Crossroads
The ACM Student Magazine

XCSL Tutorial

(XML Constraint Specification Language)

by ***Marta Jacinto***, ***Giovani Rubert Librelotto***, ***José Carlos Ramalho***, and ***Pedro Rangel Henriques***

Introduction

As a descendant of SGML (Standard Generalized Markup Language) [5], XML (eXtensible Markup Language) [3] allows the specification of a document's structure. XML brought the concept of well-formedness to the world of structured documents. An XML document can be well-formed or valid. A document is **well-formed** if it follows the appropriate XML format and syntax. Specifically, a document is well-formed if for every opening tag there is a corresponding closing tag, there is only one root element, elements do not overlap, attribute values are quoted, and the characters '<' and '&' are only used as tags and entities, respectively.

A document is **valid** if it obeys the rules specified in a Document Type Definition (DTD) [10] or XML Schema [2], files containing rules that define the structure of the document, stating the allowed elements and their positions.

DTDs enable us to specify structure rules and rudimentary dynamic semantics. By obeying the rules specified in the DTD, a document will be valid syntactically. However, *syntactically* correct documents are not necessarily *semantically* correct; for instance, we can not force the value of an element/attribute to be a number. XML Schemas enable us to specify some static semantics (the basic data types are available, but no comparison between the value of elements/attributes is available). However, there are applications where we need to specify more complex invariants or constraints (for instance, the value of x element must be the same as the y element's one). Moreover, some of these constraints are structural but many of them are non-structural and have some degree of complexity.

From the publisher's point of view, it is desirable not to produce invalid documents. For example, consider an official document, describing land under consideration for purchase, that lists the several dates of the previous purchase and the current date. To be semantically correct, each purchase date must be previous to the current date.

To address problems related to semantic correctness, similar to the previous example, we have been working on a solution, XCSL, which enables the specification of extra semantics. This approach is similar to that employed to specify programming languages. We use a specification language to define a set of contextual conditions over the textual content of elements and the attribute values that should be satisfied by an XML instance. Those conditions restrict the set of syntactically correct documents to the set of semantically valid ones.

We can classify constraints as belonging to one of the four categories defined below:

- **Domain range checking**:
  The most common constraint is used to ensure a content/value lies between a pair of values (inside a certain domain). Normally, this kind of constraint is used when data has a type of either date or number.
- **Dependencies between two elements or attributes**:
  Often an element or attribute's value depends on the value of another element or attribute located in a different branch of the document tree. Consequently, a context-dependant constraint is needed.
- **Pattern matching against a Regular Expression**:
  Sometimes we need to guarantee that content follows a certain format (as in the case of dates or telephone numbers, of which there are numerous possible formats).
- **Complex constraints**:
  We denote remaining constraint types 'complex' because they are unusual and require nested loop behavior or complex nested calculations.
  Common complex constraints include:
  - **Mixed content constraint**: When we want to enforce some cardinality and/or some order, for example over a mixed content.
  - **Unicity constraint**: When we want to enforce an uniqueness invariant over an element inside a certain context.

This clear separation in categories and the experience we gained working with XML documents allowed the consolidation of our approach (XCSL - XML Constraint Specification Language) to deal with the semantic specification problem.

Using XCSL is advantageous because:

- XCSL is XML, so we can use all the available tools to manipulate and process XCSL specifications.
- XCSL may be used to formalize all enumerated kinds of constraints, as we are going to see with the reservations example.
- XCSL may be used as a standalone validating application (XML-Schema or DTDs are not always necessary, sometimes a simple XCSL specification can solve the problem, or one may rather use only one language), or together with DTDs or XML-Schemas.

Available Constraint Specification Languages at a Glance

Besides XCSL, we can use XML Schemas, as said before, but also Schematron. Both XCSL and Schematron are able to specify a variety of constraints, whereas XML Schemas are limited to three cases: domain range checking, pattern matching against a regular expression and complex constraints (mixed content) [4].

XCSL

XML Constraint Specification Language (XCSL) is a domain-specific language conceived to allow XML designers to restrict the content of XML documents. It is a small and simple language for writing contextual constraints over the textual value of XML elements and attributes.

The language provides constructors necessary to define the actions to be taken by a semantic validator.

Those actions will be triggered whenever a Boolean expression, over the value of the attributes or the content of the elements, evaluates to false. It means that XCSL also designates a processor that traverses the document's internal representation (the tree) and checks its correctness from a semantic point of view (we shall stress that the structural correctness is validated by the parser that builds the tree).

Moreover, XCSL is an XML language; so it becomes possible to add restrictions to XML documents using an XML dialect. That approach offers document designers a complete XML framework to couple with syntax and semantics.

As shown in **Figure 1**, XCSL processing is easy and fits in the processing of XML documents in a very natural way; that is, we can validate a document semantically the same way we validate it structurally: instead of the XML Parser (to validate the syntax), we invoke a XSL Transformer (to process its semantics). The XSL Stylesheet (which plays the role that the DTD plays in the syntactic validation) that drives the semantic processing, can be generated automatically from the XCSL specification (**http://www.di.uminho.pt/~jcr/PROJS/xcsl-www/**) using a tool like Saxon (**http://users.iclway.co.uk/mhkay/saxon**).

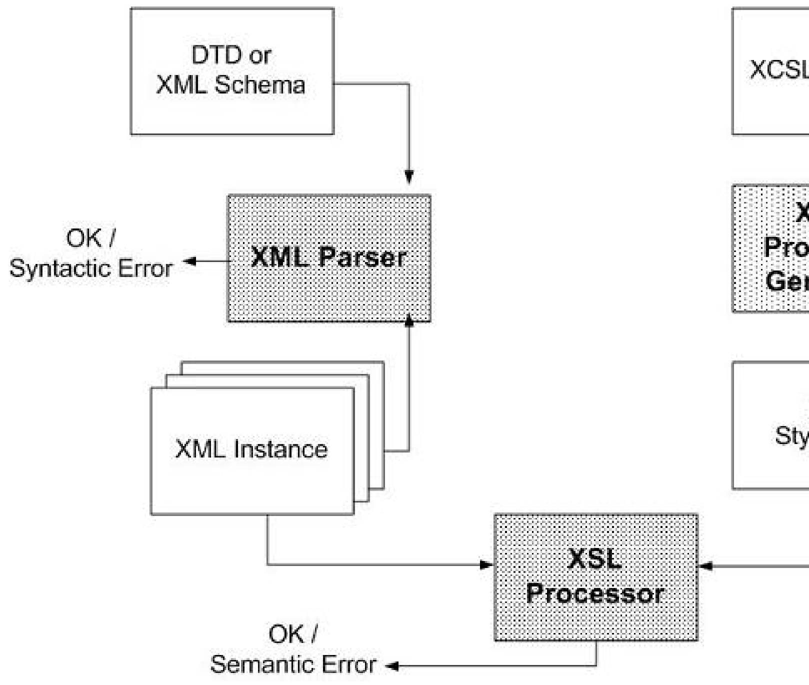


Figure 1: XCSL workflow.

Validating XML instances using XCSL is a two step procedure. First, the constraints document is compiled into a run-time validator. Next, each instance is validated. Users engage both the steps explicitly, invoking a command-line tool like Saxon.

XML Schemas and DTDs

XML Schemas were created when the syntax of DTDs was insufficient for XML users. The aims of the W3C XML Schema Working Group were to create a language that would be more expressive than DTDs and written in XML Syntax. In addition, it would also allow authors to place restrictions on the elements' content and attribute values in terms of primitive datatypes found in most languages.

When using DTDs to specify constraints. we need to use a constraining language (such as XCSL) and, consequently, need two documents to completely validate an XML instance. By using XML-Schemas, only one document is needed to validate XML instances. Unfortunately, the range of constraints we can validate with XML-Schemas is not exhaustive.

To validate XML instances with an XML-Schema, several parsers are available. Those parsers are similar to the traditional XML validators, but now, instead of a DTD, they are driven by an XML-Schema.

Schematron

Schematron is an XML schema language which was designed and implemented by Rick Jelliffe at the Academia Sinica Computing Centre, Taiwan. It combines powerful validation capabilities with a simple syntax and implementation framework. A description of Schematron can be found in [**1**]. Its DTD and XML-Schema can be found at **http://www.ascc.net/xml/schematron/**.

Validating XML instances using Schematron occurs in two stages. First the constraints document is compiled into a run-time validator. Next, each instance is validated by the validator. Two options are available. With the **topologi schematron validator**, the two step process is transparent to the user, and all that he/she needs to do is provide the constraints document and the XML instances. With the **schematron-report**, the user explicitly has to engage both the steps. Whereas schematron-report is processed in the command-line by invoking a tool like Saxon (as we do with XCSL), topologi schematron validator is a window-based interactive environment.

