Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national standards bodies for voting. Publication as an International Standard requires approval of at least 75% of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements in this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 19757-8 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 34, Document Description and Processing Languages.

ISO/IEC 19757 consists of the following parts, under the general title *Document Schema Definition Languages (DSDL)*:

– *Part 1: Overview*

– *Part 2: Regular-grammar-based validation – RELAX NG*

– *Part 3: Rule-based validation – Schematron*

– *Part 4: Namespace-based validation dispatching language – NVDL*

– *Part 5: Datatypes*

– *Part 6: Path-based integrity constraints*

– *Part 7: Character repertoire description language – CRDL*

– *Part 8: Document schema renaming language – DSRL*

– *Part 9: Datatype- and namespace-aware DTDs*

– *Part 10: Validation management*

Introduction

This International Standard defines a set of Document Schema Definition Languages (DSDL) that can be used to specify one or more validation processes performed against Extensible Markup Language (XML) or Standard Generalized Markup Language (SGML) documents. (XML is an application profile SGML, ISO 8879:1986.)

A document model is an expression of the constraints to be placed on the structure and content of documents to be validated with the model. A number of technologies have been developed through various formal and informal consortia since the development of Document Type Definitions (DTDs) as part of ISO 8879, notably by the World Wide Web Consortium (W3C) and the Organization for the Advancement of Structured Information Standards (OASIS). A number of validation technologies are standardized in DSDL to complement those already available as standards or from industry.

To validate that a structured document conforms to specified constraints in structure and content relieves the potentially many applications acting on the document from having to duplicate the task of confirming that such requirements have been met. Historically, such tasks and expressions have been developed and utilized in isolation, without consideration for how the features and functionality available in other technologies might enhance validation objectives.

The main objective of this International Standard is to bring together different validation-related tasks and expressions to form a single extensible framework that allows technologies to work in series or in parallel to produce a single or a set of validation results. The extensibility of DSDL accommodates validation technologies not yet designed or specified.

In the past, different design and use criteria have led users to choose different validation technologies for different portions of their information. Bringing together information within a single XML document sometimes prevents existing document models from being used to validate sections of data. By providing an integrated suite of constraint description languages that can be applied to different subsets of a single XML document, this International Standard allows different validation technologies to be integrated under a well-defined validation policy.

This multi-part International Standard integrates constraint description technologies into a suite that:

– provides user control of names, order and repeatability of information objects (elements)

– allows users to identify restrictions on the co-concurrence of elements and element contents

– allows specific subsets of structured documents to be validated

– allows restrictions to be placed on the contents of specific elements, including restrictions based on the content of other elements in the same document

– allows the character set that can be used within specific elements to be managed, based on the application of the ISO/IEC 10646 Universal Multiple-Octet Coded Character Set (UCS)

– allows default values to be assigned to element contents and attribute values

– allows SGML to be used to declare document structure constraints that extend DTDs to include functions such as namespace-controlled validation and datatypes.

Document Schema Definition Languages (DSDL) – Part 8: Document Schema Renaming Language – DSRL

1 Scope

The Document Schema Renaming Language (DSRL) provides a mechanism whereby users can assign locally meaningful names to XML elements, attributes and entities without having to completely rewrite the DTD or schema to which they are required to conform. In addition, DSRL provides an XML-based format for defining entities.

DSRL allows users to define default values for both element content and attribute values. Default values can be forced to apply for all occurrences of the element or attribute, or only for those for which no value is provided in the document instance.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 19757. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 19757 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

Each of the following documents has a unique identifier that is used to cite the document in the text. The unique identifier consists of the part of the reference up to the first comma.

IRI, IETF RFC 3987, *Internationalized Resource Identifiers (IRIs)*, Internet Standards Track Specification, January 2005, http://www.ietf.org/rfc/rfc3987.txt

RELAX NG Compact Syntax, *ISO/IEC 19757-2:2003/Amd 1:2006 Information Technology – Document Schema Definition Languages (DSDL) – Part 2: Regular-grammar-based validation – RELAX NG*

Schematron, *ISO/IEC 19757-3:2006 Information Technology – Document Schema Definition Languages (DSDL) – Part 3: Rule-based validation – Schematron*

XML, *Extensible Markup Language (XML) 1.0 (Third Edition)*, W3C Recommendation, 4 February 2004, http://www.w3.org/TR/2004/REC-xml-20040204

XML-Infoset, *XML Information Set*, W3C Recommendation, 24 October 2001, http://www.w3.org/TR/2001/REC-xml-infoset-20011024/

XML-Names, *Namespaces in XML*, W3C Recommendation, 14 January 1999, http://www.w3.org/TR/1999/REC-xml-names-19990114/

XPath, *XML Path Language (XPath) Version 1.0*, W3C Recommendation, 16 November 1999, http://www.w3.org/TR/1999/REC-xpath-19991116

XML Schema, *XML Schema*, W3C Recommendation, 24 October 2001, http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

XSLT, *XSL Transformations (XSLT) Version 1.0*, W3C Recommendation, http://www.w3.org/TR/1999/REC-xslt-19991116

3 Terms and definitions

The terms and definitions of Parts 1, 2 and 3 of this standard also apply to this part, together with the following definitions:

3.1 DSDL

Document Schema Definition Languages defined in ISO/IEC 19757

3.2 DSRL

Document Schema Renaming Language defined in this Part of ISO/IEC 19757

3.3 DSRL map

set of rules that are used to map a document instance to a document model defined by one or more schemas

4 The role of the Document Schema Renaming Language

The Document Schema Renaming Language (DSRL) provides a mechanism whereby users can assign locally meaningful names to XML elements, attributes, entities and processing instructions without having to completely rewrite the DTD or schema to which they are required to conform. In addition, DSRL provides an XML-based format for defining entities.

DSRL allows users to define default values for both element content and attribute values. Default values can be forced to apply for all occurrences of the element or attribute, or only for those for which no value is provided in the document instance.

DSRL maps can be used to:

– Map document instances to validation schema

– Create schema that can be used to validate documents coded using alternative element or attribute names.

4.1 Namespace

Elements and attributes that conform to this Part of DSDL shall have an XML namespace definition (as defined in XML-Names) whose associated resource identifier (IRI) is:

```
http://purl.oclc.org/dsdl/dsrl
```

In this Part the prefix `dsrl:` is used to identify points at which this IRI defines the namespace.

NOTE 1: In most applications of DSRL this namespace will not be required as the IRI can be assigned as the default XML namespace.

Other namespaces required to group elements and attributes into processable units can be assigned as required for validation.

5 DSRL maps

The outermost element of a DSRL map has the following structure:

```
<dsrl:maps xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl">
  ...
</dsrl:maps>
```

Two optional attributes may be associated with this element:

– `targetNamespace` can be used to record the IRI assigned as the target namespace for the validating schema

– `schemaLocation` can be used to record the IRI assigned to a copy of the schema to be used to validate mapped document instances.

Are there any circumstances in which the target namespace (or schema location) are required to be specified?

The formal declaration for this element, defined using the RELAX NG Compact Syntax, is:

```
namespace xsd="http://www.w3.org/2001/XMLSchema-datatypes"
namespace dsrl="http://purl.oclc.org/dsdl/dsrl"

maps =  maps =  element dsrl:maps
                {target-namespace?, schema-location?,
                 (element-map | attribute-map | map-pi-target )+,
                 entity-name-map?, define-entity*
                }

target-namespace = attribute targetNamespace {xsd:anyURI}
schema-location = attribute targetSchemaLocation {xsd:anyURI}
```

6 Mapping user-defined names to schema-defined names

6.1 Reassigning element and attribute names

The `dsrl:element-map` element is used to record transliterations that apply to element names and to thier associated attributes. The model for this element is:

```
element-map = element dsrl:element-map (within?, (name | name-map),
                                        attribute-map*, default-content?)

within = element dsrl:within { text }
name = element dsrl:name { text }
name-map = { (from, to) }
from = element dsrl:from { text }
to = element dsrl:to { text }
```

The contents of a `dsrl:element-map` element consists of a sequence of elements that define which name in a document instance is to be matched to which element in the validation schema, and which attributes of the element are to be mapped. Optionally default content can also be defined for the element.

The name of the element to be mapped is recorded in the content of the `dsrl:from` element. If this element needs to be transliterated in different ways in different contexts the optional `dsrl:within` element can be used to record XML path (XPath) locations that distinguish between the different contexts in which transliteration of the name is to be applied. No two `dsrl:element-map` elements shall have the same contents for both their `dsrl:within` element and their `dsrl:from` element.

The name to be applied to the mapped element when it is validated is recorded as the contents of the `dsrl:to` element.

If the element name is to stay the same, but one or more attributes of the element is to have its name or values mapped, the `dsrl:name` element can be used in place of the `dsrl:from` and `dsrl:to` pair.

