# Enforcing the Complex Business Rules of NIEM Exchanges

Priscilla Walmsley

Joel Byford

# What are Business Rules?

- Constraints on data in an exchange
  - Usually contextual or conditional
  - Can be clearly stated
  - Ideally can be automatically enforced
- Examples:

| |
|---|
| A person must have a last name. |
| An activity start date must precede an activity end date. |
| A person should not have a death date if their "living indicator" is set to "true". |
| Every arrest must have a related incident. |

# Why Formally Define Business Rules?

- Validation
  - They provide an automated way of checking whether a message is coherent and complete.
- Documentation
  - They shed valuable light on the purpose and relationship of data elements in an exchange.
- A formal contract
  - They serve as an understanding among parties sharing information about what is to be exchanged.

# How Strict Should Rules Be?

- The strictness of the rules depends on the use case.
  - Use stricter rules if:
    - The information is being processed to perform a particular activity, e.g. registering a gun or filing a court document.
    - The information is destined for a database or other system that requires data integrity.
  - Use less strict rules if:
    - The information is to be gathered from many different providers, all of whom might have different levels of completeness and data integrity.
    - The information will only be used for display to an end user.

# More on Rule Strictness

- Consider having different sets of rules for different phases or information partners in an exchange.

- Consider having rules with different strictness levels.
  - Rules that identify what MUST happen
    - result in an error when violated
  - Rules that identify what SHOULD happen
    - result in a warning when violated

# Technologies for Enforcing Business Rules

- NIEM Subset/Exchange/Extension Schemas (XML Schema)
- NIEM Constraint Schemas (XML Schema)
- Schematron
- Application code

Expressiveness increases as you go down this list

# NIEM Subset/Exchange/Extension Schemas

- When you subset NIEM, you impose simple constraints on the NIEM model by:
  - eliminating the elements that you are not using
  - controlling the minimum and maximum occurrences of each element (0, one, many)
  - limiting code lists (valid values) to the subset you want
  - choosing whether elements are nillable or not
- Your extension and exchange schemas can contain these same kinds of constraints.

# NIEM Constraint Schemas

- Like other NIEM schemas, they are expressed in XML Schema.
  - ...but they are subject to far fewer rules for NIEM conformance
- They *add* constraints to existing NIEM Schemas.
  - Instances must be valid according to *both* the original NIEM schema and the constraint schema.
- Constraint schemas usually apply to subset schemas, but can also apply to extension and exchange schemas.

# Constraint Schemas: Simple Data Types

- Patterns (`pattern`)

  SSN can be either 9 digits of the form NNNNNNNNN, or 11 characters of the form NNN-NN-NNNN where 'N' is a digit.

- Specific code lists (`enumeration`)

  Data sensitivity classification level for each message must be "SBU" or "LES".

- Length (`length`, `minLength`, `maxLength`)

  An arrest transaction number must be at least 5 characters and can be no more than 50 characters

# Constraint Schemas: Simple Data Types (cont.)

- Ranges (`minInclusive`, `minExclusive`, `maxInclusive`, `maxExclusive`)

  The angle of direction must be between -179 and 180.

- Decimal precision (`totalDigits`, `fractionDigits`)

  The longitude seconds value can be expressed to a maximum of 4 decimal places

# Constraint Schema Simple Data Type Example

SSN can be either 9 digits of the form NNNNNNNNN, or 11 characters of the form NNN-NN-NNNN where 'N' is a digit.

```
<xsd:simpleType name="SSNType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-\d{2}-\d{4}|\d{9}"/>
  </xsd:restriction>
</xsd:simpleType>
```

more examples at http://datypic.com/niemruleexamples

# Constraint Schemas: Cardinalities

- A choice of elements that may (or must) appear

  Either a person full name, or a person first name AND person last name must be provided (but not all three).

- Cardinality interdependence

  If a county is provided, then a state must also be provided.

NOTE: These constraints only work within a single element/type in XML Schema; you cannot express interdependence among components that appear in different parts of a document.

