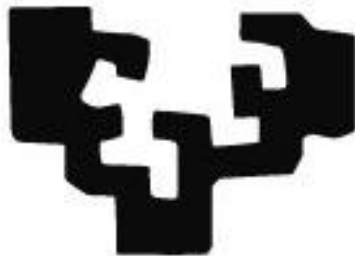


# Schematron

eman ta zabal zazu



informatika  
fakultatea



facultad de  
informática

**Arantza Irastorza Goñi**

**Informazioaren Kudeaketa Aurreratua**

**Gradua Ingeniaritza Informatikoan**

**Esp. Software Ingeniaritza**

**Lengoiak eta Sistema Informatikoak saila**

2017 urtarrila

# Outline

- Why Schematron? Business Rules
- Schematron constructs
- Validation process
- Facing domain experts
- Examples (taken from AMIS)
- References

Acknowledgements: Examples borrowed from R. Costelo

# Outline

## ➡ Why Schematron? Rules, Business Rules

- Schematron constructs
- Validation process
- Facing domain experts
- Examples (taken from AMIS)
- References

Acknowledgements: Examples borrowed from R. Costelo

# Business Rules

- Definition of “business rules” has been standardized by OMG since Jan 2008

## rule

Definition:

proposition **that** is a claim of obligation **or** of necessity

Dictionary Basis:

one of a set of explicit or understood regulations or principles governing conduct or procedure within a particular area of activity ... a law or principle that operates within a particular sphere of knowledge, describing, or prescribing what is possible or allowable. [ODE]

## business rule

Definition:

rule **that is** under business jurisdiction

General Concept:

rule, element of guidance

Note:

A rule's being “under business jurisdiction” means that it is under the jurisdiction of the semantic community that it governs or guides - that the semantic community can opt to change or discard the rule. Laws of physics may be relevant to a company (or other semantic community); legislation and regulations may be imposed on it; external standards and best practices may be adopted. These things are not business rules from the company's perspective, since it does not have the authority to change them. The company will decide how to react to laws and regulations, and will create business rules to ensure compliance with them. Similarly, it will create business rules to ensure that standards or best practices are implemented as intended. See subclause A.2.3

# Business rules: Example

## ➤ Rule

- Document content should only be disclosure to appropriate agents

## ➤ Business rule:

- a document will be composed of paragraphs
- both documents and paragraphs can be classified along a fixed set of values (e.g. top-secret, confidential, unclassified)
- the <Para> classification value cannot be more sensitive than the <Document> classification value

# Business rules: Example

```
<Document classification="secret">  
  <Para classification="unclassified"> One if by land, two if by sea; </Para>  
  – <Para classification="confidential">  
    And I on the opposite shore will be, Ready to ride and spread the alarm  
  </Para>  
  – <Para classification="unclassified">  
    Ready to ride and spread the alarm Through every Middlesex, village and farm,  
  </Para>  
  – <Para classification="secret">  
    For the country folk to be up and to arm.  
  </Para>  
</Document>
```

# Realization of business rules

- the presence and **order of elements** in an XML instance document,
  - the **number of occurrences** of each element,
  - the **contents and datatype** of each element and attribute
- 
- the **value** of certain elements must conform to a rule or algorithm