Names may be qualified providing the relevant namespaces have been declared within the schema. They may not contain spaces, or any other character that is not a valid name character as defined in the W3C XML specification.

The contents of the `within` element must form a valid XPath location that identifies a valid parent for the element to be renamed.

NOTE 2: XPath locations must not end with a `/`.

NOTE 3: A typical example of a DSRL element name map for which no attribute mapping is required would be:

```
<dsrl:element-map>
  <dsrl:from>adresse</dsrl:from>
  <dsrl:to>address</dsrl:to>
</dsrl:element-map>
```

The dsrl:attribute-map element is used within dsrl:element-maps to record transliterations that apply to attribute names and values. The model for this element is:

```
attribute-map = element dsrl:attribute-map {(name | name-map),
                                            values-map?, default-value?}
```

Each dsrl:attribute-map transliterates a single attribute. The name of the attribute to be mapped is recorded as the content of the dsrl:from element. No two dsrl:attribute-map elements within a given dsrl:element-map may have the same value for their dsrl:from element.

The name to be applied to the mapped attribute when it is validated is recorded as the contents of the immediately following dsrl:to element. If the dsrl:to element is empty the attribute named in the dsrl:from element is to be removed prior to validation.

If the values of an attribute are to be mapped without the name of the attribute changing a dsrl:name element can be used in place of the dsrl:from and dsrl:to pair.

If the attribute map is defined as child of a dsrl:maps element, rather than a dsrl:element-map element, the attribute map will be applied to all elements that have an attribute of that name for which a specific mapping has not been declared.

Attribute names may be qualified providing the relevant namespaces have been declared within the schema. They may not contain spaces, or any other character that is not a valid name character as defined in the XML specification.

NOTE 4: A typical example of a DSRL attribute name map which could be nested with the dsrl:element-map example shown above is:

```
<dsrl:attribute-map>
  <dsrl:from>sorte</dsrl:from>
  <dsrl:to>type</dsrl:to>
</dsrl:attribute-map>
```

When name mapping has been applied the XML information set (XML-Infoset) will contain the name required by the schema to validate each element or attribute. Optionally the system may record the original names of the element and its attributes in a processing instruction that immediately follows the element's start-tag. The PItarget of the processing instruction shall be DSRL. The original name of the element shall be recorded in an original-element-name property. The names used for each attribute shall be recorded in an original-attribute-names property as a pair of values, the first of which records the attribute name in the source document instance and the second the attribute name used in the validated document.

NOTE 5: A typical procesing instruction record of a mapping will have the form:

```
<?DSRL original-element-name="main-text"
       original-attribute-names="number-of-columns cols"?>
```

6.2 Mapping attribute value tokens

Where an attribute is a member of a defined name token group, or a valid name, a mapping can be declared between names in a source document and names in a target schema. The model for the dsrl:values-map element that is nested within the appropriate dsrl:attribute-map element is:

```
values-map = element dsrl:values-map {name-map+}
```

The value to be mapped is recorded as the content of a `dsrl:from` element. No two `dsrl:from` elements within the same `dsrl:values-map` element may have the same contents.

The name to be assigned to the mapped attribute when it is validated is recorded as the contents of the associated `dsrl:to` element.

NOTE 6: A typical example of a DSRL attribute values map is:

```
<dsrl:values-map>
  <dsrl:from>maison</dsrl:from><dsrl:to>home</dsrl:to>
  <dsrl:from>bureau</dsrl:from><dsrl:to>office</dsrl:to>
</dsrl:values-map>
```

Should it be possible to record original attribute values in a DSRL PI?

6.3 Default attribute values

A `dsrl:default-value` element can be used to define a default value for an attribute at the end of an attribute map. The model for this element is:

```
default-value= element dsrl:default-value {force-default?, text
force-default = attribute force-default {xsd:boolean}
```

The default value to be assigned is recorded as the contents of the `dsrl:default-value` element. If the value of the `force-default` attribute is set to `true` the contents will be used to validate the attribute irrespective of what is in the element start-tag within the document instance. Otherwise any value assigned to the named attribute within the document instance will be used during validation of the document, but if the attribute is not present the default value will be applied.

NOTE 7: A typical example of a DSRL default attribute value definition, as introduced as the last element in an `dsrl:attribute-map` element, is:

```
<dsrl:default-value force-default="true">iso3166</dsrl:default-value>
```

6.4 Default content

A `dsrl:default-content` element can used to define a default value for an element defined in a schema. The model for this element is:

```
default-content = element dsrl:default-content {force-default?, after, any-content}
after = attribute after {text}
any-content = (mixed {any-element*})
any-element = element * {any-attribute, any-content}
any-attribute = (attribute * {text})*
```

The default content for the element is recorded as the content of the `dsrl:default-content` element. If the value of the `force-default` attribute is set to `true` the contents will be used to validate the document instance irrespective of what is in the element within the document instance. Otherwise any contents assigned to the element within the document instance will be used during validation of the document, but if the element is empty, or not present, the default content will be applied during validation.

The `after` attribute of the `default content` element records the name of the source element after which the content has to be placed if missing. The element the default content is to be placed after must be declared within the same DSRL map, even if it is not altered in any way. To ensure that the after attribute can be applied in the appropriate context the containing `element-map` element must start with a `within` statement that contains the name of the element within which the element identified by the `after` attribute occurs.

Should the after attribute identify the element after which the default is to be inserted in the result schema rather than the current definition, which is based on which source element it needs to follow? (NB Mapping to the source element makes the XSLT easier!)

NOTE 8: A typical example of a DSRL default content definition is:

```
<dsrl:default-content force-default="true" after="rue">Downtown</dsrl:default-content>
```

NOTE 9: A typical example of a DSRL element map with mapped attribute values and a default attribute value, might be:

```
<dsrl:element-map>
  <dsrl:within>adresse</dsrl:within>
  <dsrl:from>ville</dsrl:from>
  <dsrl:to>locality</dsrl:to>
  <dsrl:attribute-map>
      <dsrl:name>required</dsrl:name>
      <dsrl:default-value force-default="true">maybe</dsrl:default-value>
   </dsrl:attribute-map>
   <dsrl:attribute-map>
        <dsrl:name>imported</dsrl:name>
        <dsrl:default-value force-default="false">what</dsrl:default-value>
     </dsrl:attribute-map>
  <dsrl:default-content force-default="true" after="rue">Downtown</dsrl:default-content>
</dsrl:element-map>
```

6.5 Renaming processing instruction targets

Where the names and properties of processing instructions have not been defined in terms understandable to user-communities, users of DSRL can create a mapping rule that associates alternative processing instruction names used in document instances with the name of the processing instruction target to be used during validation using a `dsrl:map-pi-target` element. The model for this element is:

```
map-pi-target = element dsrl:map-pi-target {name-map+}
```

The processing instruction name to be mapped is recorded as the content of a `dsrl:from` element. No two `dsrl:map-pi-targete` elements may have the same contents in their `dsrl:from` element.

The name to be assigned to the mapped processing instruction when it is validated is recorded as the contents of the associated `dsrl:to` element.

If two or more `dsrl:map-pi-target` elements occur within a DSRL map their contents shall be concatenated to form a single mapping of processing instruction targets.

NOTE 10: A typical example of a DSRL processing instruction name map is:

```
<dsrl:map-pi-target>
 <dsrl:from>MyPIname</dsrl:from><dsrl:to>PItarget</dsrl:to>
 <dsrl:from>AlternativePIname</dsrl:from><dsrl:to>PItarget</dsrl:to>
</dsrl:map-pi-target>
```

Should it be possible to record original processing instruction names in a DSRL PI, or as an DSRL property of the mapped processing instruction?

6.6 Mapping entity references

Neither Part 2 of this International Standard, the RELAX NG regular-grammar-based validation language, nor W3C XML schemas provide a mechanism for defining XML entities that can be referenced within document instances. Only XML DTDs can be used to specify general entities other than the five specified as default entities within the XML specification (`&`, `<`, `&ge;`, `&apos;` and `&quote;`).

NOTE 11: An alternative mechanism for defining the equivalent of general entities is provided within Clause 7 of this standard.

Often the names assigned to entity references, including the default ones defined for XML, are difficult for users to understand or remember, especially when they are specified using a language which is not the native language of a particular user community. The facilities in this clause allow locally-significant names to be mapped to those used to define entities in a referenced entity set.

A `dsrl:entity-name-map` element is used to identify reusable mappings between names used in entity definitions and those used in entity references. The model for this element is:

```
entity-name-map = element dsrl:entity-name-map {name-map+}
```

The entity name to be mapped is recorded as the content of a `dsrl:from` element. No two `dsrl:from` elements within a `dsrl:entity-name-map` element may have the same contents.

The name to be assigned to the mapped entity when it is validated is recorded as the contents of the immediately following `dsrl:to` element.

