CONTROL OF ASYNCHRONOUS TACTICAL GRAPHICS DISPLAY

Harlan H. Black
CENTER FOR C3 SYSTEMS

December 1987

DISTRIBUTION STATEMENT
Approved for public release;
distribution is unlimited.

CECOM
US ARMY COMMUNICATIONS-ELECTRONICS COMMAND
FORT MONMOUTH, NEW JERSEY 07703-5000
CONTROL OF ASYNCHRONOUS TACTICAL GRAPHICS DISPLAY (U)

13b. TIME COVERED
FROM TO
14. DATE OF REPORT (Year, Month, Day) 1987 December
15. PAGE COUNT 7

16. SUPPLEMENTARY NOTATION

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Computer Graphics; Tactical Displays

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
This report describes an approach and methodology for providing automated control in a multi-process environment for the display and management of tactical graphical icons in an object-oriented environment.
CONTROL OF ASYNCHRONOUS TACTICAL GRAPHIC DISPLAY
Harlan H. Black
US Army
Communications/Electronics Command
Attn: AMSEL-RD-C3-IR-1(Black)
Fort Monmouth NJ 07703

ABSTRACT
This paper describes an approach and methodology for providing automated control in a multi-process environment for the display and management of tactical graphical icons in an object-oriented environment.

I. THE RESEARCH DOMAIN

A. CORPS MANEUVER CONTROL PLANNING
The US Army Communications-Electronics Command at Fort Monmouth, New Jersey, has been performing exploratory research to apply Artificial Intelligence (AI) technology to the problem of maneuver control planning for a corps commander. The project consists of a group of coordinated research efforts in object-oriented tactical graphics, man-machine interface, terrain reasoning, planning, plan recognition, knowledge acquisition, and representation.

B. THE DEVELOPMENT ENVIRONMENT
An experimental test-bed was constructed which consists of a network of Lisp machines and a large-screen tactical display. This provides a state-of-the-art AI environment in which the capabilities of an object-oriented approach can be explored for tactical decision aids. An icon on the screen represents a Lisp object, and associated with it can be its graphical and reasoning attributes, as well as its functionality, via message passing.

C. THE MAN-MACHINE INTERFACE
To the user, the prototype system is an intelligent plan editor. It monitors his inputs during plan development and provides critiques. It's designed to support his planning, not to do the planning for him.

The prototype's man-machine interface provides the following functionality:
- It brings system planning capabilities to the user.
- It shows the state of the planning system and database to the user.
- It allows the user to provide textual and graphical input.
- It permits the user to asynchronously modify the situation, goals, and resources present in the various knowledge bases.
- It presents a computer mediated planning environment as close as possible to that in which current planning activities are carried out.

Additional interface functionality, not yet implemented, can allow the user to control the display of information and graphics on the tactical displays.

Currently, the prototype uses two display monitors. A monochrome screen displays a command menu and four plan-editing windows for textual input. Each window is of a type that matches a particular planning function. The user may use the command menu to select a particular type of window for display. The second monitor is a color graphical display of the battlefield background, overlaid with symbology.

D. THE PROCESS MODEL
On a machine reasoning level, the maneuver control planning problem was seen to be best expressed in terms of a collection of asynchronous, cooperative processes. The user himself is considered a process. These processes perform different planning tasks and communicate with each other directly through message passing and indirectly through one or more shared knowledge bases. They work in parallel, just like the corps command staff. The display windows on the monochrome display are associated with unique reasoning processes and provide the user interface to them.

