Using Aspects to Convert Single User Applications into Multiple User Applications

Edward Tse
University of Calgary
2500 University Dr. N.W.
Calgary, Alberta, Canada
(403) 210-9501

tsee@cpsc.ucalgary.ca

ABSTRACT
This paper details the process of converting a single user application into a multiple user application through the use of Aspect Oriented Programming (AOP). While AOP hopes to enable developers to capture crosscutting concerns (e.g., features that affect different classes and modules of source code) my goal is to treat multiple user functionality as a cross cutting concern that should be easily added to a single user application.

This primary contribution of this paper is the detailing of the issues encountered in the exercise of trying to apply aspects to existing single user applications. Through a detailed analysis of the issues encountered there is the potential to refine the design of current and future Aspect Oriented Tools.

Categories and Subject Descriptors
D.2.10 [**Design**]: Software Engineering Design Methodologies.

General Terms
Human Factors, Software Architecture

Keywords
Single Display Groupware, Aspect Oriented Programming, Software Architecture Design

1. Introduction
Conventional programming environments employ a design paradigm known as object oriented development. The main concept of object oriented development is that everything is modeled within modules (e.g., a class or a collection of classes) that encompass certain components of functionality within an application or tool [1]. The problem is that the concept of separating functionality into clearly defined modules with limited cross module communication (i.e., coupling) sounds good in theory but rarely occurs in reality.

Often a particular function is needed in numerous locations within the source base. For example adding a logging function within a program would require a programmer to manually add a `FunctionEntered()` and `FunctionExited()` method for every public function within every class of an application. This is tedious and prone to error if done on large programs.

Aspects allow logging functionality to be added programmatically to source code at compile time or at run time. This significantly reduces the effort involved and reduces the chances of programmer error.

Researchers in Single Display Groupware (SDG) explore how multiple users share a single display such as a computer monitor, a large wall display, or an electronic tabletop display. Yet today's personal computers are designed with the assumption that one person interacts with the display at a time. Thus researchers and programmers face considerable hurdles if they wish to develop SDG. As part of my MSc Thesis topic [3], I developed the SDG Toolkit, a software library that allowed multiple user applications to be developed using multiple mice and multiple keyboards. The SDG Toolkit provided a simple to use API that presented SDG input in a similar fashion to mouse events except that an additional ID parameter was added to specify the input device (e.g., Mouse 1).

1.1 Problem
While the SDG Toolkit simplifies the development of SDG applications from scratch, the task of converting existing single user applications into a multiple user applications is difficult and prone to error. The problem is that it requires programmer time and effort to learn how to use the SDG Toolkit. A lack of SDG Toolkit understanding leads to errors and frustration on the part of the SDG programmer.

This paper attempts to introduce SDG functionality as an aspect so that SDG functionality can be used with minimal learning of the SDG Toolkit.

2. Aspects
To illustrate the features and limitations of aspects I provide a brief example through a logging example illustration for those familiar with Microsoft .NET [2]. While there are numerous implementations of Aspects, .NET was used because this is the environment that the SDG Toolkit is designed for.

Using the open source utility Aspect Sharp, one could create a console logging utility using two chunks of code as seen in Figure 1. First, an aspect must be declared. In this example we created a sample (line 3) that contains a pointcut to all methods (line 5). A pointcut is like a syntactic search on the source code base, where every method is searched for the opening and closing braces. The pointcut is set to call the advice called LoggerAspect (line 6), which is an aspect that contains logging functionality.

If we examine the LoggerAspect Implementation we see that it overrides a method called Invoke (line 12). This function is invoked each time a method is called since the pointcut was declared for every method (line 5). Before a method invocation (line 15), the method is written to the screen (line 14) and after

Tse, Edward (2005) Using Aspects to Convert Single User Applications to Multiple User Applications. Report 2005-785-16, Department of Computer Science, University of Calgary, Calgary, Alberta CANADA T2N 1N4. April.

```
1  //Aspect Declaration
2  import HelloWorldApplication
3  aspect sample for [ MyNameSpace ]
4    include StandardMixin
5    pointcut method(*)
6      advice(LoggerAspect)
7    end
8  end
9  //Advice Implementation
10 public class LoggerAspect : IMethodInterceptor
11 {
12  public object Invoke(IMethodInvocation invocation)
13  {
14    Console.WriteLine("Entering " + invocation.Method);
15    object retVal = invocation.Proceed();
16    Console.WriteLine("Exiting " + invocation.Method);
17    return retVal;
18  }
19 }
20 [Sample Output]
21 Entering Void MyFunction()
22 Exiting Void MyFunction()
```

Figure 1. A Simple Logger

each method invocation the exiting method is printed to the screen (line 16).