NOTE 12: A typical example of a DSRL attribute name map is:

```
<dsrl:entity-name-map>
 <dsrl:from>and</dsrl:from><dsrl:to>amp</dsrl:to>
 <dsrl:from>open-tag</dsrl:from><dsrl:to>lt</dsrl:to>
 <dsrl:from>close-tag</dsrl:from><dsrl:to>gt</dsrl:to>
 <dsrl:from>e</dsrl:from><dsrl:to>eacute</dsrl:to>
</dsrl:entity-name-map>
```

7 Defining entities

The `dsrl:define-entity` element provides a simple XML-based mechanism for defining replacement text that can be referenced using entity references. The model for this element is:

```
define-entity = element dsrl:define-entity {from, replacement-text}
define replacement-text = element dsrl:replacement-text  {any-content}
```

The name of the entity to be defined is recorded as the content of a `dsrl:from` element. No two `dsrl:from` elements within `dsrl:define-entity` elements within a DSRL map may have the same contents.

The replacement text for any refrences made to the named entity is recorded as the contents of the immediately following `dsrl:replacement-text` element. If the replacement text contains markup it must be defined as a CDATA marked section.

NOTE 13: A typical example of a DSRL entity definition is:

```
<dsrl:define-entity>
  <dsrl:from>e</dsrl:from><dsrl:replacement-text>&#233;</dsrl:replacement-text>
  <dsrl:from>CSWi</dsrl:from><dsrl:replacement-text>CWS Informatics</dsrl:replacement-text>
</dsrl:define-entity>
<dsrl:define-entity>
  <dsrl:from>XML</dsrl:from>
  <dsrl:replacement-text>
    <![CDATA[the W3C Extensible Markup Language (<acronym>XML</acronym>)]] >
  </dsrl:replacement-text>
</dsrl:define-entity>
```

Annex A
(normative)

Validation of declarative document architectures

The normative schemas defined in this annex provide formal definitions for the elements and attributes used to define DSRL maps.

A.1 RELAX NG XML Schema for Validating DSRL

The following ISO/IEC 19757-2, RELAX NG, schema can be used to validate DSRL maps:

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl"
    xmlns="http://relaxng.org/ns/structure/1.0"
    datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <start>
    <ref name="maps"/>
  </start>
  <define name="maps">
    <element name="dsrl:maps">
      <optional>
        <ref name="target-namespace"/>
      </optional>
      <optional>
        <ref name="schema-location"/>
      </optional>
      <oneOrMore>
        <choice>
          <ref name="element-map"/>
          <ref name="attribute-map"/>
          <ref name="map-pi-target"/>
        </choice>
      </oneOrMore>
      <optional>
        <ref name="entity-name-map"/>
      </optional>
      <zeroOrMore>
        <ref name="define-entity"/>
      </zeroOrMore>
    </element>
  </define>
  <define name="target-namespace">
    <attribute name="targetNamespace">
      <data type="anyURI"/>
    </attribute>
  </define>
  <define name="schema-location">
    <attribute name="targetSchemaLocation">
      <data type="anyURI"/>
    </attribute>
  </define>
  <define name="element-map">
    <element name="dsrl:element-map">
      <optional>
        <ref name="within"/>
```

```
    </optional>
    <choice>
      <ref name="name"/>
      <ref name="name-map"/>
    </choice>
    <zeroOrMore>
      <ref name="attribute-map"/>
    </zeroOrMore>
    <optional>
      <ref name="default-content"/>
    </optional>
  </element>
</define>
<define name="within">
  <element name="dsrl:within">
    <text/>
  </element>
</define>
<define name="name">
  <element name="dsrl:name">
    <text/>
  </element>
</define>
<define name="name-map">
  <ref name="from"/>
  <ref name="to"/>
</define>
<define name="from">
  <element name="dsrl:from">
    <text/>
  </element>
</define>
<define name="to">
  <element name="dsrl:to">
    <text/>
  </element>
</define>
<define name="attribute-map">
  <element name="dsrl:attribute-map">
    <choice>
      <ref name="name"/>
      <ref name="name-map"/>
    </choice>
    <optional>
      <ref name="values-map"/>
    </optional>
    <optional>
      <ref name="default-value"/>
    </optional>
  </element>
</define>
<define name="values-map">
  <element name="dsrl:values-map">
    <oneOrMore>
      <ref name="name-map"/>
    </oneOrMore>
  </element>
</define>
<define name="default-value">
  <element name="dsrl:default-value">
```

```
      <optional>
        <ref name="force-default"/>
      </optional>
      <text/>
    </element>
  </define>
  <define name="force-default">
    <attribute name="force-default">
      <data type="boolean"/>
    </attribute>
  </define>
  <define name="map-pi-target">
    <element name="dsrl:map-pi-target">
      <oneOrMore>
        <ref name="name-map"/>
      </oneOrMore>
    </element>
  </define>
  <define name="entity-name-map">
    <element name="dsrl:entity-name-map">
      <oneOrMore>
        <ref name="name-map"/>
      </oneOrMore>
    </element>
  </define>
  <define name="define-entity">
    <element name="dsrl:define-entity">
      <ref name="from"/>
      <ref name="replacement-text"/>
    </element>
  </define>
  <define name="replacement-text">
    <element name="dsrl:replacement-text">
      <ref name="any-content"/>
    </element>
  </define>
  <define name="any-content">
    <mixed>
      <zeroOrMore>
        <ref name="any-element"/>
      </zeroOrMore>
    </mixed>
  </define>
  <define name="any-element">
    <element>
      <anyName/>
      <ref name="any-attribute"/>
      <ref name="any-content"/>
    </element>
  </define>
  <define name="any-attribute">
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </define>
  <define name="default-content">
    <element name="dsrl:default-content">
      <optional>
```

```
        <ref name="force-default"/>
      </optional>
      <ref name="after"/>
      <ref name="any-content"/>
    </element>
  </define>
  <define name="after">
    <attribute name="after"/>
  </define>
</grammar>
```

Figure 1 illustrates the contents of this schema.

A.2 RELAX NG Compact Schema for Validating DSRL maps

When names maps are stored externally the following RELAX NG compact syntax schema can be used to validate the map:

```
namespace rng = "http://relaxng.org/ns/structure/1.0"
datatypes xsd  = "http://www.w3.org/2001/XMLSchema-datatypes"
namespace dsrl = "http://purl.oclc.org/dsdl/dsrl"

start = maps
maps =  element dsrl:maps
                {target-namespace?, schema-location?,
                 (element-map | attribute-map | map-pi-target )+,
                 entity-name-map?, define-entity*
                }

target-namespace = attribute targetNamespace {xsd:anyURI}
schema-location = attribute targetSchemaLocation {xsd:anyURI}

element-map = element dsrl:element-map {(within?, (name | name-map),
                                         attribute-map*, default-content?)}

within = element dsrl:within { text }
name = element dsrl:name { text }
name-map = (from, to)
from = element dsrl:from { text }
to = element dsrl:to { text }

attribute-map = element dsrl:attribute-map {(name | name-map), values-map?, default-value?}
values-map = element dsrl:values-map {name-map+}
default-value= element dsrl:default-value {force-default?, text }
force-default = attribute force-default {xsd:boolean}

map-pi-target = element dsrl:map-pi-target {name-map+}

entity-name-map = element dsrl:entity-name-map {name-map+}

define-entity = element dsrl:define-entity {from, replacement-text}

replacement-text = element dsrl:replacement-text  {any-content}

any-content = (mixed {any-element*})
any-element = element * {any-attribute, any-content}
any-attribute = (attribute * {text})*

