P1450.4 meeting minutes - 12/03/03

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

Agenda:

- Issues from minutes of last meeting.
- Any other open issues
- Discuss different layers of test program from a flow perspective. Capture and agree on those conceptualizations.
- Jose: STC activities discussion that STC group wants to have - in the form of a letter that STC will send to our group about finding alignment between STC activities and STIL .4 activities. Wait for Yuhai . . . If no Yuhai, we'll talk about this next week.

NOTE: there was no meeting on 11/26/03. Only Jim, Eric, and Yuhai were on the call. Adjourned after waiting 30 minutes for others.

SUMMARY:

The discussion today was focussed mainly on the test program layer, its behavior, and how it relates to some of the lower-level layers we've previously discussed.

Test program layer:

- We seem to be agreeing that, conceptually, the test program block is the top-level block or container for test program/testflow-related objects. We haven't settled on what it will be actually called, but the concept seems to be OK for everyone.
- The test program block is probably the object that would, in an actual implementation, interface to the tester OS.
- We also (I think) agreed that we should probably allow more than one test program block in a single STIL file (if we just allowed a single test program block in any single STIL file, one could, of course, have multiple test programs by having multiple STIL files . . .)
- One point still under discussion is whether the top level (the test program level) can be treated as a subflow or a test (which will, when executed, return a pass/fail result). I think we agree that it has some of the characteristics of a subflow or test, but is fundamentally different from those types of objects.
- I believe that we're also agreeing to the utility of a variety of independent flows which can perform specific tasks (on-load, on-start, on-reset, on-alarm, etc.). The test program block would specify the name of a flow to use for a specific function (kind of like setting up a signal handler in a Unix C program . . .). Does the name **entry-point flow** work for these types of flows? Let's talk about this in the next meeting.
- I further believe that we're agreeing about the fact that having a set of these entry-point flows (whose execution is sequenced either by the tester OS in a specific implementation of STIL, or perhaps specified in some yet-to-be-defined STIL block), rather than having a single top-level flow which sequences procedurally through the various entry-point flows under control of yet-to-be-defined initialization flags (so that an on-start can't get executed until the on-load is done, for instance).

Transcript (paraphrased):

Ernie: Can we update drawings to show that flow node module points to a test module, rather than containing a test module.

Dave: Haven't redrawn - was overloading everyone with updates. After today, if we can identify the other layers, I'll update. Previous meeting minutes had indicated agreement on having a pointer to a test module which contains the pre- and post- entities (regardless of how it's represented syntactically)

Ernie: Test module exit actions should be pass actions/fail actions, instead of actions 1 and actions 2. I think we had agreement on this . . .

Dave: Had already updated, but not distributed the diagrams. I think we're all in agreement on those points.

Dave: From minutes of 11/19/03 meeting, there's a request for elaboration

> 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)**

about the difference between test type and test instance. Perhaps an email discussing this, or a reference to something in an existing document would be helpful.

Ernie: OK - I'll follow up and clarify that issue.

Dave: Let's look the different layers of the test program (as shown in the P1450.4 1999 document and our working draft). Refer to the current draft of the 1450.4 document from 12/09/02 (December 9 2002 (pdf)). We've started in the middle, and have alluded to the fact that there are layers above (and below) the ones we've been discussing (i.e., top-level flow(s)). We need to flesh out some of the upper-level constructs, and I'll then try to diagram them (along with the corrections to the existing diagrams), and get all the conceptual diagrams into the document.

Ernie: The diagram on p13 shows test program, with various top-level flows (entry points?). I thought that we had at least agreed, in principle, that the test program structure was a useful entity, with a set of proposed top-level flows - but I don't know if we decided what those flows (editor's note: entry points?) should be, but we can use the ones shown in the document as a starting point.

Dave: Should there be a top-level flow which calls the test program? This top-level flow may contain the ON_* subflows, a test module, a flow-node, etc. I don't care how it's packaged, but some kind of container that represents the different variants of a test program (sort, final test, etc.), either in a single executable test program or separate executable test programs - different pieces of overhead flows, and one or more "main" flows" that get pointed to at load time.

Ernie: I thought the proposal was to have a test program level - on-start/on-reset/on-load would be flows, as opposed to having a single flow at the test program level that would have (if on-start do on-start flow if on-reset do on-reset) subflows. We could represent things either way; I don't know why the current proposals prefer the former over the latter, but it's my understanding the current thinking DOES prefer the former.