The Reservations Example

Next we present a case-study. It begins with a brief introduction, followed by a DTD designed for this particular family of documents, the complete restrictions document, and the stylesheet generated. Afterwards, we analyze each constraint separately, explaining the goal and the method used to solve each problem. To complete the explanation, we show a set of applications, one for each possible sub-kind of documents. Finally, we show the way to have several output languages and just one constraint document.

The Problem

Let us suppose we want to use an XML document to register the reservations of a certain hotel. To do that, we design a DTD which allows one or more rooms to be reported. Each `room` will have two sub-elements: a group in charge (`gicharge`, made up of two employees - `picharge`) and a set of `events`; and two attributes: `floor` - to indicate the location of the room, and `id` - to uniquely identify the room. Each `events` element has any number of `event` sub-elements, each one referring to an event for which the room is booked. Each `event` has three sub-elements: `date` - made up of the `beginning date` and the `final date`, `companies` - the set of organizer companies, and `title` - the title of the event in which any number of `compn` (company name) and `tradem` (trade-mark) elements may occur. Each `company` is identified by a name (`compn`), an `organizer` and a contact (`compc`).

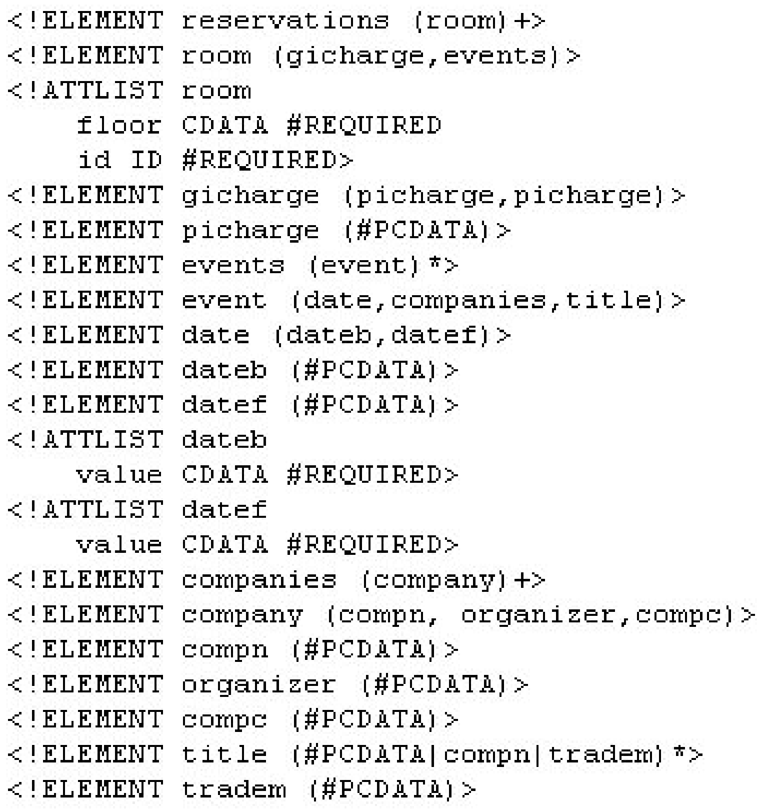
After validating the structure of the document, at least four semantic constraints must be validated in order to have a completely valid document.

These constraints are: first - the value of every `floor` attribute descendant of the `room` elements must be

12 or less (or any other number that represents the number of floors the hotel has); second - for each event, the `final date` must occur after the `beginning date`; third - every contact must have a valid format (only numbers, exactly nine, and begin either with a 2, 91, 93 or 96 - in the portuguese situation); fourth - every `compn` element that occurs inside a certain `title` element must be of an organizer company. Notice that these four constraints correspond to the four categories of constraints enumerated in the Introduction.

DTD

The structure of a document like this can be described with:



```
<!ELEMENT reservations (room)+>
<!ELEMENT room (gicharge,events)>
<!ATTLIST room
    floor CDATA #REQUIRED
    id ID #REQUIRED>
<!ELEMENT gicharge (picharge,picharge)>
<!ELEMENT picharge (#PCDATA)>
<!ELEMENT events (event)*>
<!ELEMENT event (date,companies,title)>
<!ELEMENT date (dateb,datef)>
<!ELEMENT dateb (#PCDATA)>
<!ELEMENT datef (#PCDATA)>
<!ATTLIST dateb
    value CDATA #REQUIRED>
<!ATTLIST datef
    value CDATA #REQUIRED>
<!ELEMENT companies (company)+>
<!ELEMENT company (compn, organizer,compc)>
<!ELEMENT compn (#PCDATA)>
<!ELEMENT organizer (#PCDATA)>
<!ELEMENT compc (#PCDATA)>
<!ELEMENT title (#PCDATA|compn|tradem)*>
<!ELEMENT tradem (#PCDATA)>
```

Solution

In order to solve the enumerated problems, we will need to write the following constraint document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cs>
    <!-- 1 -->
    <constraint>
        <selector selexp="/reservations/room"/>
        <cc>
        @floor <= 12
    </cc>
        <action>
            <message>
            The floor number <value selexp="@floor"/> does not exist.
        </message>
        </action>
    </constraint>
    <!-- 2 -->
    <constraint>
        <selector selexp="//room/events/event/date"/>
        <cc>
        dateb/@value <= datef/@value
    </cc>
        <action>
            <message>
            The final date: <value selexp="datef"/> occurs before the beginning
            date: <value selexp="dateb"/> -this is not allowed.
        </message>
        </action>
    </constraint>
    <!-- 3 -->
    <constraint>
        <selector selexp="//compc"/>
        <cc>
        string-length(number(.)) = 9 and (substring(.,1,1)=2 or substring(.,1,2)=91 or
        substring(.,1,2)=93 or substring(.,1,2)=96)
    </cc>
        <action>
            <message>
            The contact for the company <value selexp="../compn"/> is not a valid
            phone number.
        </message>
        </action>
    </constraint>
    <!-- 4 -->
    <constraint>
        <selector selexp="//title/compn"/>
        <let name="keycompn" value="."/>
        <cc>
    (count(../../companies/company[compn=$keycompn]) >= 1)
    </cc>
        <action>
            <message>
        The title of the event must not contain any company's name outside the set
        of organizer companies, as <value selexp="."/> in a
        reservation of
        the room <value selexp="../../../../@id"/>.
      </message>
        </action>
    </constraint>
</cs>
```

After processing this constraint document with our generator, XCSL, the following XSL stylesheet will be generated (for instance while using Saxon, we would have to write `saxon.exe constraintdoc.xml XCSL.xsl > constraintdoc.xsl`):

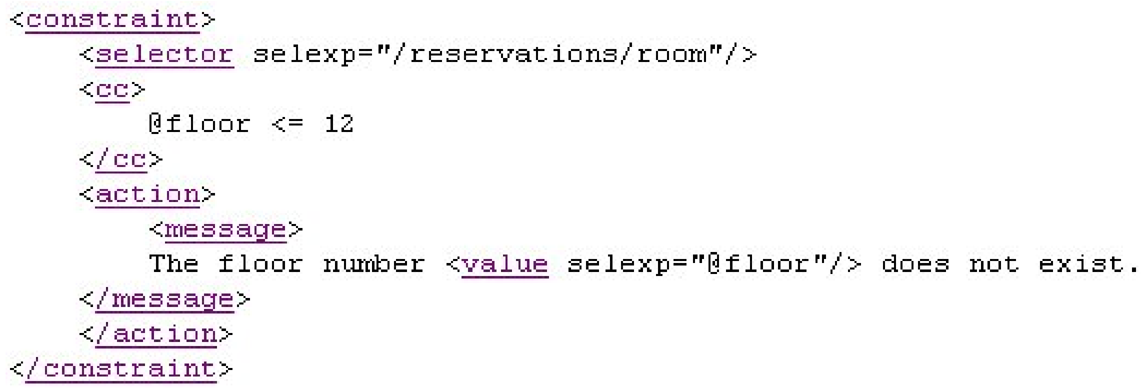
```
<?xml version="1.0" encoding="utf-8" standalone="yes"?><!--
      Preprocessor for the XCSL Language
      http://www.di.uminho.pt/~jcr/PROJS/xcsl-www/
      Copyright (C) 2001 José Carlos Ramalho
      Permission to use granted under GPL or MPL.
      Version: 3.0
    -->
<my:stylesheet xmlns:my="http://www.w3.org/1999/XSL/Transform"
              xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <my:output method="xml" omit-xml-declaration="no" encoding="iso-8859-1"
    standalone="yes" indent="yes"/>
   <my:template match="/">
      <doc-status>
         <my:apply-templates mode="constraint1"/>
         <my:apply-templates mode="constraint2"/>
         <my:apply-templates mode="constraint3"/>
         <my:apply-templates mode="constraint4"/>
      </doc-status>
   </my:template><!--

