Adaptable User Interface Design

Wesley Pierce, CLA

Founder and Chief Engineer

Pierce Controls, LLC

PIERCE CONTROLS

Expertise

- System Design
- System Integration
- LabVIEW Development
- Turnkey Solutions

Industries Served

- Manufacturing
- Automotive Test
- Electronics Assembly
- Oil & Gas Exploration
- University Research Programs

Adaptable UI Design

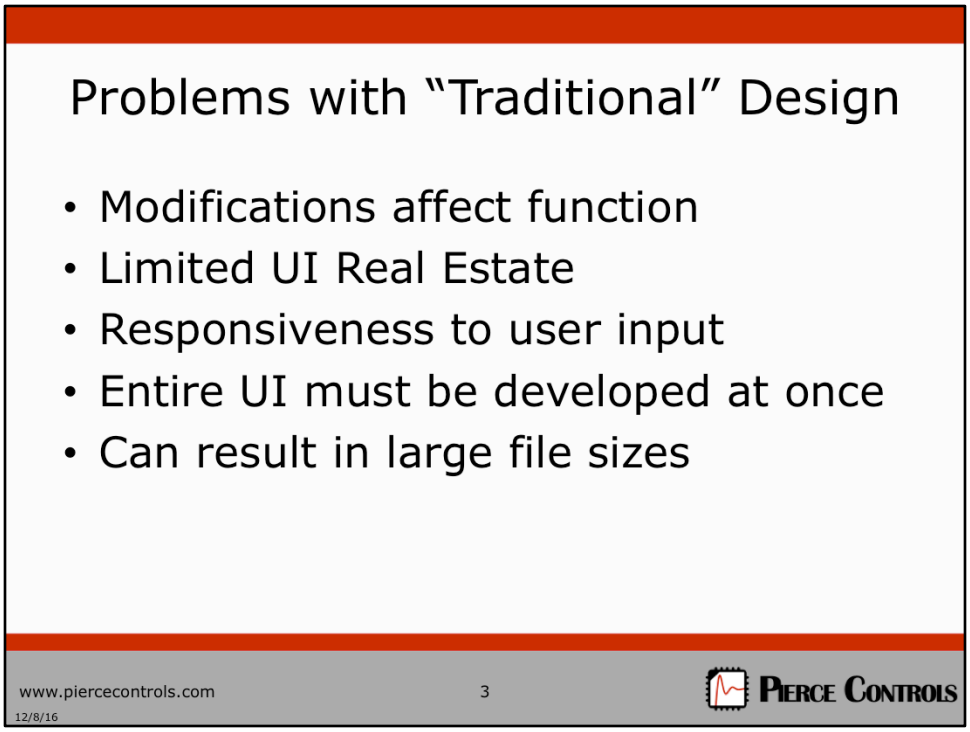
- Using Subpanels
- Detachable UI Design
- Other UI Approaches
  - TCP & UDP
  - Web App
  - Shared Variable API

What is Adaptable UI design?

On the whole, it is a philosophy of creating user interfaces that are not very tightly coupled with the code that they are interfacing with. The goal is to make UI design simpler, and make it easier to accomplish what you really want to do with your UI design.

In this presentation, we are focusing on only a couple of practical methods, but the ideas contained herein should apply throughout.

We will look at methods and examples related to utilizing subpanels, then we'll take a look at an approach for Detachable UI Design, and we'll wrap up with noting a few other UI strategies



Traditional design is that where you have a VI, and the block diagram is the code, and the FP is the UI. This works fine for small applications, but you can quickly encounter problems with larger applications, particularly where there are many areas of responsibility in the code.