Dave: Looking at it from the types of diagrams that we've done before, would it be fair to say that we could have a top-level flow containing these things (subflows?). Does that fit everyone's conceptual representation?

Ernie: Yes - again, don't know why we'd prefer one over the other.

Don: One advantage to such an approach (as opposed to if-statements) is that there's more info conveyable to the environment. If you're loading a program and you know that you'll use only some of the start points, then you have the option of not loading constructs associated with the unused start points (which could be an advantage if your tester has limited resources).

Dave: Yes - I agree . . .

Ernie: The on-load somehow seems qualitatively different than the other flows

Dave: Eric, Jim M. - would it be usable in your platforms to have such a top-level entity that wraps up the different pieces in one place?

Eric: Not sure what you're asking . . .

Dave: Test program contains various entry-point flows - on-start/on-reset. Inside test program - we may also have "wafer sort flow" or "characterization flow" (which may contain the entry-point flows, and which would contain a full test program (flow) for wafer sort or characterization. In contrast, someone might want to distinguish between such test conditions in test methods, and have only a single program flow). Regardless, there's a place where you prescribe a test program flow once the load and init things are done, and that flow contains the start-to-end test for a given device.

Eric: Looks OK for our platform - we have similar constructs.

Dave: Is that conceptual view (having the test program as the main container) OK to work with?

Jim M: Having the top-level entity to keep track of the various activities (alternate flows, exception-handling flows, etc.) IS a useful construct?

Don: I like the concept as described.

Ernie: I'm OK with it also. Having accepted it as a useful concept, note that p8 of current-draft document shows what the differences between the various entry-point flows (on-start/on-init/...) are.

Dave: Let's not go into the details yet . . . but just want to get an upper-level view that holds together for everyone. If I consider the "test program" (the top-level container), can we describe that level as a special type of subflow? Is there any value to that notion?

Ernie: Yes . . . it may allow you to pass in specific arguments to a top-level flow, to allow you to distinguish it from others (?), esp. when you start dealing with types. If you want to stamp out certain types, you want to be able to say that this flow is of a different type than other types of flows.

Dave: Is it a good idea, that we could have more than one of these, so that we could have separate ones for various circumstances (i.e., wafer sort, characterization), or is it better to have only one, and within that one, have different variations on the test program?

Ernie: Thought we were agreeing that we'd have top-level flows for the various entry points (on-start/on-reset) within a test program.

Dave: Yes, that's right - but there's also a top-level test program. Is it useful to have more than one test program at the top-level, or should we limit it to one?

Jose: For me, top-flow could be considered as the tester interface; as such, it can contain subflows (one of which would be the main test program flow - which contain flow-nodes).

Dave: Not tied to the existing diagrams on p13 and p21 in the current-draft document; thought it was a good place to catch some of the entities that we'd dealt with before - not trying to say whether these diagrams are "good" or not - just trying to get a conceptual view of the upper levels, and work down to the levels we've been discussing. Is there virtue in having only one test program, or should we allow more than one?

Jose: Should be able to contain as many subflows as necessary . . .

Ernie: Initial reaction - allow only one . . . but on further reflection, that may be too restrictive. It would probably be my preference, for maintainability reasons, to keep all the various pieces together, and use some switches to choose, say, the wafer sort or the characterization path. But I can see that others may want to, or need to, use more than one (test program container). So I'd be open t

Dave: Conceptually, we haven't precluded the methodology you describe as your preference - haven't specified what the various switches would be, but are allowing for that type of flow-control. Other wild ideas - following the .3 (tester targeting WG) work - we may want to be able to prescribe different flows (and different test methods?) for different types of testers. Didn't want to bias the group, but was leaning towards the usefulness of multiple top-level test programs.

Jose: Could specify multiple references to the same production flow, but for different testers, would be useful

Jim M.: Should be able to have as many top-level test programs (debug flow, different prod'n flows) as necessary.

Don: I'm a generalist - this is a small step in the language, doesn't add much complexity to the language, and buys a lot of flexibility. A good step to take.

Jose: Remember, we ARE defining a language - and allowing multiple top-level containers (test program?) gives the language consistency and allows for easier packaging. We allow multiple entry-point flows and subflows - and we're considering treating the test program as a special type of subflow, but a subflow nonetheless, so it wouldn't make sense to restrict the number of that type of subflow, when we allow more than one of other types of subflows.