# Constraint Schema Cardinality Example

```
<xsd:complexType name="PersonNameType">
  <xsd:complexContent>
    <xsd:extension base="s:ComplexObjectType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:sequence>
            <xsd:element ref="nc:PersonGivenName"/>
            <xsd:element ref="nc:PersonSurName"/>
          </xsd:sequence>
          <xsd:element ref="nc:PersonFullName"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Either a person full name, or a person first name AND person last name must be provided (but not all three).

# Constraint Schemas: Cardinalities (cont.)

- Context-specific cardinalities

Organization ID is required for law enforcement organizations, but optional for criminal organizations

NOTE: This requires some reworking of schemas, since typically all organizations will share the same type: OrganizationType. A new type would need to be defined, specifically for law enforcement organizations, that requires Organization ID.

# Constraint Schemas: Pros and Cons

**Advantages**

- No need to learn a new schema language
- Formally part of the NIEM architecture

**Disadvantages**

- Not much more expressive than regular NIEM schemas; can't express complex business rules
- Can be cumbersome and require significant departure from the original NIEM schemas

# What is Schematron?

- ISO Standard schema language
- A rules-based schema language as opposed to a grammar-based schema language
  - Rather than defining every element in the exchange, it simply lists the rules to which the messages must conform.
- Uses XPath (1.0 or 2.0) as a way to express the rules
- More details later!

# Schematron Rules

- Cross-field value constraints

  Arrest date must be the same as or prior to the booking date.

- Cross-field value/cardinality constraints

  A person must not have a death date if their "living indicator" is set to "true".

- Dynamic constraints

  A person's birth date cannot be in the future.

# Schematron Rules 2

- Cross-reference checking

  All persons provided must be part of an association indicating their relationship to the incident activity.

  All gang organization references must point to an organization whose category is "GANG".

- ...plus all the kinds of constraints that can be expressed in XML Schema

# Schematron Example

A person must not have a death date if their "living indicator" is set to "true".

```
<sch:pattern id="NIEMNTE-1">
  <sch:title>NIEMNTE-1: Person Death Date</sch:title>
  <sch:rule
    context="nc:Person[nc:PersonLivingIndicator = 'true']">

    <sch:assert test="not(nc:PersonDeathDate)">A person must
       not have a death date if their "living indicator" is
       set to "true".</sch:assert>

