UIDE (User Interface Development Environments)

A UIDE main goal is to make it easier for the programmer to develop the user interface. Creating the graphics behind a user interface is complicated but is mostly handled by the operating system. The UIDE is responsible for making the interaction with the controls exposed by the operating system reliable, understandable, and interactive. Since there are so many different UIDE's that have been produced and are currently being used there is no way to give a full review of them all. Instead the main areas of the graphical interface will be explored and possible solutions that can help for those areas will be reviewed. The main model of the graphical user interface is the MVC (Model View Controller) each of the three sections will be explored.

View

Creating an automated graphical view was the first thing that was done in the evolution of the UIDE with the creation of ResEdit for the MacOS (Foley et al, 1991). Enabling the creation of the view through the use of direct manipulation is the action that is most often associated with the UIDE since it is what most developers use an UIDE for.

With the creation of the ability of quickly creating the view it has enabled developers to start using the development environment to create a prototype of the program. It can be something that is never meant to be part of the final product. This can cause problems. People that are only familiar with the prototype during the software life cycle might think that the quickly created view shows a more complete version of the software then really exists. Since the view looks done and that is the part of the program that people interact with they will assume that the rest of the application is in the same state.

With the UIDE creating a visual view in the machine code there can be conflicts with the developers work. Since the program is creating the view it writes the code that is bound to that visual view. The ability of a machine to create human readable code is not a trivial task and is a problem that must be dealt with if the developer is able to interact with the code that the UIDE creates.

When the developer modifies the code that the UIDE creates for the visual view then the UIDE must be able to parse that code to modify the view based on the information. Parsing textual information to alter a visual object is not easily done as pointed out by the existence of this the user interface programming class. Standardizing how different UIDEs parse the text given to it after it is near impossible.

To illustrate the creation of the view using a UIDE I used .Net with Visual Studio 2003. I was able to easily create different views ranging from simple to complex. However with complex views I needed to have access to the code that was created by .Net. .Net offers access to the code that is created but with the warning it is “Windows Form Designer generated code”. Windows does not try to sync the code created by the developer. Anytime the “design” view is changed this area of the code is written over and any changes are lost. This forces the developer to create the basic layout using .Net and then abandon using it again with the knowledge that changing anything using the “design” view will erase all developer entered work. There is no way to disable the “design” view in .Net so the need for a code repository is very high.

Even with .Net creating the code for the view it did not guarantee that the same view for other programs. Mono is open source software that can read the interpreted language created by .Net. Microsoft supports the Mono project but when running the code from .Net there were minor differences between what was shown under Windows and using Mono.

For practice with debugging the graphical view I used jSpy. It is a java program that can read the code created for a Java program as it runs. By clicking on the different areas of the program it shows the pertinent parts of the code and hierarchal information. It would be helpful in the case of trying to find which panel has been put where by the window manager to make sure they will all react correctly to window events such as resizing.

Controller

The controller was first automated by NeXT with the creation of Interface Builder. Up to this point Interface Builder is really the only UIDE created that allows to visually link the controller to an action. I was not able to find any other applications that do anything similar or information as to why. The only reason that I can guess is because of some kind of patent keeping others from following.

The way that Interface Builder links the controller is through a visual line. When the mouse is dragged from the object to the controller it creates a link between the two. When the link is created a window pops up with the links to the project code. The operation is very easy to understand and linking already created actions can be done with out ever opening Xcode.

Problems arise when trying to do something out of the ordinary. While it is easy to add new code, modifying existing code can cause problems. I experienced a spatial disconnect between using Xcode and then finding the same stub of code in Interface builder. Even if they had both been in the same application it would have been easier. For large applications the stubs of code controlling actions can range across many files. With Interface builder it only tries to show code having to do with actions so it picks and chooses what to show the developer. Any spatial sense the developer has for the code in Xcode is lost when Interface Builder reformats it. I really don't see a way around this though since otherwise there would be too much un-needed information on the screen.