**XML  
Schema**

**????**

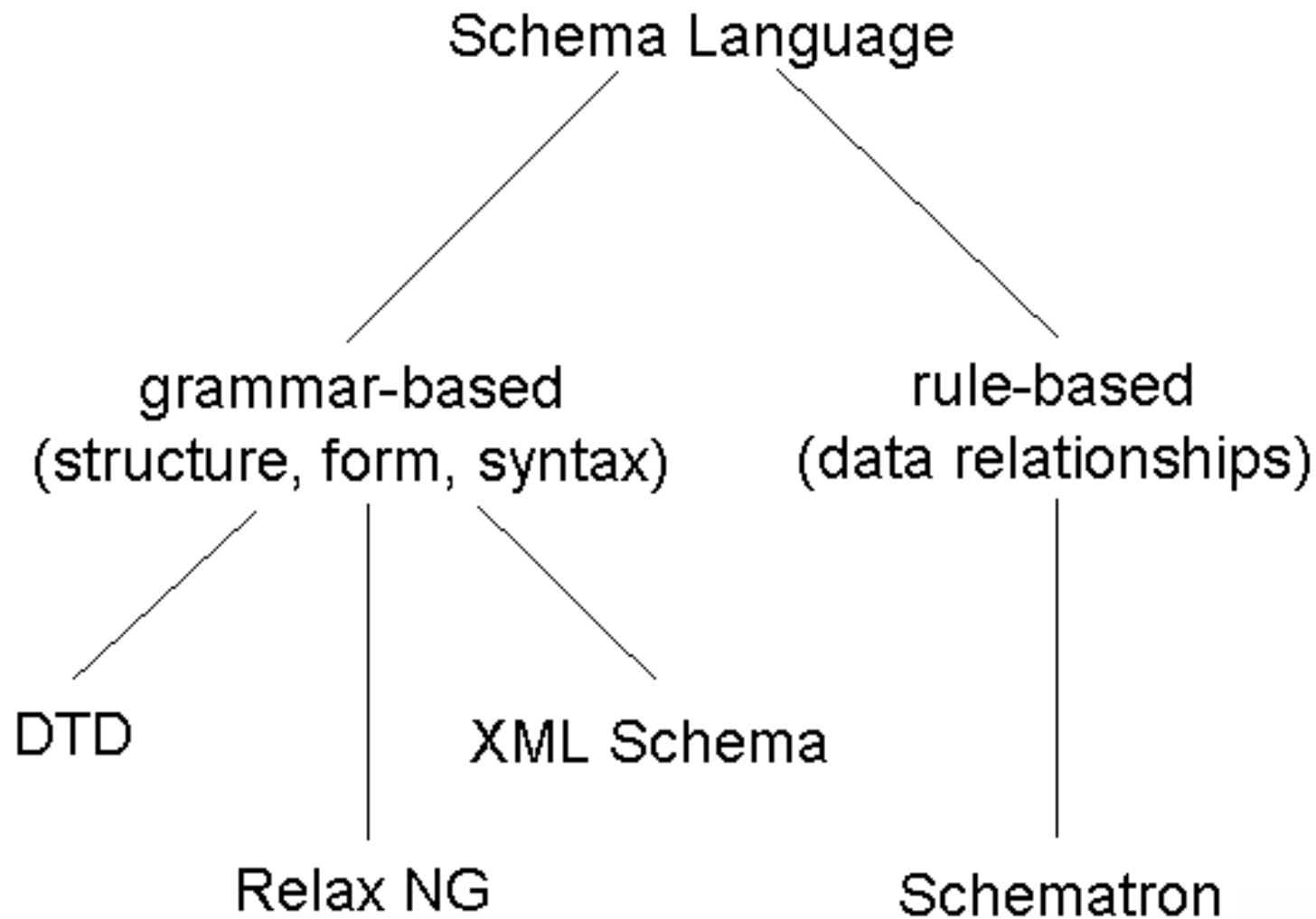
# Business rules: Example

- XML-Schema expressible rules:
  - *a document will be composed of paragraphs*
  - *both documents and paragraphs can be classified along a fixed set of values (e.g. top-secret, confidential, unclassified)*
- How to ensure...?
  - [co-constraint] *a <Para> classification value cannot be more sensitive than the <Document> classification value*





# Business rules: two approaches



# Schematron

- An open language for making assertions about data in XML documents
  - Using XPath expressions
  - To:
    - Verify data interdependencies (co-constraints)
    - Check data cardinality
    - Perform algorithm checking
- Its specification is standardized: ISO/IEC 19757

# Outline

- Why Schematron? Rules, Business Rules
- ➡ Schematron constructs
- Validation process
- Facing domain experts
- Examples (taken from AMIS)
- References

Acknowledgements: Examples borrowed from R. Costelo

# Schematron schema

- A Schematron schema is an XML document
  - The Schematron elements are in this namespace:

<http://purl.oclc.org/dsdl/schematron>

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
...
</sch:schema>
```

- By convention, the filename of a Schematron schema has the suffix ".sch"
- Also, by convention the namespace prefix used in Schematron schemas is "sch:" (e.g. *sch:assert*)

# Schematron schema (2)

- it supports one of more business rules
- Rule:
  - *a <Para> classification value cannot be more sensitive than the <Document> classification value*
- Business Rules (assertions)
  - *If there is a Para labelled "top-secret" then the Document must be labelled "top-secret"*
  - *If there is a Para labelled "secret" then the Document must be labelled either "secret" or "top-secret"*
  - *If there is a Para labelled "confidential" then the Document must be labelled either "confidential", "secret" or "top-secret"*

```
<?xml version="1.0"?>  
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
```

If there is a Para labeled "top-secret" then the Document must be labeled top-secret

If there is a Para labeled "secret" then the Document must be labeled either secret or top-secret

If there is a Para labeled "confidential" then the Document must be labeled either confidential, secret or top-secret

```
</sch:schema>
```

# <assert>

- Simple declarative sentence in natural language
  - A test attribute with a Xpath expression
  - The contents of <assert> is a human-readable version of the XPath expression
- The content is displayed, if the assert's test is false

```
<sch:assert test="/Document[@classification='top-secret']">
```

If there is a Para labeled "top-secret" then the Document must be labeled top-secret

```
</sch:assert>
```

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
```

```
<sch:assert test="/Document[@classification='top-secret']">
  If there is a Para labeled "top-secret" then the Document
  must be labeled top-secret
</sch:assert>
```

```
<sch:assert test="(/Document[@classification='top-secret']) or
  (/Document[@classification='secret'])">
  If there is a Para labeled "secret" then the Document
  must be labeled either secret or top-secret
</sch:assert>
```

```
<sch:assert test="(/Document[@classification='top-secret']) or
  (/Document[@classification='secret']) or
  (/Document[@classification='confidential'])">
  If there is a Para labeled "confidential" then the Document
  must be labeled either confidential, secret or top-secret
</sch:assert>
```

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

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

```
</sch:schema>
```

```
<Document classification="secret">
  <Para classification="unclassified"> One if by land, two if
  - <Para classification="confidential">
    And I on the opposite shore will be, Ready to ride and sp
  </Para>
  - <Para classification="unclassified">
    Ready to ride and spread the alarm Through every Middl
  </Para>
  - <Para classification="secret">
    For the country folk to be up and to arm.
  </Para>
</Document>
```



# <rule>

- Used to group assertions that share the same context
- Has a context attribute
  - To specify the context in the XML document where the assertion should be applied
  - If the context matches, the assertions are tested
- The validator evaluates every assertion within a rule and generates an error for every assertion that fails

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
```

```
<sch:rule context="Para[@classification='top-secret']">
```

```
<sch:assert test="/Document[@classification='top-secret']">
  If there is a Para labeled "top-secret" then the Document
  must be labeled top-secret
</sch:assert>
```

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

```
<sch:rule context="Para[@classification='secret']">
```

```
<sch:assert test="(/Document[@classification='top-secret']) or
  (/Document[@classification='secret'])">
  If there is a Para labeled "secret" then the Document
  must be labeled either secret or top-secret
</sch:assert>
```

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

```
<sch:rule context="Para[@classification='confidential']">
```

```
<sch:assert test="(/Document[@classification='top-secret']) or
  (/Document[@classification='secret']) or
  (/Document[@classification='confidential'])">
  If there is a Para labeled "confidential" then the Document
  must be labeled either confidential, secret or top-secret
</sch:assert>
```

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

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

```
</sch:schema>
```

```
<Document classification="secret">
  <Para classification="unclassified"> One if by land, two if
  - <Para classification="confidential">
    And I on the opposite shore will be, Ready to ride and sp
  </Para>
  - <Para classification="unclassified">
    Ready to ride and spread the alarm Through every Middl
  </Para>
  - <Para classification="secret">
    For the country folk to be up and to arm.
  </Para>
</Document>
```

# <pattern>

➤ Used to group rules having the same objective

- ~~name~~ attribute will be displayed in the output when the pattern is checked (failed or even succeed)
  - When failed the output also contain the content of the assertion

optional

```
<sch:pattern id="...">  
  <sch:p>Brief about the pattern's objective</sch:p>  
  
