Final Scientific and Technical Report for:
Active Methods for Warfighters

Sponsored by:
Defense Advanced Research Projects Agency
IXO

DARPA Order C043/60

Issued by:
U.S Army Aviation and Missile Command under
Contract No. DAAH01-02-C-R203
Project No. RTW BD02

Notices: Not applicable.

1 August 2003

Barrett Richey
VP Advanced Concepts
Polexis, Inc.
2815 Camino del Rio South
San Diego, CA 92108
Office: 619-542-7226
Fax: 619-542-8675
brichey@polexis.com
www.polexis.com

Abstract

The exploration investigated technologies that could be used to automatically configure desktop applications to help the warfighter complete repetitive and/or process-driven tasks by presenting the warfighter with just the right desktop applications positioned in just the right place and configured in just the right way with just the right content. Managing commonly used COTS applications and constructing manageable custom applications were investigated.

Table of Contents

List of Figures

List of Figures

Not applicable.

1. Introduction

The exploration investigated technologies that could allow the end-user to easily:

- Save the configuration (layout, context, and contents) of the user's running desktop applications.
- Open a saved configuration, reconstituting the layout, context, and contents of the desktop applications defined in the configuration.
- Print the contents of all running desktop applications.
- Close all running desktop applications.

The problem was decomposed into two main areas of investigation:

1. Managing COTS applications.
2. Constructing manageable desktop applications.

2. Methods, Assumptions, and Procedures

2.1. Managing COTS Applications

With regards to managing COTS applications, there were two underlying facts that helped focus the research.

1. Each operating system (e.g., Windows, Unix/Linux, Mac) provides vastly different mechanisms for controlling applications.
2. The most widely used COTS applications in DoD are Outlook®, Internet Explorer®, Word®, PowerPoint®, and Windows Explorer®.

Two technologies were investigated, AutoMate and COM/.NET via the Windows Applications Manager (WAM) prototype.

2.1.1. AutoMate

The exploration investigated the use of AutoMate, a Unisyn Software product (their one and only product), to control the placement, layout, and content of various desktop applications, such as Groove, Word, File Explorer, Internet Explorer, and Visual Studio. The product allows non-programmers to specify a series of simple actions that can be used to launch, position, control (e.g., resize, select buttons, select menu items), and close applications on the Windows platform. The left-hand portion of the following graphic shows the various actions that can be performed.

