XML Representation of Text Specifications for Realization by SimpleNLG

Author: Christopher Howell, Agfa Healthcare Corporation

Author: Albert Gatt, University of Malta

Contents

Document history

29/03/2011 – Version 1 – first version of the documentation for the XMl wrappers for SimpleNLG

20/04/2011 – Version 2 – the early xml examples are somewhat out-of-date since the schema has slightly changed. The Document element is contained in a Request Element, contained in a NLGSpec top-level element. Regression testing capability documented. This uses a Recording element to store requests and results in the xml file.

25/05/2011 – Version 3 – final version of document, in preparation for integration of XML framework to the main code stream for simplenlg.

31/05/2011 – Version 4 – new section on naming conventions for wrapper classes

05/05/2014 – Version 5 – Revised description of regression testing.

Purpose

The SimpleNLG surface realiser produces actual natural language text given an abstract representation of document structure, sentence, phrase and lexical information of the text.

The input to the realiser is called a text specification. A text specification, together with its children (for example, SPhraseSpecs) can be expressed in XML, based on a predefined XML schema that mirrors the relevant parts of the internal structure of a simplenlg specification.

The xmlrealiser is a simplenlg component that takes the XML representation of a text specification, maps it and its children to simplenlg framework objects, invokes a Realiser, and outputs the realisation as text.

The xmlrealiser and the XML schema for text specification provide the following benefits:

- Separates simplenlg from applications using it with a well-defined interface.
- Enables NLG processing in an XML pipeline using XSLT.
- Enables use of automatic code generators of text specification wrapper classes in C++, C#, Java or Visual Basic, making it possible to process inputs and construct the right representations for simplenlg (using the wrapper classes for a specific programming language) and then pass these to simplenlg itself for realization. Thus it also:
- Creates the basis for a simplenlg web service.
- Defines a format for representing text specifications as strings.
- Infrastructure for regression testing

XML Text Specification as simplified abstraction of simplenlg classes.

To build a text specification, it is sufficient to use and understand only a subset of simplenlg. Everything needed is in the framework, phrasespec and features packages. For the xmlrealiser, the input is an XML string that represents a **DocumentElement**, with children that represent whatever linguistic structures simplenlg can handle. The XML string must conform to the xml schema, a version of which is now available with the simplenlg distribution (under *res/xml/RealiserSchema.xsd*). The schema is described below.

The full generality of **NLGElement** with feature name/value pairs is not exposed in the xml interface. For each xml-derived class, the useful features are available as properties.

The properties, and which types of element they attach to, were derived from an analysis of the javadoc for **Features.Feature**. For each feature, if "Created by" indicates the feature can be set by the user, then a property was created. The "Applies to" determines which classes have that property.

Each phrase type can have FrontModifiers, PreModifiers, Head, Complements, and PostModifers.

SPhraseSpec also has, for convenience, and to match the simplenlg class, **cuePhrase**, **subject**, and **verbPhrase**.

