Supporting Mixed Presence Groupware in Tabletop Applications

Peter Hutterer[1,2], Benjamin S. Close[1,2], and Bruce H. Thomas[1,2]

[1]*Wearable Computer Laboratory*
School of Computer and Information Science,
University of South Australia
Mawson Lakes SA 5095
{peter|cisbjc|thomas}@cs.unisa.edu.au

[2]*National ICT Australia*
Australia Technology Park
Bay 15 Locomotive Workshop
Eveleigh NSW 1430

Abstract

In this paper we present the Transparent Input Device Layer framework to extend Java applications with support for multiple distributed input devices, a major requirement for tabletop applications. This overcomes a key restriction of current graphical environments to support only a single system cursor and one keyboard, and allows the cursor and keyboard control of applications to be performed by input devices that are connected to other hosts on the network. Applications can be developed with this framework and therefore allow operations such as simultaneous drag-and-drop by multiple users. Additionally, we have created a wrapper application to inject support for multiple input devices into legacy applications at runtime—without the need for code alteration or recompilation.

We present two tabletop applications that make use of our framework: One is a graphical front-end to a military course of action scheduling application and was developed with the framework. The second application, a component based data visualisation application, employs the injection wrapper application to gain support for distributed multiple input devices at runtime.

1. Introduction

Tabletop displays for collaborative workspaces introduce a new set of challenges. Applications for desktop computing are traditionally designed for a single user, while tabletop displays are ideally suited to support multiple users on one site working shoulder to shoulder. Our NICTA colleagues and ourselves are investigating the use of tabletop display technology, in conjunction with Access Grid video teleconferencing technology to increase telepresence in the collaboration. The NICTA Visualisation and Interaction over Collaborative Access Tables (ViCAT) project employs three tabletop displays with Access Grid nodes at three geographically different locations. Each table or CAT consists of a vertical projection area that is utilised by two short-throw projectors and one back-projected horizontal area, as shown in Figure 1. We use the vertical displays for video conferencing; the horizontal display supplies the main working area.

ViCAT supports multiple users at each site and over a wide area network; a CSCW domain known as Mixed Presence Groupware (MPG). MPG connects both co-located and distributed collaborators and their disparate displays via a common shared virtual workspace.

Video conferencing is supported via AccessGrid [4]. Not having the video and audio connection in the groupware application itself restricts the application from using the conferencing system for very specific tasks. One of these uses has been presented in [8], where the video system can be used to draw onto traditional paper and then overlay the image onto the application. This allows users to annotate using traditional input devices such as pens and pencils. However, an application independent conferencing system allows the use of different groupware applications simultaneously.

We have created a MPG framework, the Transparent Input Device Layer (TIDL), that gives application developers the following functionality: 1) The ability to develop multi-user applications with a

Figure 1. A Collaborative Access Table (CAT).

traditionally GUI API, and 2) To provide a meaningful subset of multi-user support to legacy applications without modifications to the source code or the need for recompilation.

TIDL overcomes the limitation of current graphic environments that only effectively support one input device. Although all major operating systems support multiple devices, the devices' input merge down to one system cursor and one system keyboard. This limitation prevents the independent control of multiple cursors or having simultaneous keyboard foci on different GUI components in one application. If an application has the need to support multiple and independent input devices, this support has to be implemented by the application developer by implementing the functionality of multiple system cursors. This replicated functionality includes the following: drawing the different mouse cursors on top of the application's GUI, the handling of multiple foci for different users and network distribution of input device events. Consequently, supporting multiple input devices at the application level results in inconsistent behaviour across applications and can result in difficulties running multiple applications simultaneously. TIDL provides a generic and consistent manner of using local and remote input devices across applications. We have successfully run applications with twelve mice concurrently, four at one host and eight at the second host.

TIDL provides the following features to applications linked to its API and to legacy applications: network transparency for all input device events, uniquely coloured cursors with specific colouring for a particular host and an annotation layer which operates on a per user basis. For applications developed with the TIDL API, unique device identification is provided in all events generated by the toolkit to any application that wishes to use the information.

