SML - Editor and Validation Scheme

Change History:

Name:	Date:	Revised Sections:
Ali Mehregani	October 16th, 2006	Initial Creation
Ali Mehregani	October 25th, 2006	Modified the document based on Valentina Popescu's review.
Ali Mehregani	November 13th, 2006	Modified the document based on Harm Sluiman's suggestions in an e-mail received on Oct 31st/2006.
Ali Mehregani	November 15th, 2006	Incorporated Valentina Popescu's comments
Ali Mehregani	January 3rd, 2006	Updated the last page with tasks that will need to be done to support SML unit validation and provide editing capabilities.
Ali Mehregani	January 16th, 2006	- Removed the concept of having a model folder - Added new tasks - Created screen mockups of the export and import wizard - Added the concept of a meta-data when importing an SML-IF document

0.1 Terminologies/Acronyms

The terminologies/acronyms below are commonly used throughout this document. The list below defines each term:

Term	Definition
SML	Service Model Language
SML-IF	Service Model Language – Interchange Format
Genic	A genic is one of two types of documents supported by SML- IF. A genic document is used to describe the structure and constraints of the model.
Phenic	A phenic is one of two types of documents supported by SML-IF. A phenic document describes a model entity.
SML validation	Validating an SML model in accordance with the SML specification.
SML-IF validation	The process of validating an SML-IF document in accordance to its structure and validating its contained SML documents.
Template document	An SML instance defining a common pattern that can be re-used and adapted in different domain models
Domain model	The root of an SML-IF document. Contains a set of phenic and genic documents
Resource domain	A set of genic and/or template documents that can be used to build a domain
Type editor	A tool that can create genic and template documents
Model editor	An editor that can create domain model instances, based on a set of predefined templates
SML-based project	A project on the file system containing documents used to build a Resource domain

0.2 Purpose

This document outlines the general structure of a project containing an SML model. It describes the operations that can be performed on an SML-based project, including validation and import/export operations.

The document describes a proposed implementation of a tool that allows validation and creation of Domain models.

Project Structure

An SML model can be stored in any arbitrary folder structure under a project. The structure can arbitrarily be nested based on the user's preference. Figure 0.1 illustrates the structure of a project consisting of two folders at the project level: "software" and "hardware".

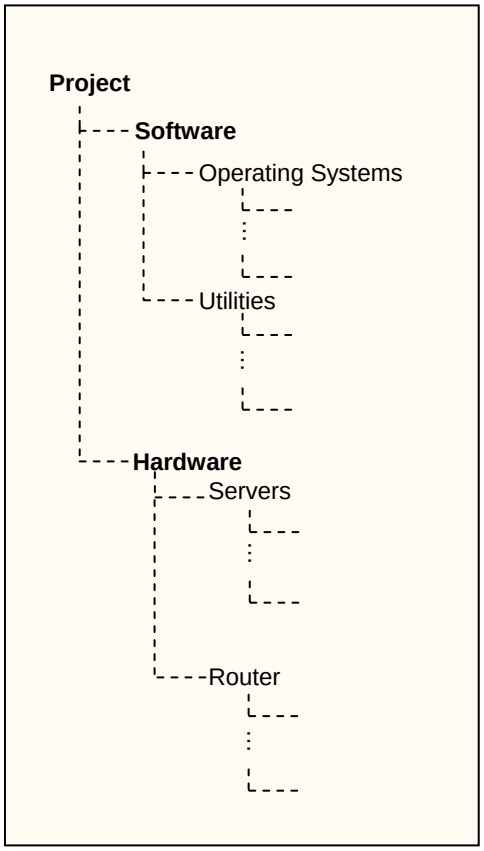

Figure 0.1 – Project structure example

The model units are the XML, schema, and schematron files making up the resource model. Each unit has an associated editor that can be used to modify the file.

Operations

An SML-IF document can be formed from a project by selecting export from the context menu of one or more resource that appears under a project. The operation will determine the genic or phenic documents based on a registered content type that describes each type of the document. The user will have the option of changing the document type (i.e. genic/phenic) if a unit's type is incorrectly assigned. Figure 1.2 and 1.3 are screen mockups of the first and second page of the export wizard.

The export operation will be integrated with Eclipse's export wizard under the category: "Service Model Language".