  <sch:rule context="..."> ... </sch:rule>  
  <sch:rule context="..."> ... </sch:rule>  
  <sch:rule context="..."> ... </sch:rule>  
  
</sch:pattern>
```

```

<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:pattern id="SCP">
    <!-- Security Classification Policy -->
    <sch:p>A Para's classification value cannot be more sensitive
      than the Document's classification value.</sch:p>

    <sch:rule context="Para[@classification='top-secret']">

      <sch:assert test="/Document/@classification='top-secret'">
        If there is a Para labeled "top-secret" then the Document
        must be labeled top-secret
      </sch:assert>

    </sch:rule>

    <sch:rule context="Para[@classification='secret']">

      <sch:assert test="(/Document/@classification='top-secret') or
        (/Document/@classification='secret')">
        If there is a Para labeled "secret" then the Document
        must be labeled either secret or top-secret
      </sch:assert>

    </sch:rule>

    <sch:rule context="Para[@classification='confidential']">

      <sch:assert test="(/Document/@classification='top-secret') or
        (/Document/@classification='secret') or
        (/Document/@classification='confidential')">
        If there is a Para labeled "confidential" then the Document
        must be labeled either confidential, secret or top-secret
      </sch:assert>

    </sch:rule>
  </sch:pattern>
</sch:schema>

```

# <phase>

- **Progressive validation** is the validation of constraints in **phases**
  - Rather than validating everything all at once
- By default, all <pattern> elements are active. However, the **<phase> element enables** you to override the default behavior, and **control which <pattern> elements are active**
  - Active patterns referenced through **<active>** element
    - Its **pattern** attribute has an IDREF value, that references an id attribute on a <pattern> element
- The <phase> element is uniquely identified with an **id** attribute

# <phase>

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
  <sch:phase id="...">
    <sch:active pattern="..."> ... </sch:active>
    ...
  </sch:phase>
  <sch:phase id="..."> ... </sch:phase>
  ...
  <sch:pattern id="..."> ... </sch:pattern>
  <sch:pattern id="..."> ... </sch:pattern>
  ...
  <sch:diagnostics>...</sch:diagnostics>
</sch:schema>
```

# <phase>. Example

- Two rules applied to the same document
  - *Security Classification Policy* (SCP)
  - *Reserved Word Filter* (RWF)

```
<sch:pattern id="RWF">
  <sch:p>These reserved words are not allowed anywhere in the
    document: SCRIPT, FUNCTION.</sch:p>
  <sch:rule context="Document">

    <sch:assert test="count(//node())[contains(.,'SCRIPT')]] = 0">
      The document must not contain the word SCRIPT
    </sch:assert>
    <sch:assert test="count(//node())[contains(.,'FUNCTION')]] = 0">
      The document must not contain the word FUNCTION
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

# <phase>. Example

- At validation time, either of the two phases can be selected

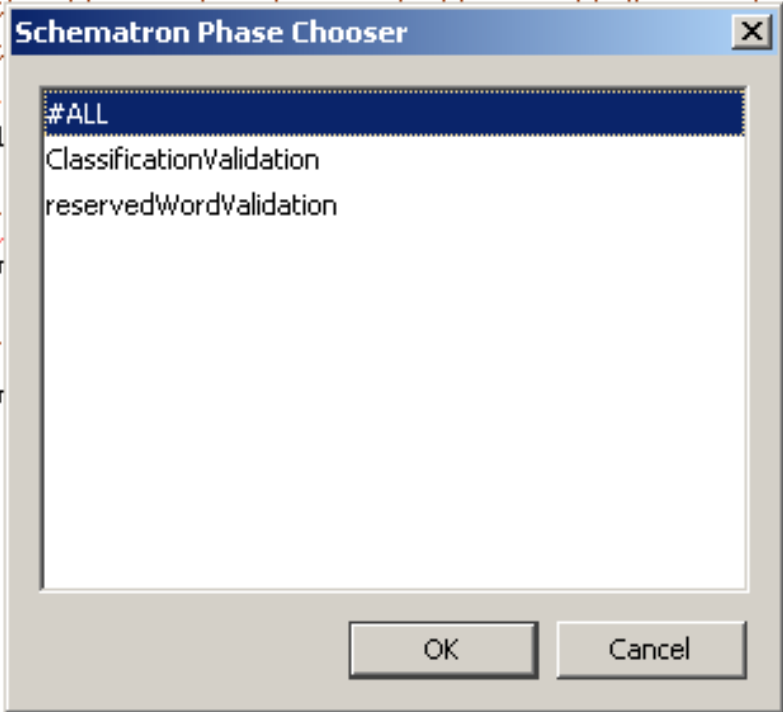
```
<sch:phase id="classificationValidation">
  <sch:active pattern="SCP" />
  <sch:active pattern="xxx" />
  ....
</sch:phase>
<sch:phase id="reservedWordValidation">
  <sch:active pattern="RWF" />
  <sch:active pattern="zzz" />
  ....
</sch:phase>
<sch:pattern id="SCP">...</sch:pattern>
<sch:pattern id="RWF">...</sch:pattern>
<sch:pattern id="xxx">...</sch:pattern>
<sch:pattern id="zzz">...</sch:pattern>
<sch:pattern id="ppp">...</sch:pattern>
```



# <phase>. Example

## ➤ In the validation ...

```
xml version="1.0" encoding="UTF-8"?>
document xmlns="http://www.schematron.info/document6"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.schematron.info/document6
    classification="secret"
    <Para classification="secret"
      One if by land
    </Para>
    <Para classification="secret"
      One if by land; tw
    </Para>
    <Para classification="secret"
      One if by land; tw
    </Para>
  </document>
```



The image shows a 'Schematron Phase Chooser' dialog box. It has a title bar with a close button. The main area is a list box containing three items: '#ALL' (which is selected), 'ClassificationValidation', and 'reservedWordValidation'. At the bottom of the dialog are 'OK' and 'Cancel' buttons.

# Namespaces

- Schematron schemas for XML instance documents that use namespaces ...

