Image Cover Sheet

CLASSIFICATION	SYSTEM NUMBER
UNCLASSIFIED	511955

TITLE

Intelligent Analysis in the LOCATE Workspace Layout Tool - Final Report

System Number:

Patron Number:

Requester:

Notes:

DSIS Use only:

Deliver to:

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUL 1999		**00-00-1999 to 00-00-1999**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Intelligent Analysis in the LOCATE Workspace Layout Tool	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Artificial Intelligence Management and Development Cooperation,206 Keewatin Avenue,Toronto, ON M4P 1Z8 Canada,**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

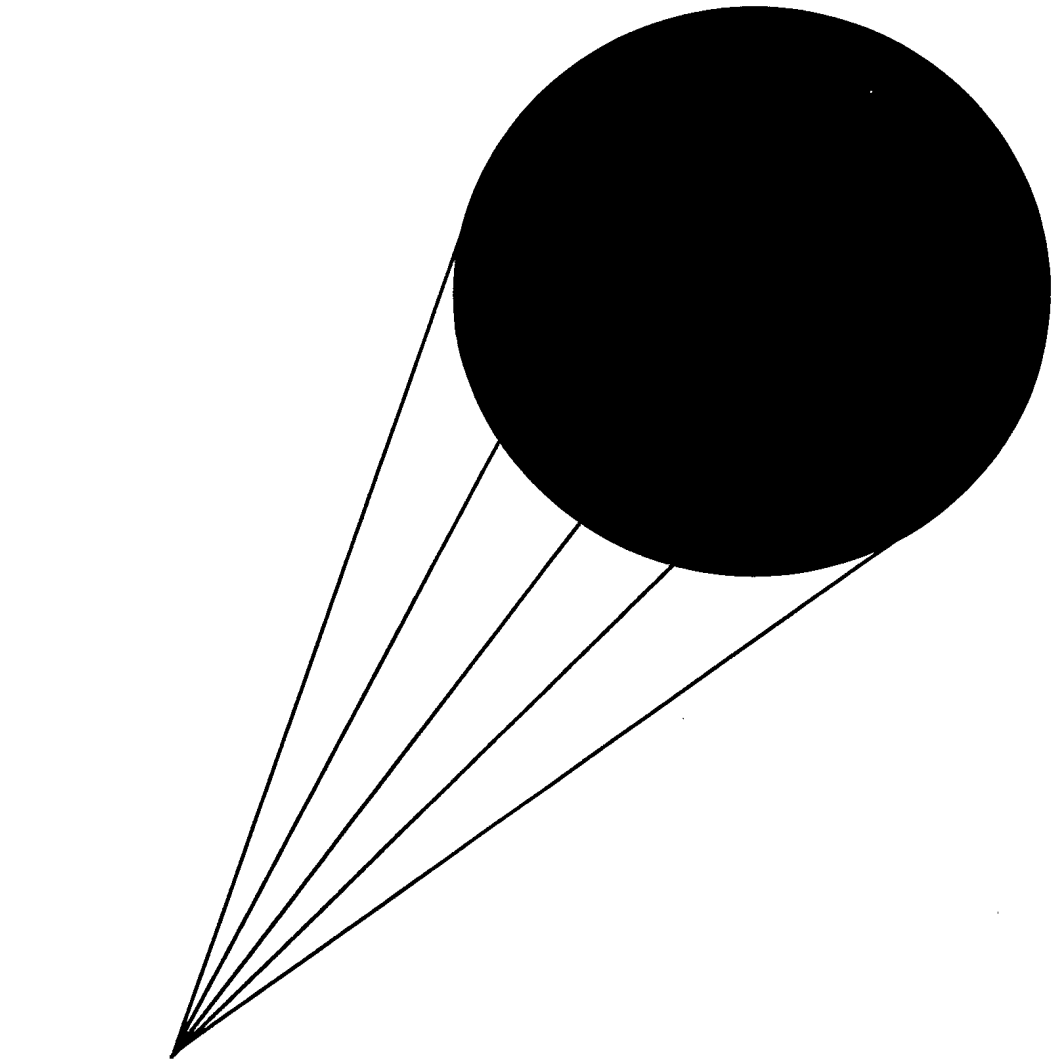
13. SUPPLEMENTARY NOTES

14. ABSTRACT

The two-fold purpose of this work was to extend the basic functionality of the LOCATE Workspace Layout Tool and to support users in intelligent ways in the analysis of their designs. Intelligent aiding during a LOCATE analysis was achieved principally through the addition of an optimisation procedure and the incorporation of qualitative rules to help guide the optimisation process. Further, considerable work was done on laying a framework for inferring high-level user goals and plans through the tracking of user interface actions. Part of the work involved the implementation of task, user and system (self) models that provide ways of monitoring LOCATE's understanding of what the user is doing, what he or she knows and how that information might be used to aid users in a variety of contexts. A number of features were added to LOCATE's basic functionality including a manual grouping procedure, tabbed windows containing object attributes, and link function and priority weight information. Prior work on easing the interpretative burden for users with respect to LOCATE analyses was augmented through organising the results in LOCATE's Cost Function History Window and through the creation of an example overview summary for inspection and modification of link function data. Future work will round out standard LOCATE features by adding alignment options a snap-to grid and extending work on links to other CAD packages. Further work on providing intelligent analysis should include additional qualitative rules for guiding optimisation, smart summaries of LOCATE results and features that are likely to emerge from tracking user actions and inferences of user goals and plans.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**93**	

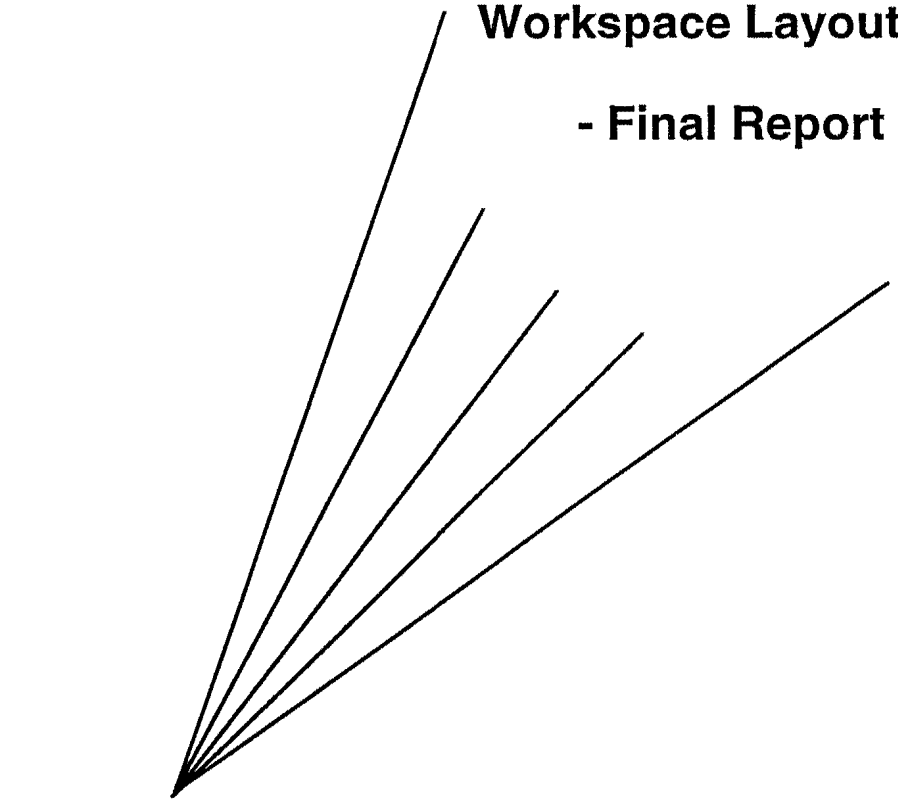


Artificial Intelligence
Management and Development Corporation

Prepared for: Public Works and Government Services
Prepared by: Artificial Intelligence Management and Development Corporation
AIM (AC197, July, 1999)
Contract: W7711-8-7476/001/SRV
"Intelligent Analysis in the *LOCATE* Workspace Layout Tool"
Scientific Authority:
Mr. Keith C. Hendy
Simulation and Modelling for Acquisition, Rehearsal and Training (SMART)
Defence and Civil Institute of Environmental Medicine

Intelligent Analysis in the *LOCATE* Workspace Layout Tool

- Final Report -



Artificial Intelligence
Management and Development Corporation
206 Keewatin Ave., Toronto, Ontario M4P 1Z8

Table of Contents

FIGURES

Abstract

The two-fold purpose of this work was to extend the basic functionality of the *LOCATE* Workspace Layout Tool and to support users in intelligent ways in the analysis of their designs.

Intelligent aiding during a LOCATE analysis was achieved principally through the addition of an optimisation procedure and the incorporation of qualitative rules to help guide the optimisation process. Further, considerable work was done on laying a framework for inferring high-level user goals and plans through the tracking of user interface actions. Part of the work involved the implementation of task, user and system (self) models that provide ways of monitoring LOCATE's understanding of what the user is doing, what he or she knows and how that information might be used to aid users in a variety of contexts.

A number of features were added to LOCATE's basic functionality including a manual grouping procedure, tabbed windows containing object attributes, and link function and priority weight information. Prior work on easing the interpretative burden for users with respect to LOCATE analyses was augmented through organising the results in LOCATE's Cost Function History Window and through the creation of an example overview summary for inspection and modification of link function data.

Future work will round out standard LOCATE features by adding alignment options, a snap-to grid and extending work on links to other CAD packages. Further work on providing intelligent analysis should include additional qualitative rules for guiding optimisation, smart summaries of LOCATE results and features that are likely to emerge from tracking user actions and inferences of user goals and plans.

Background

LOCATE is a system for computer-aided design, developed originally for workstation layout but generalizable to a variety of other areas (Hendy, 1984, 1989). An underlying program embedded within LOCATE allows a user to input information about the position, orientation and content of workstations in a workspace layout and, when satisfied with a given design configuration, to have the program evaluate its cost efficiency, or that of its components, through the generation of cost functions. Different configurations may be tried, and their cost functions compared, as a way of helping a user determine the best design.

LOCATE forms part of a broader initiative to provide software support for military missions. That initiative is intended to provide support for, among other things, mission analysis, scenario generation, function analysis, function allocation, task analysis and network analysis. In that context, LOCATE will be used by engineers to design workstation layouts and to help evaluate the effectiveness of those designs.

A key element of the initiative will be providing intelligent aiding to users in a variety of contexts. A number of earlier initiatives have been moving LOCATE in that direction.

In an study unrelated to any of the recent development work, LOCATE formed the basis for a prototype that illustrated ideas on how to design and build intelligent interfaces (W7711-0-7119). Specifications for an Intelligent LOCATE (IntL) were generated using the approach of "Explicit-Models Design" (Edwards, 1990, 1994; Edwards and Hendy, 1992). E-M (pronounced "EM") design isolates four major components of intelligent, adaptive systems modelled as separate knowledge sources within a computer. Those four components are the task , user, dialogue and system-self models.

More recent work on LOCATE itself (W7711-6-7321/001/SRV) added a hypertext help system and incorporated the tracking of user activities as a basis for providing feedback on how to use the system more effectively.

Most recently, development work added significantly to the ease with which users are able to interpret the results of communication cost analyses (W7711-7-7408/001/SRV). Prior to that effort, if a user wished to identify, say, which workstations accounted for the highest communication costs in some real-world example, he or she would have to sift through large matrices of numbers to find that answer. Now, users can inspect colour-coded graphical

displays of those costs and quickly identify which workstations are associated with which category of costs.

The current contract will add new features to ease a user's job of interpreting the results of a LOCATE communication analysis. In part, this will be accomplished by providing the user with such things as displays of summary information, which normally is inferred from output data. In some respects, more importantly, however, it will be done by extending LOCATE's intelligent aiding capabilities to the results of workspace analysis. The latter should provide enhancements useful not only at various stages in a LOCATE analysis but also in the application of intelligent aiding to outcomes from by other, related software.

In addition to the above primary focus of this contract, some effort will be expended in making select enhancements to LOCATE's basic functionality.

Research Approach

The general approach taken in this contract will be to explore ways of facilitating the interpretation of LOCATE's communication analyses. Simple additions to the interface that summarise results of communication costs for a design configuration will be combined with more complex and subtle processes to help users in the interpretation of those results.

On-going discussions with the Scientific Authority will identify opportunities for incorporating both simple and complex help about the results of LOCATE's workspace analysis. Where possible, discussions with workspace design experts will reveal what information they find useful in the analysis of workspace layouts. That information, in turn, should provide clear direction for the kinds of items to incorporate into LOCATE to make it a smarter and more effective analytical tool.

A portion of the contract will be devoted to adding and modifying a few basic features of the LOCATE software. A number of desirable features will be examined and, in consultation with the Scientific Authority, choices made as to which are to be included.

Study Objectives

The study objectives for this research are:

- to explore various ways to augment LOCATE's interface so as to ease a user's interpretative burden with respect to the results of its workspace analysis;
- to implement some of those ideas within the LOCATE interface;
- to explore intelligent aiding strategies that should help users more easily understand and interpret the results of LOCATE's workspace analyses;
- to combine enhanced monitoring of user interface actions with various of those strategies into new features for LOCATE;
- to identify workspace design experts and, where possible, explore with them the kinds of information they find useful in performing analyses on workspace layouts;

- to identify that information, if possible, in the context of software currently used by those experts, or in the context of an illustrative use of the LOCATE software;
- to explore how task, user and system self models, implemented as separate knowledge sources within LOCATE, can be used to aid user interpretation of workspace analysis results;
- to make use of some of the information in those separate knowledge sources when implementing intelligent help features that aid in the interpretation of LOCATE's workspace analysis results;
- to identify and implement a limited number of features that will extend LOCATE's basic functionality.

Intelligent Adding in LOCATE

Easing the Interpretative Burden

In earlier work on LOCATE (W7711-7-7408), users who wished to examine costs associated with workstations within the communication domains analysed often needed to examine large matrices of numbers.

Since the numbers in those matrices have no absolute meaning, that is, are only relevant for comparative purposes within a set of constraints, such as the number of workstations being compared, a user has the extra burden of determining the range of numbers in the matrices before any meaningful comparisons are possible.

Even after determining the range of numbers, which may vary between zero and one, the job of comparing costs in any real-world example is a daunting one when matrices are the only source of information about costs.

To ease the user's burden, colour cost displays were constructed to allow for a much easier discrimination as to which workstations contribute to the high costs of communication within and across domains. In addition, designers were given control over the criteria that define the colour-coded cost categories through a LOCATE "Display Editor" that was designed and implemented for that purpose.

Figure 1 shows a graphic matrix for the combined costs for a design with ten workstations. The title of the window tells which of five potential displays one is viewing. In this example, it is the "Total" costs, or the combined costs for all domains being analysed.

To identify the number of domains being analysed one clicks on the pop-up menu in the "Windows:" panel. Highlighted domains in that pop-up indicate which domains the user has selected for analysis. Although not visible in Figure 1, only the visual domain has been selected for analysis, and so, the window for the combined domains is equivalent to that of the visual domain alone.

Other buttons in the cost display window allow the user to display measures of Link Quality and Weighted Costs and display each of those in normalised and non-normalised form, to print the matrix display and to close the active Cost Display Window or all display windows currently open.

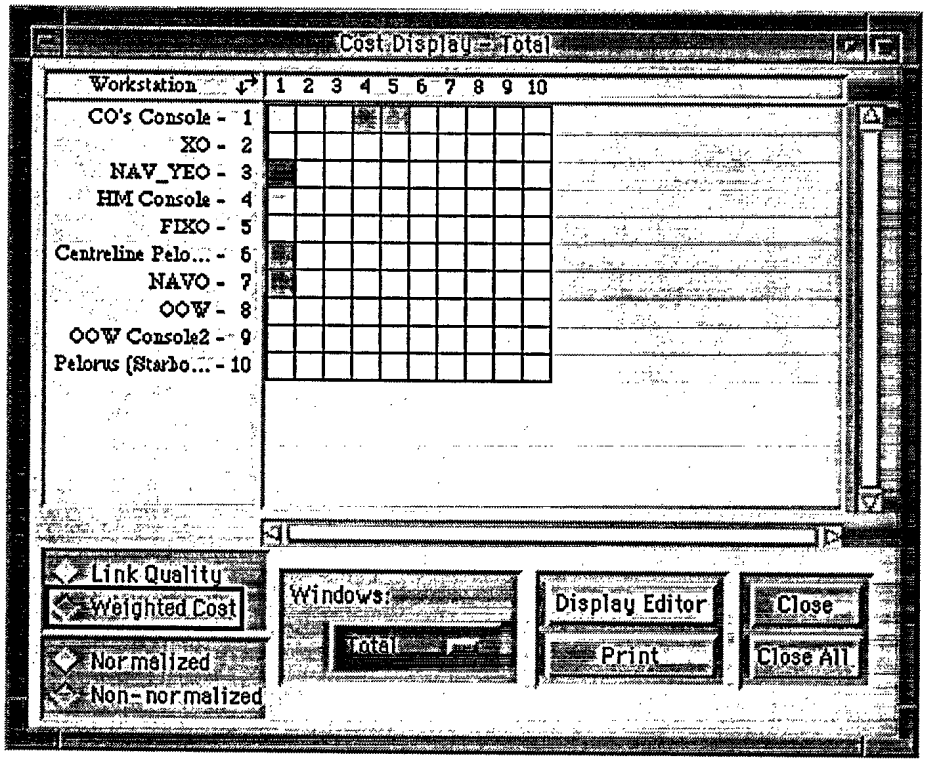


Figure 1. Colour Display of Detailed (Weighted) Costs for a Ten-Workstation Design

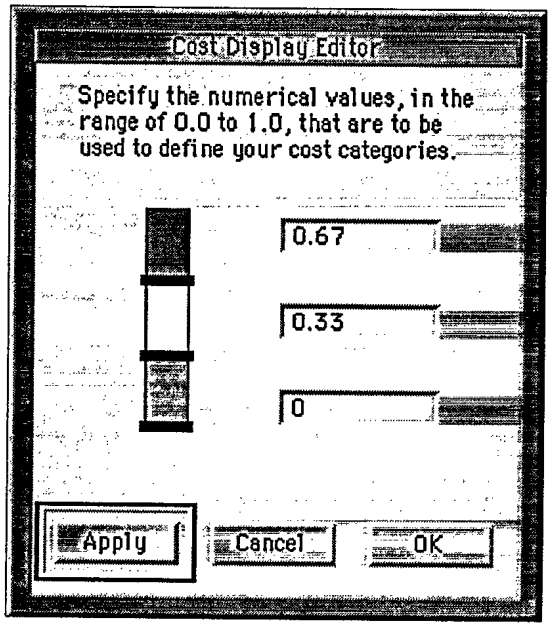


Figure 2. Display Editor for Specifying the Criteria for Cost Categories

Figure 2 on the previous page shows the Display Editor Window, which allows a user to specify the criteria for the various colour-coded, cost categories.

For example, the four criteria used in creating Figure 1, from lowest to highest, are 0.00; 0.20; .040; and, 0.60. Changing those to the three criteria of 0.00, 0.33 and 0.67 (see Figure 2) yields the colour matrix shown in Figure 3, below.

Note that setting the first edit text box value to 0.00 effectively eliminates the fourth category used in Figure 1. The user not only has control over the criteria for the various categories but also the number of categories as well.

