Testing Interoperability of SML & SML-IF Implementations

Change Log:

Date	Person	Remarks
4/4/2008	Kumar Pandit	Created initial draft.

1. Overview

The SML working group is required to have at least one implementation of each SML & SML-IF required feature and preferably two inter-operable implementations of validating SML-IF consumers in order to meet the Candidate Recommendation (CR) exit criteria [link to process step]. This document defines the approach adopted by the SML working group to meet that goal.

Comment: Might still need to tweak the term, do not have a specification handy. Important point is that we are testing implementations that not only enforce SML-IF's constraints, but that also test the SML-validity of the interchanged model.

This is defined in the following sections:

Comment: What? See nothing being defined.

1. Test packaging
2. Test storage and organization
3. Test execution
4. Analyzing test results
5. Test Cases

2. Test Packaging

Each test involves processing a set of schema/rule documents and instance documents. There are two basic approaches to package these documents.

1. Keep each document in its own file and have a test-description file per test that points to files involved in that test.

   Although this method saves disk space by eliminating duplication of files, it has the following disadvantages:

   a. Change to a single file potentially affects multiple tests.
   b. The test-description file must store information about file category (definition/instance/rule), file URLs (aliases), schema-completeness, rule bindings, etc. In other words, this file must define and use syntax that is already defined for the SML-IF files.
   c. Sending a complete test case to someone is complicated because one needs to carefully copy all required files and the associated test-description file.

Comment: (a) and (b) could be seen equally as adv and disadv, depending upon your intent. If you want to use the same file in 15 test cases, and discover a bug in it, fixing it 1x is an adv.

Comment: And the structure, e.g. which files are definition documents, etc

Deleted: 4/15/2008 7:07 PM

d. A test harness must have additional code that understands how to use the information in the test-description file.

2. Package all files required for a test in a single SML-IF file.

   Although this method requires more disk-space than the previous method, the actual amount of disk space additional disk-space is insignificant compared to current disk sizes. This method has the following advantages:

   a. Changes to each test are localized to that test.
   b. No special syntax needs to be invented because SML-IF already defines it.
   c. Sending a complete test case to someone is easy because one needs to send only 1 file.
   d. An SML-IF consumer already knows how to process an SML-IF file therefore that part does not have to be implemented by a test harness.

3. Combine the attributes of the previous two methods: source the test files using SML-IF with locators, and create an XSLT stylesheet to replace each locator with its fully embedded equivalent. Widely available platform-independent tools like ANT can be leveraged for the transforms.

   This method does not force either extreme of input file replication when files are used in multiple tests (as might commonly be the case for valid models).

   a. Input files can be re-used as long as they are truly intended to be re-used.
      1. If a change needs to be made to one for a particular test case, a new clone is made, given a new name, and its content changed.
      2. if a change needs to be made to one to correct a bug effecting multiple test cases, it is made once.
   b. No special syntax needs to be invented because SML-IF already defines it.
   c. Sending a complete test case to someone is easy because one needs to send only 1 file.
   d. An SML-IF consumer already knows how to process an SML-IF file therefore that part does not have to be implemented by a test harness.

In view of the above, each SML test case is packaged in a single SML-IF file. There is only one exception. Testing non-embedded documents pointed to by a locator element requires multiple files per test. However, since support for the locator element is optional, that test is not included in interoperability testing.

3. *Test Storage and Organization*

Test cases and associated files are stored on W3C servers so that they can be accessed by people who need them. We anticipate having less than 200 test cases that cover SML & SML-IF, based on known implementations' testing history. This allows the test cases to be stored in a relatively flat directory structure. They are stored under the sml directory in the cvs repository as shown below.

Deleted: must be written

Comment: The test plan should assume that multiple implementations of all features will be present, even if that criteria is not met at the moment. We don't want to be forced into re-jiggling the whole plan just because a new complete implementation shows up during CfI.

