Employing Usability, Efficiency and Evolvability in the CEXI Toolkit

Edward Tse
University of Calgary
2500 University Dr. N.W.
Calgary, Alberta, Canada
(403) 210-9501
tsee@cpsc.ucalgary.ca

ABSTRACT

Computer displays are expanding beyond the upright desktop and towards personal devices such as Tablet PCs and large public displays (such as walls and tables). These different form factors require researchers to develop suitable interaction techniques. The fundamental problem is that existing development environments assume that everyone will using a mouse for all pointing input. Thus most applications are not able to take advantage of the extra features provided by novel input devices such as the point sizes provided by the Smart Technologies DViT Board. Most input device developers provide Software Development Kits (SDKs) written with legacy C++ code and different SDKs provide different APIs making it hard to port code written for one input device to another.

This paper describes the Centralized External Input (CEXI) toolkit, a toolkit that supports the rapid prototyping of applications with a variety of novel input devices. Since this is a third generation tool, I wanted to the toolkit to be usable, efficient and evolvable. These are three lessons (or patterns) gleamed from my experiences and the experiences of other toolkit developers.

To make the toolkit API easy to use, I limit the assumptions made in the API, for example I do not expect programmers to know how to traverse an object oriented class hierarchy of different input events, instead I provide all the important event information in a single monolithic event argument. To make the toolkit efficient, I use event queueing in the control panel to control the rate of events per second and I use quenching in both the input forwarder and the client to ensure that they receive only the information that they are interested in. Finally, I make the toolkit evolvable by making the source code available and making it easy for third parties to develop their own input forwarders.

1. Introduction

During my Masters thesis, I worked on the Single Display Groupware (SDG) Toolkit. SDG is an area of research that explores how multiple users share a single display such as a computer monitor, a large wall display, or an electronic tabletop display using multiple input devices such as multiple mice. Since existing programming environments assumed that there would only be a system single mouse researchers and programmers faced considerable hurdles if they wish to develop SDG.

The SDG Toolkit was designed to simplify SDG development by treating input from multiple mice as separate streams, automatically drawing multiple cursors, handling different orientations of a mouse around a table and providing a means of developing widgets that understood multiple users.

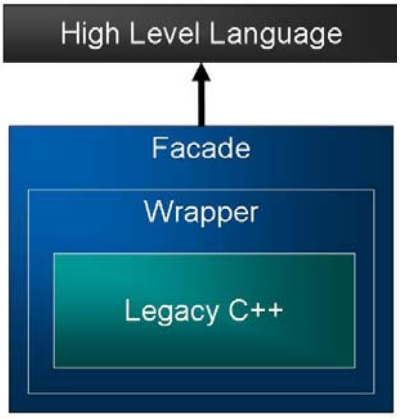


Figure 1. Our early approach to accessing legacy C++ code

Life was good for about six months, then our lab obtained several new input devices such as the MERL Diamond Touch [Dietz and Leigh, 2001] and the Smart Technologies DViT SmartBoard [http://www.smarttech.com] that could support multiple simultaneous touches. We used an open source computer vision library [http://www.intel.com/research/mrl/research/opencv/] from Intel to track a 3 Dimensional wand with multiple web cameras and we purchased a wacom tablet and some Tablet PCs.

Rob Diaz [2] and I tried to use SDG Toolkit principles to make separate toolkits for each input device but it quickly became apparent that we needed a more general solution.

The fundamental problem was that our old strategy for handling input devices used a heavyweight two layer process as seen in Figure 2. Since every new input device provided a Legacy C++ example, we would take this Legacy C++ code (e.g., the Diamond Touch SDK) and wrap it with tools such as ATL COM or the .NET Interoperability Services. After wrapping, we would try to create an easy to use interface around the wrapper (called a façade) so that it was accessible to those using high level languages such as Visual Basic. We used a strategy similar to Sneed's encapsulation of legacy software paper [2000].

