P1450.4 meeting minutes - 04/14/04

Attendees: Dave Dowding, Jim O'Reilly, Ernie Wahl, Tom Micek, Tony Taylor, Yuhai Ma, Don Organ, Doug Sprague, Jim Mosley

Not present: Eric Nguyen, Dan Fan, Bruce Parnas, Jose Santiago

Agenda:
- Overview of Tony's STIL-EBNF'd OTPL document.

Overview of Tony's STIL-EBNF'd OTPL document:
Code examples pulled verbatim from OTPL document.
OTPL appears to be very file oriented (many file types that can be imported). Yuhai confirmed this. As document was being translated, Tony is coming up with quite a number of questions. He'll update the STIL-EBNF translation of OTPL, annotating with questions where they arise.

OTPL areas likely to be of most interest to STIL .4 WG
- UserVars. There's a UserVars section of the .tpl file, and also a UserVars file - can UserVars can be specified in both places interchangeably?
- Counters (variables which live over multiple executions of the test program?)
- Flow (flow gets defined here. Groups together multiple FlowItems. Question – can <TestName> parameter of "FlowItem" be another flow, or can it only be a TestMethod name?
- FlowDef (equivalent to our entry points). OTPL appears to have a somewhat richer set of entry points than we've considered. We may want to add some of these to STIL .4

Tony feels that between OTPL, the original .4 syntax draft, Stylus, and the two variants we've developed (Don's and Ernie's), we should be able to stitch together a cohesive language.

Dave: Tony, what about alignment of OTPL to STIL?

Tony: Similar to STIL in many places. Two issues – alignment with STIL in general, and alignment with our .4 direction. On the first issue, there are deviations from existing STIL in OTPL. I don't understand the reasons for the deviations. I don't see any additional benefits for the (OTPL) alternative. On the flow itself, I see a fair degree of commonality between OTPL and .4.

Will OTPL adopt STIL, or simply have similar capability? If they're different, remember that the details can kill you - you can be 90% (or 99%) in alignment, but from a user point of view, you'll spend all your time dealing with the 10% (or 1%) of differences.

Dave: Comments?

Yuhai: Why did Advantest not use STIL? General answer was that STIL was still developing, and couldn't cover all areas which needed to be covered. Advantest is still considering STIL, as it further develops, but for the immediate future, the Advantest tester language will be OTPL.

Dave: Collaboration on .4 should hopefully lead to fuller cooperation and alignment on other language constructs.

Tony: Yes. There are a number of areas which we could fairly quickly agree on (i.e., list of entry point names, using a name (OTPL and .4) for flow nodes or a number (Stylus)).

Dave: Tony, what are your next steps (to get the questions posed and answered)?

Tony: Don't want to get into a complete review of OTPL. Will capture questions about the flow-specific issues in the STIL-EBNF'd OTPL description document. Will update that document and distribute. Other WG members should also read through it and add any questions/comments of their own.

Dave: Don't use STIL.4 reflector to distribute either the OTPL document or the STIL-EBNF'd version. Want to keep this distribution within the working group, per collaboration agreements with STC. As each of us go through that document and develop questions and comments, it would be helpful to send them out by email to the entire WG (do NOT use the STIL.4 reflector!).

Doug: The translation helps quite a bit – makes the language easier to read and understand at a glance.

Tom: Yuhai, were STIL constructs migrated into OTPL at customer request (Motorola, for one, was encouraging Advantest to base their next-generation tester language on STIL), or was the STIL influence planned prior to any customer input on this issue?

Yuhai: I believe that it was planned before any customers had expressed a desire for a STIL-based language. There was also a debate on whether to use pure STIL as the native language, or develop one which is not STIL, but which contains much of the STIL influence. The decision (obviously) was in favor of the latter.

Dave: OTPL appears to be a container (of various language constructs) but also aligns heavily with C++ in order to implement the tests. That's a significant difference from our approach; we anticipate an interface into the actual tests which doesn't rely on the structure or details of a particular language (i.e., C++). Do we expect any difficulties in alignment (with OTPL) because of this?

Tony: Yes, I do. OTPL's decision was (apparently) that the implementation language would be C++, so it's OK to build some C++-isms into the OTPL syntax. STIL syntax was designed to have no bias towards an implementation language (leaving that choice up to the implementers). For example, OTPL "User Vars" are C++ definitions, whereas in STIL, we have described a more limited set of variable types.

Dave: We want STIL to remain at a level of abstraction somewhat above that of the language(s) used to implement it.

Tony: I would expect that STIL will be a subset of the total capabilities of OTPL.

Ernie: We must have a clear mapping of constructs from one language to another. And I feel that, from a test programming language perspective, the variable handling capability of STIL is somewhat incomplete.

Dave: Do we want, as a language design goal, to remain abstracted above the implementation language? (General consensus on this point was yes).