When this aspect is run on a simple application with a single method called MyFunction the output seen in lines 21-22 is shown.

Logging is a very standard sample example that introduces several basic concepts of Aspect Oriented Programming. Current AOP environments focus on the syntactic elements of source code. That is, they do not have any semantic understanding of source code, such as if a program is written with the assumption of a single user or not.

3. The SDG Toolkit

For the purposes of this paper the core SDG Toolkit [3] functionality that I am hoping to support in existing single user applications is the following for multiple mice only:

1. ***Multiple input and identification:*** SDG Toolkit provides a convenient way to gain and uniquely identify the multiple input streams from mice and keyboards.
2. ***Multiple cursors:*** Since almost all SDG applications require multiple cursors, the SDG Toolkit provides a cursor for each attached mouse.
3. ***Table orientation:*** Since developers face considerable hurdles circumventing orientation issues that occur when end users are seated at different sides of the table, e.g., how the cursor appears, how the mouse behaves, how coordinates are handled, the SDG Toolkit provides orientation functionality for a table top.

The use of the SDG Toolkit is best illustrated through example. Figure 2 shows a basic drawing application that uses multiple cursors. The first step is to add an SDG Manager component (using the visual designer) and set it's RelativeTo property to the form (line 2) so that all coordinates are provided relative to the form as this is the norm expected by most programming environments. Next, a mouse event handler is added to the mouse movement event (line 3) for our drawing application. The mouse event handler (line 6) is almost identical to the normal windows mouse event handler except that an `SdgMouseEventArgs` class containing an `e.ID` parameter is used instead of a `MouseEventArgs` class.

To draw a diagonal line onto the screen we create a graphics object to draw to our form (line 7). Next, we set the colour of the pen (which we are going to use for our drawing) to Black (line 8), if the e.ID parameter is greater than zero we set its colour to Red (line 9). Finally we check to see if the left mouse button is pressed (line 10), if it is then we draw a diagonal line on the screen (line 11). The final result application is shown on the left side of Figure 2.

3.1 Converting a Single User Application

If one wanted to a single user application into a multiple user application they would need to perform the following three steps:

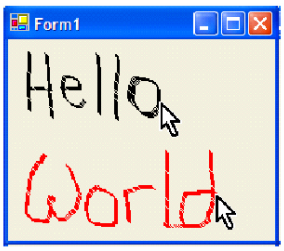
1. Add an SDG Manager Component to the form. This involves creating an SDG Manager Class and adding it to the component container class. For example:

```
this.Mgr1 = new SdgManager(this.components);
```

2. Set the coordinates to be Relative to the form application (e.g., Figure 2, line 2).
3. Removal all existing MouseMove event handlers and replace them with event handlers in the SDG Toolkit (e.g., Figure 2, line 3).
4. Modify all existing MouseMove even callbacks and replace them with callbacks compatible with the SDG Toolkit (e.g., Figure 2, line 3).
5. Manually add code to take advantage of the `e.ID` parameter (e.g., Figure 2, line 9). This is a highly cognitive task that requires a programmer to understand the single user assumptions made in their application and modify them so that they support the simultaneous activities of multiple users.

4. Hypothesis

The hypothesis of this paper is that it would be possible to use aspect oriented programming to perform the first four tasks in




```
1  private void InitializeComponent () {
2    Form1.sdgMgr.RelativeTo = this;  //'this' refers to the top level window form
3    this.sdgMgr.MouseMove +=new SdgMouseEventHandler(this.sdgMgr_MouseMove);
4     …
5  }
6  private void sdgMgr_MouseMove(object sender, SdgMouseEventArgs e) {
7    Graphics g = this.CreateGraphics();
8    Pen penColour = Pens.Black;
9    if (e.ID > 0) penColour = Pens.Red;
10   if ((e.Button & MouseButtons.Left) > 0)
11     g.DrawLine(penColour, new Point(e.X-1, e.Y-1),new Point(e.X+1, e.Y+1));
12 }
```

Figure 2. A Basic Hello World Example using the SDG Toolkit

converting a single user application (Section 3.1).