  ........................NEW CONSTRAINT........................
  -->
   <my:template mode="constraint1" match="/reservations/room">
      <my:if test="not(@floor <= 12)">
         <err-message>
            The floor number <my:value-of select="@floor"/> does not exist.
        </err-message>
      </my:if>
   </my:template>
   <my:template match="text()" priority="-1" mode="constraint1"/><!--

  ........................NEW CONSTRAINT........................
  -->
   <my:template mode="constraint2" match="//room/events/event/date">
      <my:if test="not(dateb/@value <= datef/@value)">
         <err-message>
            The final date: <my:value-of select="datef"/> occurs before the beginning
            date: <my:value-of select="dateb"/> -this is not allowed.
        </err-message>
      </my:if>
   </my:template>
   <my:template match="text()" priority="-1" mode="constraint2"/><!--

  ........................NEW CONSTRAINT........................
  -->
   <my:template mode="constraint3" match="//compc">
      <my:if test="not(string-length(number(.)) = 9 and (substring(.,1,1)=2 or substring
(.,1,2)=91
          or substring(.,1,2)=93 or substring(.,1,2)=96))">
         <err-message>
            The contact for the company <my:value-of select="../compn"/> is not a
            valid phone number.
        </err-message>
      </my:if>
   </my:template>
   <my:template match="text()" priority="-1" mode="constraint3"/><!--

  .........................NEW CONSTRAINT.........................
  -->
   <my:template mode="constraint4" match="//title/compn">
      <my:variable name="keycompn">
         <my:value-of select="."/>
      </my:variable>
      <my:if test="not((count(../../companies/company[compn=$keycompn]) >= 1))">
         <err-message>
        The title of the event must not contain any company's name outside the set
        of organizer companies, as <my:value-of select="."/> in a reservation of the room
        <my:value-of select="../../../../@id"/>.
      </err-message>
      </my:if>
   </my:template>
   <my:template match="text()" priority="-1" mode="constraint4"/>
   <my:template match="text()" priority="-1"/>
</my:stylesheet>
```

The Constraints One at a Time

Constraint 1

The value of every `floor` attribute descendant of the `room` elements must be 12 or less.

The XCSL code needed to validate this condition is:



```
<constraint>
    <selector selexp="/reservations/room"/>
    <cc>
        @floor <= 12
    </cc>
    <action>
        <message>
        The floor number <value selexp="@floor"/> does not exist.
    </message>
    </action>
</constraint>
```

We check that the value of the `floor` attribute of each `/reservations/room` is less than or equal to 12. We set the context to `/reservations/room` analyzing every occurrence of this. The `message` element is used to specify the output we want to return to the user every time the condition does not hold: for this constraint, we display the number of the floor that triggers the error message.

Constraint 2

For each event, the final date must occur after the beginning date.

The XCSL code needed to validate this condition is:

```
<constraint>
    <selector selexp="//room/events/event/date"/>
    <cc>
        dateb/@value <= datef/@value
    </cc>
    <action>
        <message>
            The final date: <value selexp="datef"/> occurs before the beginning date:
            <value selexp="dateb"/> -this is not allowed.
        </message>
    </action>
</constraint>
```

We set the context to //room/events/event/date, to analyze every occurrence of date. Then we check if dateb is previous to datef.

Constraint 3

Every contact must have a valid format - only numbers, in number of nine, and begin either with a 2, 91, 93 or 96.

The XCSL code needed to validate this condition is:

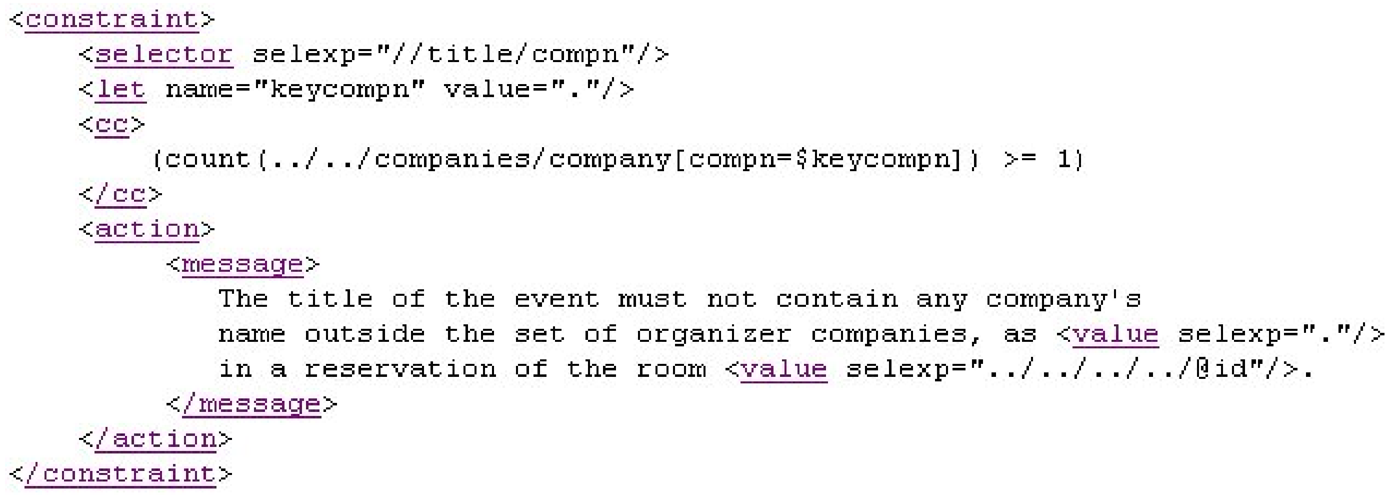
```
<constraint>
    <selector selexp="//compc"/>
    <cc>
        string-length(number(.)) = 9 and (substring(.,1,1)=2 or substring(.,1,2)=91 or
        substring(.,1,2)=93 or substring(.,1,2)=96)
    </cc>
    <action>
        <message>
            The contact for the company <value selexp="../compn"/> is not a valid phone
            number.
        </message>
    </action>
</constraint>
```

The context is now set to //compc. We verify that the contact is only made up of digits by checking that the length of the number obtained after the application of the number function -which returns only the digits, for instance number(232s)=232- is 9. Afterwards we check if the beginning substring of the contact is either equal to 2, 91, 93 or 96. Finally we make the interception of these two Boolean values. The return message includes the name of the company which contact has a wrong format.

Constraint 4

Every compn element that occurs inside a certain `title` element must be of an organizer company.

The XCSL code needed to validate this condition is:



```
<constraint>
    <selector selexp="//title/compn"/>
    <let name="keycompn" value="."/>
    <cc>
        (count(../../companies/company[compn=$keycompn]) >= 1)
    </cc>
    <action>
         <message>
            The title of the event must not contain any company's
            name outside the set of organizer companies, as <value selexp="."/>
            in a reservation of the room <value selexp="../../../../@id"/>.
         </message>
    </action>
</constraint>
```

We use a `let` element to place in keycompn the list of values that the compn element assumes in the // `title`/compn context. Later, the number of occurrences of each instance of compn (in companies/company descendant of the current's title event) in the keycompn list is counted and compared with 1. The action will be triggered whenever the negation of "greater than or equal" occurs, i.e., the name does not occur as an organizer of the event.

Applications

Application 1

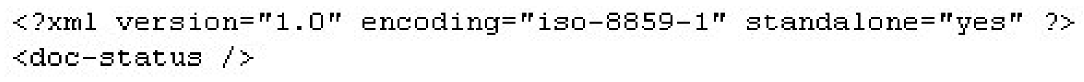
Documents where the value of every `floor` attribute descendant of the room elements is 12 or less; for each event, the `final date` occurs after the beginning `date`; every contact has a valid format; and every compn element that occurs inside a certain `title` element refers an organizer company.

One possible XML instance is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE reservations SYSTEM "reserv.dtd">
<reservations>
    <room floor="0" id="s1">
        <gicharge>
            <picharge>Ana</picharge>
            <picharge>Manuel</picharge>
        </gicharge>
        <events>
            <event>
                <date>
                    <dateb value="20010515">15th May 2001</dateb>
                    <datef value="20010515">15th May 2001</datef>
                </date>
                <companies>
                    <company>
                        <compn>Candle</compn>
                        <organizer>Gabriel</organizer>
                        <compc>214848737</compc>
                    </company>
                </companies>
                <title>
                    <compn>Candle</compn>Net</title>
            </event>
            <event>
                <date>
                    <dateb value="20010626">26th June 2001</dateb>
                    <datef value="20010626">26th June 2001</datef>
                </date>
                <companies>
                    <company>
                        <compn>EMC</compn>
                        <organizer>Flavio</organizer>
                        <compc>219458372</compc>
                    </company>
                </companies>
                <title>
                    <compn>EMC</compn> developments for <tradem>OS/390</tradem>
                </title>
            </event>
        </events>
    </room>
    <room floor="1" id="s2">
        <gicharge>
            <picharge>Ana</picharge>
            <picharge>Manuel</picharge>
        </gicharge>
        <events/>
    </room>
    <room floor="1" id="s3">
        <gicharge>
            <picharge>José</picharge>
            <picharge>Rita</picharge>
        </gicharge>
        <events>
            <event>
                <date>
                    <dateb value="20010524">24th May 2001</dateb>
                    <datef value="20010525">25th May 2001</datef>
                </date>
```

```
                <companies>
                    <company>
                        <compn>SOL-S</compn>
                        <organizer>Maria</organizer>
                        <compc>213487659</compc>
                    </company>
                    <company>
                        <compn>CheckPoint</compn>
                        <organizer>Carlos</organizer>
                        <compc>224357985</compc>
                    </company>
                    <company>
                        <compn>Remedy</compn>
                        <organizer>Bruno</organizer>
                        <compc>218705464</compc>
                    </company>
                </companies>
                <title>EBusiness2000 - <compn>CheckPoint</compn> and
                 <compn>Remedy</compn>
                </title>
            </event>
        </events>
    </room>
</reservations>
```