default-content = element dsrl:default-content {force-default?, after, any-content}
after = attribute after {text}
```

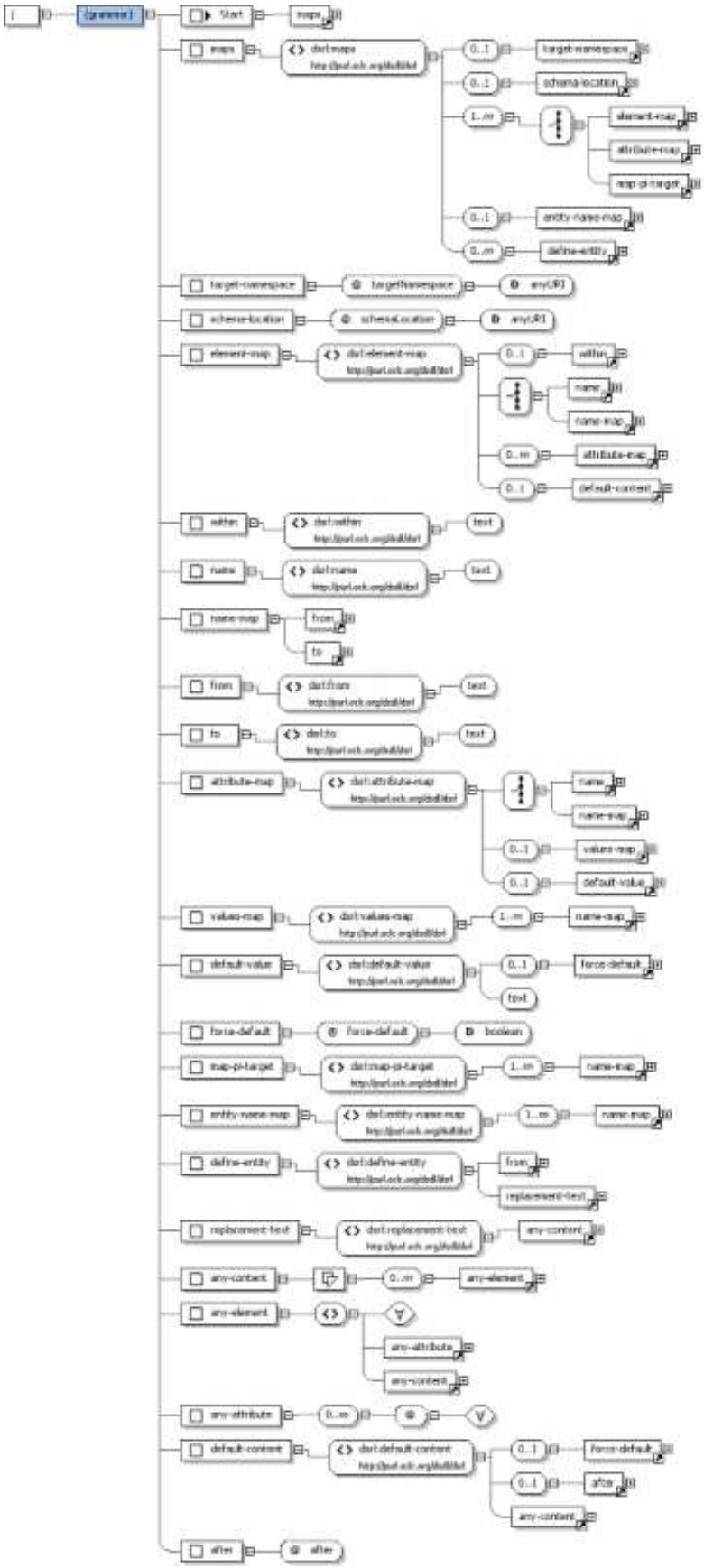

Figure 1: Diagrammatic representation of schema for DSRL

A.3 Schematron Rules for Validating DSRL

The following Schematron rules can be used to validate that `dsrl:default-content` elements have been defined validly:

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron"
xml:lang="en">
<sch:title>Schema for Additional Constraints for ISO/IEC 19757-8: DSRL</sch:title>
<sch:ns prefix="dsrl" uri="http://purl.oclc.org/dsdl/dsrl" />
<sch:p>This schema supplies some constraints in addition to those given
  in the ISO/IEC 19757-8 (Document Schema Renaming Language) schema.
</sch:p>
<sch:pattern>
<sch:rule context="dsrl:element-map/dsrl:default-content/@after">
<sch:assert  test=".=../../dsrl:from or .=../../dsrl:name">
  The contents of the after element must match the name of an
  element included in the same map.
</sch:assert>
</sch:rule>
<sch:rule context="dsrl:element-map/dsrl:default-content">
<sch:assert test="../dsrl:within">
  Whenever default content is assigned to an element a dsrl:within
  declaration must  occur at the same level in the element map.
</sch:assert>
</sch:rule>
</sch:pattern>
</sch:schema>
```

Annex B
(informative)

Using DSRL and XSLT to Transform Schemas and Documents

B.1 Using free-standing DSRL rules to transform document instances

DSRL rules that are defined in a separate mapping file can be converted into XSLT transformation rules that can be used to convert a document instance into a form that can be validated by the relevant validation schema by use of the following XSLT 2.0 transform:

NOTE 14: To allow declarations to fit onto the printed page, some addtional character returns and spaces have been added to the declarations shown in this example. For the latest version of this XSLT transformation, without the added line breaks, visit the DSDL.org website.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/1999/XSL/TransformAlias"
  xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xi="http://www.w3.org/2001/XInclude"
  exclude-result-prefixes="dsrl xsl xsd xi">
  <!-- Example of how XSLT 2.0 can be used to transform a DSRL map to create
        an XSL styleseet, called DSRLtranform.xsl, that can be used to transform
        XML instances into validatable documents.

        Also creates a file, DSRLentities.ent that redefines mapped entities in format
        required to allow transformation using DSRLtransform.xsl.

        © ISO SC34/WG1, 2006
  -->
  <xsl:output name="transforms" method="xml" indent="yes"/>
  <xsl:output name="entity-definitions" method="text"/>
  <xsl:output name="mapped-entities" method="text"
     extension-element-prefixes="#default"/>

  <xsl:namespace-alias stylesheet-prefix="xs" result-prefix="xsl"/>

  <!-- Variable for storing information of local working directory:
        Needs to be customized for local environment -->
  <xsl:param name="output-directory">/DSDL/DSDL8%20Examples/</xsl:param>
  <xsl:param name="entity-redefinition">DSRLentities.ent</xsl:param>
  <xsl:param name="MappedEntities">MappedEntities.ent</xsl:param>

  <xsl:template match="dsrl:maps">
    <xsl:result-document href="{$output-directory}DSRLtransform.xsl"
      format="transforms">
      <xsl:text disable-output-escaping="yes">
&#60;!DOCTYPE xsl:stylesheet [
&#60;!ENTITY % MappedEntities SYSTEM "</xsl:text>
      <xsl:value-of select="$MappedEntities"/>
      <xsl:text disable-output-escaping="yes">"&#62; %MappedEntities;
]&#62;
</xsl:text>
      <xs:stylesheet version="1.0" xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl" >
        <xsl:namespace name="" select="/dsrl:maps/@targetNamespace"/>
```

```
      <xs:output method="xml" indent="yes"/>
      <xsl:apply-templates/>
      <xs:template match="*|@*">
        <xs:copy>
          <xs:apply-templates select="@*"/>
          <xs:apply-templates select="*|text()|comment()|processing-instruction()"/>
        </xs:copy>
      </xs:template>
    </xs:stylesheet>
  </xsl:result-document>
</xsl:template>

<xsl:template match="dsrl:element-map">
  <xsl:variable name="target">
    <xsl:choose>
      <xsl:when test="dsrl:name">
        <xsl:if test="dsrl:within"><xsl:value-of select="dsrl:within"/>/</xsl:if>
        <xsl:value-of select="dsrl:name"/>
      </xsl:when>
      <xsl:when test="dsrl:from">
        <xsl:if test="dsrl:within"><xsl:value-of select="dsrl:within"/>/</xsl:if>
        <xsl:value-of select="dsrl:from"/>
      </xsl:when>
    </xsl:choose>
  </xsl:variable>
  <xsl:variable name="new-name">
    <xsl:choose>
      <xsl:when test="dsrl:name">
        <xsl:value-of select="dsrl:name"/>
      </xsl:when>
      <xsl:when test="dsrl:to">
        <xsl:value-of select="dsrl:to"/>
      </xsl:when>
    </xsl:choose>
  </xsl:variable>
  <xs:template match="{$target}">
    <xs:element name="{$new-name}">
      <xsl:if test="@targetSchemaLocation">
        <xs:attribute name="schemaLocation"
          namespace="http://www.w3.org/2001/XMLSchema-instance">
          <xsl:value-of select="/dsrl:maps//@targetNamespace"/>
          <xsl:text> </xsl:text>
          <xsl:value-of select="/dsrl:maps//@targetSchemaLocation"/>
        </xs:attribute>
      </xsl:if>
      <xsl:if test="dsrl:attribute-map">
        <xs:for-each select="@*">
          <xsl:call-template name="attribute-map"/>
        </xs:for-each>
        <xsl:for-each select="descendant::dsrl:default-value">
          <xsl:variable name="attribute-name">
            <xsl:if test="../dsrl:name">
              <xsl:value-of select="../dsrl:name"/>
            </xsl:if>
            <xsl:if test="../dsrl:from">
              <xsl:value-of select="../dsrl:from"/>
            </xsl:if>
          </xsl:variable>
          <xs:if test="not(@{$attribute-name})">
```