A critical requirement is to allow users to dynamically add or remove input devices to the local machine and to collaborate with groups at different geographic locations. Each user's input devices are independent of the other devices and all users work simultaneously on different elements on the application's GUI or even on different applications. The MPG support must have a policy free floor control paradigm, to allow these decisions to be made by the end user or application programmer.

This paper starts with a description of the related work to our framework, and we then present TIDL, a Java based framework, that allows developers to create applications with support for multiple input devices across different hosts. A wrapper application with TIDL is described that enables users to use legacy applications that do not support multiple input devices. The use of this wrapper application does not require modification or recompilation of the application's source code. Two example tabletop applications are presented as a demonstration of the framework in operation. The paper finishes with some concluding remarks.

2. Related Work

Collaborative software can be divided into real-time and non-real-time groupware. Non-real-time groupware such as IBM Lotus Notes or Microsoft's Outlook calendar is commonly used in business environments today. Presently, there exist three different variants of real-time groupware: Single Display Groupware (SDG), Shared Display Groupware (ShDG) and Mixed Presence Groupware (MPG). Single Display Groupware allows multiple users with an arbitrary number of input devices, simultaneously on one computer. Shared Display Groupware connects single users on different machines to work collaboratively on one application, with one set of input devices on each machine. Mixed Presence Groupware combines these two approaches, allowing multiple users simultaneously on each node in the network. Figure 2 illustrates the distinction between real-time and non-real-time groupware and the differences between SDG, ShDG and MPG.

2.1 Single Display Groupware

Steward et al. [14] coined the term Single Display Groupware. Stewart et al. [13] conducted a set of studies which showed that people prefer to have individual and independent input devices when working simultaneously on the same machine.

The Multi-Device Multi-User Multi-Editor (MMM) [2] was one of the earliest applications to support multiple independent devices. Additionally to independent devices, all users also had a specific home area and user dependent menus. The PEBBLES project [9] used PDAs as input devices. The stylus could be used to move cursors on an application running on a PC and to type text on the PDA's on-screen keyboard. Using a PDA as an input device, also supplies the users with a small private screen.

The MID toolkit [7] allows multiple devices within one application. This Java toolkit uses calls to the Microsoft Windows 98 API to extract the device information for each connected device and then resembles AWT events by sending MID events with a device number instead of the normal AWT events. The MIDDesktop [12] is constructed with the MID toolkit. MIDDesktop supports the execution of multiple applets simultaneously within one desktop frame. All applications can receive events, however, the applets naturally only support one input device at a time. Simultaneous drag and drop is not possible if there is no support from the applet.

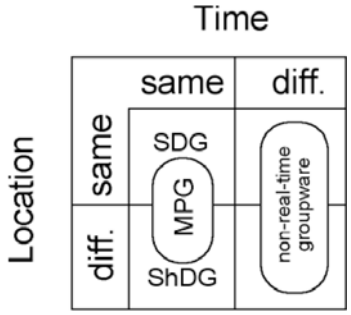

Figure 2. Time-Location matrix for groupware. SDG, ShDG and MPG are all branches of real-time groupware.

Quite similar to MID is the SDGToolkit [18], which makes use of the Windows XP RawInput API to determine the specific device causing a particular event. The SDGToolkit was designed with tabletop displays in mind and can orient the cursors according to the position of the people around the table. MID and the SDGToolkit both use a listener concept to notify the application of input events and augment the events with the input device's ID.

The DiamondTouch input device supports SDG features in hardware [6]. This touch device can track the finger positions of two users at the same time and delivers the appropriate events to the software. Rogers et al. utilised the DiamondTouch to study how equal access to a tabletop infrastructure influences the decision making process [10].

2.2 Shared Display Groupware

Shared Display Groupware (ShDG) supports single users with one computer each. Their desktop or application window is shared over the network with a set of single users operating on individual computers. Instead of supporting multiple input devices on one host, ShDG toolkits and applications utilise input devices on different hosts. The main concern is synchronising the devices as events from remote devices may have significant delays.