The user can also import an SML-IF model into its units using the import operation. The import operation will simply divide up an SML-IF document into individual genic and phenic documents and store them in a desired target folder. See figure 1.4 for a proposed import wizard. The import operation will be integrated with Eclipse's import wizard under the category: "Service Model Language". Ideally an import operation followed by an export operation should generate a file that is semantically equivalent to the original file imported. Unfortunately there is some loss of data when an SML-IF document is broken into its respective SML model units. For example, the aliases associated with a document or schematron rule bindings are not preserved in SML model units. To preserve this information, a meta-data file is generated with every SML-IF document import. Using the meta-data, the user can perform an export operation that will result in a file that is semantically equivalent to the original SML-IF document imported.

Validation can be done at a project, folder, or a unit level. To validate an entire model the user can select validate from the context menu of a project. Alternatively the user can validate a subset of the model by selecting validate from the context menu of a parent folder or a set of model units.

Validation

Validation is done in three phase:

1) XML Validation
2) SML schema extension validation
3) Schematron validation

Each phase must be able to execute independently according to the user's preference. A complete validation run will apply each phase sequentially. A model is considered to conform to SML standards if and only if the three phase of the validation process is considered successful.

The XML validation phase is required to validate the interchange format document to ensure that it conforms to the SML-IF schema. It also needs to validate the phenic documents based on the associated schema documents that they are expected to conform to (i.e. in the genic section or as specified by their schema location attribute). The genic documents will also need to be validated to ensure that they are well formed. To summarize, this phase will:

1) Ensure that the SML-IF document is valid and well formed
2) Ensure that all genic documents are well formed
3) Ensure that all phenic documents conform to their associated genic documents

The SML schema extension validation phase will ensure that the documents conform to the schema extensions as specified by the SML specification. This includes the following extensions:

1) sml:acyclic
2) sml:targetElement
3) sml:targetType
4) sml:key
5) sml:unique
6) sml:keyref

In addition to validating the additional constraints that SML adds on top of Schema, this phase will also validate all inter-document references using equivalence of URIs as defined by RFC 3986.

The Schematron validation phase will ensure that all assertions and reports are executed against each document. This validation step will depend on the Schematron phase that the documents are validated against.

1.0 Requirements

1) Allow the separate execution of each validation phase as described in section 0.2. The user will have the option to either perform a full validation (involving all three steps) or perform each validation step separately.

2) Provide an editors for each unit of the model (XML, schema, schematron) and SML-IF documents.

3) Ability to import SML-IF documents into a project and export SML model units into an SML-IF document.

4) Each phase of the validation scheme must be customizable. Every stage of each phase that is implantation-based dependent must also be customizable. The following is a list of tasks that each contributor may want to implement slightly differently:

    a. Validating SML Schema extensions: although the semantics of each extension is expected to be the same, contributors may wish to provide alternative algorithms that maybe more efficient in validating some extensions. The validation for extension must be decoupled to encourage re-usability where possible and customizability where desired.

    b. An easily pluggable framework should be provided to allow customization of actions for each available Schematron hook:

        i. process-prolog
        ii. process-root
        iii. process-assert
        iv. process-report
        v. process-pattern
        vi. process-rule
        vii. process-message

5) Users should have the ability to validate an SML-IF document in headless mode outside the Eclipse workbench. An equivalent mechanism should be available for contributors to customize the validation scheme in headless mode. A set of flags will be available to customize each phase of the validation scheme.

1.1 Use Cases

This section describes the steps involved in accomplishing common use cases.

Use case 1: Exporting model units into an SML-IF document

- Select the of folders or model units that is to be included in the interchange document
- Right click and select export
- Expand the “Resource Modeling” category
- Select “SML – Interchange Format”
- The user has the option of specifying a name, display name, description, and a base URI.
- A wizard page will display the units that are to be included under the document. Each unit will be grouped either under the genic or the phenic section. The user will have the option of freely moving a unit from one document type to another.
- Click finish to create the document
- The document is created and its associated editor is opened

Use case 2: Adding/Importing documents to an SML-IF instance

- Open the editor of an existing SML-IF instance
- Select the “Documents” page.
- The user can click the “Add” button To import a new model unit to the SML-IF instance

Use case 3: Validating an SML-IF instance in Eclipse

- Select a desired document to be validated.
- Right click the document and select Validate > “Validate entire document”
- Alternatively, the user can open the editor > select the Overview page > and click the validate entire document link.
- If the Schematrons associated with the document contains multiple phases, then the user is prompted with a dialog to select the set of phase that they wish to validate the document against.
- All errors/warnings/information are reported in the problems view. If there is no warning/error/information to be displayed, then a message dialog appears to notify the user that the document is valid and well formed. The only elements that fall under the information category are satisfied report conditions in associated schematrons.