```
<?xml version="1.0"?>
<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron">
  <sch:ns uri="http://www.example.org" prefix="ex" />

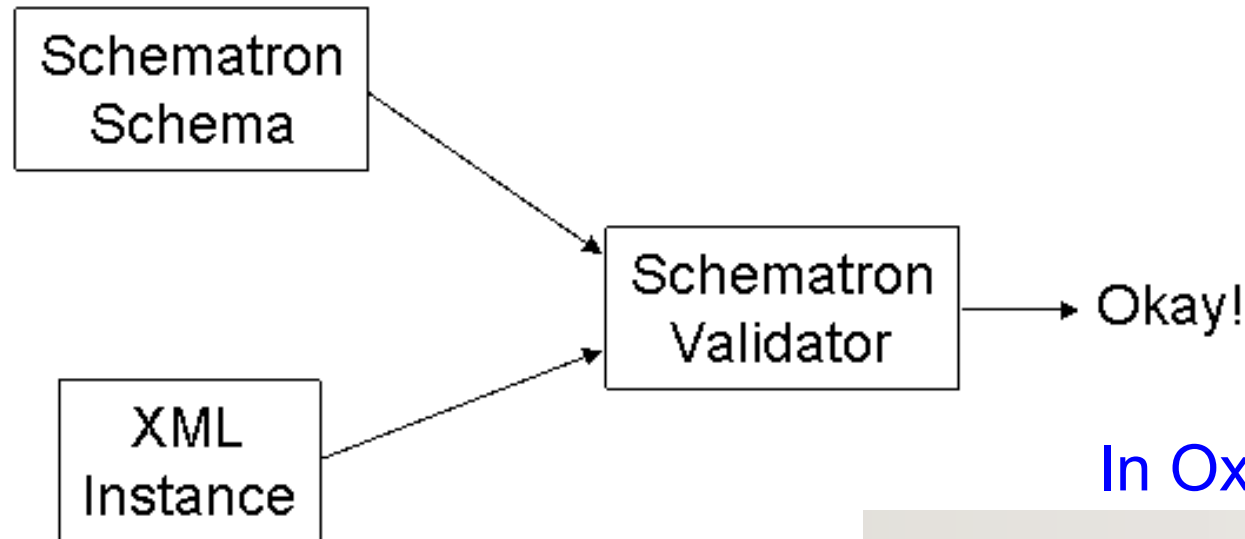
  <sch:pattern name="Security Classification Policy">
    <sch:p>A Para's classification value cannot be more sensitive
      than the Document's classification value.</sch:p>
    <sch:rule context="ex:Para[@classification='top-secret']">
      <sch:assert test="/ex:Document/@classification='top-secret'">
        If there is a Para labeled "top-secret" then the Document
        should be labeled top-secret
      </sch:assert>
    </sch:rule>
  </sch:pattern> ...
```

# Outline

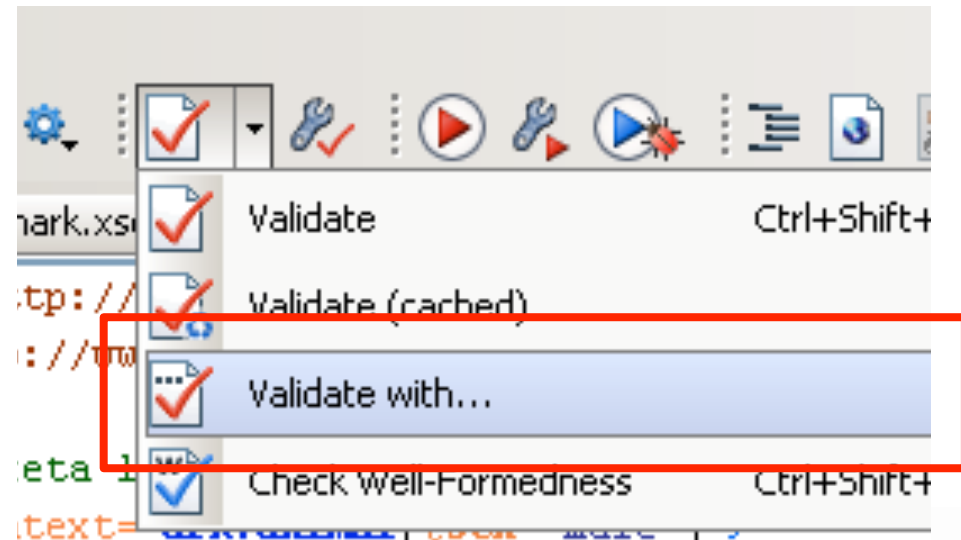
- Why Schematron? Rules, Business Rules
- Schematron constructs
- ➡ Validation process
- Facing domain experts
- Examples (taken from AMIS)