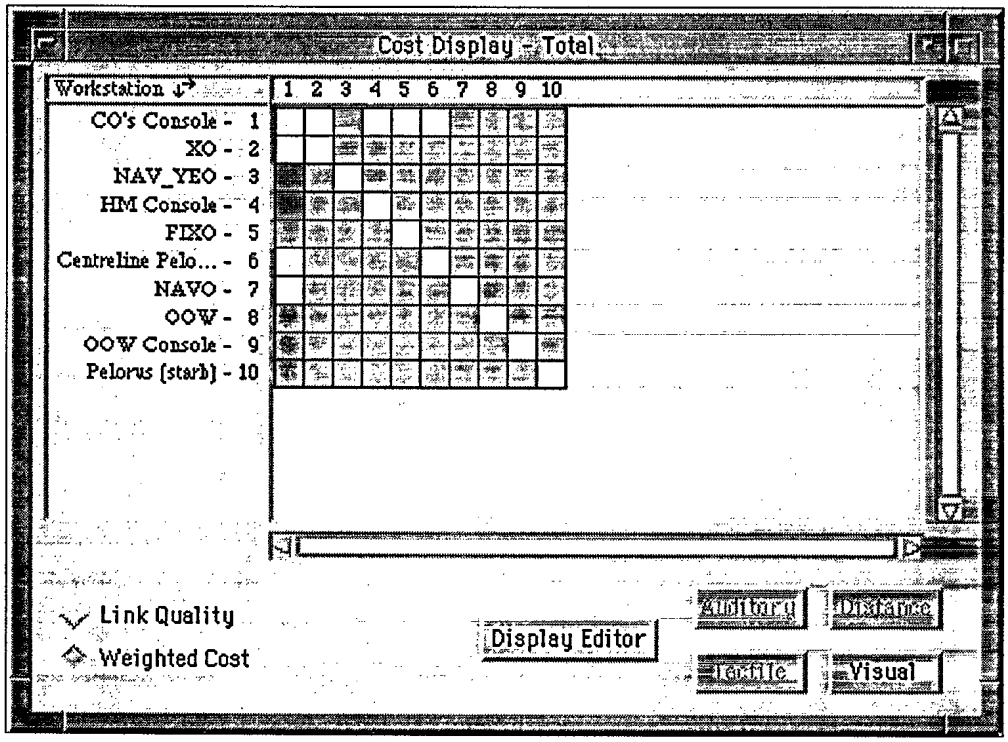


Figure 3. Ten-Workstation Display Following a Change in the Cost Criteria

To change the criteria in the Display Window, the user simply types in a new value in one of the edit text boxes to define the "floor" for the category and then clicks "Apply" to see the results immediately displayed in all open cost display windows.

Another way of changing the criteria is to click and drag one of the black bars that separate the categories on the colour slider. As the user drags one of the bars, the value for that category changes dynamically. Dragging a bar all the way to the top or the bottom of the slider eliminates the category. A user may set criteria to display as few as one and as many as four cost categories.

Another aspect of these matrices is the main diagonal. It is used to display the costs for 1st-order priority weights. Since no 1st order priority weights were specified for the example analyses in these figures, the colour of the diagonals is a simple background colour, which indicates that no costs were computed for the main diagonal elements.

In summary, the graphic displays of the various communication domains combined with an editor for specifying cost categories makes LOCATE considerably more useful to designers by easing the burden of interpretation of their results. The ability to quickly see the various categories of costs in a design by inspecting colour-coded matrices, can save many hours of effort and encourage exploration of configurations that might otherwise go unexplored.

Intelligent Help and Analysis: A Dual Approach

Intelligence in any entity, including a piece of software, is often a matter of perception. It is difficult, if not impossible, to know what is driving behaviour that is perceived to be intelligent, but the perception is often based on things like the novelty or complexity seen characterising a set of actions.

Much of the apparently intelligent behaviour of contemporary software, such as real-time correction of typing errors in a word processor, represents little more than clever implementation techniques, informed by previously collected empirical data that indicate typical errors users make in typing. The behaviour is "intelligent" in so far as it reflects the cleverness of the designer in anticipating the needs of the user and in dealing with contingencies involved in the user's interaction with the software.

As work to incorporate intelligence into the LOCATE Workspace Layout Tool proceeded, it became clear that two approaches to the general task of adding intelligent aiding were necessary:

- implement techniques of the kind just described, where the designer is able to anticipate what will happen in the interface and provide system responses based on the details of what occurs;

- modify the software so that it is able to monitor the actions taking place at the interface and represent those actions in ways that allow it to "understand" their meaning relative, say, to the overall task.

With respect to the first approach, the fact that intelligence is based on little more than the cleverness of the designer in implementing a set of contingencies should not detract from the approach's usefulness. It can be effective in informing users about how a piece of software works and how the software itself can help in a more efficient analysis of its own results, as would be the case with LOCATE.

Further, it can be effective in building user confidence in the software in ways that help the user master the features of the software as well as the task the software is constructed to support. It can be effective also in predisposing the user to want to work with the software.

The second approach of incorporating intelligence into LOCATE involves a more fundamental structuring of information *qua* knowledge for the system. Such knowledge has the potential of providing substantial and reasoned intelligent help to its users.

Once the representational groundwork is laid, knowledge about what is happening at the interface can become part of a reasoning system for determining the kind of intelligent analysis to provide to a user, when that analysis should be provided and in what presentational form.

Both the above approaches, especially the latter, depend on LOCATE's ability to monitor user actions and their effects at the interface. The second approach requires a transformation of those actions and results into more meaningful representations useful in intelligent aiding. Specifically, the second approach requires LOCATE to "understand" the goals implied by a user's actions as well as its own **supporting goals and actions** necessary to complement those of the user.

Monitoring User Interface Actions

In previous contracts, tracking user interface actions occurred in two contexts. The first was in a usability study (W7711-6-7320) in which actions of human factors experts were tracked as they gained experience with LOCATE.

At the end of each working session, a separate application called simply, a "Log Reader", summarized the results of the actions of those experts, specifically indicating what LOCATE windows had been opened, how often each was opened and the duration of time each remained open.

The second context in which tracking was used in LOCATE was in a proof-of-concept study to demonstrate intelligent help (W7711-6-7321). In that study, user actions were tracked and linked to rules in LOCATE's underlying expert system. The rules were then used to generate help information when certain events occurred.

The type of help LOCATE provided was meant only to illustrate the direction for future development. The resulting, "intelligent" help fell more into the category of the clever implementation techniques mentioned above, although it clearly made use of LOCATE's ability to monitor what the user was doing at the interface.

Extending LOCATE's Tracking Ability. In a study that paralleled the present effort (W7711-8-7480), the number and type of actions being monitored were substantially increased from earlier studies. LOCATE is now able to monitor most all user interface actions, including the following:

- all menu item selections;
- all palette item selections;
- all keyboard shortcuts;
- all selections of objects in the workspace;

LOCATE is also able to distinguish whether an object or palette item has been clicked, double-clicked or clicked in combination with some keyboard entry. Finally, it has been extended to track changes in edit text boxes in the main workstation window.

The next two sections deal with how LOCATE infers goals from user actions and with the application of the "explicit models" approach for representing those goals and

actions. These sections are taken largely from the already cited parallel development work on intelligent help.

User and System Goals Inferred from Interface Actions

As a user creates and analyses a design, LOCATE tracks the actions of that user at the interface. Most, if not all, of those actions imply both the goals the user is trying to achieve and the supporting goals and actions required of LOCATE in order for a user to achieve his or her goals.

In human-human interaction, other people's actions continually give rise (in an observer) to inferences about what those people are trying to accomplish, that is, the goals they are trying to achieve. In effect, the inferences serve to reveal the purpose behind the actions observed.[1] When implemented in software, such inferences can lay the foundation for providing truly intelligent help, in the current context: help regarding the analysis of design configurations.

To illustrate the process of goal inference, imagine a user clicking on a workstation in a LOCATE workspace and then selecting "Workstation" from the View Menu to display its attributes window (double-clicking on a workstation object has the same effect).

Once the window has been displayed, the user can click the "Link Functions" tab at the top of the window to display the portion of the window that allows for the modification or entry of link function data for that workstation.

The example below shows the goals LOCATE currently infers when the link functions tab in such a window is selected. User actions appear in black, user goals in red (preceded by a "U") and system (support) goals in blue (preceded by an "S").

- Link Functions tab clicked in window of [named] workstation
  - U • To change the [default] link function data for the [named] workstation
    - S • To record changes in the [default] link function data for the [named] workstation
      - U • To examine the [default] link function data for the [named] workstation
        - S • To display the [default] link function data for the [named] workstation
          - U • To show the Link Functions portion of the [named] workstation window
            - S • To display the Link Functions portion of the [named] workstation window

---

[1] In the plan recognition literature, for an uninvolved observer, this is known as "keyhole" recognition (Kautz & Allen, 1986).

These goals appear in a hierarchy that displays those most closely associated with the user's **primitive actions**[2] at the bottom of the hierarchy and, above those, other inferred, higher-level goals. which provide a broader context for the action taken. Thus, the user goal, "To show the Link Functions portion of the [named] workstation window", is the last user goal listed above and is supported by a LOCATE subgoal responsible for the display of that portion of the window in the interface.

The goal of examining the link function data entails the lower-level goal of displaying the link function portion of the window. Thus, the user "shows" the link function portion of the window and, having done so, is immediately able to satisfy the next higher-level goal of examining the link function data.

Just as the goal of examining link function data required a window on that information to be opened, so the goal of changing those data depends on the user being able to examine them. Therefore, once the goal of examining the data has been satisfied, the goal of changing those data may be pursued. The satisfaction of the "examine "goal has the status of a prerequisite in more complex plan structures, which are made up of goals, subgoals and actions.

Of course, it may be that the user only wants to examine some of the link functions and does not intend to change them but, since LOCATE cannot be sure, it must infer both goals as likely possibilities. If the user closes the Link Functions Window without making any changes, then the goal, "To change the link function data...", is cancelled and recorded as such when the goal is removed from the list of current goals being pursued.

In future, probabilities will be associated with such inferred goals and subsequent actions (and goals) will be used to help confirm or disconfirm those inferences, that is, increase or decrease their associated probabilities.

System Support Goals. Currently, each user goal inferred from an interface action has one or more corresponding system-support goals necessary to its realisation.

User goals may be subgoals or supergoals of other user goals and similar relationships may hold true for the LOCATE (system support) goals. Thus, separate

---

2 "Primitive" in the sense of a convenient and useful level of abstraction.

hierarchies of user and system goals may exist but only the user hierarchy is shown in the above example. System support goals are shown hierarchically relative to the user goals they help to support, however, they are not to be understood as being in a hierarchical relationship to other system goals except when multiple instances appear under a given user goal.

Other actions in the Link Functions window lead LOCATE to infer more specific goals. The example goal structure below illustrates this point. Again, supporting subgoals, adopted and pursued by LOCATE as a consequence of inferred user goals, are indicated in blue, (preceded by an "S") user goals in red (preceded by a "U") and primitive actions in black. For ease of layout, user goals are not shown hierarchically.

- Auditory-source-angular component in Link Functions (LF) window of [named] workstation clicked

U • To change the attributes of the [named] LF component of the [named] workstation

S • To record changes in the attributes of the [named] LF component of the [named] workstation

U • To examine the attributes of the [named] LF component of the [named] workstation

S • To display the attributes of the [named] LF component of the [named] workstation

- Butterworth function for the auditory-source-angular component in Link Functions (LF) window of [named] workstation selected

U • To select the [named] function for [named] LF component of the [named] workstation

S • To display the [named] function for the [named] link function component of the [named] workstation

S • To display the name, "Butterworth" in the pop-up menu for the [named] link function component of the [named] workstation

S • To record the [named] link function for the [named] link function component of the [named] workstation

As user and system support goals are satisfied, they are removed from a list of current goals being pursued. If the user clicks the "Apply" button instead of "OK", after changing a link function or its arguments, the goals of changing that component and examining its attributes are determined to have been satisfied, and are moved into an "Action and Goal History."

Although they have been removed from the current goals list into a history list, a new set of goals for revising and examining the data are instantiated and placed into the list of current goals. This happens because the user has left the Link Functions portion of the

window open and so still may want either to examine the data for that link function component or make further changes to them.

If the user clicks another component, however, the goal of examining the old component is marked as satisfied but the goal of changing the attributes of that link component is cancelled.

To get a more complete idea of the extent of the goals inferred by LOCATE, see Appendix A in a parallel report entitled, *Intelligent Help in the LOCATE Workspace Layout Tool (Contract No. W7711-8-7480/001/SRV)*, which lists all of the goals currently being inferred.

Using C++ to Handle Object Complexity. Much of LOCATE's functionality is derived from the original C code in which it was written, along with a rule and object-based expert system to support LOCATE inferences.

The creation and spawning of goals, however, is now being handled in C++. To get an idea of the nature of the abstract goal class used as a model for creating goal instances, see the goal class requirements in Appendix B of the associated report cited earlier, *Intelligent Help in the LOCATE Workspace Layout Tool (Contract No. W7711-8-7480/001/SRV)*. The currently implemented structure of this goal class is a subset of that shown.

Explicit Models: The Basis for LOCATE's "Understanding"

Explicit Models Design. The approach used in building a fundamental framework for intelligent help in LOCATE is that of Explicit Models (EM) Design (Edwards, 1990; Edwards & Hendy, 1992; Edwards & Sinclair, in press). The main thrust of EM Design is to make explicit to the designer, the user and to the system itself, as much as is possible and useful about the nature of the task being performed, the characteristics of the human user, the characteristics of the computer and its support for the user, and the nature of the interaction between the two.

EM Design draws on aspects of several paradigms used in Artificial Intelligence and attempts to provide mechanisms for systems to model themselves through meta-abstraction and the use of metaobjects at the level of programming.

Models of Task, User and System. Following the EM Design approach, three of four general, explicit models are being designed for LOCATE: **task, user and system (self) models.** The fourth model, the **dialogue** model, should prove useful in future.

The *task model* contains information about what is happening relative to the task of creating and analysing designs. Task representation is in the form of actions and goals, specifically, the interface actions and the goals of the user and system. Example actions and goals were given in the previous section. As the user and system's action and goal history is generated and studied, it should be possible to combine sequences into more complex plan representations.

In, the meantime, local representations of what the user (and system) are doing will feed rules that will decide the kind of intelligent help to provide to a user, and when and how that help should be given. Preliminary examples that point to the kind of help that is likely to emerge will be discussed presently.

User and system (self) models, like the task model, are separate knowledge sources that will ultimately inform the help system as it makes decisions about what help to give. Unlike the task model, these models should be thought of as repositories of the knowledge that the user and system possess, respectively. More specifically, they reflect the beliefs the system holds about the user and LOCATE itself.

Two types of knowledge are the knowledge about LOCATE functionality and, less important at this stage, the knowledge about how to create and analyse workspace layout designs. Of course, knowledge of LOCATE's features of necessity is closely tied to the creation and analysis of designs.

The system's beliefs about what the user knows about LOCATE's functionality is critical to providing intelligent help. Aspects of that knowledge will be covered in more detail presently.

The Object-Oriented Paradigm. The principal context for the analysis, design and implementation of explicit models is the object-oriented paradigm and a key issue is the identification of the kinds of objects, including their defining attributes and behaviours, required to create an Intelligent LOCATE (IntL).

Many objects are already in place and are integrated with expressions of user task goals and supporting system (LOCATE) subgoals, as described earlier. Thus, when a user double clicks on an object in the workspace such as a workstation, LOCATE knows that the object has been selected and its attributes window opened. Further, it is able to infer a likely set of hierarchical user and associated system support goals.

Interface Issues. In the interface, knowledge contained in each model is separated into its own window, that is, Task Model, User Model or System (Self) Model Window. To view any of those windows, the user simply selects its name from LOCATE's Help Menu.

The **Task Model Window** is divided into three sections:

- Latest Action;
- Current Goals;
- Action and Goal History.

Figure 4 shows an example of that Window. The "Latest Action" section of the window shows the last action the user performed in the interface; the example is the selection of the workstation icon in the LOCATE tool palette.

The level of abstraction chosen for representing actions in LOCATE is that of single and double mouse clicks on identifiable objects, selections of items in menus, keyboard commands and typed text in specified contexts. This is the level of *primitive action*, referred to earlier.

The second section of the Task Model Window is "Current Goals", which provides a list of the goals currently being pursued that LOCATE has inferred from the user's action. Those goals include not only the goals the user currently holds and is pursuing but also the subgoals that LOCATE holds and is pursuing in support of the user's goals.

Notice that the last two goals listed in that portion of the example figure have an "(s)" at the end. The "s" indicates that a goal has been satisfied.

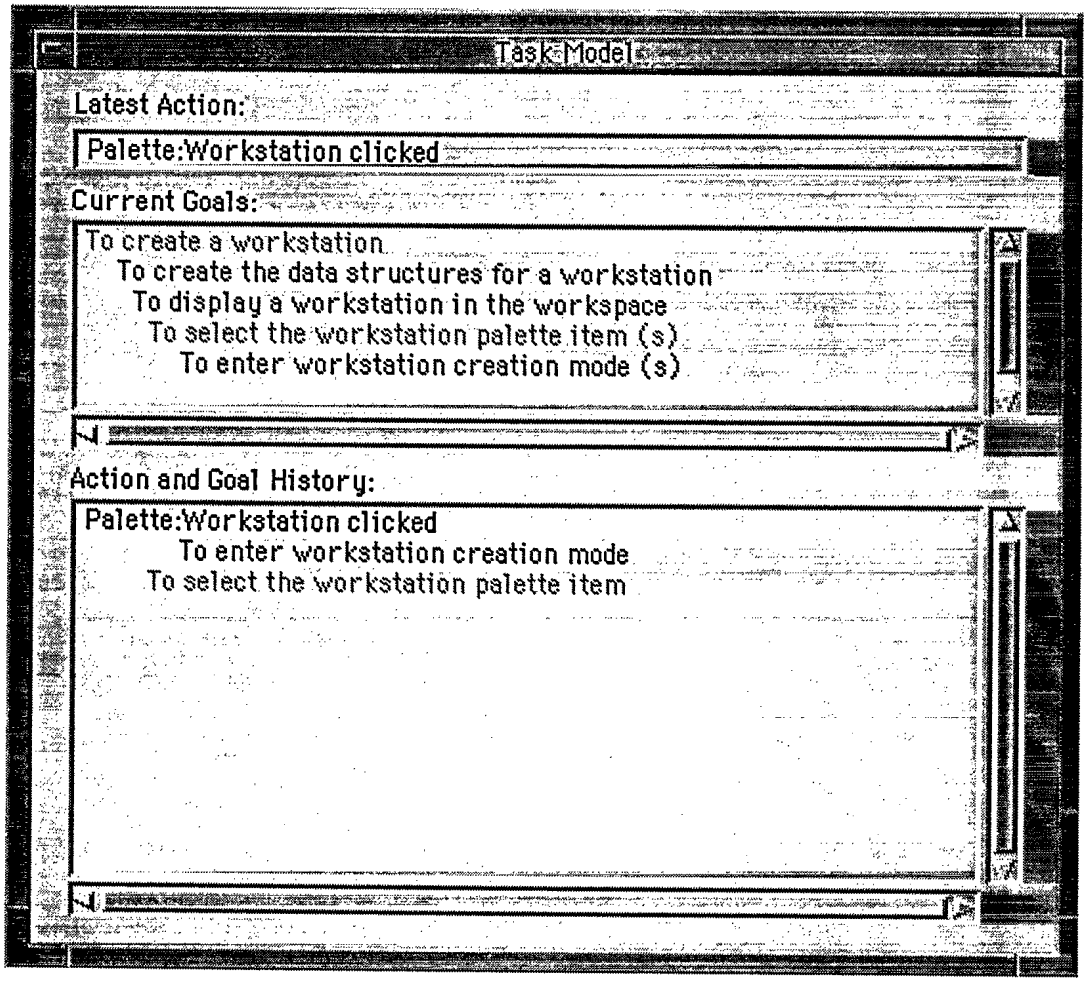


Figure 4. LOCATE's Task Model Window

Normally, when the user performs a new action, the "current goals" and "latest action" are transferred to the Action and Goal History portion of the window. If the goals have been satisfied, which most are, the trailing "(s)" is dropped as a way of reducing clutter in the history window, however, if they have not been satisfied, a trailing "(-)"is added.

The action and goal history eventually will provide an "audit trail" of everything the user and the system have done during a session. Monitoring actions and goals is a first step to a version of LOCATE that can provide truly intelligent help to its users. If the system can "understand" what is happening, as it is happening, help to the user can be provided on that basis and will be more richly contextual.

The User Model is represented in the interface by the **User Model Window**, which contains the beliefs that LOCATE maintains about the current user.

The first information placed in that window at the beginning of each session is a question about the identity of the current user. If the user selects "About You..." from the Help Menu and fills in the information in the resulting window, LOCATE is able to answer that question. The information requested of the user in the About You..." window is shown in Figure 5.

