P1450.4 meeting minutes - 11/17/04

Attendees: Dave Dowding, Ernie Wahl, Don Organ, Tony Taylor, Jim O'Reilly, Oscar Rodriguez

Not present: Jim Mosley, Eric Nguyen, Dan Fan, Tom Micek, Jose Santiago, Ernie Wahl, Doug Sprague, Yuhai Ma

Agenda:

- Discussion of comments/feedback from ITC presentation.
- Discussion of points raised in last week's meeting (listed in meeting minutes) and the responses to those points.

ITC presentation:

- Overall reception of presentation was good. Attendees were generally warm to the idea of the progress of our efforts. Crosstalk between STC OA and STIL (STC OA – standard HW and SW interface) – more resistance to HW standardization. SW standards – interface – level below interface – Ernie's interest – tester-independent interface. How would STIL fit into this scheme? Overlap/overflow of views from OA sessions - interface
- Comments/questions from audience.
    - Why include GUI coordinates in the language?

Syntax discussion:

- Discussion about need for TestObject block.
    - Inside a flow, can you directly execute a TestMethod, or must you instantiate a TestMethod as a TestObject before you can use it in a flow? Ernie's responses/comments designated by >

```
>   The conceptual model requires the instantiation of a TestMethod however,
>   this can be done one of two ways:
>
>   1. define before use, e.g., given some TestMethod Fn
>      Fn fn(timing_levels, patlist)
>      creates an instance that can be referred to by its id "fn".
>   2. inline instantiation inside a FlowNode, e.g.,
>      Fn(timing_levels, patlist)
>      creates an anonymous instance
```

        - NOTE: Inline instantiation will look as though the test method were called directly, though behind the scenes, an transient object (consuming some amount of memory) will be created for a finite lifetime. This lifetime will only be as long as the information from this TestObject is needed – i.e.:
            - Until exiting the TestNode containing the inline-instantiated TestNode
            - Until the next TestObject (defined-before-use or inline-instantiated) is called.
            - ?

          We need to think through the ramifications of each of these alternatives.

    - What are the benefits of a standalone TestObject?

```
>   1. it can be referred to by multiple FlowNodes possibly in different
>   Flows. This is a communications/maintenance issue, e.g., in order
>   for the test program to work properly, FlowNodes a and b must remain
>   in sync: if a user unwittingly edits the one's TestObject, the other
>   stays in sync by design.
>
>   2. any standalone TestObject can be designated as an entry point via
>   the EntryPoint object.
```

- Discussion revolved around the issue of two different TestNodes using the same TestObject, and the effects if that TestObject were edited from one or the other of the TestNodes. If (when!) that occurs, the user MUST realize that the tests invoked by BOTH TestNodes have been modified. If the user requires different behavior from the TestObjects invoked by one of the two TestNodes, a separate TestObject would be required. If edited in a GUI editor, it would certainly be possible for such an editor to keep track of the number of places a TestObject would be used; if that number were >1, the user could be notified that changing the TestObject in TestNode A would also change it in TestNode B. We're all pretty much in agreement on the desirability of this behavior.

```
>   1. it can be referred to by multiple FlowNodes possibly in different
>   Flows. This is a communications/maintenance issue, e.g., in order
>   for the test program to work properly, FlowNodes a and b must remain
>   in sync: if a user unwittingly edits the one's TestObject, the other
>   stays in sync by design.
>
>   2. any standalone TestObject can be designated as an entry point via
>   the EntryPoint object.
```

  - What would be the benefits of a standalone TestNode? (D13c currently doesn't allow these . . .)

```
>   Preface 1. What's in a name: the conceptual model calls this a
>   FlowNode because its main function is to be a node in a directed
>   graph (the referred to TestObject performs the test).
>
>   Preface 2. the conceptual model allows a standalone TestNode but
>   only in the context of a Flow.
>
>   The benefits are to allow the existence of incomplete states and to
>   test the actions during test program development.
```

- Why has the Next block moved outside PostActions in the TestNode?
  - Answer: In order to more closely map onto our conceptual models. Those models show the arbiter as selecting one of N exit paths, each with its own set of PostActions. The connection of a particular exit path to whatever it does (or **doesn't!**) connect to is an attribute of the particular ExitPath chosen. It seemed illogical to have that connection specified in the PostActions block. Further, a side effect of this change is that the PostActions for the TestNode are now identical to the PostActions for the TestObject – and that appealed to our love of symmetry!

```
>   Our love of symmetry is based in a desire to simplify, i.e., to
>   distill the minimum number of objects which in turn minimizes the
>   number of explanations required to impart understanding and the
>   amount of software that needs to be written and maintained to
>   support this system.
```

Further discussion about the desirability of having separate common and ReturnState- or ExitPort-specific PostActions for the TestObject and TestNode. Tony suggested that the (conditional) syntax of the common PostActions could serve as the arbitrator, and that the syntax description could be collapsed, while still maintaining the capabilities we're after. Jim will draft such a version of the syntax, and distribute it.

- Left over from last week and previous weeks:
    - We need to define explicitly the behavior of the Exit and Stop actions in the PostActions block of both the TestNode and the TestObject (if we keep that object). The anticipated behavior of the Stop action (stop this test program) seems pretty clear. However, the anticipated behavior of the Exit action (Exit this test flow) may need more thought.
        - If the flow being exited is a top-level flow (as called by an EntryPoint), then it effectively "returns" to the test program that called it, and execution stops at that point.
        - If, however, the flow being exited is a subflow, the return is to the TestObject which called the subflow. What conditions (binning, pass/fail status, etc.) would apply here? I suppose that if a TestFlow were considered as a specific type of TestMethod, it would, at its simplest, return pass or fail – no branching would take place at the boundaries of a flow, but only within a flow. So, in that case, there's not really any ambiguity, and the "Exit"ing of a (sub)flow simply means that the execution continues at whatever TestObject or TestNode (or EntryPoint) called the subflow. We'll need to gin up some examples to show this behavior.
    - We have some further work to do on the syntax of the Arbiter block, as well as some work on the exact keywords we'll use for various blocks in this document.
    - We also have some further work to do in discussing Ernie's proposal to include an integral TestMethod type of Flow (so that TestMethods can inherit the properties of a flow, if desired).