This hypothesis was ultimately not achieved but the discussion of the problems encountered may inform the design of future Aspect Oriented Programs.

5. Methodology

While Aspect Sharp was the platform that was ultimately tested, two other AOP tools were tested: AspectC# and AspectJ [5].

Aspect J is a commercial AOP tool for Java that was used as a learning exercise for understanding how Aspects work. It includes a number of useful online video tutorials that provide step by step instructions on using the AspectJ environment in Java (http://eclipse.org/aspectj/).

Aspect C# is a AOP implementation for C# that was done as a Masters thesis at the University of Dublin in 2002 [1]. Aspect C# has an architecture that is almost identical to AspectSharp except that Aspect Sharp provides Aspectual Polymorphism. That is, Aspects can be easily turned on and off in Aspect Sharp.

When using AspectC# I noticed another implementation difference, AspectC# has a separate compiler for combining C# code, thus the Microsoft Visual Studio Integrated Development Environment (IDE) cannot be used for compiling code (since there is no option to switch compilers in Microsoft .NET). AspectSharp is implemented as a number of libraries that can be added to an existing C# application. Aspect Sharp was used because results could be rapidly viewed in the Microsoft Visual Studio IDE. The source code for AspectSharp was available so that I could potentially modify AspectSharp to suit my needs.

6. Results

After building the basic logging application in Figure 1, I attempted to encapsulate SDG Toolkit functionality as an aspect or a collection of aspects since each aspect could only contain one method interceptor. I will describe the first four steps of converting a single user application with AspectSharp.

First, we need to add an Sdg Manager to our form. Recall from Section 3.1:

```
this.sdgMgr1 = new SdgManager(this.components);
```

The Microsoft Visual Studio IDE will fail to compile this Aspect code since `this.components` is not a member of the SdgAspect. The invocation class (e.g., Figure 1, line 15) provided a This property that I could cast as a windows Form as seen below:

```
Form form1 = invocation.This as Form;
```

This approach fails as `components` is a private member variable within the Microsoft .NET Form class. Thus it is not possible to access this hidden member variable even if I have a reference to the originating method class.

Second, to make all coordinates Relative to the application we need to add the following line of code.

```
this.sdgMgr1.RelativeTo = this;
```

We can use the `invocation.This` property to place a form in the location of `this`. Thus there are no issues preventing the implementation this particular dependency.

Third, to removal all existing MouseMove event handlers we require a pointcut that allows one to remove event handlers. It is possible to remove an event handler by simply deleting all code within the event handler, that is one could choose not to call the `invocation.Proceed()` method and the body of the function would never be called. The problem with this approach is that we still want do not want to remove the callback method but rather the addition of the event handler callback (like Figure 2, line 3).

Fourth, to modify existing Mouse Move callbacks we need a pointcut that allows one to modify method signatures. The four pointcut types provided in AspectSharp are method, property, property read, and property write. Unfortunately, there is no facility provided to modify the method signature of a pointcut in AspectSharp.

7. Practical Considerations

The SDG Toolkit could be modified to remove dependency on the private `components` member variable. This dependency was added to provide end programmers with the ability to add the SDG Manager in the Visual Studio Form Designer. That is, one could add an SDG Manager to a form just like they would add a timer and easily modify its properties and add event handlers.

AspectSharp could be modified to support the removal of event handler invocations. This would involve a pointcut that would capture event signatures (e.g., MouseMove).

To support the modification of method signatures (e.g., the Mouse Move event callback in Figure 2, line 6) one could add a write capability to the method signature. For example

```
invocation.Method = "SdgMouseMove(object sender,
                    SdgEventArgs e)";
```

Even if all of these functions were provided we would encounter problems in step 5 as use of the `e.ID` parameter would cause compiler errors since it is not declared in the conventional Mouse Event arguments class.

8. Using SDG Aspects

If we assume that all the technical problems preventing the SDG Toolkit from being implemented as an aspect were resolved we would require the following steps to convert a single user application into a multiple user application.

