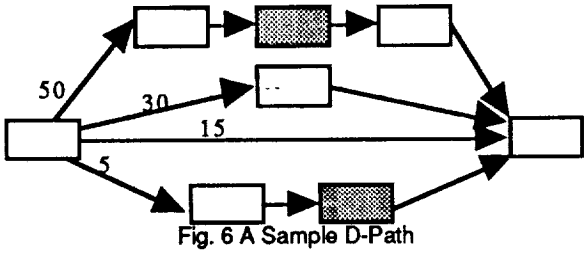


Fig. 6 A Sample D-Path

In addition to the two-dimensional flow, TARGET provides users with the ability to decompose required and optional tasks into a task hierarchy. Task decomposition is used to make complex tasks easier to understand and complicated layers smaller. Tasks that are decomposed are characterized by displaying a shadow and have "child" tasks associated with them. The child tasks are not seen at the "parent" layer but can be found when traversing down the task hierarchy into lower levels (Figure 7).

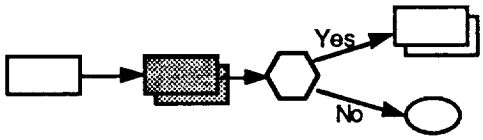


Fig. 7 Task Chain Showing Child Tasks

Navigating through a task hierarchy is done simply by placing the cursor on a leveled task and double-clicking. The screen will clear and be rebuilt showing the selected child tasks. Moving up in the task hierarchy is just as simple, it is done by double-clicking on an area where there is no task. The screen will clear and tasks from the level above will be displayed.

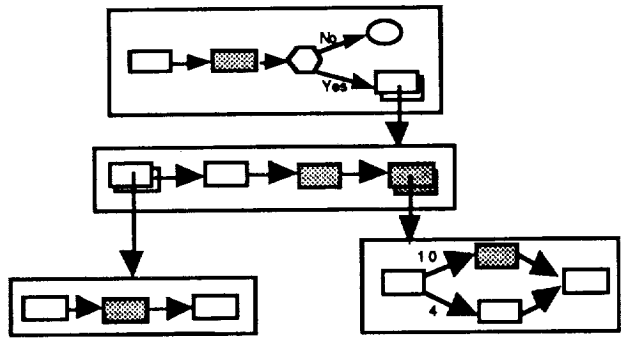


Fig. 8 Decomposition of a Task Hierarchy

Input Features

To facilitate its implementation on many hardware platforms, TARGET was developed around the one-button mouse concept. Using a single mouse button was determined to be the best way of porting the TARGET program to other platforms such as a Macintosh™ computer. Double-clicking the mouse button will move up or down a task hierarchy and single-clicking will activate all other functions. Since single-clicking on a one-button mouse is limiting, TARGET, like VISTA, was designed to be a cursor-driven program.

Users may select a desired function from the main menu, toolbox or keyboard (Figure 9). The cursor will change to reflect the operation that is to be performed. Located on the left-hand side of the screen, the toolbox is one of the most important user interface functions provided by TARGET. Using the toolbox, one can directly manipulate task structures on the current screen by editing, creating, deleting, moving, linking, and un-linking tasks. Help and a display of the task hierarchy may also be accessed through this toolbox.

Menu selections provide an interface for file I/O, for task manipulation, to set user preferences for how TARGET displays the task flow, to control output, and to obtain help. File I/O deals with the reading and writing of TARGET files (.TGT) to and from disk storage. In addition to the .TGT files, .DMN files (generated from VISTA) can also be directly imported. The EDIT menu helps to find tasks as well as permit the moving of tasks up and down the task hierarchy.

The VIEW menu is especially helpful. Using view, one can display tasks in a horizontal or vertical orientation. Optional tasks and the background grid can be hidden or displayed. The entire layout may be redrawn using automatic task placement. HELP provides information regarding TARGET questions as well as an explanation of the help facility itself. OUTPUT provides a mechanism to generate reports or simple rule structures on the screen, printer, or in ASCII format on disks.

