P1450.4 meeting minutes - 08/22/07

Attendees:Ajay Koche, Bruce Parnas, Jim O'Reilly

Not present: Jose Santiago, Doug Sprague, Ernie Wahl

Agenda:

- Preamble:
    - Record Meeting (*2)
        - To listen to the meeting recording, do the following:
            - Call the (US) dial-in numbers 1-877-421-0003 (toll free) or 1-770-615-1374 (toll)
            - Enter the passcode code 747464
            - Once dialed in with the proper access code, enter *3 (star 3)
            - Then enter the file number 25901901 for this conference (this number will change each week).
            - Press 1 to listen to the conference.
    - IEEE Meeting Preamble (No discussion of proprietary information).
- Discuss change of time/date for meeting.
        - Starting next week, Jim has an ongoing conflict with the Wednesday, 10:00 am Pacific/1:00 pm Eastern meeting time. Need to find an alternate time which works for all WG members. Jim to poll WG members for alternatives, and propose new time.
- Discussion of how TestBase is provided to reader/writer.
    - Discussion of semantic rules which specify when TestBase is used, and when another type (TestType or FlowType) is inherited.
        - Initial rule was that if "Inherit <myType>" was used in TestType or FlowType, the named type would be inherited, and if "Inherit <myType>" was NOT used, then TestBase would be used instead.
        - Some WG members prefer that the "inheritance" be explictly stated, even when using TestBase. But we don't necessarily want to say "Inherit TestBase", since we're trying avoid having people think in terms of inheritance (an advanced concept) for simple cases. It seems better to allow people to think of TestBase as a set of defaults, instead of a base type.
        - If explicitness is desired (which is not necessarily a bad idea), perhaps the statement "UseDefaults" or "UseTestBase" (or "Use TestBase") could be used in place of "Inherit TestBase".
    - Discussion about the rules for user-extended TestBase.
        - The rules currently state that if the user provides their own definition for TestBase, it must include all the elements of the dot4-mandated TestBase, plus the user's added elements. The reader or writer must check that the user's TestBase includes the standards-mandated set of elements – which implies that it must know what those elements are. There was general agreement that a reader/writer would probably NOT hard-code that knowledge, but rather read it from a file. Does that imply that there would be two TestBase structures read by the reader/writer? The first block would contain the standard elements (to be used for error checking), and the second would contain either:
            - The standard elements plus the user-defined elements, or
            - just the user-defined elements which are added to the standard elements (as suggested by Ajay, as a way to aid in supporting possible different versions of TestBase over the years – see below).
    - Discussion about user-provided TestBase (for v1), and how that might work if the standard-mandated TestBase ever changed to v2 (probably containing more elements than the v1 TestBase).

    - If the user extended TestBase restates all the v1 elements in addition to the user-provided elements, that could cause a problem if a v2 standard TestBase was used instead. The user would have to then modify their user-provided (v1-based) TestBase to include the additional elements that v2 TestBase now provides.
    - On the other hand, if the user's TestBase includes ONLY the additions to the standard-provided TestBase, then it makes such an update easier. The user's TestBase automatically gets the standard-provided TestBase elements (either v1 or v2), and adds to that what the user needs. The drawback is that the user's TestBase now doesn't explicitly state everything that TestBase contains, which forces the user to look elsewhere to determine the entire set of elements.
  - Jim to consider these issues, and come up with proposal for dealing with them. The use of the STIL statement

    STIL IEEE_1450_0_IDENTIFIER {
    (EXT_NAME EXT_VERSION;)+
    }

    with an extension identifier for the flow extension could be useful for this purpose. As an example, the statement

    STIL 1.0 {
    Flow 2008; // Let's be optimistic and say it will be approved in '08!
    }

    would be included in the input file. If there were later revs of TestBase (dated, say, 2010), the reader or writer would have the responsibility to select the proper TestBase against which to compare the user-specified "standard" TestBase elements.

- Syntax of variables blocks with TestTypes and FlowTypes.
  - Should we use the following:

```
Variables myTestVars {
    Int A;
    String myString { InitialValue = "myString"; }
}

TestType MyTestType {
    Parameters {
        }
        Variables myTestVars;  // Vars defined outside block, instantiated inside.

        . . .
        . . .
}

Test MyTestType MyTest;
```

    as specified by dot1, or

```
TestType MyTestType {
    Parameters {
        }
        Variables {  // Vars defined and instantiated inside block
            Int A;
            String myString { InitialValue = "myString"; }
        }
        . . .
        . . .
}

Test MyTestType MyTest;
```

    - The first version is the form which follows from the dot1 syntax – but has the drawback that variables are defined outside the block in question, but instantianted inside (though since the variables are “instantiated” in a type definition, they do not actually instantiate until the TestType or FlowType definition itself is instantiated). Some on the WG prefer to have an additional syntax which allows defining the variables completely inside the block in which they’re used – in effect, making them local variables visible ONLY within the scope in which they’re defined/declared. The current syntax supports both forms.
    - In addition, the two forms could be of use to distinguish between a type’s private variables (invisible to derived types) and protected variables (visible to derived types). If we determine that such behavior is needed, variables declared with the first form would have a “protected” attribute (visible within the scope of the type itself and derived types; invisible otherwise), while variables declared with the second form would have the “private” attribute (visible ONLY within the scope of the type itself; invisible within the scope of derived types and otherwise).
    - Bruce brought up the question about whether or not variables instantianted within a type block using form A could be accessed by name outside the block. Since the name fo the variables block is known, and ultimately, the name of the Test or Flow block within which they’re instantiated is also known, it would be possible to allow access using a <block_name>.<variable_name> notation. In general, we’ve chosen to keep local variables for a block local, and to NOT allow access outside the scope of the block. If you want a variable to be accessible to the outside world, declare it as a parameter and (optionally) assign it an initial value (if an initial value is not specified, each instantiation would need to specify the initial value if the variable was to have a value other than “undefined”).
- Flows within a test.
    - Within the TestType block, actual test execution is specified by the statement

      (**TestExec; | FuncExec** *func_stmt* **| FuncExec** { (*func_stmt*)* } **|** *flownode_stmt***+**)

      Bruce was asked about the need to support execution of flows within Tests (based on his use model, it seems unnecessary). The rationale was to support Envision- and Stylus-style MicroFlows. We need to examine Envision and Stylus usage of these to determine how it could (and should) be supported.

Next meeting:

- Next meeting 08/29/2007.

For reference STIL .4 information can be found at the IEEE STIL website:
http://grouper.ieee.org/groups/1450/ (select the P1450.4 link from the table) or use the direct link
http://grouper.ieee.org/groups/1450/dot4/index.html