Use case 4: Navigating and fixing a problem

- The user can double click an error/warning entry in the problems view to navigate to the SML-IF editor and the appropriate line number that is in violation.
- The line number of the SML-IF document will be marked with an error/warning icon (similar to the Java editor).
- If a resolution is available the user can single click the icon to select the appropriate resolution from a given list.

Use case 5: Validating an SML-IF instance outside the Eclipse environment

- Given an SML-IF document, the user can validate using a single JAR file that will be available from an Eclipse plug-in:
    - Java –classpath *<path to sml-validate.jar>*
      org.eclipse.cosmos.rm.provisional.validation.Validate
      <path to the SML-IF document>
    - The following set of program flags can be used to customize the validation scheme:
      -xml *<A class that will perform the XML validation>*
      -sml *<A class that will perform the SML (schema extension) validation>*
      -schematron *<A class that will perform the Schematron validation>*

      The validation classes of SML and Schematron will be required to defined the scope of the validation (see section 1.3 for more details)

- By default, the output will appear in system output or can be redirected to a file. Alternatively contributors can register a different Schematron validation class that defines the source of output for assertion/report messages.

1.2 Graphical Layout

As mentioned earlier the following resources will have an associated editor:

- XML documents
- SML Schema documents

- Schematron documents
- SML-IF documents

The editor of the first three resources will be a color-based text-editor. The SML-IF editor will be a multi-page editor that will contain three pages: overview, documents, and sml-if. The first page will only display general information about the SML-IF document, the second page will display the genic/phenic documents and the last page will display the plain XML format. See figure 1.0-1.1 for more details:

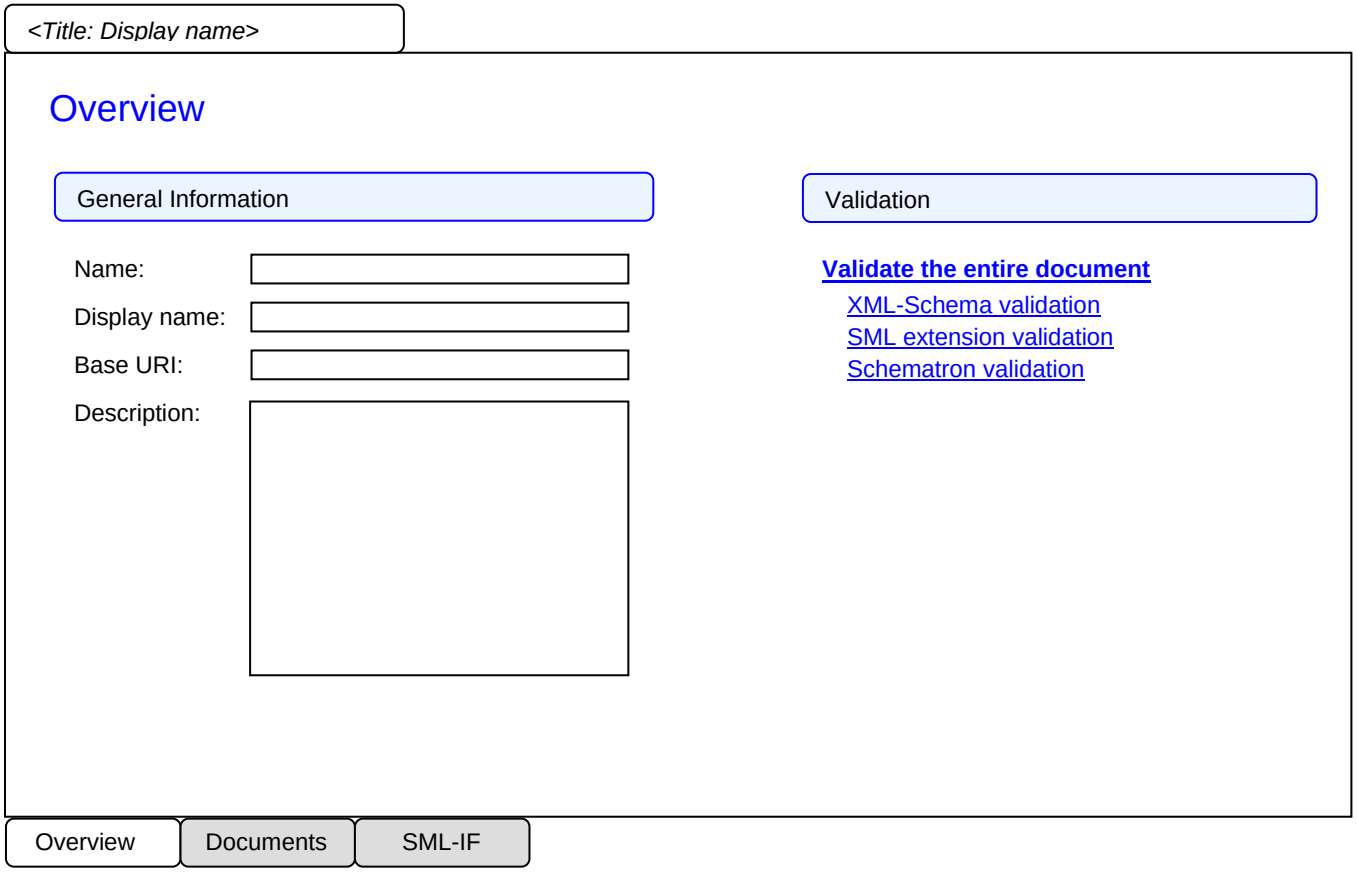

Figure 1.0 – The overview page of the SML-IF editor

Description of figure 1.0

The validation section allows the user to validate the entire document or run each validation phase separately. “Validate the entire document” link is equivalent to clicking the “XML-Schema validation”, “SML extension validation”, and “Schematron validation” links sequentially