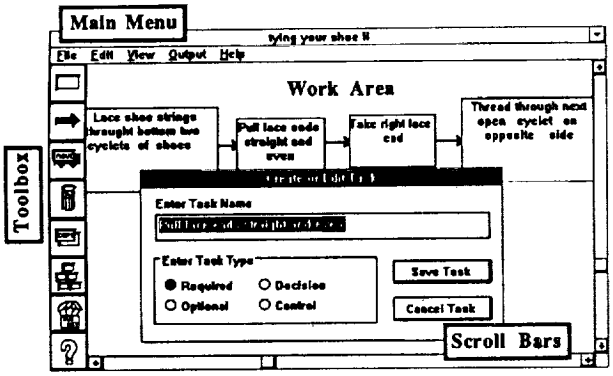


Fig. 9 TARGET User Interface

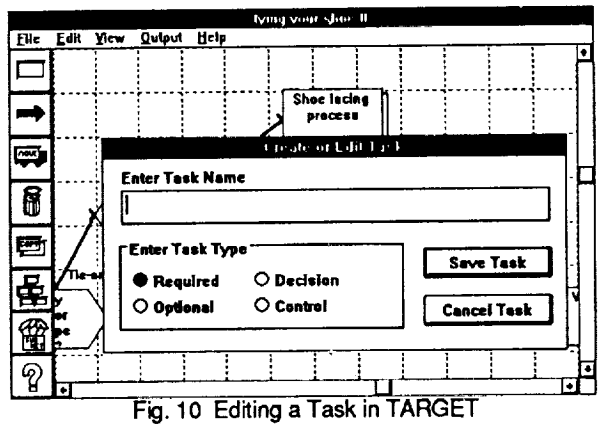


Fig. 10 Editing a Task in TARGET

Documentation Strategies

TARGET's overall visual representation style employs directed graph strategies combined with the box-flow and entity-relationship diagram structures. Task hierarchies require large amounts of layout space while visual analysis is performed[9]. Previously, where most task analysis efforts were done by paper and pencil, the graphical medium provides a facility to organize and manipulate various levels of tasks[10].

Decomposing the task into single steps is the recommended strategy within the TARGET environment. TARGET provides features for organizing selected missions or tasks into distinct hierarchical levels. Individual steps can be laid out and studied interactively by the SME and/or knowledge expert. Immediate graphical feedback for analysis is possible. TARGET will also accommodate the text editing process as well as maintain box/task step order.

The basic human factors rule-of-thumb recommends the "7 plus-or-minus 2" boxes/tasks per screen for creating hierarchical box diagrams. When documenting complex tasks, however, the user should not be limited (especially in his initial effort) to a specific number of boxes[11]. What counts is the ability to keep track of the task networks being developed. Within TARGET, however, there is currently a limit, within a specific level, of 100 boxes.

TARGET's documentation input requirements pose no constraints at this time. The "Enter Task Name" function allows free-form input. However, within the TARGET environment, addressing another person would be the most effective way to phrase a task. Communication with that someone should be done in first person with an implied "you". For example, "You should":

- Turn on light
- Flip switch up
- Report to Commander
- Allocate extended memory to DOS
- Shut down all systems
- Fire missile

Once accustomed to the approach TARGET takes in documenting task flows, the domain expert can establish multiple mission/task levels hierarchically using his or her own classification or categorization strategies to express specific procedures. TARGET provides a traversing mechanism in which the user can create, maintain, and check task subnetworks with minimal keystrokes.

TARGET will provide a template, breaking down the task steps into sequential dependencies, to facilitate the construction of the left-hand and right-hand sides of a CLIPS rule. For example, a step will have to be performed before another step is executed. The previous step becomes the dependency or left-hand condition for the right-hand or current step. A task box will encapsulate all of the necessary information that makes that particular task significant, whether it be required or optional (Figure 12). The user need not be concerned with "keeping score" on these issues, TARGET, through its graphical maintenance facility, will manage the top-down representation and coordination of tasks.

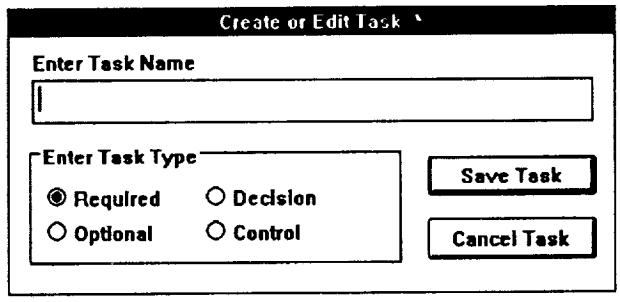


Fig. 11 Create/Edit Dialog

Within the CLIPS rule environment, steps documented in the TARGET style will automatically support the rule composition process. Each step will operate as part of the left-hand or right-hand side of a rule where applicable. TARGET manages context and sequence from the user interface.

As knowledge acquisition iterations progress, the task steps will have to evolve into executable forms of knowledge for implementation within an expert system environment. Generating rule representations induced

from a top-down, sequential list assumes that high-level verification has been performed. TARGET will allow specific levels of manual, visual, and list verifications throughout the task hierarchies. However, with TARGET, verification and validation will also have to be carried out in CLIPS on the generated rule-base since TARGET has no automated verification and validation component at this point.