Creating the wrapper and then the façade was a long process that often would take weeks to complete. A problem with wrapper code is that it does not evolve well, when a new version of an SDK is released, it takes quite a bit of work to modify the wrapper and the connection with the façade to support the new modified API. With ATL COM it took less time to completely rewrite the wrapper than it did to make a simple modification. Also, the façades were bound to specific programming

Tse, Edward (2005) Employing usability, efficiency and evolvability in the CEXI Toolkit. Report 2005-783-14, Department of Computer Science, University of Calgary, Calgary, Alberta CANADA T2N 1N4. April.

environments and we had a different façade and wrapper for each input device. Thus end programmers could not easily migrate across different input devices.

For example, often programmers would want to prototype applications using multiple mice and then adapt their application to a DViT Smart Board. This involved commenting out all of the mouse code, changing the event method callback and making sure that the existing code did not depend on any features that were not mouse specific (e.g., the mouse wheel). This greatly increased the complexity of prototyping applications across multiple input devices as existing code was not easily portable.

Consequently I created a new version of the toolkit using a technique similar to Purtilo's polylith architecture [1994] to simplify the capture of input from legacy C++ SDKs and to make the end programmer experience the same regardless of what input device was used. The toolkit would use a centralized database and allow external applications to forward and receive input. Thus the name of this toolkit is the Centralized External Input (or CEXI) Toolkit.

2. Toolkit Patterns

I wanted to design a toolkit that learned from my previous mistakes and the mistakes of other toolkit developers. Thus, I read papers related to toolkit development in the Software Engineering and HCI Community and tried to apply their understanding into a set of three patterns for toolkit development:

1. **Usable**: Often assumptions built into the end programmer API can prohibit its use. For example, the Jazz Toolkit [Bederson, et al., 1994] required every programmer to understand a node class hierarchy to build even simple applications. The authors stated that this prohibited its use by the Human Computer Interaction (HCI) community and thus abandoned the class hierarchy and created a new toolkit that used a monolithic class that had almost everything that an end programmer would need.
2. **Efficient**: The CEXI Toolkit is a toolkit designed to solve problems related to input devices in the HCI domain. If the toolkit is not efficient this will greatly limit the number of problems that can be solved using the toolkit. Similarly, Bederson's Piccolo toolkit is being redesigned using DirectX as the current Java implementation proved to be too slow for many novel applications.
3. **Evolvable**: The two layer approach used by the SDG Toolkit made it difficult to modify the SDG Toolkit to support different input devices. While there is no way to predict the future, usually it is possible to envision the things that are likely to change in the near future. For example, when I created the SDG Toolkit, I knew that new input devices were coming out and that they would likely need a high level API to support development.

These three lessons were gleamed during an HCI course that required me to explore several different HCI toolkits. Almost every toolkit was weak in at least one of these three categories. Some toolkits were evolvable but hard to use as they required complex code to set up and configure (e.g., COAST [7]). Others were useable but the needs of the HCI community had evolved and the toolkits were not able to support these growing needs (e.g., SDG Toolkit [8]). Finally there were useable and evolvable toolkits that were not efficient, thus the number of problems that could be solved with the toolkit were limited (e.g., Piccolo [6]).

I would rank usability as the most important aspect of an HCI toolkit. Many HCI toolkits could not be tested during our course because no one could figure out how to use them. Sometimes the tool made certain assumptions regarding the order of operations that was not immediately obvious to the end programmer. Often the cost of setting up a toolkit greatly exceeded the benefits that the toolkit provided.

Efficiency is the second most important criteria as HCI researchers are interested in developing novel interactions that push the limit of interactivity. Thus an inefficient toolkit would not be able to solve these problems. Also, the time spent optimizing one's code takes time away from the task of designing novel interactions.

Evolvability is the least important of the three criteria as the needs of HCI researchers is continually changing. While it is impossible to create a toolkit that will meet all future needs in a particular domain, one can greatly increase the longevity of a toolkit by supporting the needs of researchers in the immediate future.

The design of the CEXI Toolkit is based on these three toolkit patterns. This purpose of this paper is to provide a practical application of these patterns through a description of CEXI Toolkit implementation.