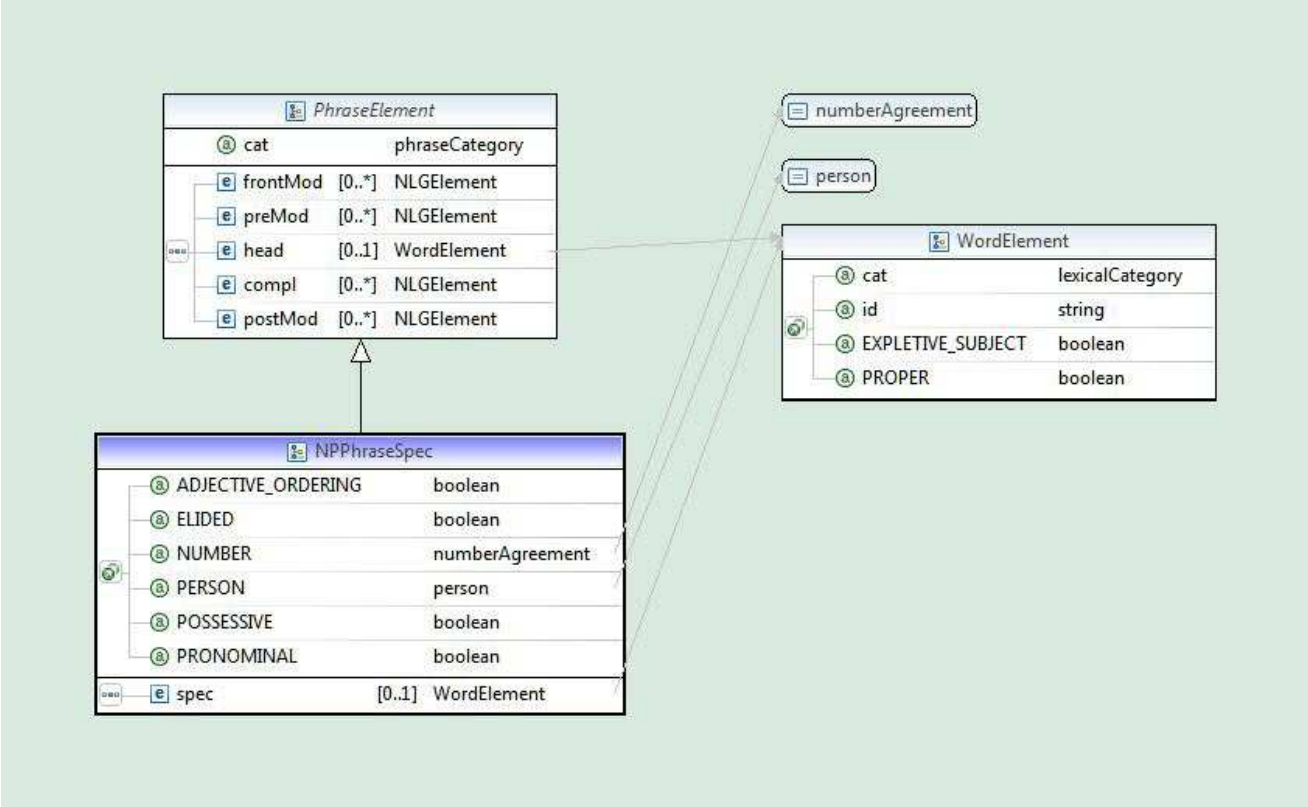
WordElement is used as phrase head, and in several other places. **StringElement** can be used wherever **PhraseElement** occurs.

The XML Schema

The xml schema is designed so the code classes generated from the schema by automatic code generators are actually useful, and easy to use. The classes are data transfer objects. You can use them to define a text specification for simplenlg, but you cannot actually perform any simplenlg operations with the data transfer objects.

The only XML elements that have text values are **WordElement** and **StringElement**, which are leaf nodes in the text specification tree. The higher level nodes are all xml elements with child nodes.

To see how the XML works, consider a **NPPhraseSpec**



As a **PhraseElement**, it has a **phraseCategory** property. The property name is **cat**, and the property value type is **phraseCategory**, which is defined in the schema as an enumeration, limited to strings which correspond to the simplenlg **PhraseCategory** enumeration values. For noun phrases, this value is **NOUN_PHRASE**. However, **PhraseElement** is abstract, any actual sub-type, like **NPPhraseSpec** has its **PhraseCategory** implicitly determined.

The properties **ADJECTIVE_ORDERING**, **ELIDED**, **NUMBER**, **PERSON**, **POSSESSIVE**, and **PRONOMINAL** are those that map to simplenlg features relevant to noun phrases. The **numberAgreement** and **person** types map to the simplenlg enumerations, as **phraseCategory** does.

The NPPhraseSpec **spec** element, and the **PhraseElement head** element are of type **WordElement**. The **spec** element corresponds to the simplenlg **NPPhraseSpec.setSpecifier** and **NPPhraseSpec.getSpecifier** methods which, in effect, define an optional simplenlg property that only **NPPhraseSpec** objects have.

The xml value of a **WordElement** is interpreted as the base form of a word. The property **cat** is the name of the **lexicalCategory** attribute of a **WordElement**. The values of **lexicalCategory**-type attributes are strings that are the names of members of the simplenlg **LexicalCategory** enumeration. **EXPLETIVE_SUBJECT** and **PROPER** properties are taken from the **LexicalFeature** members, being features that can be set by the user, although perhaps these would be better associated with a subtype of **PhraseElement** than with a **WordElement**, as they are perhaps only useful in **NPPhraseSpec** elements.

The xml elements that are sub-elements of a **PhraseElement**: **frontMod**, **preMod**, **compl** and **postMod** enable the components of any phrase to be written in xml.

Naming conventions

In order to avoid confusion with duplicate class names under simplenlg.xmlrealiser.wrapper and the actual simplenlg classes, the schema defines a naming convention whereby **for any simplenlg class represented in the schema , xjc generates a wrapper class with the same name as the simplenlg class plus the prefix "Xml".** Thus, the wrapper class generated for SPhraseSpec is XmlSPhraseSpec, and so on.