ShDG is usually built using either a replicated or centralised architecture. A replicated architecture requires the application to be run on each host. Only input events are distributed amongst the system. All instances are expected to react in the same way to the input events, thus keeping all hosts synchronised. In a centralised architecture, the application runs on one central server and its output is broadcast to each node. The replicated approach requires significantly less bandwidth but is less robust than the centralised approach. If one of the input events is lost, the applications can become unsynchronized and it is hard to restore a synchronized state.

An early implementation of a ShDG toolkit is Rapport [1]. A main design goal of Rapport is to allow the execution of legacy single-user applications. Rapport utilises the principle of virtual meeting rooms with the ability that users can enter and leave any room as they wish, resembling the behaviour of users in real-life meetings. Rapport uses a single-site execution scheme, running the application on one computer and displaying the application's output on all connected machines.

MMConf [5] employs a multi-site execution scheme that executes the application on all hosts and synchronises the input events on all machines. One benefit of this execution scheme is the low bandwidth requirements, but if events get lost, the applications states can become inconsistent. GroupKIT [11] is a toolkit to support the development of real-time groupware by leveraging event synchronization, registration of participants and more.

2.3 Mixed Presence Groupware

MPG applications and toolkits need to support multiple input devices simultaneously on a local host but also input devices on remote hosts. Tang et al. [16] [17] implemented a group drawing tool, MPGSketch, that allowed multiple users to draw on one shared surface. They identified a strong disparity in the conversation between the co-located peers and the remote users. Although the remote users were able to draw on the same surface, most of the communication occured between co-located peers. Tang et al. drew digital shadows of the users' arms onto the drawing surface to make distributed collaborators more aware of other users' actions. MPGSketch uses the SDGToolkit [18] to gather events from multiple devices on a single host. The distribution of the events to the remote sites is done using the Collabrary shared dictionary [3]. However, their MPGSketch is a single application and does not provide a toolkit for implementing other MPG applications.

3. Transparent Input Device Layer

TIDL is an API to allow applications to utilise an arbitrary number of input devices on different hosts. Instead of fetching information about the position and state of the system cursor, we access the underlying operating system's interface to gather events from the devices. These events are then dispatched using Java's GUI event processing system.

We use a replicated architecture, with applications operating on each host. TIDL provides the necessary framework to broadcast the events to each of these applications. TIDL operates in conjunction with the application (if the application employs the TIDL API at creation time by the

developer) or through a TIDL wrapper application for legacy applications (if injected into the application at start up time).

3.1 Support for groupware applications

TIDL is designed to minimise the complexity of developing MPG applications, and to be no more complex than the current Java AWT event API. Our MPG event system fits neatly into Java's own event system and does not require changes in existing code unless the developer wishes to utilise the additional input device information provided by TIDL. Using the input device information allows the developer to provide programmatic control for simultaneous events that may occur. For example, a disaster coordination team looking at a map displaying the disaster's area may be distributed across different sites, with multiple emergency team coordinators on each site. Each of those coordinators have their own input devices to move emergency units such as fire brigade trucks or ambulance cars on a map to their designated location. Because of the independence of the input devices, team coordinators can discuss and annotate the map and the position of the emergency teams while other coordinators can move around other teams at the same time. TIDL assigns colours to each cursor, so by looking at the map the coordinators can see who is doing changes or annotations.

TIDL allows any user to use an annotation layer and draw and write comments onto this layer. Annotation is user specific, so while one user annotates the application, all other users can still use the application. The application developer can utilise the extra information provided by TIDL to restrict the viewing of annotations to a specific user.

TIDL automatically caters for network synchronisation, the distribution of events to all nodes in the network and draws a mouse cursor on top of the application for every input device in the system..