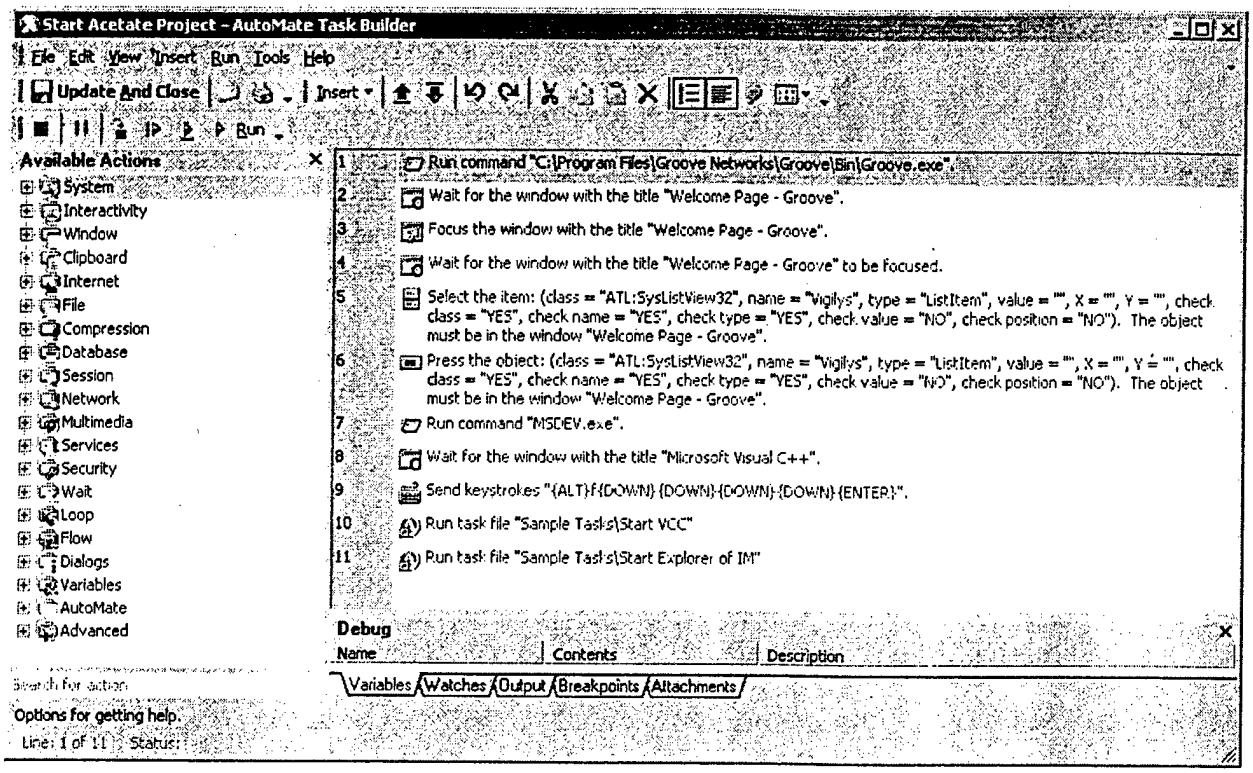

Figure 1: AutoMate Task Builder – Available Options

The right-hand side shows a series of actions for a given task. The Interactivity and Window actions provide many of the actions necessary to control an application's window and dialogs as shown in the figure below.