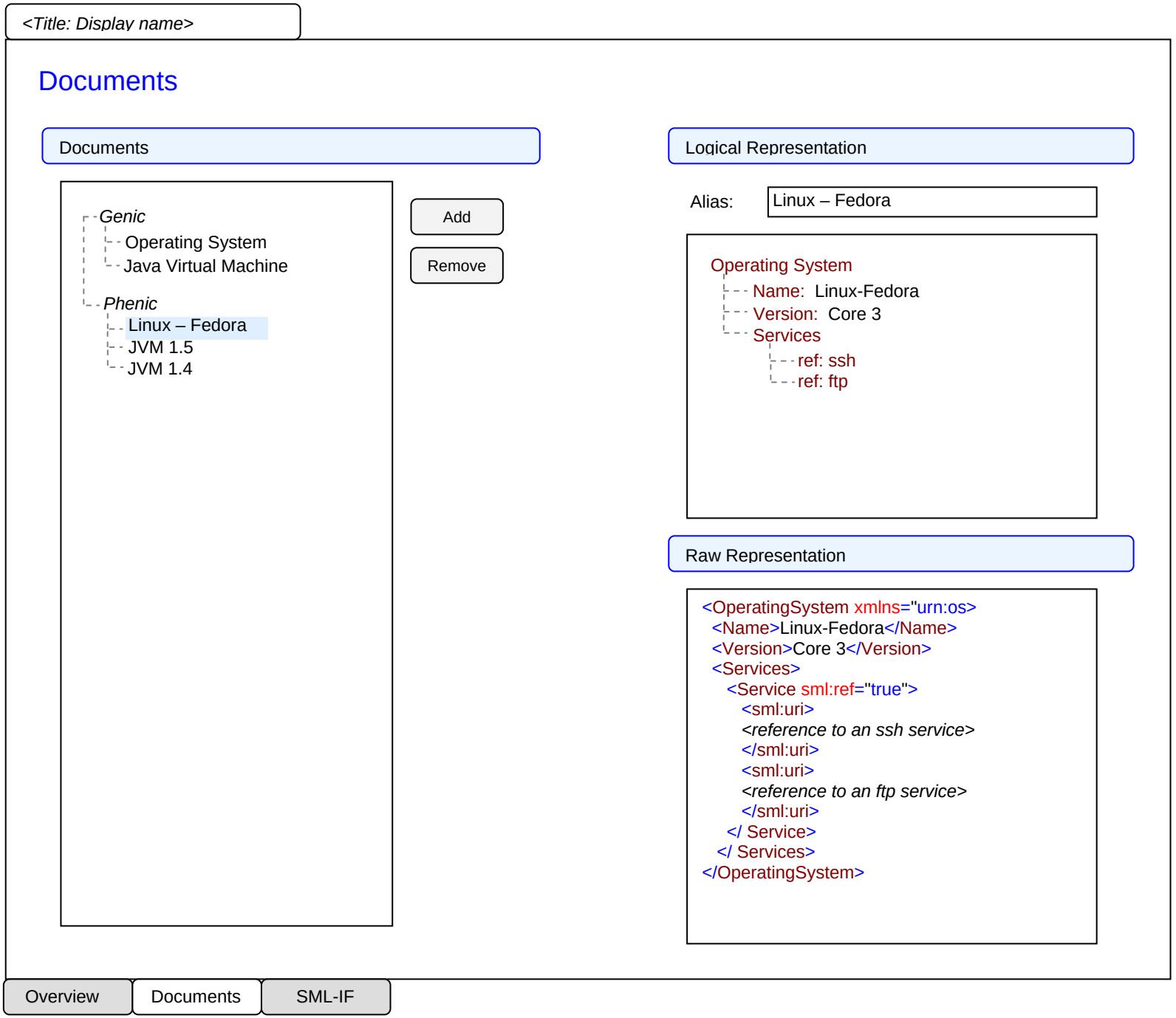

Figure 1.1 – The Documents page of the SML-IF editor

Description of figure 1.1

The user can add a new phenic/genic document by clicking the “Add” button and selecting a resource from a project. Similarly, a document can be removed by clicking the “Remove” button.

Figure 1.2 and 1.3 are screen mockups of the first and second page of the export wizard. Figure 1.4 is the import wizard used to import an SML-IF document into the workspace.