```
            <xs:attribute name="{$attribute-name}">
              <xsl:value-of select="."/>
            </xs:attribute>
          </xs:if>
        </xsl:for-each>
      </xsl:if>
      <xsl:if test="not(dsrl:attribute-map)">
        <xs:apply-templates select="@*"/>
      </xsl:if>
      <xsl:choose>
        <xsl:when test="dsrl:default-content/@force-default='true'">
          <xsl:value-of select="descendant::dsrl:default-content"/>
        </xsl:when>
        <xsl:when test="dsrl:default-content">
          <xs:if test=".=''">
           <xsl:value-of select="dsrl:default-content"/>
          </xs:if>
          <xs:if test="not(.='')">
            <xs:value-of select="."/>
          </xs:if>
        </xsl:when>
        <xsl:otherwise>
          <xs:apply-templates/>
        </xsl:otherwise>
      </xsl:choose>
    </xs:element>
  </xs:template>
  <xsl:if test="dsrl:default-content/@force-default='true'">
    <xsl:variable name="within">
      <xsl:value-of select="dsrl:within"/>
    </xsl:variable>
    <xsl:variable name="to">
      <xsl:value-of select="//dsrl:element-map/
                            dsrl:from[.=$within]/following-sibling::dsrl:to[1]"/>
    </xsl:variable>
    <xs:template match="{dsrl:within}" priority="1">
      <xs:element name="{$to}">
        <xsl:if test="dsrl:attribute-map">
          <xs:for-each select="@*">
            <xsl:for-each select="//dsrl:element-map">
              <xsl:if test="dsrl:name=$within">
                <xsl:call-template name="attribute-map"/>
              </xsl:if>
              <xsl:if test="dsrl:from=$within">
                <xsl:call-template name="attribute-map"/>
              </xsl:if>
            </xsl:for-each>
          </xs:for-each>
        </xsl:if>
        <xs:apply-templates/>
      </xs:element>
    </xs:template>
    <xs:template match="{dsrl:default-content/@after}" priority="1">
      <xsl:variable name="after">
        <xsl:value-of select="dsrl:default-content/@after"/>
      </xsl:variable>
      <xsl:variable name="after-to">
        <xsl:value-of select="//dsrl:from[.=$after]/following-sibling::*[1]"/>
        <xsl:if test="//dsrl:from[.='$after']">
          <xsl:value-of select="//dsrl:from[.=$after]/following-sibling::*[1]"/></xsl:if>
```

```
        <xsl:if test="//dsrl:name[.='$after']"><xsl:value-of select="$after"/></xsl:if>
      </xsl:variable>
      <xs:element name="{$after-to}">
        <xs:for-each select="@*">
          <xsl:for-each select="//dsrl:element-map">
            <xsl:choose>
              <xsl:when test="dsrl:name[.=$after]/following-sibling::dsrl:attribute-map">
                <xsl:call-template name="attribute-map"/>
              </xsl:when>
              <xsl:when test="dsrl:from[.=$after]/following-sibling::dsrl:attribute-map">
                <xsl:call-template name="attribute-map"/>
              </xsl:when>
            </xsl:choose>
          </xsl:for-each>
          <xsl:if
            test="not(//dsrl:element-map/dsrl:from[.=$after]/
                   following-sibling::dsrl:attribute-map |
                  //dsrl:element-map/dsrl:name[.=$after]/
                   following-sibling::dsrl:attribute-map)">
            <xs:copy-of select="."/>
          </xsl:if>
        </xs:for-each>
        <xs:apply-templates/>
      </xs:element>
      <xs:if test="not(../{substring-after($target, '/')})">
          <xs:element name="{$new-name}">
            <xsl:for-each select="dsrl:attribute-map">
              <xs:attribute>
                <xsl:attribute name="name">
                   <xsl:value-of select="dsrl:name"/>
                  </xsl:attribute>
                <xsl:value-of select="dsrl:default-value"/>
              </xs:attribute>
             </xsl:for-each>
            <xsl:value-of select="dsrl:default-content"/>
          </xs:element>
        </xs:if>
    </xs:template>

  </xsl:if>
</xsl:template>

<xsl:template match="dsrl:maps/dsrl:attribute-map">
<xsl:variable name="target">
    <xsl:choose>
      <xsl:when test="dsrl:name">
        <xsl:if test="dsrl:within"><xsl:value-of select="dsrl:within"/>/</xsl:if>
        <xsl:value-of select="dsrl:name"/>
      </xsl:when>
      <xsl:when test="dsrl:from">
        <xsl:if test="dsrl:within"><xsl:value-of select="dsrl:within"/>/</xsl:if>
        <xsl:value-of select="dsrl:from"/>
      </xsl:when>
    </xsl:choose>
  </xsl:variable>
  <xsl:variable name="new-name">
    <xsl:choose>
      <xsl:when test="dsrl:name">
        <xsl:value-of select="dsrl:name"/>
      </xsl:when>
```

```
      <xsl:when test="dsrl:to">
        <xsl:value-of select="dsrl:to"/>
      </xsl:when>
    </xsl:choose>
  </xsl:variable>
  <xs:template match="@{$target}">
   <xsl:if test="not(dsrl:to='')">
     <xs:attribute name="{$new-name}">
       <xsl:call-template name="values-map"/>
     </xs:attribute>
   </xsl:if>
  </xs:template>
</xsl:template>

<xsl:template name="attribute-map">
  <xs:choose>
    <xsl:for-each select="dsrl:attribute-map">
      <xsl:variable name="attribute-name">
        <xsl:if test="dsrl:name">
          <xsl:value-of select="dsrl:name"/>
        </xsl:if>
        <xsl:if test="dsrl:from">
          <xsl:value-of select="dsrl:from"/>
        </xsl:if>
      </xsl:variable>
      <xsl:variable name="new-attribute-name">
        <xsl:if test="dsrl:name">
          <xsl:value-of select="dsrl:name"/>
        </xsl:if>
        <xsl:if test="dsrl:to">
          <xsl:value-of select="dsrl:to"/>
        </xsl:if>
      </xsl:variable>
      <xs:when test="name()='{$attribute-name}'">
        <xsl:if test="not($new-attribute-name='')">
        <xs:attribute name="{$new-attribute-name}">
          <xsl:if test="dsrl:default-value[@force-default='true']">
            <xsl:value-of select="dsrl:default-value"/>
          </xsl:if>
          <xsl:if test="dsrl:default-value[@force-default='false']">
            <xs:if test=".=''">
              <xsl:value-of select="dsrl:default-value"/>
            </xs:if>
            <xs:if test="not(.='')">
              <xsl:call-template name="values-map"/>
            </xs:if>
          </xsl:if>
          <xsl:if test="not(dsrl:default-value)">
            <xsl:call-template name="values-map"/>
          </xsl:if>
        </xs:attribute></xsl:if>
      </xs:when>
    </xsl:for-each>
    <xs:otherwise>
      <xs:copy-of select="."/>
    </xs:otherwise>
  </xs:choose>
</xsl:template>

<xsl:template name="values-map">
```

```
    <xsl:choose>
      <xsl:when test="dsrl:values-map">
        <xs:choose>
          <xsl:for-each select="dsrl:values-map/dsrl:from">
            <xs:when test=".='{.}'">
              <xsl:value-of select="following-sibling::dsrl:to[1]"/>
            </xs:when>
          </xsl:for-each>
          <xs:otherwise>
            <xs:value-of select="."/>
          </xs:otherwise>
        </xs:choose>
      </xsl:when>
      <xsl:otherwise>
        <xs:value-of select="."/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template match="dsrl:map-pi-target">
    <xsl:for-each select="dsrl:from">
      <xs:template match="processing-instruction('{text()}')">
        <xs:text><xsl:text>
</xsl:text></xs:text>
        <xs:processing-instruction name="{following-sibling::dsrl:to[1]}">
          <!--Need to add rules to process dsrl:property-names here-->
          <xs:value-of select="."/>
        </xs:processing-instruction>
        <xs:text><xsl:text>
</xsl:text></xs:text>
      </xs:template>
    </xsl:for-each>
  </xsl:template>
  <xsl:template match="dsrl:entity-name-map">
    <xs:template match="text()">
      <xs:call-template name="find-entity-ref">
        <xs:with-param name="t" select="."/>
      </xs:call-template>
    </xs:template>

    <xs:template name="find-entity-ref">
      <!-- Converts entity references to a validatable form.
       Presumes that all entities mapped to are defined in the relevant document type.
       Remember that when XML schemas are being used for validation only the
       five predefined entities, amp, lt, gt, quot and apos are defined; any other name
       specified for validation will generate an error.
     -->
      <xs:param name="t"/>
      <xsl:variable name="entities">
        <xsl:for-each select="dsrl:from">
          <xsl:text disable-output-escaping="yes">&#60;</xsl:text>!ENTITY
           <xsl:value-of select="text()"/>
             "[[entity::<xsl:value-of select="text()"/>]]"
            <xsl:text disable-output-escaping="yes">&#62;
</xsl:text>
        </xsl:for-each>
        <xsl:for-each select="//dsrl:define-entity">
          <xsl:text disable-output-escaping="yes">&#60;</xsl:text>!ENTITY
          <xsl:value-of select="dsrl:from"/>
             "[[entity::<xsl:value-of select="dsrl:from"/>]]"
```