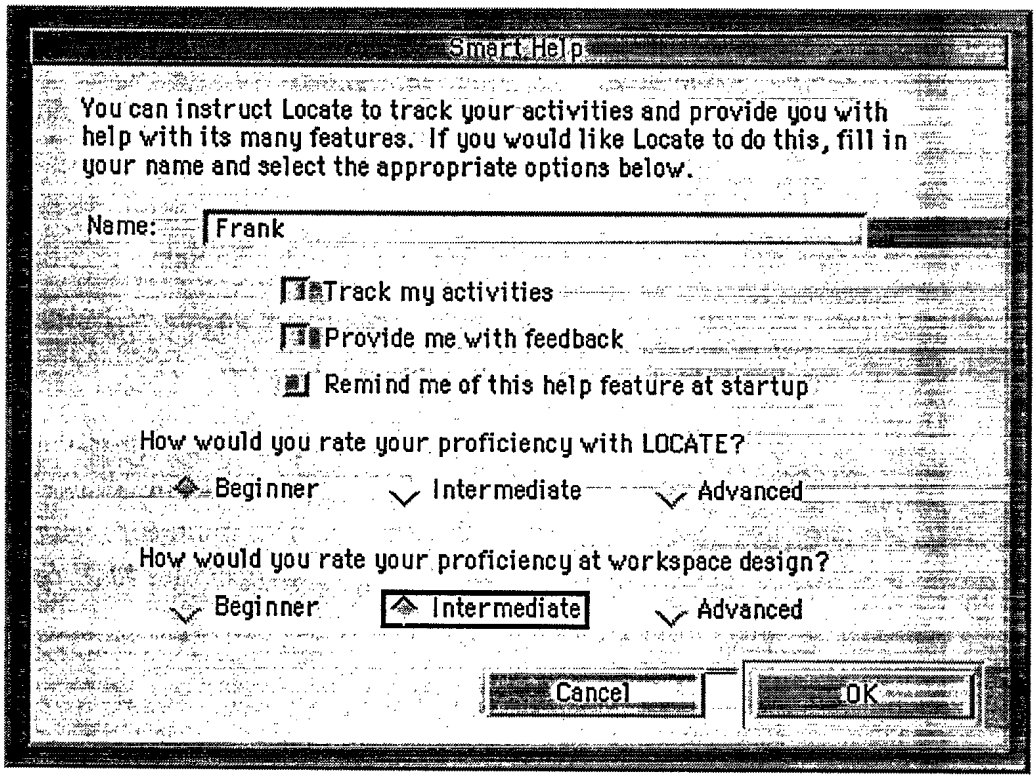


Figure 5. LOCATE's "About You..." Window.

Having now been informed about the identity of the user, LOCATE is able to answer its initial question. LOCATE's initial question and the resulting answer are shown in LOCATE's User Model Window, example of which appears in Figure 6.

The information in this example of the User Model Window represents only one source of knowledge that LOCATE uses in acquiring information about what the user knows, namely, knowledge derived from what it has been told by the user.

This and two other sources of knowledge about the user will now be discussed.

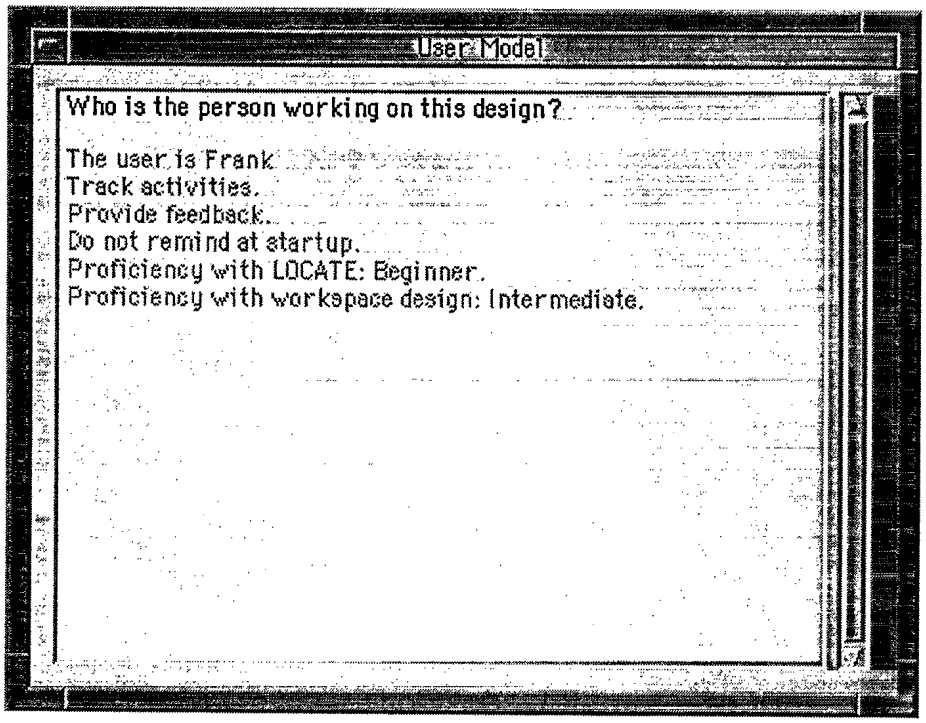


Figure 6. Question & Answer Example in LOCATE's User Model Window

Providing Intelligent Help and Analysis I: The User Model

As already mentioned, the system's knowledge about what the user knows about LOCATE's functionality is critical to intelligent aiding. That knowledge is gained in three ways by LOCATE:

- direct statements by a user about what he or she knows (the example in Figure 6);
- observation of user actions and consequent inferences that can be drawn;
- help information provided to the user by LOCATE.

Knowledge Derived from Direct User Input. Currently, some minimal information is obtained directly from the user. If the user decides to allow LOCATE provide that help his activities, he needs to identify himself, ask LOCATE to track his activities and to provide him with feedback. Further, he can ask LOCATE to remind him of the intelligent help feature each time the application is opened. All this information is obtained directly from the user when he selects "About You..." from the Help Menu (Figure 5 above).

Knowledge Based on System Inferences from User Actions. With respect to observation and inference, LOCATE is able to track a few user activities and provide some clever help. One example from an earlier study is LOCATE's ability to know whether the user is opening windows by double-clicking or using a menu item selection. As LOCATE tracks these activities, it may draw conclusions about what the user likely knows or doesn't know.

Knowledge Deduced from Help Provided to the User by LOCATE. A third source of knowledge about what the user knows is derived from the results of LOCATE's presentation of help information.

In the example mentioned in the previous section, after several instances in which a new user has made menu selections to open an object's attributes window, LOCATE puts up a help window indicating to the user that a specific window or, more generally, object attributes windows, can be opened by simply double-clicking on the object. Figure 7 shows an example window illustrating that kind of help.

After putting up the help window, LOCATE can conclude that the user likely knows how to open workstation windows using the double-click method.

Displaying such a help window and obtaining the user's answer as to whether he or she knew that double-clicking on a workstation opens its attributes window means that LOCATE can now draw inferences about what the user likely knows and can store that knowledge in its User Model. Notice that the window not only derives information about the user from the help it has given but also directly from the user through his answer to the question in that window.

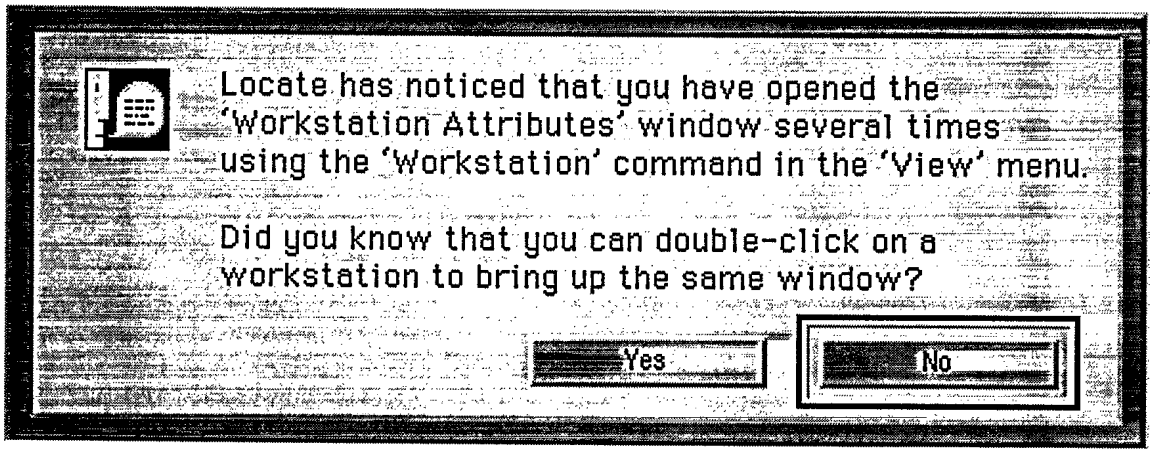


Figure 7. Help Window Invoked from Monitoring User Interface Actions

An example of such knowledge appears in Figure 8, with the usual colour-coding, that is, LOCATE comments and inferences in black, LOCATE help in blue and User responses in red.

Detecting the user's subsequent use of menu item selection will now have different implications for help than it did before. If the user continues to use menu selection to open workstation and other windows, LOCATE will be led to the conclusion that he or she may prefer that method over the more efficient, double-clicking.

In summary, information about what LOCATE believes the user knows is now being stored in LOCATE's User Model and is based on the three sources of knowledge identified above: direct information provided by the user, observation of user interface actions and knowledge about the content of help information provided to the user by LOCATE.

Such knowledge will be useful in providing intelligent analysis through understanding what a user knows about LOCATE's analytical capabilities, through maintaining a history of what has been analysed and when, and by keeping records of the results of particular analyses.

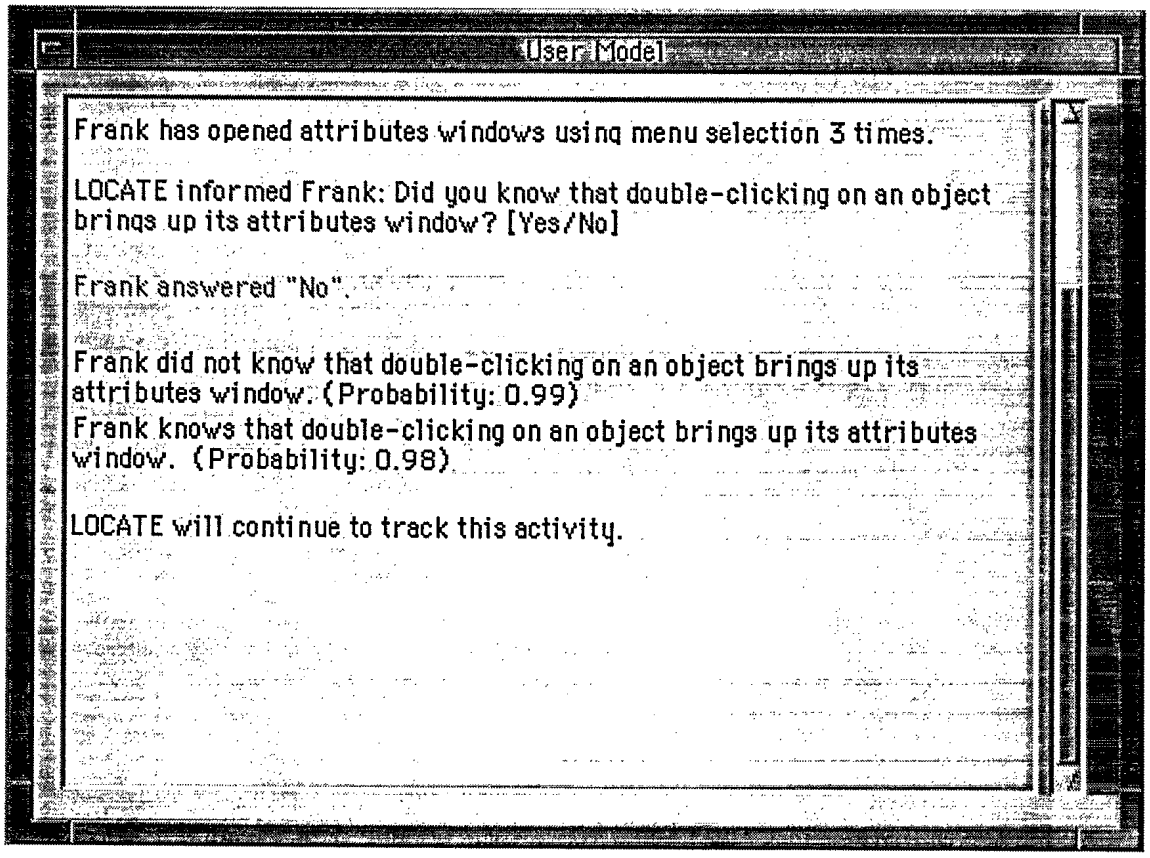


Figure 8. Example Information Placed in LOCATE's User Model Following a User's Answer to the Question in the Help Window Shown in Figure 7.

Providing Intelligent Help and Analysis II: The System Model

In addition to knowledge about the user, intelligent help and analysis can arise from information stored in a system (self) model.

The knowledge a system can have about itself includes knowledge (beliefs) about its own capabilities. The primary reason for creating such a model is to aid the user and so, the knowledge contained in a system (self) model should be organised around providing intelligent help and analysis to users about the features of the LOCATE system.

For that reason, early work on LOCATE's system model links a new Smart Help Menu item with its System Model. Further, it was determined that much of the help that LOCATE might offer to a user could take advantage of LOCATE's goal orientation as represented in its Task Model.

An example of that kind of help appears in Figure 9. The window in that Figure is displayed whenever the user selects "Smart Help" from the Help Menu. Smart Help is organised around a concept of "How to...."

In the example, the user has asked LOCATE how to print and LOCATE responds to that query by displaying a list of goals associated with the concepts of "print" that it understands. That list is displayed in a panel entitled, "Goals" on the left side of the window.

The list of goals provides the user with choices that focus his inquiry. When the user chooses one of the options in the list, a set of subgoals (a rudimentary plan) appears on the right which tells the user how to achieve the goal on the left..

In the current example, the user has selected "To print a design" and the system has displayed a list of subgoals on the right that show the user the subgoals *qua* actions that are necessary for him to be able to print a design.

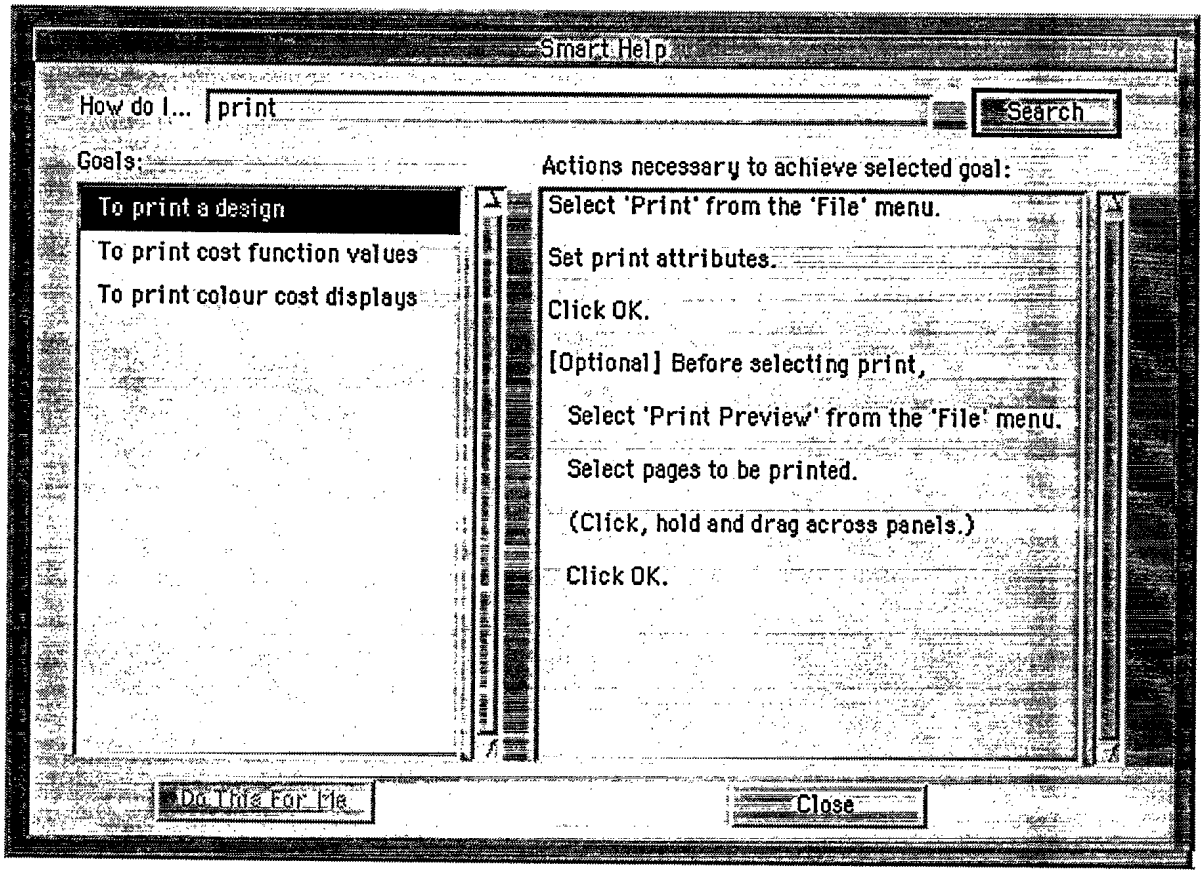


Figure 9. Example of LOCATE's "Smart Help" Window.

Notice that the query from the user is in terms of an action, the high-level response from LOCATE in terms of goals and the consequent plan framed in a panel entitled, "Actions necessary to achieve the selected goal." Of course, those actions are a mixture of subgoals, conditionals and "primitive" actions among other things, but that mixture is constructed by LOCATE in a way that makes sense to the user.

Just as the task model reflects an understanding of what is happening in the interface through goal and plan recognition, LOCATE's help and analysis facilities support the user through a process of goal and plan generation. For recent work elaborating intelligent help as plan generation, see Küpper and Kobsa (1999).

As the user requests and receives help from LOCATE, the system's system (self) model tracks what is taking place. Figure 10 shows the consequences for the system model of the interaction that has just occurred in the Smart Help Window.

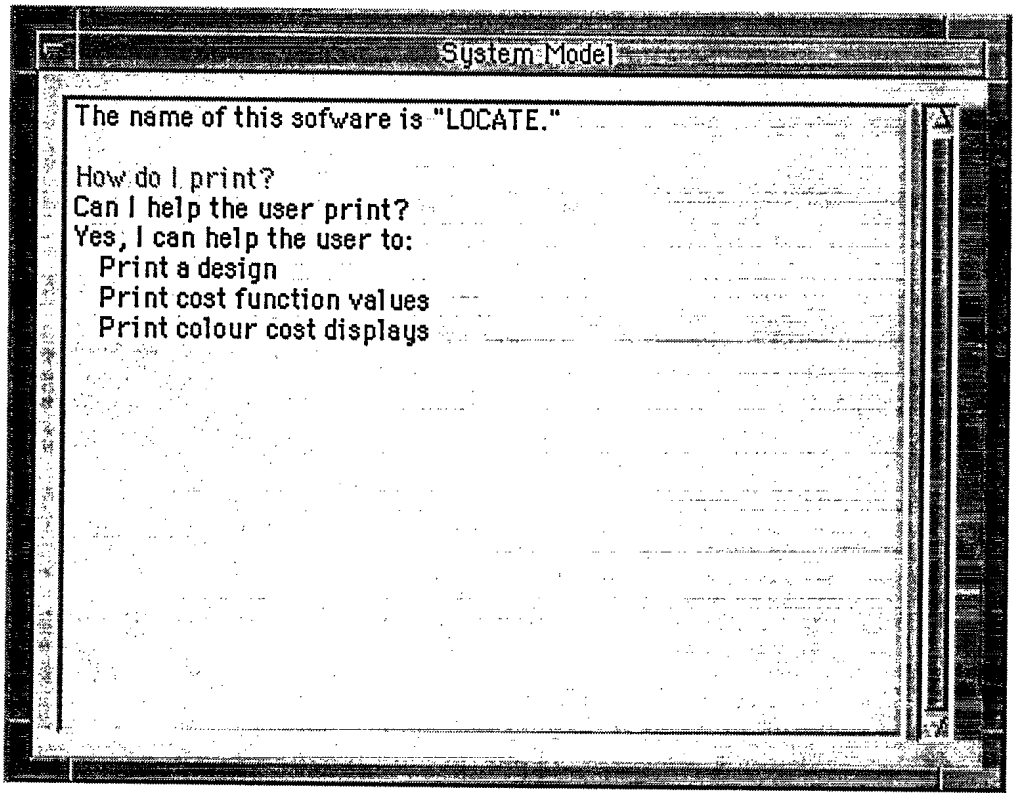


Figure 10. Consequence for System (Self) Model of User Query in Figure 9.

The first entry in the window is a standard entry placed there at startup, which simply identifies the system (in its own self model) for the user. "The name of this software is LOCATE." A first party view was thought to be too anthropomorphic.

The next set of entries reflects the user's request for help. These entries are constructed in a way that is meant to reflect a sense of self enquiry on the part of the system, relative to whether it is able to help the user with some identified problem.

This approach has several advantages. First, it provides an intuitive way for a developer to track the system's exploration of its own knowledge and capabilities relative to user enquires. It also permits the developer to request greater detail about that exploration at any point in the process.