3.2 Support for legacy applications

The TIDL API allows developers to build applications with active support for multiple independent input devices. TIDL also allows legacy applications to make use of multiple input devices without altering or recompiling the application. A legacy application can be initiated from within a TIDL wrapper application to insert an invisible layer in front of the application. This layer then works as an annotation layer and intercepts all AWT events that are delivered to the application. Those events are translated into custom events and dispatched to the application, as shown in Figure 3. Because the custom events are subclassed from the AWT event class, the application does not know that the events are generated from a different source than real AWT events. TIDL also draws the mouse cursors of each input device onto an invisible layer atop of the application. TIDL supports different foci for all connected keyboards, and this allows users to type in different text fields simultaneously even without native support from the application. Each mouse and keyboard combination is handled as user in TIDL and a mouse click on a specific component changes the keyboard focus (if applicable) for the user who initiated the click. Succeeding keyboard events are then sent to the appropriate component.

4. Implementation

The TIDL framework consists of two different libraries. One is the TIDL library that broadcasts events to different hosts, containing the TIDLGlassPane and injection functionality. The second library is the Multiple Device Direct Interface (MDDI) which processes low level events for input devices from the operating system, TIDL itself relies on the events received from MDDI. The following sections will outline each of these libraries.

4.1 Multiple Device Direct Interface

MDDI accesses the devices connected to the local computer and wraps the device's events into Java objects which can then be broadcasted to remote nodes by TIDL. Due to Java's limitation to one system cursor we have to access the devices using operating system dependent APIs. Currently, we have implementations for Microsoft Windows XP and Linux. Processing the devices through the native APIs allows only relative mouse events instead of absolute events in both implementations.

The Windows XP Raw Input API provides developers with methods to receive events caused by any connected input devices. If an application registers a device on the Raw Input API, it will receive those messages in the same way as a standard application window receives notifications such as window closing events or the like. Different to standard input events however, Raw Input also supplies a handle to identify the device producing that event. From this handler a unique device number is created to mark all events. Using the Java Native Interface (JNI), the data is wrapped into a Java object and then passed onto the TIDL layer.

Under Linux, we chose a different approach. Instead accessing the devices via a high-level API, the device files are read from in the devfs virtual file system. Each connected device is represented as a file (i.e. the first mouse is /dev/input/mouse0) and the raw data coming from the device can be read directly from the file. Writing to the file sends data back to the device, but standard permissions on these

files are very restrictive in a default installation. Reading from or writing to the files using input streams in Java requires root access. Running any Java application as root imposes a security risk on the system. To avoid this, a python script fetches the data from the devices and provides it on a TCP socket. The Java interface to the Linux implementation of MDDI then reads data from this socket and wraps it into objects to be passed on to TIDL. Alternatively, one could set the permissions on the device file to be readable and writeable by any user.

MDDI adds the device identifiers to each message passed on to TIDL, so it can be used as library for any application that needs to utilise SDG paradigms.

4.2 TIDLGlassPane

The TIDLGlassPane is the central component for applications that make use of the TIDL architecture. A GlassPane can be used in every javax.swing.RootPaneContainer, which is the interface for applets, frames, dialogs and windows in Swing. A GlassPane is by default transparent and appears to be on top of the application window, and a GlassPane is the first component to receive AWT events caused by the Java Virtual Machine (JVM).

The TIDLGlassPane intercepts AWT events and consumes them. The application itself never receives those events. Instead, the TIDLGlassPane receives events via the TIDL framework and converts them into custom subclassed AWT events, and passes them on to the application. Figure 3 illustrates this concept. An application can therefore use the same listeners it would use for receiving regular AWT events. The TIDL AWT events convert the relative coordinates to absolute coordinates as required by the AWT event.

A GlassPane is by default transparent, all TIDL supported mouse cursors are drawn on this surface. Every mouse cursor has a specific colour, dependent on the host name and the device number. For example, all devices on host *A* may have blue cursors in different shades while all devices on host *B* have red cursors in different shades. One user (one keyboard/mouse combination) always has the same colour, even if an application is closed and another application is started. This assists in easy identification of who is controlling a cursor.