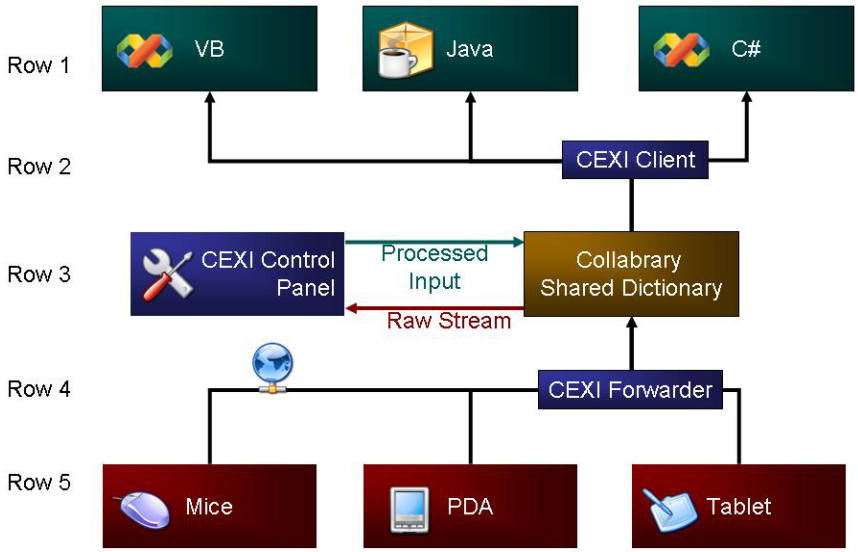
3. The CEXI Infrastructure

The infrastructure of the CEXI Toolkit is broken up into three different parts: the forwarder (Figure 2, rows 4 and 5), the control panel (row 3) and the clients (rows 1 and 2).

The CEXI Forwarder is designed to support the evolution of the CEXI Toolkit as it allows third party developers to create their own input forwarders. Data generated from input devices (row 5) can be easily sent to a centralized database (the Collabrary Shared Dictionary by Boyle and Greenberg [2002]) using a CEXI Forwarder. The forwarder is a simple to use component that is responsible for establishing a connection with the database and forwarding input. I describe how an example legacy C++ application can be modified to forward input in Section XXXX.

The Collabrary is a toolkit designed to efficiently support distributed collaboration through the use of shared key value pairs. This makes it easy to add extra event information as it is simply a matter of adding another key value pair to the dictionary. Also, the forwarder can be used to send input from a different computer over the network. For example, the mouse in Figure 2, Row 5 is forwarding input to the Collabrary Shared Dictionary over the network. To improve efficiency the CEXI Forwarder and Client use quenching to ensure that they only receive input events from the Collabrary Shared Dictionary that are important to them.

The CEXI Control Panel (Row 3) is designed to support the efficiency of CEXI applications through event queuing. Input stored in the Collabrary Shared Dictionary is then streamed to the CEXI Control Panel on the local machine. The Control Panel queues input events so that they do not overwhelm the high level application. It is also responsible for the configuration and processing of input. For example, the configuration of multiple mice orientations around a table would be configured with the



control panel on a per computer basis rather than once for every computer and application and input device (as was the case for the SDG Toolkit). The processed input is returned to a separate entry in the collabrary shared dictionary that clients would use in high level programming environments.

Finally the CEXI Client is designed to make programming multiple input devices easy. The client is consistent across all different input devices thus one can easily switch between a mouse and a DViT Smart Board since they would all be configured in the Control Panel.

4. CEXI Events

All event data is placed into an event arguments class and is sent through a CEXI event stream.

4.1 Event Arguments

ID, Device ID

Location (X, Y, Z)

Orientation (Y, P, R)

Button, Wheel, Hover, Pressure

Bounds (Size, All Points)

Finger

Extra Info

Figure 3. Elements of the Event Arguments Class

The event arguments class is designed to cover the features offered by the pointing input devices in our lab. All of these variables are 32 bit integers, thus any extra information must be in the form of an integer. This design decision was made because most devices provide integers or float values. Those that provide float values often provide a number between from 0 to 1, thus they can be easily converted into an integer with no loss of data accuracy. Doubles were not used because they would double the amount of data that would need to be transferred over the network.

