ManyVis: Multiple Applications in an Integrated Visualization Environment

Atul Rungta, Brian Summa, Dogan Demir, Peer-Timo Bremer, *Member, IEEE*, and Valerio Pascucci, *Member, IEEE*

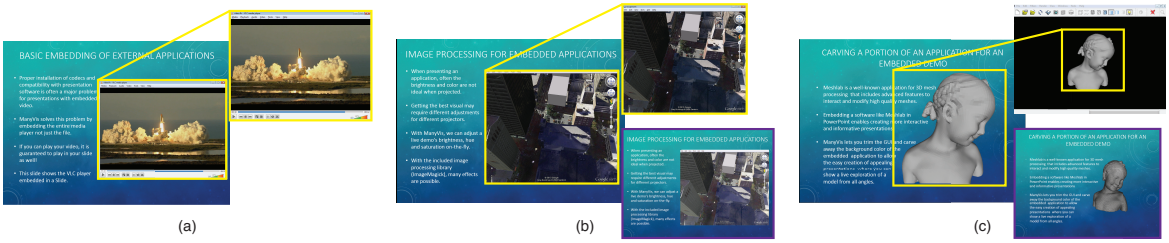

Fig. 1. ManyVis allows the custom integration of disparate applications into a single, seamless ManyApp. This figure illustrates some of the functionality of our PowerPoint Presentation ManyApp. This integrated application allows a user to embed and manipulate external applications into their PowerPoint presentation. (a) Video codecs are often a problem when embedding video. With ManyVis a presenter can just embed the video player (VLC) itself. (b) Embedding a demo application is also simple. (b purple inset) Often projector and room conditions may cause a demo to be presented poorly. With ManyVis a presenter can adjust the color, brightness and contrast in real-time. (c) More sophisticated manipulation is possible with ManyVis. In this example a presenter creates a fully integrated MeshLab [2] demo by cropping the unnecessary GUI and applying an alpha transparency to the embedded application (c purple inset). The application maintains full interactivity. (d) Often demo programs contain more GUI elements than necessary for a presentation (d purple inset). ManyVis can customize the demo's layout for a better presentation.

Abstract—As the visualization field matures, an increasing number of general toolkits are developed to cover a broad range of applications. However, no general tool can incorporate the latest capabilities for all possible applications, nor can the user interfaces and workflows be easily adjusted to accommodate all user communities. As a result, users will often chose either substandard solutions presented in familiar, customized tools or assemble a patchwork of individual applications glued through ad-hoc scripts and extensive, manual intervention. Instead, we need the ability to easily and rapidly assemble the best-in-task tools into custom interfaces and workflows to optimally serve any given application community. Unfortunately, creating such meta-applications at the API or SDK level is difficult, time consuming, and often infeasible due to the sheer variety of data models, design philosophies, limits in functionality, and the use of closed commercial systems. In this paper, we present the ManyVis framework which enables custom solutions to be built both rapidly and simply by allowing coordination and communication across existing unrelated applications. ManyVis allows users to combine software tools with complementary characteristics into one virtual application driven by a single, custom-designed interface.

Index Terms—Visualization environments, integrated applications, macros, linked views

1 Introduction

Visualization is an integral part of advanced research in science and engineering, therefore various excellent visualization tools exist [20, 34, 19] each with its own strengths. These tools are very good at certain tasks but are not very well- suited for others; for example, despite providing a very general library for visualization, The Visualization Toolkit (VTK) cannot contain all the numerical capabilities of software like MATLAB and Mathematica. Many of these tools are complementary (for example, VTK, MATLAB, and PowerPoint) but it is very difficult, if not impossible, to make them work together in a single, integrated environment. Researchers are savvy in understanding the positives and negatives of these tools and will often manually integrate several into their workflows.

A common integration strategy is to generate data from one tool, convert it to a common format, and pass it as an input to another. Synchronizing data between tools can be error prone, tedious, and time consuming. A common solution to this problem is to extend tools via provided application programming interfaces (APIs). These APIs, although very powerful, are often too limited in functionality to provide an ideal solution. More importantly, each API is specific to a certain application (even for an application version) and therefore cannot provide generality. For example, a developer may allow communication between two programs via their APIs, but integrating a third application would require a significantly new code base.

Visualization researchers on the other hand are faced with the corresponding challenge of deploying their solutions. Even using good software design principles, integrating capabilities into a custom system requires significant time and effort. This can be frustrating for both the scientific collaborators anticipating short term solutions as well as the visualization researchers for whom one-time implementation efforts are of low priority. Alternatively, visualization researchers can make their solution available as stand alone tools or libraries shifting the integration effort to potential users. However, in many cases, application scientists have neither the resources nor the expertise nec-

- *Atul Rungta is with the SCI Institute, University of Utah. E-mail: arungta@sci.utah.edu*
- *Brian Summa is with the SCI Institute, University of Utah. E-mail: bsumma@sci.utah.edu.*
- *Dogan Demir is with the SCI Institute, University of Utah. E-mail: ddemir@sci.utah.edu.*
- *Valerio Pascucci is with the SCI Institute, University of Utah. E-mail: pascucci@sci.utah.edu.*
- *Peer-Timo Bremer is with Lawrence Livermore National Laboratory. E-mail: bremer5@llnl.gov.*

Manuscript received 31 March 2013; accepted 1 August 2013; posted online 13 October 2013; mailed on 4 October 2013.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

1077-2626/13/$31.00 © 2013 IEEE Published by the IEEE Computer Society

essary to successfully integrate disparate tools into their current systems. Therefore it is common for visualization researchers to provide small, specific tools to collaborators thereby integrating the new techniques into the scientists' pipeline described above.

In this paper, we propose an alternative method of deployment which is fast, provides immediate benefits to scientific collaborators, and allows visualization researchers to achieve a wide dissemination of their algorithms through single, stand-alone implementations. Our framework, called ManyVis, drastically reduces the time lost by users in dealing with multiple applications and, for the first time, provides an integrated application in which separate tools communicate and coordinate. ManyVis sits between the user input and the windowing system, recording, augmenting, and automating user interactions and display to create a single, seamless user experience. In the design of this framework, our guiding principle: if a user can accomplish a task, ManyVis should support such a task seamlessly. We provide several examples of the ManyVis accomplishing the type of tasks common in the scientific community using a combination of several open source, commercial, and custom applications. Using ManyVis, the development of these examples, from conception to a workable solution, required only a few hours as opposed to the days or months of effort that traditional methods, if at all feasible, would have required.