  </sch:rule>
</sch:pattern>
```

# Schematron: Pros and Cons

**Advantages**

- Much more expressive than XML Schema
- A standardized, portable way of expressing rules
- Likely to fit in with the future NIEM architecture

**Disadvantages**

- Difficult to refer to external sources, e.g. databases or other applications
- Scope of validation is a single XML document

# Application Code

- Most messages in an exchange are typically parsed and processed by application code.
  - e.g. Java or C# classes are used to read the message, break it down into its data elements, and process them in some way (put them in a database, call another application, display them to a user)
- Rules can also be enforced during this process.
  - either directly upon reading the XML message, or later by imposing constraints on the objects that are created
- This is the most flexible but least portable method of enforcing rules.

# Application Code Rules

- Cross-document constraints

  > Message sequence number must be a unique sequential number assigned to the message to differentiate a message from previous message submissions and order message processing.

- External validation constraints

  > Data submitter organization ORI must be completed with the nine-character identifier (ORI) assigned by the FBI CJIS staff to the organization requesting the transaction.

- ...plus all the kinds of constraints that can be expressed in XML Schema or Schematron

# Application Code: Pros and Cons

**Advantages**

- Maximum flexibility and expressiveness
- Can access external sources
  - Databases, other applications' APIs, Web services, etc.
- Will generally perform better

**Disadvantages**

- Platform- and/or IT-environment-dependent
- Unlikely to be a formal part of the IEPD, shared/used by all parties in an exchange

# Rules Not Suitable for Automation

- Hard to automate rules

  The Data Submitter organization must not contain an abbreviation.

  Photos of vehicles must show a side view of the vehicle.

- Subjective rules

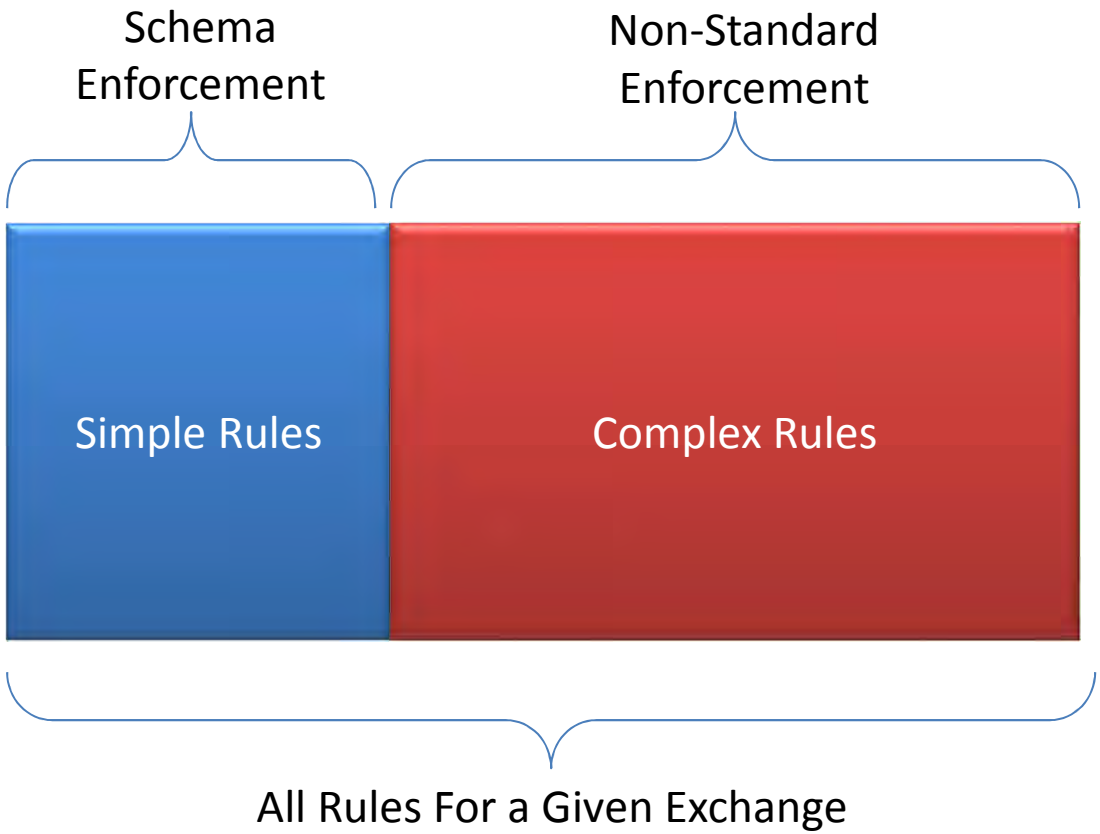
  Persons must represent different people; there can not be two person elements that represent the same physical person.

  A person description must not include information redundant with other elements, such as hair color or eye color.

# Documenting Business Rules

- Provide human-readable documentation of the rules.
  - The IEPD Master Document is a good place for this.
- Give each rule a unique identifier to facilitate debugging and error reporting.