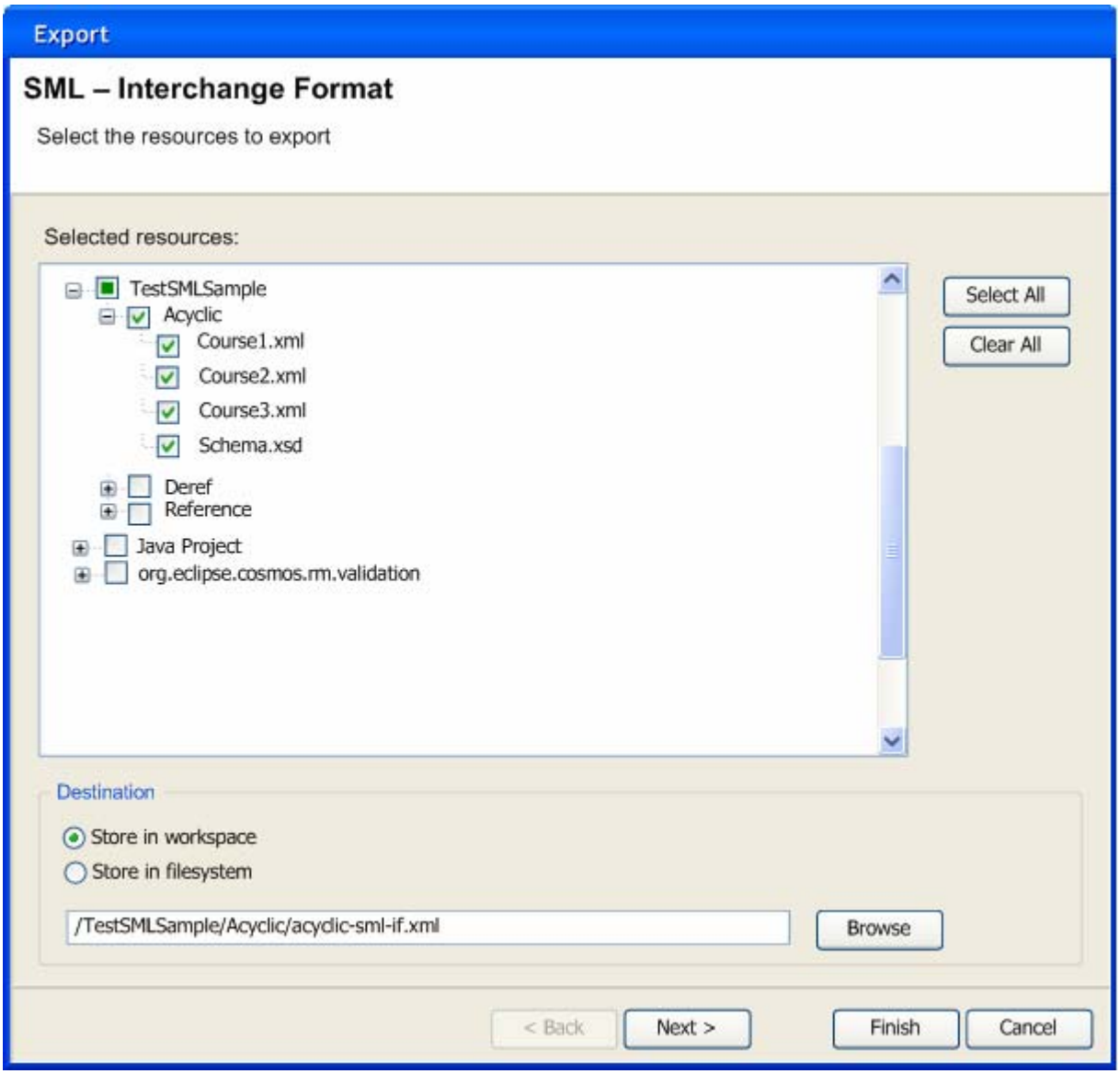

Figure 1.2 – First page of the export wizard

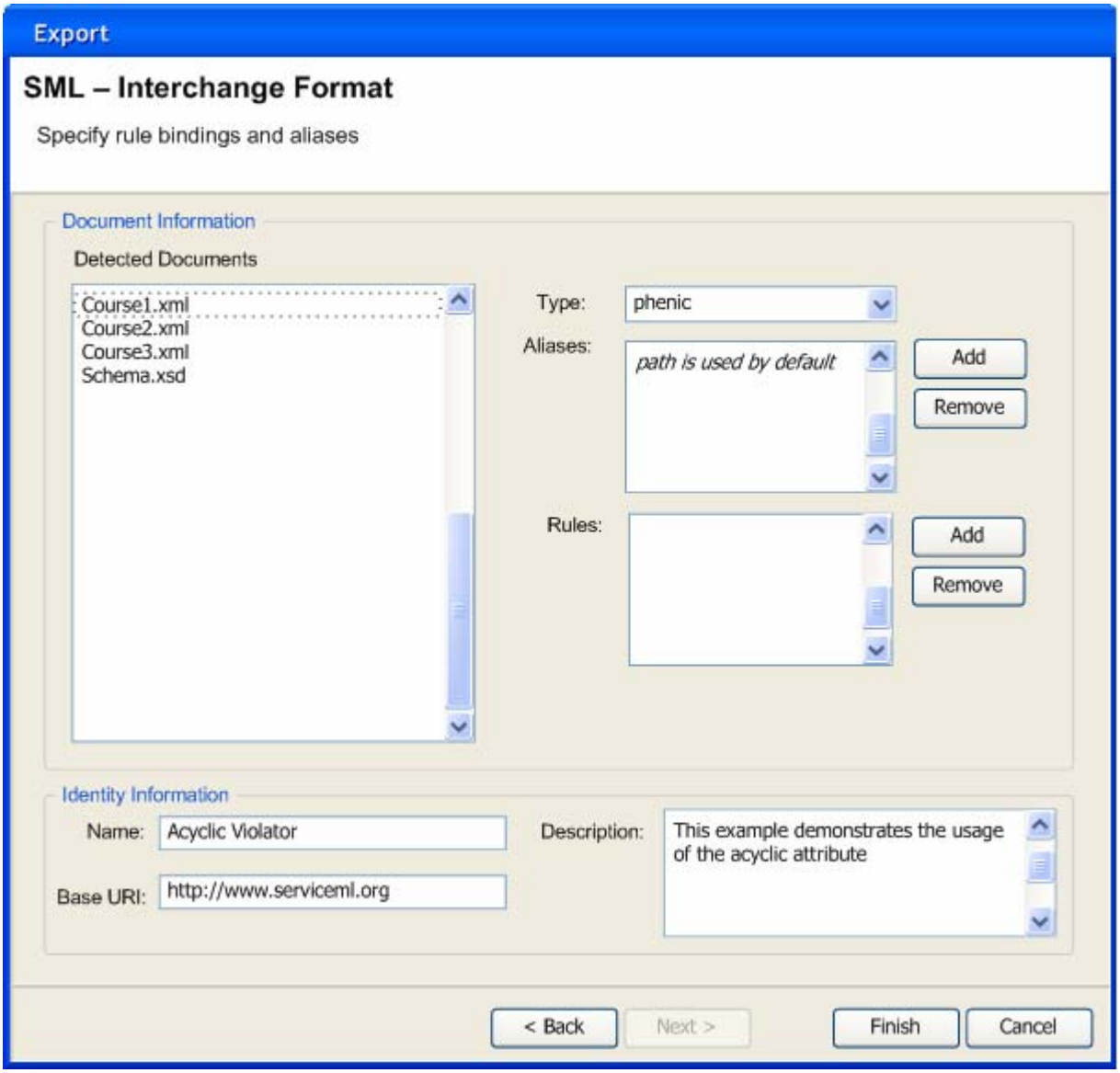

Figure 1.3 – Second page of the export wizard

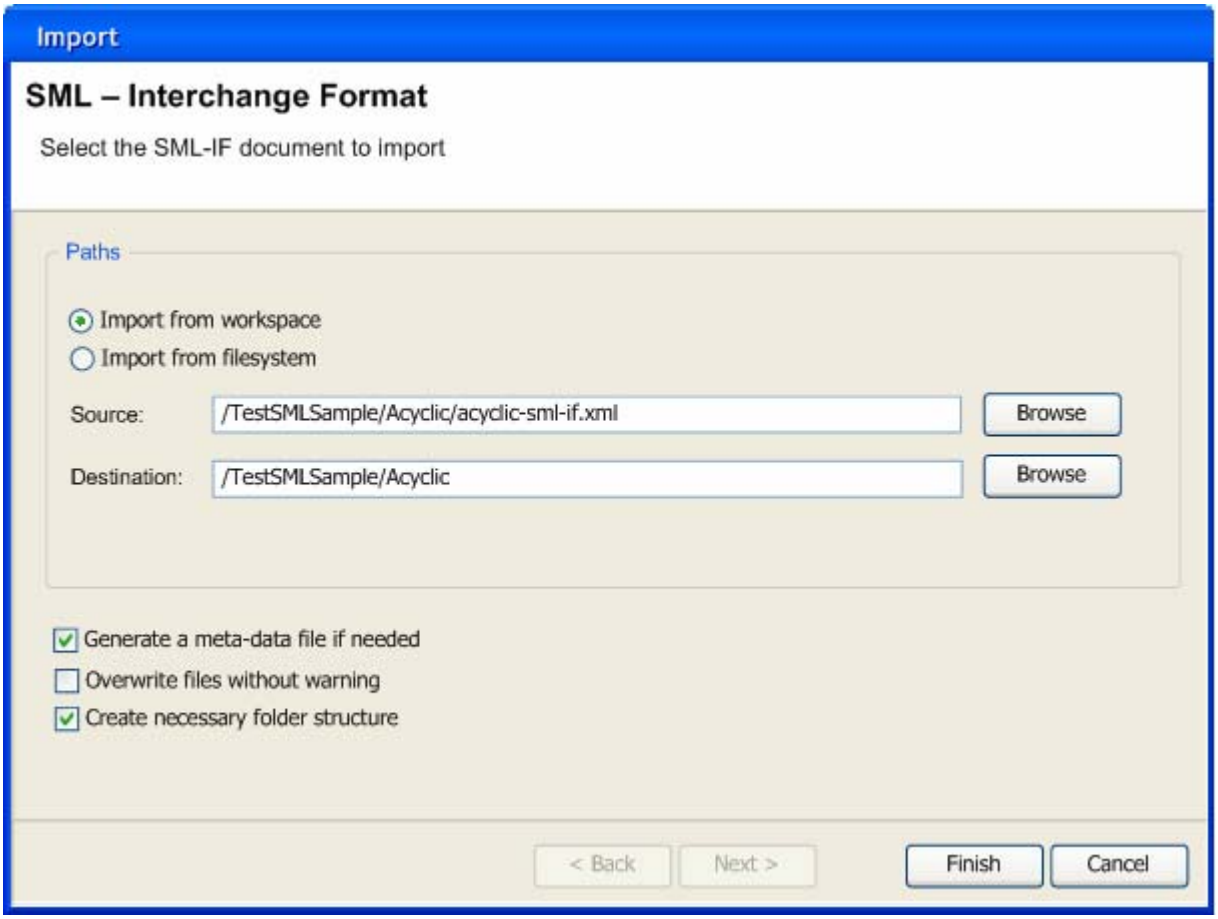

Figure 1.4 – The import wizard

1.3 Extension Points

As discussed in section 1.0, each phase of the validation scheme must be customizable. A single extension point will be provided to allow the user to customize each phase. Associated with each class is a priority field that indicates which set of classes to use in validating a document. A higher priority is indicated by a lower number (i.e. a class with priority 1 takes precedence over a class with priority 2). The smlValidation and schematronValidation elements define a scope that they are associated with. Contributors have the option to overwrite the target scope for one or more element. The target element "all" can be used to indicate that the scope of a validation class includes everything supported under smlValidation/schmatronValidation) (e.g. <target element="all"/>).

The priorities of the registered validation schemes can be modified by the user under a preference page. The user settings will always take precedence over the static priorities defined under extensions.

Name	`org.eclipse.cosmos.rm.validation.xmlValidation`
Description	`Defines an implementation for the first phase of the validation: i.e. XML validation. This class is required to ensure that the XML fragments conform to the schema that they are associated with.`
Example	`<extension point = "org.eclipse.cosmos.rm.validation.smlValidation"> <xmlValidation`

```
    priority = “10”
    class = “org.eclipse.cosmos.rm.validation.internal.XMLValidation”>
  </xmlValidation>