In particular, our contributions are:

- the ManyVis framework that intercepts, coordinates, and processes low-level user interactions and allows combining them into higher level, task-oriented operations,
- the ManyMacro system built using ManyVis to allow the easy creation and execution of custom scripts and applications which leverage the ManyVis core primitives,
- the ManyWorkflow system, which schedules and coordinates complex workflows with multiple applications and exposes the user to a unified interface of a seamless, interactive environment called a ManyApp,
- a demonstration of our new approach with several ManyApps, including an exemplary case of building Powerpoint presentations that integrate live demonstrations of external software tools.

In Section 2, we will introduce the ManyVis framework and provide details on ManyMacros and ManyWorkflows. In Section 3, we will describe how the framework, ManyMacros, and ManyWorkflows can be used to design new integrated applications. Finally, in Section 4 we will discuss our approach and provide a thorough performance evaluation and limitations.

1.1 Related Work

In a system such as ManyVis the two primary challenges are to enable the sharing of information across disparate tools as well the automation of common user interactions. This section discusses some existing approaches aimed at addressing these problems.

Inter-application interaction and application extension. Inter-application (or inter-process) communication is a fundamental component of all modern computer operating systems. Communication between applications can vary from simple file passing or shared memory to more complex message passing via pipes or sockets. Typically, this communication operates at the lower system level, therefore specifications for communication must be decided at the time of development. Consequently, communication standardization often only exists on a per application basis. Frequently, it is problematic (if not impossible) to allow two programs to communicate if they were not designed to do so from the outset. Of particular note are groups such as The Common Component Architecture (CCA) Forum [1] or the commercial Common Object Request Broker Architecture (CORBA) [28] which work to standardize communication across separate applications. Despite these efforts, at this time such standardization in communication is not widely adopted. However, two notable exceptions to this rule exist: a) the operating system's clipboard [25, 9] (pasteboard, etc.), which is often standardized by the operating system and ubiquitous in modern applications and b) file passing between programs, which frequently supports open or well documented formats. In this paper, we will show how our system, ManyVis, will take advantage of these two exceptions to allow for communication between programs which were never designed to do so.

In addition to the visualization tools outlined in the previous section, commercial software companies such as Adobe [3] and Microsoft [26] support extension and inter-application communication amongst some of their products with development kits that employ a proprietary application programming interface (API). Closed source APIs are often very limited in scope by only allowing extensions in areas the company's developers want or predict will be useful. In addition, these systems are typically in direct competition and therefore collaboration between companies to allow communication between their software is nonexistent. In contrast, open source systems offer a potential limitless scope for extension. However, for large open-source projects modifying an implementation often requires a significant investment in effort to learn the intricacies of the system. Therefore development teams for these projects will, again, supply a limited API [15, 29] to developers.

Automated and scripted interaction. The support of automating common user interactions with a graphical user interface (GUI), especially when the interactions are repetitive, is a desirable and useful feature found in a wide-range of applications. Modern operating systems provide resources to aid developers in adding this support. Examples include Apple's AppleScript or Automator and Microsoft's Visual Basic and JScript. Scripting support on the program level includes examples such as Maya Embedded Language (MEL) or Python scripting support in Autodesk software and Python scripting support in The Blender Foundation's Blender. Scripting user interactions need not involve traditional programming, and can be automatically recorded by the user through the visual interface. Adobe's Actions are one such example where the user can record their actions to re-execute common interactions.

Automating user interaction is a topic typically studied in the human/computer interaction (HCI) community. Of particular interest, is the work in *Programming by Demonstration or Example* (PBD) [12]. As a research area, PBD hopes to replace the programming of new system behaviors with a user's example input or scripted user interactions. This allows for a rudimentary programming model that requires no expertise from the user and has the ability to allow communication and coordination of separate programs without the need to use specific system APIs. PBD also has applications in the design of intelligent help systems, where an expert's interactions are recorded to be replayed in order to help a novice user. Examples include applications for guided tutorials [10], technical support [22], help across different applications and dynamic environments [33], printed tutorials for image editing applications [16], content-adaptive image manipulation macros [11], or full documentation of image content creation [17]. Work has even been done to make user macros more stable by introducing debugging schemes [11].

PBD can also be used in the design of *interface agents*, software to aid users in accomplishing tasks that are too complex or repetitive to accomplish alone. Past work has shown that these agents, when combined with GUI interactions, can be used to interface with closed-source, commercial applications or handle the coordination between multiple programs [23]. This work has also shown that for an agent to be general, it must have an internal model of the program it is manipulating on the user's behalf. By coupling user interaction with image processing such as pattern recognition or segmentation, agents can build such a model with machine learning [38, 7]. These internal application models have obvious implications in cognitive modeling and have the potential to give new cognitive modeling techniques access to a wide range of software [6]. Given our system's target use, the overhead due to model building would be undesirable and as we show in the following sections, also largely unnecessary achieve powerful applications.

Of particular interest is the work in PBD to automatically create a sequence of user interactions which is used to perform a specific or number of tasks. This sequence is typically called a *macro*. PBD is frequently used to record a macro or create a macro script based on a user's actions. PBD macro generation has proven to be popular in a

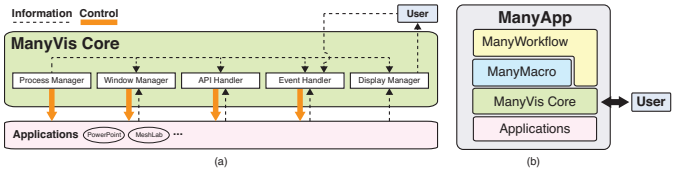

Fig. 2. (a) The primitive operations provided by ManyVis along with their control and information flows. ManyMacros and ManyWorkflows utilize these primitives to build custom unified ManyApps. (b) A block system diagram of a ManyApp. The ManyVis core primitives interface with the user and applications directly. ManyMacros leverage these primitives into more sophisticated operations. ManyWorkflows handle the coordination and synchronization of ManyMacros, ManyMacro scripts, and possibly ManyVis core elements.

variety of application contexts [12, 24]. As mentioned earlier, similar techniques have been adopted in software systems such as Adobe's Photoshop [4]. Macros created via PBD have been used for systems to work with 2D graphics [21], desktop actions [27], business email tasks [36], data analysis tasks [14], and web browsing [35]. Recent research [13] has applied PBD to aid visual programming for GUI testing as well. Again like agents, most of these examples typically require a system to have some high level knowledge of the application which it is manipulating. The VisMap [39] and TRIGGERS [5, 31, 32] systems are particularly relevant since they have been designed to work generally with any program. Even though they have only been shown to work with simple examples, these systems give a sense of how such interactions can be used as a powerful tool. ManyVis uses these lessons learned to provide the first fully automated GUI interaction system for visualization.