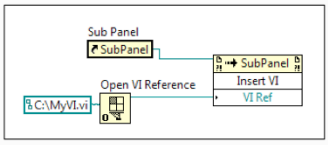
Problems that exist with this method are:

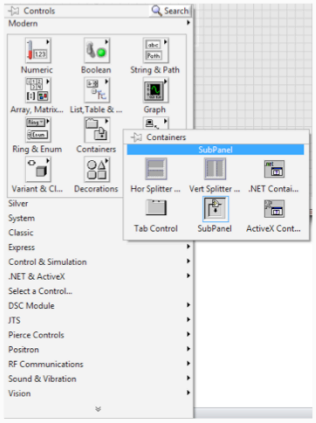
- Modifications affect function – since the code and UI are one and the same, you risk affecting behavior of the application if you need to perform a UI function, such as locking or unlocking a control, or clearing a chart.
- Limited UI Real Estate – If you lay out your entire UI on one space, you can run out of room quickly. Often, there is information displayed that only needs to be displayed sometimes….doesn't need to be there all the time.
- Responsiveness to user input – this can be mitigated with good design even in a single VI, but still, the more processing responsibility contained in a single VI, the more the UI will be interfering with that.
- Entire UI must be developed at once – with this approach, you cannot modify sections of your code later, such as a configuration window, or an IO display.
- Can result in large file sizes – this is more about modular design than merely the UI, but it is a relevant point. Denso VI.

Subpanels!

Remedies by Subpanel

- Expand FP real estate
- Decide when a user can interact with certain content
- Use with plug-in architectures



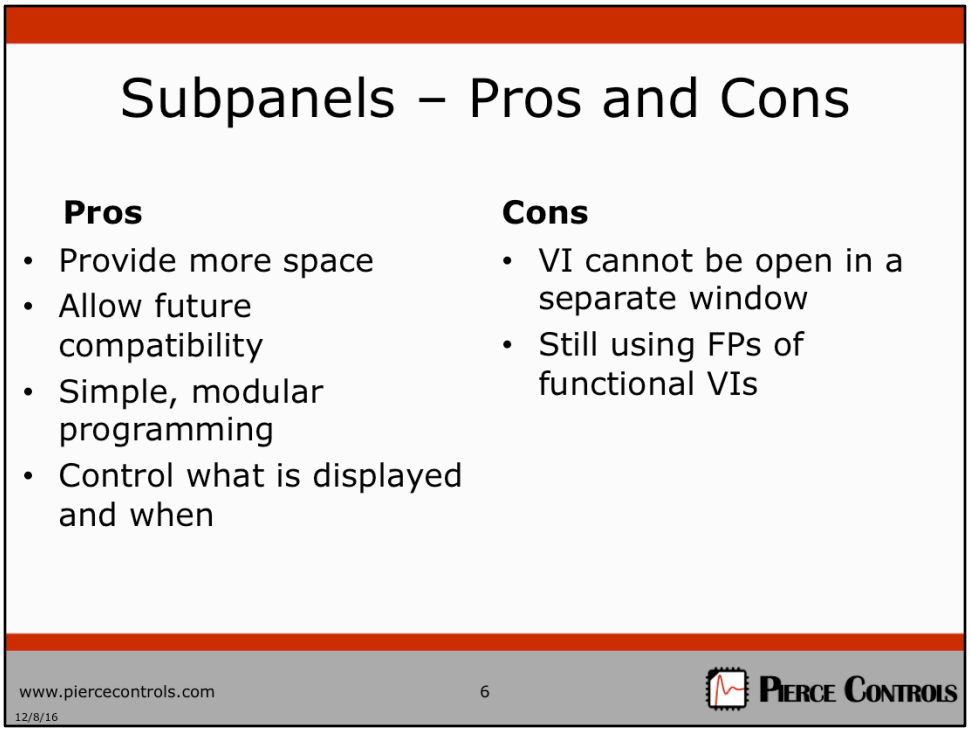


Subpanels are a container that you can place on a FP. It comes with it's own VI Server reference that you can interact with via property nodes and invoke nodes.

It is a way of running a second VI's FP inside the FP of the first VI. You can programmatically select which VI you would like to place in the subpanel by executing code in the Main VI.