For a user who might wish to view the system model, it communicates that the central purpose of self enquiry on the part of the system is to help the user better achieve his or her goals. It also may be that both the content and nature of such a self-enquiry approach is non-threatening to users in the context of a system that keeps track of its own knowledge and capabilities.

Providing Intelligent Help and Analysis III: Intelligent Optimisation

Earlier work, already cited, eases the user's burden in interpreting LOCATE's results through graphical, colour-coded matrices. Those matrices give the user an overview of the communication costs for a given analysis and permit the user to quickly pinpoint the high costs in a given analysis.

An associated facility allows users to change the criteria for what constitutes "high", "medium" and "low" costs and to see the consequences of those changes immediately in the updated colour displays (see Figures 1-3, above).

In the current work, new and different ways to summarise the results of LOCATE's analyses were explored and options discussed as to the nature of result summaries that might be presented to a user. Timing of the presentation during the course of analysis was a key concern.

Two key issues were explored: 1) information a designer needs or might like to have about the results of a given LOCATE analysis; and 2) information required about the results of multiple analyses performed on different configurations of the same design.

To illustrate, one such summary table was implemented and other similar tables should be incorporated in future as part of LOCATE's basic functionality. Figure 11 on the next page shows an example of this summary table for link function data for all the current workstations in LOCATE's workspace. The example is for the visual domain.

Future work should implement similar tables for priority weights and even broader tables that show link functions and priority weights for all domains simultaneously.

As discussions proceeded, it became clear that much of what might be accomplished by way of intelligent analysis revolved around the incorporation of an optimiser. Plans had been discussed about adding a third-party optimiser to LOCATE but it had been decided that other features had higher priority. The addition of an optimiser had been delayed until those other features were incorporated[3].

The AIM Optimiser. AIM had been working on an optimiser as part of other development efforts and, although it was still a relatively simple tool, it was thought that, with some modifications, it could be incorporated into LOCATE to provide a useful object around which intelligent analysis could be discussed and developed.

Although no provision for adding or building an optimiser appears as part of the current contract requirements, code duplicating some portions of the AIM Optimiser was incorporated into the LOCATE code.

After creating a design, the user is now able to instruct LOCATE to optimise that design by selecting, "Optimize Workspace" from the Execute Menu.

When a user selects that Menu item, LOCATE displays a window in which the user may choose one of two methods for optimisation:

- swapping workstation positions while holding constant their locations and angles of rotation;
- changing the locations and angles of rotation of workstations.

---

3 In the early stages of the development of the LOCATE program, prior to any work on its graphical user interface, a thrid-party optimiser, the Nag Optimser, had been configured to LOCATE.

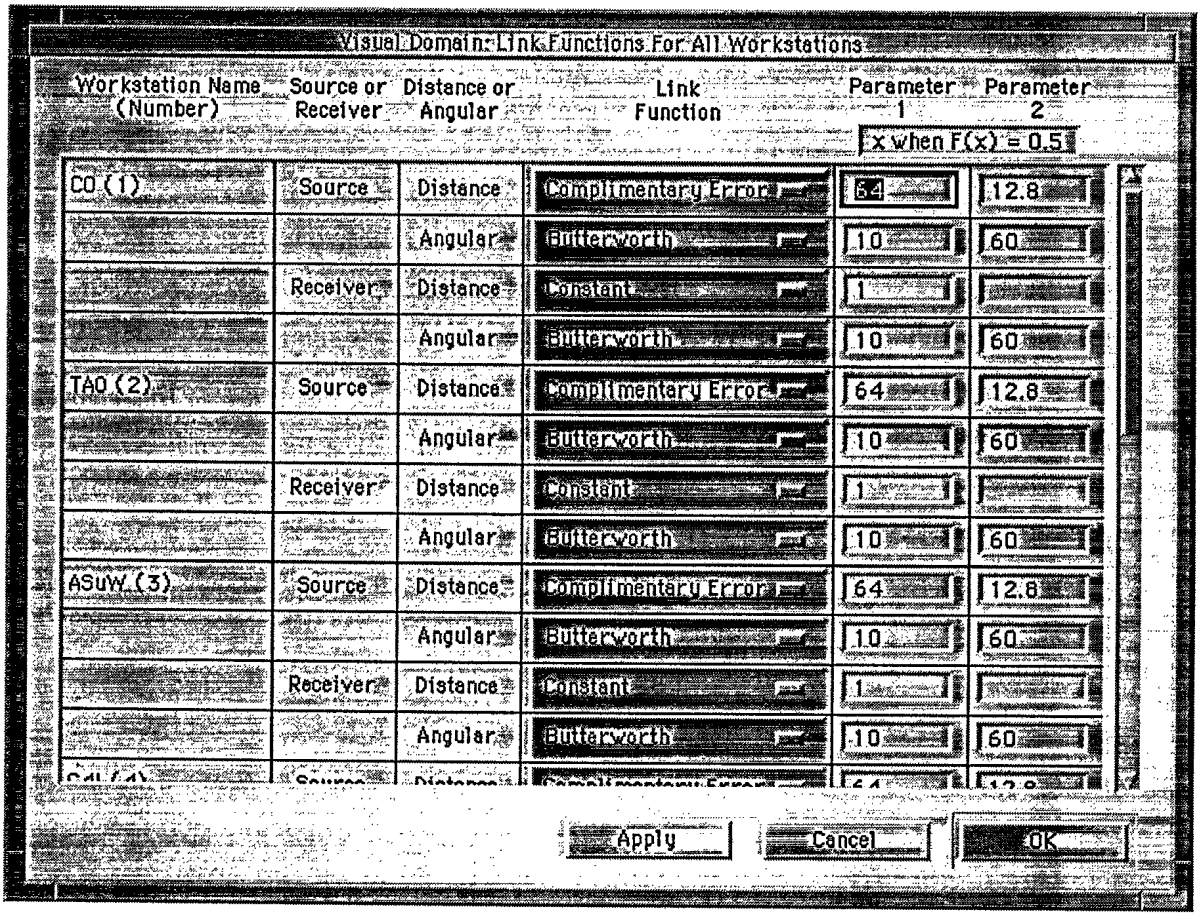


Figure 11. Summary Link Functions Window for All Workstations for the Visual Domain.

Figure 12 shows the Optimise Window that is displayed when the "Optimize Workspace" command is selected.

In the latter method, the user may alter the default values for the step size in units and the number of degrees by which each workstation is moved or rotated, respectively, during optimisation. Choosing the other method of swapping workstation positions dims those options.

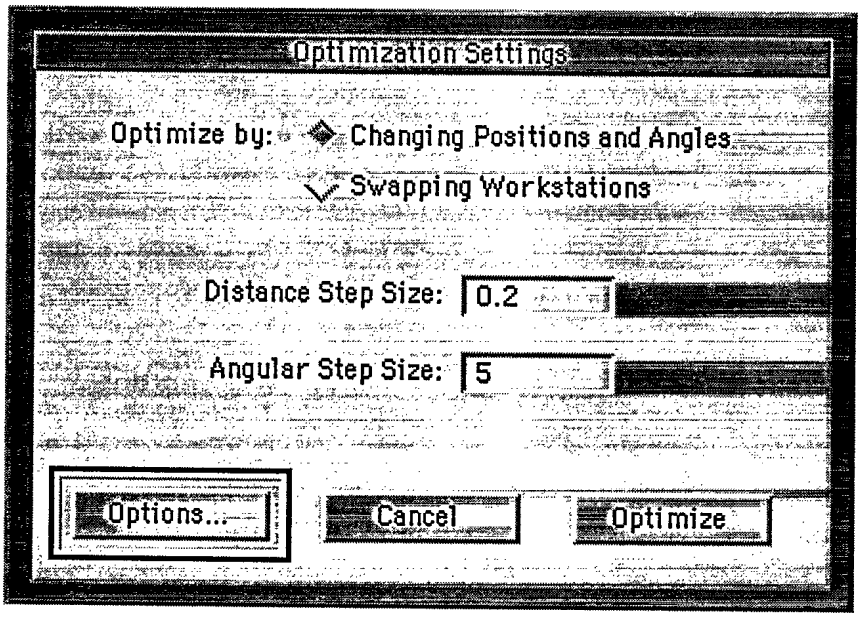


Figure 12. Optimization Settings Window.

Clicking the "Options" button allows the user to instruct LOCATE to keep certain workstations together and others apart, as optimisation proceeds. Separation distances are based on the maximum dimension of the current workspace and may be customised by the user. Figure 13 shows those expanded options, which are linked with LOCATE's underlying expert system and which are discussed more thoroughly in the next section.

After specifying the various options, the user clicks "Optimize" to begin the optimisation process. The positions and angles of workstations are updated on the screen every five seconds and the optimisation may be paused, stopped or aborted at any point during the process. An example of the Optimisation Window appears in Figure 14 showing optimisation interrupted (paused) after a little over a minute into the process.

Items in the window show the initial cost function value, the currently optimised cost function value and the level of improvement in optimisation.

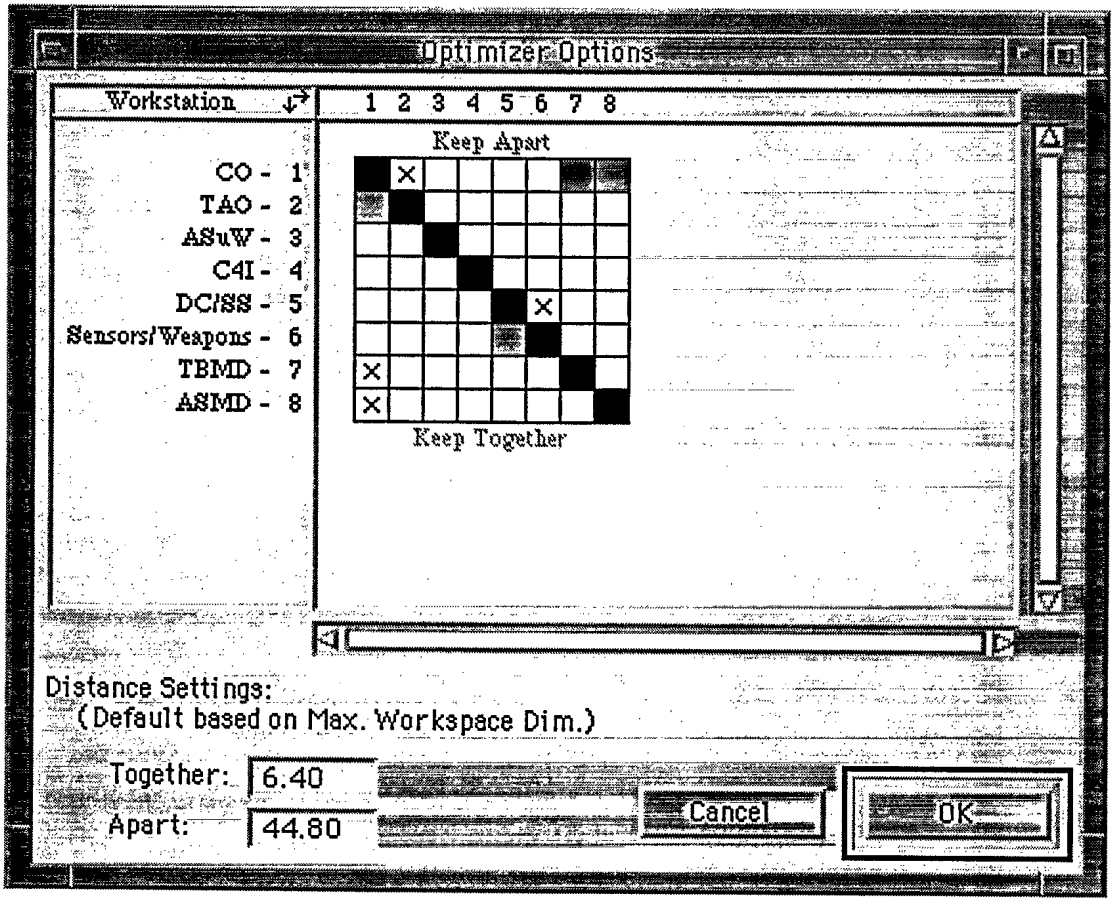


Figure 13. Expanded Options for Optimization.

After pausing or stopping the optimisation, a user has the options of saving the cost function value to the cost function history for recall at a later time and saving the design as a permanent file.

In a recent study in which LOCATE was used to aid the US Navy in analysing some of the notional designs for its Integrated Command Environment (ICE), this new optimisation feature proved useful. Results of the application of the optimiser appear in the report, *Applying the LOCATE Workspace Layout Tool to the US Navy's Notional ICE Designs*, which was prepared as part of a separate contract on intelligent help (W7711-8-7480).

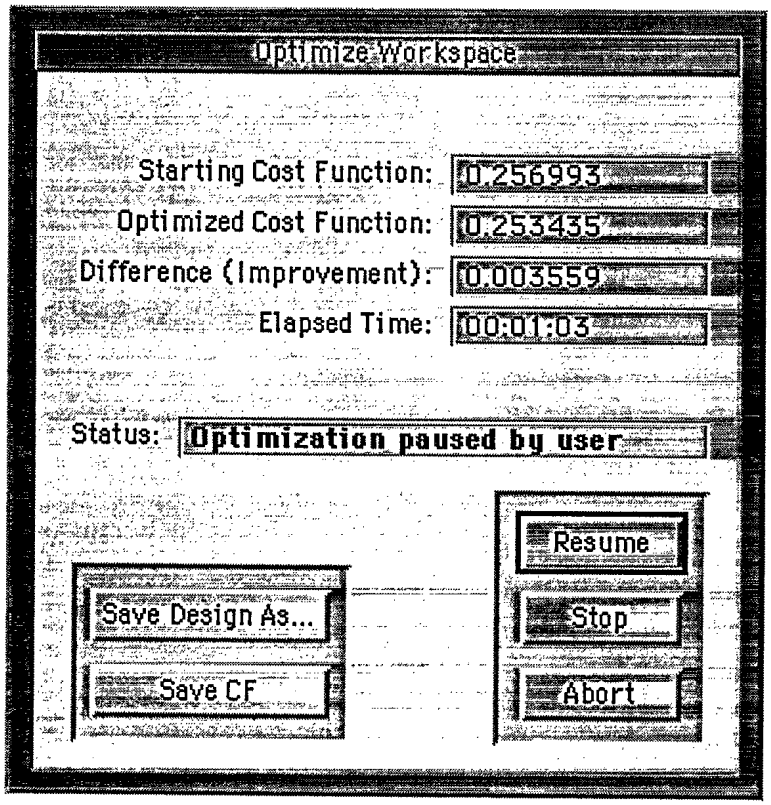


Figure 14. Monitoring the Optimization Process with LOCATE's "Optimize Workspace" Window.

Intelligent Aiding in Support of Optimisation. Intelligence has been added to the optimiser by allowing the user to specify which workstations he or she wishes to keep together or apart during the optimisation process.

Figure 13 above shows a matrix in which the names and numbers of all current workstations in the workspace, in rows on the left, are crossed with the workstation numbers in the columns at the top. In the example figure, the user has specified that the CO and TAO workstations and the DC/SS and Sensors and Weapons workstations are to be kept together; the CO is to be kept apart from the TBMD and the ASMD workstations.

The labels "Keep Together" and "Keep Apart" indicate which section of the matrix applies to which type of specification. An redundant clue makes use of the colour of the cells when clicked: green for keep together and red for keep apart.

Notice further that when a cell is clicked in one half of the matrix, an "X" is automatically entered its corresponding cell in the other half. This lets the user know that he cannot click cells to indicate to the system that it is to keep the same workstations both together and apart. Of course, if the user clicks on a coloured cell, it will turn off that cell and remove the "X" from its corresponding cell in the other half of the matrix.

The user may alter the default values for the separation distances the system will use in keeping workstations apart or together.

Once the user has decided which workstations are to be kept together and which apart and has settled on the separation distance criteria, he can then proceed to optimise the workspace.

At each step in the optimisation process, rules in LOCATE's expert system decide whether the separation distances among workstations have been violated or may be about to be violated. In the former case, it will move workstations further apart from one another; in the latter, it will not move them closer to each other.

Although LOCATE's cost function is based on default and/or user-specified priority weights that help to push workstations further apart (negative priority weights) or pull them closer together (positive priority weights), those contributions are realised as part of a mathematical formula that contains a number of other quantities.

The qualitative rules of LOCATE's expert system operate independently and more quickly to position workstations and to allow for precise specification of separation distances. Thus, their effects are more dramatic and happen more quickly than those produced by the priority weights.

The qualitative rules along with other modifications to LOCATE's expert system are found in Appendix A.

Future Work on Intelligent Analysis. Future work will develop rules for checking the consistency between the quantitative and qualitative specifications used in the optimisation described in the previous section.

Other smart additions to the optimisation process could be based on a variety of workstation attributes that might be added. For example, if workstations were identified with certain characteristics, e.g., if a workstation were a noisy workstation, it could be treated in a different way than other workstations. Further, if it were a machine not requiring a human operator that was noisy as well, it could be treated differently still.

Below are suggestions for adding various summaries that will give users a better overview of their results. From those summaries, LOCATE should be able to draw conclusions and provide even higher-level results summaries. This area is worth considerably more attention.

The user should be able to consider hypotheticals and play "what-if" games in an easy and straightforward way. LOCATE already makes it easy to change the attributes of workstations such as size, position and angle of rotation, and then recompute the cost function values. Additional hypotheticals could include various link and obstruction function values and priority weights.

For example, a user might like to see the effect of setting all distance component functions to the same function type with the same arguments and recomputing the cost function values. Alternatively, particular functions might be changed by a certain percentage for all workstations and cost function values recomputed.

The suggestions mentioned above for adding various summaries to aid in a better overview of LOCATE results are meant to provide simple, as opposed to intelligent aids to the user helping him understand and interpret the outcomes from LOCATE analyses. Those and other related suggestions are summarised below:

1. Summary Windows
   - Create summary windows for 1) Link Functions (LFs); 2) Obstruction Functions (ObFs); and, 3) Priority Weights (PWs):
     - Make the windows scrolling windows and the values in the windows editable;
     - Produce separate windows for each domain being analysed;
     - Model the windows on those used to summarise data for the US Navy in the study cited earlier;
     - The windows should also contain:
       - pop-up menus for link function selection;
       - a column that is either blank or contains a "•." The bullet indicates a note associated with the decision about the associated link function, which the user may double-click on the cell to display or edit the note;
   - Incorporate checks that PWs range from -1.00 to +1.00;
   - Add a Commit column having a check box for committing the values in the LFs and ObFs window;
   - Incorporate a Cost Function (CF) value display and a "Save CF" button.

2. Cost Function Display Window(s)
   - Add a CF value display inside each colour cost display window;
   - Add an "All" to the display pop-up inside each colour cost display window; similar to the "Close All" button;
   - Display the percentage value in each cell of the Cost Display window. (This needs further clarification as to the particular method that should be used.) The values could be presented in several ways:
     - Clicking on a cell momentarily shows the value for that cell;
     - Add a check box for displaying all values;
     - Place a box near the above check box, which, when clicked, displays all values until the mouse button is released;
   - Printed output of colour cost displays should include the title of the design, the domain, the cost function, whether it is a Link Quality or Weighted Cost view, and whether it is a normalised or non-normalised view;

3. Colour Coding Defaults and Committed Values
   - Amber and green colours are to be used to indicate to the user that a value is a default value (amber) vs. a user-entered or user-committed value (green). These colours should be used in the new summary windows for LFs, ObFs and PWs, above. In the Cost Function Checks (CFCs) Window, only the default data that has not been committed would be shown;
   - Consider if the user would ever like to know whether the values he or she has committed match the original defaults. This could be provided, if useful.

Working with Design Experts

Part of the proposed work on this contract was to identify workspace design experts and, where possible, explore with them the kinds of information they find useful in performing analyses on workspace layouts.

Some opportunity for realising the second part of this goal occurred in the context of work with the US Navy. It was hoped that some usability data might be collected from the naval personnel with whom AIM and DCIEM staff worked in applying LOCATE to their notional ICE (Integrated Command Environment) designs.

Unfortunately, the timing of that work and constraints encountered in applying LOCATE prevented a systematic collection of usability data. Some informal discussions were possible, however, and it became clear that decisions regarding priority weight data involve complex multifaceted issues.

Future work should look more closely at those decisions, seek to make that complexity explicit and explore the possibility of multiple dimensions for weighting. This could lead to modifications and additions to the LOCATE formula for computing cost functions. By extension, related decisions regarding link and obstruction function data should be explored.

Enhancing the Basic Functionality of LOCATE

Although LOCATE is a relatively mature software tool for workspace design and analysis, modifications and additions remain to be made that will make it an even more attractive tool for designers.