2 MANYVIS

ManyVis is an abstract, low-level framework for managing application execution, application window management, intercepting and processing user inputs, accessing application API elements, and augmenting an application's display. Fig. 2 illustrates the structure of the framework. Apart from the inputs, ManyVis coordinates process management, windowing, and possible communication between applications. This enables ManyVis to function as a quasi-virtualization environment giving enhanced/augmented (low-level) control over applications, allowing them to work together. Using ManyVis, a user can, for example, coordinate time-varying data across multiple applications using a single time line, edit images with Photoshop which are too big to be imported directly, and/or create presentations which can have live applications embedded. Our prototype is designed for the Windows 7 operating system and relies on Win32 API calls to intercept user inputs, although the framework itself is general and can be applied to any underlying OS. The main components of ManyVis are:

Process Manager. At startup, a ManyVis session launches a set of applications to manage. Applications are initialized by creating the corresponding process via system API calls and retrieving the handles to windows the application creates. Application windows are addressed by the (P, T, C) triple where P is the ID of the process that created the window. T and C are the title and the class name of the window respectively. This allows ManyVis to identify windows uniquely and associate them to a process. Although the title and class of a window are normally enough to identify a window uniquely, this approach doesn't work for cases where there are multiple instances of the same application. The triple ensures that the title and class of a window along with its process ID uniquely identify a window.

Window Manager. To allow the display and coordination of multiple applications each with the possibility of having multiple windows, ManyVis ties into the main operating system's window manager. This coupling allows ManyVis to move, resize, or change the current window focus. Additionally, developers can create custom user interface elements (e.g., buttons, sliders) via the Window Manager for later integration into their ManyApp.

Event Handler. ManyVis achieves much of its functionality acting as an intermediary for user input. In its simplest form, ManyVis can determine which program the user wishes to interact with and passes that information along to the proper program (provided by the Process Manager). As detailed in Section 2.1, when recorded, edited and saved into a ManyMacro this handler allows for powerful functionality. Series of events can be created to perform one or multiple operations on one or multiple applications. Events can be passed as either using messages or inputs (according to the Win32 API). Using the former, the operating system passes events to windows using messages, while sending direct inputs to the foreground window. A major advantage of using messages is that it does not require the mouse to be physically present at a particular position. Inputs, on the other hand, require the mouse cursor to be physically present at a particular position to function. This may cause undesired results if there is accidental mouse movement during event playback (ManyMacros). That being said, messages suffer from a major drawback: Posting a message to a particular window may not always work due to the window composition. If a window is composed of several smaller windows, sending a message to the parent window does not guarantee the events are passed to the proper child window. Inputs on the other hand, despite their drawbacks, are guaranteed to send the right input to the right window. Although, ManyVis has provisions for both (messages and inputs), the prototype system uses inputs due to this guarantee. Our current prototype blocks user movements during playback to ensure proper ManyMacro executions.

API Handler. User input events are a powerful tool used by ManyVis to achieve much functionality. Although, they may not provide all (or event the best) functionalities necessary to achieve a desired ManyVis application. Therefore, ManyVis includes a module to access applications' APIs using its native scripting language (for example: VBScript, MEL etc). The use of APIs helps create *ManyVis Objects* which are application specific. For example, Microsoft PowerPoint exposes a rich set of functions giving access to many of the objects that comprise a presentation. ManyVis uses this to create custom ManyVis objects allowing access to the underlying application. A PowerPoint shape is a ManyVis Object (amongst others) and can embed any application easily in a PowerPoint slide. It allows the user to start/end presentations, change the size of boxes, etc., all at runtime.

Display Manager. To give the end-user the impression of a common application, ManyVis also resides just above the application level between each application and the display. Since ManyVis has access to all the windows of the applications, it is possible to alter the window contents. Since an application window from creation to display on the screen is simply an image, ManyVis allows for the integration of any image processing technique as well. For our prototype application, we show how to integrate the ImageMagick library to process application windows before they display on the screen. This enables a user to apply a wide gamut of filters and effects on windows' contents while maintaining interactivity with applications. For example, this allows the ability for a user to change color or contrast, crop, splice, or apply transparency to a window. This component is optional and can be disabled for an application if no processing is necessary.

2.1 ManyMacro

The components detailed above are the primitive functionalities of the ManyVis framework. One or several of these primitives can be developed into a sophisticated ManyMacro element and a ManyMacro script is a sequential collection of these new elements. ManyMacro elements can be thought as a custom, mini-application which uses the ManyVis Core primitives as an API-like interface. A ManyMacro script is recorded as a collection of elements saved as XML. Each element in the script stores its needed state and behavior allowing each to be independent of the elements preceding or succeeding it.

In its simplest form, a ManyMacro resembles a sophisticated macro system by recording and playing back mouse and keyboard events via a direct interface with the Event Handler. Using ManyVis' Process Manager, a user input recorder associates an interaction with the proper application window. Since the process ID is different every time an application is started, ManyMacro stores "normalized"

(scoped) process IDs which are assigned in the order the process was started. This makes the interactions scoped to a particular process and allows multiple instances of the same application to be handled correctly. The ManyMacro captures the size of a window and coordinates of the mouse pointer relative to the window to make the playback independent of the size and position of the window. The ManyMacro also stores the time elapsed between each event to create a timeline and to make the playback mimic the original user interactions accurately. Playback is simply the replaying of the recorded events sequentially with the proper timings. Although, ManyMacros offer far more expressive operations due to their close coupling with the ManyVis framework. For example, processes can be launched or killed via the Process Manager. Windows can be moved, resized and/or brought into focus via the Window Manager. Program specific calls via API Handler operations or custom elements developed using these operations can be made.

ManyMacros can also use ManyVis' Display Handler to allow presentation of a final custom application in a reduced, purpose-oriented interface by letting the ManyVis developer control the content of application windows. When working with multiple applications, it is very often the case that the screen space is utilized more by the interface elements than the area of interaction. For example, an image editor application interface may consist of a number of toolbars and buttons while the user needs to actively use only one tool. This is acceptable for a workflow with a few applications; however, as the complexity of the task increases, this causes many unnecessary interface elements to be on screen at once, causing diversion of focus. For such a scenario, the adjustment of interface is needed. Adobe Photoshop [4] at the time of this writing supports creation of multiple custom workspace layouts, yet this is far from being a feature widely implemented in the rest of available commercial applications. Even for applications where the workspace can be customized, the use of the same tool for different tasks often requires different workspace arrangements. The workflow interface can be built by drawing the contents on the screen in a way defined by the user through actions such as cropping, resizing the dynamic content or even processing pixel data. Such a model not only allows the user to eliminate unnecessary interface elements, but to also append further actions to the ones that exist. The workflow manager provides this functionality via ManyVis' Display Handler. Multiple applications can be presented to a user as a single, GUI-minimal view. Furthermore, ManyVis allows full integration of image processing libraries (ImageMagick in our prototype system), which a workflow can use to provide a wide range of effects and filters to apply to the application window images.