How the XML is processed

The xml realiser framework works by:

1. Reading in the XML schema and generating wrapper classes for the relevant elements in the schema. This can be achieved in a few seconds with a code generation tool. The code generation tools, xjc for java, and xsd for C#, have successfully been used to generate classes. These wrapper classes act as Data Transfer Objects by which a client application can invoke simplenlg to get the realised text. Wrapper classes need only be generated once, and only if changes to the XML schema are actually made. Wrapper classes are contained in the package **simplenlg.xmlrealiser.wrapper**. These classes have the same names as real simplenlg classes, with the prefix "Xml". Java users should note that the xjc code generator is distributed with the Java SDK; a windows batch file to generate the wrapper classes is also provided (see *res/xml/runxjc.bat*).

2. Given an XML specification of a DocumentElement, conforming to the schema, the **simplnlg.xmlrealiser.UnWrapper** class uses the java DTO objects that are created by processing (unmarshalling) the xml. A **javax.xml.bind.Unmarshaller** is used to create a **simplenlg.xmlrealiser.wrapper.XmlDocumentElement** object that represents the xml. In **simplenlg.xmlrealiser.wrapper** are classes of the same name as real simplenlg classes. Unwrapper recursively processes the DTO objects, producing a **simplenlg.framework.DocumentElement** which is then passed to the realiser, and realized in the usual way.

Processing instructions (work in progress)

The xml schema for xml data that is processed by the xml realiser has been designed to facilitate use of the xml data to control a server. The top element **<NLGSpec>** contains a request to the server. The contained **<Request>** element, of type **RequestType** determines the request. A **<Document>** requests realisation

It was imagined that other **RequestType** elements might be used to set the lexicon database, or perform simplenlg operations that return simplenlg structures instead of realised text. The intent was to define the schema in a way that such features could be added without breaking the schema by defining other **RequestType** elements.

A request might instruct the **xmlrealiser** to perform aggregation using the **simplenlg.aggregation** classes.

Currently the **xmlrealiser** is hard-coded to use the **NIHLexicon**.

Some Examples

Some examples of xml and the realised text.

1

```
<?xml version="1.0" encoding="utf-8"?>
<Document xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
cat="PARAGRAPH" xsi:schemaLocation="http://code.google.com/p/simplenlg/schemas/version1"
xmlns="http://code.google.com/p/simplenlg/schemas/version1">
  <child xsi:type="SPhraseSpec">
    <subj xsi:type="NPPhraseSpec">
      <head cat="NOUN">transfusion of whole blood</head>
    </subj>
    <vp xsi:type="VPPhraseSpec" PASSIVE="true" TENSE="PRESENT">
      <head cat="VERB">indicate</head>
    </vp>
  </child>
</Document>
```

Transfusion of whole blood is indicated.

2

```
<?xml version="1.0" encoding="utf-8"?>
<Document xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
cat="PARAGRAPH" xsi:schemaLocation="http://code.google.com/p/simplenlg/schemas/version1"
xmlns="http://code.google.com/p/simplenlg/schemas/version1">
  <child xsi:type="SPhraseSpec">
    <preMod xsi:type="PPPhraseSpec">
      <head cat="PREPOSITION">as a result of</head>
      <compl xsi:type="NPPhraseSpec">
        <head cat="NOUN">procedure</head>
        <spec cat="DETERMINER">the</spec>
      </compl>
    </preMod>
    <subj xsi:type="NPPhraseSpec">
      <head cat="NOUN">patient</head>
      <spec cat="DETERMINER">the</spec>
    </subj>
    <vp xsi:type="CoordinatedPhraseElement" conj="and">
      <coord xsi:type="VPPhraseSpec" TENSE="PAST">
        <head cat="VERB">have</head>
        <compl xsi:type="NPPhraseSpec">
          <head cat="NOUN">adverse contrast media reaction</head>
          <spec cat="DETERMINER">a</spec>
        </compl>
      </coord>
      <coord xsi:type="VPPhraseSpec" TENSE="PAST">
        <head cat="VERB">have</head>
        <compl xsi:type="NPPhraseSpec">
          <head cat="NOUN">decreased platelet count</head>
          <spec cat="DETERMINER">a</spec>
        </compl>
      </coord>
      <coord xsi:type="VPPhraseSpec" TENSE="PAST">
        <head cat="VERB">go</head>
        <postMod xsi:type="PPPhraseSpec">
          <head cat="PREPOSITION">into</head>
          <compl xsi:type="NPPhraseSpec">
            <head cat="NOUN">cardiogenic shock</head>
          </compl>
        </postMod>
      </coord>
    </vp>
  </child>
</Document>
```

The patient as a result of the procedure had an adverse contrast media reaction, had a decreased platelet count and went into cardiogenic shock.