- Clearly state which data elements the rules refer to, possibly by using the NIEM names in the rule description.
- Document all rules, whether they are automatable or not.
  - If you are providing a validation report to users, you can also list the rules that must be checked manually.

# Recap – Not All Rules Enforced by Schema



# Recap – Schema Capabilities

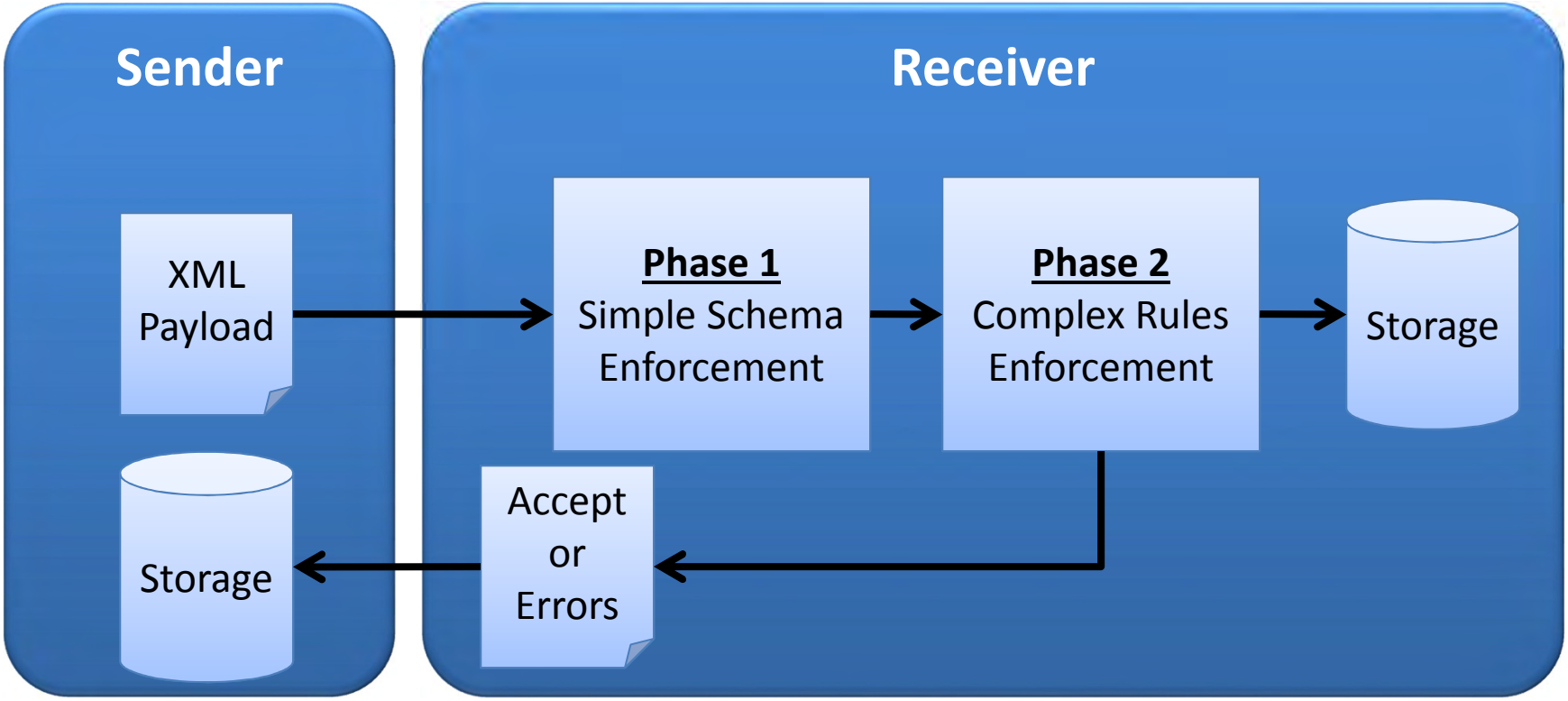
## Enforceable

- Code List Enforcement
  - *e.g. Eye Color Code*
- Strong Data Typing
  - *e.g. Boolean)*
- String Pattern Enforcement
  - *e.g. xxx-xx-xxxx*
- Cardinalities
  - *e.g. at least 1 charge per Citation*

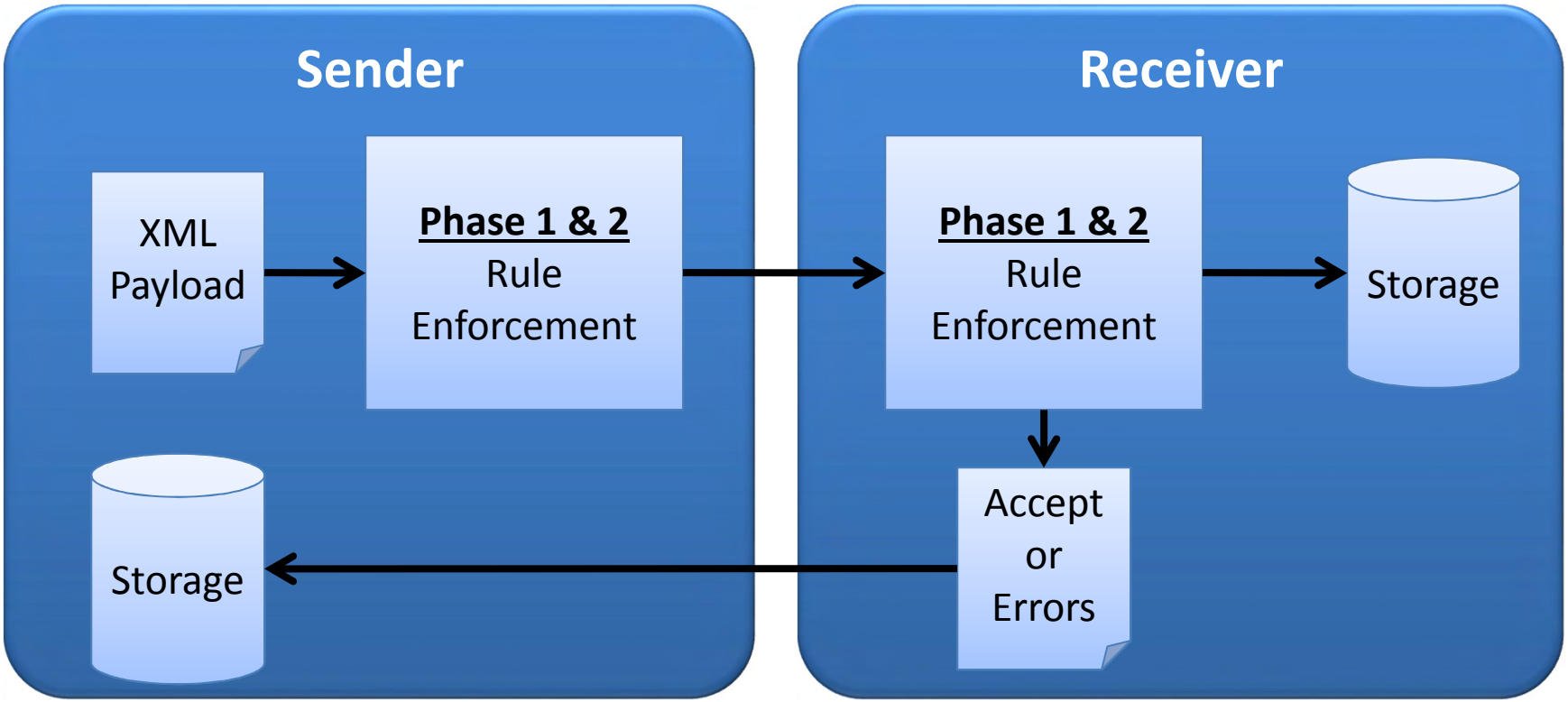
## Unenforceable

- Conditional Requirements
  - e.g. Parent if under 18 years
- Cross-Field Enforcement
  - e.g. Honda vs. Ford Mustang
- Externally Defined Code List
  - e.g. Plain text list of statutes

# Two-Phase Validation



# Consistently Applied Two-Phase Validation



# Potential Issues with Approach

- Parties Must Consistently Adopt and Enforce Rules
  - In order to be most effective, each exchange stakeholder should help enforce the rules.

- Inconsistent Application of Rules
  - Without a standardized Machine-Readable way to enforce rules across different systems, rules are often applied inconsistently.