For the reasoning subsystem, user control is causal. Reasoning is data driven by modifications to the tactical database. Plans are evaluated as new information arrives or old information changes, and other processes are invoked or spawned to evaluate plan consistency. For the textual and graphic displays, the user shares control with the reasoning processes.
E. PROCESS COOPERATION AND SYMBOLOGY
CONTROL

"Screen clutter is a major concern" [1]. "For tactical applications, the transition to ADP systems depends, in part, on a viable resolution to the clutter problem" [2]. On the textual display, declutter is no issue, as there are always four windows visible. The only concern is that of contention. When it occurs from conflicting requests by reasoning processes, the user is notified and decides. This was not viewed as being a distraction, as it relates to the reasoning, itself, and may provide valuable insight to the user about how the system is processing or viewing the problem at hand. However, for the tactical display, screen content needs to be kept at a minimum. When a process no longer requires a symbol to be seen on the screen, it needs to issue a request for its erasure. This can create a conflict if another process may also desire its display. To ask the user to resolve matters as they come up on an icon by icon basis is distracting. A method of providing display control in an automated manner was required and is the subject of this paper.

II. DISPLAY ACCESS LANGUAGE

A. REQUIREMENTS

To provide the prototype developers a uniform way of performing graphical operations and to resolve the display control issue in an automated manner, a display access language was designed and implemented.

1. GRAPHICAL REQUIREMENTS In conventional tactical command centers, tactical icons and symbology are taped onto one or more plastic overlays that may be mounted or saved. A mechanism for grouping icons for display operations was therefore needed. An icon may be placed on a plastic overlay that is not yet mounted on the map and is therefore not visible to the user. Conditional icon display was therefore also needed. For efficiency, calls for

<table>
<thead>
<tr>
<th>Icon Attributes</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Type and Contents</td>
</tr>
<tr>
<td>Ownning-process</td>
<td>Name-of-process</td>
</tr>
<tr>
<td>Location</td>
<td>Point, Point-list, List-of-point-lists, List-of-list-of-point-lists</td>
</tr>
<tr>
<td>Associated-with-overlays</td>
<td>List-of-names-of-overlays</td>
</tr>
<tr>
<td>Visible-on-maps</td>
<td>((Name-of-map color-list t-or-nil) ...)</td>
</tr>
<tr>
<td>Visibility-reasons</td>
<td>((Name-of-map (Name-of-icon-or-overlay name-of-process)...)...)</td>
</tr>
<tr>
<td>Highlighted-on-maps</td>
<td>((Name-of-map color-list t-or-nil) ...)</td>
</tr>
<tr>
<td>Highlight-reasons</td>
<td>((Name-of-map (Name-of-icon-or-overlay name-of-process)...)...)</td>
</tr>
</tbody>
</table>

Table 1: Selected Icon Attributes, Attribute Types, and Purposes
display operations needed to be minimized. The system had to know not to issue a call for icon display if the icon was already visible. Also, the system should know not to highlight an icon that was not visible on the map. Because tactical commanders often simultaneously refer to several maps of different scales, multiple color displays had to be managed. Finally, a method of highlighting or displaying an icon in a special color was required.

2. DISPLAY MANAGEMENT REQUIREMENTS

A method of controlling the display and erasure of each icon was needed. A decluttering mechanism, that is, a means of providing the user with control and override for an icon's display in an automated environment, was also required.

B. VIRTUAL OVERLAYS

To meet the graphical requirements, a virtual overlay, a Lisp object, was designed with attributes, attribute values, and a defined functionality. Tactical icons were given an associated-with-overlays attribute where the names of all overlays that the icon was ‘on’ could be stored in a list. Every member of this list was unique. Overlay objects were given a similar overlay-components attribute, a list of names of icons. Thus, graphical operations could be performed on an single icon and on a group of icons. The overlay had an on-maps attribute, a list of names of map displays. This signified whether the overlay was ‘mounted’ on a particular map or not. By default, a call for an icon’s display when its associated overlay was not mounted on its map would not be executed, providing a mechanism for conditional display. Associated with the icon was a visible-on-maps attribute, a list of lists. If no process requested the icon’s display, the list was nil. Otherwise, each sub-list consisted of the name of a map display, the name of the color(s) that were used to draw the icon, and the Lisp atom t or nil. The latter was used to designate whether the user requested the icon’s erasure, for declutter. Every map name was unique. Thus, the system could easily determine whether a call to display an icon was unnecessary.