The result of the semantic validation was a document with no errors (for instance while using Saxon, we would have to write `saxon.exe instancedoc.xml constraintdoc.xsl > errordoc.xml`):



```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes" ?>
<doc-status />
```

Application 2:

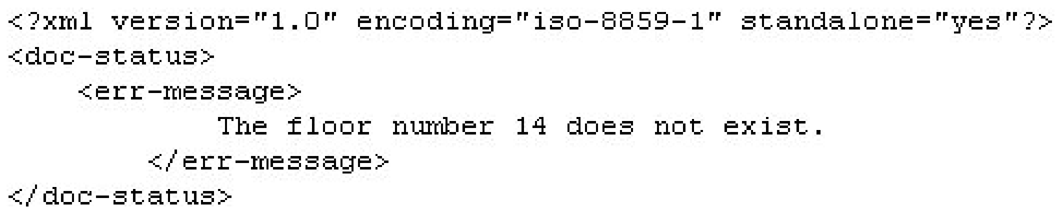
Documents where **the value of one or more floor attributes descendant of the room elements is greater than 12**; for each event, the final date occurs after the beginning date; every contact has a valid format; and every compn element that occurs inside a certain title element refers to an organizer company.

One possible XML instance is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE reservations SYSTEM "reserv.dtd">
<reservations>
    <room floor="14" id="s1">
        <gicharge>
            <picharge>Ana</picharge>
            <picharge>Manuel</picharge>
        </gicharge>
        <events>
```

```
            <event>
                <date>
                    <dateb value="20010626">26th June 2001</dateb>
                    <datef value="20010626">26th June 2001</datef>
                </date>
                <companies>
                    <company>
                        <compn>EMC</compn>
                        <organizer>Flavio</organizer>
                        <compc>219458372</compc>
                    </company>
                </companies>
                <title>
                    <compn>EMC</compn> developments
                </title>
            </event>
        </events>
    </room>
    <room floor="1" id="s3">
        <gicharge>
            <picharge>José</picharge>
            <picharge>Bonifácio</picharge>
        </gicharge>
        <events>
            <event>
                <date>
                    <dateb value="20010524">24th May 2001</dateb>
                    <datef value="20010525">25th May 2001</datef>
                </date>
                <companies>
                    <company>
                        <compn>Remedy</compn>
                        <organizer>Bruno</organizer>
                        <compc>218705464</compc>
                    </company>
                </companies>
                <title>EBusiness2000 - <compn>Remedy</compn> products
                </title>
            </event>
        </events>
    </room>
</reservations>
```

In this case we have two `room` elements and one is said to be located in the 14th floor. The result of the semantic validation will be a document showing the error:



```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<doc-status>
    <err-message>
            The floor number 14 does not exist.
        </err-message>
</doc-status>
```

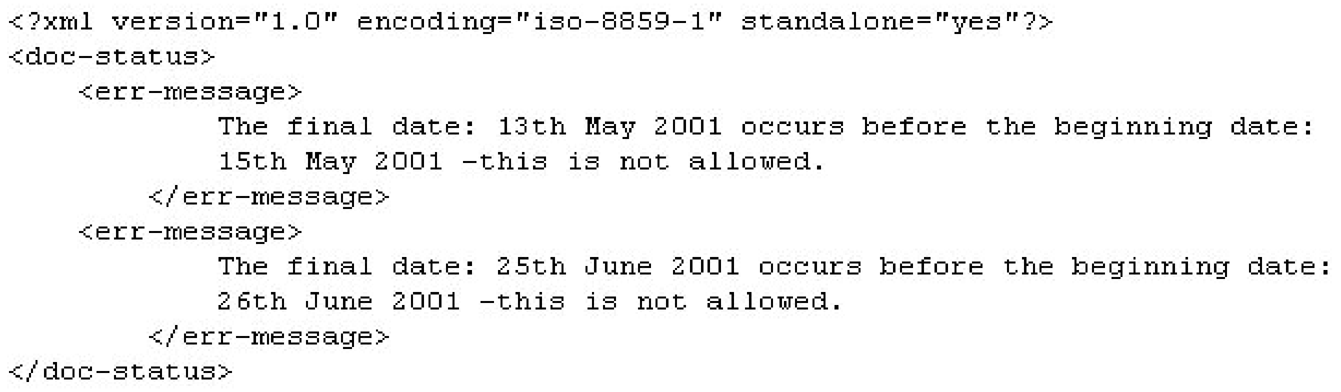
Application 3:

Documents where the value of every `floor` attribute descendant of the `room` elements is 12 or less; **for one or more events, the `final date` occurs before the `beginning date`**; every contact has a valid format; and every `compn` element that occurs inside a certain `title` element refers an organizer company.

One possible XML instance is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE reservations SYSTEM "reserv.dtd">
<reservations>
    <room floor="0" id="s1">
        <gicharge>
            <picharge>Ana</picharge>
            <picharge>Manuel</picharge>
        </gicharge>
        <events>
            <event>
                <date>
                    <dateb value="20010515">15th May 2001</dateb>
                    <datef value="20010513">13th May 2001</datef>
                </date>
                <companies>
                    <company>
                        <compn>Promosoft</compn>
                        <organizer>Carlos</organizer>
                        <compc>219878586</compc>
                    </company>
                </companies>
                <title>
                    Portal/SME</title>
            </event>
            <event>
                <date>
                    <dateb value="20010626">26th June 2001</dateb>
                    <datef value="20010625">25th June 2001</datef>
                </date>
                <companies>
                    <company>
                        <compn>EMC</compn>
                        <organizer>Flavio</organizer>
                        <compc>219458372</compc>
                    </company>
                </companies>
                <title>
                    <tradem>OS/390</tradem>
                </title>
            </event>
        </events>
    </room>
</reservations>
```

In this case we have two event elements and both have wrong dates. The result of the semantic validation will be a document showing the errors:



```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<doc-status>
    <err-message>
            The final date: 13th May 2001 occurs before the beginning date:
            15th May 2001 -this is not allowed.
        </err-message>
    <err-message>
            The final date: 25th June 2001 occurs before the beginning date:
            26th June 2001 -this is not allowed.
        </err-message>
</doc-status>
```

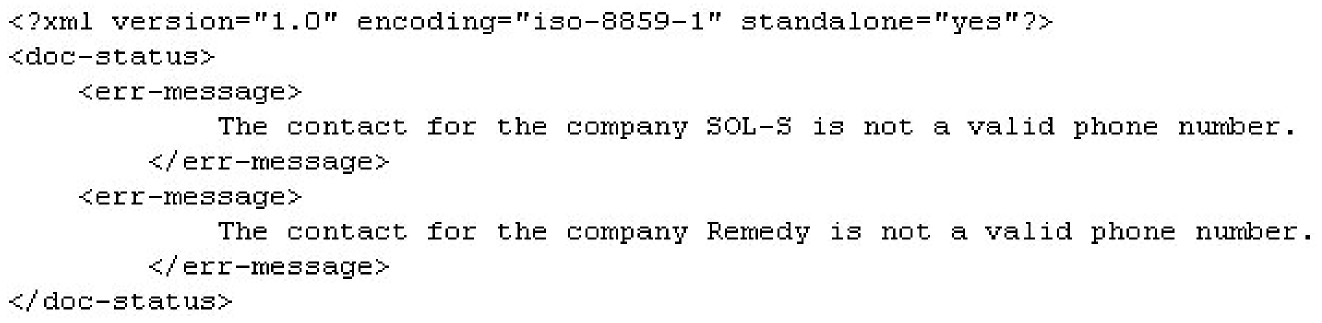
Application 4:

Documents where the value of every `floor` attribute descendant of the `room` elements is 12 or less; for each event, the `final date` occurs after the `beginning date`; **one or more contacts have an invalid format**; and every `compn` element that occurs inside a certain `title` element refers an organizer company.