Multiple simultaneous foci for mouse cursors and keyboards is supported through reassembly of the AWT events in the TIDLGlassPane, and does not rely on the events generated by the JVM. The multiple simultaneous foci are supported for devices connected locally and remotely. For example, this enables users to type into different text fields simultaneously.

The basic building blocks of TIDL are modules that receive all events from TIDL that would otherwise be passed on to an underlying component. Modules share the graphics context with the GlassPane, thus everything a module draws onto this context is represented on the same layer as the mouse cursors. The *annotation* module supports the ability to comment on the screen overlaying the application. As the GlassPane covers the whole frame including the menu bars, users can even use the annotation layer to annotate or comment menus. The annotation layer works on a per-user basis so other users can still work on the application normally while one or more users place annotations on the screen.

4.3 TIDLInject

TIDL provides an elegant API to make applications aware of multiple distributed input devices. However, an application is required to instantiate the TIDLGlassPane and assign the TIDLGlassPane to the application's window to receive TIDL events. While this is straightforward for newly designed applications, legacy applications require a different approach. One option is to modify the source code of legacy application with a recompilation. A second approach (the one we have employed) is to utilise our TIDLGlassPane with any legacy application without the need for recompilation.

We have created a wrapper application called TIDLInject that allows us to inject our TIDLGlassPane into an application at runtime. Instead of starting the application normally, the application's main method is invoked by TIDLInject. After the application has fully started up, TIDLInject uses the java.awt.Frame.getFrames() method to get references to all active frames and searches for a JFrame. Once the JFrame is found, a TIDLGlassPane is set to the GlassPane of the JFrame. As the TIDL AWT events are subclasses of the standard AWT events, the application does not notice that it receives TIDL AWT events.

If the application starts up multiple JFrames,

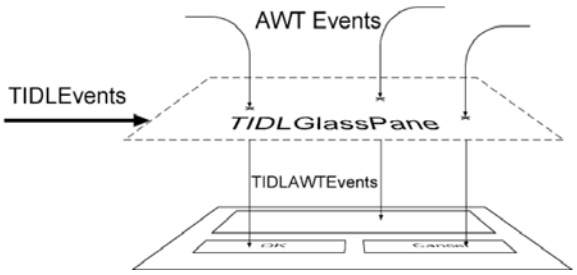

Figure 3. The TIDLGlassPane intercepts all AWT Events and discards them. The TIDLGlassPane receives TIDLEvents via the TIDL system and converts them into TIDLAWTEvents. These events are then passed on to the application.

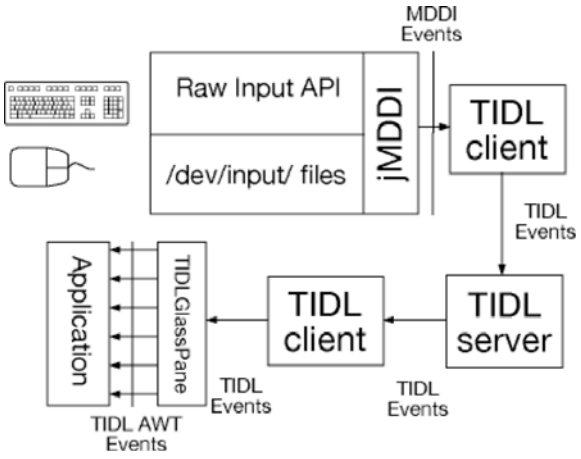

Figure 4. Architecture of the MDDI and TIDL system.

TIDLInject creates a frame the size of the entire display, and within this frame, it resembles the application's JFrames as JInternalFrames. Swing makes use of ContentPanes and we can just use the ContentPane of any JFrame as ContentPane of a JInternalFrame to make an exact copy of the GUI of any application. Using JInternalFrames also allows for TIDL to operate over multiple applications at the same time, supporting multiple input devices across the different applications simultaneously.