Deleted: There are

Deleted: 4/15/2008 7:07 PM

- SML
  - Test inputs
    - testsForRequiredFeatures
    - testsForOptionalFeatures
    - documentation
  - implementation characteristics
    - implementation 1
    - implementation 2
    - …
  - Test outputs
    - Impl1
      - Required
      - Optional features
    - Impl2
      - …
    - …

The testsForRequiredFeatures directory has the SML-IF files that contain tests for the required features. The testsForOptionalFeatures directory has the SML-IF files that contain tests for the optional features. The documentation directory has test documentation, including this document.

4. Test Execution

How the test case files are supplied to a test harness and how the test harness is invoked depends on each implementation. This document does not define requirements for the following:

1. Hardware configuration
2. Operating System or its version
3. Test execution behavior of any test harness.

5. Analyzing Test Results

As mentioned earlier the SML working group is required to have at least one implementation of each SML & SML-IF feature and at least two inter-operable implementations. The process of determining whether all features are covered by the test cases cannot be automated. The working group must determine this based on a review of the test cases. However, the inter-operability can be determined in an automated fashion.

For the purpose of this test plan, two implementations are said to be interoperable if they produce identical model validation results for each test case that tests a required feature of SML and SML-IF. Two implementations are allowed to produce a different model validation result for a test case that tests an optional feature.

Comment: I don't really care if we keep this stuff in CVS or not, just easier to show it here as a coherent whole

Comment: We can however do things like name the features to make this job easier.

Comment: Assuming either the requirements of the optional feature allow this, or they do not declare the same level of support for the feature (if their levels of support are the same and the feature completely prescribes the output, they are comparable). Conceptually each testcase determines a unique (binary) output depending upon the set of supported features, i.e. R = f(f1, f2, … fn), Implementation = { f1, f3, … }. If two implementation feature sets are related such that one is a superset of the other, their outputs should be comparable if the features prescribe output.

Deleted: 4/15/2008 7:07 PM

Comparing test results is trivial where the results indicate a valid model. Comparing test results where a model is invalid is not so easy. This is because a model could be invalid because of a number of reasons and the SML & SML-IF specifications do not define an inter-operable way to encode or compare the reasons. This is because of the following:

1. The SML specification does not define the order in which model validity assessment steps must be carried out. One implementation may evaluate id constraints before Schematron constraints and some other implementation may do it in the reverse order.
2. The SML specification does not define whether model validation must stop at the first error and whether all possible errors are returned to the invoker. This combined with the previous item can result in two implementations producing two different results for the same invalid model.
3. The SML/SML-IF specifications do not define specific error codes or error messages that represent an error condition. This means that even though two implementations find the same error in a model, they may produce different error codes/messages. There is no inter-operable way to compare them.
4. An SML-IF consumer that validates the interchange model might choose to check conditions imposed by SML before, after, or concurrently with those imposed by SML-IF. The SML-IF consumer might call an OTS SML validator, for example.

As a result of these difficulties, this test document does not require granular comparison of test results between two different implementations. The result comparison is limited to comparing model validity as pairs of Boolean values.

Each participating implementation must pass each test that tests a required feature. A test is said to pass if the actual model validity result, expressed as a pair of Boolean values, is identical to the expected result. In order to facilitate comparison of results, a consistent output format is defined that can be used to declare the expected and actual result for each testcase. For example,

```
  <Actual>
    <implName> string </implName>
    <testName> token? </testName>
  <Result>
    <validity sml="true" smlif="false" />
    <errors> <!-- seq of 0:n copies of (1 code + 0:m messages) -->
       <code> string (probably name of feature violated)
              ideally, an open string enumeration </code>
       <message> string that might be useful for comparing
                 results across implementations </message>
    </errors>
  </Result>
  </Actual>
```

The name of each test file includes the the expected result. This obviates the need to maintain a separate test metadata file. For example,

Comment: This could be read to mean we are testing only SML validity, not what SML-IF adds. They are conceptually separate functions with independent outputs (I can put any in/valid SML model into an SML-IF doc correctly or incorrectly).