```
           <xsl:text disable-output-escaping="yes">&#62;
</xsl:text>
        </xsl:for-each>
      </xsl:variable>
      <xsl:result-document href="{$output-directory}{$entity-redefinition}"
        format="entity-definitions">
        <!-- This file should be used to update the entity definitions of all mapped
          entities by adding it to the end of the input DOCTYPE defintion.
          Typically this will involve the addition of a parameter entity with
          the following generalized form:
           <!ENTITY % mapped-entities SYSTEM "DSRLentities.ent"> %mapped-entities;
    -->
        <xsl:value-of select="$entities"/>

      </xsl:result-document>

      <xsl:result-document href="{$output-directory}{$MappedEntities}"
         format="mapped-entities" >
        <!-- This file should be used to update the entity definitions of all
          mapped entities by adding it to the end of the input DOCTYPE definition.
          Typically this will involve the addition of a parameter entity with
          the following generalized form:
            <!ENTITY % mapped-entities SYSTEM "MappedEntities.ent"> %mapped-entities;
    -->
         <xsl:for-each select="//dsrl:entity-name-map/dsrl:from">
          <xsl:if  test="not(following-sibling::dsrl:to[1]='amp') and
                           not(following-sibling::dsrl:to[1]='lt') and
                           not(following-sibling::dsrl:to[1]='gt') and
                           not(following-sibling::dsrl:to[1]='apos') and
                           not(following-sibling::dsrl:to[1]='quot')">
            <xsl:variable name="entity-definition2">
              [[Need definition for
                <xsl:value-of select="following-sibling::dsrl:to[1]"/> here]]
             </xsl:variable>
            <xsl:text disable-output-escaping="yes">&#60;</xsl:text>!ENTITY
             <xsl:value-of  select="following-sibling::dsrl:to[1]"/>
             "<xsl:value-of select="$entity-definition2"/>"
             <xsl:text disable-output-escaping="yes">&#62;
</xsl:text>
          </xsl:if>
        </xsl:for-each>
        <xsl:for-each select="//dsrl:define-entity">
         <xsl:text disable-output-escaping="yes">&#60;</xsl:text>!ENTITY
          <xsl:value-of select="dsrl:from"/>
          "<xsl:copy-of select="dsrl:replacement-text/node()" />"
        <xsl:text disable-output-escaping="yes">&#62;
 </xsl:text>
          </xsl:for-each>
      </xsl:result-document>
      <xs:choose>
        <xsl:for-each select="dsrl:from">
          <xsl:variable name="entity-name-entered">
            <xsl:value-of select="text()"/>
          </xsl:variable>
          <xsl:variable name="entity-to-be-validated">
            <xsl:value-of select="following-sibling::dsrl:to[1]"/>
          </xsl:variable>
          <xs:when test="contains($t, '[[entity::{$entity-name-entered}]]')">
            <xs:variable name="starts"
              select="substring-before($t, '[[entity::{$entity-name-entered}]]')"/>
```

```
            <xs:call-template name="find-entity-ref">
              <xs:with-param name="t" select="$starts"/>
            </xs:call-template>
              <xs:text><xsl:text disable-output-escaping="yes">&#38;</xsl:text>
              <xsl:value-of select="$entity-to-be-validated"/>;</xs:text><xs:variable
                name="ends"
                select="substring-after($t, '[[entity::{$entity-name-entered}]]')"/>
            <xs:call-template name="find-entity-ref">
              <xs:with-param name="t" select="$ends"/>
            </xs:call-template>
          </xs:when>
        </xsl:for-each>
        <xsl:for-each select="//dsrl:define-entity">
          <xsl:variable name="entity-name-entered">
            <xsl:value-of select="dsrl:from"/>
          </xsl:variable>
          <xsl:variable name="entity-to-be-validated">
            <xsl:value-of select="dsrl:from"/>
          </xsl:variable>
          <xs:when test="contains($t, '[[entity::{$entity-name-entered}]]')">
            <xs:variable name="starts"
              select="substring-before($t, '[[entity::{$entity-name-entered}]]')"/>
            <xs:call-template name="find-entity-ref">
              <xs:with-param name="t" select="$starts"/>
            </xs:call-template>
              <xsl:text disable-output-escaping="yes">&#38;</xsl:text>
              <xsl:value-of select="$entity-to-be-validated"/>;<xs:variable
                  name="ends"
                  select="substring-after($t, '[[entity::{$entity-name-entered}]]')"/>
            <xs:call-template name="find-entity-ref">
              <xs:with-param name="t" select="$ends"/>
            </xs:call-template>
          </xs:when>
        </xsl:for-each>
        <xs:otherwise>
          <xs:value-of select="$t"/>
        </xs:otherwise>
      </xs:choose>
    </xs:template>
  </xsl:template>

  <xsl:template match="dsrl:define-entity">
    <!--Entity definitions are mapped as part of entity maps. This empty
    definition ensures that their contents are discarded-->
    <!--May need to add functionality to process entity definitions when
    there is no entity map!--></xsl:template>
</xsl:stylesheet>
```

To demonstrate how this transformation can be applied, we will show how a file consisting of a number of French addresses, marked up using French element and attribute names, and referencing entities defined with French names, can be mapped to a schema for European addresses which uses English for its markup names. The example document to be transformed has the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="DSRLtransform.xsl"?>
<!DOCTYPE doc SYSTEM "Adresse.dtd" >
<?AlternativePIname PI proceeds root?>
<doc>
  <?PInameAsInput Embedded PI?>
  <adresse sorte="maison">
```

```
    <numero>&open-tag;this&and;that&close-tag; - the home of &xml;</numero>
    <rue location="12345.67890">Rue Bricot</rue>
    <ville required="true">Monmartre</ville>
    <cité>Paris</cité>
    <département numero="95">Île de France</département>
    <code-postal>F-95010</code-postal>
    <pays>France</pays>
  </adresse>
  <?MyPI Another mapped PI?>
  <adresse sorte="bureau">
    <numero>2</numero>
    <rue>Avenue Charles de Gaulle</rue>
    <cité>Toulon</cité>
    <département>Aud&ea; &et; Dauphine</département>
    <code-postal>F-12345</code-postal>
    <pays>France</pays>
  </adresse>
</doc>
<?AlternativePIname PI follows root?>
```

A simplified DTD that could be used to validate this document instance might have the form:

```
<!ELEMENT doc (adresse+) >
<!ELEMENT adresse (numero, rue, ville?, cité, département, code-postal, pays?)>
<!ATTLIST adresse   sorte CDATA #REQUIRED >
<!ELEMENT cité (#PCDATA)>
<!ELEMENT code-postal (#PCDATA)>
<!ELEMENT département (#PCDATA)>
<!ATTLIST département numero CDATA #IMPLIED>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT pays (#PCDATA)>
<!ELEMENT rue (#PCDATA)>
<!ATTLIST rue location CDATA #IMPLIED >
<!ELEMENT ville (#PCDATA)>
<!ATTLIST ville required (true|false|maybe) #REQUIRED >
```

The document will be validated against the following W3C XML Schema:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://csw.co.uk/addresses" targetNamespace="http://csw.co.uk/addresses"
 elementFormDefault="qualified">
  <xs:element name="address">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="building-identifier"/>
        <xs:element ref="road"/>
        <xs:element ref="locality" minOccurs="0"/>
        <xs:element ref="postal-town"/>
        <xs:element ref="county" minOccurs="0"/>
        <xs:element ref="postcode"/>
        <xs:element ref="country" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="type" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="home"/>
            <xs:enumeration value="office"/>
          </xs:restriction>
        </xs:simpleType>
```

```
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="building-identifier">
    <xs:simpleType>
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="country">
    <xs:simpleType>
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="county">
    <xs:simpleType>
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="locality">
    <xs:simpleType>
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="postal-town">
    <xs:simpleType>
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="postcode">
    <xs:simpleType>
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="road">
    <xs:simpleType>
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="doc">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="address" maxOccurs="unbounded"/>
        <xs:any namespace="xi"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The map used to transform an instance marked up using the DTD has the form:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="TransformDSRLmaps.xsl"?>
<dsrl:maps xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl" xmlns=""
  targetNamespace="http://csw.co.uk/addresses"
  targetSchemaLocation="EuropeanAddress.xsd">

  <!--Mapping of element and attribute names and attribute values-->
  <dsrl:element-map>
    <dsrl:from>doc</dsrl:from>
    <dsrl:to>document</dsrl:to>
```

```
  </dsrl:element-map>

  <dsrl:element-map>
    <dsrl:from>adresse</dsrl:from>
    <dsrl:to>address</dsrl:to>
    <dsrl:attribute-map>
      <dsrl:from>sorte</dsrl:from>
      <dsrl:to>type</dsrl:to>
      <dsrl:values-map>
        <dsrl:from>maison</dsrl:from>
        <dsrl:to>home</dsrl:to>
        <dsrl:from>bureau</dsrl:from>
        <dsrl:to>office</dsrl:to>
      </dsrl:values-map>
    </dsrl:attribute-map>
  </dsrl:element-map>