Subpanels – How They Work

- Design the main UI with a fixed space
- Create several module VIs
- Create a method for loading/ unloading a VI from the subpanel

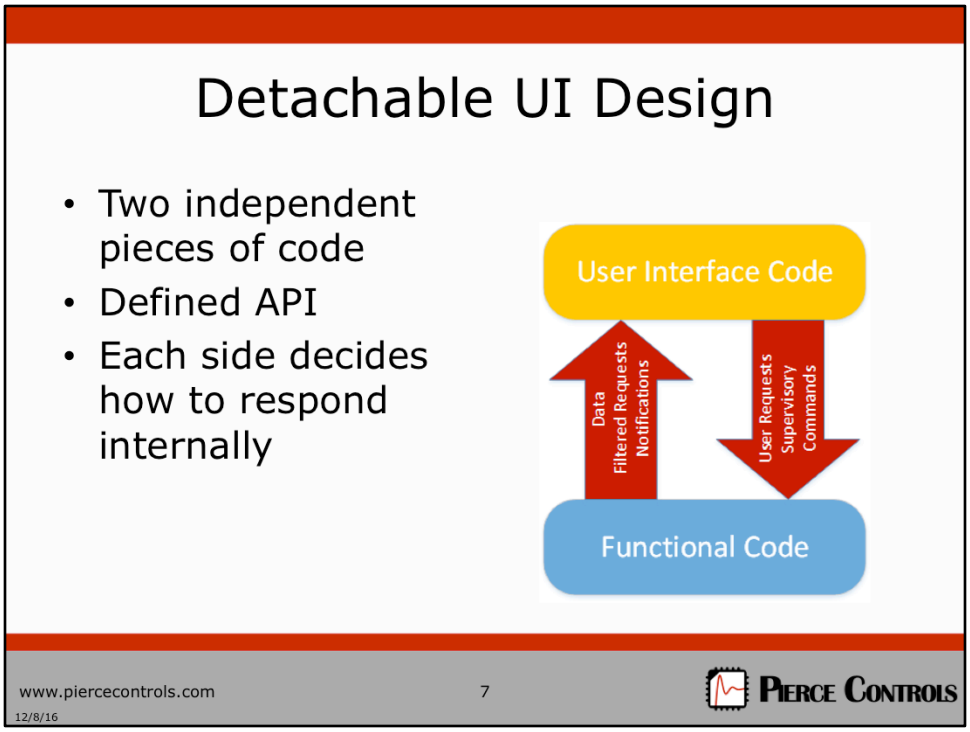


Looking back at this slide from earlier, we see that subpanels do help to address some of those concerns. Namely, real estate, progressive/plugin UI dev.

There are a few things that it doesn't help with either:

We did create a higher level UI that calls lower level code modules, but modifications to those UI elements still can affection function.
We didn't really address the issue of responsiveness because code is not split out.

We still have the problem of communicating between all of these subpanel VIs, and between the main UI. Is there another way?



- Create functional code
- Create communication method
- Build UI
- Communicate with functional code

The idea is that the functional code exposes a way to communicate with it. The UI is then built on top of that. The UI code uses the API exposed by the functional code to pass along user inputs. The UI does not pass on code responsibilities like control property nodes, locking/unlocking controls, basically anything UI related and not data related. If the functional code needs to tell the UI how to respond to something, the correct approach is to create a message that explains the situation, and let the UI decide how to handle that message.

Detachable UI Design

Remedies

- Dedicated functional code
- Dedicated UI code
- Multiple UIs
  - Different user classes
  - Different platforms
- Simple UI Updates

Dedicated functional code does not have to worry about dealing with user interface operations

Dedicated UI code does not get hung up by operations be performed.

This allows for multiple user interfaces to be used, or even dropped in.

UI updates only affect the UI.

What you Need

User Interface

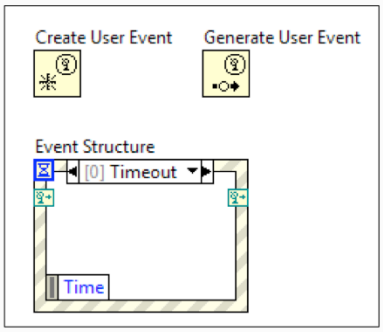
- Ability to send messages to code
- Ability to listen to responses

Functional Code

- Ability to broadcast information
- Ability to process requests from UI
- Expose API for UI to use to send requests