Accordingly, a portion of this contract was devoted to modifications and additions to LOCATE's basic functionality. The following list details the items that were to be considered; the list is in the order originally proposed. An "*" in front of an item indicates that it was addressed as part of the current work on this contract.

* • Implement manual grouping, being careful to define rules relative to LOCATE's automatic grouping procedures;
* • Resolve automatic and manual grouping issues relative to S/R nodes;
  • Revisit the built-in web browser, examining options for Netscape and Microsoft Explorer;
* • Maintain user settings for the overview Browser;
  • Allow for the re-importing of generic objects after exporting them to AutoCAD;
  • Identify the scope of the remaining work needed to provide comprehensive support for exporting and importing files to and from other CAD packages;
* • Categorise cost functions by design within the Cost Function History Window;
* • Clean up problems with the text tool that occur when fonts are changed;
* • Fix the interaction between the zero point and left, right, top, bottom workspace dimensions in the Workspace Attributes Window;
* • Allow for the saving and loading of LOCATE files to and from any folder;
* • Construct separate folders for Ltemp & Hypertext files;
  • Extend the work on a proprietary file format for LOCATE;
* • Reorganise the Link Function Window in the interest of space efficiency;
* • Make attributes windows for workstations and obstructions into tabbed windows;
* • Double clicking in the workspace should bring up the Workspace Attributes Window;
* • Implement a "Save Changes?" prompt when closing a design, if changes have been made since the last "Save.";
* • Make the appropriate distinction between the "Save" and "Save As..." commands;
* • Modify the minimise boundary command so that it is computed consistently under differing conditions;

* • Improve the "Object Count" portion of Workspace Attributes Window, particularly the link counts;
* • Add a 90° constraint to the line tool;
  • Make minor changes to windows, including the removal of close boxes from windows with "OK" & "Cancel" and impose limits on resizability of all windows so that objects are not distorted and text is not truncated;
* • Fix the dotted line problem that appears in pop-up menus.

Although many of these enhancements are straightforward and require no comment, work in a few areas needs elaboration. A complete annotated list of LOCATE's source code files, which include these and other changes, can be found in Appendix B.

Manual Grouping Procedures

Considerably more work was required that anticipated to create a manual grouping procedure and integrate it with the previously implemented automatic grouping procedure for workstations and obstructions. Further, a few areas remain in which problems arise when using the manual procedure and those should be cleared up in the next round of work on LOCATE's basic functionality.

The rules designed that define how the manual grouping procedure behaves appear in Appendix C and the results of testing the implementation of those rules are given in Appendix D.

Categorising Cost Functions Within and Across Designs

Another useful addition to analysing LOCATE's results is a modified Cost Function History Window. Although previously users were able to open several designs during a session, run cost functions and then recall any of those designs from within the Cost Function History Window, it was not clear which cost functions belonged to which designs.

Figures 15a shows a modified Cost Functions History Window in which radio buttons appear at the top of the window that allow a user to separate out cost functions according to the designs to which they belong: the "Within Designs" button.

If a user is working with several configurations that require quick comparisons of their cost functions, he may choose to move between the designs to see which produced the most or least efficient cost functions or, he may select the "Across Designs" radio button and LOCATE will thread the cost functions for all of the designs together, showing the appropriate design name next to each cost function and organizing the cost functions in the sort order chosen (Figure 15b). The previously available sort orders of "Ascending", "Descending" and "Order of Creation" apply to both the newly added "Designs" options.

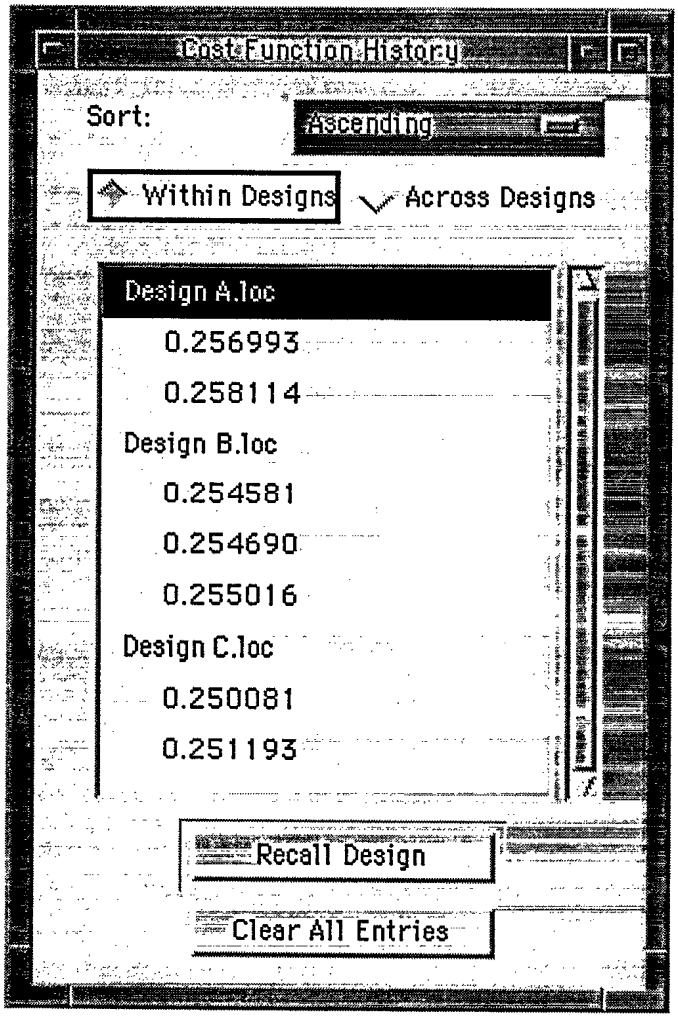


Figure 15a. Cost Function History Window: "Within Designs" Mode.

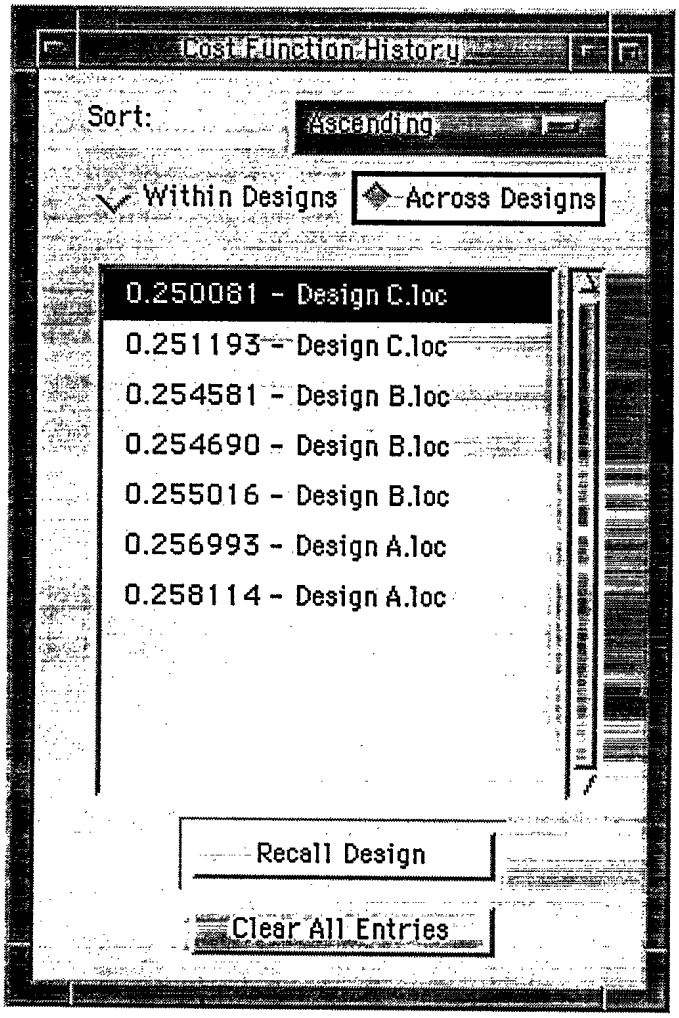


Figure 15b. Cost Function History Window: "Across Designs" Mode.

Other Additions to LOCATE

A few other additions to LOCATE's functionality are worth mentioning. First, information regarding attributes, link functions and priority weights have been combined into tabbed windows for workstations. This feature groups together a variety of data that relate to workstations and means that users do not have to open and close multiple windows to get access to those data.

The feature combines nicely with that of accessing any workstation from any other workstation's window, thus further reducing the amount of time users spend opening and closing workstation windows.

These features apply in a similar way to elemental and fixed obstructions and the reorganisation of all such windows has meant greater space efficiency.

A second point worth noting is that all files related to the LOCATE application no longer have to appear in the same folder. Users may now store LOCATE files in any folder on their hard drive. They may double-click the file or open it using the File Menu and, in so doing, invoke the LOCATE application.

Users should take care, however, to ensure that only one copy of LOCATE resides on their hard drive. As with most other applications, if more than one copy exists, one cannot be sure which copy has opened when an application-type file has been double-clicked.

Finally, LOCATE's help files are now stored in a separate "Help Files" folder and the temporary "Ltemp" files, which allow users to recall designs created during the current design session, are located in a separate "Temporary Files" folder.

Problem Fixes to the LOCATE Analysis Code

While working on additions to LOCATE's functionality, two problems emerged that required alteration to some existing LOCATE analysis code.

The first problem was a distance obstruction problem which was found to arise during the computation of link attenuation for rectilinear obstructions in the distance domain. A detailed description, sample code illustration and proposed solution, which was accepted by the client, appear in Appendix E.

The second problem related to non-normalised colour-cost displays was found to arise when a value in the link quality or weighted cost matrix fell outside the range 0.0 to 1.0.

The main symptom was that the squares in the cost matrix (representing workstation pairs) that contain such values were not being appropriately colour-coded. Instead the result appeared as a grey background "colour", similar to the grey squares of the main diagonal when first-order effects are not represented. A more detailed description and resolution of this problem are presented in Appendix F.

Summary

Work on this contract focused on adding features to LOCATE's basic functionality and on exploring and implementing intelligent aiding that would support more effective understanding and interpretation of LOCATE analyses.

A number of new features were added to LOCATE including a manual grouping procedure that interacts with the previously implemented automatic grouping within workstations and obstructions, the categorisation of cost functions in the Cost Function History Window, which makes identifying the designs a user may wish to recall an easier task, the addition of tabbed windows, which conserves space and makes data entry and modification more efficient, the option of opening a file from any location, which allows for more effective organisation of a user's files and, finally, the option of invoking the LOCATE application by double-clicking on any LOCATE document file.

Work on intelligent analysis centred around discussions about common features that might improve user understanding and interpretation of the LOCATE analysis and around intelligent ways in which the application might support that understanding and interpretation. Work from a related project helped to lay the groundwork for truly intelligent aiding by expanding the number and type of interface actions that LOCATE can now track.

Other work extended the tracking of actions to include the inference of user and system support goals that those actions imply. LOCATE is now able to track almost all user actions at the interface and infer their associated goals.

More specific work on intelligent analysis centred on the incorporation of a simple variant of the AIM Optimiser. The incorporation of this quantitative optimisation feature within LOCATE proved useful in its application in another context in work for the US Navy.

In addition to optimisation being a useful tool to generate new configurations for a design, an intelligent component was added in the form of qualitative rules that allow users to specify which workstations should be kept apart and which together during the optimisation process. As optimisation proceeds, those rules are invoked to assure that specified separation distances are not violated.

Future work on intelligent analysis will range from the incorporation of expanded summaries of LOCATE results, so as to give users a clearer overview of results, to the

addition of intelligent aiding features that will inform users about the nature of the results obtained. Tracking the user's actions and understanding his or her goals and plans, with respect to analysis, should form part of an intelligent LOCATE system.

Other work on adding intelligence to the optimiser could take the form of decision making based on various attributes associated with workstations, obstructions and other objects within the LOCATE workspace. Investigation into object attributes that are likely to impact on the relative placement of objects in a design should guide such future implementation.

Several features remain to be implemented in order to round out LOCATE's basic functionality. These include such things as a snap-to grid, alignment options for workspace objects, expansion of CAD export and import options and the ability to handle scaling issues as they impact on the creation of LOCATE objects and text. Additional work is also needed to resolve subtle problems that remain with the interaction between LOCATE's manual and automatic grouping procedures.

In spite of the work remaining to be done, considerable progress was made during this work phase in adding truly intelligent aiding to the LOCATE workspace layout tool. Future work should add considerably to LOCATE's appeal and serve as a model for other complex and many-featured applications.

References

Broadbent, G. (1988). *Design in architecture.* London: David Fulton Publishers.

Dixon, J. R. (1966). *Design engineering: Inventiveness, analysis, and decision making.* New York: McGraw-Hill.

Edwards, J. L. (1990). Intelligent dialogue in air traffic control systems. In J. A. Wise, V. D. Hopkin and M. L. Smith (Eds.), *Automation and systems issues in Air Traffic Control.* New York: Springer-Verlag.

Edwards, J. L., & Hendy, K. (1992). Development and validation of user models in an air traffic control simulation. Paper presented at the Second International Workshop on User Modeling. International Conference and Research Center for Computer Science (IBFI), Dagstuhl Castle, Germany, August 10-13, 1992.

Edwards, J. L., & D. W. Sinclair (1994, in press). Designing intelligence: A case of explicit models and layered protocols. In M. M. Taylor, F. Néel and D. Bouwhuis (Eds.), The Structure of Multimodal Dialogue II. North-Holland.

Hendy, K. C. (1984). 'Locate': A program for computer-aided workspace layout. Master's Thesis, Department of Electrical Engineering, Monash University, Clayton, Victoria, Australia.

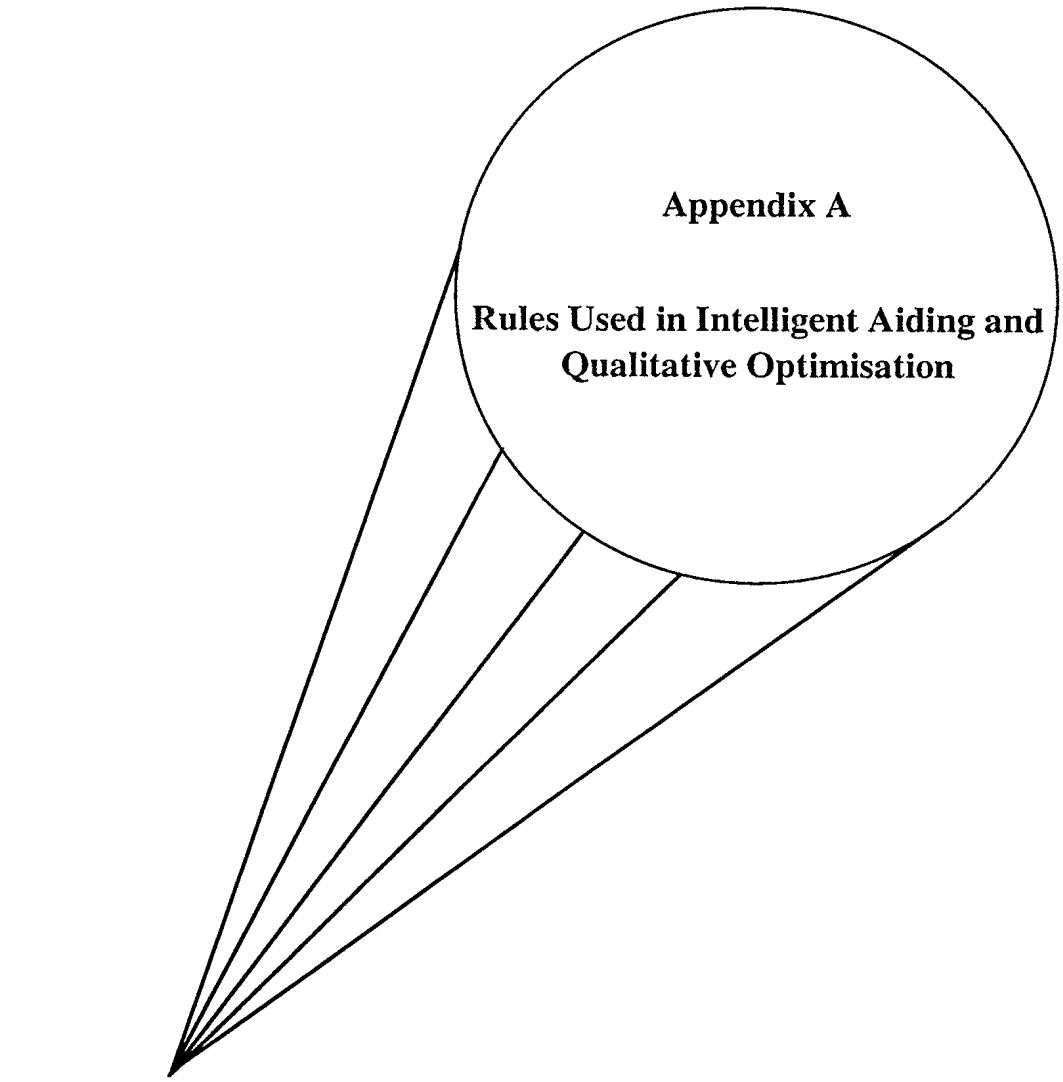
Hendy, K. C. (1989). A Model for Human-Machine-Human Interaction in Workspace Layout Problems. *Human Factors, 31(5),* 593-610.

Kautz, H., and Allen, J. (1986). Generalized plan recognition. In the Proceedings of the 5th National Conference on AI, pp. 32–37.

Küpper, D., & Kobsa, A. (1999). User-Tailored Plan Generation. Paper presented at the Seventh International Workshop on User Modeling (UM99), Banff Centre, Banff, Alberta, Canada, June 20-24, 1999.

Sage, A. P. (1992). *Systems engineering.* New York: Wiley.

Simon, H. A. (1981). *The sciences of the artificial.* 2nd edition. Cambridge, MA: MIT Press.