One possible XML instance is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE reservations SYSTEM "reserv.dtd">
<reservations>
    <room floor="1" id="s3">
        <gicharge>
            <picharge>José</picharge>
            <picharge>Bonifácio</picharge>
        </gicharge>
        <events>
            <event>
                <date>
                    <dateb value="20010524">24th May 2001</dateb>
                    <datef value="20010525">25th May 2001</datef>
                </date>
                <companies>
                    <company>
                        <compn>SOL-S</compn>
                        <organizer>Maria</organizer>
                        <compc>413487659</compc>
                    </company>
                    <company>
                        <compn>CheckPoint</compn>
                        <organizer>Carlos</organizer>
                        <compc>224357985</compc>
                    </company>
                    <company>
                        <compn>Remedy</compn>
                        <organizer>Bruno</organizer>
                        <compc>818705464</compc>
                    </company>
                </companies>
                <title>EBusiness2000 - <compn>CheckPoint</compn>
                and <compn>Remedy</compn>
                </title>
            </event>
        </events>
    </room>
</reservations>
```

In this XML instance we have three company elements and two of them have an invalid format. The result of the semantic validation will be a document showing the errors:



```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<doc-status>
    <err-message>
            The contact for the company SOL-S is not a valid phone number.
        </err-message>
    <err-message>
            The contact for the company Remedy is not a valid phone number.
        </err-message>
</doc-status>
```

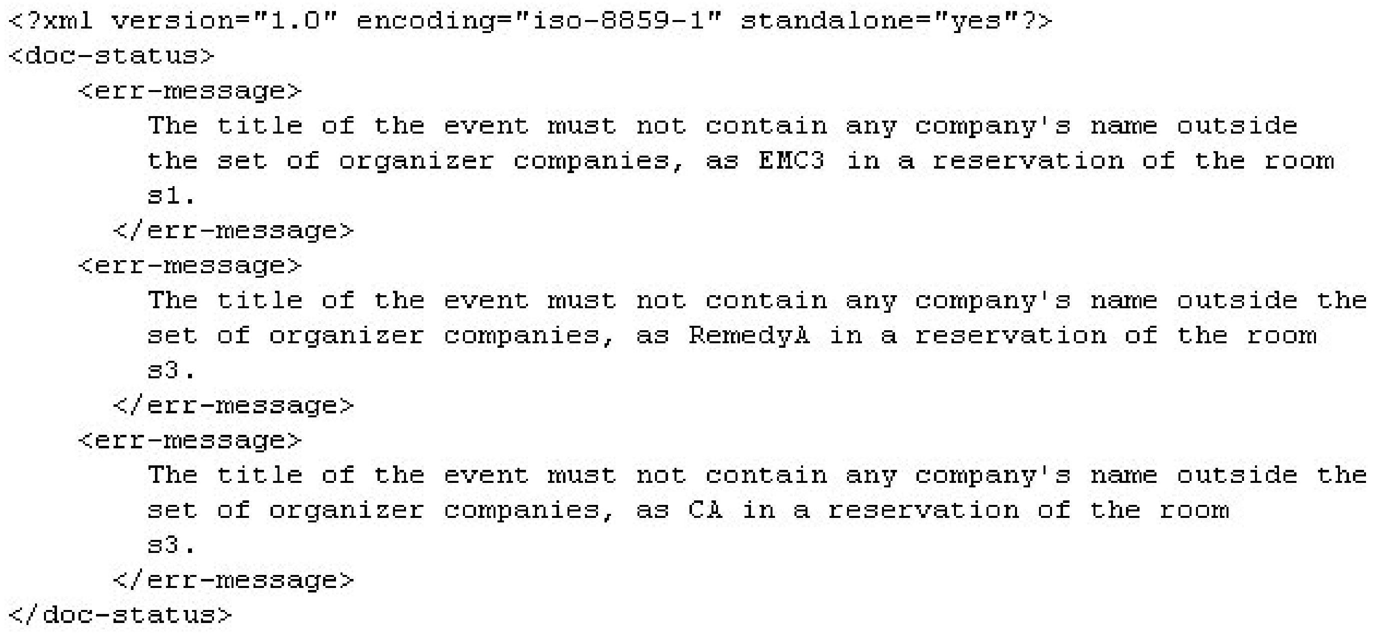
Application 5:

Documents where the value of every floor attribute descendant of the room elements is 12 or less; for each event, the final date occurs after the beginning date; every contact has a valid format; and **one or more compn elements occurring inside a certain title element refer to a non-organizer company**.

One possible XML instance is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE reservations SYSTEM "reserv.dtd">
<reservations>
    <room floor="0" id="s1">
        <gicharge>
            <picharge>Ana</picharge>
            <picharge>Manuel</picharge>
        </gicharge>
        <events>
            <event>
                <date>
                    <dateb value="20010626">26th June 2001</dateb>
                    <datef value="20010626">26th June 2001</datef>
                </date>
                <companies>
                    <company>
                        <compn>EMC</compn>
                        <organizer>Flavio</organizer>
                        <compc>219458372</compc>
                    </company>
                </companies>
                <title>
                    <compn>EMC3</compn> developments
                </title>
            </event>
        </events>
    </room>
    <room floor="1" id="s3">
        <gicharge>
            <picharge>José</picharge>
            <picharge>Bonifácio</picharge>
        </gicharge>
        <events>
            <event>
                <date>
                    <dateb value="20010524">24th May 2001</dateb>
                    <datef value="20010525">25th May 2001</datef>
                </date>
                <companies>
                    <company>
                        <compn>SOL-S</compn>
                        <organizer>Coimbra</organizer>
                        <compc>213487659</compc>
                    </company>
                    <company>
                        <compn>CheckPoint</compn>
                        <organizer>Esteves</organizer>
                        <compc>224357985</compc>
                    </company>
                    <company>
                        <compn>Remedy</compn>
                        <organizer>Botas</organizer>
                        <compc>218705464</compc>
                    </company>
                </companies>
                <title>EBusiness2000 - <compn>CheckPoint</compn> and
                <compn>RemedyA</compn> and <compn>CA</compn>
                </title>
            </event>
        </events>
    </room>
</reservations>
```

In this case we have two events, in the first one's `title` the only compn that occurs refers to a non-organizer company, and in the second one's `title` the second and third compn that occur refer to non-organizer companies. The result of the semantic validation will be a document showing the errors:



```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<doc-status>
    <err-message>
        The title of the event must not contain any company's name outside
        the set of organizer companies, as EMC3 in a reservation of the room
        s1.
      </err-message>
    <err-message>
        The title of the event must not contain any company's name outside the
        set of organizer companies, as RemedyA in a reservation of the room
        s3.
      </err-message>
    <err-message>
        The title of the event must not contain any company's name outside the
        set of organizer companies, as CA in a reservation of the room
        s3.
      </err-message>
</doc-status>
```

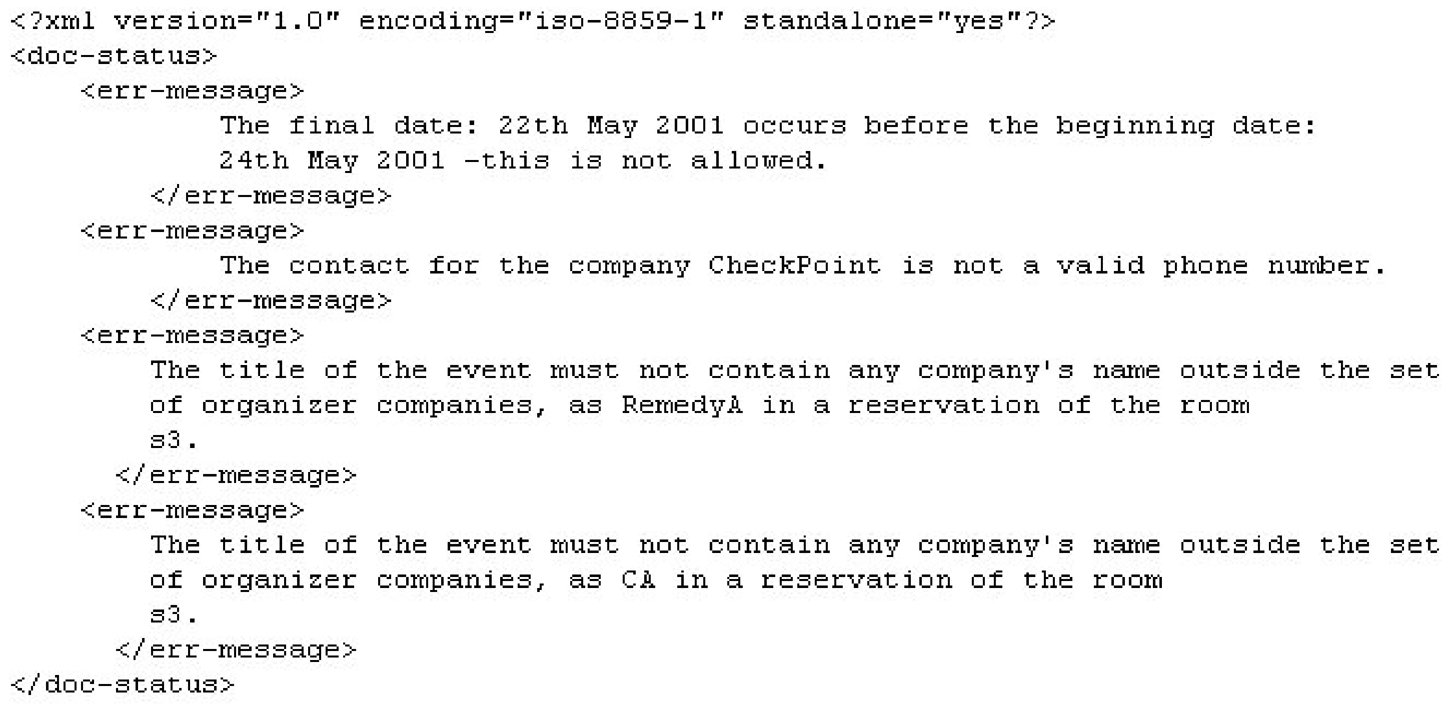
Application 6:

Documents where a combination of the previous four situations occur, i.e., one or more conditions are violated.

One possible XML instance is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE reservations SYSTEM "reserv.dtd">
<reservations>
    <room floor="1" id="s3">
        <gicharge>
            <picharge>José</picharge>
            <picharge>Bonifácio</picharge>
        </gicharge>
        <events>
            <event>
                <date>
                    <dateb value="20010524">24th May 2001</dateb>
                    <datef value="20010522">22th May 2001</datef>
                </date>
                <companies>
                    <company>
                        <compn>CheckPoint</compn>
                        <organizer>Carlos</organizer>
                        <compc>824357985</compc>
                    </company>
                    <company>
                        <compn>Remedy</compn>
                        <organizer>Bruno</organizer>
                        <compc>218705464</compc>
                    </company>
                </companies>
                <title>EBusiness2000 - <compn>CheckPoint</compn> and
                <compn>RemedyA</compn> and <compn>CA</compn>
                </title>
            </event>
        </events>
    </room>
</reservations>
```

The particular instance shown before has one event, the `final date` occurs before the beginning `date`, one contact does not have the allowed form and two of the compn elements which occur in the `title` are of non-organizer companies. The result of the semantic validation will be a document showing the errors:



```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<doc-status>
    <err-message>
            The final date: 22th May 2001 occurs before the beginning date:
            24th May 2001 -this is not allowed.
        </err-message>
    <err-message>
            The contact for the company CheckPoint is not a valid phone number.
        </err-message>
    <err-message>
        The title of the event must not contain any company's name outside the set
        of organizer companies, as RemedyA in a reservation of the room
        s3.
      </err-message>
    <err-message>
        The title of the event must not contain any company's name outside the set
        of organizer companies, as CA in a reservation of the room
        s3.
      </err-message>
</doc-status>
```

Application 7:

Documents where, simultaneously, the value of one or more `floor` attributes descendant of the room elements is greater than 12; for one or more events, the `final date` occurs before the beginning `date`; one or more contacts have an invalid format; and one or more compn elements occurring inside a certain title element refer to a non-organizer company.

One possible XML instance is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE reservations SYSTEM "reserv.dtd">
<reservations>
    <room floor="17" id="s3">
        <gicharge>
            <picharge>José</picharge>
            <picharge>Rita</picharge>
        </gicharge>
        <events>
            <event>
                <date>
                    <dateb value="20010524">24th May 2001</dateb>
                    <datef value="20010522">22nd May 2001</datef>
                </date>
                <companies>
                    <company>
                        <compn>CheckPoint</compn>
                        <organizer>Esteves</organizer>
                        <compc>824357985</compc>
                    </company>
                </companies>
                <title>EBusiness2000 - <compn>CheckPoint</compn> and
                <compn>CA</compn>
                </title>
            </event>
        </events>
    </room>
</reservations>
```

In this case we have one room indicated to be in the 14th floor, where only one event with the final date before the beginning date occurs, one contact with a disallowed form and one compn element occurring in the title being of a non-organizer company. The result of the semantic validation will be a document showing the errors:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<doc-status>
    <err-message>
            The floor number 17 does not exist.
        </err-message>
    <err-message>
            The final date: 22nd May 2001 occurs before the beginning date:
            24th May 2001 -this is not allowed.
        </err-message>
    <err-message>
            The contact for the company CheckPoint is not a valid phone number.
        </err-message>
    <err-message>
        The title of the event must not contain any company's name outside the
        set of organizer companies, as CA in a reservation of the room
        s3.
      </err-message>
</doc-status>
```

Language of the Validation Output

If we want to have several possible output languages and only one constraint document, we have to use the lang attribute of the XCSL's message element. This happens, for instance, when we need to provide different outputs for different mother tongue users. Imagine a chain of hotels with branches in England and Portugal. It would be great to provide an English output for the hotels in England and a Portuguese output for the other ones. Therefore, for each constraint, each action element will have two message sub-elements instead of one. The first one, apart from having the lang attribute with the value "en", will be exactly as we wrote previously. The second one will be in Portuguese and the lang attribute will be "pt".

For instance for the first constraint, we will have:

```
<message lang="en">
    The floor number <value selexp="@floor"/> does not exist.
</message>
```

and, for the Portuguese version:

```
<message lang="pt">
  O andar <value selexp="@floor"/> não existe.
</message>
```

Notice that we refer to the document's elements exactly the way, only the free text is translated into Portuguese. This way, the complete constraint is:

```
<constraint>
    <selector selexp="/reservations/room"/>
    <cc>
          @floor <= 12
    </cc>
    <action>
       <message lang="en">
            The floor number <value selexp="@floor"/> does not exist.
       </message>
       <message lang="pt">
            O andar <value selexp="@floor"/> não existe.
       </message>
    </action>
</constraint>
```

While using this languages option, we may either invoke saxon the same way we did,

```
saxon.exe constraintdoc.xml XCSL.xsl > constraintdoc.xsl

    saxon.exe instancedoc.xml constraintdoc.xsl > errordoc.xml
```

which will return the error message in the default language, i.e. English:

```
<err-message>
  The floor number 14 does not exist.
</err-message>
```

Or directly specify the output language, for instance for the Portuguese messages:

```
saxon.exe constraintdoc.xml XCSL.xsl > constraintdoc.xsl lang=pt

    saxon.exe instancedoc.xml constraintdoc.xsl > errordoc.xml
```

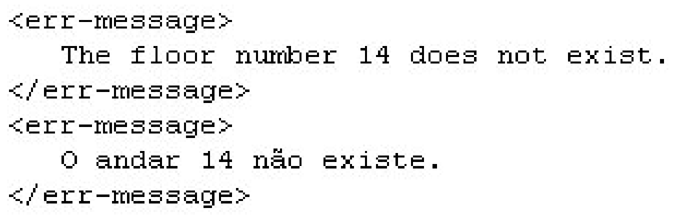
which will produce the following output:

```
<err-message>
   O andar 14 não existe.
</err-message>
```

The last option is to produce messages for all the available languages at the same time, for instance for debugging purposes. This is done by specifying "all" in the `lang` option:

```
saxon.exe constraintdoc.xml XCSL.xsl > constraintdoc.xsl lang=all

    saxon.exe instancedoc.xml constraintdoc.xsl > errordoc.xml
```

The result will be, as expected:



```
<err-message>
   The floor number 14 does not exist.
</err-message>
<err-message>
   O andar 14 não existe.
</err-message>
```

The XCSL Language

The first version of the XML Constraint Specification Language is formally defined in [6]. This exercise of formalization helped us to find the core structure of the language.

A specification in XCSL is composed by one or more tuples. Each tuple has three parts [7]:

- **Context Selector**: the expression that selects the context where we want to enforce the constraint.
- **Context Condition**: the condition we want to enforce.
- **Action**: the action we want to trigger every time the condition does not hold.

In a more formal notation we can write:

```
ConstraintSpec = Constraint+
Constraint = (ContextSelector, ContextCondition, Action)
```

We could use a grammar to define the language, but as we stated before, we decided to use XSLT [9] to specify the constraints; in order to be coherent, we needed an XML wrapper for the XSLT expressions (like in XSL). So, each XCSL specification is defined as an XML instance and the XCSL language is defined by a DTD (or an XML-Schema); the present version of the DTD that specifies XCSL (named xcsl.dtd) is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- XCSL: XML Constraint Specification Language -->
<!ELEMENT cs (constraint)+>
<!ATTLIST cs
    dtd CDATA #IMPLIED
    date CDATA #IMPLIED
    version CDATA #IMPLIED>
<!ELEMENT constraint (selector,let*,cc,action)>
<!ELEMENT selector EMPTY>
<!ATTLIST selector
    selexp CDATA #REQUIRED>
<!ELEMENT let EMPTY>
<!ATTLIST let
    name CDATA #REQUIRED
    value CDATA #REQUIRED>
<!ELEMENT cc (#PCDATA|variable)*>
<!ELEMENT variable EMPTY>
<!ATTLIST variable
    selexp CDATA #REQUIRED>
<!ELEMENT action (message*)>
<!ELEMENT message (#PCDATA|value)*>
<!ATTLIST message
    lang CDATA #IMPLIED>
<!ELEMENT value EMPTY>
<!ATTLIST value
    selexp CDATA #REQUIRED>
```

XML-Schemas are surpassing DTDs as the common mode of defining classes for XML instances. However, as XML-Schemas are more verbose than equivalent DTDs, we include here the DTD and include a diagrammatic description of the XCSL XML-Schema in **Figure 2** for clarity.