  <smlValidation
    priority = “10”
    class = “org.eclipse.cosmos.rm.validation.internal.SMLValidation”>
    <scope>
      <target element=”sml:acyclic”/>
      <target element=”sml:targetElement”/>
      <target element=”sml:targetType”/>
      <target element=”sml:key”/>
      <target element=”sml:unique”/>
      <target element=”sml:keyref”/>
      <target element=”references”/>
    </scope>
  </smlValidation>

  <schematronValidation
    priority = “10”
    class = “org.eclipse.cosmos.rm.validation.internal.Schematron
    Validation”>
    <scope>
      <target element=”process-prolog”/>
      <target element=”process-root”/>
      <target element=”process-assert”/>
      <target element=”process-report”/>
      <target element=”process-pattern”/>
      <target element=”process-rule”/>
      <target element=”process-message”/>
    </scope>
  </schematron>
</extension>
```

2.0 Task Breakdown

01 Add the validation scheme extension point
05 Create a manager class that will use the priority field to determine the set of classes to use as part of the validation
06 Provide a common interface for handling and reporting errors
10 A default implementation for the XML validation phase
15 A default implementation for the SML extension validation phase
20 A default implementation for the Schematron validation phase, which will include displaying items in the problems view based on report/assertion conditions
25-50 Create a default validation scheme for the SML extensions: sml:acyclic, sml:targetElement, sml:targetType, sml:key, sml:unique, and sml:keyref
51 An implementation of RFC 3986 to resolve inter-document references
52 Verify that the implementations cover the test cases of the workshop scenario (using JUnits)