- References

Acknowledgements: Examples borrowed from R. Costelo

# Validation of an instance document



In Oxygen Editor ...



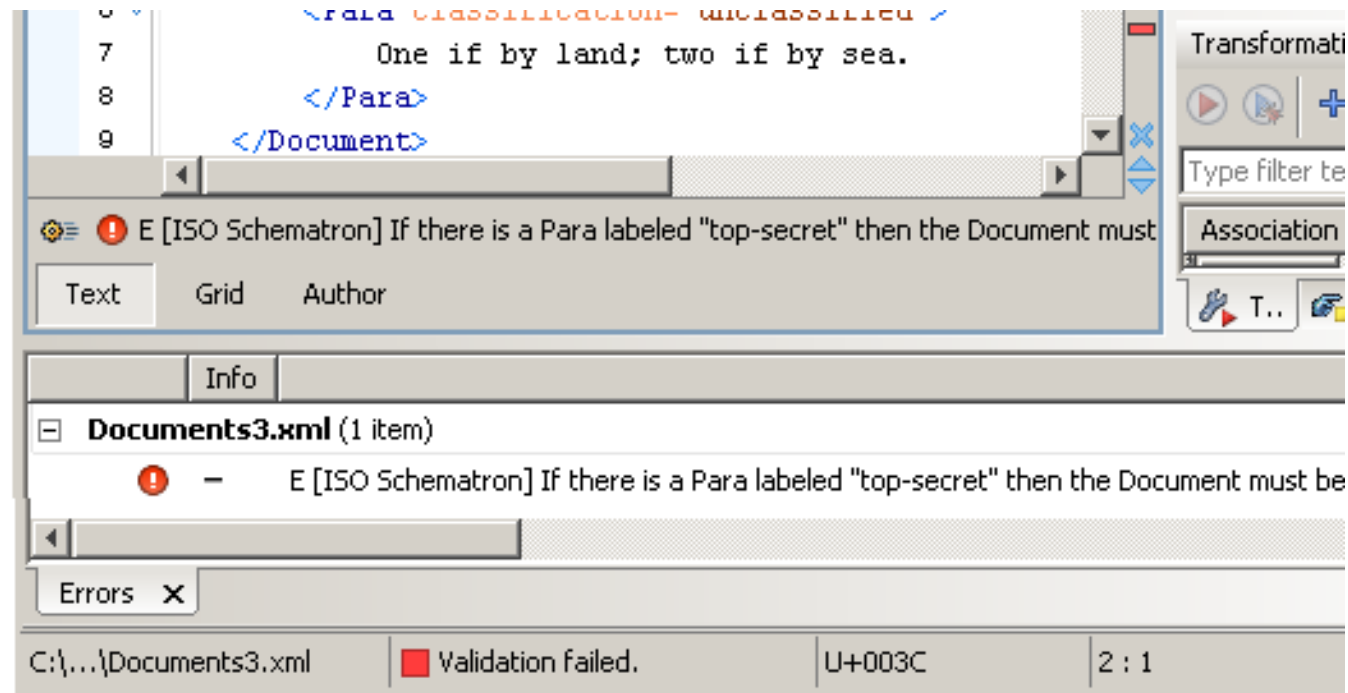
And specify the \*.sch file

# Validation of an instance document

- Visit each element in the XML instance document, in document order
- For each element,
  - look for the active patterns (in case of phases)
  - look for a rule that specifies a context which matches the element;
  - fire the first rule in each pattern that matches;
  - process all assertions in the rule.

# Validation of an instance document

## ➤ Output of the validation...



# Validation of an instance document

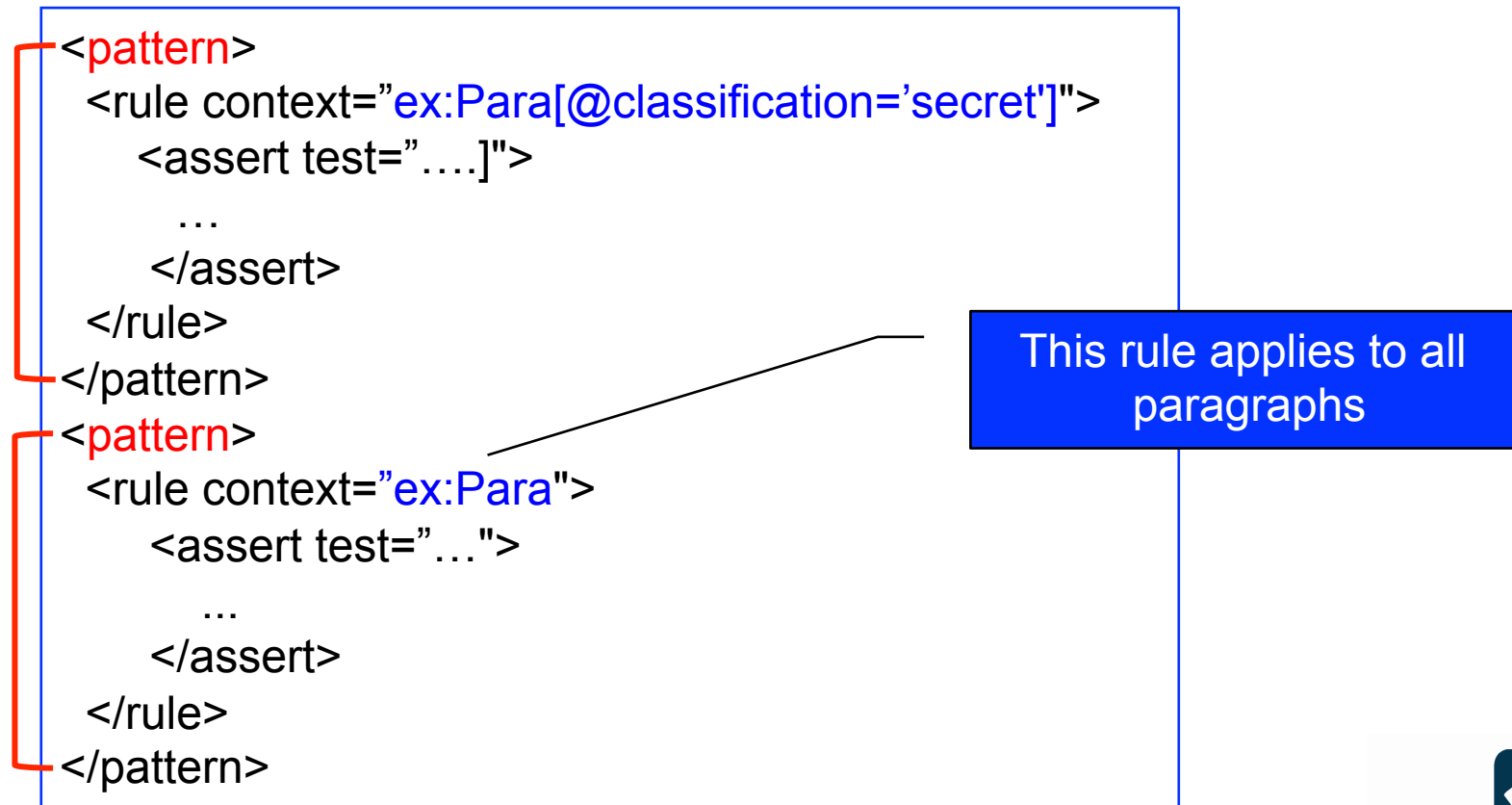
- If a node applies to the context condition of several rules of a pattern, **the node will only be selected for the first suitable rule in the pattern**