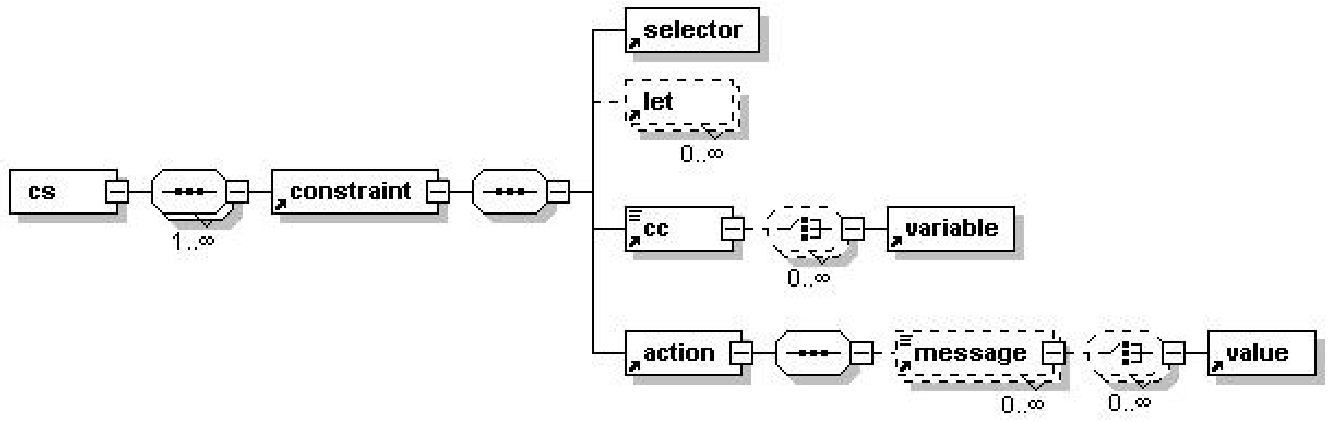


Figure 2: XCSL version 1.0 XML-Schema diagram

Elements

An XCSL constraint document consists of one or more `constraint` elements. Each `constraint` element specifies one constraint and the action that will be triggered when this constraint is not respected. The following table summarizes all the XCSL elements.

Element	**cs**

Description	Is the root element of an XML Constraint Specification Document. An XCSL constraint document consists of one or more **constraint** elements.
Required	true

Attributes

dtd	name of the DTD that is to be associated with the constraints being specified;	Required: false
date	XCSL document's date;	Required: false
version	XCSL document's version.	Required: false

Children	**constraint**
Parent	**/**

DTD

```
<!ELEMENT cs (constraint)+>
<!ATTLIST cs
      dtd CDATA #IMPLIED
      date CDATA #IMPLIED
      version CDATA #IMPLIED>
```

Element	**constraint**
Description	This element specifies a constraint and the action that will be triggered when this constraint is not respected. This element is formed by a sequence of elements: a **selector** element, zero or more **let** elements, a **cc** element, and an **action** element.
Required	true
Children	**selector** **let** **cc** **action**
Parent	**cs**

DTD

```
<!ELEMENT constraint (selector,let*,cc,action)>
```

Element	**selector**
Description	This element, as the name suggests, selects the context (the element or elements) within which the conditions should be tested.
Required	true

Attributes	selexp	this attribute holds an XPath (XML Path Language) expression that performs the selection.	Required: true
Parent	**constraint**		
DTD	`<!ELEMENT selector EMPTY>`		

Element	**let**		
Description	This element has an important role in complex constraints, it allows us to specify multi-step evaluations, enabling the simplification of that kind of constraints. With this element we can save, in a variable (name) the result of an XPath expression applied to any XPath path, or still the set of values that belong to that context. It is an empty element.		
Required	false		
Attributes	name	it specifies the variable's name;	Required: true
	value	holds an XPath expression applied to the context or to any XPath path.	Required: true
Parent	**constraint**		

DTD	`<!ELEMENT let EMPTY>`
	`<!ATTLIST let`
	`      name CDATA #REQUIRED`
	`      value CDATA #REQUIRED>`		

Element	**cc**
Description	It is an element that specifies the constraint that has to be verified - regular expression or XPath function. The action will be triggered whenever that expression or function is evaluated to false. It has a mixed content - text in which **variable** elements can occur any number of times.
Required	true
Children	**variable**
Parent	**constraint**
DTD	`<!ELEMENT cc (#PCDATA|variable)*>`

Element	**variable**
Description	This element is used when we are enforcing a constraint over an element or attribute and we want to guarantee that it will be evaluated only if that element or attribute occurs in the document instance. If the element or attribute in question is absent from the document, the **constraint** will not be evaluated.
Required	false
Attributes	selexp | it consists in the XPath path to that element or elements. | Required: true
Parent	**cc**
DTD	`<!ELEMENT variable EMPTY>` `<!ATTLIST variable` `selexp CDATA #REQUIRED>`

```
<!ELEMENT variable EMPTY>
<!ATTLIST variable
      selexp CDATA #REQUIRED>
```

Element	**action**
Description	This is a required element that returns messages or results of evaluated expressions when the constraint being evaluated results in a false value. It consists of a sequence of messages. The contents of the **message** element is returned when the **action** element is unchained. It has mixed content: text with **value** elements.
Required	true
Children	**message**
Parent	**constraint**
DTD	`<!ELEMENT action (message*)>`

Element	**message**
Description	The content of this element is returned when the **action** element is unchained. It has mixed content: text with **value** element.
Required	false
Attributes	lang | Language for which the message shall be displayed. | Required: false
Children	**value**

Parent	**action**
DTD	`<!ELEMENT message (#PCDATA|value)*>`

Element	**value**
Description	This element is used to include some value of elements or attributes in the messages.
Required	false
Attributes	selexp | consists in a XPath path to the element or attribute which value we want to show. | Required: true
Parent	**message**
DTD	`<!ELEMENT value EMPTY>` `<!ATTLIST value` `selexp CDATA #REQUIRED>`

The XCSL Processor Generator

The XCSL processor generator is the main piece in our architecture. It takes an XML instance, written according to the XCSL language, and generates an XSL stylesheet that will test the specified constraints when processed by a standard XSL processor like Saxon or Xalan. The first versions of the generator were coded in Perl, using an XML down-translation module called XML::DT [8]. We chose Perl because it is open and has strong text processing capabilities. XML::DT is a Perl module developed to process transformations over XML documents. It has some specific built-in operators and functions, but we can still use all the functionalities available in Perl. During the development of this generator we found some problems that had a strong impact in the final algorithm. The most important were:

- Optional or non-filled elements - suppose you have specified a constraint for every element named X; suppose that element X is optional and in certain parts of the instance the user did not insert it; the system will trigger the action for every nonexistent element X (in XSL the absence of an element that is part of a condition will make that condition always false).
- Ambiguity in context selection - until now, we have just said that an XCSL specification is composed by a set of constraints; we did not say that these constraints should be disjoint in terms of context; in some cases there is a certain overlap between the contexts of different conditions; this overlap will cause an error when transposed to XSL; XSL processors can only match one context at a time; there is one solution to this problem that is to run each constraint in a different mode (in XSL each mode corresponds to a different traversal of the document tree).