53 Ensure that all validation schemes will work in an SML-IF context.
54 Modifying the core component to allow support for individual SML unit validation
55-57 Supporting validation at SML unit level: XML, Schema, Schematron validation
58-64 Supporting validation at SML unit level: sml:acyclic, sml:targetElement, sml:targetType, sml:targetRequired, sml:key, sml:unique, and sml:keyref.
70 Integrate the import and export operation with Eclipse's import/export wizard
72 Make the export and import wizard appear like the screen mock ups displayed under section 1.2
73 Generate a meta-data file that stores information which is lost when importing an SML-IF document (see section 0.2 – operations)
75 Add an extension for registering a multi-page editor that will be associated with SML-IF documents
80 Create the Overview page of the SML-IF editor
85 Create the Documents page of the SML-IF editor
90 Create the SML-IF page of the SML-IF editor
95 Create an action to create a new SML-IF document
100 Create a new action that will allow the user to export model units into an SML-IF document and import an SML-IF document to its SML units.
105 Support drag and drop of the model units into an SML-IF document
110 Provide an editor for xml files
115 Provide an editor for schema files

Completed tasks
Tasks to be completed

The first milestone will include the completion of the following tasks. The goal is to satisfy the test cases outlined in the workshop scenario:
06, 15, 20, 30, 35, 40, 45, 50, 51, 52, 53

The second milestone will involve supporting validation at a SML unit level and providing editing capabilities:

Ali:
54-64, 75, 80, 85, 105, 115, 01, 05

David:
100, 70, 72, 73, 90, 95, 110, 120