```
<pattern>
  <rule context="ex:Para[@classification='secret']">
    <assert test="....]">
      ...
    </assert>
  </rule>
  <rule context="ex:Para">
    <assert test="...">
      ...
    </assert>
  </rule>
</pattern>
```

This rule will only apply  
to paragraphs that are  
NOT secret

# Validation of an instance document

- If a node applies to the context condition of several rules of a pattern, the node will only be selected for the first suitable rule in the pattern





# Validation of an instance document. Two ways to check Schematron rules

## ➤ In isolation

```
<?xml version="1.0" encoding="UTF-8"?>  
<?oxygen SCHSchema="checkClassification.sch" type="xml"?>  
<Document> ...
```

it contains the  
Schematron rules

## ➤ In conjunction with the XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>  
<?xml-model href="topsecret.xsd" type="application/xml"  
  schematypens="http://purl.oclc.org/dsdl/schematron"?>  
<Document xmlns= ...> ...
```

it contains the  
Schematron rules

# To check Schematron rules: in isolation

```
<?xml version="1.0" encoding="UTF-8"?>
<?oxygen SCHSchema="Documents.sch" type="xml"?>

<Document xmlns="http://www.schematron.info/document"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.schematron.info/document Documents.xsd"
  classification="secret">

  <Para classification="unclassified">
    One if by land; two if by sea.
  </Para>
  <Para classification="top-secret">
    And I on the opposite shore will be,
  </Para>

</Document>
```

# To check Schematron rules: in conjunction with xml schema

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="Documents5.xsd" type="application/xml"
  schematypens="http://purl.oclc.org/dsdl/schematron"?>

<Document xmlns="http://www.schematron.info/document5"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.schematron.info/document5 Documents5.xsd"
  classification="secret">

  <Para classification="unclassified">
    One if by land; two if by sea.
  </Para>
  <Para classification="top-secret">
    And I on the opposite shore will be,
  </Para>

</Document>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:sch="http://purl.oclc.org/dsdl/schematron" targetNamespace="http://www.schematron.info/document5"
  xmlns="http://www.schematron.info/document5" elementFormDefault="qualified">
```

```
<xs:annotation>
  <xs:appinfo>
    <sch:title>some rules on Document...</sch:title>
    <sch:ns uri="http://www.schematron.info/document5" prefix="doc"/>
  </xs:appinfo>
</xs:annotation>
```

An xml schema file  
with schematron  
rules embedded

```
<xs:element name="Document">
  <xs:complexType>
    <xs:sequence> <xs:element ref="Para" maxOccurs="unbounded"></xs:element> </xs:sequence>
    <xs:attribute name="classification"> ... </xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="Para" >
```

```
<xs:annotation>
  <xs:appinfo>
    <sch:pattern id="bat">
      <sch:rule context="doc:Para[@classification='top-secret']">
        <sch:assert test="/doc:Document/@classification='top-secret'"> If there is a Para labeled ...</sch:assert>
      </sch:rule>
      ...
    </sch:pattern>
  </xs:appinfo>
</xs:annotation>
```

```
<xs:complexType> .... </xs:complexType>
</xs:element>
</xs:schema>
```

# Outline

- Why Schematron? Rules, Business Rules
- Schematron constructs
- Validation process
- ➡ Facing domain experts
- Examples (taken from AMIS)
- References

Acknowledgements: Examples borrowed from R. Costelo

# Domain experts

- Oftentimes the results of a validation are delivered to users who are not XML experts, or technology whizzes
- Issues:
  - the ‘how’: how to describe **unfulfillments of the business rules** (i.e. the assertion text)
  - the ‘what’: how to describe **unfulfillments of the rule** (i.e. the whole pattern)
  - how to **trace** the schematron rules back to its source

## Issue 1: how to describe unfulfillments of the business rules (i.e. the assertion text)

- *“The meeting's startTime must be before its endTime”*
- This is expressed using a Schematron rule:

```
<sch:rule context="meeting">  
  <sch:assert test="startTime < endTime">  
    Assertion Text  
  </sch:assert>  
</sch:rule>
```

Specify the Assertion Text  
using **terminology of the  
problem domain**

- Read as: *"In the context of the meeting element, I assert that the value of startTime must be less than the value of endTime"*

The assertion is intended to be used as a statement of a contractual obligation

➤ Precision and definiteness are the overriding consideration



➤ Contractual assertions:

- *The meeting's start time shall be before its end time*
- *The meeting's start time must be before its end time*



# The assertion is intended to be used as a friendly message to guide users

- Users react strongly to message they think are strident and which seem to blame the user. So, shifting "must" or "shall" to "should" may be appropriate:
  - *The meeting's start time should be before its end time*
- An important consideration in phrasing Assertion Text is to decide whether you are addressing :
  - people who know what the rules are, and just need to be told they've slipped up, or
  - people who don't know the rules and need to be told what they are

## Issue 2: how to describe unfulfillments of the rule (i.e. the whole pattern)

- The `<diagnostic>` element enables to compose user-friendly error messages
- It is comprised of mixed content: text and `<sch:value-of>` elements.

```
<sch:diagnostic id="invalid-security-label">
```

Your document does not meet the Security Classification Policy. Your Para has a security label of `<sch:value-of select="./@classification"/>`, which has a higher sensitivity than the document's sensitivity:

```
<sch:value-of select="Document/@classification"/>
```

```
</sch:diagnostic>
```

- Read as: this diagnostic has an identifier (*invalid-security-label*). When invoked, this diagnostic is to output some text along with the value of the classification attribute of the current - context - node (Para), and the value of the classification attribute of the `<Document>` element

# Associating Assertions to Diagnostic

- The <assert> element has an optional **diagnostics** attribute. The value of the diagnostics attribute is the value of a <diagnostic> element's id attribute

```
<sch:assert test="/Document/@classification='top-secret'"
```

```
  diagnostics="invalid-security-label">
```

If there is a Para labeled "top-secret" then the Document  
should be labeled top-secret

```
</sch:assert>
```

- During validation, if the assertion fails, then the diagnostic identified as *"invalid-security-label"* will be executed

## Issue 3: how to trace the schematron rules back to its source

### ➤ Traceability ensures that:

- spurious constraints are not introduced,
- the successful implementation of requirements can be ascertained,
- requirements that have unworkable ramifications can be identified

# Schematron “see” attribute

- The assert element has an optional **see** attribute, which may be used to link an assertion to the requirement it implements

