Advanced VisualWorks Course Outline

1. Advanced Coding Techniques (2 days)

1.1. Blocks – Advanced Use

Blocks are a powerful tool in Smalltalk. Their effective use can improve the readability, reusability and efficiency of code. However, because most programmers are weaned on languages with no equivalent, many programmers do not make good use of blocks.

This module includes hints on how to make good use of blocks, and describes some advanced features.

1.2. Processes and Concurrency

This module explores the features available in Smalltalk for the expression of concurrency. It introduces classes Process and ProcessorScheduler. Classes to support various synchronization operations, including Semaphore, SharedQueue and critical sections are explored though the use of examples and exercises. Class Delay is also considered. This module concludes by describing how instances of class Promise may be used to provide background tasks.

Pre–requisite: "Blocks – Advanced Use"

1.3. Handling Exceptional Conditions

All real programs have to deal with exceptional conditions. Following a summary of different approaches to exception handling in VisualWorks, this module concentrates on describing classes Exception and Signal, giving examples of their correct usage.

An exceptional condition is something that is expected to occur infrequently, and does not fit into the usual pattern. More often than not the condition is detected in a piece of code that cannot properly deal with the condition, because the context in which it has been detected is inappropriate. Hence, the condition must be signalled to a wider context, i.e., the calling code.

Pre–requisite: "Blocks – Advanced Use"

1.4. Miscellaneous Tricks

In this module we explain several tricks that we have found of use to the advanced VisualWorks developer. These include: multiple dispatching, the perform: message, the become: message, the doesNotUnderstand: message, and using nil as a superclass to build an Encapsulator. The module contains a description of each trick, combined with examples and exercises.

1.5. Weak References

Most object references in VisualWorks are strong. If there is a chain of strong references to an object from one of the system roots (e.g., the system dictionary, Smalltalk), then the garbage collector will not reclaim the object. However, if the object is only reachable via one or more chains with at least one weak reference in them, then it will be reclaimed. Weak references can only appear (directly) in instances of WeakArray (not even in instances of a subclass).

1.6. Metaclasses

This module considers the implementation of the class structure within Smalltalk, and introduces the Metaclass concept. Classes such as ClassDescription and Behavior are explored. The concepts here are widely misunderstood, possibly because of the tongue–twisting terminology used; an attempt is made to clear away the confusion in this chapter. (It's also worth saying that some of the ideas presented here can be difficult to understand, and in practice, you need to know almost nothing about metaclasses to use the system effectively.)

2. VisualWorks Optimization (0.5 days)

This module demonstrates how to overcome the performance bottlenecks in your VisualWorks application.

Performance problems are usually due to a bad choice of algorithm, or poor implementation of the algorithm. This module will show you how to find where your application is spending its time and provide some tips and techniques to improve its performance.

3. Advanced Application Building (2.5 days)

3.1. Review of Application Model Framework

The VisualWorks Canvas mechanism provides an application framework on top of MVC. This module first differentiates between application models and domain models as they pertain to the frameworks. We then discuss how to connect a widget to its underlying model, along with a review of the dependency mechanism as utilized within the Canvas mechanism via the notification property or the more generic onChangeSend:to: message.

This module also provides a review of the ApplicationModel class, including the method that should be subclassed: preBuildWith:, postBuildWith:, etc.

3.2. Subcanvasses

The Subcanvas widget provides one means of reusing visual entities, such as commonly used layouts, etc. Visual Reuse is just as important as code reuse. In this module we demonstrate how to reuse visual components through use of class DateModel.

3.3. Notebook Widget

The Notebook widget is the most complex widget, and hence often overlooked as a user interface widget. In this module we describe how to create a complex notebook widget, by providing an example and exercises.

Pre–requisite: "Subcanvasses"

3.4. Models

This module describes the classes whose instances can be used as models for VisualWorks widgets. We begin by describing class ProtocolAdaptor, whose subclasses provide the behavior to "adapt" Domain Models. We then examine two other subclasses of ValueModel: BufferedValueHolder and BlockValue. In addition, we describe how a UIBuilder uses bindings to determine the model to be used for each widget.

The module ends with a description of some classes prepared by the authors that exploit the techniques described.

3.5. Window Operations

This module describes windows, in particular, how to bring up modal dialogs and how to connect the opening and closing behavior of several related windows.

3.6. Drag and Drop

In this module, we describe how to use the Drag and Drop framework and provide examples of its use. Drag and Drop is a new addition in VisualWorks 2.5. A good example of the use of Drag and Drop features is in the VisualWorks tools themselves, the Launcher, browsers and file list. The Drag and Drop version of the tools may be filed in from extras / tooldd.st

3.7. Coding for Multiple Platforms

One of the attractions that VisualWorks offers is the seamless way in which it is portable across multiple platforms. In this module we provide examples of good implementation practice for platform–independence including an example of the way in which the VisualWorks user interface widgets can be used to provide platform–specific dialog boxes.

3.8. ViewHolder Widget

The Palette contains numerous widgets to display some aspect of your domain model. However, there are bound to be occasions when you wish to provide the user with a more graphical display. The ViewHolder widget provides a means of displaying such a "custom view" in a Canvas. In this module we describe how to create a new view and use it within a Canvas.

3.9. Adding a Widget

So far in this course, we have considered how to build applications with user interface components from various sources: a widget provided by the Palette; reusing Canvasses using the SubCanvas widget; and embedding a custom view using the ViewHolder

widget. In this module, we provide a basic introduction to one other source: a locally–provided widget added to the Palette. Since the mechanism by which a new widget is created is extremely complex, this module provides an overview of creating a new widget by way of a worked example.

Pre–requisite: "Blocks – Advanced Use" and "ViewHolder Widget"