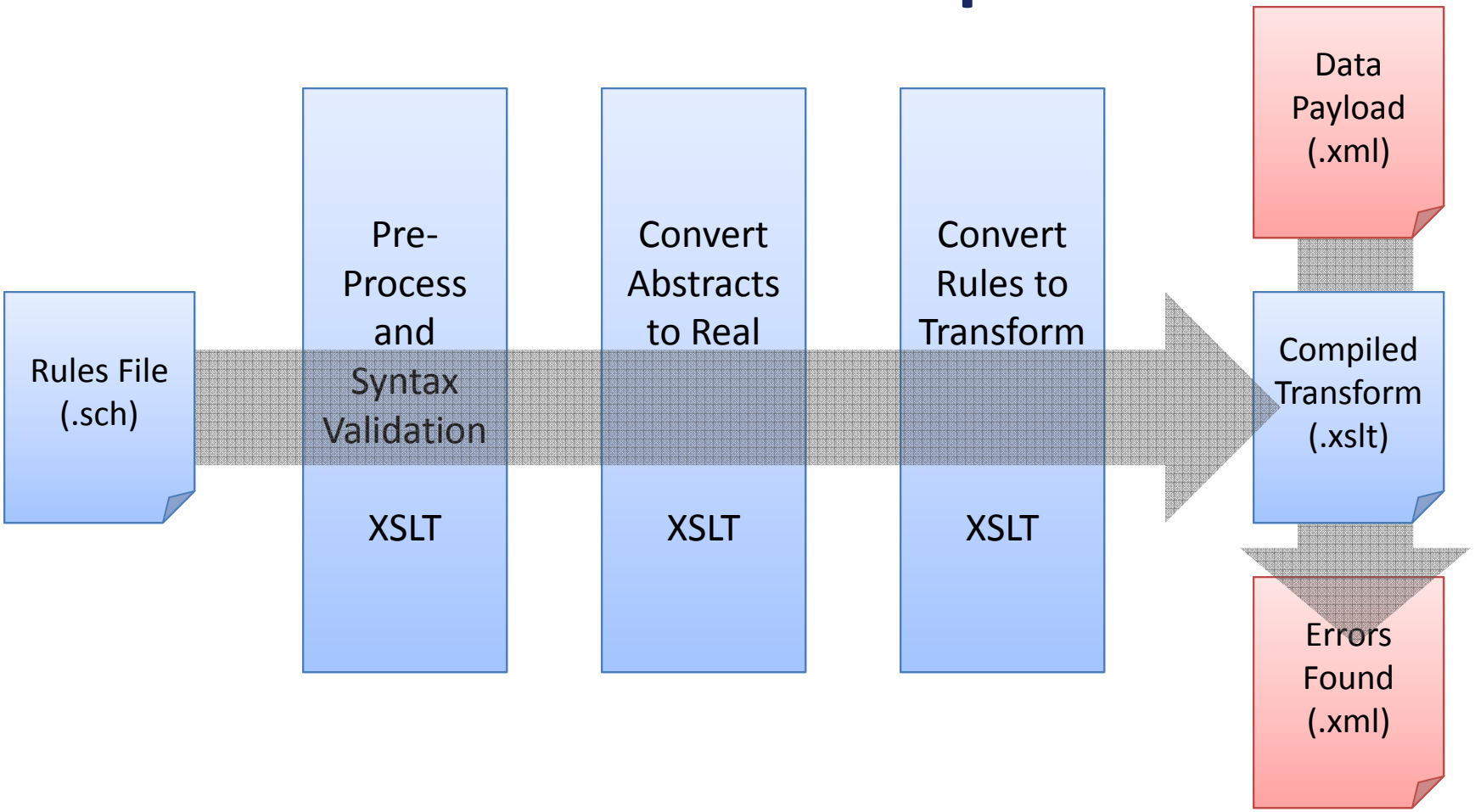
# Ultimately Required – Standardization of 2nd Phase

- Open Web Service Option
  - Online Calls to an Externalized API or Service
  - Requires Network Connectivity for All Parties
  - Allows for Consistent Real-Time Enforcement
- Published Rule-Sets Option
  - Machine Readable yet Technology Agnostic
  - Ideally Limited to Leveraging Modern and Standard Technology.
  - Schematron a Possible and ISO Standard Approach fitting the bill.

# Recap - Schematron

- ISO Standard - Open ISO Standard adopted by countless agencies worldwide (ISO/IEC 19757-3).
- Platform Agnostic - Based on and "Compiles" to XSLT which can be applied on any number of systems consistently.
- Easy Distribution - Distribution of updates simple through usage XML files.
- Leverage Existing XML Investment - Developers leverage existing knowledge in XPath and XQuery.
- Free - Royalty free to implement.

# Reference Schematron Implementation



# Schematron Examples and Exercise

Please navigate to the following from your laptop to see examples of Schematron and Perform a instructor-led exercise to craft and apply a rudimentary rule-set in Schematron:

http://app.xmlclassroom.com/

User ID: [Anything Unique]

Passcode: NIEM-NTE

# Supporting Slides

# Header

