P1450.4 meeting minutes - 11/19/03

Attendees: Jim O'Reilly , Don Organ, Dave Dowding, Eric Nguyen, Yuhai Ma, Jose Santiago, Doug Sprague, Ernie Wahl, Jim Mosely
Not present:

Agenda:

- Review of minutes from previous meeting.
- Discussion of what the new level of flow hierarchy (the variant of flow-node body which contains a test, rather than a subflow) should be called.
- Proposal about how we our flow extension constructs link into the remainder of 1450. P1450.1 environment block used by P1450.6 WG to place their extension information.

SUMMARY:

The discussion today centered around two main topics

- Integration of our flow constructs into the remainder of STIL
- Discussion about what we've been lately referring to the "test" entity which can be called or pointed to by the flow-node body. See Fig. 2a below. The flow-node body can also invoke a subflow (made up of other flow-nodes) - as shown in Fig. 2b below. In particular, we're trying to come up with a label (acceptable to all) for this object.

Integration of our constructs into the remainder of STIL
Dave has made a proposal that we consider the use of the STIL environment block for encapsulating our flow constructs. While not everyone is convinced (yet!) that the use of the environment block is necessary, it's possible we'll need to do this to allow STIL programs which contain these constructs to be read by older STIL readers which aren't aware of these constructs. Stay tuned in coming weeks . . .

Flow-node module contents:
We seem to be agreeing that the flow structure hierarchy should include an entity between the flow-node and the test method. This entity can be called by the flow-node (via a call or reference or pointer or ? in the flow-node body), and contains pre-actions, the test method call, post-actions, an arbiter and pass/fail actions, both of which lead to a common exit path (see Fig. 2a, choice 1 or choice 2, below). During this discussion, we also spent a good bit of time talking about issues that we seem to be already agreeing on - perhaps with the implied intent that ongoing discussion helps us all to feel more comfortable with the concepts, and to solidify the agreement?

One of the problems we've been wrestling with is what to call this additional layer. Calling it simply a "test" seems to be a bit too vague - the word "test" is pretty overused, but in general, since it contains a pointer or call to a test method, having the name of this type of object include the word "test" seems to be a good idea.

Some of the alternatives for names are: "test block", "test instance", "test module". Of these, the name "test module" seemed to be the most acceptable - so for now (if and until we can think of a more appropriate name), we'll refer to the object as a "test module".

Other issues we've agreed on (if I'm mis-stating, please correct me!).

- The test module must provide for the behavior of both Fig. 2a choice 1 and choice 2. The actual structure will be that of choice 2, with defaults for the pre- and post-actions, arbiter, and pass/fail exit actions such that the test module can be made to behave as a simple test method call, with any decisions and branching based on the test results being carried out by the flow-node post-actions, arbiter, and exit-actions. In this case, the test-module pre-actions will simply do nothing (as an implementation, perhaps the test module pre-action list would simply be an empty list)
- The flow-node body must be able to invoke (call, point to, reference) either a subflow or a test module.

- The flow-node body does not CONTAIN a subflow instance or test module instance, but simply invokes (calls, points to, or references) previously and separately defined instances of either a subflow or test module.

Action items:

- Dave to update the diagrams with the currently-agreed-upon labels
- Jim to create a reference document of example test program flow fragments from various vendors' tester languages, and annotate those constructs with the (work-in-progress) STIL 1450.4 terminology. This document will serve as a baseline reference, showing how currently-available languages deal with and describe the constructs.