Extra information can be easily added in the event argument since we are using the Collabrary Shared Dictionary. To access extra information on a client they would need to call a get extra info method on the event arguments class. Even if additional parameters are added, old versions of the forwarder, control panel and the client will still work, they will just be oblivious to the extra information. The event argument structure is designed to be evolvable while still providing backwards compatibility for old clients and forwarders.

The biggest evolvability risk of the CEXI Event Arguments class is the possibility of the needing to be change existing parameters. The parameters of Figure 3 have been carefully chosen to support all of the novel input devices we have in our lab and will have in the foreseeable future.

4.2 Events

Move (Stream)

Down, Up

Delta, Drag, Hover, Pressure, Double Down

Extra Event

Figure 4. Events Provided by the CEXI Clients

The CEXI Forwarder need only worry about handling Down, Up and Move events. All other events (e.g., Delta, Drag and Hover in Figure 4) are inferred through the CEXI Event Arguments class. If the device driver requires an additional event, they can fire an extra event by specifying an event name. There is an event on the client side called "ExtraEvent" that is a catch all for unrecognized events.

5. Using the CEXI Toolkit

To illustrate the features provided by the CEXI Toolkit, I describe its three main components: the CEXI Forwarder, Control Panel and Client.

5.1 Forwarder

Most novel input devices (e.g., Diamond Touch, DViT Smart Board, Open CV) provide SDKs to show an end programmer how to build their own applications. The problem is that almost all SDKs are written in legacy Visual Studio 6.0 C++ code. This

makes it difficult to access input from high level languages such as Visual C# and Java. The Cexi Forwarder is designed to minimize the amount of changes needed to convert an example input application into a CEXI Forwarder.

There are four steps involved with forwarding input from an existing SDK application. All of which can be easily copied and pasted into an existing application

1. **Variables**: Variables need to be added to the SDK application so that the CEXI Forwarder will remain resident in memory.
2. **Initialization**: Upon application initialization, the CEXI Forwarder would be started and any default configuration variables could be set (e.g., the name, description and the default number of events per second).
3. **Event Forwarding**: Each time there is an event, a CEXI Event Arguments class is created, all of the appropriate variables loaded and the event is fired. Extra event parameters are specified in the `setExtraInfo(string)` method of the Event Arguments class. Extra Events can be fired by calling the `FireExtraEvent(string)` method of the Forwarder class.
4. **Clean up**: When the program closes, a method is called to remove the CEXI Forwarder from memory.

This entire process can be completed in less than fifteen minutes, which is an order of magnitude less than the several weeks required to write a two layer wrapper. The goal of making the CEXI Forwarder easy to use is to encourage the development of third party device forwarders.

Also, the CEXI Forwarder separates the device driver writer from the end user API (the façade in Figure 1). This allows the device driver writer to focus on the task of obtaining input from the legacy C++ SDK rather than worrying about making a consistent, easy to use interface. With the wrapping approach used in the SDG Toolkit, each device driver provider would have try to create a similar interface for each input device, this leads to problems with consistency across different APIs.

5.2 Control Panel

The Control panel improves application efficiency by taking input from the forwarder and storing it in a circular queue. This information is processed on a timer at a user specifiable rate. For example, the P5 glove in this example was set to release 60 events per second in the forwarder. 60 events per second is the default because it matches the refresh rate of some monitors and for the most part users cannot tell the difference between processing 60 events a second versus 300 events per second.

Raw device input is often quite hard for end programmers to use. Sometimes the input is a value between 0 and 256 other times can be between negative two hundred thousand and a million. The Control Panel makes end programming easier by automatically mapping input values to sensible ranges.