4.4 Limitations

There are some limitations on our approach. We are using a replicated architecture, which requires the application to run on each host. Although this requires less bandwidth, the replicated architecture is less robust than the centralized approach. If one relative mouse movement event is lost, the cursor can end up in different positions on the different sites. If a user then clicks a button, this may cause different actions, as the cursor may be over different components. Disparities also occur if the application uses random numbers or if the user needs to access a specific file on the host. In the replicated approach, every user sees only local files.

Because legacy applications have no knowledge of the multiple input devices, some operations are not simultaneously supported, such as drag and drop. The same limitation applies to pop-up menus. If one user causes a pop-up menu to appear, any click outside of the menu causes the menu to disappear. The standard GlassPane, which can be used with any JFrame in the Swing model, is employed by TIDL to insert a layer atop of the application. However, if an application uses the GlassPane itself, TIDL would replace the original GlassPane and lose some of the application's functionality.

As mentioned before, a GlassPane can only be inserted onto one JFrame. If the JFrame closes and the application opens another application window, the application also loses support for multiple input devices. Moreover, if the application causes a dialog box or a new frame to appear, this new window does not have TIDL support. However, it is possible to avoid this by continuously polling the JVM for all open frames, dialog boxes and windows and resemble new ones in the desktop-wide frame mentioned in Section 4.3.

5. Applications

We have employed MPG with two applications that make use of TIDL. The first, MPGCoast is an extension to the Course of Action Scheduling Tool (COAST) created by Zhang et al. [19], and this application is custom built using the TIDL features. The second application ViSOR was developed by the VisLab at the University of Sydney to visualize large data sets, and we employed the TIDLInject wrapper application to provide the collaborative functionality.

5.1 MPGCoast

The COAST application provides military planners with a tabular input data model to aid military planning operations. The system uses domain specific tasks, resources, and conditions to determine the required sequence of events to perform effects based planning. In the original COAST application, user input is limited to a single user. The tabular input method of COAST makes intense use of popup windows. This limits the possibilities of simultaneous user input as popup windows may occlude text fields other users are editing. COAST uses a client-server model, with the client being the data input interface and the server calculating the possible courses of action for the given data.

Our MPGCoast extension runs on the client side of COAST. Instead of displaying the tasks and dependencies in a tabular view, we employ the

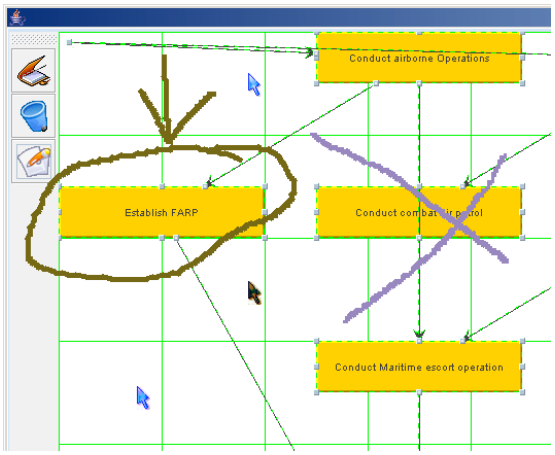

Figure 5. MPGCoast actively supports multiple devices.

jGraph libraries[1] to create a graph to visualise the dependencies of the different tasks, see Figure 5. All nodes in the graph represent a specific task and can be moved freely on the user interface. Tasks dependent on other tasks are linked to the parent task.

MPGCoast makes active use of the TIDL framework, allowing multiple users to manipulate graph nodes, add defined sub-graphs of nodes, and delete nodes in a collaborative nature. Users may switch back to the tabular view at any time to view the data in a textual representation.

One of the main benefits of using TIDL in MPGCoast is the annotation layer, which can be switched on via a button on the left hand menu. As the layer is user-dependent, different users can discuss a set of tasks visually, while other users can still move and add new tasks. As mentioned before, when the annotation layer is enabled, no events are passed on to the application. Using the jGraph library imposes some limitations. As jGraph is not designed for multi-user use, nodes cannot be moved simultaneously.