Figure 2: Variations on What a FlowNode Module Is

Figure 2A: TestMethod Module

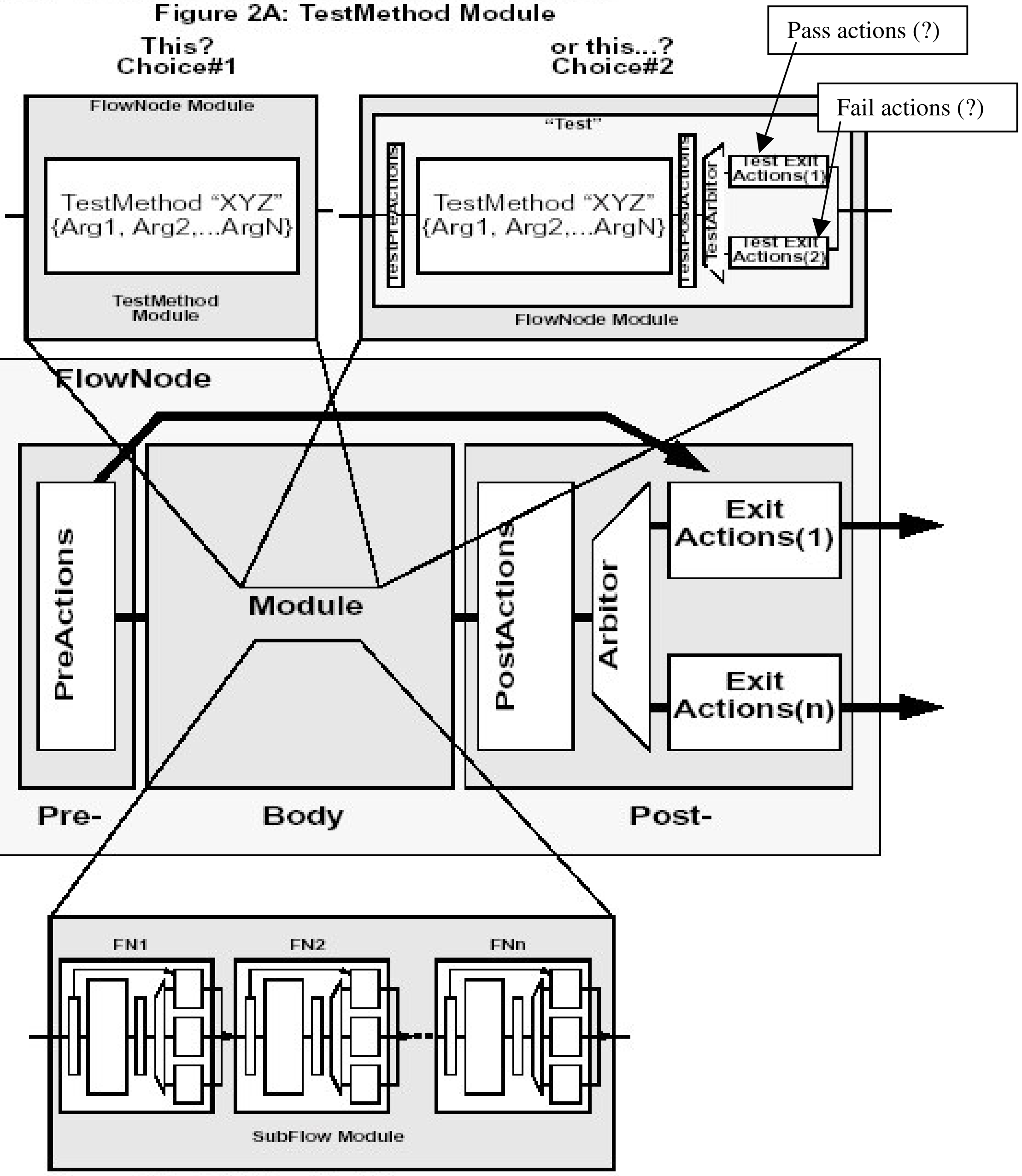

Figure 2B: SubFlow Module

Transcript: (somewhat paraphrased, but I believe it's reasonably accurate and that I've captured the essential points).

Dave: Should we use Environment block to wrap test program flow constructs (i.e., P1450.1/P1450.6)? Encapsulate test flow only in env block - or whole set of test program flow-related structures? Just as CTL (p1450.6) guys did. The advantage of environment block is that when you're talking about about a certain type of data, the environment encapsulates the data. . . ., which can be ignored (if I understand this correctly) by the STIL reader

Jim: What is the advantage of using an environment block?

Dave: STIL provides for one and only one pattern exec. It may be necessary to use environment block to avoid breaking any existing STIL programs.

Dave: appears to be an additional level in flow hierarchy (flow-node -> "test" object -> test method). We need to agree on what name we'll use for this additional level (and also agree on names we'll use for other objects we've already described

Yuhai: Will send out example Advantest program. Simple example - main flow.

Jim will annotate examples of various vendors' test programs with the labels we've been using in the WG.

Need from Yuhai - run test flow node with no branching, run test flow node with branching, how does it reference the type of test (patterns, levels, timing), and if possible, binning.

Dave: Where are we conceptually? Flow nodes - can encapsulate test method call, or a subflow. Added the "test object" layer - similar to a flow-node - contains a call to a test method. (We also have test program entity, which contain multiple flows (entry points)). Need something a bit different than a flow-node and different than a test method call.

Ernie: At some point, we'll need to talk about a test type - and that's different from a test instance. **(Ernie, what do you mean by this? - please elaborate! - Jim)**

Dave: Definitions for each of these entities: flow-node, test block, subflow, test method. Capsule, segment, test instance ( ).

Ernie: test instance is OK.

Dave: I hedge because it may be an entity that doesn't do any tests (on the device) at all, but simply does some actions (in and/or to) the test program. Mainly, the term "test" is very overloaded - may imply something that we're not trying to say.

Ernie: Don has suggested that he may need a lighter-weight object - may not be a test. But, trading off more and different kinds of objects (flexibility) against syntactical complexity. But, looking at the block, it has pre-/post-actions, I think of the conglomerate as a test, since it does have pass/fail implications. On the other hand, if it doesn't have P/F implications, either have an object which doesn't have those implications, or if the object's actions didn't fail, it passed (pass is default).

Dave: I want on naming the object - flow-node says that it's in a flow, and so will have a path in, multiple paths out, and a decision about which path out to take (indicating where to go next). Part of a flow-node's responsibility is to provide information about where to go next, based on the results of the actions which occurred in the flow-node body. This new entity, on the other hand, is something you flow into and out of - and there may, or probably will, be results - but it is not in a flow on its own. So, what do we call it?