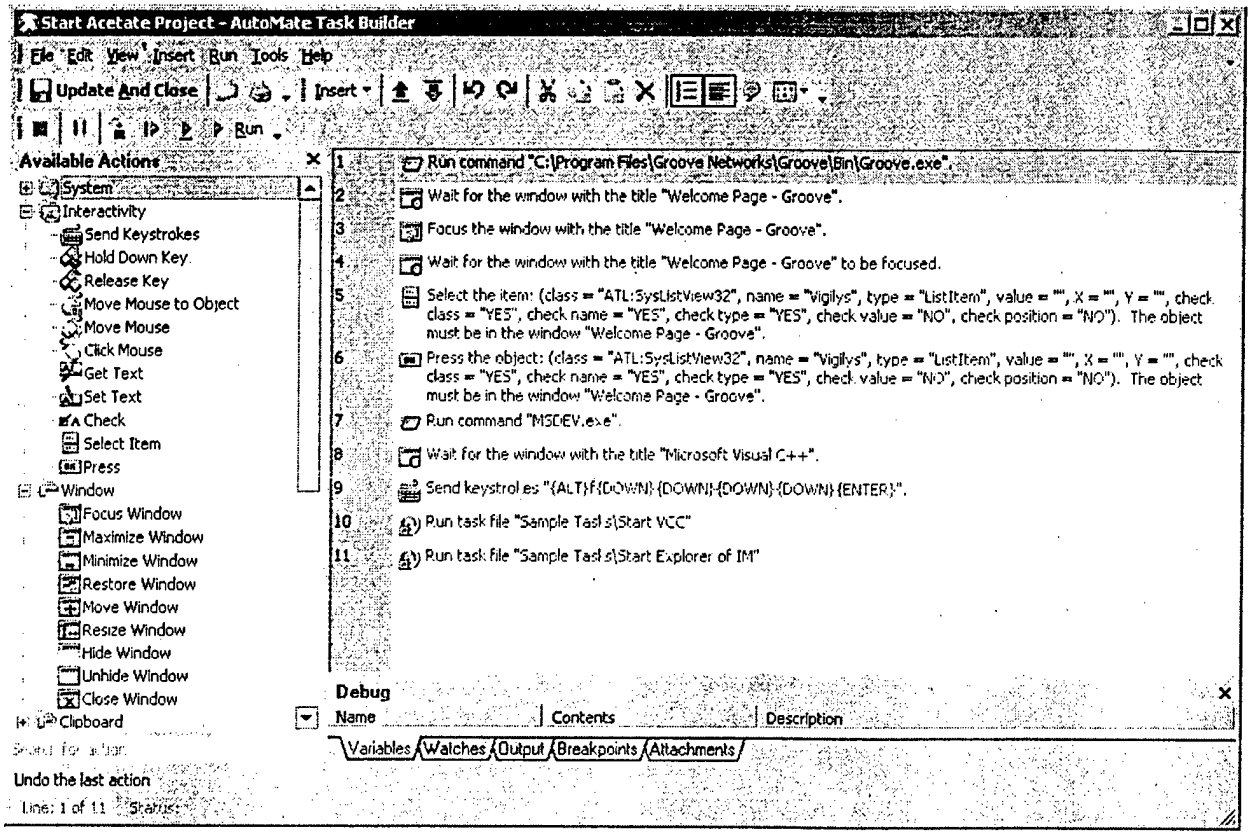

Figure 2: AutoMate Task Builder – Interactivity and Window Options

In order to evaluate the effectiveness of AutoMate, a software engineering task was created that automatically configured the environment for a couple of engineers working on some code. The AutoMate task did the following (and is pictured in the figure below):

- Launched Groove with a specific shared workspace opened and then repositioned and resized the window as shown in the figure below (window on the far left side and mostly obscured by the MS Visual Studio window).
- Launched MS Visual Studio with the skinned.dsw file opened and then repositioned and resized the window as shown in the figure below (window on the right side and taking up most of the screen).
- Launched MS Word with SDD.doc opened and then minimized the window.
- Launched MS File Explorer with a particular directory opened and then minimized the window.