After several development iterations, we converged on the following process:

1. Convert each `constraint` element into an `xsl:template`

2. Use attribute `selexp` for the `xsl:template`'s `match` attribute
3. Convert each "stamped" path (`variable` element) into a predicate [...], inside match attribute
4. Convert each `let` element into an `xsl:variable`
5. Convert `cc` element into an `xsl:if` element (the `test` attribute of the `xsl:if` element is filled with the negation of `cc`'s content)
6. Put `message` contents inside the template body converting each `value` element into an `xsl:value-of` element
7. Filter all remaining text nodes

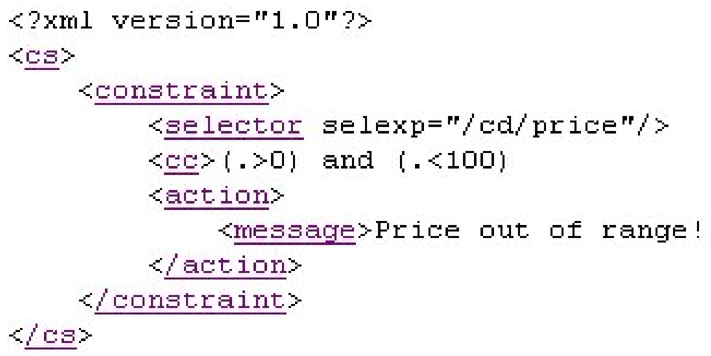
Recently, we have been developing the XSL version of the generator. The main function, which generates a template for each constraint, looks like the following:

```
<xsl:template match="constraint">
    <xsl:variable name="cc">
        <xsl:apply-templates select="cc"/>
    </xsl:variable>
    <xsl:variable name="sel" select="selector/@selexp"/>
    <xsl:variable name="pred">
        <xsl:apply-templates select="cc/variable" mode="pred"/>
    <xsl:variable>
    <xsl:comment>
        .....................NEW CONSTRAINT.....................
    </xsl:comment>
    <my:template mode="constraint{count(preceding-sibling::*)+1}"
    match="{$sel}{$pred}">
        <my:if test="not({$cc})">
            <err-message>
                <xsl:apply-templates select="action"/>
            </err-message>
        </my:if>
    </my:template>
    <my:template match="text()" priority="-1"
    mode="constraint{count(preceding-sibling::*)+1}">
        <!-- strip characters -->
    </my:template>
</xsl:template>
```

Let us now go through a very simple example. Consider the following record:

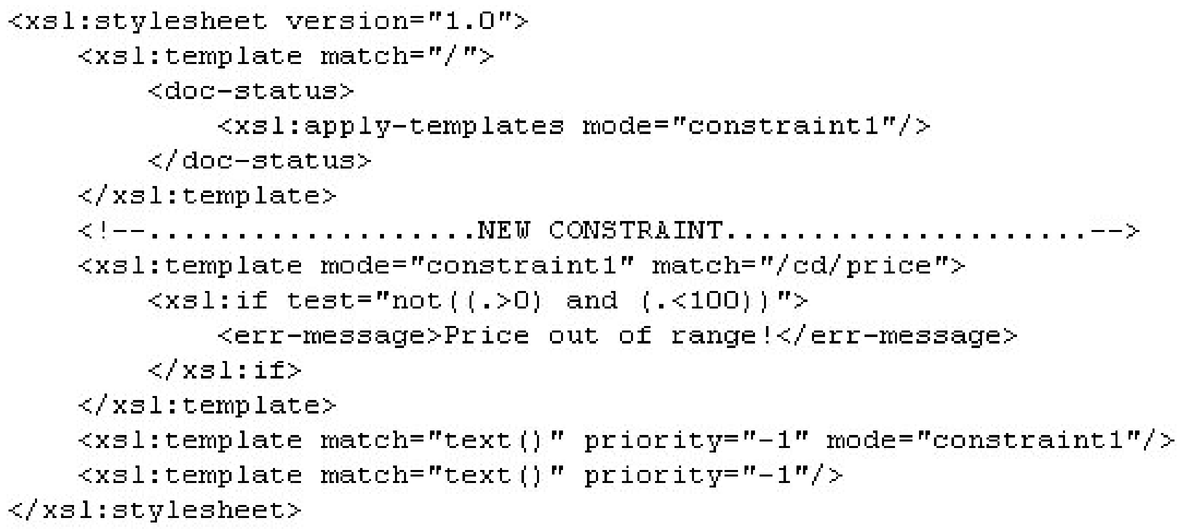
```
<?xml version="1.0"?>
<cd>
    ...
    <price>32.00
    ...
</cd>
```

We want to ensure that every cd's price is within a certain range (for instance from 0 to 100). In order to do that, we specify the following constraint:



```
<?xml version="1.0"?>
<cs>
    <constraint>
        <selector selexp="/cd/price"/>
        <cc>(.>0) and (.<100)
        <action>
            <message>Price out of range!
        </action>
    </constraint>
</cs>
```

This document, when processed by our generator, will generate the following XSL stylesheet:



```
<xsl:stylesheet version="1.0">
    <xsl:template match="/">
        <doc-status>
            <xsl:apply-templates mode="constraint1"/>
        </doc-status>
    </xsl:template>
    <!--...................NEW CONSTRAINT.....................-->
    <xsl:template mode="constraint1" match="/cd/price">
        <xsl:if test="not((.>0) and (.<100))">
            <err-message>Price out of range!</err-message>
        </xsl:if>
    </xsl:template>
    <xsl:template match="text()" priority="-1" mode="constraint1"/>
    <xsl:template match="text()" priority="-1"/>
</xsl:stylesheet>
```

When applied to XML instances, this stylesheet will generate error messages whenever the constraint does not evaluate to true.

Conclusion

In this tutorial, we introduced XCSL (XML Constraint Specification Language) and applied it to a real-life case-study demonstrating the utility of XCSL. Here we have shown only one case-study. For other case-studies see [4]. By using XCSL, we can avoid the production of documents that are syntactically correct according to a DTD or schema, but that contain semantic problems. In the future, the development of XCSL will continue in three main lines:

1. The generation of Perl (XML::DT) instead of XSL. This would allow Perl actions and the use of Perl operators in context conditions making the language and the whole system more powerful.
2. The reverse engineering of the whole system; trying to find a suitable abstract representation for these constraints and the whole model (probably High Order Attribute Grammars).
3. To upgrade the whole system, the language and the processor, to XSL 2.0 including new features like the use of quantifiers inside constraints.

References

1

Dodds L. *Schematron: Validating XML Using XSLT.* XSLT UK Conference, Keble College, Oxford, England, 2001.

2
Duckett J., Griffin, O., Mohr, S., Norton, F., Stokes-Rees, I., Willians, K., Cagle, K., Ozu, N., and Tennison, J. *Professional XML Schemas. Wrox Press, 2001.*

3
Harold, E.R., and Means, W.S. *XML in a Nutshell.* O'Reilly & Associates, 2001.

4
Jacinto, M.H., Librelotto, G.R., Ramalho, J.C., and Henriques, P.R. *Constraint Specification Languages: comparing XCSL, Schematron and XML-Schemas.* In: *Proceedings of XML Europe'2002,* Barcelona, Spain, 2002.

5
Jelliffe, R. *XML & SGML cookbook.* Prentice Hall, 1998.

6
Ramalho, J.C. *Anotação Estrutural de Documentos e sua Semântica.* Universidade do Minho - Portugal. 2000.

7
Ramalho, J.C., and Henriques, P.R. *Constraining Content: Specification and Processing.* In: *Proceedings of XML Europe'2001,* Internationales Congress Centrum (ICC), Berlin, Germany, 2001.

8
Ramalho, J.C., and Almeida, J.J. *XML::DT - a Perl Down Translation Module.* In: *Proceedings of XML Europe'99,* Granada - Spain, 1999.

9
Robie, J., Lapp, J., and Sach, D. *XSL Transformations (XSLT) - version 1.0.* In http://www.w3.org/TR/1998/WD-xsl-19980818, 2000.

10
Simpson, J.E. *Just XML.* Prentice Hall, 1999.

Biographies

Marta Jacinto has a degree in Applied Mathematics and Computation, and is currently working for ITIJ - the Computer Department of the Ministry of Justice as Systems Engineer. As a researcher, she is working on her Master thesis under the subject "Semantic Validation of XML Documents."

Giovani Librelotto has a degree and a Master on Computer Science. He is currently researching XML and Topic Maps and is preparing his Ph.D. thesis.

J. C. Ramalho is an Auxiliary Professor at the Computer Science Department of the University of Minho. He has a Masters on "Compiler Construction" and a Ph.D. on the subject "Document Semantics and Processing." He has been managing several XML projects and consulting.

Pedro Henriques is an Associate Professor of Computer Science at University of Minho. His research and teaching activity has been concerned with programming in general - paradigms, specification formalisms and languages; in particular, his main interest is the development of language processors. He completed, some years ago, his Ph.D. at University of Minho in the area of Attribute Grammars; he is, now, the leader of the "Language Specification and Processing" group. The application of the "grammatical approach to problem solving" and the use of "parsing and semantic analysis technologies" in various problem domains (namely, document processing, information retrieval and data/text mining, and geographical information systems) are the present concerns of his academic work.