For example, the X and Y coordinates are mapped to screen coordinates. Yaw, pitch and roll are mapped between 0 and 360 and all other values are mapped between 0 and 100. This is done by storing the maximum and minimum values of each input variable into an array and automatically mapping current input to the range specified by the maximum and minimum values. The

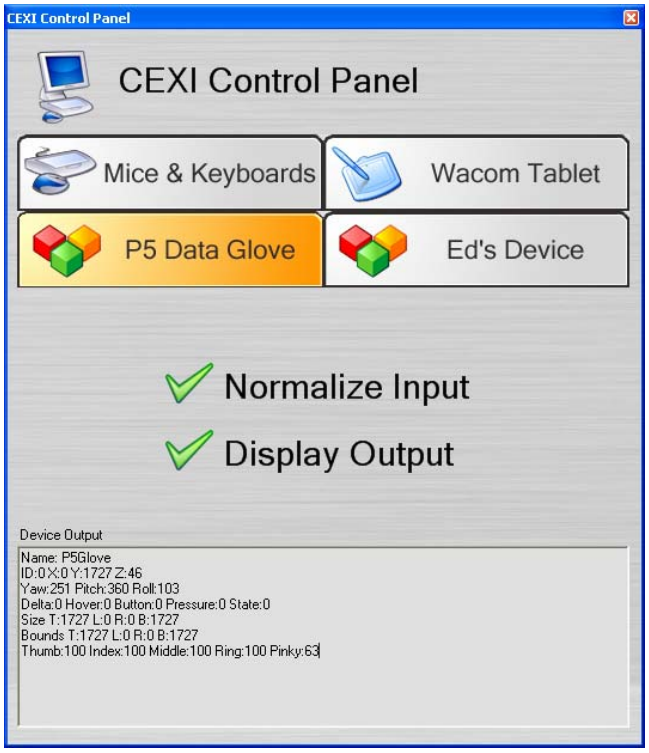


Figure 5. The CEXI Control Panel

result is input values that are meaningful to the end programmer, and easier to program across different input devices since every device is consistent.

End users expect an input device that works as smoothly and accurately as their mouse. Jitter is a significant problem for vision based input devices such as the DViT Smart Board since noise is inherent in all digital cameras. To improve the end user usability of applications, the Control Panel smoothes events to reduce jitter. For example, if the smoothing parameter of the CEXI Forwarder is set to 10, the average of the last 10 values is sent to the client rather than the most current event. This causes a noticeable difference in end user applications that expect smoothly moving input.

The CEXI Control Panel also fills in the gaps in the Event Arguments class. For example, if not values are specified for the Size and Bounding Region variables then a default size of 0 is provided for both values, this is done by creating a rectangle with the same top and bottom values and the same left and right values (see Figure 5, Bounds Variable).

Currently, the CEXI Control Panel (Figure 5) allows a user to enable/disable the normalization of input and to enable/disable the display output to the window. Device configuration is a feature that is inherent in the design of the Control Panel and will be added shortly.

5.3 Client

Clients are supported across different platforms. Currently, I support both Microsoft .NET (e.g., C#, Visual Basic) and Java.

5.3.1 C# Client

Creating an input client is simple, Figure 6 shows the code needed to add CEXI functionality to a simple windows application. This


```
private CexiClient myClient;

public Form1() {
  myClient = new CexiClient();
  myClient.Move +=new CexiEventHandler(ClientMoved);
}

private void ClientMoved(object sender, CexiEventArgs e){
this.Text = "X " + e.X + " Y " + e.Y + " Z " + e.Z +
            " Yaw " + e.Yaw + " Pitch " + e.Pitch +
            " Roll " + e.Roll;
}
```

Figure 6. A Simple C# Client Application

application changes the title bar to the current X, Y, Z, yaw, pitch and roll values. The steps

1. Using the Visual Studio Integrated Development Environment (IDE), add a reference to the CEXI C# Client.
2. A ClientClient variable needs to be added (e.g., `myClient`)
3. When the Windows Forms class is created (e.g., `Form1()`), the client must be instantiated and any event handlers need to be added.
4. The event callback (e.g., `ClientMoved`) contains an Events Argument class that can be used to access input from multiple devices.