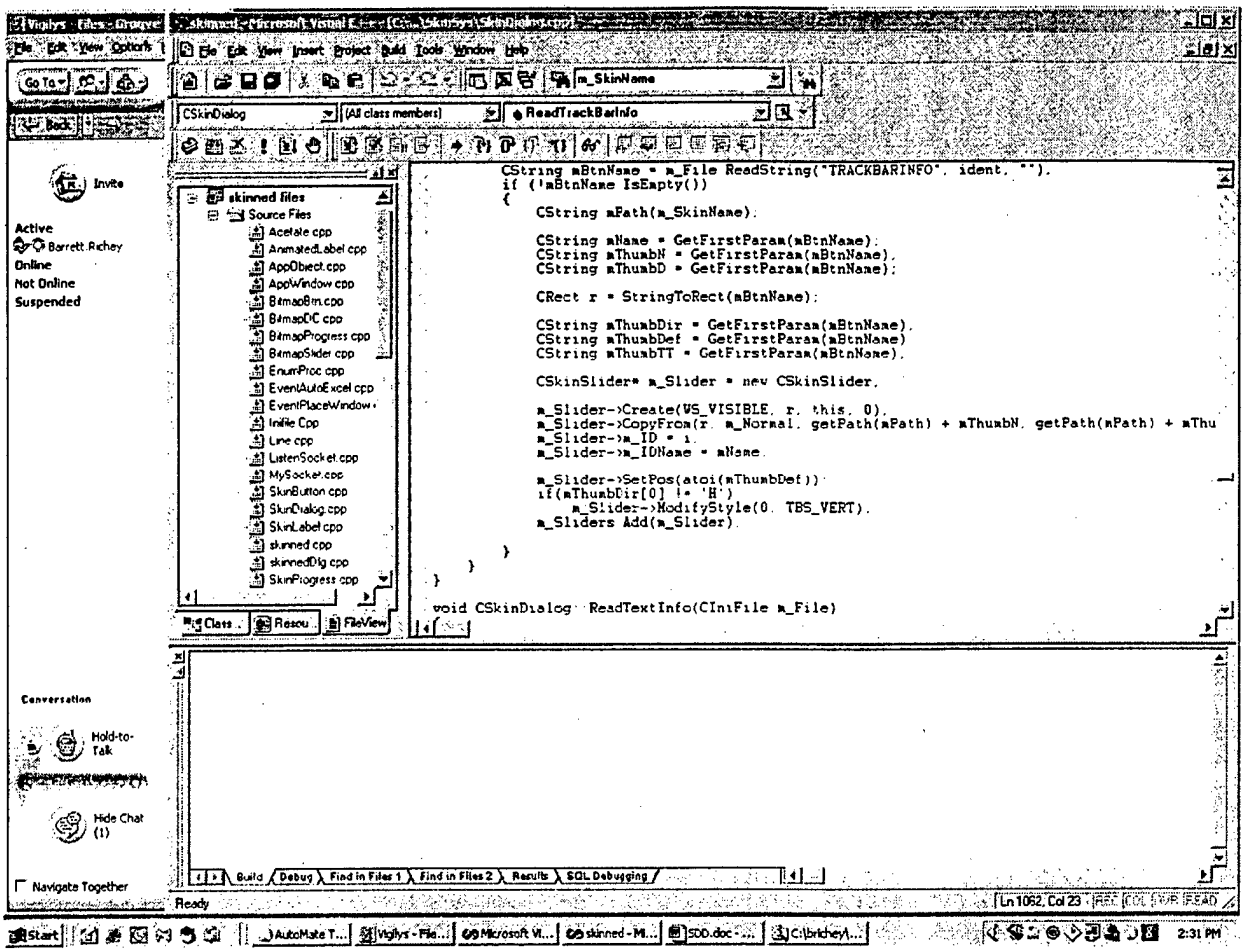

Figure 3: Screen Layout Example

AutoMate also allows the user to set various triggers that can be used to launch each task. The user may select from the following.

- Schedule watcher
- Window watcher
- Performance watcher
- Key watcher
- Event log watcher
- Idle watcher
- System change watcher
- File watcher

2.1.2. Windows Applications Manager

The exploration implemented an easy-to-use COM/.NET-based program dubbed Windows Applications Manager (WAM) that can save the configuration of the currently running versions

of Internet Explorer, Word, PowerPoint, and Windows Explorer as a named configuration so that they can be restored at a later time. The prototype capability is presented below.

The WAM program runs in the Windows system tray. Once the desktop applications are running and configured as desired, the user can select the WAM icon. Selecting the WAM icon presents the user with the following options.

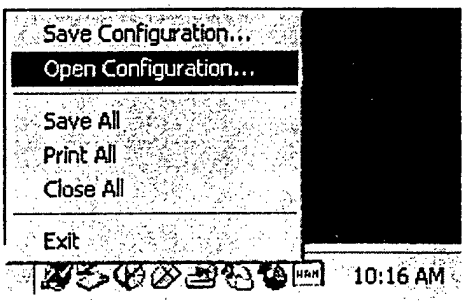

Figure 4: WAM Menu Options

The **Save Configuration...** option presents the user with the Save As dialog, shown below, that allows the user to name the current configuration and then save it.