2.2 ManyWorkflow

ManyMacros provide a developer full scripting and programming access to the ManyVis core infrastructure. However a ManyMacro script, by itself, is still a single collection of serial operations. ManyWorkflow bridges this gap by providing developers the ability to schedule and coordinate multiple ManyMacro scripts. By doing this, a developer can provide powerful new ManyVis applications (see Fig. 2) which combine several disparate applications into a single seamless environment.

The ManyWorkflow allows a developer to coordinate and augment ManyMacros or ManyMacro scripts. One or several macros or macro scripts can be bundled into workflow *actions*. Actions can be executed on a schedule, via user input captured by the Event Handler, via custom buttons provided by the Window Manager, by ManyMacros, or even by other actions. In this way, a ManyWorkflow can allow much flexibility and allow for the coordination and synchronization of applications easily. For example, if time-dependent data is being viewed or analyzed in multiple applications, a time step change in one window can trigger all applications to move time via ManyMacros or ManyMacro scripts. This execution is unseen by the user and gives the impression of a seamless new application.

Merging and coordinating display and interactions already allows a powerful system as shown in the previous work in PBD. What makes the ManyWorkflow far more powerful is the way it allows the control of the information flow between applications, allowing communication between multiple applications whose interfaces are not designed to interact in automated way. The ManyWorkflow enables communication between programs by leveraging methods that a typical user would follow to transfer content between different applications. With this manager, programs may communicate by inserting data to and reading from the clipboard via actions. They may also be set by a ManyWorkflow action to read and write to the same file(s) on the underlying system. If no common file format is available or previous methods are not applicable, each program can also communicate with a third party process or via application objects from the Application Handler. Although flexible, actions are limited to what can be accomplished through a program's interface or via ManyVis objects. For instance, the microscopy example detailed in Section 3.3 would not be possible if the out-of-core viewer did not allow the insertion of new buffer values via the clipboard or the file menu, or did not provide a way to determine the viewer's viewport location and resolution. In other words, we exploit and coordinate the existing functionalities of the tools but do not necessarily create new ones.

3 MANYAPPS

In this section, we demonstrate how ManyWorkflows and ManyMacros can be used to create custom ManyVis applications (ManyApps). First, we will introduce a simple ManyApp useful in debugging ManyMacros. Next, we will detail our exemplar Presentation ManyApp. Finally, we will describe several additional ManyApps for scientific visualization.

3.1 Debugger ManyApp

Macros have a tendency towards instability or inefficiency as noted in previous work [11] on event driven macros. Therefore a debugger is necessary in order to enable the achievement of a desired behavior more easily. ManyVis provides an initial bootstrap application which provides a step-by-step ManyWorkflow debugger. Note that this debugger is an application built using its own ManyWorkflow. In the debugger, a user can step through all ManyMacros sequentially or skip to a particular macro element. If the user jumps forward, all intermediate ManyMacros are executed. At each breakpoint, the debugger prints the related ManyVis state information to the console, exposing the state of the primitives. The user is also presented visual feedback of the actual ManyWorkflow as the ManyMacros are executed while the current ManyMacro is presented to user either directly or as a highlighted element in corresponding ManyMacro script. The debugger lets the user "step back" by rolling back certain ManyMacro elements. While useful, this backwards step relies on the fact that the element did not change the application state. For instance, user interactions which load a new file would be unsupported. Even with this limitation, this backward movement in the debugger is still useful enough to warrant its inclusion. For example, a drag or scroll operation can often be undone. With this debugger ManyWorkflows and ManyMacros can correct undesired behavior.

3.2 Presentation ManyApp

Commonly, visualization researchers will give presentations on new techniques or algorithms. These presentations will often include a live demo of a prototype application. Switching between the presentation software and the demo is a cumbersome, error-prone, and stressful process. Additionally, due to room or projector conditions the brightness, contrast or color of the demo may not present the application in the best light.

Finally, successful embedding of a video into a presentation can be highly variable due to codecs and format of the video file. Often, researchers will switch to a robust video player, like VLC [30], instead of being at the mercy of the presentation software's embedding. In this section, we detail how to design a Presentation ManyApp via a custom ManyWorkflow. Using the building blocks detailed previously, we show how a developer can create a highly customized, interactive presentation that alleviates all of these common problems.

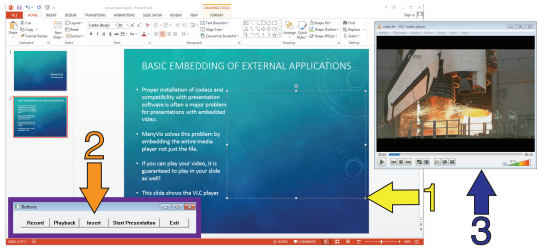

Fig. 3. **Presentation User Interface.** The Presentation ManyApp allows users to easily insert virtually any application without the need for programming. In this figure, the steps are outlined for the insertion of a video player into a presentation. (1) A user creates or selects a PowerPoint shape. (2) They choose to insert from the supplied GUI (purple). (3) To complete insertion they select the proper application window. PowerPoint and the embedded application remain interactive.

At a high level, the ManyWorkflow consists of a single presentation application, Microsoft PowerPoint, with one or several applications that the user wishes to embed. The presentation application is customized by the creation of specialized ManyMacros that use Application Handler API objects tied to PowerPoint VBscript. The presentation software can be considered a "host" application, which drives the actions and display of the "embedded" applications.

Creation. The creation of a presentation inside the new ManyApp is its own custom workflow. As an initial process, a user manually creates a ManyMacro script to denote which applications will be used in the new application (with PowerPoint being the host application). ManyMacros can be called for each embedded application, or new ManyMacros can be recorded at this time, to bring each program to its desired state. To embed the applications, the user creates one or multiple powerpoint shapes on the desired slide, selects the right application (by putting focus on them), and via the provided GUI indicates to ManyVis to insert the application. ManyVis, in turn uses a custom ManyMacro that utilizes objects in the Application Handler to find the coordinates of the selected shape and the window. The ManyMacro also records the shape ID and slide number in order to identify the shape uniquely in the presentation. Insertions are saved if the user is satisfied with the result as a custom ManyMacro script that is called if the presentation is relaunched. To provide a seamless user experience, the screenshot of the current state of the application is inserted into the slide. By doing this, a placeholder shape for the program appears in the slide and can be edited via PowerPoint. If the application needs to be adjusted, it can be un-embedded, modified, and re-inserted.