```
<?xml version="1.0" encoding="UTF-8"?>

<schema
   xmlns="http://purl.oclc.org/dsdl/schematron"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   queryBinding="xslt"
   schemaVersion="1">
```

Any Number of valid bindings supported including XQuery, XSLT1, XSLT2, etc.

# Namespace Support

```
<ns uri="http://niem.gov/niem/niem-core/2.0" prefix="nc"/>
<ns uri="http://www.w3.org/1999/XSL/Transform" prefix="xsl"/>
<ns uri="http://www.w3.org/2001/XMLSchema" prefix="xsd"/>
<ns uri="http://www.niematron.org/examples/survey/"
prefix="surv"/>
```

Allows rules to leverage various namespaces in expressions.

# Patterns & Rules

```
<pattern id="OtherRuleEnforcement">

  <title>Other value provided</title>

  <rule context = "/surv:SurveyDocument /SurveyQuestion
                          [CODE-ELEMENT = 'OTHER']">

    <assert test="string-length(DESCRIPTION-ELEMENT) &gt; 0“>
                ERROR MESSAGE
    </assert>
  </rule>
</pattern>
```

Loop through each element within the provided “context” and apply <assert> and <report> tests to each.

# Patterns & Rules

```
<pattern id="OtherRuleEnforcement">

  <title>Other value provided</title>

  <rule context = "/surv:SurveyDocument /SurveyQuestion
                              [CODE-ELEMENT = 'OTHER']">

    <assert test="string-length(DESCRIPTION-ELEMENT) &gt; 0“>
                  ERROR MESSAGE
    </assert>
  </rule>
</pattern>
```

Fill in standard XPath and XSLT statements to evaluate .

# Patterns & Rules

```
<pattern id="OtherRuleEnforcement">

  <title>Other value provided</title>

  <rule context = "/surv:SurveyDocument /SurveyQuestion
                          [QuestionResponseCode = 'OTHER']">

    <assert test="string-length(QuestionResponseText) &gt; 0“>
                Text must be provided if "OTHER" is selected!
    </assert>
  </rule>
</pattern>
```

# Exercise 1 – Compile SCH to XSLT

- Use the template and code snippet tool to create a valid Schematron file.
- Add a rule verifying a description is provided when 'Other' is selected.
- Save file.
- Validate the file.
- Transform to XSLT.
- Examine resulting XSLT.

# Exercise 2 – Apply the Compiled XSLT

- Open the sample XML payload provided in the examples.
- Save the file as XML.
- Transform the sample payload using the compiled XSLT created in Excercise 1.
- Examine the results.

# Exercise 3 – Break the Rules

- Open the sample XML payload provided in the examples.
- Modify the XML to break or otherwise violate the rule being tested.
- Save the file as XML.
- Transform the sample payload using the compiled XSLT created in Excercise 1.
- Examine the results.