U/OO/234056-17

Using Schematron for Cross Domain Security Policy Enforcement

Version 1.0
29 June 2012

National Security Agency
9800 Savage Rd, Suite 6721
Ft. George G. Meade. MD 20755

Authored/Released by:
Unified Cross Domain Capabilities Office
cds_tech@nsa.gov

DOCUMENT CHANGE HISTORY

Date	Version	Description
06/29/2012	1.0	Initial Release
12/13/2017	1.0	Updated Contact information, IAC Logo, Cited Trademarks and Copyrights, Expanded Acronyms, and added Legal Disclaimer

DISCLAIMER OF WARRANTIES AND ENDORSEMENT

The information and opinions contained in this document are provided "as is" and without any warranties or guarantees. References herein to any specific commercial product, process, or service by trade name, trademark, manufacturer or otherwise, does not constitute or imply its endorsement, recommendation or favoring by the United States Government and this guidance shall not be used for advertising or product endorsement purposes.

.

ABSTRACT

This document provides recommendations for using Schematron schemas to enforce data constraints on the contents of Extensible Markup Language (XML) documents being transferred between security domains. Schematron is a rule-based schema language used for making assertions about patterns found in XML documents. The risk of transferring invalid or unauthorized XML data into or out of a sensitive security domain can be reduced by validating the XML data against a schema that fully describes and constrains the data. These more restrictive schemas are not necessarily the same as those that might be used to validate data being transferred within a single security domain. Schematron can be used as part of a Cross Domain Solution (CDS) to address security problems that may be difficult to solve using grammar-based XML Schema languages.

The document also discusses some of the data constraints that Schematron can easily express, notably, data cardinality and co-constraints, and additional features that Schematron provides. Schematron can be used to supplement the validation capabilities of a grammar-based XML Schema language. Using both rule-based and grammar-based schema languages should increase the robustness of the validation capabilities of CDSs, thereby reducing the risk of transferring malicious or unauthorized XML data between security domains.

Disclaimer

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Purpose

The *XML Schema and RELAX NG Guidance for Cross Domain Security Policy Enforcement* [1] document provides guidance for using grammar-based schema languages to create schemas that can help validate the format and contents of Extensible Markup Language (XML) [2] instance documents being transferred between security domains. This document provides recommendations for using the rule-based Schematron schema language to create schemas containing business and operational rules that further constrain the format and contents of such documents.

The Schematron schema language can easily express data constraints such as data cardinality and co-constraints, which would be difficult to express using grammar-based schema languages alone. Validating XML instance documents against properly designed grammar-based and rule-based schemas can reduce the risk of transferring unauthorized data between security domains. These schemas can therefore help enforce Department of Defense (DoD) and Intelligence Community (IC) cross domain security policies [3].

1.1 Background

XML is widely used to alleviate many of the interoperability problems associated with sharing data. XML instance documents[1] can contain virtually any type of data, such as command and control information, procurement orders, personnel records, or laboratory results. To define the structure and allowable contents for each type of XML instance document, users can create a schema document using a schema language such as XML Schema [4, 5, 6], RELAX NG [7], or Schematron [8].

There are two types of schema languages: grammar-based languages and rule-based languages. Grammar-based schema languages, such as XML Schema and Regular Language for XML Next Generation (RELAX NG), specify the structure, form, and syntax of elements and attributes in an XML instance document. A grammar-based schema language can be used to constrain the contents and data type of each element or attribute in an XML instance document.

Wherever this document mentions XML Schema, it is referring to XML Schema 1.0. XML Schema 1.1 has the ability to express many relationships between data, and thus may eventually replace Schematron for some things; however, at the time of this writing virtually all schemas are written using XML Schema 1.0. Further, XML Schema 1.1 does not allow the expression of inter-document rules, whereas Schematron does, so even if XML Schema 1.1 does become the dominate schema language there will still be a need for Schematron.

[1] Each XML document that conforms to a class of XML data is called an "XML instance document."

Unlike grammar-based XML schema languages, Schematron is a rule-based (or assertion-based) schema language that specifies the relationships that must hold between the elements and attributes in an XML instance document. A rule-based schema language can be used to verify data constraints within an XML instance document.

A schema that is acceptable for defining XML data being exchanged within a single security domain[2] might not be sufficient for describing XML data that is authorized for transfer between security domains.

1.2 Key Premise

The key premise of this document is that XML instance documents being transferred between security domains can be more completely validated by adding a step to the validation process that checks for any data constraints. This step would confirm that the format and contents of the XML instance document conform to multiple schemas (schemas that properly incorporate all security policy requirements), which would help ensure that the cross domain transfer complies with the security policy.

"This premise applies whether transferring XML data into or out of a security domain. For example, when exporting data out of a sensitive security domain, the primary information assurance requirement is to protect the confidentiality of sensitive information that is not authorized for transfer. If the data that is authorized for transfer can be properly and completely described in a schema, then data that has been validated and filtered will consist of authorized data. Similarly, when transferring data into a sensitive security domain, a key requirement is to protect against importing malicious data that might violate the security of the sensitive domain. If a schema accurately describes the data that is to be imported, then the validation process in conjunction with content filtering will allow only acceptable data to enter the sensitive domain." [1]

[2] The term "security domain" includes the hierarchical security classifications Unclassified, Confidential, Secret, and Top Secret as well as the foreign (disclosure) domains.

1.3 Document Organization

An assumption has been made that the reader has a basic understanding of XML technologies and cross domain security. The document is organized as follows:

- Section 1 presents introductory information, which includes the document's purpose, background, and key premise.
- Section 2 provides additional background information about the use of XML technologies for enforcing cross domain security policies.
- Section 3 provides an overview of the Schematron schema language and the reasons why it is important.
- Section 4 provides recommendations for using Schematron to enforce data constraints, such as cardinality and co-constraints, in a Cross Domain Solution (CDS).
- Section 5 lists additional features that Schematron offers that may prove useful to a CDS.
- Section 6 discusses Schematron implementation decisions and issues.
- Section 7 provides conclusions and recommendations.
- Appendix A provides a more complete introduction to Schematron.
- Appendix B provides references.
- Appendix C presents a glossary of acronyms and commonly used terms.

2 BACKGROUND

This section provides additional background information about the use of XML technologies for enforcing cross domain security policies. It focuses on schemas in cross domain security, DoD and IC policies governing cross domain data transfers, and the rationale for supporting net-centric enterprise architectures.

2.1 Schemas in Cross Domain Security

This document, like the *XML Schema and RELAX NG Guidance for Cross Domain Security Policy Enforcement* document [1], "proposes a far stricter use of schemas than is common within commercial, DoD, or IC settings. For example, XML standards do not require XML instance documents to be validated against a schema."

Reference [1] states, "It is important to understand what ―validation means in this context. Validation goes well beyond well-formedness. Well-formed XML complies with syntax rules as defined in the XML specification and does not need to conform to a schema. Data in an XML instance document is said to be valid if it is well-formed and conforms to the definitions in the appropriate schema. For example, if the schema restricts a value to either one or zero, any other value is invalid. However, validation cannot determine if the data is correct. Additional content filters and/or Reliable Human Review may be required to determine if the data is correct. In the same example, validation cannot determine if the value is supposed to be one or zero because both values are valid. Similarly, the validation is only as good as the quality of the schema and the tool used to perform the validation. If the schema is poorly written or if the validation tool has flaws, the validation tool might incorrectly indicate the validity of the XML instance document."

It also says, "XML users sometimes decide against creating schemas or validating XML instance documents because omitting these steps might reduce costs, simplify processing, or make the XML instance document more flexible. Even when users validate XML instance documents, they often use very flexible schemas that do not constrain the format or contents. This section discusses why a much stricter use of XML is recommended in a cross domain environment" [1].

The subsections below discuss in greater detail the following XML validation concerns within cross domain security:

- Validation is needed at security domain boundaries.
- Validating XML documents might not be sufficient.
- More than one validation action might be necessary.

2.1.1 Validation Is Needed at Security Domain Boundaries

"In a cross domain environment, the data being transferred must be validated. Current DoD and IC security policies governing cross domain data transfers require that either an authorized individual must approve the transfer, or, in the case of fixed formatted messages, an automated process may be approved to use for validation prior to data transfer." [See also Section 2.2 below.]

"The requirement for this compliance means that organizations might need to strengthen their existing schemas or create a second set of more restrictive schemas if they want to use schemas to assist in validating the XML data being transferred between security

domains. These organizations could continue to use their original schemas for data being transferred within a single security domain, but they would need to use the newly created, more constraining schemas to validate the XML data prior to transferring it between security domains" [1].

Section 4 provides recommendations that might guide the design of additional schemas to enforce a cross domain security policy. Organizations can consider these recommendations to determine whether or not to use such schemas. It is assumed that technologies will be available for properly validating XML documents against these more restrictive (constraining) schemas; however, not all available XML tools may be useful and accreditable for transferring XML data between security domains.

2.1.2 Validating XML Documents Might Not Be Sufficient

"The cross domain security policy might require data checking that cannot be performed via schema validation. This guidance document does not advocate that format and content validation is sufficient to enforce all security policies. For example, a policy might require the data or format to be changed. Or, a rule is enforced only if the document contains a certain value. Schema validation does not change or restructure XML documents, and it is not designed to provide an if-then-else validation capability. Other tools are necessary to do this."

"Other data checks might reduce the security risk of a cross domain data transfer, but they cannot be performed by schema validation. For example, authenticating the sender can be used to prohibit data transfers from unauthorized sources. Similarly, checking data integrity helps to reduce risk by rejecting data that has been changed without authorization during transmission. These types of data checks are beyond the scope of this document because validation against a schema will not perform these data checks" [1].