```
<sch:assert test="@classification='top-secret' or  
                @classification='secret' or  
                @classification='confidential' or  
                @classification='unclassified'  
                see='http://www.example.com/military/security.html#A.1.2.3.4.5'>
```

The value of a classification must be one of top-secret, secret, confidential, or unclassified, because of MILSPEC XXXX (1999) section A.1.2.3.4.5

```
</sch:assert>
```

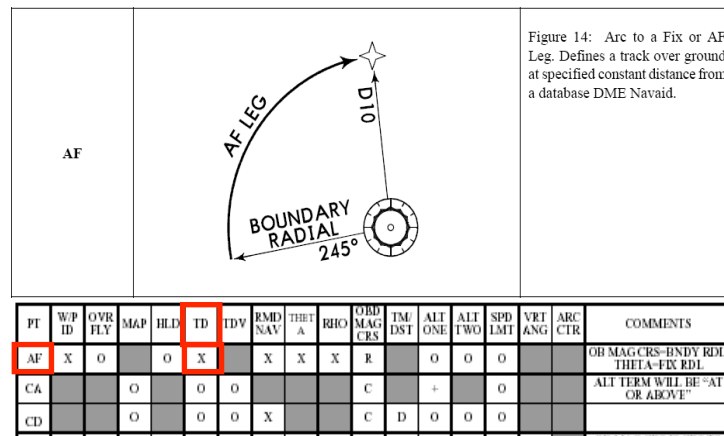
# Outline

- Why Schematron? Rules, Business Rules
- Schematron constructs
- Validation process
- Facing domain experts
- ➡ Examples (taken from AMIS)
- References

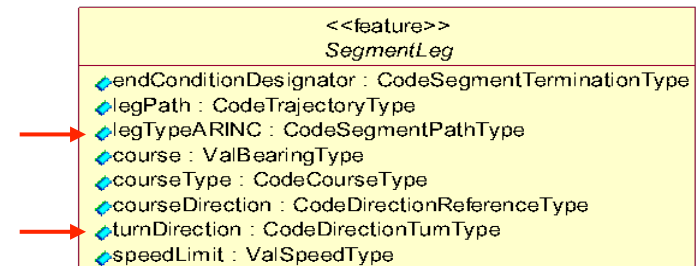
Acknowledgements: Examples borrowed from R. Costelo

# Validating technical documents

A424: *an AF leg requires a turn direction*



Translation in AIXM: *property turnDirection is required for a SegmentLeg of type AF*



## Simplified Schematron example

```

<SegmentLeg legTypeARINC="AF" turnDirection="L">
  <legPath>...</legPath>
  ...
</SegmentLeg>
    
```

```

<rule id="1" context="//
SegmentLeg[@legTypeARINC='AF'] ">

  <assert id="1_1"
    test="boolean(@turnDirection)">
  </assert>
</rule>
    
```

**<rule id="1"**

-> unique identifier of the rule

**context="// SegmentLeg[@legTypeARINC='AF'] "**

-> Defines the conditions of the rule (If attribute SegmentLeg.legTypeARINC= "AF").

->The rule will be tested if the context is true.

**<assert id="1\_1"**

-> unique identifier of the assertion

**test="boolean(@turnDirection)"**

-> the logical test to be performed. The "assert" element matches if the logical test returns false (in this case, if the turnDirection is not provided).

# Algorithmic validation.

## Example: Validating polls

- Consider this XML instance document:

```
<xml version="1.0"?>
<Election>
  <ResultsByPercentage>
    <Candidate name="John">61</Candidate>
    <Candidate name="Sara">24</Candidate>
    <Candidate name="Bill">15</Candidate>
  </ResultsByPercentage>
</Election>
```

- *Vote Count Rule*: For the instance document to be valid, the sum of the *Candidate* values must be 100



# Algorithmic validation.

## Example: Validating polls

```
<?xml version="1.0"?>
```

```
<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron">
```

```
<sch:pattern id="VoteCount">
```

```
<sch:p>The election results must add up to 100%.</sch:p>
```

```
<sch:rule context="ResultsByPercentage">
```

```
<sch:assert test="sum(Candidate) = 100">
```

```
  The sum of the election results must be 100
```

```
</sch:assert>
```

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

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

```
</sch:schema>
```

# Example: AMIS case study

- Two xml schemas: *common.xsd* , *company.xsd*
- Business rules:
  - An employee may not be the manager of himself.
  - All normal employees (so not a manager) must have a manager (I've added this one).
  - There is only one manager without a manager (only one president).
  - All employees should have less salary than their respective manager.
  - All employees should have less salary than any manager (=employee in department "Managers").
  - The relation manager and employee is a valid one, so the manager of an employee must exist.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:cmp="http://amis.nl/schematrondemo/company"
  xmlns:com="http://amis.nl/schematrondemo/common">
  <soapenv:Header/>
  <soapenv:Body>
    <cmp:Company>
      <cmp:Department name="Ground One" abbr="GR1">
        <cmp:Employees>
          <com:Employee id="10" manager_id="19">
            <com:Name>J. Jansen</com:Name>
            <com:Salary>1500</com:Salary>
          </com:Employee>
          <com:Employee id="11" manager_id="15">
            <com:Name>K. Smith</com:Name>
            <com:Salary>1600</com:Salary>
          </com:Employee>
        </cmp:Employees>
      </cmp:Department>
      <cmp:Department name="Ground Two" abbr="GR2">
        <cmp:Employees>
          <com:Employee id="13" manager_id="20"> ... </com:Employee>
          <com:Employee id="14" manager_id="20"> ... </com:Employee>
        </cmp:Employees>
      </cmp:Department>
      <cmp:Department name="Managers" abbr="MAN">
        <cmp:Employees>
          <com:Employee id="15"> ... </com:Employee>
          <com:Employee id="20" manager_id="25"> ... </com:Employee>
          <com:Employee id="25"> ... </com:Employee>
        </cmp:Employees>
      </cmp:Department>
    </cmp:Company>
```