<table>
<thead>
<tr>
<th>Display Access Language</th>
<th>For Tactical Icons</th>
<th>For Tactical Overlays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display:</td>
<td>Show-Icon</td>
<td>Show-Overlay-Icons</td>
</tr>
<tr>
<td></td>
<td>Erase-Icon</td>
<td>Erase-Overlay-Icons</td>
</tr>
<tr>
<td>Highlight:</td>
<td>Highlight-Icon</td>
<td>Highlight-Overlay-Icons</td>
</tr>
<tr>
<td></td>
<td>Dehighlight-Icon</td>
<td>Dehighlight-Overlay-Icons</td>
</tr>
<tr>
<td>User Override:</td>
<td>Declutter-Icon</td>
<td>Declutter-Overlay-Icons</td>
</tr>
<tr>
<td></td>
<td>Restore-Icon</td>
<td>Restore-Overlay-Icons</td>
</tr>
<tr>
<td>Grouping:</td>
<td>Associate-Icon-With-Overlays</td>
<td>Associate-Overlay-With-Icons</td>
</tr>
<tr>
<td></td>
<td>Dissociate-Icon-From-Overlays</td>
<td>Dissociate-Overlay-From-Icons</td>
</tr>
<tr>
<td>Utility:</td>
<td>Move-Icon</td>
<td>Mount-Overlay-Onto-Maps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove-Overlay-From-Maps</td>
</tr>
</tbody>
</table>

Table 3: Display Access Language For Tactical Graphics
sary. Since this was stored as a list, multiple map displays could be easily managed. Indication of user override was built into the attribute's structure. Data on icon highlighting was similarly stored in the icon's highlighted-on-maps attribute.

C. VISIBILITY AND HIGHLIGHT REASONS

To minimize screen content and to provide display control in an automated manner, for every icon that was called for display or highlighting, the reasons associated with this operation were stored in the icon's visibility-reasons and highlight-reasons attributes. The reasons specified the map that the icon is to be visible or highlighted on, the process that requested the operation, and whether the request was for the icon to

<p>| Required Arguments: ICON, unique icon identifier. |</p>
<table>
<thead>
<tr>
<th>Optional Keyword Arguments:</th>
<th>Data Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map-alu</td>
<td>List of two elements. First is the name of a map object. Second is the name of a color object.</td>
<td>Nill</td>
</tr>
<tr>
<td>Overlays</td>
<td>List of names of overlay objects.</td>
<td>Overlays that icon is associated with</td>
</tr>
<tr>
<td>Conditional-show</td>
<td>T or nil</td>
<td>T</td>
</tr>
<tr>
<td>Caller</td>
<td>Name of a process.</td>
<td>Name of process that owns/created icon</td>
</tr>
</tbody>
</table>

Purpose:

Draws icon on map window if not already visible and there is no indication of user override. If there is no entry for map in icon's visible-on-maps attribute, function adds one. Entry is of the form (Name-of-map color-list t). If there is no entry for map in icon's visibility-reasons attribute, function adds one. Entry is of the form (Name-of-map visibility-reason). If there is a map entry but no icon-related visibility reason for the calling process, function adds one. Visibility-reason is of the form (Name-of-icon name-of-process)

Options:

Specification of map window and color:

Uses map-alu argument, if provided. Otherwise, determines map from specified or implied overlays, and determines color from icon, if color attribute is non-nil, or from overlays

Conditional drawing of icon:

If conditional-show argument is t, only draws icon when overlay is mounted on the map.

Specification of visibility reason:

If caller argument is non-nil, new visibility reason to add for the map is of the form (name-of-icon caller).

Example:

Given icon C with null visible-on-maps and visibility-reasons attributes, whose owning process is plan-process and which is associated with overlay O. Given overlay O that has not yet been mounted on map M. Function call (Show-icon C :conditional-show nil) causes C to be drawn on M, sets C's visible-on-maps attribute to (M color-list T), and sets the visibility-reasons attribute to (M (C plan-process)).