Appendix A

Rules Used in Intelligent Aiding and Qualitative Optimisation

Artificial Intelligence
Management and Development Corporation

Section I: LOCATE Rules Used in Intelligent Aiding

```
(@VERSION=  040)
(@PROPERTY= Delta1   @TYPE=Float;)
(@PROPERTY= Delta2   @TYPE=Float;)
(@PROPERTY= DistanceApart     @TYPE=Float;)
(@PROPERTY= DistanceTogether @TYPE=Float;)
(@PROPERTY= DistApart         @TYPE=Float;)
(@PROPERTY= DistTogether      @TYPE=Float;)
(@PROPERTY= EW       @TYPE=Integer;)
(@PROPERTY= floatval  @TYPE=Float;)
(@PROPERTY= GenCount          @TYPE=Integer;)
(@PROPERTY= General  @TYPE=Integer;)
(@PROPERTY= GenTarget         @TYPE=Integer;)
(@PROPERTY= OB       @TYPE=Integer;)
(@PROPERTY= OO       @TYPE=Integer;)
(@PROPERTY= ReturnFlag        @TYPE=Integer;)
(@PROPERTY= SpecCount         @TYPE=Integer;)
(@PROPERTY= SpecTarget        @TYPE=Integer;)
(@PROPERTY= val      @TYPE=Integer;)
(@PROPERTY= XPos1    @TYPE=Float;)
(@PROPERTY= XPos2    @TYPE=Float;)
(@PROPERTY= Xposition         @TYPE=Float;)
(@PROPERTY= YPos1    @TYPE=Float;)
(@PROPERTY= YPos2    @TYPE=Float;)
(@PROPERTY= Yposition         @TYPE=Float;)


(@CLASS=      Counter
        (@PUBLICPROPS=
                GenCount
                GenTarget
                SpecCount
                SpecTarget
        )
)
```

```
(@OBJECT=     count
        (@PUBLICPROPS=
                val
        )
)

(@OBJECT=     Distance
        (@PUBLICPROPS=
                floatval
        )
)

(@OBJECT=     EW_Check
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     EW_Increment
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     EWwindow
        (@CLASSES=
                Counter
        )
        (@PUBLICPROPS=
                GenCount
                GenTarget
                SpecCount
                SpecTarget
        )
)
```

```
(@OBJECT=     General_Check
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     General_Message
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     IncAndCheck_
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     IncCount_
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     Initialize
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     MessageFlags
        (@PUBLICPROPS=
                EW
                General
                OB
                OO
        )
)
```

```
(@OBJECT=     OB_Check
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     OB_Increment
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     OBwindow
        (@CLASSES=
                Counter
        )
        (@PUBLICPROPS=
                GenCount
                GenTarget
                SpecCount
                SpecTarget
        )
)

(@OBJECT=     OO_Check
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     OO_Increment
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)
```

```
(@OBJECT=     OOwindow
        (@CLASSES=
                Counter
        )
        (@PUBLICPROPS=
                GenCount
                GenTarget
                SpecCount
                SpecTarget
        )
)

(@OBJECT=     Opt_CheckApart
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     Opt_CheckTogether
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     OptimizerData
        (@PUBLICPROPS=
                Delta1
                Delta2
                DistanceApart
                DistanceTogether
                DistApart
                DistTogether
                ReturnFlag
                XPos1
                XPos2
                YPos1
                YPos2
        )
)
```

```
(@OBJECT=     PerformChecks
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     PutUpAlert_
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     ResetIncCount_
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     two
        (@PUBLICPROPS=
                val
        )
)

(@OBJECT=     Workstation1
        (@PUBLICPROPS=
                Xposition
                Yposition
        )
)

(@OBJECT=     Workstation2
        (@PUBLICPROPS=
                Xposition
                Yposition
        )
)
```

```
(@OBJECT=    ZeroCount_
        (@PUBLICPROPS=
                Value   @TYPE=Boolean;
        )
)


(@RULE=      R_EW_Check
        (@LHS=
                (=      (EWwindow.SpecCount)  (EWwindow.SpecTarget))
        )
        (@HYPO=      EW_Check)
        (@RHS=
                (Assign (1)      (MessageFlags.EW))
                (Assign (EWwindow.SpecTarget*5)      (EWwindow.SpecTarget))
                (Assign (0)      (EWwindow.SpecCount))
                (Assign (0)      (OBwindow.SpecCount))
                (Assign (0)      (OOwindow.SpecCount))
                (Reset  (EW_Check))
        )
        (@EHS=
                (Assign (0)      (MessageFlags.EW))
                (Reset  (EW_Check))
        )
)

(@RULE=      R_EW_Increment
        (@LHS=
                (Assign (EWwindow.SpecCount+1)      (EWwindow.SpecCount))
                (Assign (EWwindow.GenCount+1)       (EWwindow.GenCount))
        )
        (@HYPO=      EW_Increment)
        (@RHS=
                (Reset  (EW_Increment))
        )
)
```

```
(@RULE=     R_General_Message
        (@LHS=
                (Yes    (General_Check))
        )
        (@HYPO=     General_Message)
        (@RHS=
                (Assign (EWwindow.GenTarget*5)      (EWwindow.GenTarget))
                (Assign (OBwindow.GenTarget*5)      (OBwindow.GenTarget))
                (Assign (OOwindow.GenTarget*5)      (OOwindow.GenTarget))
                (Assign (0)     (EWwindow.GenCount))
                (Assign (0)     (OBwindow.GenCount))
                (Assign (0)     (OOwindow.GenCount))
                (Assign (0)     (EWwindow.SpecCount))
                (Assign (0)     (OBwindow.SpecCount))
                (Assign (0)     (OOwindow.SpecCount))
                (Assign (1)     (MessageFlags.General))
                (Reset  (General_Message))
        )
        (@EHS=
                (Assign (0)     (MessageFlags.General))
                (Reset  (General_Message))
        )
)

(@RULE=     R_GeneralCheck_1
        (@LHS=
                (>=     (EWwindow.GenCount)  (EWwindow.GenTarget))
                (>=     (OBwindow.GenCount)  (OBwindow.GenTarget))
        )
        (@HYPO=     General_Check)
)

(@RULE=     R_GeneralCheck_2
        (@LHS=
                (>=     (EWwindow.GenCount)  (EWwindow.GenTarget))
                (>=     (OOwindow.GenCount)  (OOwindow.GenTarget))
        )
        (@HYPO=     General_Check)
)
```

```
(@RULE=     R_GeneralCheck_3
        (@LHS=
                (>=     (OBwindow.GenCount)   (OBwindow.GenTarget))
                (>=     (OOwindow.GenCount)   (OOwindow.GenTarget))
        )
        (@HYPO=     General_Check)
)

(@RULE=     R_Initialize
        (@LHS=
                (Assign (0)     (EWwindow.GenCount))
                (Assign (0)     (EWwindow.SpecCount))
                (Assign (4)     (EWwindow.GenTarget))
                (Assign (3)     (EWwindow.SpecTarget))
                (Assign (0)     (OBwindow.GenCount))
                (Assign (0)     (OBwindow.SpecCount))
                (Assign (4)     (OBwindow.GenTarget))
                (Assign (3)     (OBwindow.SpecTarget))
                (Assign (0)     (OOwindow.GenCount))
                (Assign (0)     (OOwindow.SpecCount))
                (Assign (4)     (OOwindow.GenTarget))
                (Assign (3)     (OOwindow.SpecTarget))
        )
        (@HYPO=     Initialize)
        (@RHS=
                (Reset  (Initialize))
        )
)
```

```
(@RULE=     R_OB_Check
        (@LHS=
                (=      (OBwindow.SpecCount)  (OBwindow.SpecTarget))
        )
        (@HYPO=     OB_Check)
        (@RHS=
                (Assign (1)     (MessageFlags.OB))
                (Assign (OBwindow.SpecTarget*5)       (OBwindow.SpecTarget))
                (Assign (0)     (EWwindow.SpecCount))
                (Assign (0)     (OBwindow.SpecCount))
                (Assign (0)     (OOwindow.SpecCount))
                (Reset  (OB_Check))
        )
        (@EHS=
                (Assign (0)     (MessageFlags.OB))
                (Reset  (OB_Check))
        )
)

(@RULE=     R_OB_Increment
        (@LHS=
                (Assign (OBwindow.SpecCount+1)       (OBwindow.SpecCount))
                (Assign (OBwindow.GenCount+1)(OBwindow.GenCount))
        )
        (@HYPO=     OB_Increment)
        (@RHS=
                (Reset  (OB_Increment))
        )
)
```

```
(@RULE=     R_OO_Check
        (@LHS=
                (=      (OOwindow.SpecCount)  (OOwindow.SpecTarget))
        )
        (@HYPO=     OO_Check)
        (@RHS=
                (Assign (1)     (MessageFlags.OO))
                (Assign (OOwindow.SpecTarget*5)     (OOwindow.SpecTarget))
                (Assign (0)     (EWwindow.SpecCount))
                (Assign (0)     (OBwindow.SpecCount))
                (Assign (0)     (OOwindow.SpecCount))
                (Reset  (OO_Check))
        )
        (@EHS=
                (Assign (0)     (MessageFlags.OO))
                (Reset  (OO_Check))
        )
)

(@RULE=     R_OO_Increment
        (@LHS=
                (Assign (OOwindow.SpecCount+1)     (OOwindow.SpecCount))
                (Assign (OOwindow.GenCount+1)(OOwindow.GenCount))
        )
        (@HYPO=     OO_Increment)
        (@RHS=
                (Reset  (OO_Increment))
        )
)
```

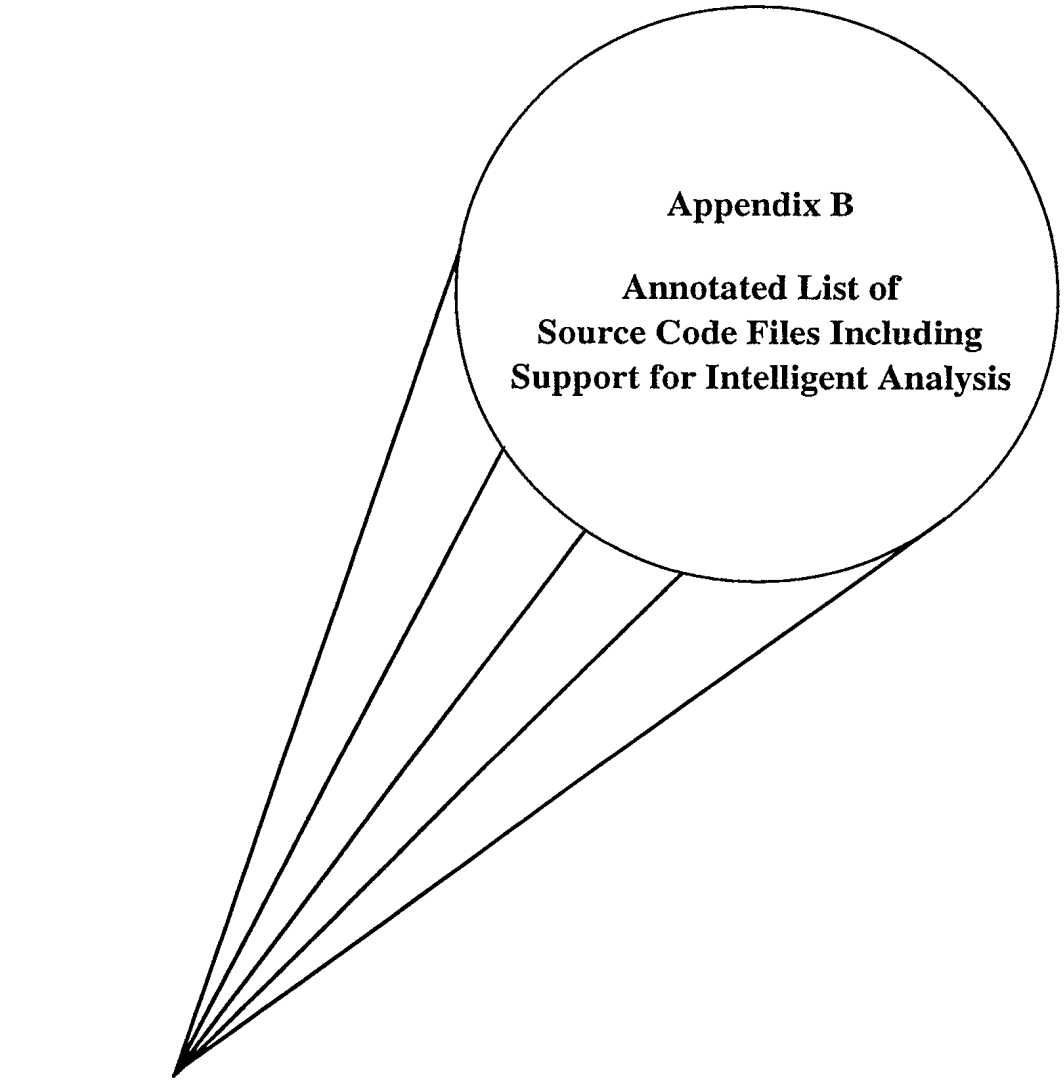
```
(@RULE=     R_Opt_CheckApart
        (@LHS=
                (>=     (SQRT((OptimizerData.XPos2-OptimizerData.XPos1)*(OptimizerData.XPos2-Optim\
izerData.XPos1)+(OptimizerData.YPos2-OptimizerData.YPos1)*(OptimizerData.YPos2-O\
ptimizerData.YPos1)))     (OptimizerData.DistApart+OptimizerData.Delta1+OptimizerDat\
a.Delta2))
        )
        (@HYPO=     Opt_CheckApart)
        (@RHS=
                (Assign (1)     (OptimizerData.ReturnFlag))
        )
        (@EHS=
                (Assign (0)     (OptimizerData.ReturnFlag))
        )
)

(@RULE=     R_Opt_CheckTogether
        (@LHS=
                (<=     (SQRT((OptimizerData.XPos2-OptimizerData.XPos1)*(OptimizerData.XPos2-Optim\
izerData.XPos1)+(OptimizerData.YPos2-OptimizerData.YPos1)*(OptimizerData.YPos2-O\
ptimizerData.YPos1)))     (OptimizerData.DistTogether+OptimizerData.Delta1+Optimizer\
Data.Delta2))
        )
        (@HYPO=     Opt_CheckTogether)
        (@RHS=
                (Assign (1)     (OptimizerData.ReturnFlag))
        )
        (@EHS=
                (Assign (0)     (OptimizerData.ReturnFlag))
        )
)

(@GLOBALS=
        @INHVALUP=FALSE;
        @INHVALDOWN=TRUE;
        @INHOBJUP=FALSE;
        @INHOBJDOWN=FALSE;
        @INHCLASSUP=FALSE;
        @INHCLASSDOWN=TRUE;
        @INHBREADTH=TRUE;
```

```
        @INHPARENT=FALSE;
        @PWTRUE=TRUE;
        @PWFALSE=TRUE;
        @PWNOTKNOWN=TRUE;
        @EXHBWRD=TRUE;
        @PTGATES=TRUE;
        @PFACTIONS=FALSE;
        @SOURCESON=TRUE;
        @CACTIONSON=TRUE;
        @VALIDUSER=FALSE;
        @VALIDENGINE=FALSE;
        @PFEACTIONS=FALSE;
        @PFMACTIONS=GLOBAL;
        @PFMEACTIONS=FALSE;
)
```

Section II: Qualitative Rules Used in Optimisation

```
(@VERSION=   040)
(@PROPERTY= Delta1   @TYPE=Float;)
(@PROPERTY= Delta2   @TYPE=Float;)
(@PROPERTY= DistanceApart     @TYPE=Float;)
(@PROPERTY= DistanceTogether @TYPE=Float;)
(@PROPERTY= ReturnFlag        @TYPE=Integer;)
(@PROPERTY= XPos1   @TYPE=Float;)
(@PROPERTY= XPos2   @TYPE=Float;)
(@PROPERTY= YPos1   @TYPE=Float;)
(@PROPERTY= YPos2   @TYPE=Float;)


(@OBJECT=     Opt_CheckApart
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     Opt_CheckTogether
        (@PUBLICPROPS=
                Value    @TYPE=Boolean;
        )
)

(@OBJECT=     OptimizerData
        (@PUBLICPROPS=
                Delta1
                Delta2
                DistanceApart
                DistanceTogether
                ReturnFlag
                XPos1
                XPos2
                YPos1
                YPos2
        )
)
```

```
(@RULE=       R_Opt_CheckApart
        (@LHS=
                (>=     (SQRT((OptimizerData.XPos2-OptimizerData.XPos1)*(OptimizerData.XPos2-Optim\
izerData.XPos1)+(OptimizerData.YPos2-OptimizerData.YPos1)*(OptimizerData.YPos2-O\
ptimizerData.YPos1)))     (OptimizerData.DistApart+OptimizerData.Delta1+OptimizerDat\
a.Delta2))
        )
        (@HYPO=       Opt_CheckApart)
        (@RHS=
                (Assign (1)       (OptimizerData.ReturnFlag))
        )
        (@EHS=
                (Assign (0)       (OptimizerData.ReturnFlag))
        )
)

(@RULE=       R_Opt_CheckTogether
        (@LHS=
                (<=     (SQRT((OptimizerData.XPos2-OptimizerData.XPos1)*(OptimizerData.XPos2-Optim\
izerData.XPos1)+(OptimizerData.YPos2-OptimizerData.YPos1)*(OptimizerData.YPos2-O\
ptimizerData.YPos1)))     (OptimizerData.DistTogether+OptimizerData.Delta1+Optimizer\
Data.Delta2))
        )
        (@HYPO=       Opt_CheckTogether)
        (@RHS=
                (Assign (1)       (OptimizerData.ReturnFlag))
        )
        (@EHS=
                (Assign (0)       (OptimizerData.ReturnFlag))
        )
)
```



Artificial Intelligence
Management and Development Corporation

The source coding listings that relate strictly to the current work appear below inside borders.

Notes:
i) .c files are C source code
.h files are C header files
.rc files are Open Interface resource description files
ii) Some extraneous files in the Locate folders are outdated and will be erased in future.

AEVENT.C
- This file contains the code to handle Apple Events for opening documents by double-clicking in the Finder.

ALLOBJINFO.C, ALLOBJINFO.RC
- The "AllObjInfo" module contains the code and Open Interface resources necessary for the "All Objects Info" window.

AOBS.C, AOBS.RC
- The "AObs" module contains the code and Open Interface resources necessary for the "Fixed Obstruction" window.

ASSIGN.C
- Original Locate C file

CFALERT.C, CFALERT.RC
- The "CFAlert" module contains the code and Open Interface resources necessary for the alert box that appears when there have been changes to the design but no cost function has been run since those changes have been made. The alert box appears before displaying the Cost Function History window .

CFBROWSE.C, CFBROWSE.RC
- The "CFBrowse" module contains the code and Open Interface resources necessary for the "Cost Function History" window.

CFCHECK.C, CFCHECK.RC
- The "CFCheck" module contains the code and Open Interface resources necessary for the "Cost Function Checks" window.

COSTCOLR.C, COSTCOLR.H, COSTCOLR.RC
- The "CostColr" module contains the code and Open Interface resources necessary for the "Cost Display Editor" window.

COSTDISP.C, COSTDISP.H, COSTDISP.RC
- The "CostDisp" module contains the code and Open Interface resources necessary for the "Cost Display" window.

COSTFN.C, COSTFN.RC
- The "CostFn" module contains the code and Open Interface resources necessary for the "Cost Function" window.

DRAWROTD.C, DRAWROTD.H
- C code for handling the drawing of rotated objects

DXF.H
- Header file with DXF format constants

DXFOPT.C, DXFOPT.H
- The "DXFOpt" module contains the code and Open Interface resources necessary for the "DXF Import Options" window.

EDITOR.C, EDITOR2.C, EDITOR.RC
- The "Editor" module contains the code and Open Interface resources necessary for the main Locate window (includes code for Diagrammer, palette, rulers).

EVAL1.C
- Original Locate C file

EWATTR.C, EWATTR.H, EWATTR.RC
- The "EWAttr" module contains the code and Open Interface resources necessary for the "Workstation" window, which now includes separate tabbed sections for Attributes, Link Functions and Priority Weights.

EXTERN.H
- Original Locate header file

FORMAT.H
- Original Locate header file

FUNCT1.C
- Original Locate C file

GOALOBJ.CPP
- Contains code for defining and handling the C++ goal object

HEADER.DXF
- Contains information that gets added to all exported DXF files

IMPRTDXF.C
- C code for handling the importing of a workspace from DXF format

INFOUPD.C, INFOUPD.H, INFOUPD.RC
- The "InfoUpd" module contains the code and Open Interface resources necessary for the "Information Update" window.

INFOWIN.C
- C code for handling the "Object Info" window

LFSUMM.C, LFSUMM.RC
- The "LFSumm" module contains the code and Open Interface resources necessary for the "Link Function Summary" window.

LINKDISP.C, LINKDISP.RC
- The "LinkDisp" module contains the code and Open Interface resources necessary for the "Link Display" window.

LOCATE.C
- Based on the original LOCATE.C file, this contains the code necessary for loading in a workspace and for computing the cost function.

LOCATE.DAT
- Open Interface compiled resources that are used by the Locate application at run-time.

LOCATE
- The Locate application

LOCATE.H
- Original Locate header file

LOCATE.μ
- Locate project for CodeWarrior 11

LOCATE.RC
- Open Interface resources in text format

LOCNEW.C
- C code for handling the creation of a new workspace

LOCNEWEW.C
- C code for handling the creation and deletion of workstations and obstructions

LOCSAVE.C
- C code for handling the saving of a workspace

LOCSAVEDXF.C
- C code for handling the saving of a workspace in DXF format

MAIN.C, MAIN.RC
- The "Main" module contains the "main" function which starts up the application.

MISC.C
- Original Locate C file

MISCRSRC.RC
- The "MiscRsrc" module contains Open Interface resources needed by the application (primarily menu and icon resources).

MULTIOBJ.C, MULTIOBJ.RC
- The "MultiObj" module contains the code and Open Interface resources necessary for the "Multiple Object Creation" window.

NEWUSER.C, NEWUSER.H, NEWUSER.RC
- The "NewUser" module contains the code and Open Interface resources necessary for the "About You" window.

OPT.C
- Optimizer code for changing positions and angles

OPTIM.C
- Original Locate C file

OPTOPT.C, OPTOPT.RC
- The "OptOpt" module contains the code and Open Interface resources necessary for the "Optimizer Options" window.

OPTSET.C, OPTSET.RC
- The "OptSet" module contains the code and Open Interface resources necessary for the "Optimizer Settings" window.

OPTSTAT.C, OPTSTAT.RC
- The "OptStat" module contains the code and Open Interface resources necessary for the "Optimizer Status" window.

OPTSWAP.C
- Optimizer code for swapping workstations

ORIGIN.C
- Original Locate C file

OTHEROBJ.C, OTHEROBJ.RC
- The "OtherObj" module contains the code and Open Interface resources necessary for the "Other Object" window.

OUTPUT.C
- Original Locate C file

PALEDIT.C, PALEDIT.RC
- The "PalEdit" module contains the code and Open Interface resources necessary for the "Palette Editor" window.

PRINTPREV.C, PRINTPREV.RC
- The "PrintPrev" module contains the code and Open Interface resources necessary for the "Print Preview" window.