Presentation. During a presentation, a custom API ManyMacro embeds the application by overlaying the selected application window over its corresponding shape, which for our demo applications is a rectangle. In presentation mode, shape coordinates change, therefore as a first step the ManyMacro queries the shape IDs of the embedded applications for their new locations. Shape IDs are indexed locally to each slide, therefore the ManyMacro polls PowerPoint on a timer to keep track of the current slide in order to resolve each shape to the proper application. During the presentation, the placeholder screenshot for the application is swapped for the real application when interaction is requested. The application is resized or scaled (e.g. lanczos downsampling) via a Display Handler ManyMacro to fit seamlessly into the new shape size. Any mouse interactions that occur within the shape, or specified keyboard interactions, can be passed to the embedded application.

At this point, the application description assumes a user would like to embed the entire application window into the PowerPoint shape like the video example in Fig. 1 (a). Although, using ManyMacros that leverage the ManyVis Display Handler, there are many more options possible. Custom GUIs can be created in the slide by cropping, moving, and resizing the original GUI elements before they are embedded into the shape. Multiple GUI elements from the same window can be embedded into different shapes using the same process. The embedded application's color, brightness, contrast, and saturation can be adjusted by enabling and disabling ManyMacro filters as in Fig. 1 (b). These

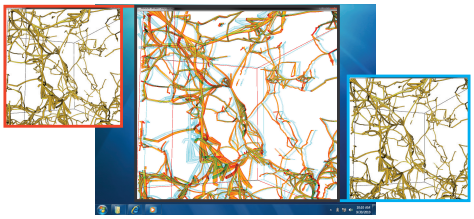

Fig. 4. **Ad-hoc Anaglyph Visualization.** Two instances of a 3D visualization tool are combined for red and cyan 3D anaglyph.

filters and their parameters can be tied to user interactions and therefore can be adjusted in real-time during the presentation. Moreover, more complex filters, like the transparency color replacement in Fig. 1 (c) or image flood-filling, can be applied to the embedded application to provide a fully interactive and integrated PowerPoint application demo.

User Experience. In the previous paragraphs, we have detailed how a developer can provide the functionality necessary for a PowerPoint Presentation ManyApp. After this initial creation, a user can embed virtually any application to create a variety of different presentations. Before launch, a user specifies a list of programs which they would like to embed via an XML file. At launch, ManyVis starts each application along with PowerPoint. The ManyApp provides a simple user interface to aid the presentation creation, see Fig. 3. A typical user workflow would be the following: First, the user can select an application then record or replay a macro to bring the program to a desired state via the ManyVis GUI. Next, the user can draw a PowerPoint rectangle to denote where to embed an application. With the rectangle selected, a user can embed a selected window via the ManyVis GUI. The application remains interactive after embedding for further manipulation. The user can also switch between PowerPoint's editing and presentation modes with the application embedded. After they are satisfied, they can save the entire ManyApp in its current state for later relaunch. Image processing components such as rescaling or alpha transparency can be enabled by editing the XML script. Both Fig. 1 and the companion video demonstrate a variety of applications seamlessly integrated into PowerPoint using the ManyVis system. As previously mentioned, each example provides a solution to a real problem faced by visualization researchers when presenting their work. For instance, the MeshLab example in Fig. 1 (c) shows how ManyApps save time and effort along with providing dynamic context. In this example, MeshLab model can be aligned with the text in fluid manner. To achieve this effect outside of our system would require: saving a screenshot, loading the screenshot into Photoshop for the alpha transparency, and placing the final image into the presentation. This takes many iterations and on the order of minutes to complete. With the ManyApp, importing and aligning the model is trivial and instantaneous. The application is also interactive during the presentation for live demos.

3.3 Additional ManyApps

In the following section we detail additional examples of using ManyVis and show how simple extensions can lead to powerful visualization tools.

Ad-hoc Anaglyph Visualization. In this example, we show how one can add new visualization functionality to a tool. Specifically, we add ad-hoc stereo anaglyph to a 3D application which does not initially support the functionality. Fig. 4 provides an example anaglyph tool. To provide a "toe-in" type anaglyph, we need to provide two 3D views with a slight rotational difference to achieve the desired effect. The two (left and right stereo) views are presented overlaid to the user filtered by the colors that correspond to the type of 3D glasses used. In our particular example, these colors are red and cyan. A ManyMacro first launches two instances of the visualization application with each application image filtering the output image with the appropriate color The filtering is achieved with a Display Handler ManyMacro. A final ManyMacro overlays the two images for display. As an initial phase, the ManyVis tool allows the user to manually rotate a single view to

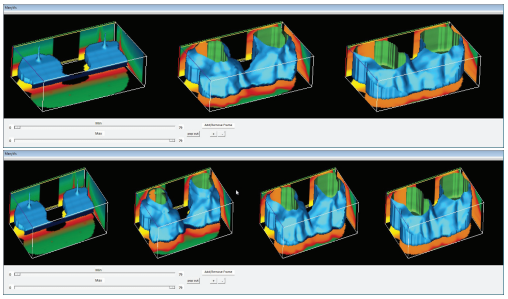

Fig. 5. **Interactive Simulation Filmstrip.** Automatic creation of a filmstrip illustration for time-dependent data. ManyVis provides buttons to define the number of windows to display. Two additional sliders provide user input to denote the desired range of time steps.

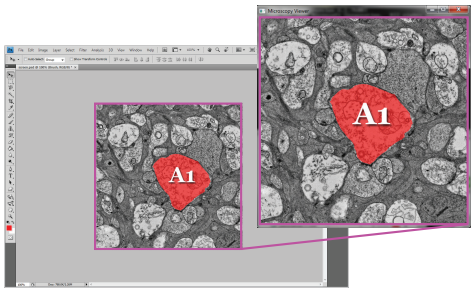

Fig. 6. **Annotation of Microscopy Data.** In this example, ManyVis combines a large-scale microscopy image renderer with Adobe Photoshop to provide a tool for rapid annotation of very high-resolution dataset, a common workflow in using microscopy data.

achieve the desired offset for 3D viewing or automatically apply the offset via a Event Handler ManyMacro. Note, if the viewer exposes an API that allows the input of the view matrix directly, this manual step can be traded for an automatic API Handler ManyMacro. After this initial stage, all input given to the ManyWorkflow is passed to both viewers, keeping the views in sync and in stereo.

Interactive Simulation Filmstrip. Given time dependent data, a common illustration is to provide a contact sheet or filmstrip of key simulation time. As Fig. 5 shows, the ManyVis can be used to aid in its creation. Given an application with time input (slider or text), the a ManyWorkflow can create an initial number of filmstrip slides given a user input. Each slide is a separate instance of the program. However, more sophisticated ManyMacros can be created to leverage the Event and Display Handlers to achieve the same effect with a single application. The time steps for the initial slides are set to be evenly distributed between the first and last desired time steps. This range is an additional user input given to a custom ManyMacro in the ManyWorkflow. After this initial setup, each individual application can be "uncaptured" to fine-tune the desired time step. The ManyWorkflow passes all view interactions to all windows to keep views synchronized.