Ernie: We had talked about this object as being an executable. Don't necessarily like that name - because most of the time, we're talking about a test (something that performs a test on the device), so why not include "test" in the entity name. In my view, "Test" is pre-actions, test method call, post-actions, arbiter, and pass/fail exit actions. But Don has suggested that, in some circumstances, all that structure may not be needed. Perhaps the user simply wants to perform some actions, with no pass/fail results like a test would, so there might be a need for some other entity - which has been referred to as an executable.

Dave: Fig. 2a., choice 2 is what I would consider a test instance.

Jim: Is Fig 2a, choice 2 what we're calling an executable?

Don: Executable maps closer to fig 2a, choice 1. Call to test method, with arguments, but no indication of where (which flow-node) to go next. May include some of the pre- and post-actions as well - but no determination of next step. So it could be something in between Fig. 2a, choice 1 and choice 2.

Ernie: How about test instance? But if there's another name that makes sense, that would be OK with me.

Jose: Call it a flow-node instance?

Ernie: Flow-node is pointing to the "test instance", not vice-versa - unless we want to have a two-way pointer (actually an N-way pointer, since a test can be pointed to by many flow-nodes). So using the name "flow-node instance" for this object doesn't make sense.

Jose: Didn't want to use "test instance" - since the body of a flow node can contain either a simple test method call, or a test method call surrounded by pre- and post-actions, arbiter, and pass/fail actions, calling it a "test instance" may be misleading or ambiguous. If we are referring to the flow-node, then an article

Ernie: Remember, the test should NOT be considered as an integral part of the flow-node, but is just pointed to, or referred to, or called, by the flow node.

Jose: either Fig 2a, choice 1 or choice 2, is what I'd consider a flow-node instance.

Ernie: What seems to be getting lost is that the flow-node doesn't contain a test, but simply contains a pointer to a test - and I wish that would be clearer in the diagrams (if we're agreeing on it - and if not , at least offer it as an alternative).

Dave: How should we represent that relationship? If you've got an example or idea, let me know and I'll update the diagrams.

Ernie: Perhaps just use an arrow pointing from the flow-node body to the test instance.

Jim: Are we agreed that whatever is pointed to by a flow-node body (either Fig 2a, choices 1 and/or 2, or Fig. 2b - a subflow) can be pointed to by more than one flow-node?

Group: general agreement on this point.

Ernie: But when I hear people talk, that point sometimes seems to be glossed over or forgotten.

Dave: But what we really need is a NAME for that object (the "test instance") that we can all agree on.

Jim: Some examples:

Agilent SmarTest - test (canned test method) or test function (user procedure in C code).

Inovys Stylus example (from odd_even.stil) - flow-node = segment - segment has a type, simply called "test").

NPTest: flow-node = segment. Flow node (segment) referenced test (which could be of many different types of tests - dctest, ftest, etc., which references timing and levels.

Advantest: flow-node = flow-item. "Test-instance" is implicitly referred to as "flow-test".

So, we could call it a test block. Thus, the hierarchy would be: flow-node -> test block -> test method.

Ernie: In the past, I've used "test block" to mean subflow. And I think that Credence has used "test block" for a similar purpose.

Jim: How about "test module"?

Jose: ... or "macro"?

Dave: We probably should stay away from macro - that's even more overused than "test"! Why don't we, for now, call it "Test module" for now? And I'll update the diagrams to reflect that, so we can talk about the various layers and know what is being referred to.

Jim: Do we want to allow both Fig.2a, choice 1 and choice2? Or should we set up the defaults for Fig. 2a, choice 2 so that it can devolve into choice 1?

Dave: I believe we need both.

Ernie: I have no use for choice 1 - but wouldn't object to it if others needed.

Don: Choice 1 is trivial form of choice 2, if defaults are chosen appropriately.

Ernie: Choice 2 implies a common-denominator structure for a test - pre- and post-actions, arbiter, exit (pass/fail) actions. But there's more - maybe the pass/fail flag should be part of the common denominator, so that the flow-node always knows how to access the pass/fail flag (and other elements) in order to perform it's arbitration.

Jim: Would like to arrive at some decisions - and weed out the non-chosen alternatives from discussions and diagrams. This would help to clarify our discussions and help to move us forward.

Jose: It's not a choice between choice 1 and choice 2, but more a matter of usage - i.e., can I make choice 2 look like choice 1?

Yuhai: I agree - simplest approach is choice 1. If we use choice 2, we must allow choice 2 to devolve to choice 1 (via intelligent assignment of defaults for pre- and post-actions, arbiter, and pass/fail actions) if the user desires the simpler form.

Jim: I propose that future versions of the diagrams show only Fig 2a, choice 2 (and show - elsewhere - that Fig. 2a, choice 2 can be made to behave as choice 1).

Jim: And are we also agreed that we want either or both of Fig. 2a, choice 1 and 2, and also Fig 2b (a subflow)?

Group: General agreement on that point.

At this point, the call concluded.

Till next week,

Jim