Comment: What does this refer back to? reads as if it wants to explain why we have no error codes.

Deleted: a

Deleted: test name followed by

Deleted: 4/15/2008 7:07 PM

id-constraint-KeyMissing-invalid-seq#.xml
ref-dangling-valid-0001.xml

An implementation report can be generated, if desired, by a transformation process like XSLT that iterates over each test case, pulling expected results from the model identity section, and over each implementation's output file, combining the results into a single row in an HTML table. E.g.:

Test name	Expected result	Implementation 1 result	Implementation 2 result	All match?
ref-dangling-valid-0001.xml	SML: valid SML-IF: valid	SML:valid SML-IF:invalid	SML:valid SML-IF:invalid	SML: yes SML-IF: NO

Comment: I would start with specification shortname

Comment: Looks a lot like a feature name. Naming the features would go a long way towards making this stuff understandable. Would NOT want to put that in normative specifications (could in theory, but seems like the wrong tradeoff between perfection and progress)

Comment: We need a seq number to allow for more than one test case per feature.

Comment: Don't care if we use long-ish flat names, a dir structure, or some mixture.

Formatted: Indent: First line: 0.5"

6. *Test Cases*

As defined earlier, model validation results are compared as a pair of Boolean values. Test cases are written such that they focus on a single issue at a time and consequently they result in a single model validation error. This avoids ambiguity in cases where a model may be invalid due to multiple reasons.

Comment: Not really. In fact, it makes the comparison less useful. If you declare a model invalid for "the right" reason and I generate a completely spurious error, we pass (as this is written).

The following sections lists all test cases used for interoperability testing. They are divided in 2 parts,

1. Tests for required features:
   Each participating implementation must pass all tests in this section.

2. Tests for optional features:
   Not all implementations support optional features, therefore the behavior of an implementation cannot be reliably compared with that of another when an optional feature is involved. An implementation may silently ignore the presence of an optional feature or it may produce a fatal error or it may produce a warning if required by SML or SML-IF specification.

Comment: Unless both implementations assert support for it; then it may be reliably comparable.

General form of the test case names: concatenation of the following, using "-" as a separator, followed by ".xml", each containing a single SML-IF testcase using ONLY locators

- Specification short name (sml or smlif)
- Feature name
- Condition (inv or val)
- Sequence number (for multiple test cases where others held constant)
- 'output' if the file represents the result of a single implementation executing the test case (i.e. filenames come in logically related pairs, an input [string].xml and the corresponding output [string]-output.xml).

Tests for Required Features

tbd

filenames: SML-features.xml, SML-IF-features.xml

Comment: Since the filenames (at least, and I think basic organization too) imply a list of named features, suggest we formalize this as a file

Deleted: 4/15/2008 7:07 PM

Each can be separately maintained by 1-2 working group members.

Each specification file contains (example) sequence of

```
   <feature ID="sml-ref-dangling”>
      <name> token? </name>
      <description> string </description>
   </feature>
```

Each test case’s model identity contains a list of features needed to get a predictable result (example)

```
  <expectedResult>
    <featuresRequired>
      <featureList sml:ref="true">
         <sml:uri>pointer to required features</sml:uri>
      </featureList>
      <feature sml:ref="true"> sml:uri to an optional feature supported
    </featuresRequired>
    <testName> token? </testName> <!-- would be nice not to require this in
test case itself -->
  <Result>
    <validity sml="true" smlif="false" />
    <errors> <!-- seq of 0:n copies of (1 code + 0:m messages) -->
       <code> string (probably name of feature violated)
              ideally, an open string enumeration </code>
       <message> string that might be useful for comparing
                 results across implementations </message>
    </errors>
  </Result>
  <expectedResult>
```

Formatted: HTML Preformatted

Formatted: HTML Preformatted

Tests for Optional Features

tbd

Deleted: 4/15/2008 7:07 PM