Even though schema validation alone may not be a sufficient check in some cases, it is still important and necessary, as described in section 2.1.1. The following quotation from [1] further reinforces this point: "...it remains important to validate the format and contents of XML data being transferred between security domains. An authenticated sender might transfer unauthorized XML data, with its integrity intact. Schema validation cannot authenticate the sender or check the data integrity, but it can identify data in the wrong format or that contains invalid data. Therefore, even with authenticity and integrity checks, the risk can be reduced by examining and validating the data being transferred."

2.1.3 More Than One Validation Action Might Be Necessary

"Multiple validation actions might be necessary in a cross domain environment. In some cases, XML instance documents that are being transferred between security domains will be changed before entering the recipient's security domain. Validation against a schema should be applied before and after the instance document is changed, to ensure that the document was valid before being altered and that the changed result was still valid when it arrived at the recipient's security domain. The guidance in this document applies to all schemas used in this manner" [1].

2.2 DoD and IC Policies Governing Cross Domain Data Transfers

"DoD policy [9] (and also a draft IC policy [10]) allows for the automated cross domain transfer of highly structured/formatted 7-bit American Standard Code for Information Interchange (ASCII) text without human review. These policies apply to all documents that contain highly structured or formatted text, including XML instance documents."

"Properly designed schemas can enforce these format and content restrictions for XML instance documents. Similarly, validation tools can be designed to ensure that the format and contents of each XML instance document conform to a properly constraining schema during cross domain transfers" [1].

Echoing reference [1], the recommendations presented in this document are intended to help organizations produce properly designed Schematron schemas that "reduce risks when used to validate XML instance documents that are being transferred between security domains. An accredited cross domain controlled interface can validate XML instance documents against appropriate (constraining) schemas. If schemas meet all DoD and IC policy requirements for use with cross domain-controlled interfaces, then reliable human review of highly structured XML instance documents might not be required."

"However, not all XML documents are highly structured, nor are they restricted to 7-bit ASCII text[3]. The guidance in this document applies to highly structured ASCII text documents. The presence of non-ASCII characters[4] in an XML instance document increases the risk of transferring unauthorized data. Additionally, it might be possible to specify schema guidance for highly structured or formatted *binary* data that has been encoded in base 64 or hexadecimal text, but research is needed to confirm or refute this possibility."

In general, to comply with DoD and IC security policies when using available XML technologies, this document recommends a rather conservative approach. However, there is no guarantee that a schema conforming to the guidance herein can be used and accredited in a cross domain controlled interface. Organizations should consult with security engineers and evaluators who have XML-based cross domain security expertise to implement a practical XML-based cross domain capability" [1] at the lowest cost.

2.3 Rationale for Supporting Net-centric Enterprise Architectures

In late 2005, the Deputy Assistant Secretary of Defense for Information Assurance identified the ability for CDSs to process XML instance documents as a critical enabling technology required for support of emerging net-centric enterprise architectures. Since 2006, NSA's Cross Domain Products Division (of the Information Assurance Directorate) has engaged with the CDS community to enhance the ability of its products to process XML data. The NSA XML CDS program funded and tasked certain CDS projects to incorporate XML technologies into their respective CDS technologies.[5] These XML technologies add the ability to validate XML instance documents in accordance with XML Schema, RELAX NG, and Schematron standards and to support the Extensible Stylesheet Language Transformation (XSLT) transformation standard.

This document is one of a set that provides guidance on how XML Schema, RELAX NG, Schematron, and XSLT can be used in a CDS to process a cross domain flow consisting of XML instance documents.

[3] Refer to the previously referenced DoD and IC policies to determine how to transfer unstructured, binary, or free-text data between security domains.

[4] It is possible to mitigate the increased risk caused by transferring an XML instance document that contains non-ASCII characters. The instance document must be validated against an XML Schema that enumerates a list of authorized text strings containing the non-ASCII characters.

[5] JFCOM: DataSync Guard; SPAWAR: Radiant Mercury guard; and AFRL: ISSE guard.

3 OVERVIEW OF SCHEMATRON SCHEMA LANGUAGE

This section provides an overview of the Schematron schema language. It presents a description of Schematron technology, discusses the advantages and disadvantages of its use, and identifies versions of Schematron and related technologies.

3.1 Schematron Description

Schematron is a rule-based schema language used for making assertions about patterns found in XML documents. A Schematron schema can contain business and operational rules that constrain the format and contents of XML data. The schema can be used to determine the validity of XML instance documents.

Grammar-based schema languages, such as XML Schema and RELAX NG, specify the structure, form, and syntax of elements and attributes in an XML instance document. A grammar-based schema language may be used to constrain the contents and data type of each element or attribute in an XML instance document. Unlike grammar-based XML Schema languages, Schematron is a rule-based (or assertion-based) schema language that specifies the relationships that must hold between the elements and attributes in an XML instance document. Schematron expresses assertions using the XML Path Language (XPath); the assertions can contain any function supported by an XPath statement or XSLT test condition. Schematron is well suited for verifying data constraints that cannot be easily expressed using grammar-based schema languages, such as co-constraints, cardinality, and algorithmic checks. For a more complete overview of Schematron, please see Appendix A.

3.2 The Value of Schematron

Although grammar-based schema languages are very useful for validating XML documents, they cannot perform certain types of validation. In particular, they have limited ability to express data constraints. For example, although they can detail the proper format of the time of departure (from a source) or the time of arrival, they cannot express the requirement that the departure time must be before the arrival time.

Schematron is well suited to expressing validation requirements that grammar-based languages cannot, especially validation requirements known as “data constraints.” A data constraint is a condition that must hold within a single document or across documents that are being released within the same package.

Schematron has additional benefits compared to current grammar-based languages because it uses XPath expressions and has the ability to produce detailed error messages. Schematron specifies which relationships and patterns should hold true in the data by making assertions using XPath expressions. It uses the XPath expressions to locate and evaluate data in specific context paths within a parsed XML instance document. Schematron designers can create detailed diagnostic messages using plain language that can be displayed when an assertion fails as well as when an assertion is satisfied. For example, if an XML instance document fails to meet an assertion, Schematron will display the diagnostic message supplied by the author of the Schematron schema.

This document recommends that CDSs use Schematron to enforce applicable data constraints, in addition to using a grammar-based language to enforce content and format rules. When the two are used together, they can provide a more robust validation capability.

Use of Schematron to replace a grammar-based language is not recommended. There is some overlap between Schematron and grammar-based languages; that is, Schematron can perform some of the same types of validation as grammar-based languages. For example, both can express the requirement that the value of an element should be a value from an enumerated list. However, this document recommends that CDSs use grammar-based languages to express requirements for structure, form, and syntax because these languages are more fully capable of performing this function.

3.3 Advantages of Using Schematron

Although it is possible to implement the schema validation capabilities of Schematron using a lower level language, such as XSLT 2.0, Java, or C, using Schematron has several advantages:

- Schematron is an XML community-adopted approach for rule-based schema validation.
- Schematron is designed to be used for schema validation; therefore, it performs validation tasks well.
- Schematron has a limited, predefined set of capabilities; therefore, there is less of a learning curve.
- Schematron is a declarative language, which allows the developer to focus more on the task at hand and less on the programming language.
- Schematron uses natural language assertions; the results, especially error messages, are easier for humans to interpret.
- Schematron is a (comparatively) simple language, which should require fewer lines of code and less time to write.

None of these advantages compels developers to use Schematron; they can use a lower level language if they wish; however, these benefits, when considered collectively, are compelling reasons to recommend using Schematron.

3.4 Schematron Versions

There are two major versions of Schematron: International Organization for Standardization (ISO) Schematron and Schematron 1.5.[6] The recommendations in this document comply with both versions, although using ISO Schematron has two advantages. First, ISO Schematron is managed by an international standards body, the ISO, and by the International Electrotechnical Commission (IEC) [8]. As such, it is part of the ISO Document Schema Description Language (DSDL) standard, which is designed to allow multiple, well-focused XML validation languages to work together. Second, ISO Schematron offers more features than the older Schematron 1.5. For example, ISO Schematron supports the use of abstract patterns, which promote reuse. ISO Schematron also introduces the `let` element, which allows schemas to declare variables and thus simplifies the implementation of algorithmic checking.

[6] A second edition of the ISO Schematron was proposed in 2010 but has not been approved as of June 2012.

3.5 Versions of Related Technologies

The recommendations presented in this document are based on the use of Schematron schemas in conjunction with XPath 2.0 [11] and XSLT 2.0 [12] to enforce data constraints on XML instance documents, because these languages have more capable sets of features. Schematron schemas require the use of a query-binding language, such as XPath, to define the rules and checks. XPath 2.0 supports all of the simple primitive data types built into XML Schema. XPath 2.0 also supports *for* and *if-then-else* expressions that, among other functions, facilitate the use of algorithms in Schematron schemas. XSLT 2.0 is used because it is tightly integrated with XPath 2.0.

Although XSLT 2.0 and XPath 2.0 are the most capable query-binding languages, neither is required for use with Schematron; Schematron can use a variety of different query-binding languages. Section 6.4 of the ISO Schematron specification [8] provides more information on available query-binding languages.

4 RECOMMENDED USE OF SCHEMATRON

This document recommends the use of Schematron for schema validation as a complement to a grammar-based schema language. As described in the previous section, Schematron can express data constraints that cannot be easily expressed with a grammar-based schema language. Schematron's ability to express data constraints can be organized into two broad areas, described in this section:

- Data cardinality checking
- Co-constraint checking

4.1 Data Cardinality Checking

Data cardinality constraints limit the occurrence of data, elements, or attributes in an XML instance document. They can be applied over an entire XML instance document or to sections of a document. Schematron can easily be used to verify data cardinality constraints.