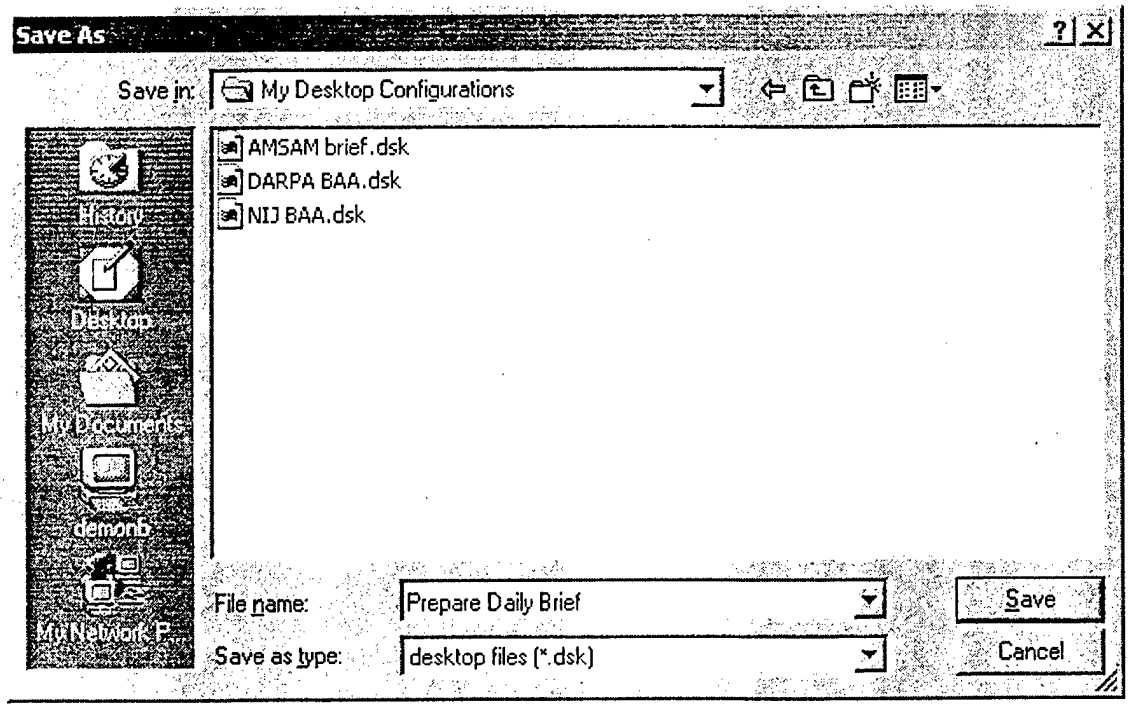

Figure 5: Save As Dialog

The tray icon also allows the user to open a previously saved configuration. The **Open Configuration...** option presents the user with the Open dialog, shown below, that allows the user to open a named configuration, which restores the layout, context, and content of the desktop applications at the time the configuration was saved.

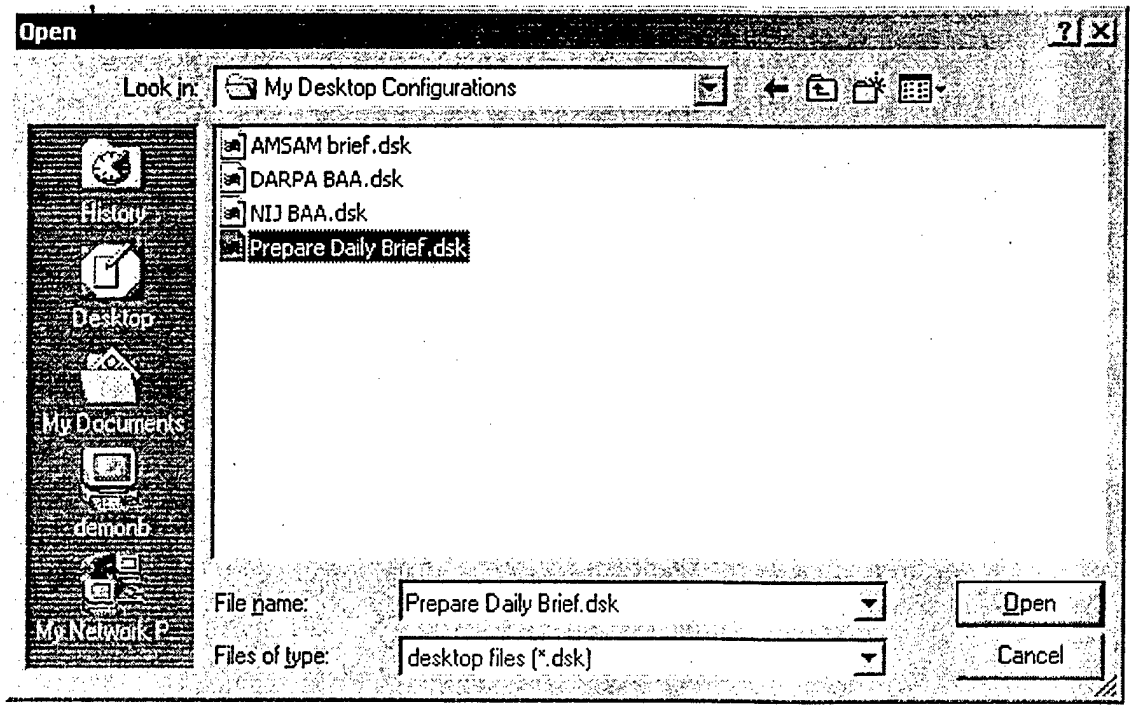

Figure 6: Open Dialog

Although not fully implemented, the Save All option tells the desktop applications to invoke their save function, and the Print All option tells the desktop applications to invoke their print function. The Close All option tells the desktop applications to invoke their Exit function.

