Smart People. Expert Solutions.®
LabVIEW User Group Meeting
New York Metro Area (Farmingdale)
2015-12-03
Agenda

1. DMC
2. Design Patterns – What? Why?
3. Basic Tools
4. Simple Patterns
5. Reference Architectures
Company Overview
Established in 1996, offices in New York, Boston, Chicago, Denver, and Houston.

**Industries Served:**

- Automotive
- Bio-medical
- Chemical and Food Processing
- Defense
- Electronics/Semiconductor
- Fuel Cells/Alternative Energy
- Hydraulics
- Laboratory Testing
- Machine Tool
- Material Handling
- Medical Devices
- Packaging
- Pharmaceutical
- Printing & Textiles
Certifications

National Instruments Silver Alliance Partner Vision Specialty

National Instruments LabVIEW Certified Developer

National Instruments LabVIEW Certified Architect

National Instruments Certified Professional Instructor

CSIA Certified

CVP Certified Vision Professional Advanced

DMC

Smart People. Expert Solutions.
DESIGN PATTERNS
Why should I use design patterns?

- Save development time
- Improve Modularity
- Increase Readability
- Take advantage of proven code/architecture
- SMoRES
SMoRES – criteria for a well designed application

**Scalable**: extending to N+1 should be simple

**Modular**: the application is broken into well-defined components that can stand on their own

**Reusable**: the code structured in a way that it could be reused in different applications

**Extensible**: new features can be added easily

**Simple**: simplest solution that meets all of the requirements
SMoRES – criteria for a well designed application

**Scalable:** extending to N+1 should be simple

**Modular:** the application is broken into well-defined components that can stand on their own

**Reusable:** the code structured in a way that it could be reused in different applications

**Extensible:** new features can be added easily

**Simple:** simplest solution that meets all of the requirements
SMoRES – criteria for a well designed application

**Scalable**: extending to N+1 should be simple

**Modular**: the application is broken into well-defined components that can stand on their own

**Reusable**: the code structured in a way that it could be reused in different applications

**Extensible**: new features can be added easily

**Simple**: simplest solution that meets all of the requirements
SMoRES – criteria for a well designed application

**Scalable**: extending to N+1 should be simple

**Modular**: the application is broken into well-defined components that can stand on their own

**Reusable**: the code structured in a way that it could be reused in different applications

**Extensible**: new features can be added easily

**Simple**: simplest solution that meets all of the requirements
SMoRES – criteria for a well designed application

**Scalable**: extending to N+1 should be simple

**Modular**: the application is broken into well-defined components that can stand on their own

**Reusable**: the code structured in a way that it could be reused in different applications

**Extensible**: new features can be added easily

**Simple**: simplest solution that meets all of the requirements
Basic Tools

- For/While Loops
- Shift Registers
- Enums
- Case Structures
- Event Structures
- Queues
Simple Patterns

- Functional Global Variables
- State Machine
- Event Driven User Interface
Functional Global Variables

Need: share data across a large application

• Local Variables?
• Global Variables?
Functional Global Variables

How does it work?

1. Functional Global is a Non-Reentrant SubVI
2. Actions can be performed on data
3. Enumerator (input) selects action, case structure action engine
4. Stores data in un-initialized shift register
5. Loop only executes once
Functional Global Variables

JRPB 04/08/2013
Use ProgramData directory, since in newer versions of windows, applications won't have R/W permissions to program files directories if windows user isn't on Administrator, so can't store system parameters files there when running as a compiled executable.
State Machine

Need: execute a sequence of events but the order is determined programmatically (usually user driven)
State Machine

Case structure has a case for every state

Transition code determines next state based on results of step execution

Shift registers used to carry state

Step Execution

Transition Code

FIRST STATE

NEXT STATE

FIRST STATE
State Machine

How does it work?

1. Case structure inside of a While-loop
2. Each state in the case structure holds code to be executed in its state
3. Each state has decision making code that determines next state
4. Enumerators (case selector) is used to pass next state via shift registers
Event Driven User Interface

Need: to detect user actions without slowing your application/missing them

- Event structure within While-loop
- Blocking function until event is registered or timeout
- Event structure configured to choose which events are registered (button presses, mouse click, etc)
Event Driven User Interface

How does it work?