example.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:com="http://amis.nl/schematrondemo/common"
  xmlns="http://amis.nl/schematrondemo/company"
  targetNamespace="http://amis.nl/schematrondemo/company"
  elementFormDefault="qualified">
  <xsd:import schemaLocation="Common.xsd"
    namespace="http://amis.nl/schematrondemo/common"/>
  <xsd:element name="Company">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Department" type="tDepartment" maxOccurs="unboundec
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="tDepartment">
    <xsd:sequence>
      <xsd:element name="Employees" type="com:tEmployees"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="abbr" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:schema>
```

Company.xsd

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://amis.nl/schematrondemo/common"
  targetNamespace="http://amis.nl/schematrondemo/common"
  elementFormDefault="qualified">
  <xsd:element name="Employees" type="tEmployees"/>
  <xsd:complexType name="tEmployees">
    <xsd:sequence>
      <xsd:element name="Employee" type="tEmployee" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="tEmployee">
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string"/>
      <xsd:element name="Salary" type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:long" use="required"/>
    <xsd:attribute name="manager_id" type="xsd:long" use="optional"/>
  </xsd:complexType>
</xsd:schema>
```

# AMIS case study. Examples (1)

An employee may not be the manager of himself

```
<pattern id="example1">  
  <rule context="//com:Employee[@manager_id]">  
    <assert test="@manager_id != @id">Employee may not manage himself</assert>  
  </rule>  
</pattern>
```

```
<xsd:complexType name="tEmployee">  
  <xsd:sequence>  
    <xsd:element name="Name" type="xsd:string"/>  
    <xsd:element name="Salary" type="xsd:decimal"/>  
  </xsd:sequence>  
  <xsd:attribute name="id" type="xsd:long" use="required"/>  
  <xsd:attribute name="manager_id" type="xsd:long" use="optional"/>  
</xsd:complexType>
```

# AMIS case study. Examples (2)

All normal employees (so not a manager) must have a manager

```
<pattern id="example2">
  <rule context= "//cmp:Company/cmp:Department[@name!='Managers']/
                 cmp:Employees/com:Employee">
    <assert test="@manager_id">Employee must have a manager</assert>
  </rule>
</pattern>
```

oap/envelope/"

```
<soapenv:Body>
  <cmp:Company>
    <cmp:Department name="Ground One" abbr="GR1">
      <cmp:Employees>
        <com:Employee id="10" manager_id="19">
          <com:Name>J. Jansen</com:Name>
          <com:Salary>1500</com:Salary>
        </com:Employee>
        ....
      </cmp:Employees>
    </cmp:Department>
    <cmp:Department name="Ground Two" abbr="GR2">
      <cmp:Employees>
        <com:Employee id="13" manager_id="20"> ... </com:Employee>
        <com:Employee id="14" manager_id="20"> ... </com:Employee>
      </cmp:Employees>
    </cmp:Department>
    <cmp:Department name="Managers" abbr="MAN">
      <cmp:Employees>
        <com:Employee id="15"> ... </com:Employee>
        ...
      </cmp:Employees>
    </cmp:Department>
  </cmp:Company>
</soapenv:Body>
```

# AMIS case study. Examples (3)

There is only one manager without a manager (only one president)

```
<pattern id="example3">
  <rule context="//cmp:Company/cmp:Department[@name='Managers']/cmp:Employees">
    <assert test="count(com:Employee[not(@manager_id)]) = 1">
      There should be only one president
    </assert>
  </rule>
</pattern>
```

```
      <com:Name>J. Jansen</com:Name>
      <com:Salary>1500</com:Salary>
    </com:Employee>
    ....
  </cmp:Department>
  <cmp:Department name="Ground Two" abbr="GR2">
    <cmp:Employees>
      <com:Employee id="13" manager_id="20"> ... </com:Employee>
      <com:Employee id="14" manager_id="20"> ... </com:Employee>
    </cmp:Employees>
  </cmp:Department>
  <cmp:Department name="Managers" abbr="MAN">
    <cmp:Employees>
      <com:Employee id="15"> ... </com:Employee>
      <com:Employee id="20" manager_id="25"> ... </com:Employee>
      <com:Employee id="25"> ... </com:Employee>
    </cmp:Employees>
  </cmp:Department>
```



# AMIS case study. Examples (4)

All employees should have less salary than their respective manager

```
<pattern id="example4">
  <rule context="//cmp:Department[@name!='Managers']/cmp:Employees/com:Employee">
    <assert test="com:Salary <
      (/com:Employee[@id = current()]/@manager_id)/com:Salary)">
      Employee earns too much
    </assert>
  </rule>
</pattern>
```

```
    <com:Employee id="10" manager_id="19">
      <com:Name>J. Jansen</com:Name>
      <com:Salary>1500</com:Salary>
    </com:Employee>
    ....
  </cmp:Department>
  <cmp:Department name="Managers" abbr="MAN">
    <cmp:Employees>
      <com:Employee id="19"> ... </com:Employee>
      <com:Employee id="20" manager_id="25"> ... </com:Employee>
      <com:Employee id="25"> ... </com:Employee>
    </cmp:Employees>
```

# AMIS case study. Examples (5)

All employees should have less salary than any manager  
(manager = employee in department “Managers”)

```
<pattern id="example4">
  <rule context="//cmp:Department[@name!='Managers']/cmp:Employees/com:Employee">
    <assert test="com:Salary < min(//cmp:Department[@name='Managers']/
      cmp:Employees/com:Employee/com:Salary)">
      Employee earns too much
    </assert>
  </rule>
</pattern>
```

# AMIS case study. Examples (6)

The manager of an employee must exist

```
<pattern id="example5">
  <rule context="//com:Employee[@manager_id]">
    <assert test= "//cmp:Company/cmp:Department[@name='Managers']/
                  cmp:Employees/com:Employee[@id=current()/@manager_id]">
      No valid manager
    </assert>
  </rule>
</pattern>
```

# References

- Tutorial on Schematron by Roger Cosello
  - <http://www.xfront.com/schematron/index.html>
- ISO Schematron tutorial by Dave Pawson, Roger Costello and Florent Georges
  - <http://www.dpawson.co.uk/schematron/>
- Overview of Schematron
  - <http://www.data2type.de/en/xml-xslt-xslfo/schematron/>