1. Add five library references to your current application (4 for AspectSharp and 1 for the Sdg Toolkit).
2. Import the appropriate SDG Toolkit Aspect classes so that SDG functionality could be added.
3. Modify the applications Main() function to load the Aspect Engine and appropriate aspects.
4. Create an interface that contained all of the core application functionality as the application class would need to be wrapped at run time. This can be a rather involved process as the end programmer must manually type in all of the functions that will be affected by the aspect (e.g., MouseMove, MouseDown).
5. Manually add code to take advantage of the `e.ID` parameter as before.

The time spent getting started (steps 1-4) with the SDG Toolkit is minimal compared to the time spent modifying source code to take advantage of the multiple user functionality of the SDG Toolkit (step 5).

9. Conclusion

Converting single user applications into multiple user applications is not a simple task. This paper showed some of the practical limitations of using AspectSharp to add SDG Toolkit functionality to existing Single User Applications. Clearly, this is something that AspectSharp was not intended to support. The time required to get AspectSharp to work would exceed the time needed to add an SDG Manager to a form.

Even after the technical issues of using aspects in Microsoft .NET are resolved there are issues related to compiling code with SDG functionality. It would be easier for an end programmer to understand the effects of an aspect if the aspect code appeared in the source code browser.

This paper was not designed to point out the weaknesses of AspectSharp. It was also not designed to say that aspects are not useful. Rather, the purpose is to show the potential of Aspects in aiding the conversion of single user applications into multiple user applications and to suggest how aspects could be adjusted to meet the needs of SDG developer.

10. Future Work

The future work of this paper is best described through an example single user application that was later converted into a multiple user application.

Rush Hour is a simple online puzzle game, where the player must move cars around until they can get the special red car to the red exit marker (Figure 3). I decided to implement an SDG version of this game, where multiple players can move multiple cars simultaneously. First, I implemented a single user version of this game using the standard features of C# and .NET. Second, I modified this game to add multiple user capability via the SDGToolkit.

Collision detection was implemented by treating the puzzle board as a two dimensional array of numbers. If a car was positioned over a square, its corresponding value would be 1 in the array, otherwise it would be 0. When a car was dragged, the vehicle was removed from the collision detection array and the application would check to see if the vehicle was collided into the space of another vehicle. If it did then, movement would not be restricted.

Problems arose when multiple cars were moved simultaneously. They would both be removed from the collision detection array and we would often observe two vehicles moving through each other. The reason for this problem was that our collision detection array was designed with the assumption that there would only be one car moving at any given time. To resolve this problem, the ID of the car was placed into the collision detection array (instead of 1 and 0) so vehicles could be moved and the values of the current card ID would be ignored.

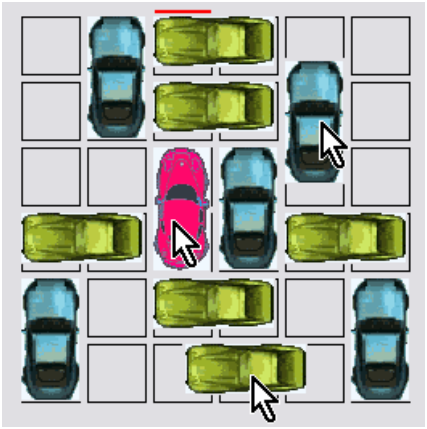


Figure 3. The SDG Rush Hour Application

This example introduces some of the complexities involved with converting single user applications into SDG applications. While a syntactic tool such as Aspect Sharp could aid SDG development, a greater benefit could be accrued by using a semantic level tool that could point out the single user assumptions in existing applications.

11. ACKNOWLEDGMENTS

I would like to thank Robert Walker for all of his insights about Aspect Oriented Programming. I would have never thought to explore the concept of combining Aspects and SDG Toolkit functionality if he had not pointed me to it.

12. REFERENCES

[1] Kim, H. Aspect C#: An AOSD implementation for C#, MSc Thesis, University of Dublin, 2002

[2] Microsoft .NET, http://msdn.microsoft.com/netframework, 2005

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

[4] AspectSharp, http://aspectsharp.sourceforge.net/

[5] AspectJ, http://eclipse.org/aspectj/