Like grammar-based languages, Schematron can validate that an element exists, that an element exists a specific number of times, or that an element has a value from an enumerated list; however, this document recommends the use of Schematron to verify data cardinality constraints that exist across an entire XML instance document or across multiple XML instance documents that are being released within the same package. Grammar-based languages do not have this capability. Schematron can require that a word must exist (or not exist) anywhere in an XML instance document or across multiple XML instance documents. It can perform existence checking, which is a type of data cardinality constraint, to verify that a value exists or does not exist in an XML instance document.

The cardinality checks discussed below are as follows:

- Specify that the existence of text is required.
- Specify that the existence of text is prohibited.

4.1.1 Specify That the Existence of Text Is Required

Schematron can specify that certain text is required to exist within the document; that is, a given character, word, or phrase must exist in the document (or in element A or in attribute B).

4.1.2 Specify That the Existence of Text Is Prohibited

Schematron can specify that certain text is prohibited from existing within the document; that is, a given character, word, or phrase must not exist in the document (or in element A or in attribute B).

There are generally two approaches for controlling what is permitted to cross domains. The White List approach checks an XML instance document for authorized data (data that can be permitted to cross security boundaries). If all of the authorized data is present, the document is considered valid and can cross security boundaries. The Black List approach checks an XML instance document for prohibited data (data that is not permitted to cross security boundaries). If any prohibited data is present, the document is considered invalid and cannot cross security boundaries unless all the prohibited data is removed.

For example, suppose a document is intended for release to the public. There may be a White List requirement that states, "The document must contain the name of an authorizing official. The document must contain an `Authorized` element containing the phrase 'This document has been authorized for release.'" Within the document, an `AuthorizingOfficial` element must exist with a string value indicating the name of the official who allowed the document to be released. The name must match that of one of the officials from a list of authorized officials. An `Authorized` element must also exist within the document. The phrase "This document has been authorized for release." must exist somewhere within that element.

Additionally, there may be a Black List requirement that states, "The document must not contain any information with a Top Secret classification. The document must not contain the phrase 'This document has not been authorized for release.'" Within the document, there must not be any elements containing a `CLASSIFICATION` attribute with a value of "TS". Also, the document must not contain the phrase "This document not has been authorized for release."

```
<Document>
  <AuthorizingOfficial>John Doe</AuthorizingOfficial>
  <Authorized>This document has been authorized for release.
  </Authorized>
  <Para CLASSIFICATION='U'>This paragraph contains information
about…</Para>
</Document>
```

To validate the document, developers must create two pattern elements for this Schematron schema. The White List pattern will contain all rules and assertions pertaining to authorized data that must be found in the XML instance document. The Black List pattern will contain all rules and assertions pertaining to prohibited data that must not be found in the XML instance document.

```
<sch:pattern id="White-List-Check"></sch:pattern>
<sch:pattern id="Black-List-Check"></sch:pattern>
```

To verify that all required items from the White List can be found within the document, a Schematron rule can be created with an assertion related to each required item. The context of the rule is set to be within the root element of the XML instance document (in this case, the `Document` element):

```
<sch:rule context="Document"></sch:rule>
```

Next, the rule must contain assertions about the required data, one for each required item. The first assertion should check for the presence of the `AuthorizingOfficial` element and the second assertion should step through each `Authorized` node in the document and check the contents of that node against the required phrase "This document has been authorized for release."

Each assertion element is required to have a test attribute. The test attribute is an XPath expression of the assertion test to be evaluated in the current context.

The first assertion would be stated as follows:

```
<sch:assert test="(AuthorizingOfficial castable as xs:string) and
                        (AuthorizingOfficial = 'John Smith') or
                        (AuthorizingOfficial = 'Jane Doe')">
     The document must contain an AuthorizingOfficial element
     containing the name of the official who allowed the document to
     be released. The following officials are authorized: John Smith
     and Jane Doe.
</sch:assert>
```

The XPath expression above reads as "An `AuthorizingOfficial` element whose value is of XML Schema type string and is John Smith or Jane Doe."

If this assertion test evaluates to false, the assertion statement will be output, informing the user that the document must contain the `AuthorizingOfficial` element with one of the two specified values.

The second assertion would be stated as follows:

```
<sch:assert test="count(//node()[contains(Authorized, 'This document
     has been authorized for release.')]) &gt;= 1">
     The Authorized element must contain the phrase: This document has
     been authorized for release.
</sch:assert>
```

The XPath expression above reads as "The count of the number of `Authorized` elements containing 'This document has been authorized for release.' is greater than or equal to one."

If this second assertion test evaluates to false, the assertion statement will be output, informing the user that the document must contain an `Authorized` element and that element must contain the required phrase. The `Authorized` element may contain other information, attributes, or elements, but somewhere within the element the phrase "This document has been authorized for release." must exist so that the assertion test evaluates to true.

To verify that none of the prohibited items from the Black List can be found within the document, a Schematron rule can be created with an assertion related to each prohibited item. The context of the rule is set to be within the root element of the XML instance document (in this case, the `Document` element):

```
<sch:rule context="Document"></sch:rule>
```

Next, the rule must contain two assertions about the prohibited data, one for each prohibited item. The first assertion should check for the presence of any element containing a `CLASSIFICATION` attribute with a value of "TS". The assertion would be stated as follows:

```
<sch:assert test="count(*[@CLASSIFICATION='TS']) = 0">
     The document must contain not any elements containing a
     classification attribute with a value of "TS".
</sch:assert>
```

The XPath expression above reads as "The count of the number of elements in the document whose `CLASSIFICATION` attribute has a value of 'TS' is equal to zero."

If this first assertion test evaluates to false, the assertion statement will be output, informing the user that the document must not contain any elements containing a `CLASSIFICATION` attribute with a value of "TS".

The second assertion should step through each node in the document and check the contents of that node against the prohibited phrase "This document has not been authorized for release." The assertion would be stated as follows:

```
<sch:assert test="count(//node()[contains(.,'This document has not
     been authorized for release.')]) = 0">
     The document must not contain the phrase: This document has not
     been authorized for release.
</sch:assert>
```

The XPath expression above reads as "The count of the number of nodes in the document containing 'This document has not been authorized for release.' is equal to zero."

If this second assertion test evaluates to false, the assertion statement will be output, informing the user that the document must not contain the prohibited phrase.

Below is the complete Schematron schema for this example:

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/Schematron">
  <sch:pattern id="White-List-Check">
    <sch:rule context="Document">
     <sch:assert test="(AuthorizingOfficial castable as xs:string) and
                       (AuthorizingOfficial = 'John Smith') or
                       (AuthorizingOfficial = 'Jane Doe') or
                       (AuthorizingOfficial = 'Bob Howdy')">
           The document must contain an AuthorizingOfficial element
           containing the name of the official who allowed the
           document to be released. The following officials are
           authorized: John Smith, Jane Doe, and Bob Howdy.
     </sch:assert>
     <sch:assert test="count(//node()[contains(Authorized,'This
         document has been authorized for release.')]) &gt;= 1">
           The Authorized element must contain the phrase: This
           document has been authorized for release.
     </sch:assert>
    </sch:rule>
  </sch:pattern>
  <sch:pattern id="Black-List-Check">
    <sch:rule context="Document">
     <sch:assert test="count(*[@CLASSIFICATION='TS']) = 0">
           The document must not contain any elements containing a
           classification attribute with a value of "TS".
     </sch:assert>
     <sch:assert test="count(//node()[contains(.,'This document has
          not been authorized for release.')]) = 0">
           The document must not contain the phrase: This document has
           not been authorized for release.
     </sch:assert>
    </sch:rule>
  </sch:pattern>
     </sch:schema>
```

4.2 Co-constraint Checking

Co-constraints are data relationships that must hold true. A co-constraint can apply to any number of dependencies among XML elements, attributes, and data values. A co-constraint may be enforced within a single XML instance document or across multiple XML instance documents that are being released in the same package. This document recommends that Schematron be used to compare data values and verify co-constraints.

The co-constraint checks discussed in this section are listed below and then described in detail:

- Specify that a value is required by the existence of an item.
- Specify that one value is required by another value.

4.2.1 Specify That a Value Is Required by the Existence of an Item

Schematron can specify that the value in one part of an XML instance document depends on the existence (or absence) of a node (element or attribute) elsewhere in the instance document. That is, if element A (or attribute B) exists, then element X (or attribute Y) must (or must not) have the value Z.

For example, imagine a mission-planning scenario. A planner creates a mission plan and needs to coordinate the plan with coalition partners in the United Kingdom (UK) and New Zealand (NZ). There may be a requirement that states, "All mission-planning information that is to be shared with coalition partners must be unclassified." Within the mission-planning document, if the `RELEASABILITY` element exists, then the `CLASSIFICATION` attribute on the root element `MISSION_PLAN` must be "U" for Unclassified.