Annotation of Microscopy Data. A common workflow in microscopy is the labeling and annotating of data. Methods exist for the automatic annotation of digital microscopy data, though they can often be insufficient or specific to a particular test-case. Therefore there is often a manual portion of this workflow where an expert verifies, corrects and even adds additional annotation to the work of the automatic method. Often this microscopy data is extremely high resolution and can be gigapixels in size. This large size can be significant in the complexity of implementing an annotation system. Moreover, the annotation tools created for the scientist must be anticipated in advance. As is often the case, this predetermined solution may be insufficient for some tasks. Depending on the complexity of the tools, this may add significant development time for an implementation. This example is the result of conversations the authors have had with three microbiologists who commonly annotate microscopy images as part of their day-to-day work. Their workflow consists of using Image-Pro to capture and view their microscopy data then exporting this data into

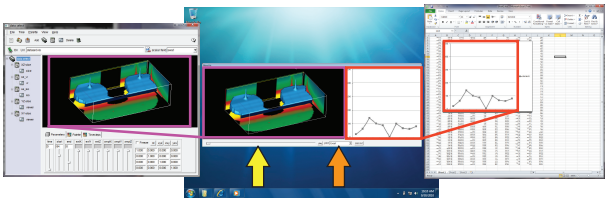

Fig. 7. **Isosurface Custom Visualization.** An isorenderer, Microsoft Excel, and Mathworks MATLAB (not shown) embedded into a single application for an oil reservoir simulation. Users can adjust the timestep selected in all three programs using the slider indicated by the yellow arrow or switch the 1D plotting program between MATLAB and Excel using the button indicated by the orange arrow.

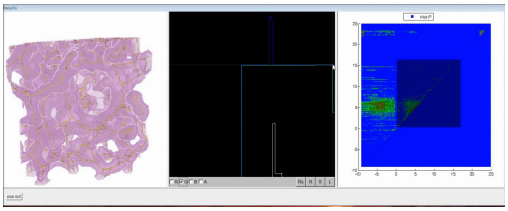

Fig. 8. **Exploring Parameter Space Using a Custom Histogram.** A custom ManyApp for the exploration of the topological parameter space of porous media. This application combines a custom volume renderer and transfer function editor (left and middle) with a 2D histogram provided by MATLAB (right).

photoshop for annotation. After annotation, Neurolucida software is used for analysis of the data. Each stage of their workflow requires tedious saving, loading or converting of data. Moreover, our partner scientists have recently begun to capture entire slides as gigapixel-sized images and desire a quick solution for this data's integration into their workflow. This example represents an initial prototype ManyApp to provide a simple, gigapixel solution for two stages of their workflow.

Assuming we have a stand-alone out-of-core visualizer for the microscopy data, we would like to use ManyVis to rapidly create a meta-annotation system for the dataset from the work of Anderson et al. [8]. We have chosen to use ManyVis to integrate the out-of-core visualizer with Adobe Photoshop [4] inheriting its many sophisticated image editing tools without going through laborious data conversion, see Fig. 6.

Given an instance of the out-of-core visualizer and the image editing program, a ManyMacro can overlay the visualizer's data on the image editor's canvas. The viewer's GUI is trimmed and therefore it appears to the user as if the data is already in the editor's canvas. Since an editor's canvas is often distinct, this overlay is a simple ManyMacro image processing problem to detect the canvas and place the viewer image on it. If the user pans or zooms the microscopy data, the user interaction is passed through to the viewer by the ManyWorkflow. When the user has reached the desired location, the buffer from the viewer window is inserted into the editor's canvas. This is triggered when the user's mouse leaves the canvas area and is seamless. The image data is passed to the editor via a ManyWorkflow with ManyMacros that copy the data from the viewer to the system clipboard and insert into the image editor with a COPY-PASTE operation. When the user is satisfied with their annotation and a save command is given, ManyMacros trigger the image editing application to copy the image buffer back into the clipboard. Our out-of-core visualizer is then triggered by this same ManyMacro to paste the clipboard buffer into its viewing buffer and then save the buffer into the dataset at the given resolution and viewport to alternative color channels. If a viewer does not have this functionality, we could have easily extended the ManyWorkflow/-ManyMacros to save and load a common file. For resolutions finer than the resolution of the edited buffer, we have found linear interpolation of the annotation data to be sufficient to fill in the missing data. However, ManyVis still maintains the flexibility to have a ManyMacro trigger an out-of-core processor to fill in the finer resolution.

	Throughput (inst./s)					
	320	448	483	577	640	700
Max Inst.	200K	200K	53K	14K	10.5K	9K
Final Delay (s)	0	0	6.7	6.4	6.6	6.6
OS/App (inst./s)	320	448	455	457	456	463

Fig. 9. ManyVis is capable of replaying events at a rate of 700 events/second. The throughput of events (clicks) determines the maximum instructions the operating system and calculator application can process without loss (Max Inst.) and the delay in the application after the maximum events are sent (Final Delay (s)). Considering the total runtime with delay, we can estimate the maximum throughput for the OS/application (OS (inst./s)). With a throughput of approximately 450 events/second, one can expect no loss and no delay during replay.

In our companion video we show how this simple ManyApp gives a rapid deployment of an annotation system with a seamless experience as if a user were just working with the single image editing application. Initial feedback has been positive from our microbiologist collaborators and discussions have begun on extensions to cover more of their workflow and deployment strategies.

Isosurface Custom Visualization. There is a need in creating simple environments for distribution of visualization tools for both the dissemination of work, as well as the creation of simple tools to accomplish portions of a scientific workflow. In this example, ManyVis is used to provide a visualization mashup. This application is comprised of three separate applications, two of which are closed-source and commercial. In particular, for this example ManyVis provides a custom ManyApp for the analysis of an oil reservoir simulation. For this analysis, the user is interested in exploring over time both the isosurface of water saturation alongside a 1D plot of oil pressure, see Fig. 7. Microsoft Excel and Mathworks MATLAB are used to provide the 1D visualization and a stand-alone isosurface renderer provides the 3D visualization.