WAM's Use of Win32, COM, and .NET

WAM uses a combination of Win32, COM and .NET COM/Interop to perform its functions. It is implemented in .NET because the COM/Interop has a convenient COM wrapper capability. The .NET COM wrapper capability makes working with COM easier than the equivalent in C++. However, C++ and COM provide the greatest flexibility and efficiency and would be the recommended approach when developing a production version of WAM.

Saving An Application's State

Applications that run on the Windows platform may use one of a variety of interfaces to allow other processes to retrieving their current state (e.g., position on the screen, size, file being edited, URL, etc.). WAM uses a variety of heuristics to guess what data an application may have open. These heuristics use low level Win32 calls and COM methods to interrogate applications for things such as window position, size, window title, etc. Window position and size can be fairly well determined using Win32 calls by walking the current list of active windows. The trickier job is finding the documents and URLs that an application may have opened. WAM does this by using a variety of mechanisms. Window Monikers are well known application strings that can be examined for currently opened files. Window titles may also provide a good idea of what

an application is doing. COM can be used on some applications, such as Office, to retrieve the documents that are opened.

Unfortunately, very few applications, including Office and Explorer, tell you exactly what portions of a file are being displayed, and the entire set of user preferences inside the application.

Restoring An Application's State

Restoring an application's state is a simpler process than saving it. Most applications provide a way to start them up with an associated document or URL. Windows Office applications and Window Explorer have a simple memory regarding their screen size and position. The Office application memory is too limited for WAM goals. For instance:

- Office doesn't remember its window size and position based on the document it had opened. It only remembers its last window size, and in the case of multiple windows and multiple documents, if forgets most of the other windows' states.
- Windows Explorer does a good job of remembering window size and position based on the folder a user has opened.
- Internet Explorer does a poor job of remembering size and position based on a URL.

In order for WAM to ensure that all applications are restored to their original size and position for each document, low-level Win32 calls are used. WAM begins by starting all applications with their associated documents. It then traverses the list of windows on the desktop and resizes and repositions the windows to the sizes and positions saved in the named configuration file.

Closing All Applications

The ability to close all applications is accomplished through low-level Win32 calls. WAM traverses the window list and closes all top-level applications associated with these windows.

Saving All Applications Documents

Saving an application's current document is only performed on Office applications. Windows Explorer and Internet Explorer are primarily used to browse existing information sources so save does not make sense for these applications. WAM accomplished this global Office save by traversing the window list and monikers, finding the associated applications with each window and moniker, and then calling the save function on each office application through COM.

Printing All Applications Documents

Printing an application's current document is only performed on Office applications. WAM accomplished this global Office print by traversing the window list and monikers, finding the associated applications with each window and moniker, and then calling the print function on each office application through COM.

2.2. Constructing Manageable Desktop Applications

With regards to constructing manageable desktop applications, there are a couple of drivers that helped focus the research.

1. Java is good for developing cross-platform applications, but
2. Java-based desktop applications are difficult to control without some support constructed with the application

2.2.1. Adaptive Battlespace Awareness Templates

Adaptive Battlespace Awareness (ABA) Templates were developed by Polexis, Inc. for the DARPA/DISA JPO. Their goal was to develop an interface specification for constructing components that can save and restore their layout, context, and content, where each visual component (e.g., map, table, and graph views) implements an interface. A reference implementation written in Java was developed by Polexis, Inc. and integrated into the ABA version of Extensible Information Systems™ (XIS™).

The templating mechanism relies on a central controller to store and retrieve templates from an XML repository. The storage and retrieval of the contents of each view requires the views to use the XIS LeifDataItem as an interface into the raw data accessed by the views.