RULEEW.C, RULEEW.H, RULEEW.RC
- The "RuleEW" module contains the code and Open Interface resources necessary for the window that informs the user about double-clicking to bring up Workstation attributes.

RULEGEN.C, RULEGEN.H, RULEGEN.RC
- The "RuleGen" module contains the code and Open Interface resources necessary for the window that informs the user about double-clicking to bring up object attributes.

RULEOB.C, RULEOB.H, RULEOB.RC
- The "RuleOB" module contains the code and Open Interface resources necessary for the window that informs the user about double-clicking to bring up Obstruction attributes.

RULEOO.C, RULEOO.H, RULEOO.RC
- The "RuleOO" module contains the code and Open Interface resources necessary for the window that informs the user about double-clicking to bring up Other Object attributes.

RULER.C, RULER.RC
- The "Ruler" module contains the code and Open Interface resources necessary for the "Ruler" window.

SMRTHELP.C, SMRTHELP.RC
- The "SmrtHelp" module contains the code and Open Interface resources necessary for the "Smart Help" window.

SPLASH.C, SPLASH.RC
- The "Splash" module contains the code and Open Interface resources necessary for the startup screen.

SPLASH2.C, SPLASH2.H, SPLASH2.RC
- The "Splash2" module contains the code and Open Interface resources necessary for the "More on Locate" window.

START.C, START.RC

- The "Start" module contains the code and Open Interface resources necessary for the usability "Start" window.

STARTUP.C, STARTUP.RC

- The "Startup" module contains the code and Open Interface resources necessary for the help reminder at startup.

SYSMODL.C, SYSMODL.RC

- The "SysModl" module contains the code and Open Interface resources necessary for the "System Model" window.

TASKMODL.C, TASKMODL.RC

- The "TaskModl" module contains the code and Open Interface resources necessary for the "Task Model" window.

USERMODL.C, USERMODL.RC

- The "UserModl" module contains the code and Open Interface resources necessary for the "User Model" window.

WEBBROWS.C, WEBBROWS.H, WEBBROWS.RC

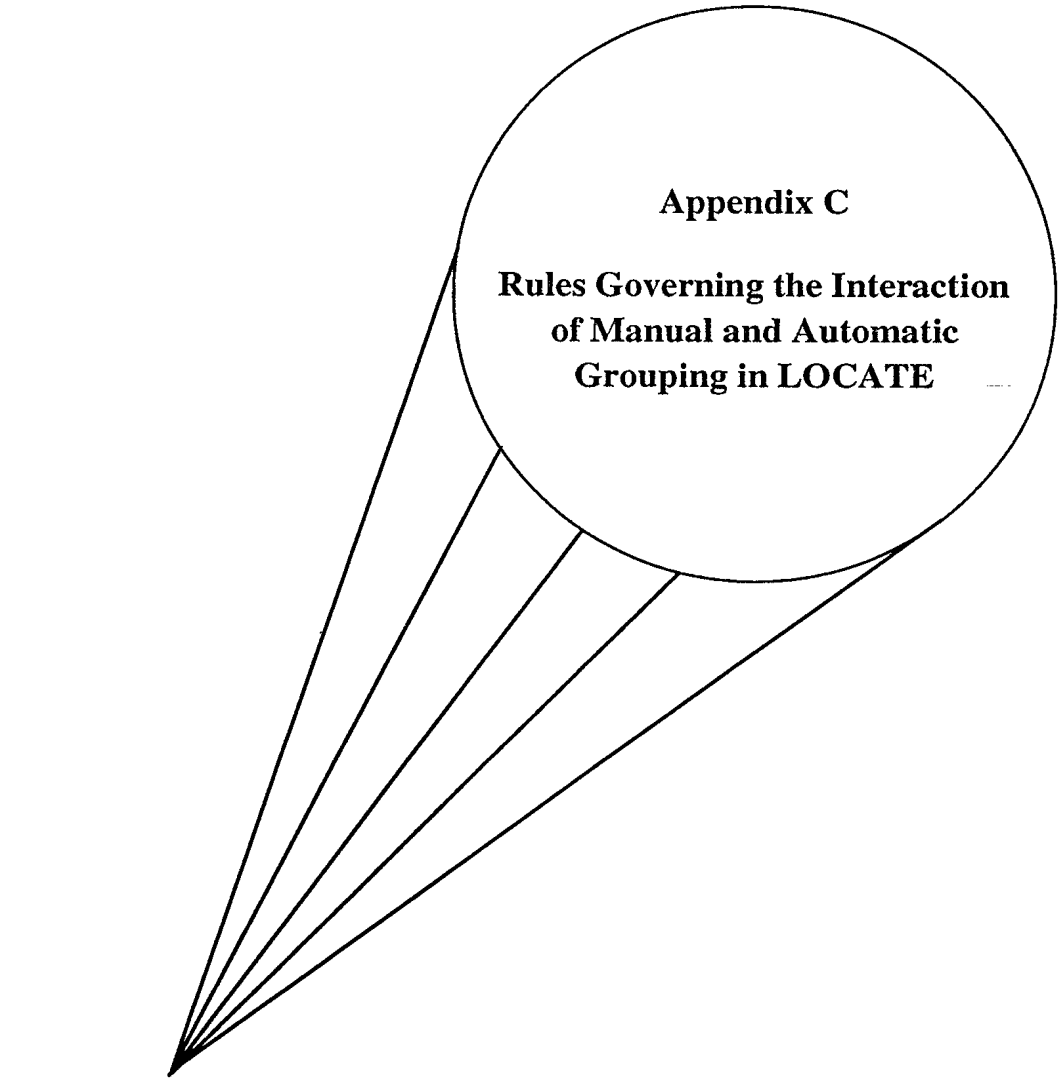
- The "WebBrows" module contains the code and Open Interface resources necessary for the "Web Browser" window.

WOBS.C, WOBS.RC

- The "WObs" module contains the code and Open Interface resources necessary for the "Elemental Obstruction" window.

WSATTR.C, WSATTR.RC

- The "WSAttr" module contains the code and Open Interface resources necessary for the "Workspace Attributes" window.



Appendix C

Rules Governing the Interaction of Manual and Automatic Grouping in LOCATE

Artificial Intelligence
Management and Development Corporation

I. Elemental Workstations (EWs) and S/R Nodes

- Grouping multiple EWs is permitted;

- Grouping items within EWs is permitted with the following restrictions:
  - grouping of multiple EObs is permitted;
  - grouping of EObs with generic (or customised) objects is permitted when the generic objects are not contained within any EOb;
  - grouping of multiple generic (or customised) objects is permitted when those objects are all within a single EOb;
  - grouping an S/R node inside an EOb is permitted;
    Qualifier: S/R nodes are not considered part of an EOb for purposes of CF computation
  - grouping an S/R node with one or more EOb is permitted when the S/R node is not within any of the EObs;
  - grouping an S/R node with generic objects is permitted when both are inside an EOb;
  - grouping an S/R node with generic objects is permitted when both are not within any EOb;

- Grouping items contained within more than one EW is not permitted; the grouping of such items is done by grouping EWs;

- Grouping EWs and Fixed Obstructions (FObs) is permitted;

- Grouping EWs and generic (or customised) objects is permitted, so long as the generic (or customised) objects are not in another EW, EOb or FOb;

- Grouping items in an EW with an FOb is not permitted; the grouping of such items is done by grouping EWs and FObs;

- Grouping items in FObs with an EW is not permitted; the grouping of such items is done by grouping EWs and FObs.

II. Elemental Obstructions (EObs)

- (see second bullet in I., above);

III. Fixed Obstructions (FObs)

- Grouping multiple FObs is permitted;
- Grouping items within FObs is permitted with the following restrictions:
  - grouping of multiple generic (or customised) objects is permitted when those objects are all within a single FOb;
- Grouping FObs and generic (or customised) objects is permitted, so long as the generic (or customised) objects are not in another FOb, EW, EOb;
- (see last two bullets in I., above)

IV. Generic (Customised) Objects (GObs)

- Grouping generic objects to form a customised object is permitted;
  - Such objects should be definable as a **new type** of object.
- Grouping multiple generic (or customised) objects is permitted when those objects are not contained within any other object;
- (see also second bullets in I. & III., above)

V. Data Access to Manually Grouped Objects

- When an EW is grouped with another EW or with an FOb, no access is permitted to Link Function and Priority Weight data for the individual EWs or to Obstruction Function data for the FOb;
- The above rule applies both to the grouping of multiple EObs and the grouping of multiple FObs;
- The rule also applies to the grouping of EWs, EObs, or FObs with single or multiple generic (or customised) objects;
- When multiple generic (or customised) objects are grouped, no access is permitted to the attributes of the individual objects;

VI. Menu Command Access

- When a grouped object is selected that includes multiple EWs, EObs, FObs, generic or customised objects, the menus that allow access to the attributes, LF, PW and Obstruction data for those objects will be dimmed;
- [**Notes**: If an object is selected, for which access is permitted, other object types in the View Menu are dimmed.

  If the object is not an EW, then the Link Function and Priority Weight items in the Data Menu are dimmed.

"Obstruction Functions" are included in the Data Menu to bring up the Obstruction Function portion of the Obstruction Attributes Window.]

The above has been minimally implemented, i.e., for simple EW, EOb and FOb and generic object selections

VII. Manipulating Grouped Objects

- Grouped (or multiply selected) generic (or customised) objects can have standard editing functions applied to them. Thus, they may be copied, pasted, cut, duplicated and cleared;

- Those same objects may be repositioned, rotated and resized. When a grouped generic (or customised) objects are double-clicked or selected and "Object" selected from the View Menu, an attribute window will be displayed that contains the same attributes as generic objects. The title of the window will be "Custom (Grouped) Object, as opposed to the tile for simple generic objects such as, "Chair", "Desk, etc.
  If the objects are simply selected but not grouped, double-clicking will have no effect and the word, "Object" in the View Menu will be dimmed (along with other items in that Menu)

- Grouped objects that contain any of LOCATE's complex objects, i.e., EWs, EObs or FObs, may only be deleted. Thus, a user will not cut, copy, paste or duplicate those grouped objects. If he attempts to do so, an alert will be posted to the effect that the function is not allowed. (A better alternative --> dim the editing functions!)
  If the user wishes to delete such grouped (complex) objects, he may do so but will not be allowed to undo his deletion. A warning will be posted to that effect and the user allowed to cancel or proceed with the operation.
  If the user ungroups the object, then he may once again apply standard editing functions to its individual elements;

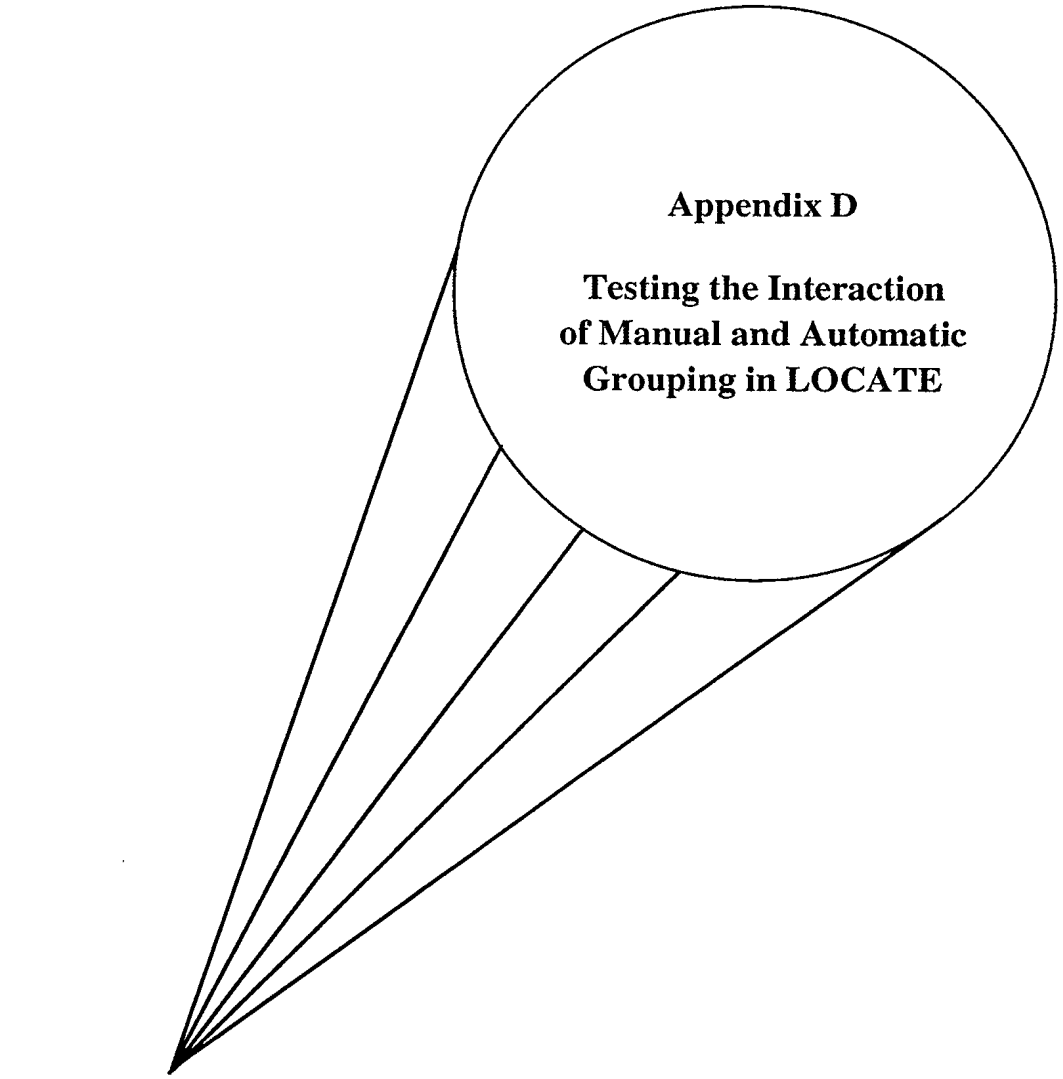
- Further, grouped (complex) objects may be repositioned and rotated but may not be resized. As with grouped generic objects, an attributes window may be displayed by double-clicking or choosing "Object" from the View Menu. The attributes window will allow the user to specify a new position for the object and rotate it but the width and height edit text boxes will be dimmed.

- These same rules apply as one would expect to groups of grouped objects.

VIII. Dimmed Items in Attribute Windows

- Drop down menus in various attributes and data windows will be modified when objects are grouped. Any EW, EOb or FOb that has been grouped will not be accessible from the attributes and data windows of other such objects that are not grouped. The names of the grouped items in the drop-down menus of attributes and data windows will be dimmed;

- Similarly, a user will not be allowed to type in the number of those objects in the attributes and data windows of other EW, EOb or FOb objects that have not been grouped.



Appendix D

Testing the Interaction of Manual and Automatic Grouping in LOCATE

Artificial Intelligence
Management and Development Corporation

The manual grouping procedures took much more time that had been anticipated to incorporate into LOCATE. There remain a number of problems with those procedures that need to be addressed and, because time did not permit a return to these issues during this contract period, that work must wait for future efforts.

The items below reflect problems encountered with the procedures that need to be fixed. Testing of other rules still need to be addressed and any problems uncovered will need to be fixed.

Given the difficulty of accommodating the manual and automatic grouping procedures, it is not clear how long that effort will take. It is hoped that the problems are only local and that the fixes will not involve more general solutions that require any fundamental changes in the LOCATE code.

I. Elemental Workstations (EWs) and S/R Nodes

- Grouping multiple EWs is permitted;

  In attempting to group EWs, you cannot select them with the marquee, since that selects not only the EW, but all its contents. You have to Shift-Click each one. This needs to be reviewed.

  What is a problem here is that when you do Shift-Click each EW, then group, then ungroup, the ungrouped EWs have everything in each EW selected. That should not happen.

- Grouping items within EWs is permitted with the following restrictions:
  - grouping of multiple EObs is permitted;

    Placing a generic object inside an EOb, minimised boundary and system crashed (not able to reproduce). Grouped EObs can be dragged outside of an EW—that should not be allowed.
  - grouping of EObs with generic (or customised) objects is permitted when the generic objects are not contained within any EOb;

    You seem to be able to select an EOb and then Shift-select objects within it. I'm not sure this should be allowed??

    Also, grouped EObs should continue to behave as EObs, that is, they should be selectable directly, i.e., the user should not have to hold down the Cmd key to select them. Perhaps if they are grouped with generic objects that are outside of other EObs or with S/R nodes, then a Cmd-Click should be necessary.
  - grouping of multiple generic (or customised) objects is permitted when those objects are all within a single EOb;

- grouping an S/R node inside an EOb is permitted;
  Qualifier: S/R nodes are not considered part of an EOb for purposes of CF computation

  I'm assuming this meant that the S/R node could be grouped inside an EOb, although the language used is vague. Putting the S/R node inside an EOb is currently allowed and is done with automatic grouping.

  Other interpretations arise when I first tried to interpret this bullet. In testing, it does not appear that one can group S/R (outside of EObs) with EObs, but it does appear that one can group them with generic items inside an EOb. I'm not sure this should be allowed. Also, one can group S/R nodes with generic objects as long as those Generic objects are inside the same EW but not contained within another EOb in that EW. A problem arises here. If you group the S/R node with a desk that is to the right of the S/R node, you can move the grouped object to the right and outside of the EW. In that case, the S/R node portion of the grouped object remains inside of the EW. However, if you move it to the left and outside, the S/R node portion of the grouped object ends up outside the EW. There needs to be a way for the system to know when the S/R portion of the grouped object comes to the edge of the EW and stop it there.

  I also tried doing a minimise boundary when the grouped object was outside of the EW and, unlike some other experiments in which the boundary expands to include objects, here it contracted to the centre point of the desk portion of the grouped desk and S/R, leaving most of the grouped object outside.
- grouping an S/R node with one or more EOb is permitted when the S/R node is not within any of the EObs;

  See above; this does not appear to be allowed. No, I'm wrong in a measure. If you first select the EOb and then Cmd-Shift-Click the S/R node, the selection is permitted and then grouping is possible. However, if you first select the S/R node and then attempt to Shift-Click on the EOb, no selection is permitted.
- grouping an S/R node with generic objects is permitted when both are inside an EOb;

  This is quite impressive!! Also tried the minimise boundary and it seemed to work fine.
- grouping an S/R node with generic objects is permitted when both are not within any EOb;

  When an S/R node is within an EOb and the generic object is outside, they can both be selected but not grouped. The same thing appears to be the case when the situation is reversed—Those situations are fine.

- Grouping items contained within more than one EW is not permitted; the grouping of such items is done by grouping EWs;

  Selectable, but not groupable—Good!

- Grouping EWs and Fixed Obstructions (FObs) is permitted;

  This produced a real problem (see similar problem in next item and a possible explanation)! I first grouped an EW with one Fob, moved it around and everything was OK. Then I ungrouped, noticed the problem that everything in the EW was selected, de-selected everything, then selected the EW, and two FObs. When I then tried to group, I got a query about placing the object outside the WS, which of course was not an issue. Saying "No" didn't do anything and if I tried to move the object, I again got the prompt. Saying, "Yes" results in the EW and, I believe, the two FObs disappearing, leaving the contents of the EW including the S/R node. I then tried to select the S/R node and move it. I got the same prompt about placing the object outside the WS; I said. "Yes" and LOCATE quit.

  I saved the file before the quit. It's entitled, "Problem1.loc". However, trying to open this file produced a quit.

- Grouping EWs and generic (or customised) objects is permitted, so long as the generic (or customised) objects are not in another EW, EOb or FOb;

  I was able to select an EW and an S/R node in another EW. When I moved the EW, the S/R node followed and could be dragged outside of the second EW. I de-selected the two and when I moved the S/R node, it jumped back to the edge of the second EW.