```
<MISSION_PLAN CLASSIFICATION="U">
 <RELEASABILITY>
  <COALITION_PARTNERS>
   <COUNTRY>UK</COUNTRY>
   <COUNTRY>NZ</COUNTRY>
  </COALITION_PARTNERS>
 </RELEASABILITY>
 <MISSION_INFORMATION>
  <!-- Mission Information -->
 </MISSION_INFORMATION>
 <MISSION_DETAILS>
  <!-- Mission Details -->
 </MISSION_DETAILS>
</MISSION_PLAN>
```

A Schematron rule can enforce the requirement. The rule should step through the `MISSION_PLAN` in the document and check for the existence of the `RELEASABILITY` element. The context of the rule is set to be any `MISSION_PLAN` that contains a `RELEASABILITY` element:

```
<sch:rule context="MISSION_PLAN/RELEASABILITY"></sch:rule>
```

Next, the rule must contain an assertion about the data:

```
<sch:assert test="..[@CLASSIFICATION='U']">
      If the RELEASABILITY element exists, the Mission Plan must be
      labeled U (UNCLASSIFIED).
</sch:assert>
```

The XPath expression above reads as "The parent of the current node has a classification attribute with a value of 'U'."

If the assertion test evaluates to false, the assertion statement will be output, informing the user that the `CLASSIFICATION` element must have a value of “U”. Below is the complete Schematron schema for this example:

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:pattern id="Classification-Co-Constraint">
    <sch:rule context="MISSION_PLAN/RELEASABILITY">
      <sch:assert test="..[@CLASSIFICATION='U']">
           If the RELEASABILITY element exists, the Mission Plan must
           be labeled U (UNCLASSIFIED).
      </sch:assert>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

4.2.2 Specify That One Value Is Required by Another Value

Schematron can specify that a value in one part of an XML document requires a different part of the document to have a specified value. That is, if element A (or attribute B) has the value C, then element X (or attribute Y) must (or must not) have the value Z.

This specification is useful in a cross domain scenario to ensure that the classification value of data in a document will not be more sensitive than the classification value of the document. Assume there are only three possible classification values (from highest to lowest): TS (Top Secret), S (Secret), and C (Confidential). If a document is labeled at the Secret level, its contents must not contain data that is labeled at the Top Secret level. In general, for an XML instance document to be valid, no `Para` element may have a classification value higher than the classification value of `Document`.

```
<Document CLASSIFICATION='S'>
  <Para CLASSIFICATION='S'>
    <!-- Secret text here -->
  </Para>
  <Para CLASSIFICATION='C'>
    <!-- Confidential text here -->
  </Para>
  <Para CLASSIFICATION='TS'>
    <!-- Top Secret text here -->
  </Para>
</Document>
```

To check for the appropriate classification value, multiple Schematron rules are needed. The rules should step through the `Para` elements in the document, checking that the value of the classification attribute is not higher than the value of the classification attribute of the `Document`.

If a `Para` element has a classification value of "TS", then the `Document` must also have a classification value of "TS". The context of the first rule is set to be within any `Para` element with a classification value of "TS":

```
<sch:rule context="Para[@CLASSIFICATION='TS']"></sch:rule>
```

The XPath expression above reads as "A `Para` element whose classification value is 'TS'."

Next, the rule must contain an assertion about the data:

```
<sch:assert test="/Document/@ CLASSIFICATION='TS'">
 There is a Para labeled "TS", so the document must be labeled TS.
</sch:assert>
```

The XPath expression above reads as "The classification value of the `Document` element must be 'TS'."

If the assertion test evaluates to false, the assertion statement will be printed, informing the user that the document contains a `Para` element that has a higher classification value than the classification value of the document.

Two additional rules must be created to test within the context of `Para` elements with classification values of "S" and "C". Because all three rules are required for fully testing an XML instance document, these rules are wrapped by a single Schematron pattern element.

Below is the complete Schematron schema for this example:

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
   <sch:pattern id="Security-Classification-Check">
      <sch:rule context="Para[@CLASSIFICATION='TS']">
         <sch:assert test="/Document/@CLASSIFICATION='TS'">
             There is a Para labeled "TS", so the document
             must be labeled TS.
         </sch:assert>
      </sch:rule>
      <sch:rule context="Para[@CLASSIFICATION='S']">
         <sch:assert test="(/Document/@CLASSIFICATION='TS') or
                           (/Document/@CLASSIFICATION='S')">
             There is a Para labeled "S", so the document
             must be labeled either S or TS.
         </sch:assert>
      </sch:rule>
      <sch:rule context="Para[@CLASSIFICATION='C']">
         <sch:assert test="(/Document/@CLASSIFICATION='TS') or
                           (/Document/@CLASSIFICATION='S') or
                           (/Document/@CLASSIFICATION='C')">
             There is a Para labeled "C", so the document
             must be labeled either C, S, or TS.
         </sch:assert>
      </sch:rule>
   </sch:pattern>
</sch:schema>
```

4.2.3 Additional Co-constraint Checks

Currently, there are no recommendations for use of the following co-constraint checks to enforce cross domain security policies:

- Specify that one value must be related to another value.
- Specify that a certain total number of items is required.
- Specify that the existence of one item is required by the existence of another item.
- Specify that a value requires the existence of an item.

See Section 5.2.1 for a discussion of these additional co-constraint checks.

5 ADDITIONAL FEATURES

Schematron has some additional features that could be used in a CDS. This document does not specifically recommend the use of these additional features to enforce cross domain security policies. Instead, it recommends that CDS vendors should perform an analysis of alternatives to determine if it is more appropriate to use Schematron or another technology or programming language to implement the feature. This section divides these additional Schematron features into the following two categories:

- Features that *might be helpful* in a CDS
- Features that *could be used* in a CDS

5.1 Features That Might Be Helpful in a CDS

This section discusses additional Schematron features that might be helpful when used in a CDS. While this document does not specifically recommend the use of these additional features to enforce cross domain security policies, it does provide examples for how these helpful features might be implemented.

The following features might be helpful when used in a CDS:

- Presenting meaningful error messages
- Reading parameters from external files
- Cross-checking multi-part files

5.1.1 Presenting Meaningful Error Messages

Schematron has two types of error messages: assertion messages and diagnostic messages. The results of a validation are often used by those who are domain experts, not XML experts. Assertion messages can provide a CDS with more meaningful error messages because Schematron allows the schema author to write the error messages (as opposed to the messages being automatically generated by the Schematron processor); therefore, the errors can be reported in terms that are meaningful to operational users. The schema author can explain the error in an understandable way and provide directions for correcting the problem using terminology appropriate to the domain.

Diagnostic messages provide specific details about validation failure. These details can be used to determine exactly where and why each problem occurred. Both types of error messages can be recorded in a CDS log file if desired, helping the guard security administrators investigate what happened.

If a Schematron assertion fails, the system displays the appropriate assertion message to the user. The assertion message is a general statement about what should be found in the schema. For example:

- A unit should not be tasked beyond its available assets.
- The number of mission detail elements must equal the number of aircraft in the mission report.

Assertion messages can be composed of mixed content: text and `value-of` elements. A Schematron `value-of` element has the same functionality as the XSLT `value-of` element. It has a `select` attribute that is an XPath expression used to identify a node in the document. The Schematron tool indicates the value of the node that is identified.

Assertions can be connected to a diagnostic element, providing additional details about why an XML instance document failed an assertion. For example:

- These tasking orders do not meet the mission criteria. The unit asset designated "X" has "Y" available assets, but the mission details require "Z" assets.
- This mission report does not meet the mission criteria. Mission number "X" has "Y' aircraft, but there are mission details for "Z" aircraft. Each aircraft must submit details of the mission.

Diagnostic messages are also composed of mixed content: text and `value-of` elements. The `assert` element has an optional `diagnostics` attribute. The value of the `diagnostics` attribute is the value of a `diagnostic` element's `id` attribute. During validation, if the assertion fails, then the appropriate diagnostic will be executed; thus, the assertion message and the diagnostic message will both be executed, giving the user both pieces of information. An assertion element can contain multiple diagnostics.

Using the example shown in Section 5.2.1.2, a diagnostic message could be added to the schema to provide the user with a more specific message stating why the assertion failed. Instead of stating that the number of mission detail elements must equal the number of aircraft in the mission report, the `value-of` element can be used to select and output the actual values that are causing the failed assertion:

```
<sch:diagnostics>
      <sch:diagnostic id="invalid-mission-details">
       This mission report does not meet the mission criteria.
       Mission number <sch:value-of select="MISSION_NUMBER"/> has
       <sch:value-of select="$number_of_aircraft"/> aircraft, but
       there are mission details for <sch:value-of
       select="$number_of_mission_details"/> aircraft.  Each aircraft
       must submit details of the mission.
      </sch:diagnostic>
</sch:diagnostics>
```

To make this diagnostic message output on a failed assertion, the message must be connected to the assertion using the `assert` element's `diagnostics` attribute. The `diagnostics` attribute's value should equal the value of the `diagnostic` element's `id` attribute:

```
<sch:assert test="$number_of_aircraft = $number_of_mission_details"
            diagnostics="invalid-mission-details">
```

Below is the complete Schematron schema for this example:

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:pattern id="Number-Of-Aircraft-Co-Constraint">
    <sch:rule context="MISSION_REPORT">
      <sch:let name="number_of_aircraft"
               value="MISSION_INFORMATION/NUMBER_OF_AIRCRAFT" />
      <sch:let name="number_of_mission_details"
               value="count(MISSION_DETAILS)" />
      <sch:assert test="$number_of_aircraft =
          $number_of_mission_details"
          diagnostics="invalid-mission-details">
           The number of mission detail elements must equal the number
           of aircraft in the mission report.
      </sch:assert>
    </sch:rule>
  </sch:pattern>
  <sch:diagnostics>
      <sch:diagnostic id="invalid-mission-details">
       This mission report does not meet the mission criteria.
       Mission number <sch:value-of select="MISSION_NUMBER"/> has
       <sch:value-of select="$number_of_aircraft"/> aircraft, but
       there are mission details for <sch:value-of
       select="$number_of_mission_details"/> aircraft. Each aircraft
       must submit details of the mission.
      </sch:diagnostic>
  </sch:diagnostics>
</sch:schema>
```

5.1.2 Reading Parameters from External Files

Schematron can read information from external files. This capability might be used to maintain a list of restricted words within the CDS for reserved word detection.[7] As files are moved from one domain to another, it is desirable to see if they contain certain restricted words. If they do contain certain restricted words, they can either be stripped of these words prior to crossing domains or they can simply not be allowed to cross domains.

Different classifications often require different sets of data constraints. For example, a reserved word list for a Secret document may be unlike a reserved word list for a Top Secret document, because Top Secret documents are expected to contain more sensitive material.

[7] The Cross Domain community often refers to a reserved word as a "dirty word."

These differences could result in the creation of many Schematron schemas whose only difference is in the hard-coded reserved words.[8] Schematron schemas can dynamically read parameters, such as reserved word lists, from an external file. This ability allows users to reuse the same Schematron schema for all classification values by changing the file containing the appropriate word list.

For example, for an XML instance document to be valid, it must not contain the restricted words obtained from the `reserved-words-list.xml` file:

```
<Reserved-Words>
  <reserved-word>SENSITIVE</reserved-word>
  <reserved-word>NOFORN</reserved-word>
  <reserved-word>DESCLASSIFY</reserved-word>
</Reserved-Words>
```

A Schematron pattern must be created to check for the presence of a restricted word. First, the `let` element is used to declare a variable named `reserved-words`. The value of this variable is a document node from the `reserved-words-list.xml` file. The `reserved-words-list.xml` file is obtained using the XPath `document()` function. The XPath expression selects all of the `reserved-word` elements in the `reserved-words-list.xml` file:

```
<sch:let name="reserved-words" value="document('reserved-words-
list.xml')/Reserved-Words/reserved-word"/>
```

Next, a Schematron rule can be created to step through each node in the document and check the contents of that node against the restricted word. The context of the rule is set to be the root element of the XML instance document (in this case, the `Document` element):

```
<sch:rule context="Document"></sch:rule>
```

Next, the rule must contain an assertion about the data:

```
<sch:assert test="sum(for $i in $reserved-words
                  return count(//node()[contains(., $i)])) = 0">
     The document must not contain a reserved word.
</sch:assert>
```

Many different XPath expressions can be created to test if a node contains specific text. The XPath expression above reads as "The sum of the count of the number of nodes in the current context that contain a reserved word is equal to zero."

If the assertion test evaluates to false, the assertion statement will be output, informing the user that the document must not contain the restricted word.

[8] "Hard-coding" refers to the practice of embedding data directly into source code instead of obtaining the data from external sources.

Below is the complete Schematron schema for this example:

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:pattern id="Reserved-Word-Detection">
    <sch:let name="reserved-words" value="document('reserved-words-
list.xml')/Reserved-Words/reserved-word"/>
    <sch:rule context="Document">
      <sch:assert test="sum(for $i in $reserved-words return
count(//node()[contains(., $i)])) = 0">
          The document must not contain a reserved word.
      </sch:assert>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

5.1.3 Cross-checking Multi-part Files

Schematron has the ability to check data across multiple XML instance documents. For example, a co-constraint may occur within an XML document or across multiple XML documents. Using the `document()` XPath function, a schema can obtain a reference to an external source document and check data from multiple XML instance documents. For example, a CDS may need to process a multi-part payload of related files that are required to cross security domains as a package.

5.2 Features That Could Be Used in a CDS

This section discusses additional Schematron features whose added value for a CDS has yet to be determined. While this document does not recommend the use of these additional features to enforce cross domain security policies, it does provide examples showing how these features might be implemented.

The following features could be used in a CDS:

- Additional co-constraint checking
- Algorithmic checking
- Data type checking using Schematron and XPath 2.0

5.2.1 Additional Co-constraint Checking

Section 4.2 discussed co-constraints that could be used in a cross domain scenario to enforce security policies. The following co-constraint checks could also be used in a CDS:

- Specify that one value must be related to another value.
- Specify that a specific number of items is required.
- Specify that the existence of one item is required by the existence of another item.
- Specify that a value requires the existence of an item.

5.2.1.1 Specify That One Value Must Be Related to Another Value

Schematron can specify that a value in one part of an XML document must have a specified relationship to a value in a different part of the document; that is, if element A (or attribute B) exists, then element X (or attribute Y) must (or must not) have a related value.

For example, suppose a pilot is submitting a mission report. There may be a requirement that states, "The time of arrival at the arrival location must be after the time of departure from the departure location." Within the mission report document, if there is an `ARRIVAL_TIME`, it must be after the `DEPARTURE_TIME`:

```
<MISSION_REPORT>
 <MISSION_DETAILS>
  <DEPARTURE_LOCATION>KLFI</DEPARTURE_LOCATION>
  <DEPARTURE_TIME>2008-03-25T11:45Z</DEPARTURE_TIME>
  <ARRIVAL_LOCATION>KLFI</ARRIVAL_LOCATION>
  <ARRIVAL_TIME>2008-03-25T13:15Z</ARRIVAL_TIME>
 </MISSION_DETAILS>
</MISSION_REPORT>
```

A Schematron rule can be created to check that the arrival time is after the departure time. The rule should step through each `MISSION_DETAILS` element in the document and compare the contents of the `ARRIVAL_TIME` and `DEPARTURE_TIME` elements. The context of the rule is set to be the `MISSION_DETAILS` of the `MISSION_REPORT`:

```
<sch:rule context="MISSION_REPORT/MISSION_DETAILS"></sch:rule>
```

Next, the rule must contain an assertion about the data:

```
<sch:assert test="(string(ARRIVAL_TIME) cast as xs:dateTime) &gt;
                  (string(DEPARTURE_TIME) cast as xs:dateTime)">
     Arrival time must be after departure time.
</sch:assert>
```

The XPath expression above takes the string values of the elements and casts them as XML Schema `dateTime` data types before comparing the values using the "greater than" operator `>`. The expression reads as "The arrival `dateTime` is greater than the departure `dateTime`."

If the assertion test evaluates to false, the assertion statement will be output, informing the user that the arrival time is not after the departure time.

Below is the complete Schematron schema for this example:

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:pattern id="Mission-Time-Co-Constraint">
    <sch:rule context="MISSION_REPORT/MISSION_DETAILS">
      <sch:assert test="(string(ARRIVAL_TIME) cast as xs:dateTime) &gt;
                        (string(DEPARTURE_TIME) cast as xs:dateTime)">
        Arrival time must be after departure time.
       </sch:assert>
    </sch:rule>
  </sch:pattern>
     </sch:schema>
```

5.2.1.2 Specify That a Total Number of Items Is Required

Schematron can specify that a specific number of related parts of an XML document must be required to exist; that is, the total number of a set of elements (or a set of attributes) must (or must not) equal a specific number (or the value of element A or the total number of another set of elements [or a set of attributes]).

For example, suppose a pilot is submitting a mission report for a multi-aircraft mission. There may be a requirement that states, "Every aircraft pilot must submit the details of his/her mission." Within the mission report document, the number of `MISSION_DETAILS` elements must equal the value of the `NUMBER_OF_AIRCRAFT` element:

```
<MISSION_REPORT>
 <MISSION_INFORMATION>
  <MISSION_NUMBER>LM789</MISSION_NUMBER>
  <MISSION_TYPE>ATTACK</MISSION_TYPE>
  <NUMBER_OF_AIRCRAFT>2</NUMBER_OF_AIRCRAFT>
 </MISSION_INFORMATION>
 <MISSION_DETAILS>
  <DEPARTURE_LOCATION>KLFI</DEPARTURE_LOCATION>
  <DEPARTURE_TIME>2008-03-25T11:45Z</DEPARTURE_TIME>
  <ARRIVAL_LOCATION>KLFI</ARRIVAL_LOCATION>
  <ARRIVAL_TIME>2008-03-25T13:15Z</ARRIVAL_TIME>
  <MISSION_RESULTS>DESTROYED</MISSION_RESULTS>
  <SIGHTINGS>Column of tanks heading north on Hwy 80</SIGHTINGS>
 </MISSION_DETAILS>
 <MISSION_DETAILS>
  <DEPARTURE_LOCATION>KLFI</DEPARTURE_LOCATION>
  <DEPARTURE_TIME>2008-03-25T11:46Z</DEPARTURE_TIME>
  <ARRIVAL_LOCATION>KLFI</ARRIVAL_LOCATION>
  <ARRIVAL_TIME>2008-03-25T13:16Z</ARRIVAL_TIME>
  <MISSION_RESULTS>MISSED</MISSION_RESULTS>
  <SURFACE_TO_AIR_ACTIVITY> SAM fired north of
  Freeville</SURFACE_TO_AIR_ACTIVITY>
 </MISSION_DETAILS>
</MISSION_REPORT>
```

A Schematron rule can be created to check that the number of MISSION_DETAILS elements is equal to the value of the NUMBER_OF_AIRCRAFT element. The rule should step through the MISSION_REPORT and check that the number of MISSION_DETAILS elements is equal to the value of the NUMBER_OF_AIRCRAFT element. The context of the rule is set to be the MISSION_REPORT:

```
<sch:rule context="MISSION_REPORT"></sch:rule>
```

Next, the let element is used to declare two variables within the context of the MISSION_REPORT. The let element has two attributes: name and value. The name attribute is the name of the variable and the value attribute is an XPath expression that selects a value in the current context. The first variable is named number_of_aircraft and is the value of the NUMBER_OF_AIRCRAFT element found in the MISSION_INFORMATION. The second variable is named number_of_mission_details and is the value of the number of MISSION_DETAILS elements found in the MISSION_REPORT:

```
<sch:let name="number_of_aircraft"
                value="MISSION_INFORMATION/NUMBER_OF_AIRCRAFT"/>
<sch:let name="number_of_mission_details"
                value="count(MISSION_DETAILS)"/>
```

Next, the rule must contain an assertion about the data:

```
<sch:assert test="$number_of_aircraft = $number_of_mission_details">
      The number of mission detail elements must equal the number of
      aircraft in the mission report.
</sch:assert>
```

The XPath expression uses the $ operator to access the variables that were declared above. The expression reads as “The value of the $number_of_aircraft variable is equal to the value of the $number_of_mission_details variable.”

If the assertion test evaluates to false, the assertion statement will be output, informing the user that the number of mission detail elements must equal the number of aircraft in the mission report.

