STIL P1450.4 Rosetta Stone

Conceptual Model Rev K, 9/28/05	Syntax Document Rev D16, 12/05/05	Comments
Test Program	TestProgram	
EntryPoint	EntryPoints	
TestFlow	TestFlow	Type definition – must be instantiated in order to be used. See Note 1.
FlowNode	FlowNode	
TestMethod	TestModule	Type definition – must be instantiated in order to be used. See Note 1.
TestBase (sec. 2.4.1)		
Defaults		
	TestMethodDefs	See note 2.
TestObject	TestInstances	Creates a named instance of TestModule or TestFlow, which can be executed by an EntryPoints or TestExec statement
TaskNode (sec. 2.7)		Not needed. Can be synthesized from existing elements.
DecisonNode (sec. 2.7)		Not needed. Can be synthesized from existing elements.
ObjectRef/ModuleRef	TestExec	Implied in the conceptual model definition of TestMethod as the lowest level block. See Notes 2 and 3
PreActions	PreActions	
PostAction	PostActions	
SkipPath	Bypass	
Arbiter	*boolean_expr* of ExitPort block	See Note 5.
PassActions/FailActions Blocks	ExitPorts	For TestModule/TestFlow See Notes 4 and 5
ExitActions Block	ExitPorts	For FlowNodes See Notes 4 and 5
BinMap	BinMap	
BinNode	BinDefs/Pass/Fail/Bin	

Note 1: Both TestModule and TestFlow represent type definitions. They cannot be executed directly; they MUST be instantiated into TestInstances in order to be executed. Instantiation can be explicit, via the TestInstances statement, or implicit, when either type is used in a TestExec statement.

In the first case, a named instance of the desired type is created; this type remains in existence even after it's been executed; it's attributes can be queried (though we haven't yet defined WHAT attributes it has which can be queried!).

In the second case, when a TestMethod or TestFlow type name is used in the TestExec statement, an anonymous, unnamed (in-line, dynamically created) instance is created; however, it's lifetime is only long enough to return information to it's caller. Once the caller takes control back, the object ceases to exist. Therefore, it cannot be queried for its attributes after the caller resumes control.

Note 2:

The conceptual model contains an implied TestExec, by virtue of the TestMethod representing the lowest-level block. The assumption is that TestMethod represents an object type which contains and Execute( ) method. Once an object of type TestMethod is created, the execution of that object results in the object's Execute( ) method being executed behind the scenes (between the PreActions and PostActions) by some unspecified and implementation-dependent mechanism.

In contrast, the syntax document specifies a block currently called TestMethodDefs, which describes a function or method signature (name + parameter lists) which resides in an externally-loadable library. It's this specific function which can be executed by the TestExec statement or the EntryPoints statement.

Note 3:

The conceptual model contains a few figures which refer to an entity called ModuleRef or ObjectRef. See Figs. 2 (section 3, p6) and 3 (section 4, p8). These figures show that from a FlowNode, an TestObject can be executed. This object can either be previously created and named, or can be created in-line, dynamically. In the syntax document, the TestExec keyword fills exactly the same role – it can execute a previously-created and named object or an in-line, dynamically created object.

Note 4:

For clarity, I think the ExitPorts block as used in the TestModule and TestFlow should probably be renamed as ExitActions, ExitPath, ReturnActions, ReturnPath, or ReturnState to distinguish it from the ExitPorts of the FlowNode. The reasoning is that the FlowNode can branch, so we have multiple paths out of a FlowNode. In contrast, the TestModule and TestFlow can execute different sets of ExitActions depending on the outcome of the test, but neither the TestModule nor the TestFlow actually branch. The outputs of the one or more ExitActions blocks reconverge before the TestFlow or TestModule passes control back to its caller. To me, and to probably many people in the WG, the term "ExitPorts" implies branching, and so I feel it should be used ONLY for FlowNodes.

Note 5:

(**ExitPorts** {
  (*boolean_expr* { (exit_port_stmt)* })* // Arbiter/ExitPorts
}) // end ExitPorts

In the example above, the line "(*boolean_expr* { (exit_port_stmt)* })" can be repeated an arbitrary number of times. Each line represents a unique exit port. The boolean expression represents the arbiter – generally, the set of boolean expressions is set up so that only one will evaluate as true. However, the expressions are considered in the order in which they appear; if more than one can evaluate as true, then the first one which evaluates as true will determine the ExitPort taken.