Implementation – User Events

- Send message
- Receive message



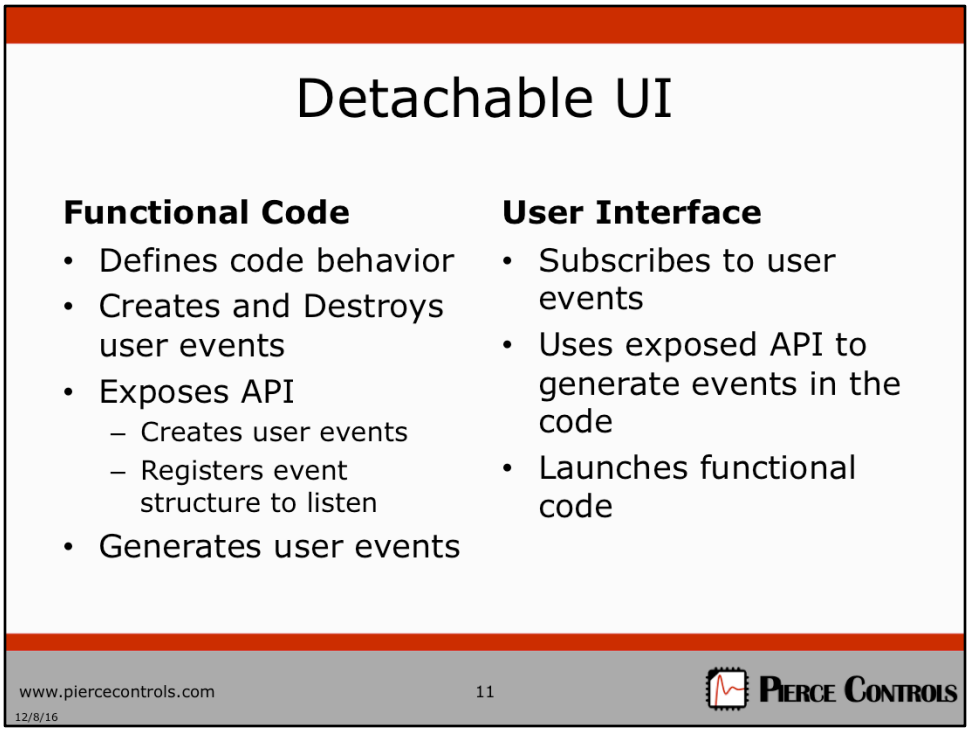
This is one implementation that is moderately easy to use. The principle can be used in a number of different cases which I will touch on at the end.

The main mechanism for sending messages will be user events.

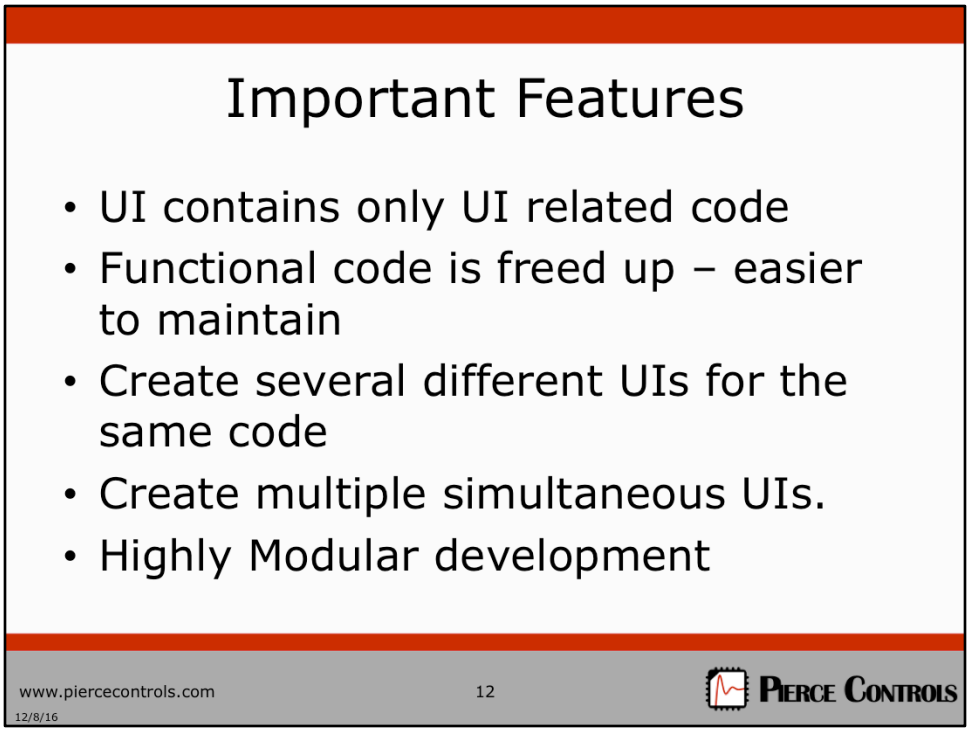
The main mechanism for receiving messages will be event structure.