Another problem arises when trying to test the code. The problem is the fact that there is no code. The visual link between the object and the controller really is the only way to see the information. Interface builder does not have to face the problem of trying to interpret code from the user or to translate its actions into something readable. There is simply no way to access the links outside of the visual representation which, as expected for large projects, can get messy.

Model

The model of the program is usually seen as where the data is stored. Others think of it as the part of the program that exists outside of the user interface. Both ideas convey the concept that this is something that exists for all applications. The ability to visually see what is happening to the data as the program runs is something that would be useful for most applications. This is probably why there are the most applications compared to the view or controller that are devoted to allowing the developer to view this. Most of the time this is seen as visually debugging the program.

Being able to see the memory changes in a program is a lofty goal. Most of the time that an application has need of tracking its data it has grown large enough that trying to represent all the data and changes to that data at the same speed that the application is running at is either impossible or would be such an over load of information that it would not be useful.

Another problem is that outside of graphs there are a lot of algorithms that do not have an easy way to visualize what is happening to the algorithm. This is simply a complexity problem. At some point in the future people should find a way that the human mind can understand larger amounts of data; most likely with the help of more processing power that will be required to display that information.

For the evaluation for this presentation I used ddd, which is most likely the most popular program for graphically viewing the memory usage by an application. To keep the amount of information being presented to a minimum the variables to be tracked must be selected and break points put in to stop the program at specific points for viewing.

Visual Programming

There are a lot of software projects that are trying to side step the problem of translating back and forth between the machine-generated code created and code created by the developer. The way they are doing this is through the use of visual programming. The concept is that, unlike normal programming languages, visual programming languages syntax is conveyed through form and images.

These types of languages are usually thought of for helping those who are not familiar with the constructs of computer science to be able to program such as in the case of teaching children to program using Alice. However there are many other uses. They can be anything from automation programs that are linking other chunks of code together or professional programs that are very visually based so they are easier to understand through the use of pictures.

To practice visual programming I used Automator for the non-professional side and Quartz composer for the professional program. Automator allows linking small bits of already created code to each other allowing multiple application to be scripted complete with joined output without needing to type anything. It works good for easy tasks but will fall apart as expected in situations where the needed code does not already exist. Quartz composer is used to create graphical output of three-dimensional objects. All information can be linked to each other. It was easy to understand what was going on in the program, which is something unusual for the three-dimensional programs. As far as I used it, it appeared to be easy to learn. For more difficult tasks there might be limits reached but I could not find them in the short time using it.

Conclusion

One problem that is always present with the display of visual information is accessibility. It is hard to translate information back and forth between visual and non-visual and those that do not have sight would have a hard time understanding a lot of what was discussed. In some cases such as Interface builder's controller links there is no way.

The constant theme throughout has been the difficulty in trying to keep the information that is computer generated and what the developer is inputting to stay synchronized. There is no easy way to do this as was shown and this problem has not been fully solved but there are many different ideas and they can all help a developer to create a user interface faster and easier with a UIDE.

References:

J. Foley, W. Kim, S. Kovacevic and K. Murray. UIDE - An Intelligent User Interface Design Environment. In J. Sullivan and S. Tyler (eds.), Architectures for Intelligent User Interfaces: Elements and Prototypes, Addison-Wesley, 1991.

Visual Programming information:
http://en.wikipedia.org/wiki/Visual_programming

Interface Builder:
http://developer.apple.com/documentation/DeveloperTools/InterfaceBuilder-date.html

Automator:
http://www.apple.com/macosx/features/automator/

Quartz Composer:
http://developer.apple.com/documentation/GraphicsImaging/Conceptual/QuartzComposer/index.html#//apple_ref/doc/uid/TP40001357

ddd:
http://linuxgazette.net/issue73/mauerer.html

jSpy:
http://www.forwardlab.com/download.htm

.Net:
http://msdn2.microsoft.com/en-us/library/ms950416.aspx

Mono:
http://www.mono-project.com/Mono:OSX