  <dsrl:element-map>
    <dsrl:from>numero</dsrl:from>
    <dsrl:to>building-identifier</dsrl:to>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:from>rue</dsrl:from>
    <dsrl:to>road</dsrl:to>
    <dsrl:attribute-map>
      <dsrl:from>location</dsrl:from>
      <dsrl:to></dsrl:to>
    </dsrl:attribute-map>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:within>adresse</dsrl:within>
    <dsrl:from>ville</dsrl:from>
    <dsrl:to>locality</dsrl:to>
    <dsrl:attribute-map>
      <dsrl:name>required</dsrl:name>
      <dsrl:default-value force-default="true">maybe</dsrl:default-value>
    </dsrl:attribute-map>
    <dsrl:attribute-map>
      <dsrl:name>imported</dsrl:name>
      <dsrl:default-value force-default="false">what</dsrl:default-value>
    </dsrl:attribute-map>
    <dsrl:default-content force-default="true" after="rue">Downtown</dsrl:default-content>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:from>cité</dsrl:from>
    <dsrl:to>postal-town</dsrl:to>
    <dsrl:default-content force-default="false" after="ville">Bordeaux</dsrl:default-content>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:from>département</dsrl:from>
    <dsrl:to>county</dsrl:to>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:from>code-postal</dsrl:from>
    <dsrl:to>postcode</dsrl:to>
  </dsrl:element-map>
  <dsrl:element-map>
    <dsrl:from>pays</dsrl:from>
    <dsrl:to>country</dsrl:to>
    <dsrl:attribute-map>
```

```
      <dsrl:name>code-system</dsrl:name>
      <dsrl:default-value force-default="true">iso3166</dsrl:default-value>
    </dsrl:attribute-map>
    <dsrl:attribute-map>
      <dsrl:name>ISO-code</dsrl:name>
      <dsrl:default-value force-default="false">FR</dsrl:default-value>
    </dsrl:attribute-map>
  </dsrl:element-map>

 <dsrl:attribute-map>
   <dsrl:from>numero</dsrl:from>
   <dsrl:to></dsrl:to>
 </dsrl:attribute-map>

  <dsrl:map-pi-target>
    <dsrl:from>PInameAsInput</dsrl:from>
    <dsrl:to>PIname</dsrl:to>
    <dsrl:from>AlternativePIname</dsrl:from>
    <dsrl:to>PIname</dsrl:to>
    <dsrl:from>MyPI</dsrl:from>
    <dsrl:to>ProcessThis</dsrl:to>
  </dsrl:map-pi-target>

  <dsrl:entity-name-map>
    <dsrl:from>e</dsrl:from>
    <dsrl:to>eacute</dsrl:to>
    <dsrl:from>et</dsrl:from>
    <dsrl:to>amp</dsrl:to>
    <dsrl:from>and</dsrl:from>
    <dsrl:to>amp</dsrl:to>
    <dsrl:from>open-tag</dsrl:from>
    <dsrl:to>lt</dsrl:to>
    <dsrl:from>close-tag</dsrl:from>
    <dsrl:to>gt</dsrl:to>
  </dsrl:entity-name-map>
  <dsrl:define-entity>
    <dsrl:from>ea</dsrl:from>
    <dsrl:replacement-text>&#233;</dsrl:replacement-text>
  </dsrl:define-entity>
  <dsrl:define-entity>
    <dsrl:from>xml</dsrl:from>
    <dsrl:replacement-text>
       <![CDATA[the W3C Extensible Markup Language (<acronym>XML</acronym>)]] >
    </dsrl:replacement-text>
  </dsrl:define-entity>