Below is the complete Schematron schema for this example:

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:pattern id="Number-Of-Aircraft-Co-Constraint">
    <sch:rule context="MISSION_REPORT">
      <sch:let name="number_of_aircraft"
               value="MISSION_INFORMATION/NUMBER_OF_AIRCRAFT" />
      <sch:let name="number_of_mission_details"
               value="count(MISSION_DETAILS)" />
      <sch:assert test="$number_of_aircraft =
          $number_of_mission_details">
           The number of mission detail elements must equal the number
           of aircraft in the mission report.
      </sch:assert>
    </sch:rule>
  </sch:pattern>
     </sch:schema>
```

5.2.1.3 Specify That the Existence of One Item Is Required by the Existence of Another Item

Schematron can specify that the existence of one part of an XML document requires a different part of the document to exist also; that is, if element A (or attribute B) exists, then element X (or attribute Y) must (or must not) exist.

For example, suppose a pilot is submitting a mission report. There may be a requirement that states, “The mission that was flown was either part of an operation or an exercise, but not both.” Within the mission report document, if the `OPERATION` element exists, then the `EXERCISE` element must not exist; if the `EXERCISE` element exists, then the `OPERATION` element must not exist:

```
<MISSION_REPORT>
 <EXERCISE>
  <EXERCISE_NAME>Swift Kick</EXERCISE_NAME>
  <EXERCISE_IDENTIFIER>SK2008</EXERCISE_IDENTIFIER>
 </EXERCISE>
</MISSION_REPORT>
```

A Schematron rule can be created to check that both an OPERATION and EXERCISE element do not exist. The rule should step through the MISSION_REPORT searching for the EXERCISE and OPERATION elements. The context of the rule is set to be the MISSION_REPORT element:

```
<sch:rule context="MISSION_REPORT"></sch:rule>
```

Next, the rule must contain an assertion about the data:

```
<sch:assert test="(count(/MISSION_REPORT/OPERATION) = 1 and
                   count(/MISSION_REPORT/EXERCISE) = 0) or
                  (count(/MISSION_REPORT/OPERATION) = 0 and
                   count(/MISSION_REPORT/EXERCISE) = 1)">
     The document must contain either an EXERCISE element or an
     OPERATION element, but not both.
</sch:assert>
```

The XPath expression above reads as "The count of the number of OPERATION elements in the MISSION_REPORT is one and the count of the number of EXERCISE elements in the MISSION_REPORT is zero or the count of the number of OPERATION elements in the MISSION_REPORT is zero and the count of the number of EXERCISE elements in the MISSION_REPORT is one."

If the assertion test evaluates to false, the assertion statement will be output, informing the user that the document must contain either an EXERCISE element or an OPERATION element, but not both.

Below is the complete Schematron schema for this example:

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
<sch:pattern id="Exercise-Operation-Co-Constraint">
    <sch:rule context="MISSION_REPORT">
      <sch:assert test="(count(/MISSION_REPORT/OPERATION) = 1 and
                         count(/MISSION_REPORT/EXERCISE) = 0) or
                        (count(/MISSION_REPORT/OPERATION) = 0 and
                         count(/MISSION_REPORT/EXERCISE) = 1)">
           The document must contain either an EXERCISE element or an
           OPERATION element, but not both.
      </sch:assert>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

5.2.1.4 Specify That the Existence of an Item Is Required by a Value

Schematron can specify that a value in one part of an XML document can require a different part of the document to exist also; that is, if element A (or attribute B) has the value C, then element X (or attribute Y) must (or must not) exist.

For example, imagine a pilot submitting a mission report. There may be a requirement that states, "If the mission was not aborted, there cannot be a reason why the mission was aborted." Within the mission report document, if `MISSION_ABORT` = "No", then `REASON_MISSION_ABORTED` must not exist.

```
<MISSION_REPORT>
 <MISSION_INFORMATION>
  <MISSION_NUMBER>BC123</MISSION_NUMBER>
  <MISSION_TYPE>SEAD</MISSION_TYPE>
  <MISSION_ABORT>No</MISSION_ ABORT>
 </MISSION_INFORMATION>
</MISSION_REPORT>
```

A Schematron rule can be created to check whether or not the mission was aborted. The rule should step through each `MISSION_INFORMATION` element in the `MISSION_REPORT` document and check that the value of the `MISSION_ABORT` element is equal to "No". The context of the rule is set to be a `MISSION_REPORT`, where the `MISSION_ABORT` element is equal to "No":

```
<sch:rule
     context=MISSION_REPORT[MISSION_INFORMATION/MISSION_ABORT='No']">
</sch:rule>
```

Next, the rule must contain an assertion about the data:

```
<sch:assert test="count(//REASON_MISSION_ABORTED) = 0">
     If the mission was not aborted, there cannot be a reason why the
     mission was aborted.
</sch:assert>
```

The XPath expression above reads as "The count of the number of `REASON_MISSION_ABORTED` elements in the `MISSION_REPORT` is equal to zero."

If the assertion test evaluates to false, the assertion statement will be output, informing the user that if the mission was not aborted, then there should be no reason why the mission was aborted.

Below is the complete Schematron schema for this example:

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:pattern id="Mission-Aborted-Co-Constraint">
      <sch:rule context =
        "MISSION_REPORT[MISSION_INFORMATION/MISSION_ABORT='No']">
         <sch:assert test="count(//REASON_MISSION_ABORTED) = 0">
                 If the mission was not aborted, there cannot be a
                 reason why the mission was aborted.
         </sch:assert>
      </sch:rule>
  </sch:pattern>
</sch:schema>
```

5.2.2 Algorithmic Checking

Algorithmic checking can be used to ensure that the values in an XML instance document are consistent and reasonable. This document does not recommend that Schematron should or should not be used for algorithmic checking. Instead, this document recommends that CDS vendors should perform an analysis of alternatives to determine if it is more appropriate to use Schematron (or another technology or programming language) and then implement it accordingly.

Algorithmic checking validates an XML instance document by applying an algorithm to the data. Schematron can enforce algorithmic checks by performing computations on data in an XML instance document. The computations use XPath 2.0; although XPath 2.0 is not a general-purpose programming language (such as Java or C#), it has many of the same features.

XPath 2.0 expressions select portions of the XML instance document to create a sequence. XPath 2.0 includes arithmetic expressions, comparison expressions, logical expressions, an iteration expression, and a conditional expression. These expressions can be used together to select the desired data from the XML document for computation. XPath 2.0 functions process sequences and the data in them. XPath 2.0 includes functions for numeric values, strings, dates and times, and entire sequences. These functions can be used together to process the data selected from the XML document.

XPath 2.0 expressions and functions can be combined to perform the computations necessary to enforce an algorithmic check. The range of possible algorithmic checks is essentially limitless.

Schematron tends to be well suited for simple computations. For example, if an XML instance document details the sales numbers of a company by region, then those numbers should sum to 100% of the total company sales. Or if an XML instance document records the grades for a college student by class, then the combined grade point average of those classes should not exceed 4.0. Or if the sum of the prices of the items on an invoice exceeds $200, then shipping would be free and therefore the cost of shipping should be $0.

To use an operational example, if a tasking order details the assets available for tasking and their assigned tasks, then a unit should not be tasked beyond its available assets:

```
<TASKING_ORDERS>
 <THEATER_RESOURCES>
  <UNIT_ASSETS>
   <UNIT_DESIGNATOR>123Sqdn</UNIT_DESIGNATOR>
   <NUMBER_OF_ASSETS>6</NUMBER_OF_ASSETS>
  </UNIT_ASSETS>
  <UNIT_ASSETS>
   <UNIT_DESIGNATOR>124Sqdn</UNIT_DESIGNATOR>
   <NUMBER_OF_ASSETS>4</NUMBER_OF_ASSETS>
  </UNIT_ASSETS>
 </THEATER_RESOURCES>
 <AIR_MISSIONS>
  <AIR_MISSION>
   <MISSION_INFORMATION>
    <MISSION_NUMBER>AQ456</MISSION_NUMBER>
    <UNIT_DESIGNATOR>123Sqdn</UNIT_DESIGNATOR>
    <MISSION_TYPE>COUNTERAIR</MISSION_TYPE>
   </MISSION_INFORMATION>
   <MISSION_DETAILS>
    <AIRCRAFT_CALLSIGN>LEOPARD14</AIRCRAFT_CALLSIGN>
    <AIRCRAFT_TYPE_MODEL>F16B</AIRCRAFT_TYPE_MODEL>
   </MISSION_DETAILS>
   <MISSION_DETAILS>
    <AIRCRAFT_CALLSIGN>LEOPARD15</AIRCRAFT_CALLSIGN>
    <AIRCRAFT_TYPE_MODEL>F16B</AIRCRAFT_TYPE_MODEL>
   </MISSION_DETAILS>
   <MISSION_DETAILS>
    <AIRCRAFT_CALLSIGN>LEOPARD16</AIRCRAFT_CALLSIGN>
    <AIRCRAFT_TYPE_MODEL>F16B</AIRCRAFT_TYPE_MODEL>
   </MISSION_DETAILS>
  </AIR_MISSION>
 </AIR_MISSIONS>
</TASKING_ORDERS>
```

A Schematron rule can be created to check that a unit is not tasked beyond its available assets. The rule should step through each of the `UNIT_ASSETS` in the `TASKING_ORDERS` and check that the `NUMBER_OF_ASSETS` element is greater than or equal to the count of the `MISSION_DETAILS` elements in the `AIR_MISSION` whose `UNIT_DESIGNATOR` has the same value as that of the `UNIT_ASSETS` element's `UNIT_DESIGNATOR`. The context of the rule is set to be the `UNIT_ASSETS` within the `TASKING_ORDERS`:

```
<sch:rule context="TASKING_ORDERS/THEATER_RESOURCES/UNIT_ASSETS">
</sch:rule>
```

Next, the `let` element is used to declare two variables within the context of the `UNIT_ASSETS`. The first variable is named `number_of_assets` and is the value of the `NUMBER_OF_ASSETS` element. The second variable is named `unit_designator` and is the value of the `UNIT_DESIGNATOR` element:

```
<sch:let name="number_of_assets" value="NUMBER_OF_ASSETS" />
<sch:let name="unit_designator" value="UNIT_DESIGNATOR" />
```

Next, the rule must contain an assertion about the data:

```
<sch:assert test = "$number_of_assets &gt;=
count(//AIR_MISSIONS/AIR_MISSION[MISSION_INFORMATION/UNIT_DESIGNATOR=$
unit_designator]/MISSION_DETAILS)">
     A unit should not be tasked beyond its available assets.
</sch:assert>
```

The XPath expression uses the `$` operator to access the variables that were declared above. The expression reads as "The value of the `$number_of_assets` variable is greater than or equal to the count of the number of `MISSION_DETAILS` from the designated unit."