As mentioned earlier, the Client application can access extra variables through a `getExtraInfo(string)` method where the programmer must specify the extra event that they would like to obtain and extra events are sent to `ExtraEvent` handler.

The CEXI Client is designed in a similar way to mouse events in current high level programming languages. Mouse events are designed to support five button mice with mouse wheels even though most mice only support two or three buttons. If a program is written to take advantage of the fifth mouse button this functionality is simply not available to a three button mouse but sometimes it can be simulated by clicking multiple mouse buttons simultaneously (e.g., the left and right mouse buttons). By providing a consistent API with more features than necessary, programmers can easily build applications that work regardless of what kind of mouse is used.

By providing a single monolithic event argument, end programmers can prototype applications using any supported input device without having to recompile their code. That is, the client application in Figure 6 will work with DViT just as well as it does for Multiple Mice or a P5 Data Glove. The executable can be copied onto a USB flash disk and immediately used on another computer that supports a DViT Smart Board without even needing to copy over the application source code. If the new input device does not provide yaw, pitch and roll values, a default of zero is given.

5.3.2 Java Client

To demonstrate how the CEXI Toolkit can be used across different languages, a similar java client application is shown in Figure 7. The code is not quite as simple as the C# client because an actual java client has not yet been developed. This application directly accesses the values contained in the Collabrary Shared Dictionary through a Collabrary client developed for java and encapsulated in a jar file.


```
private SharedDictionary sd;

CexiClient() {
  this.sd = new SharedDictionary();
  this.sd.subscribe("/cexiinput/?", notified);
  this.sd.open("tcp://localhost:cexi");
}

public void notified(String key, Object value) {
   Struct s = (Struct) value;
   button1.setText("X " + s.get("X") + " Y " + s.get("Y")
        + " Z " + s.get("Z") + " Yaw " + s.get("Yaw")
        + " Pitch " + s.get("Pitch")
        + " Roll " + s.get("Roll"));
}
```

Figure 7. A Simple Java Client Application

While no official client has been produced this example proves that the CEXI Toolkit can be used with clients in various high level languages.

6. Applications

The CEXI Toolkit has been applied to two applications related to the support of real time interactive 3D puppets.

6.1 Maya Puppet Application

The goal of this project was to map the input provided by the P5 Data Glove to an orally articulated puppet such as the T-rex shown on the top left corner of Figure 8. Using a P5 Data Glove provided by essential reality, I created a CEXI Forwarder from an example application provided by the SDK that came with this input device. Using the Grouplab Widget Tap library [9], I was able to create a client application that would send input events directly into a Maya script window to control the head position and jaw opening of a 3D model in real time.

It is important to note that this entire application took only two days to complete since the hard part of obtaining input from legacy C++ code was taken care of automatically by the CEXI Toolkit. Also, the fact that input variables were mapped to sensible ranges (e.g., the bend of each finger was mapped to a value between 0 and 100) made it much easier to map onto the attribute coordinates needed in the Maya Application. While this application sounds complicated its implementation was rather simple given that the complicated parts of the implementation were handled by the CEXI Toolkit.

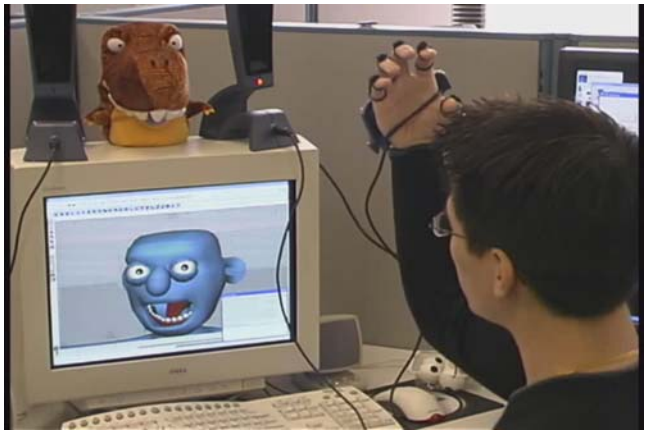


Figure 8. The Maya Puppet Application

6.2 Networked Maya Puppets

Since I wanted to support multiple simultaneous 3D puppets, I modified the CEXI Forwarder to automatically send input events to another computer (this is simply the addition of one variable in the `Start()` method of the CEXI Forwarder). Next, I set up the client application to send input events to multiple puppets in Maya. This produced tool that could be used to manipulate multiple puppets simultaneously in real time.