An early version of ABA Templates was used on this project, but because of time constraints and the relative complexity of the ABA Templates (some of the complexity has been reduced in a subsequent version), the project team failed to get a useful capability constructed.

2.2.2. Smart Widgets

Where ABA Templates worked at the component level, the smart widgets concept was designed to operate at the widget (e.g., frame, panel, list, menu item, button) level. Each smart widget extends a widget used to construct HCIs in the object-oriented language being used (e.g., SmartJFrame extends javax.swing.JFrame). Each smart widget simply writes its state information to a file each time the state of the widget changes (e.g., SmartJFrame writes out the x and y position of the frame (i.e., window) and the height and width of the frame). The smart widgets write out their state information on a per-user basis, so that each user's windows can be restored on restart of the application. Smart widgets developed were:

- SmartJFrame
- SmartJPanel
- SmartJSplitPane
- SmartJList
- SmartJMenubar
- SmartJMenu
- SmartJMenuItem
- SmartJToolBar
- SmartJButton
- SmartFileChooser
- SmartJComboBox
- SmartJSlider

3. Results and Discussion

3.1. Managing COTS Applications

In Windows, Java windows are lightweight windows and not true windows; therefore, the Java windows do not expose the widgets that true windows do. Since AutoMate and WAM only run on Windows, neither can be applied to Java-based applications.

3.1.1. AutoMate

There are drawbacks to using AutoMate for controlling desktop applications.

- Some of the actions are time-delay sensitive, which can and did cause inconsistent results. This was especially true when controlling web pages in the MS Internet Explorer application. Because of time, no specific experiments were performed; however, there was obvious visual evidence of the inconsistent results.
- Developing a task using the AutoMate Task Builder window requires considerable time to develop and test. Spending an hour or more to develop a complex, often-performed task would provide a payoff, but spending an hour or more to save off the configuration of some desktop applications would not.

3.1.2. Windows Applications Manager

There are drawbacks to using WAM for controlling desktop applications.

- WAM only works on Windows platforms that have been upgraded with the .NET runtime package. The WAM program can be re-written in C++ rather than C#. This would allow the WAM program to operate on most versions of Windows.
- Each application to be controlled must be coded into the WAM program. This reduces the burdened for the end-user, but increases the burden on the WAM program to stay up-to-date with the most used COTS desktop applications.

3.2. Constructing Manageable Desktop Applications

3.2.1. Adaptive Battlespace Awareness Templates

The biggest drawback to using ABA Templates for controlling desktop applications is that it is fairly complex. On the plus side, the ABA Templates is just an interface specification, so it can be implemented in any language.

3.2.2. Smart Widgets

Smart widgets had no obvious drawbacks. Using them on various windows was easy and provided instant benefit to the user. It is also easy to insert smart widgets into existing code. Once the user organized his/her windows (location and size) and split panes, selected items in lists, etc., the user didn't have to repeat the same activities on subsequent runs. The fact that the state information was stored in files made it easy to copy them onto a floppy diskette for sharing with other users. Additionally, although not implemented, storing the state information in files should make it easy to store and managed named configurations simply by copying the default

configuration to named directories. Upon user selection of a named configuration, the path to the appropriate named directory can be established.

4. Conclusions

Based on the ease-of-use requirements, the WAM approach is better than the AutoMate approach. Because the ABA Templates interfaces are complex, smart widgets is the better approach. A combination of both smart widgets and WAM provides a fairly complete capability. If a smart widget-based application provided a mechanism (e.g., socket-based message) to allow an external program such as WAM to tell the application to save its configuration and assign it a name as mentioned in section 3.2.2 and a mechanism to allow the external program to launch the application with a specified named configuration, then the smart application, especially Java-based applications, would play nicely with the WAM program.

5. References

None.

6. Symbols, Abbreviations, and Acronyms

ABA – Adaptive Battlespace Awareness
COTS – Commercial Off-The-Shelf
DARPA – Defense Applied Research Projects Agency
DISA – Defense Information Systems Agency
DoD – Department of Defense
HCI – Human-Computer Interface
JPO – Joint Program Office
WAM – Windows Application Manager
XIS – Extensible Information Systems
XML – Extensible Markup Language