1. Operating system broadcasts system events (mouse click, key press, button click) to applications
2. Registered events are captured by event structure and executes appropriate case
3. Event structure returns information about events to case
4. Event structure enqueues events that occur while it’s busy
Reference Architectures

- Producer/Consumer or Master/Slave
- Queued State Machine & Event-driven User Interface
- Daemon
- SEA Monster
- ??? (Yours!)
Producer/Consumer

Need: to execute code in parallel and communicate between them
Producer/Consumer

How it works:
1. Master loop with one or more slave loops
2. Master loop controls execution of slaves
3. Allows for asynchronous execution of loops
4. Decouples processes to allow multi-threading
5. Communication between loops
Producer/Consumer

Queue-based communication

• Adding elements to the queue

• Dequeueuing elements from the queue
Need: to enqueue events from a user that control the sequence of events in a state machine

- Event-driven user interface as a producer loop
- State machine as a consumer loop
- Communication between the loops using an event queue
How it works:
1. Events are captured by producer loop
2. Producer places data on the queue
3. Consumer loop dequeues data
4. State machine in consumer loop executes on data
5. Parallel processes communicate with state machine using queue references
Hello World!
Daemon

What is a daemon?

• Used to create and launch applications that run invisibly in the background
  • Auto-save
  • TCP/UDP messaging
  • Garbage collection of temp files
• Perform low-priority monitoring and/or maintenance or communication based processes
Daemon

Self-launching:
• Daemon must keep an open reference to itself to keep from being purged

Standard Launched:
• Launcher must transfer responsibility for reference to daemon VI
• Launcher must not close reference to daemon
DMC SEA Monster

Need: a very powerful, flexible, prebuilt/proven architecture to speed up development time for complex applications

Solution: internally developed (over the last 15 years) architecture based on a queued state machine & event-driven user interface producer/consumer model.

S – States
E – Events
A – Actions
Event Scheduler
Lives Here

Event Scheduler asynchronously “injects” events into Event Queue

DMC SEA Monster
## DMC SEA Monster

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Event1</td>
<td></td>
<td>Next State: StateC</td>
<td></td>
</tr>
<tr>
<td>Event2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event4</td>
<td>Actions: ActM, ActN</td>
<td>Actions: ActN</td>
<td></td>
</tr>
<tr>
<td>Event6</td>
<td>Actions: ActZ, ActY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Next State: StateC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DMC SEA Monster Logic Editor
DMC SEA Monster Logic Editor
DMC SEA Monster Logic Editor
??? (Yours!)
References

Too many to list - many readily available NI whitepapers
Questions?
Appendix - Tools

While Loop

Shift Register
Appendix - Tools

For Loop

Tunnel
Appendix - Tools

 Enums & Case Structures

Enum

Case Selector
Appendix - Tools

Event Structures

The following example shows an Event structure with the Key Down event case.

1. The event selector label specifies which events cause the currently displayed case to execute. To view other event cases, click the down arrow next to the case name.
2. The Timeout terminal specifies the number of milliseconds to wait for an event before timing out. If you wire a value to the Timeout terminal, you must provide a Timeout event case to avoid an error.
3. The dynamic event terminals accept an event registration refnum or a cluster of event registration refnums for dynamic event registration. If you wire the inside right terminal, that terminal no longer carries the same data as the left terminal. You can wire the event registration refnum or cluster of event registration refnums to the inside right terminal through a Register For Events function and modify the event dynamically. Depending on the palette from which you select the Event structure, the dynamic event terminals might not appear by default. To display these terminals, right-click the Event structure and select Show Dynamic Event Terminals from the shortcut menu.
4. The Event Data Node identifies the data LabVIEW returns when an event occurs. Like the Unbundle By Name function, you can resize the node vertically and select the items you need. Use the Event Data Node to access event data elements, such as Type and Time, which are common to all events. Other event data elements, like Char and VKey for example, vary based on the event you configure.

Note: For more information about event data elements, click the Details link in the event descriptions for the Control, Application, Pane, and VI event class topics.

5. The Event Filter Node identifies the event data you can modify before the user interface can process that data. This node appears in Event structure cases that handle filter events. If you want to change event data, you can wire and modify data items from the Event Data Node to the Event Filter Node. You also can change the event data by wiring new values to the node terminals. To completely discard an event, wire a TRUE value to the Discard? terminal. If you do not wire a value to a data item of the Event Filter Node, that data item remains unchanged.

6. Like a Case structure, the Event structure supports tunnels. However, by default you do not have to wire Event structure output tunnels in every case. All unwired tunnels use the default value for the tunnel data type. Right-click a tunnel and deselect the Default If Unwired from the shortcut menu to revert to the default Case structure behavior where tunnels must be wired in all cases. You also can configure the tunnels to wire the input and output tunnels automatically in unwired cases.