Again, while the implementation of this application sounds complex, the CEXI Toolkit made the addition of a second input device through a networked computer a trivial task.

Figure 9. Networked Maya Puppets

7. Conclusions

"Toolkits are cool and fun, but I'd caution developers of them because it is exceedingly difficult to create a broad reputation as a researcher by building them. They are very hard to publish about, and the # of publications to hours put in is very low compared to other kinds of research."

–Benjamin Bederson, developer of the Piccolo Toolkit

The fundamental argument of this paper is that the HCI community is stinting its own growth by rejecting toolkit papers that build upon existing contributions. Toolkits are more than just engineering, they are the foundation of advancement in the field of HCI research. They summarize and build upon the contributions of other researchers so that we do not have to constantly reinvent the wheel when building research prototypes. The HCI community seems to value papers with attractive images (e.g., gratuitous images of red Ferraris) and little contribution over toolkit papers. The result is that Toolkit developers are discouraged from building toolkits and thus HCI research always remains at the breakthrough and replication stages. Toolkits will always remain unusable if there is no benefit to spending extra time to make a good toolkit.

The primary contribution of this paper is the presentation of three patterns for developing toolkits. These patterns are designed to help toolkit developers create toolkits are effective for the rest of the HCI community. These patterns were applied in the creation of the CEXI Toolkit. I showed how the CEXI Toolkit worked through example and illustration. The CEXI acronym is a pun on the word sexy as the only thing lacking in toolkit papers is sex appeal.

8. ACKNOWLEDGMENTS

This project would not have been possible without the gracious help of Michael Boyle. In addition to creating the Collabrary, he helped with much of the debugging and idea forming of this toolkit.

Thanks also goes out to Ben Bederson for being in contact with me regarding his thoughts about toolkit design.

9. REFERENCES

[1] Dietz, P. and Leigh, D., DiamondTouch: A multi-user touch technology, *Proceedings of the ACM Conference on User Interface and Software Technologies (UIST '01)*, Orlando, pp. 219-266, 2001

[2] Diaz-Marino, R.A., Tse, E, and Greenberg, S. (2003) **Programming for Multiple Touches and Multiple Users: A Toolkit for the DiamondTouch Hardware.** Companion Proceedings of ACM UIST'03 Conference on User Interface Software and Technology.

[3] H. Sneed, Encapsulation of legacy software: A technique for reusing legacy software components. Annals of Software Engineering 9, 2000.

[4] J. Purtilo, The POLYLITH Software Bus. ACM Transactions on Programming Languages and Systems 16(1), 1994.

[5] Boyle, M. and Greenberg, S. (2002), GroupLab Collabrary: A Toolkit for Multimedia Groupware, Extended Abstracts of the ACM Conference on Computer Support Cooperative Work (CSCW '02), Workshop on Network Services for Groupware, New Orleans.

[6] Bederson, B., Grosjean, J., and Meyer, J. Toolkit design for Interactive Structured Graphics. In *IEEE Transactions on Software Engineering*, Vol 30, No. 8, 535-546.

[7] Schuckmann, C., Kirchner, L., Shummer, J., Haake, J., Designing Object Oriented Synchronous Groupware with COAST, CSCW, 96

[8] Tse, E. The Single Display Groupware Toolkit, MSc Thesis, 2004, University of Calgary, Alberta Canada.

[9] Greenberg, S. and Boyle, M. (2002) Customizable physical interfaces for interacting with conventional applications. Proceedings of the UIST 2002 15th Annual ACM Symposium on User Interface Software and Technology, 31-40, ACM Press.