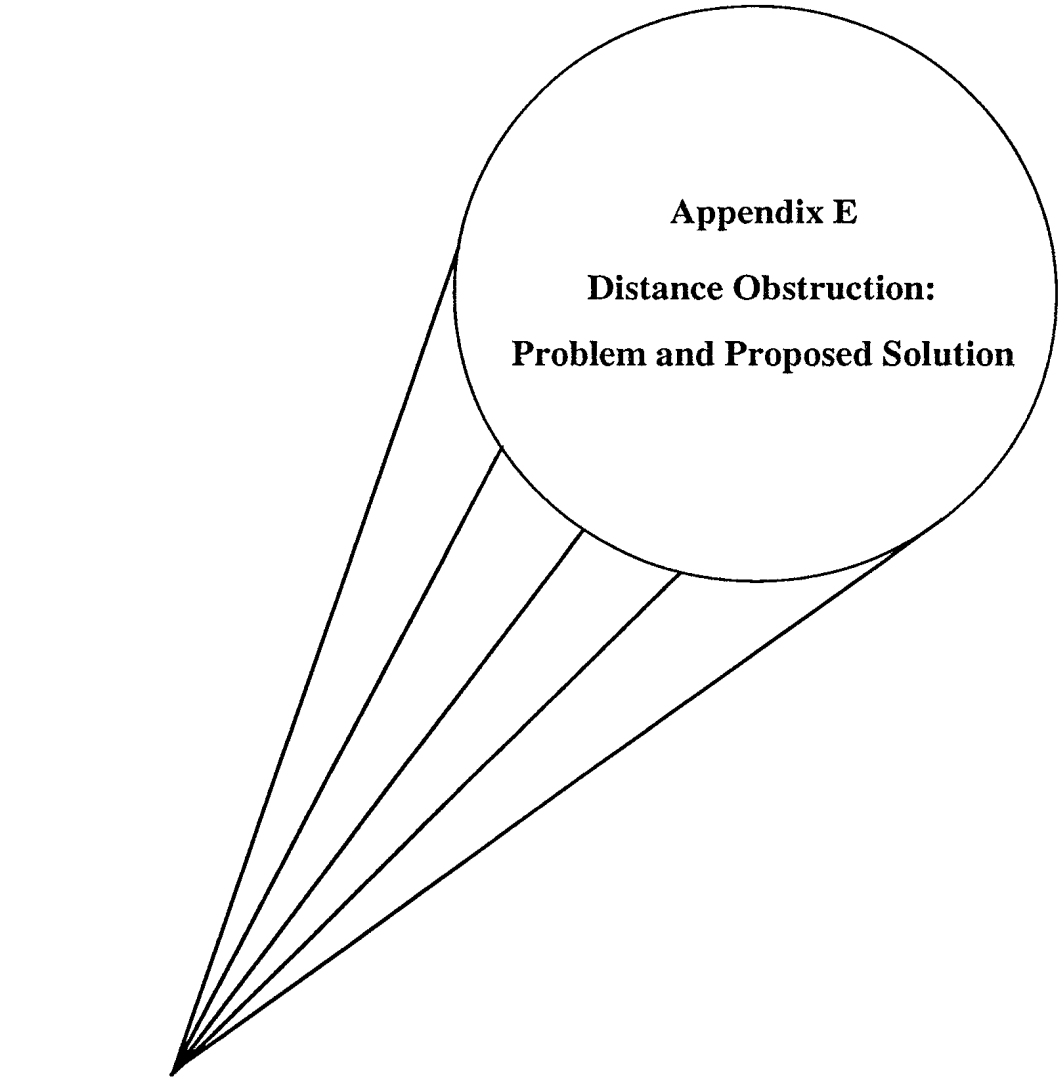
  This created another problem similar to the one in the previous note. Perhaps both problems are related. It seems that when I grouped an EW with an FOb, the EOb within the EW did not get grouped. I noticed this when I tried to move the selection. The EW and the FOb moved but not the EOb within the EW.

  When I selected the EOb and tried to move it, I got the query about moving an object outside the WS. When I answered "Yes", the EOb and its contents disappeared (like the EW in the previous example). This time I had the Object Info Window open and noticed that the EOb was positioned at X= 109.<something> This is similar to the problem that we were having a while ago when grouping and moving multiple EWs.

  This problem file was saved as "Problem2.loc". As in the previous problem, trying to re-open that file caused LOCATE to quit.

- Grouping items in an EW with an FOb is not permitted; the grouping of such items is done by grouping EWs and FObs;

  This seems to work OK, but by dragging the FOb when an FOb and an EOb are selected allows one to drag the EOb outside its EW. Also, something funny happened when I de-selected the two objects. I then selected the EOb and moved it slightly. It jumped back to the edge of its EW, which I expected. However, when I moved it again (I believe), the chair inside it (there was both a chair and desk) didn't move with it and became detached from it! I was able to move it back into the EOb with no ill effects...it seems.



Appendix E

Distance Obstruction: Problem and Proposed Solution

Artificial Intelligence
Management and Development Corporation

The Problem

This problem occurs during the computation of link attenuation for rectilinear obstructions in the distance domain (no problem was detected for elliptical obstructions). The example file "ObstructionTest.loc" (part of this attachment) illustrates the problem. In that example, the cost function value is unaffected by the location of the obstruction.

In the code, the attenuation routine (ATTEN34) calls the routine NEWTON1, which uses Newton's method to find the roots of a particular function. At the beginning of NEWTON1 some checks are made on the input values, and one of those checks fails under some conditions, resulting in no link attenuation for that particular obstruction.

In tracking down the source of the problem, the current code was compared with the original Locate C code and was found to be unchanged (there is, of course, the somewhat remote possibility that the problem may involve interaction with code elsewhere that has been changed). Thus, the problem may have existed all along. Further, the original Fortran code was examined and appears equivalent to the new C code.

The NEWTON1 routine is reproduced below in the original Fortran and the relevant line flagged with a red arrow (➡). The "GOTO1" handles failure, and there is a comment in the code suggesting that failure signals no intersection between the link and the obstruction, which is clearly not the case in the example above.

What needs to be made clear is what the check is ensuring and whether there might be an error in its formulation. Incidentally, when the code is commented out, the problem seems to disappear, but "removing" it might result in invalid cases being accepted, which would otherwise be rejected.

Those last two points are addressed in the "Proposed Solution", below.

```
C     ****************************************************************
      SUBROUTINE NEWTON1(ARG1,ARG2,ARG3,SLOPE,CINTER,XDASH,YDASH,IFLAG)
C     THIS SUBROUTINE GENERATES A NUMERICAL SOLUTION FOR THE PROBLEM
C     F(X) = (X/A)**2N + ((MX+C)/B)**2N - 1 = 0. NEWTON'S METHOD IS
C     USED.
C
C     called by: Atten
C
C     ARG1 = A
C     ARG2 = B
C     ARG3 = N
C     EXP1 = 2N
C     EXP2 = 1/2N
C     EXP3 = 2N-1
C
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION XDASH(2),YDASH(2)
      COMMON/EXPON/ EXP1,EXP3
      COMMON/ERROR/ EPSILON

        YMAX = 10.**(18.D00/ARG3)
        I = 1

C
c!(+- b - C )/M ! < a
C
C
      IF(DABS(SLOPE) .GT. EPSILON)THEN
            IF((DABS((ARG2-CINTER)/SLOPE) .GT. ARG1) .AND.
     +           (DABS((-ARG2-CINTER)/SLOPE) .GT. ARG1)) GOTO 1
      ELSE
            IF(DABS(CINTER) .GT. ARG2) GOTO 1
      ENDIF

      IF( DABS(CINTER) .LE. ARG2)THEN
            X = - ARG1
      ELSE
            X = - ARG1*DSIGN(1.D00,SLOPE)*DSIGN(1.D00,CINTER-ARG2)
c!!C! <= b
      ENDIF

      Y = SLOPE*X+CINTER

      YOVB = Y/ARG2
C
C     TEST FOR STEEPEST SLOPE (AVOID OVERFLOWS)
C

91    IF(DABS(YOVB).GE. YMAX)THEN
            X = (X - 2.D00*CINTER/SLOPE)/3.D00
C           X = (X - CINTER*SLOPE)/(SLOPE**2 + 1.D00)
            Y = SLOPE*X+CINTER
            YOVB = Y/ARG2
        GOTO 91
        ENDIF
```

```
92     IF(I .LE. 2)THEN

         XOVA = X/ARG1
         YOVB = Y/ARG2

         XOVA2N = XOVA**2
         XOVA2N = XOVA2N**(ARG3-1.D0)

         YOVB2N = YOVB**2
         YOVB2N = YOVB2N**(ARG3-1.D0)


         F = XOVA2N*XOVA**2 + YOVB2N*YOVB**2 - 1.D0

       IF (DABS(F).LT.EPSILON) THEN
         XDASH(I) = X
         YDASH(I) = Y
         I = I + 1

         IF( DABS(CINTER) .LE. ARG2)THEN
             X =  ARG1
         ELSE
             X = (DSIGN(1.D00,CINTER-ARG2)*(ARG2+EPSILON)-CINTER)/SLOPE
c!!C! <= b
         ENDIF

         Y = SLOPE*X+CINTER

         YOVB = Y/ARG2

C
C      TEST FOR STEEPEST SLOPE (AVOID OVERFLOWS)
C
93       IF(DABS(YOVB).GE. YMAX)THEN
             X = (X - 2.D00*CINTER/SLOPE)/3.D00
C            X = (X - CINTER*SLOPE)/(SLOPE**2 + 1.D00)
             Y = SLOPE*X+CINTER
             YOVB = Y/ARG2
         GOTO 93
         ENDIF

       ELSE
         FDASH = 2.D00*ARG3*(XOVA2N*XOVA/ARG1+SLOPE*YOVB2N*YOVB/ARG2)
       IF (DABS(FDASH).LT.EPSILON) THEN
c!NO REAL ROOTS
         GOTO 1
       ELSE
c!NEW VALUE
         X = X-F/FDASH
         Y = SLOPE*X+CINTER
       END IF
       IF (DABS(X).GT.ARG1) THEN
c!NO REAL ROOTS
         IFLAG = 0
         RETURN
       END IF
c!DABS(F) < EPSILON
       END IF
```

```
c!WHILE
      GOTO 92
      ENDIF

      RETURN

 1      IFLAG = 0
        RETURN
      END
C       ***************************************************************************
```

Proposed Solution

Further investigation has shed more light on the problem, and what follows is offered by way of a possible solution.

Figure 1 on the next page shows an obstruction in its local coordinate system with a width of 2*a and a height of 2*b. The link line passes through the top and bottom edges of the obstruction. The problem, as described above, involves a check done at the beginning of the NEWTON1 algorithm. This check examines the quantities (b-c)/m and (-b-c)/m, the physical meaning of which is shown in Figure 1.

Specifically, (b-c)/m is the x-coordinate value where the link passes through the top of the obstruction and (-b-c)/m is the x value where it intersects the bottom edge. The check ensures that at least one of these two quantities is between –a and a, meaning that the link passes through either the top or bottom edge, or both. If the check fails, LOCATE concludes that the link and the obstruction do not intersect.

The above problem seems to occur in situations like that depicted in Figure 2. Both of the quantities examined are outside of the range from –a to a, leading to the conclusion that there is no intersection between the link and the top and bottom edges.

In fact, despite the accuracy of that conclusion, it is clear that the link does pass through the obstruction, that is, through its left and right sides. The contention (and this needs your confirmation, Keith) is that, if the two values, (b-c)/m and (-b-c)/m, have opposite signs, then the link must pass through the obstruction, even if both values are outside the [-a, a] range.

If you concur with this analysis so far, a solution might be first, to perform the check in its original form and then, if that fails, to examine the signs of the two quantities and, if they are different, conclude that an intersection has occurred.

As a test, that change was implemented and the results do appear to correct the problem; it would be nice, however, to have your confirmation that this line of reasoning is sound.

Finally, could you confirm the assumption that the checks, though useful in optimising performance, are not necessary to produce an accurate cost function value. As you will recall (from the "Problem" section, above), when the checks were commented out, the programme still produced a cost function value.

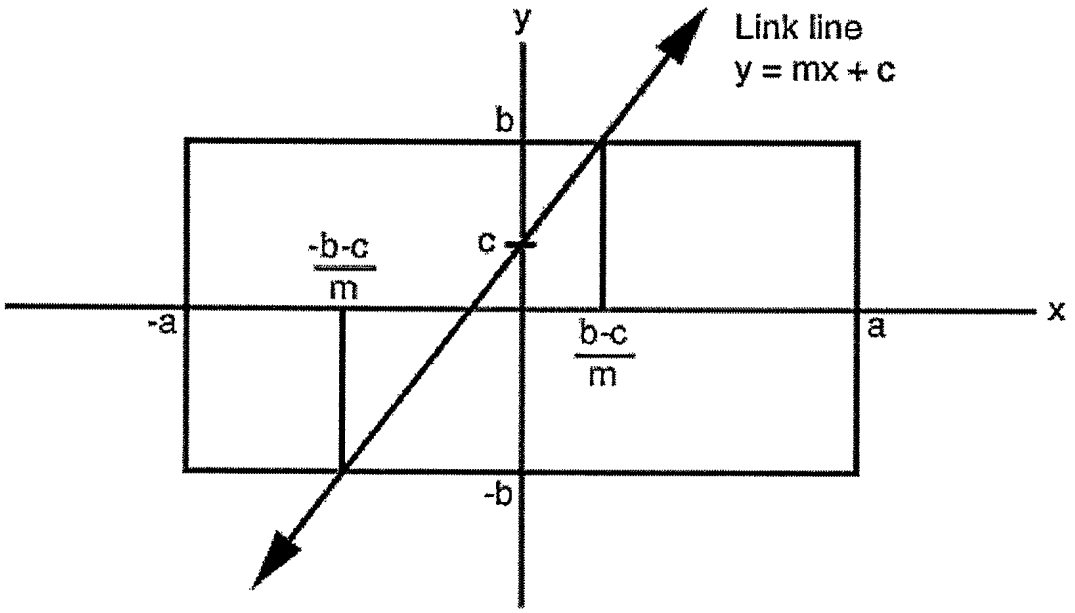


Figure 1: Link intersects top and bottom edges of the obstruction.

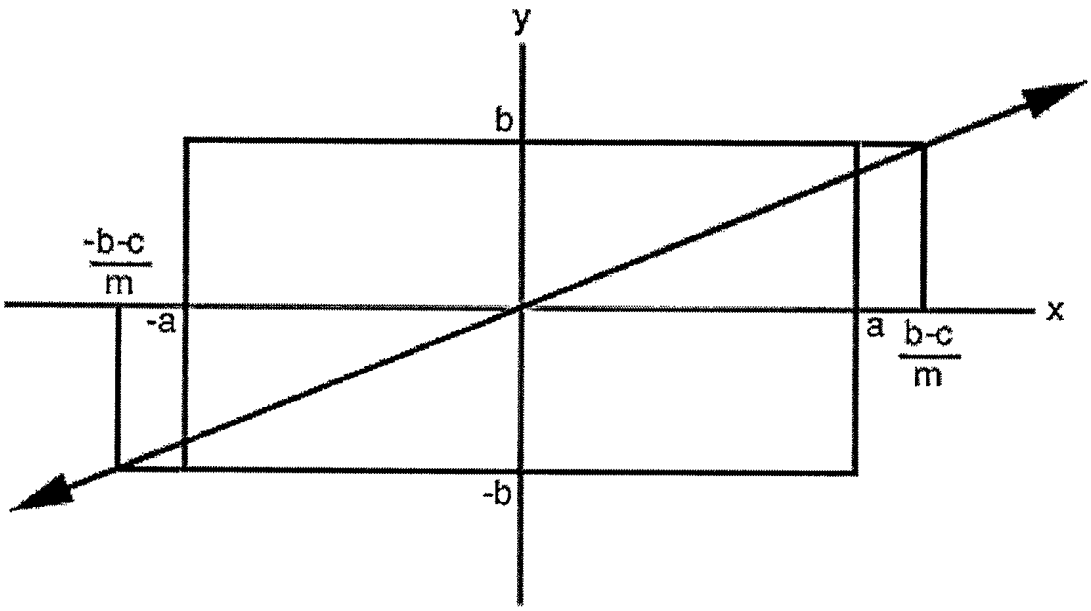
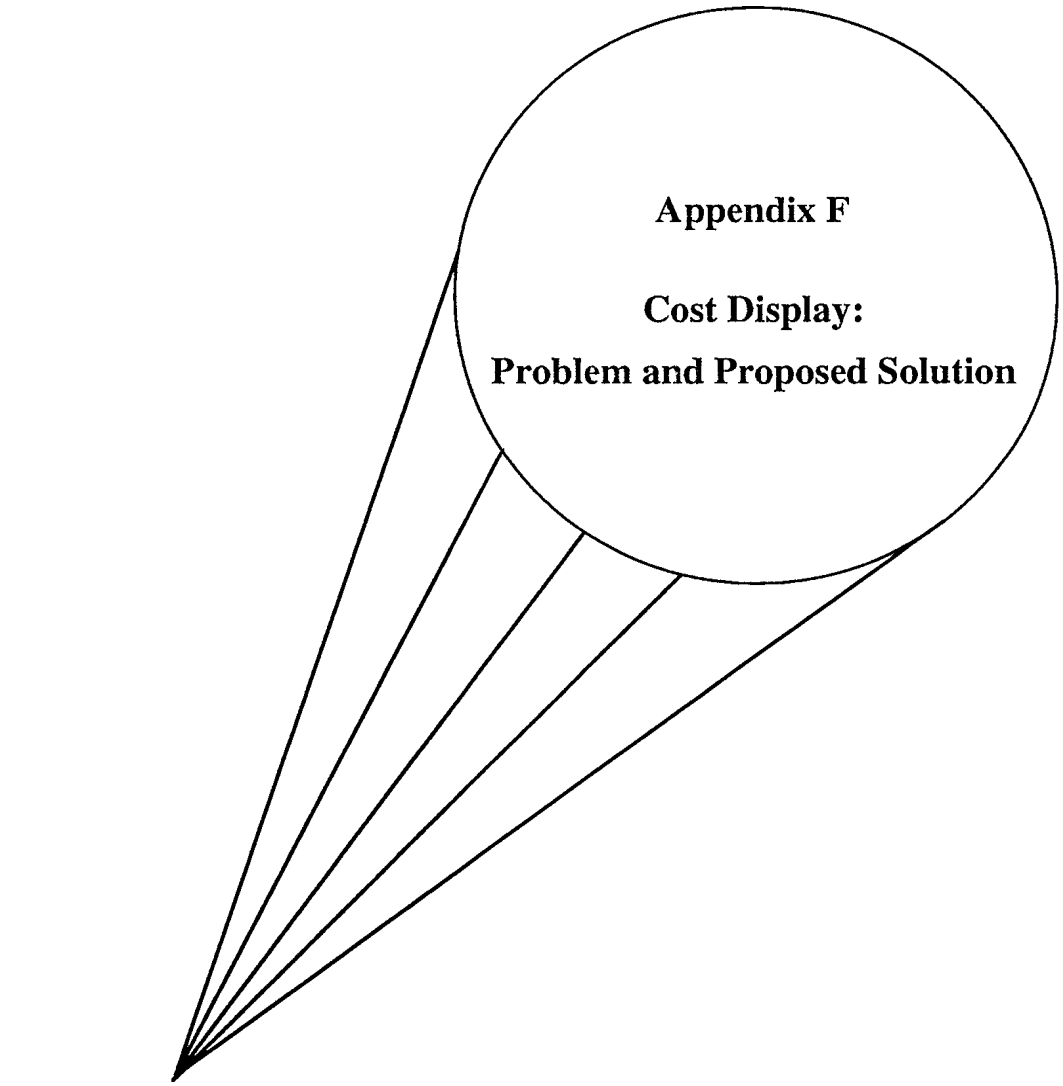


Figure 2: Link intersects left and right sides of the obstruction.



Appendix F

Cost Display:
Problem and Proposed Solution

Artificial Intelligence
Management and Development Corporation

A problem arises in the colour cost displays when a value in the link quality or weighted cost matrix is outside the range of 0.0 to 1.0.

In the old normalised views the situation never occurs because the normalisation routine ensures that values are always within the zero to one range. In the non-normalised cost displays, however, the situation does occur and the squares (representing workstation pairs) containing such values are not drawn (colour-coded). The result is a grey background colour that appears similar to the grey squares of the main diagonal when no first-order effects are being represented.

In the "Table-KH" design, values for the auditory link quality between workstations 2 and 6 were approximately -2.2×10^{-16}, which is most likely a consequence of limited precision in the computer's floating point representation.

To allow for the presence of such values, the lower bound has been extended to -10^{-15} and the upper bound to $1 + 10^{-15}$. This should effectively eliminate non-normalised link quality values that are less than zero and greater than one, due to precision error. If values begin to appear outside that range, however, it will be harder to attribute them to a precision problem and it would be necessary to confirm that the computations are sound.

To identify any values that might fall outside that range (-10^{-15} to $1 + 10^{-15}$), a **black** colour coding will be used for **suspect squares**. (The previous, default grey colour made detecting them difficult and, in fact, they were only detected as a result of a question about the minus value when examining the numerical matrix for link quality.)

At some point we might consider filtering out values smaller than 10^{-15} altogether, and rounding them to zero or one. It would be necessary, of course, to establish that those values have no effect on the cost function, which is currently being assumed.

#511955