Figure 1: Show Icon Syntax and Functionality
be displayed/highlighted or whether it was for
the overlay that the icon is associated with to be
displayed/highlighted. These attributes were
lists of lists. Each sub-list was for a unique map
display that the icon was visible/highlighted on.
The sub-lists were of the form (Map-name (Vis-
reason (Vis-reason) ...)). Each Vis-reason was of
the form (Name Process), where name is the
name of either the icon or an overlay and process
is the name of the process that requested the
operation. Every Vis-reason for a given map was
unique. The structure of the vis-reason enabled
an icon-related graphic operation to be made and
recorded for more than one process, and it
enabled more than one overlay-related graphical
operation by a single process to be made and
recorded. Thus, if a given process had more than
one reason for an icon to be seen or highlighted,
it could make an overlay for that reason, asso-
ciate the overlay and icon with each other, and
have the reasons recorded and utilized in future
graphical operations. Detailed descriptions of
selected icon and overlay attributes are pro-
vided in Tables 1 and 2.

D. CONTROL OF ERASURE

With above data structures, given a request by
process P to erase icon C which is visible on map
M, if the vis-reason (C P) was a member of the
sub-list for M in the icon’s visibility-reasons
attribute, then it was removed. If there were no
more vis-reasons for M, then the sub-list for M
was also removed, the sub-list for M in the
icon’s visible-on-maps was removed, and the
icon was then erased. A similar rule was fol-
lowed for a request to erase an overlay that C
was in. Dehighlighting was handled in the same
manner. Thus, a process could freely call for
symbolic erasure and not conflict with the
display needs of other processes.

E. FUNCTIONS FOR GRAPHICAL OPERATIONS

Table 3 lists the graphical functions that were
specified and implemented for the initial version
of the Display Access Language. For display, and
highlighting, the maps, colors, overlays, condi-
tions, and calling processes can be determined
defined by default from the icon’s attributes or they can
be explicitly specified. However, the calling
process for erasure and dehighlighting was
required to be explicitly specified, to minimize
accidental erasure. For the corresponding over-
lay functions, the calling process name that is
used when the graphical operation is performed
to the overlay’s components is always the owner
of the overlay. Therefore, to provide erasure
control, a minimal amount of cooperation was
expected from all processes (and the developers
who define them) which is that they not request
a graphical operation to be performed on another
processes’ overlay. If a process needs the icon
grouping (overlay) of another process, then it
must make a copy of the overlay and perform the
graphical operations on its own copy. The map
for declutter operations can also be determined
from the icon or it can be specified. Overlays
must be specified for icon grouping functions and
icons must be specified for all overlay grouping
functions except clear-overlay, which uses all
of the icons in the overlay’s overlay-compo-

nents attribute. The move-icon function modi-

dies only the graphics display and the icon’s lo-
cation attribute. For the tactical icons in the
study, information in the location attribute was
sufficient to redraw the icon. The mount and
remove overlay functions modified the dis-
play’s on-maps attribute and called the dis-
play or erase functions for the icons in its
overlay-components attribute.

F. SHOW ICON

The syntax and description of the show-icon
function is provided in Figure 1. The Lisp key-
word syntax permits the user to specify the
optional arguments in any order, in pairs of
keywords and argument values. An example of
its usage, utilizing message passing, is provided
on the bottom of the figure.

III. CONCLUSION

The display language provides a flexible mecha-
nism for tactical graphics control and display in
a multi-process environment. It provides sup-
port for graphical functionality which emulates
graphical operations in a conventional tactical
environment and it provides a means of extend-
ing this functionality in a battlefield automated
system.

IV. ACKNOWLEDGEMENTS

The author wishes to acknowledge the support
and encouragement of Dr. Martin Wolfe and Mr.
Edward Beach, of the US Army’s Communi-
cation/Electronics Command. The author also
wishes to acknowledge the contribution of Dr.
Norman Badler, University of Pennsylvania, who
provided valuable insight to this effort.

V. REFERENCES

Washington DC: AFECA International Press,
1986, Part IV, p. 249.

Framework for the Development of Improved
Tactical Symbology.
Alexandria VA: US Army Research Institute
END
DATE
FILMED
DTIC
4/88