If the assertion test evaluates to false, the assertion statement will be output, informing the user that the unit should not be tasked beyond its available assets.

Below is the complete Schematron schema for this example:

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:pattern id="Algorithmic-Check">
    <sch:rule context="TASKING_ORDERS/THEATER_RESOURCES/UNIT_ASSETS">
      <sch:let name="number_of_assets" value="NUMBER_OF_ASSETS" />
      <sch:let name="unit_designator" value="UNIT_DESIGNATOR" />
      <sch:assert test="$number_of_assets &gt;=
      count(//AIR_MISSIONS/AIR_MISSION[MISSION_INFORMATION/UNIT_DESIGNA
      TOR=$unit_designator]/MISSION_DETAILS)">
            A unit should not be tasked beyond its available assets.
       </sch:assert>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

5.2.3 Data Type Checking Using Schematron and XPath 2.0

If validation is performed using only the Schematron schema language, it is important to note that Schematron can be used for expressing grammar constraints, such as checking data types. Schematron is capable of expressing certain grammar constraints commonly found in schema languages such as XML Schema and RELAX NG; however, this document recommends that grammar-based schema languages be used to perform data type checking in tandem with Schematron assertions

Used in conjunction with XPath 2.0, Schematron schemas can easily implement data type checks. XPath 2.0 supports the use of XML Schema primitive types; therefore, Schematron schemas can test XML instance document data against more than nineteen simple data types. The following sections contain examples of data type checking using Schematron and XPath 2.0.

5.2.3.1 Limiting Values of a Numerical Data Type

The following Schematron assertion test uses the `castable` keyword in an XPath expression to verify that a value is of XML Schema type integer and is greater than zero and less than one hundred:

```
<sch:assert test="(. castable as xs:integer) and
                  (. &gt;= 0) and
                    (. &lt;= 100)">
      The value must be an integer greater than 0 and less than 100.
      </sch:assert>
```

5.2.3.2 Constraining Content and Length of a String

The following Schematron assertion test uses the `castable` keyword in an XPath expression to verify that a value is of XML Schema type string, has a length no greater than one hundred characters, and only contains certain characters:

```
<sch:assert test="(. castable as xs:string) and
                   (string-length() &lt;= 100) and
                   (matches(.,'[\sa-zA-Z0-9,;:\.]*'))">
      The value is a string, with a length no greater than 100
      characters, composed of only these characters: a-z, A-Z, 0-9,
      comma, period, colon, semicolon.
</sch:assert>
```

5.2.3.3 Enumerated Values

The following Schematron assertion test uses an XPath expression to verify that a value is constrained to be one of three values:

```
<sch:assert test="((. = 'UNCLASSIFIED') or
                    (. = 'CONFIDENTIAL') or
                    (. = 'SECRET'))">
      The value can be one of UNCLASSIFIED, CONFIDENTIAL, or SECRET.
</sch:assert>
```

6 IMPLEMENTATION ISSUES/DECISIONS

6.1 Approval of XML Tools and Data Validation Schemas

Many commercial and open source XML tools are not specifically designed to provide the security features that the DoD and IC need in a CDS. While these tools can provide added functionality to assist in data validation, any CDS that incorporates them will have to be thoroughly tested prior to approval for operational use.

Any XML Schemas, including Schematron schemas, used for data validation must also be approved for use. The cross domain community is just beginning to expand its current approval process to include approval of XML Schemas.

6.2 Schematron Processor versus XSLT Processor

Schematron validation can be implemented directly with a Schematron processor or indirectly using an XSLT processor. The advantage of using an XSLT processor is reuse. If a CDS already contains an approved XSLT processor, there is no need to add a Schematron processor to the CDS; however, an XSLT stylesheet, called a "skeleton stylesheet," would have to be approved for use on the CDS. The skeleton stylesheet is an XSLT script that provides all of the basic parsing and validating routines for compiling a Schematron schema into XSLT.[9]

If a CDS uses an XSLT processor implementation, the validation process occurs in two stages. In the first stage, the Schematron schema and ISO Schematron skeleton stylesheet are compiled into an XSLT stylesheet. This stage should be completed statically in a controlled environment. The resulting stylesheet is installed on the CDS so that, in the second stage, an XML instance document can be validated against the stylesheet using an XSLT processor. Since only the second stage of processing occurs on the guard, this option maximizes performance.

Schematron Validation Report Language (SVRL) is a simple reporting language that is defined as part of ISO Schematron.[10] When Schematron is executed it has the ability to generate output in SVRL format. This document recommends that Schematron validation fail if the generated SVRL output contains either a `<failed-assert> or <successful-report>` element.

6.3 Embedded versus Separate Schemas

If a CDS uses both Schematron and a grammar-based language, such as XML Schema, CDS vendors will need to choose how to relate the two schemas. Schematron rules can be embedded in a grammar-based schema file or kept in a separate file.

Co-location is one benefit of embedding the Schematron schema in the grammar-based schema, because all the rules and constraints for a given portion of an XML document can be found in one place. This co-location would be especially helpful if schema developers used an automatic schema generation tool that supported generation of both Schematron and grammar-based

[9] The stable release (as of 2010-01-25) implementation of ISO Schematron using XSLT is available at http://www.schematron.com/index html.
[10] http://www.schematron.com/validators html

schema languages. XML Schema tools created by various companies, such as oXygen[11] and Topologi,[12] support validation of schemas with embedded Schematron rules.

Separation of concerns, however, is a good reason to keep the schemas separate. If they are separate, the first benefit is the potential for increased performance. If the CDS is a multiprocessor device, the two types of validation can be executed in parallel. If the CDS is a pipelined device and the validation is conducted serially, the vendors can optimize performance by selecting which validation to perform first. Perhaps the fastest one should be done first; perhaps the most likely to fail should be done first. In either case, if the first validation fails, that would eliminate the need to execute the second.

An XSLT processor can be used to validate schemas that have embedded Schematron rules. During the validation process, the XSLT processor uses a skeleton stylesheet to extract the embedded Schematron rules from the host schema and creates a complete Schematron schema. As with separate schemas, if the CDS is a multiprocessor device, the two types of validation can be executed in parallel and the validation reports can be merged to form a single report; however, the overhead of processing the embedded Schematron rules may be unacceptable in time-critical applications.

A second benefit of separate schemas is decreased complexity of the CDS. When each step in the process is distinct, it can be developed and approved on its own, thus making it easier to gain approval.

Finally, using separate schemas is a more flexible design. It is easier to replace one processor with another or one language with another (e.g., switch from XML Schema to RELAX NG or from Schematron to CLiXML), should it be advantageous to do so in the future. This flexible design allows different tools and authors to create the schemas. Therefore, the XML Schema expert does not have to be a Schematron expert and vice versa.

[11] http://www.oxygenxml.com/.
[12] http://topologi.com/index.php.

7 CONCLUSIONS AND RECOMMENDATIONS

This document presents several approaches for using the Schematron schema language to enforce cross domain security policies, particularly those policies that cannot be enforced by a grammar-based schema language alone. Schematron can be used to supplement the validation capabilities of a grammar-based schema language and thus increase the total validation capabilities of CDSs and reduce even further the risk of transferring malicious or unauthorized XML data between security domains. This document describes some of the data constraints that Schematron can easily express, notably data cardinality and co-constraints, and some of the additional features that Schematron provides.

This document provides the following recommendations:

- Use Schematron to verify data cardinality constraints that exist across an entire XML instance document or multiple XML instance documents.
- Use Schematron to compare data values and verify co-constraints only when needed to enforce a cross domain policy.
- Request CDS vendors to perform an analysis of alternatives to determine if it is more appropriate to use Schematron or another technology or programming language for algorithmic checking and then implement the chosen alternative accordingly.

This page intentionally left blank.

APPENDIX A: INTRODUCTION TO SCHEMATRON

A.1 WHAT IS SCHEMATRON?

Schematron is a rule-based schema language used for making assertions about patterns found in Extensible Markup Language (XML) documents. A Schematron schema can contain business and operational rules that constrain the format and contents of XML data. The schema can be used to determine the validity of XML instance documents. Assertions are expressed using XPath and they can contain any function supported by an XPath statement or XSLT test condition. Schematron is well suited for verifying data constraints that cannot be easily expressed using grammar-based schema languages, such as data cardinality, co-constraints, and algorithmic checks.

A.1.1 Rule-Based versus Grammar-Based Schema Languages

There are two types of schema languages: grammar-based languages and rule-based languages. Grammar-based schema languages, such as XML Schema and RELAX NG, specify the structure, form, and syntax of elements and attributes in an XML instance document. Grammar-based schema languages may be used to constrain the contents and data type of each element or attribute in an XML instance document.