5.2 ViSOR

ViSOR (Visualization of Spatially-Oriented Relationships) [15] provides mechanisms to dynamically construct a visualisation application based on multiple users' expertise at run-time. ViSOR employs visual programming allowing users to connect independently developed software components (such as data analysis and visualisation tools) together to construct an application.

ViSOR allows the development of a visualisation application for data analysis as the result of collective efforts from different domain experts. Unlike other visual programming environments and problem solving environments, this system enables users to interactively design the visualization application while the application is executed, providing real-time feedback. The capability provided by TIDL allows the process of collaborative design in the visualisation application to be carried out by not only the local users but also remotely located collaborators. ViSOR does not actively support TIDL in its code. Instead, the TIDLInject wrapper application makes ViSOR multi-device aware. Figure 6 shows a screenshot of ViSOR visualising breast and cervical cancer rates from Pennsylvania, Kentucky and West Virginia.

By augmenting ViSOR with TIDL, scientists at different geographic locations can work on a data set simultaneously. The annotation layer in TIDL makes it easy for group members to point out specific areas of interest and discuss them without the need to modify the actual data.

[1] http://www.jgraph.com

6. Conclusion

In this paper we introduced the TIDL framework that allows the use of multiple input devices within an application. Current graphic environments do not generically support the use of multiple independent cursors and/or keyboards. TIDL supports an arbitrary number of input devices on any number of hosts, whereas previous toolkits were limited to either one host or one input device per host.

With our TIDL architecture, we send subclasses of Java AWT events that are tagged with the device's ID across a network, allowing applications to use any number of mice and keyboards from remote machines just as if they were connected locally. Our TIDLGlassPane provides an easy way to connect to other machines running TIDL and convert the TIDL events into subclasses of AWT events. The TIDLGlassPane, which is inserted in front of a Swing JFrame, draws the multiple cursors, provides a user-dependent annotation layer, and keeps track of the foci for the different keyboards. The multiple foci allow multiple users with keyboards to type into different textboxes simultaneously.

While the TIDLGlassPane can be used by a new application just with a few lines of extra code, we have developed support of TIDL features for legacy applications. We have created a wrapper application called TIDLInject that enables multiple input devices in third party applications without the need for recompilation or altering the source code. TIDLInject retrofits applications at run-time with

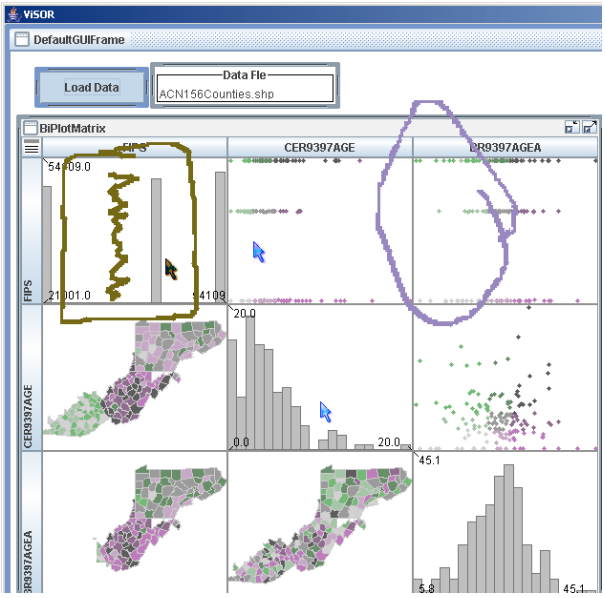

Figure 6. ViSOR is a data visualisation application that is retrofitted with support for multiple input devices at run-time.

support for multiple distributed input devices and an annotation layer. Regardless of how it is used, TIDL provides fundamental support for MPG aware applications.

7. Acknowledgements

We would like to thank the ViCAT team, especially Peter Eades, Julien Epps, Masahiro Takatsuka and Mike Wu for their help in testing and improving TIDL, the DSTO for their support in providing information about and access to COAST, and Wayne Piekarski and Stewart Itzstein for their comments on this paper. This project is funded by the National ICT Australia as part of the ViCAT project.