</dsrl:maps>
```

NOTE 15: This map contains some conversions, such as those for processing instructions, whose only real purpose is to illustrate the application of each of the features of DSRL. In practice maps will not normally include all of the DSRL constructs, as this example does.

When transformed using the XSLT 2.0 transformation shown at the start of this clause, the following XSLT transformation is generated:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsl:stylesheet [
<!ENTITY % MappedEntities SYSTEM "MappedEntities.ent"> %MappedEntities;
]>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
               xmlns="http://csw.co.uk/addresses"
               version="1.0">
 <xsl:output method="xml" indent="yes"/>

<xsl:template match="doc">
    <xsl:element name="document">
       <xsl:apply-templates select="@*"/>
       <xsl:apply-templates/>
    </xsl:element>
 </xsl:template>
<xsl:template match="adresse">
    <xsl:element name="address">
       <xsl:for-each select="@*">
          <xsl:choose>
             <xsl:when test="name()='sorte'">
                <xsl:attribute name="type">
                   <xsl:choose>
                      <xsl:when test=".='maison'">home</xsl:when>
                      <xsl:when test=".='bureau'">office</xsl:when>
                      <xsl:otherwise>
                         <xsl:value-of select="."/>
                      </xsl:otherwise>
                   </xsl:choose>
                </xsl:attribute>
             </xsl:when>
             <xsl:otherwise>
                <xsl:copy-of select="."/>
             </xsl:otherwise>
          </xsl:choose>
       </xsl:for-each>
       <xsl:apply-templates/>
    </xsl:element>
 </xsl:template>
<xsl:template match="numero">
    <xsl:element name="building-identifier">
       <xsl:apply-templates select="@*"/>
       <xsl:apply-templates/>
    </xsl:element>
 </xsl:template>
<xsl:template match="rue">
    <xsl:element name="road">
       <xsl:for-each select="@*">
          <xsl:choose>
             <xsl:when test="name()='location'"/>
             <xsl:otherwise>
                <xsl:copy-of select="."/>
             </xsl:otherwise>
          </xsl:choose>
       </xsl:for-each>
       <xsl:apply-templates/>
    </xsl:element>
 </xsl:template>
<xsl:template match="adresse/ville">
    <xsl:element name="locality">
       <xsl:for-each select="@*">
          <xsl:choose>
             <xsl:when test="name()='required'">
                <xsl:attribute name="required">maybe</xsl:attribute>
             </xsl:when>
             <xsl:when test="name()='imported'">
```

```
               <xsl:attribute name="imported">
                  <xsl:if test=".=''">what</xsl:if>
                  <xsl:if test="not(.='')">
                     <xsl:value-of select="."/>
                  </xsl:if>
               </xsl:attribute>
            </xsl:when>
            <xsl:otherwise>
               <xsl:copy-of select="."/>
            </xsl:otherwise>
         </xsl:choose>
      </xsl:for-each>
      <xsl:if test="not(@required)">
         <xsl:attribute name="required">maybe</xsl:attribute>
      </xsl:if>
      <xsl:if test="not(@imported)">
         <xsl:attribute name="imported">what</xsl:attribute>
      </xsl:if>Downtown</xsl:element>
</xsl:template>
<xsl:template match="adresse" priority="1">
   <xsl:element name="address">
      <xsl:for-each select="@*">
         <xsl:choose>
            <xsl:when test="name()='sorte'">
               <xsl:attribute name="type">
                  <xsl:choose>
                     <xsl:when test=".='maison'">home</xsl:when>
                     <xsl:when test=".='bureau'">office</xsl:when>
                     <xsl:otherwise>
                        <xsl:value-of select="."/>
                     </xsl:otherwise>
                  </xsl:choose>
               </xsl:attribute>
            </xsl:when>
            <xsl:otherwise>
               <xsl:copy-of select="."/>
            </xsl:otherwise>
         </xsl:choose>
      </xsl:for-each>
      <xsl:apply-templates/>
   </xsl:element>
</xsl:template>
<xsl:template match="rue" priority="1">
   <xsl:element name="road">
      <xsl:for-each select="@*">
         <xsl:choose>
            <xsl:when test="name()='location'"/>
            <xsl:otherwise>
               <xsl:copy-of select="."/>
            </xsl:otherwise>
         </xsl:choose>
      </xsl:for-each>
      <xsl:apply-templates/>
   </xsl:element>
   <xsl:if test="not(../ville)">
      <xsl:element name="locality">
         <xsl:attribute name="required">maybe</xsl:attribute>
         <xsl:attribute name="imported">what</xsl:attribute>Downtown</xsl:element>
   </xsl:if>
</xsl:template>
```

```
 <xsl:template match="cité">
     <xsl:element name="postal-town">
        <xsl:apply-templates select="@*"/>
        <xsl:if test=".=''">Bordeaux</xsl:if>
        <xsl:if test="not(.='')">
           <xsl:value-of select="."/>
        </xsl:if>
     </xsl:element>
  </xsl:template>
 <xsl:template match="département">
     <xsl:element name="county">
        <xsl:apply-templates select="@*"/>
        <xsl:apply-templates/>
     </xsl:element>
  </xsl:template>
 <xsl:template match="code-postal">
     <xsl:element name="postcode">
        <xsl:apply-templates select="@*"/>
        <xsl:apply-templates/>
     </xsl:element>
  </xsl:template>
 <xsl:template match="pays">
     <xsl:element name="country">
        <xsl:for-each select="@*">
           <xsl:choose>
              <xsl:when test="name()='code-system'">
                 <xsl:attribute name="code-system">iso3166</xsl:attribute>
              </xsl:when>
              <xsl:when test="name()='ISO-code'">
                 <xsl:attribute name="ISO-code">
                    <xsl:if test=".=''">FR</xsl:if>
                    <xsl:if test="not(.='')">
                       <xsl:value-of select="."/>
                    </xsl:if>
                 </xsl:attribute>
              </xsl:when>
              <xsl:otherwise>
                 <xsl:copy-of select="."/>
              </xsl:otherwise>
           </xsl:choose>
        </xsl:for-each>
        <xsl:if test="not(@code-system)">
           <xsl:attribute name="code-system">iso3166</xsl:attribute>
        </xsl:if>
        <xsl:if test="not(@ISO-code)">
           <xsl:attribute name="ISO-code">FR</xsl:attribute>
        </xsl:if>
        <xsl:apply-templates/>
     </xsl:element>
  </xsl:template>
  <xsl:template match="@numero"/>
 <xsl:template match="processing-instruction('PInameAsInput')">
     <xsl:text>
</xsl:text>
     <xsl:processing-instruction name="PIname">
        <xsl:value-of select="."/>
     </xsl:processing-instruction>
     <xsl:text>
</xsl:text>
  </xsl:template>
```

```
  <xsl:template match="processing-instruction('AlternativePIname')">
     <xsl:text>
</xsl:text>
     <xsl:processing-instruction name="PIname">
        <xsl:value-of select="."/>
     </xsl:processing-instruction>
     <xsl:text>
</xsl:text>
  </xsl:template>
  <xsl:template match="processing-instruction('MyPI')">
     <xsl:text>
</xsl:text>
     <xsl:processing-instruction name="ProcessThis">
        <xsl:value-of select="."/>
     </xsl:processing-instruction>
     <xsl:text>
</xsl:text>
  </xsl:template>

 <xsl:template match="text()">
     <xsl:call-template name="find-entity-ref">
        <xsl:with-param name="t" select="."/>
     </xsl:call-template>
  </xsl:template>
  <xsl:template name="find-entity-ref">
     <xsl:param name="t"/>
     <xsl:choose>
        <xsl:when test="contains($t, '[[entity::e]]')">
           <xsl:variable name="starts"
                         select="substring-before($t, '[[entity::e]]')"/>
           <xsl:call-template name="find-entity-ref">
              <xsl:with-param name="t" select="$starts"/>
           </xsl:call-template>
           <xsl:text>&eacute;</xsl:text>
           <xsl:variable name="ends"
                         select="substring-after($t, '[[entity::e]]')"/>
           <xsl:call-template name="find-entity-ref">
              <xsl:with-param name="t" select="$ends"/>
           </xsl:call-template>
        </xsl:when>
        <xsl:when test="contains($t, '[[entity::et]]')">
           <xsl:variable name="starts"
                         select="substring-before($t, '[[entity::et]]')"/>
           <xsl:call-template name="find-entity-ref">
              <xsl:with-param name="t" select="$starts"/>
           </xsl:call-template>
           <xsl:text>&amp;</xsl:text>
           <xsl:variable name="ends"
                         select="substring-after($t, '[[entity::et]]')"/>
           <xsl:call-template name="find-entity-ref">
              <xsl:with-param name="t" select="$ends"/>
           </xsl:call-template>
        </xsl:when>
        <xsl:when test="contains($t, '[[entity::and]]')">
           <xsl:variable name="starts"
                         select="substring-before($t, '[[entity::and]]')"/>
           <xsl:call-template name="find-entity-ref">
              <xsl:with-param name="t" select="$starts"/>
           </xsl:call-template>
           <xsl:text>&amp;</xsl:text>
```

```
      <xsl:variable name="ends"
                    select="substring-after($t, '[[entity::and]]')"/>
      <xsl:call-template name="find-entity-ref">
         <xsl:with-param name="t" select="$ends"/>
      </xsl:call-template>
   </xsl:when>
   <xsl:when test="contains($t, '[[entity::open-tag]]')">
      <xsl:variable name="starts"
                    select="substring-before($t, '[[entity::open-tag]]')"/>
      <xsl:call-template name="find-entity-ref">
         <xsl:with-param name="t" select="$starts"/>
      </xsl:call-template>
      <xsl:text>&lt;</xsl:text>
      <xsl:variable name="ends"
                    select="substring-after($t, '[[entity::open-tag]]')"/>
      <xsl:call-template name="find-entity-ref">
         <xsl:with-param name="t" select="$ends"/>
      </xsl:call-template>
   </xsl:when>
   <xsl:when test="contains($t, '[[entity::close-tag]]')">
      <xsl:variable name="starts"
                    select="substring-before($t, '[[entity::close-tag]]')"/>
      <xsl:call-template name="find-entity-ref">
         <xsl:with-param name="t" select="$starts"/>
      </xsl:call-template>
      <xsl:text>&gt;</xsl:text>
      <xsl:variable name="ends"
                    select="substring-after($t, '[[entity::close-tag]]')"/>
      <xsl:call-template name="find-entity-ref">
         <xsl:with-param name="t" select="$ends"/>
      </xsl:call-template>
   </xsl:when>
   <xsl:when test="contains($t, '[[entity::ea]]')">
      <xsl:variable name="starts"
                    select="substring-before($t, '[[entity::ea]]')"/>
      <xsl:call-template name="find-entity-ref">
         <xsl:with-param name="t" select="$starts"/>
      </xsl:call-template>&ea;
         <xsl:variable name="ends"
                    select="substring-after($t, '[[entity::ea]]')"/>
      <xsl:call-template name="find-entity-ref">
         <xsl:with-param name="t" select="$ends"/>
      </xsl:call-template>
   </xsl:when>
   <xsl:when test="contains($t, '[[entity::xml]]')">
      <xsl:variable name="starts"
                    select="substring-before($t, '[[entity::xml]]')"/>
      <xsl:call-template name="find-entity-ref">
         <xsl:with-param name="t" select="$starts"/>
      </xsl:call-template>&xml;
         <xsl:variable name="ends"
                    select="substring-after($t, '[[entity::xml]]')"/>
      <xsl:call-template name="find-entity-ref">
         <xsl:with-param name="t" select="$ends"/>
      </xsl:call-template>
   </xsl:when>
   <xsl:otherwise>
      <xsl:value-of select="$t"/>
   </xsl:otherwise>
</xsl:choose>
```

```
   </xsl:template>
   <xsl:template match="*|@*">
      <xsl:copy>
         <xsl:apply-templates select="@*"/>
         <xsl:apply-templates
                        select="*|text()|comment()|processing-instruction()"/>
      </xsl:copy>
   </xsl:template>
</xsl:stylesheet>
```

When this map is used to transform the French document instance the following European-style address is produced:

```
<?xml version="1.0" encoding="UTF-8"?>
 <?PIname PI proceeds root?>
 <document xmlns="http://csw.co.uk/addresses">
 <?PIname Embedded PI?>
 <address type="home">
      <building-identifier>&lt;this&amp;that&gt; - the home of
        the W3C Extensible Markup Language (<acronym>XML</acronym>)
      </building-identifier>
      <road>Rue Bricot</road>
      <locality required="maybe" imported="what">Downtown</locality>
      <postal-town>Paris</postal-town>
      <county>Île de France</county>
      <postcode>F-95010</postcode>
      <country code-system="iso3166" ISO-code="FR">France</country>
   </address>
   <?ProcessThis Another mapped PI?>
 <address type="office">
      <building-identifier>2</building-identifier>
      <road>Avenue Charles de Gaulle</road>
      <locality required="maybe" imported="what">Downtown</locality>
      <postal-town>Toulon</postal-town>
      <county>Audé &amp; Dauphine</county>
      <postcode>F-12345</postcode>
      <country code-system="iso3166" ISO-code="FR">France</country>
   </address>
</document>
 <?PIname PI follows root?>
```

B.2 Using DSRL rules to localize schemas

DSRL rules can be used to create a localized version of an existing schema using the following XSLT transform:

```
To be defined
```

Bibliography

[1] *XSL Transformations (XSLT) Version 2.0*, http://www.w3.org/TR/2006/CR-xslt20-20060608/

Summary of editorial comments:

[5] DSRL maps

Are there any circumstances in which the target namespace (or schema location) are required to be specified?

[6.2] Mapping attribute value tokens

Should it be possible to record original attribute values in a DSRL PI?

[6.4] Default content

Should the after attribute identify the element after which the default is to be inserted in the result schema rather than the current definition, which is based on which source element it needs to follow? (NB Mapping to the source element makes the XSLT easier!)

[6.5] Renaming processing instruction targets

Should it be possible to record original processing instruction names in a DSRL PI, or as an DSRL property of the mapped processing instruction?