The main idea is that each side has an event structure that is registered to certain events.

Benefit of user events is that they are able to send data.



The concept to focus on here is public/private events. Functional code defines the function and communication interface, and the UI just uses what the functional code has defined. This allows the functional code to define how any external code is going to interact with it.

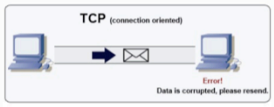


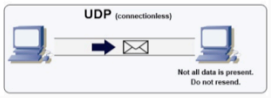
Modularity – code knows nothing of external UI. It only knows of itself.

Beyond User Events

The same approach can be accomplished using:

- TCP Server/Listener
- UDP
- Web Apps (REST)





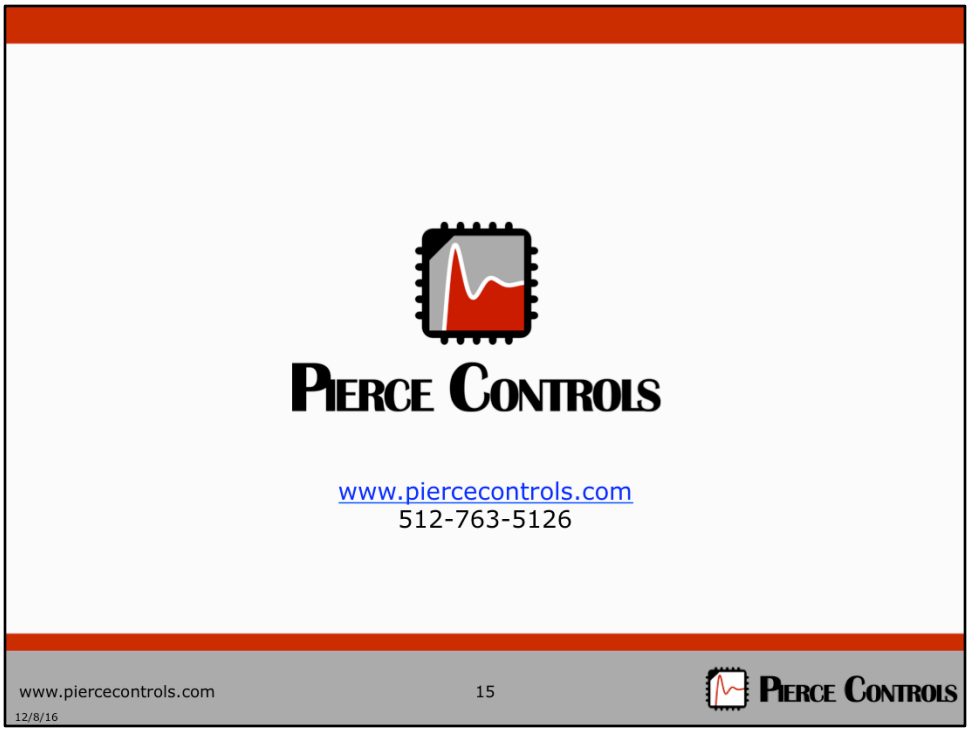
TCP/IP or UDP would be more complicated to set up, but the same basic approach could be taken. Using this method, you would be able to connect to any ADE.

You could also create and interact with web hooks to create a web-based UI. I am in the middle of developing a system like this now, that publishes data from tests to a website which is access controlled.

We are taught to de-couple code in LabVIEW and to think modularly. We need to learn to decouple the user interface from the functional code.

Additional Resources

- JKI: CLA Summit 2011: Public & Private Events Framework
  http://blog.jki.net/events/cla-summit-2011-public-private-events-framework-slides-available/
- Delacor: Delacor Queued Message Handler
  http://delacor.com/products/dqmh/



We do LabVIEW.

We are also experts in other areas, and we would love to talk with you about any LabVIEW, controls, integration, or systems problems that you are having.

Questions?

Thank you!