The main idea demonstrated in this example is the ability to combine separate tools into a single ManyApp and synchronize their presentation via a ManyWorkflow and ManyMacros. This enables a user to chose the application that she/he prefers to present the shared data. Via ManyMacros, each application will load the dataset in their preferred standard format at launch. If this format does not exist, a separate ManyMacro can be configured to automatically convert the data file into a more common format. Each tool provides an interface to switch the time-dependent data. In the spreadsheet, this is achieved by activating the next row, in isosurface renderer it is done by moving the horizontal slider in the tool's interface, and in MATLAB this is accomplished through keyboard input into workspace console by varying the index to access the desired row in the matrix. All of these operations are performed by ManyMacros and coordinated by a ManyWorkflow. The ManyWorkflow also provides a user with a custom time slider (via the ManyVis Window Manager). When the user moves this slider, each of time movement ManyMacros are triggered and each application is updated. This allows a continuous inspection of data in different formats taking advantage of each application's strengths. Therefore in a very short deployment time, a unified custom visualization tool can be built by only creating a few, simple actions.

Exploring Parameter Space Using a Custom Histogram. In the example in Fig. 8, we build a custom tool to explore parameter space of porous media. The technique and visualization's interface are made available in Gyulassy et al. [18]. The interface provides four sliders as bounding values for the active contours being visualized. In this example we chose Mathworks MATLAB to produce the 2D histogram of the dataset used in the visualization and embedded this plot within the ManyApp via ManyMacros. A semi-transparent bounding box is placed on top of the histogram image to represent the active bounding box intervals. The user of the tool is allowed to move and resize this box, which translates to adjusting the slider values on the visualization interface. The box drawing and slider adjustment are provided by custom ManyMacros. In supplied video, a user selects a region in the histogram that is of interest and the change is immediately visible in the visualization, which would otherwise require switching between applications. This example targets a common frustration in data analysis and, specifically, for the authors of the topological analysis tool. Often analysis techniques can be highly data dependent. Therefore in researching new approaches, there is often a large trial-and-error process that combines 2D, 3D and topological analysis to understand the nature of the data. Often for 2D analysis, a researcher will use software such as MATLAB rather than creating a custom 2D tool. Therefore there is commonly tedious switching and converting between different programs during analysis. This demo streamlines this typical workflow of several analysis applications into a single experience.

	Time (ms)			
	100%	75%	50%	25%
Gaussian	-	47.6	24.8	20.6
Cubic	-	47.8	24.7	20.4
Lanczos 2-lobe	-	48.3	24.8	21.3
Gaussian + Color	-	54.7	28.2	22.3
Cubic + Color	-	54.6	28.1	22.3
Lanczos 2-lobe + Color	-	54.7	28.4	22.3
Color Only	24.9	14.1	7.5	3.0

Fig. 10. Timings for the optional image (ImageMagick) processing for Google Earth embedded in our Presentation ManyApp (900 x 700 original window size). We provide results for typical downsampling filters and/or HSL color adjustment. All timings are computed as the average of 1000 runs. Overhead for image processing operations is 16 ms. As these number show, applications maintain interactivity.

4 Discussion

Performance. All timings and demos were performed on a 2.67 Ghz i7 Windows 7 system and 6 GB of memory. For macro recording, ManyVis requires on average 1.5 milliseconds (ms) per event to process and store the required data. If we compare that to our test system's USB polling rate of 8 ms (125 Hz) [37], we can safely say that ManyVis event processing will not cause any performance loss for user input even for modestly provisioned systems. During playback, ManyVis can process and send events at a rate of 700 instructions/second. As the USB polling shows, this rate is well above what one would expect from user input. Although, one cannot assume the OS and application to be designed to handle this rate. Fig. 9 provides an evaluation of this performance. This test was performed by sending click events to the OS's calculator to perform a series of additions. Lost events would lead to an incorrect result. Clicks were chosen since the OS and application must process these events. For example mouse movements factor into ManyVis's throughput but if an application does not process these events, they can be considered to be ignored. Our test system could process 9000 events at full rate before an event was lost. Even without loss, there is approximately 6.6 seconds of delay from the time the last event was sent to the time the proper value is displayed on the calculator. This timing was performed via a screenshot of the calculator taken every 100 ms. By considering number of events and the time for the application to process all events, we can calculate that the OS and application can handle a throughput of approximately 450 instructions per second. Running this test with varying throughput, we can see similar results. We can hypothesize that ManyVis can operate with throughput of approximately 450 instructions per second with no lost events or delay in the application. We have tested this rate up to 400K events and verified this hypothesis. This rate is still well above the rate expected from a user created macro. Fig. 10 provides performance results for the overhead incurred from ManyVis's image processing component. This component is optional since a window can be directly passed through and incur no overhead. In Fig. 10 we provide results for the example in Fig. 1 (b), which includes downsampling and/or color correction of a 900 x 700 window. With the component enabled, there is on average 16.1 ms overhead due to the copying and passing of buffers. For downsampling, we have chosen three common downsampling filters although all possible ImageMagick filters were tested. Even more expensive filters, not typical for real time applications, add approximately 10 ms to the filters reported. As the timings show, the application maintains interactivity. For future work, we plan to test other image processing libraries and GPU acceleration to further improve these timings.

Limitations. Our current prototype supports a variety of commercial software such as Microsoft PowerPoint, Microsoft Excel, VLC media player, Google Earth, MeshLab, Adobe Photoshop, and Mathworks Matlab and the authors' own isosurfacing, large image viewer, and topological analysis software. Adding additional software support varies in difficulty. For example, ManyApps such as the ad-hoc stereo or filmstrip example require just simple custom ManyMacros. Examples such as the exemplar Presentation ManyApp are more difficult to develop since they require custom interfaces to a presentation software's API. Therefore a change in presentation software would require new development. However a user adding new applications into the currently supported Microsoft PowerPoint is trivial and requires just interaction without programming. In the current prototype user edits on some features, such as the image processing components, can only be enabled by editing the ManyVis XML. This may not be intuitive for less advanced users, therefore future work will consider how this functionality can be exposed via a user interface. As detailed in the Event Handler description, the use of Windows API inputs when replaying user interactions may cause unintended results if a user begins to send new interactions via the mouse/keyboard while a ManyMacro is being replayed. Although a limitation, inputs are far superior to the alternative (messages) which do not handle multiple windowed applications well. Currently, user interactions are blocked during macro replay to avoid problems. Future work will be to add a visual cue for a user. Currently, our PowerPoint ManyApp is limited to embedding applications into rectangular PowerPoint shapes. Custom ManyMacros can be designed using the Display Handler's image processing routines to support a wider variety of shapes.

Conclusion. In this paper, we have shown the ManyVis framework for enabling the rapid deployment of custom visualization tools. We have detailed the design of the core framework, ManyMacros for sophisticated operations, and the ManyWorkflow for the coordination and synchronization of ManyMacros and applications. These technologies combined provide a powerful development platform which can build customized solutions. Finally, we have provided real world examples of ManyVis enabling the rapid and simple design of custom applications that serve real needs of visualization researchers and their scientific partners.