Ernie: In my view, STIL is a language which describes how to test a device, not a language which runs on a tester (i.e., the phase1/phase 2 approach). Phase 1 is a descriptive language, whereas phase 2 is a language which can be executed on a tester. As we implement the phase 1 language, I thought our consensus was NOT do anything which would get in the way of the eventual development of the phase 2 language. Remember, phase 2 is a much longer term goal, and even when we get there, there will always be legacy systems which will not be native STIL language testers, but which may want to translate STIL to their native language (EDITOR'S NOTE: Ernie didn't actually state this, at least this time, but I know he's mentioned this before, and I think that most of us probably agree with this statement).

Dave: If we can provide a clear, unambiguous information constructs, implementers (i.e., OTPL) can take that description and do the implementation in C++; we can use the language syntax and description as the boundary between the descriptive language (common to many testers) and the executable language (specific to a particular tester). Don, as one who's in the middle of this issue, any comments?

Don: Inovys is looking for an executable language (i.e., jumping directly to phase 2). There's certainly rationale for having a descriptive-only language, using translators to move back and forth to and from STIL and the tester's native language, but Inovys wants an executable language.

Tom: Want to use STIL to move from one tester to another seamlessly and in a short time(to help with test floor load balancing, etc.). Currently, Motorola spends a significant effort internally to develop translators; it's very time-consuming to develop and debug the converters. There are also some commercial tool vendors which provide products to do this job. But Motorola would like to use STIL as the common language, and we're asking ATE vendors to provide STIL readers/writers.

Dave: That requirement is a good rationale for keeping STIL abstracted above the specifics of a particular tester, and let the implementation deal with those specifics.

Tom: If we could get 80% portability, that would be tremendous. The direction Motorola is heading is to have a common language. We know it's going to be difficult to get there, but the benefits make that effort necessary.

Doug: At IBM, we automatically generate test program from a simple language (TPGL). That language is built within C++ (C++ with additional constructs). Those constructs get preprocessed and turned into C. The whole process of generating a test program is part of a C++ program; executing that program actually generates the test program. The front end (that does the generation) is tester-independent. It contains the flow constructs, which are fairly simplistic at this point. At the tester, the flow defined in TPGL gets mapped into tester-specific language constructs. (EDITOR'S NOTE: Keeping the flow constructs simple aids in being able to implement them on a variety of specific testers).

Dave: Would you rather have a descriptive language(i.e., STIL), or one that contains some implementation specifics (i.e., OTPL)?

Doug: Would probably lean toward the descriptive language, and keeping them as simple as possible, while providing the needed capability.

Ernie: While both are useful (descriptive vs. executable), the issue is not one vs. the other, I have an immediate use for phase 1. Ideally, the phase 1 language would be a subset of phase 2, which means that we shouldn't do anything in phase 1 that would get in the way of phase 2. But I'd like to focus on phase 1 (descriptive language) because it would take too long to get agreement on all the issues for a phase 2 language – and if we can get phase 1 done, it has use and value independent of phase 2.

Dave: Entry points – some could be fairly tester-aligned, others could be general(?). We should strive to cover the majority of needs for the as many types of ATE languages as we can. Others may be implementation-only issues – for instance, the pre- and post-actions at different levels – on any given system, only a subset will be used.

Ernie: That's a major concern, and a reason why we're getting a fairly rich set of capabilities in the language. I'd like to represent as much as possible in STIL for as many different types of testers as possible, so that the program information can be manipulated in STIL as much as possible before actually moving to the target test program language. In other words, it would be useful to be able to, within STIL, reorganize (and manipulate) your test program data in a way that gets fairly close to your target language, so that there's only a small leap make when actually generating the program in the target tester language. Otherwise, the programs in the target language can become more complex and unwieldy, difficult to understand and difficult to maintain.

Dave: I agree. That argues for making STIL a descriptive language, rather than an executable language (i.e., the phase 2 issues). To conclude, let's reiterate some action items, and try to get some email correspondence going.

Action Items:

Tony: will update the OTPL EBNF document, annotating with questions where they arise (all in the WG should feel free to review the document and annotate with their own comments/questions.

Ernie, Dave: Update the conceptual diagrams with more information on tests and test types.

Dave: Clean up and update the feature comparison matrix.

Don: Update the Stylus EBNF and distribute.

Jim, Ernie: Represent Ernie's model (syntax.asc) in STIL-EBNF format.

Updates:

Yuhai: Update on multi-site testing within Advantest and/or STC. No STC meeting last week, but did talk with Advantest design group. Could not get schedule for when Advantest will deliver multi-site info to STC (not complete yet). In the big picture, however, Advantest is planning to multi-site testing capability to its customers. Most likely, STC will get that information before October, but exact time frame is not yet established. Once STC receives it, it should shortly thereafter be available to STIL .4 WG.