Rule Control Structure Design

For rule propagation, TARGET will implement the ICAT control structure guidelines where rules employ two parts, the data and the task process. In general, facts will be data equivalents. The process or task step will be manifested in the form of a rule. Facts will be used to specify the current context or environment and to represent the "actions" taken by a user or the expert system. All fact combinations must be matched against rule patterns to determine which rules may fire. Then, one rule would be used to generate potentially executable actions and update the fact base. This process will be repeated until no rules can fire. More specifically, the ICAT control structure revolves around two significant points.

1) Message passing protocols are used by independent rule sets for communication and
2) Tasks are procedural/step-by-step in structure.

Report Generation with TARGET

TARGET provides several reporting mechanisms. The first report that the average user will experience is the graphical representation of tasks in a task hierarchy. Figure 12 represents a sample graphical layer that would be produced by a user in the graphical interface. Task hierarchies in a textual form can also be generated from TARGET to provide another look at the procedural process being described (Figure 13). Files that store all the information TARGET needs to build task networks are called .TGT files (Figure 14). .TGT files are closely related to the .DMN files created by VISTA. The file similarity is intended to expedite the conversion from VISTA files into TARGET representations. The last figure (Figure 15) in this section shows CLIPS rules written in final form produced by TARGET.

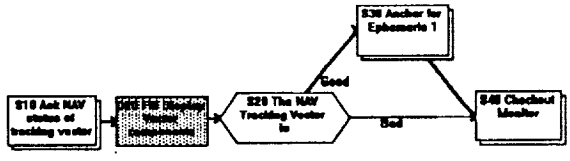


Fig. 12 Sample Task Chain Described Graphically in TARGET

1.0 Request NAV tracking vector status
- 1.1 (f) "toggle-offset-sign"
- 1.2 (f) "enter orbiter PKM offset in Worksheet"
- 1.3 (f) "enter deploy separation rate in Worksheet"
- 1.4 (f) "enter deploy spin axis declination in Worksheet"
- 1.5 (f) "enter PKM offset in Deploy Worksheet"
- 1.6 (f) "enter deploy relative right Ascension in Worksheet"
- 1.7 (f) "enter Z component of deploy delta V in Worksheet"
- 1.8 (f) "inform SDP DYN of payload data"
- 1.9 (f) "enter target PKM offset in Worksheet"

2.0 (o) check Vector Comparison Table
3.0 The NAV Tracking Vector is (Good)
- 4.0 Anchor ephemeris for shuttle
  - 4.1 (o) inspect new ephemerides
  - 4.2 (o) request Vector Comparison Table
  - 4.3 (o) request Trajectory Digitals display for ascending node
  - 4.4 (o) request Trajectory Digitals for descending node

3.0 The NAV Tracking Vector is (Bad)
5.0 request Checkout Monitor w/o Traj Digitals

Fig. 13 Task Hierarchy Description Generated from TARGET

```
REQUIRED
S10 Request NAV tracking vector status
2                                   <Task Key>
273,139                             <X,Y Pos>
1                                   <Parent>
11,10,9,8,7,6,5,4,3                 <Children>
13                                  <Connected_To>
OPTIONAL
O20 Fill Display Vector components
13                                  <Task Key>
402,140                             <X,Y Pos>
1                                   <Parent>
14                                  <Connected_To>
DECISION
S20 The NAV Vector is Good
14                                  <Task Key>
545,136                             <X,Y Pos>
1                                   <Parent>
Good
18                                  <Connected_To>
Bad
12                                  <Connected_To>
REQUIRED
S30 anchor ephemeris for shuttle
```

```
18                          <Task Key>
691,55                      <X,Y Pos>
1                           <Parent>
17,15                       <Children>
12                          <Connected_To>
```

Fig. 14 .TGT File Derived from a VISTA .DMN File

```
(defrule s10-get-nav-info "Request NAV tracking vector
status"
        (step ?s&10)
        (checkpoint expert)
=>
        (assert (message-E-to-I ?s require req-nav get-
        nav-status)
                (next-step 20))
)

(defrule o20-display-vector-comp "Fill Display Vector
components"
        (step ?s&20)
        (checkpoint expert)
 =>
        (assert
        (message-E-to-I ?s optional check-display dis-
        vector-comp VC_CUR_TIME)
        (message-E-to-I ?s optional check-display dis-
        vector-comp VC_CUR_DESC)
        (message-E-to-I ?s optional check-display dis-
        vector-comp VC_CUR_A)
        (message-E-to-I ?s optional check-display dis-
        vector-comp VC_3RD_TIME)
        (message-E-to-I ?s optional check-display dis-
        vector-comp VC_3RD_DESC)
        (message-E-to-I ?s optional check-display dis-
        vector-comp VC_3RD_A)
        (message-E-to-I ?s optional check-display dis-
        vector-comp VC_3RD_V))
)

(defrule s20-good-nav "The NAV Vector is Good"
        (step ?s&20)
        (environment 0 nav-tracking good)
        (checkpoint expert)
=>
        (assert (message-E-to-I ?s require req-nav put-
        nav-in-slot v39))
        (assert (next-step 30))
)

(defrule s20-bad-nav "The NAV Vector is Bad"
        ?f1 <- (step ?s&20)
        (environment 0 nav-tracking no-good)
        ?f2 <- (last-step ?)
        (checkpoint expert)
=>
        (retract ?f1 ?f2)
        (assert (step 40)
                (last-step ?s))
)
```