Ernie: Entirely comfortable with the idea - just thinking about how the language syntax would look. We'd probably end up with another level of context - all the various test programs sitting there, picking and choosing from the predefined instances of lower-level constructs.

Dave: That's how I view it as well . . . the organizational block, not the declarative block. Is this test program subflow/flow-block special or different from our general view of what a subflow is?

Ernie: Yes, I thought it was . . .

Dave: I feel that way too - I think the test program should be considered as a subflow, but one with characteristics unique to the fact that it's the top-level a little bit of a special subflow entity

Jim M: Yes, the top-level test program is different than other subflows.

Dave: I'll diagram it that way, then . . . If we need to revise that notion as we flesh out the details, we can adapt. Consider if we had a GUI front-end, and pointed to a top-level program, and executed it, I could run a whole test and get a pass/fail result. Or, I could double-click into the top-level flow and execute some of the subflows.

Jose: That would allow the user (in a GUI flow editor, perhaps) to navigate down through the hierarchy, all the way down to test method.

Ernie: Here's my view - the test program is different from a flow and subflow - because it isn't really a flow, but instead chooses (points to) one of the entry-point flows (i.e., on-start). I don't think of a test program as a flow per-se (it's different from a flow and different from a subflow). The test program specifies one of the entry points to execute (an entry -point is a pointer to a top-level flow), executes it, and gets back a pass/fail result). I see three levels - the test program, which specifies an entry point; one or more entry point flows, each of which point to a top-level flow; and the top-level flow(s) which point to subflows or testflow nodes.
entry-point specifier (on-start) which points to a top-level flow, which points to subflows or test-flow-nodes.

Dave: I see it both ways - I click on the top-level program, it knows which on-start to use (operating system knows whether the load has been done, and so on)

Ernie: So when the user clicks on test program (to execute it?), it (the GUI flow editor?) would choose appropriately choose to do on-start, on-reset, on-load, etc.

Dave: Yes, that's my high-level view - don't know if it's right or not . . . but that's how I pictured it.

Jose: The top-level part can be seen as hookups to the tester OS interface **(Jose: did I describe this accurately? Can you elaborate? Jim)**, while the test flow is one level down.

Dave: A tester OS may have checks in it to verify that the proper patterns/timing/etc. have been loaded. Or should we execute the on-load entry-point flow. I view each of the top-level "test program subflows" as a test - it could be executed (i.e., by a production operator) and would produce a pass/fail result without having to push down into the full flow. **(Dave: by this, do you mean "without explicitly specifying execution of the subflow"? Jim)**

Ernie: Sounds reasonable . . .

Jim M: For the execution part of the test program we've been discussing, we've used the term "session control" - script acting on the orders needed to run each of the different activities (a work-order?). Do we want to build this capability into the test program object, or just have it available in the interface to everything else that the session control uses to control the flow?

Jose: How far do we want to take this? Just to the test program layer, or beyond that to the operator interface layer (incl. control of probers/handlers). Not sure we want the language to incorporate the operator interface layer . . .

Dave: One of the areas I'm interested in seeing is whether the c stuff **(Dave: please elaborate?)** has evolved into that area or not. Maybe the OS has the intelligence to say that the on-load hasn't been done yet, so the selected main flow can't be run, or maybe not. However, I think these areas are important, and need to address them somehow - either in our language constructs or advisories to the implementation. I think there's a place in the top-level entity (the one that contains a test program level, which is a special subflow), for specification of which entry-point flows (on-start, on-load, etc.) to use.

Ernie: We should have a glossary of terms . . . I attach very specific semantic meaning to terms - i.e., "a test program is a subflow" - I already have in mind what a subflow is, and a test program is NOT it.

Dave: Yes, that would be useful. I'll put what we've discussed and said into a graphical representation - but remember, it's not cast in concrete, and it's subject to revision. I'll also make the corrections to the diagrams that we've agreed on in the past couple of weeks - and distribute the documents.

Ernie: What should I send out via email?
Dave: I think we just need elaboration on your reference from the 11/19/03 meeting to the difference between test type and test instance. We'll have to deal with this soon, but probably not next week, so anytime in the next week or two is probably fine.

At this point, the call concluded.

Till next week,

Jim