ACKNOWLEDGMENTS

This work is supported in part by NSF OCI-0906379, DOE 120341, DOE DESC0006872, DOE DESC0001922, DOE DEEE0004449, DOE P01180734, DOE DESC0007446, NTNL 0904631, and DOE/LLNL B597476. This work was also performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 (UCRL-LLNL-CONF-641029).

REFERENCES

[1] The common component architecture forum, http://www.cca-forum.org/.

[2] 3D-CoForm. Meshlab, http://meshlab.sourceforge.net.

[3] Adobe. Acrobat sdk, http://www.adobe.com/devnet/acrobat.html.

[4] Adobe. Photoshop™, http://www.adobe.com/products/photoshop.

[5] R. S. Amant, H. Lieberman, R. Potter, and L. S. Zettlemoyer. Visual generalization in programming by example. *Commun. ACM*, 43(3):107–114, 2000.

[6] R. S. Amant, M. O. Riedl, F. E. Ritter, and A. Reifers. Image processing in cognitive models with segman, 2005.

[7] R. S. Amant and L. S. Zettlemoyer. The user interface as an agent environment. In *Agents*, pages 483–490, 2000.

[8] J. R. Anderson, B. W. Jones, J.-H. Yang, M. V. Shaw, C. B. Watt, P. Koshevoy, J. Spaltenstein, E. Jurrus, K. U. Venkataraju, R. T. Whitaker, D. N. Mastronarde, T. Tasdizen, and R. Marc. Ultrastructural mapping of neural circuitry: A computational framework. In *ISBI*, pages 1135–1137. IEEE, 2009.

[9] Apple. Pasteboard documentation, http://developer.apple.com/library/mac/documentation/cocoa/Conceptual/PasteboardGuide106/PasteboardGuide106.pdf.

[10] L. Bergman, V. Castelli, T. Lau, and D. Oblinger. Docwizards: a system for authoring follow-me documentation wizards. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Cool stuff, pages 191–200, 2005.

[11] F. Berthouzoz, W. Li, M. Dontcheva, and M. Agrawala. A framework for content-adaptive photo manipulation macros: Application to face, landscape, and global manipulations. *ACM Trans. Graph*, 30(5):120, 2011.

[12] Brooks, Ruven. "watch what I do: Programming by demonstration," edited by allen cypher. *International Journal of Man-Machine Studies*, 39(6):1051–1057, 1993.

[13] T.-H. Chang, T. Yeh, and R. C. Miller. Gui testing using computer vision. In *CHI*, 2010.

[14] Derthick, Mark and Roth, Steven F. Example based generation of custom data analysis appliances. In *Proceedings of the 2001 International Conference on Intelligent User Interfaces*, pages 57–64, 2001.

[15] Gimp. Developer guide, http://developer.gimp.org.

[16] F. Grabler, M. Agrawala, W. Li, M. Dontcheva, and T. Igarashi. Generating photo manipulation tutorials by demonstration. *ACM Transactions on Graphics*, 28(3):66:1–66:9, Aug. 2009.

[17] T. Grossman, J. Matejka, and G. W. Fitzmaurice. Chronicle: capture, exploration, and playback of document workflow histories. In K. Perlin, M. Czerwinski, and R. Miller, editors, *UIST*, pages 143–152. ACM, 2010.

[18] A. Gyulassy, V. Natarajan, V. Pascucci, and B. Hamann. Efficient computation of morse-smale complexes for three-dimensional scalar functions. *IEEE Trans. Vis. Comput. Graph*, 13(6):1440–1447, 2007.

[19] Kitware. ParaView. http://www.paraview.org.

[20] Kitware. The Visualization Toolkit (VTK). http://www.kitware.com.

[21] Kurlander, David and Feiner, Steven. A history-based macro by example system. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, By Example I, pages 99–106, 1992.

[22] T. A. Lau, L. D. Bergman, V. Castelli, and D. Oblinger. Sheepdog: learning procedures for technical support. In J. Vanderdonckt, N. J. Nunes, and C. Rich, editors, *IUI*, pages 109–116. ACM, 2004.

[23] H. Lieberman. Integrating user interface agents with conventional applications. In *IUI '98: Proceedings of the 3rd international conference on Intelligent user interfaces*, pages 39–46, 1998.

[24] H. Lieberman. *Your Wish Is My Command — Programming by Example*. Morgan Kaufmann, 2001.

[25] Microsoft. Clipboard documentation,. http://msdn.microsoft.com/en-us/library/ff468801%28v=vs.85%29.aspx.

[26] Microsoft. Office sdk, http://msdn.microsoft.com/en-us/office/aa905340.

[27] F. Modugno and B. A. Myers. Pursuit: Visual programming in a visual domain. Technical Report CMU-CS-94-109, Carnegie Mellon University, The Human Computer Interaction Institute, Jan. 94.

[28] OMG. Common object request broker architecture, http://www.omg.org.

[29] OpenOffice. Developer guide, http://wiki.services.openoffice.org/wiki/Documentation/DevGuide/OpenOffice.org_Developers_Guide.

[30] V. Organization. Vlc, http://www.videolan.org.

[31] R. Potter and B. Shneiderman. Pixel data access for end-user programming and graphical macros. Technical Report CS-TR-4019, University of Maryland, College Park, May 1999.

[32] R. L. Potter. *Pixel data access :–interprocess communication in the user interface for end-user programming and graphical macros*. PhD thesis, research directed by Dept. of Computer Science, 1999.

[33] Ramachandran, Ashwin, Young, and R. Michael. Providing intelligent help across applications in dynamic user and environment contexts. In *Proceedings of the 2005 International Conference on Intelligent User Interfaces*, Short papers: personal assistants, pages 269–271, 2005.

[34] D. A. Simulation and C. Initiative. Visit. https://wci.llnl.gov/codes/visit/.

[35] A. Sugiura and Y. Koseki. Internet scrapbook: Automating web browsing tasks by demonstration. In *ACM Symposium on User Interface Software and Technology*, pages 9–18, 1998.

[36] Sugiura, Atsushi and Koseki, Yoshiyuki. Simplifying macro definition in programming by demonstration. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Papers: Programming by Demonstration, pages 173–182, 1996.

[37] O. Tscherwitschke. Mouse rate checker, . http://www.tscherwitschke.de/old/mouseratechecker.html.

[38] Zettlemoyer, L. S., S. Amant, Robert, Dulberg, and M. S. IBOTS: Agent control through the user interface. In *Proceedings of the 1999 ICIUI*, Information Retrieval Agents, pages 31–37, 1999.

[39] L. S. Zettlemoyer and R. S. Amant. A visual medium for programmatic control of interactive applications. In *CHI*, pages 199–206, 1999.