Unlike grammar-based XML Schema languages, Schematron is a rule-based (or assertion-based) schema language that specifies the relationships that must hold between the elements and attributes in an XML instance document. A rule-based schema language may be used to verify data constraints within an XML instance document.

Schematron specifies assertions about virtually any arbitrary pattern in an XML instance document, instead of merely enforcing a grammar on the document. For this reason, Schematron can easily enforce constraints that would be problematic to enforce using grammar-based schema languages alone.

A.1.2 Other Schematron Benefits

Schematron has additional benefits compared to current grammar-based languages because it uses XPath expressions and can output detailed error messages. Schematron specifies which relationships and patterns should hold true in the data by making assertions using XPath expressions. It uses XPath expressions to locate and evaluate data in specific context paths within a parsed XML instance document. Schematron designers can create detailed diagnostic messages using plain language that can be displayed when an assertion fails as well as when an assertion is satisfied. For example, if an XML instance document fails to meet an assertion, then the diagnostic message supplied by the author of the Schematron schema is displayed.

A.2 CORE ELEMENTS

The Schematron language is an ISO standard and consists of fourteen core elements: `active`, `assert`, `extends`, `include`, `let`, `name`, `ns`, `param`, `pattern`, `phase`, `report`, `rule`, `schema`, and `value-of` [8]. The following sections describe each element.

A.2.1 active

The active element is used within a phase element to specify which patterns are active in that phase. The required pattern attribute is a reference to a pattern that is active in the current phase.

A.2.2 assert

The assert element specifies a relationship that must exist in the XML instance document. The element has a required attribute, test, whose value is an XPath expression of the assertion test to be evaluated in the current context. The value of the assert element can contain plain text that will be displayed if the relationship expressed in the assertion fails. This value can be used to create detailed diagnostic or error messages.

The element has seven optional attributes that can be used to create rich interfaces and documentation, promote more detailed outcomes, and allow identification of some part of a pattern. The attributes are diagnostics, icon, see, fpi, flag, role, and subject.

A.2.3 extends

The extends element enables the reuse of assertions through abstract rules. The element has a required attribute, rule, which is used to reference the abstract rule. The current rule uses all of the assertions from the abstract rule it extends.

A.2.4 include

The include element is used to include external documents in the Schematron schema. The element has a required attribute, href, which references an external, well-formed XML document. The external document is inserted in place of the include element.

A.2.5 let

The let element is used to declare a named variable. The element has a required attribute, name, which is the name of the variable, and a required attribute, value, which is an XPath expression that is evaluated in the current context.

If the let element is the child of a rule element, the variable is calculated and scoped to the current rule and context. Otherwise, the variable is calculated with the context of the XML instance document's root.

A.2.6 name

The name element provides the names of nodes from the XML instance document. This element is useful in making clearer assertions and diagnostics. The element has an optional attribute, path, which is an expression evaluated in the current context that returns a string that is the name of a node. If the path attribute is not set, the name of the current node is used.

A.2.7 ns

The ns element is used to specify the namespaces and prefixes used in the Schematron schema. The element has a required attribute, prefix, which is an XML name with no colon character.

The element has a second required attribute, `uri`, which is a namespace Uniform Resource Identifier.

A.2.8 `param`

The `param` element is used to provide parameters to an abstract pattern. The element has a required attribute, `name`, which is an XML name with no colon. The element has a second required attribute, `value`, whose value is an XPath expression that is evaluated in the current context.

A.2.9 `pattern`

The `pattern` element is used to specify a relationship that must exist in the data found in an XML instance document. The element contains a set of related rules constraining the data. The element has an attribute, `id`, which is a unique name for the set of rules.

The element has another attribute, `abstract`, which, if set to true, defines an abstract pattern. If the abstract attribute value is true, the `id` attribute is required.

The element has an `is-a` attribute. If this attribute's value specifies the name of an abstract pattern, then the pattern is considered an instance of that abstract pattern and must not contain any `rule` elements.

The element has three optional attributes that can be used to create rich interfaces and documentation: `icon`, `see`, and `fpi`. The element can also contain `title` and `p` elements to provide additional documentation.

A.2.10 `phase`

The `phase` element is used to specify a group of patterns, enabling combinations of different patterns to be executed at different stages of validation. The element has a required attribute, `id`, which is the name of the phase. The element has three optional attributes, `icon`, `see`, and `fpi`, which can be used to create rich interfaces and documentation.

A.2.11 `report`

The `report` element is used to make an assertion in the current context. The element has a required `test` attribute whose value is an XPath expression of the assertion test to be evaluated in the current context. The element contains rich text used to express the assertion to be reported in natural language. If the assertion test evaluates to true, the report succeeds and the rich text is output.

The element has seven optional attributes that can be used to create rich interfaces and documentation, provide more detailed outcomes, and allow identification of some part of a pattern. These attributes are `diagnostics`, `icon`, `see`, `fpi`, `flag`, `role`, and `subject`.

A.2.12 `rule`

The `rule` element is used to specify a list of assertions tested within a context node in an XML instance document. The element has a required attribute, `context`, which is an XPath expression that specifies the rule context expression.

The element has another attribute, `abstract`, which, if set to true, defines an abstract rule. If the `abstract` attribute value is true, the element must not have a `context` attribute. An abstract rule is a list of assertions invoked by other rules within the same pattern using the `extends` element.

The element has seven optional attributes that can be used to create rich interfaces and documentation, provide more detailed outcomes, and allow identification of some part of a pattern. These attributes are `diagnostics`, `icon`, `see`, `fpi`, `flag`, `role`, and `subject`.

A.2.13 `schema`

The `schema` element is the top-level element of a Schematron schema. This element has an optional `schemaVersion` attribute that specifies the version of the schema. The optional `queryBinding` attribute specifies the short name of the query language binding in use. The optional `defaultPhase` attribute can be used to indicate the default phase to use during processing.

The element has three optional attributes that can be used to create rich interfaces and documentation. These attributes are `icon`, `see`, and `fpi`. The element can also contain `title` and `p` elements to provide additional documentation.

The ISO Schematron elements are in the namespace `http://purl.oclc.org/dsdl/schematron`:

```
<sch:schema
xmlns:sch="http://purl.oclc.org/dsdl/schematron"></sch:schema>
```

A.2.14 `value-of`

The `value-of` element is used to find or calculate values from an XML instance document. The `value-of` element allows for clearer assertions and diagnostics. The required `select` attribute is an XPath expression that is evaluated in the current context and returns a string value.

APPENDIX B: REFERENCES

1. Assured Information Sharing (AIS) Technologies and Products Office, National Security Agency, *XML Schema and RELAX NG Guidance for Cross Domain Security Policy Enforcement*, January 2008.
2. World Wide Web Consortium, *Extensible Markup Language (XML) 1.1 (Second Edition)*, World Wide Web Consortium Recommendation, 16 August 2006 (edited 29 September 2006). Available at http://www.w3.org/TR/xml11/.
3. Simmons, Robin A., *XML-Based Enforcement of Cross-Domain Security Policies*, MITRE Technical Report MTR 03W0000045, The MITRE Corporation, October 2003.
4. World Wide Web Consortium, *XML Schema Part 0: Primer, Second Edition*, 28 October 2004. Available at http://www.w3.org/TR/xmlschema-0/.
5. World Wide Web Consortium, *XML Schema Part 1: Structures, Second Edition*, 28 October 2004. Available at http://www.w3.org/TR/xmlschema-1/.
6. World Wide Web Consortium, *XML Schema Part 2: Datatypes, Second Edition*, 28 October 2004. Available at http://www.w3.org/TR/xmlschema-2/.
7. International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), *Information Technology – Document Schema Definition Language (DSDL)*, Part 2: "Regular-grammar-based validation - RELAX NG," ISO/IEC 19757-2:2003, 1 December 2003. Available at http://standards.iso.org/ittf/PubliclyAvailableStandards/c037605_ISO_IEC_19757-2_2003%28E%29.zip.
8. Schematron International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), *Information Technology – Document Schema Definition Languages (DSDL)*, Part 3: "Rule-based Validation – Schematron," ISO/IEC 19757-3:2006, 1 June 2006. Available at http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip.
9. Chairman of the Joint Chiefs of Staff, *Defense-In-Depth: Information Assurance (IA) and Computer Network Defense (CND)*, Chairman of the Joint Chiefs of Staff Manual 6510.01, Enclosure C, Appendix I, Paragraph 4.k, 10 August 2004.
10. Intelligence Community Chief Information Officer, *Intelligence Community Inter-Domain Transfer Policy*, Draft, January 2000.
11. World Wide Web Consortium, *XPath 2.0*, 23 January 2007. Available at http://www.w3.org/TR/xpath20/.
12. World Wide Web Consortium, *Extensible Stylesheet Language Transformations 2.0*, 23 January 2007. Available at http://www.w3.org/TR/xslt20/.

This page intentionally left blank.

APPENDIX C: GLOSSARY

C.1 Acronyms

AFRL	Air Force Research Laboratory
ASCII	American Standard Code for Information Interchange
C	Confidential
CDS	Cross Domain Solution
DoD	Department of Defense
DSDL	Document Schema Description Language
JFCOM	Joint Forces Command
IC	Intelligence Community
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
ISSE	Information Support Server Environment
NSA	National Security Agency
NZ	New Zealand
S	Secret
SPAWAR	Space and Naval Warfare Systems Command
SVRL	Schematron Validation Report Language
TS	Top Secret
UCDMO	Unified Cross Domain Management Office
UK	United Kingdom
XML	Extensible Markup Language
XPath	XML Path Language
XSLT	Extensible Stylesheet Language

C.2 Terms

Data Constraint: A condition that must hold within a single document or across documents that are being released within the same package

Schema: An XML document that defines a class of XML documents

Validate: Determine if an XML instance document properly conforms to a schema

XML Instance Document: An XML document that conforms to a document class