8. References

[1] Ahuja, S. R., Ensor, J. R., and Horn, D. N. *The rapport multimedia conferencing system.* SIGOIS Bull., Vol. 9, No. 2-3, pp 1--8, 1988.

[2] Bier, E. A., Freeman, S., and Pier, K. MMM: The multi-device multi-user multi-editor. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp 645--646, Monterey, California, United States,

[3] Boyle, M. and Greenberg, S. *GroupLab Collabrary: A toolkit for multimedia groupware.* ACM CSCW Workshop on Networking Services for Groupware, 2002.

[4] Childers, L., Disz, T., Olson, R., Papka, M. E., Stevens, R., and Udeshi, T. *Access Grid: Immersive Group-to-Group Collaborative Visualization.* Ames, Iowa, 2000.

[5] Crowley, T., Milazzo, P., Baker, E., Forsdick, H., and Tomlinson, R. MMConf: an infrastructure for building shared multimedia applications. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, pp 329--342, Los Angeles, California, United States,

[6] Dietz, P. and Leigh, D. DiamondTouch: a multi-user touch technology. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pp 219--226, Orlando, Florida,

[7] Hourcade, J. P. and Bederson, B. B. *Architecture and Implementation of a Java Package for Multiple Input Devices (MID).* College Park, MD 20742, USA, 1999.

[8] Ishii, H. and Miyake, N. *Toward an open shared workspace: computer and video fusion approach of TeamWorkStation.* Commun. ACM, Vol. 34, No. 12, pp 37--50, 1991.

[9] Myers, B. A., Stiel, H., and Gargiulo, R. Collaboration using multiple PDAs connected to a PC. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pp 285--294, Seattle, Washington, United States,

[10] Rogers, Y., Hazlewood, W., Blevis, E., and Lim, Y.-K. Finger talk: collaborative decision-making using talk and fingertip interaction around a tabletop display. In *CHI '04: Extended abstracts of the 2004 conference on Human factors and computing systems*, pp 1271--1274, Vienna, Austria,

[11] Roseman, M. and Greenberg, S. GROUPKIT: a groupware toolkit for building real-time conferencing applications. In *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pp 43--50, Toronto, Ontario, Canada,

[12] Shoemaker, G. B. D. and Inkpen, K. M. *MIDDesktop: An Application Framework for Single Display Groupware Investigations.* Burnaby, BC, Canda, 2001.

[13] Stewart, J., Raybourn, E. M., Bederson, B., and Druin, A. When two hands are better than one: enhancing collaboration using single display groupware. In *CHI '98: CHI 98 conference summary on Human factors in computing systems*, pp 287--288, Los Angeles, California, United States,

[14] Stewart, J., Bederson, B. B., and Druin, A. Single display groupware: a model for co-present collaboration. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp 286--293, Pittsburgh, Pennsylvania, United States,

[15] Takatsuka, M. *A component-oriented software authoring system for exploratory visualization.* Future Generation Computer Systems: Journal of Grid Computing: Theory, Methods and Applications, Vol. 21, No. 7, pp 1213--1222, 2005.

[16] Tang, A., Boyle, M., and Greenberg, S. Display and presence disparity in Mixed Presence Groupware. In *CRPIT '28: Proceedings of the fifth conference on Australasian user interface*, pp 73--82, Dunedin, New Zealand,

[17] Tang, A., Neustaedter, C., and Greenberg, S. *Embodiments for Mixed Presence Groupware.* Calgary, Alberta, Canada, 2004.

[18] Tse, E. and Greenberg, S. Rapidly prototyping Single Display Groupware through the SDGToolkit. In *Proceedings of the fifth conference on Australasian user interface*, pp 101--110, Dunedin, New Zealand,

[19] Zhang, L., Kristensen, L. M., Mitchell, B., Gallash, G., Mechlenborg, P., and Janczura, C. COAST - An Operational Planning Tool for Course of Action Development and Analysis. In *Proceedings of the 9th International Command and Control Research and Technology Symposium*,