Figure 15. TARGET Rule Generated File

Conclusion

TARGET will have the capability to significantly impact the development of ICAT systems as well as the development of other intelligent systems. For any procedural knowledge acquisition task TARGET can enhance the ability of the expert to visualize and organize a task or process. We believe that procedural visualization of this type will become more popular as more tools with organizational diagnosis capabilities evolve[12].

As knowledge acquisition, as a discipline within artificial intelligence, evolves, more tools to assist in the knowledge acquisition process will also become available in useful forms. TARGET, and tools like TARGET, will be employed within their own "niche" and will also be integrated with other methodologies in the future. Although TARGET models currently sequence within the task hierarchy structure for rule induction, we will dedicate additional efforts to encapsulating more peripheral knowledge into the various steps within the network. Particularly, for TARGET, such issues as gathering artifact data, selected action rationale, and interactive verification and validation of rules will be addressed in the future.

---

* Computer Sciences Corporation

** University of Houston - Downtown Campus
(bloftin@nasamail.nasa.gov)

*** VISTA (Visual Interactive System for Task Analysis)
Naval Systems Training Center, Orlando, Fl
Robert Ahlers, Project Manager

[1]Boose, J. H. (1989). A survey of knowledge acquisition techniques and tools. *Knowledge Acquisition*, March, 1 (1), 3-37.

[2]Gaines, B. R. (1988). An overview of knowledge-acquisition and transfer. in *Knowledge Acquisition for Knowledge-Based Systems*, Gaines, B. R & Boose, J. H., Eds., *Knowledge-Based Systems, Vol.1*, New York: Academic Press, 3-22.

[3]Littman, D. C. (1988). Modelling human expertise in knowledge engineering: some preliminary observations. in *Knowledge Acquisition for Knowledge-Based Systems*, Gaines, B. R. & Boose, J. H., Eds., *Knowledge-Based Systems*, Vol.1, New York: Academic Press, 93-104.

[4]Messinger, E. B., Rowe, L. A. & Henry, R. R. (1991). A divide-and-conquer algorithm for the layout of large directed graphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21 (1), 1-11.

[5]de Kleer, J., Doyle, J., Steele, G. L., Jr. & Sussman, G. J. (1985). AMORD: Explicit Control of Reasoning. in *Readings in Knowledge Representation*, Brachman, R.

J. & Levesque, H. J., Eds., Los Altos, CA: Morgan-Kaufmann Publishers, Inc., 345-355.

[6]Loftin, R. B., Wang, L., Baffes, L. & Hua, G. (1988). An Intelligent Training System for Space Shuttle Flight Controllers. *Proceedings of the 1988 Goddard Conference on Space Applications of Artificial Intelligence*, held May 24, 1988, at NASA/Goddard Space Flight Center, Greenbelt, Md, 3-10.

[7]Loftin, R. B., Wang, L., Baffes, P. & Hua, L. (1989). An Intelligent System for Training Space Shuttle Flight Controllers in Satellite Deployment Procedures. *Machine-Mediated Learning*, 3, 43-47.

[8]Payne, S., & Green, T. (1986). Task -action grammars: a model of the mental representation of task languages. *Human-Computer Interaction*, 2, 93-133.

[9]Wilson, M. (1989). Task models for knowledge elicitation. In *Knowledge Elicitation: Principles, Techniques and Applications*, Diaper, D., Ed., New York: Ellis Horwood, 197-219.

[10]Bylander, T. & Chandrasekaran, B. (1987). Generic tasks for knowledge-based reasoning: the 'right' level of abstraction for knowledge acquisition, *International Journal of Man-Machine Studies*, 26, 231-243.

[11]Wilson, M. D., Barnard, P. & MacLean, A. (1985). Analysing the learning of command sequences in a menu system. Johnson, P., & Cook, S., Eds., *People and Computers: Designing the Interface*, Cambridge: Cambridge University Press, 89-102.

[12]Akscyn, R. M., McCracken, D. L. & Yoder, E. A. (1988). "KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations", *Communications of the ACM*, July, 31 (7), 820-834.