3

```
<?xml version="1.0" encoding="utf-8"?>
<Document xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
cat="LIST" title="Conclusions" xsi:schemaLocation="http://code.google.com/p/simplenlg/schemas/version1"
xmlns="http://code.google.com/p/simplenlg/schemas/version1">
  <child xsi:type="DocumentElement" cat="LIST_ITEM"
xsi:schemaLocation="http://code.google.com/p/simplenlg/schemas/version1 ">
    <child xsi:type="DocumentElement" cat="SENTENCE"
xsi:schemaLocation="http://code.google.com/p/simplenlg/schemas/version1">
      <child xsi:type="NPPhraseSpec">
        <head cat="NOUN">normal coronary arteries</head>
      </child>
    </child>
  </child>
  <child xsi:type="DocumentElement" cat="LIST_ITEM"
xsi:schemaLocation="http://code.google.com/p/simplenlg/schemas/version1">
    <child xsi:type="DocumentElement" cat="SENTENCE"
xsi:schemaLocation="http://code.google.com/p/simplenlg/schemas/version1">
      <child xsi:type="NPPhraseSpec">
        <head cat="NOUN">normal left heart hemodynamics</head>
      </child>
    </child>
  </child>
  <child xsi:type="DocumentElement" cat="LIST_ITEM"
xsi:schemaLocation="http://code.google.com/p/simplenlg/schemas/version1">
    <child xsi:type="DocumentElement" cat="SENTENCE"
xsi:schemaLocation="http://code.google.com/p/simplenlg/schemas/version1">
      <child xsi:type="NPPhraseSpec">
        <head cat="NOUN">normal right heart hemodynamics</head>
      </child>
    </child>
  </child>
</Document>
```

Conclusions

** Normal coronary arteries.*

** Normal left heart hemodynamics.*

** Normal right heart hemodynamics.*

4

```
<?xml version="1.0" encoding="utf-8"?>
<Document xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
cat="PARAGRAPH" xsi:schemaLocation="http://code.google.com/p/simplenlg/schemas/version1"
xmlns="http://code.google.com/p/simplenlg/schemas/version1">
  <child xsi:type="SPhraseSpec">
    <subj xsi:type="NPPhraseSpec">
      <head cat="ADVERB">there</head>
    </subj>
    <vp xsi:type="VPPhraseSpec">
      <head cat="VERB">be</head>
      <compl xsi:type="NPPhraseSpec">
        <preMod xsi:type="CoordinatedPhraseElement" conj=",">
          <coord xsi:type="AdjPhraseSpec">
            <head cat="ADJECTIVE">eccentric</head>
          </coord>
```

```
          <coord xsi:type="AdjPhraseSpec">
            <head cat="ADJECTIVE">tubular</head>
          </coord>
        </preMod>
        <head cat="NOUN">restenosis</head>
        <postMod xsi:type="StringElement">
          <val>(18 mm x 1 mm)</val>
        </postMod>
        <spec>a</spec>
      </compl>
      <postMod xsi:type="PPPhraseSpec">
        <preMod xsi:type="VPPhraseSpec" FORM="GERUND">
          <head cat="VERB">extend</head>
        </preMod>
        <head cat="PREPOSITION">from</head>
        <compl xsi:type="NPPhraseSpec">
          <preMod xsi:type="AdjPhraseSpec">
            <head cat="ADJECTIVE">proximal</head>
          </preMod>
          <spec cat="DETERMINER">the</spec>
        </compl>
        <postMod xsi:type="PPPhraseSpec">
          <head cat="PREPOSITION">to</head>
          <compl xsi:type="NPPhraseSpec">
            <preMod xsi:type="AdjPhraseSpec">
              <head cat="ADJECTIVE">mid</head>
            </preMod>
            <head cat="NOUN">right coronary artery</head>
            <spec cat="DETERMINER">the</spec>
          </compl>
        </postMod>
      </postMod>
      <postMod xsi:type="PPPhraseSpec">
        <head cat="PREPOSITION">with</head>
        <compl xsi:type="NPPhraseSpec">
          <head cat="NOUN">TIMI 1 flow</head>
        </compl>
      </postMod>
    </vp>
  </child>
</Document>
```

There is an eccentric, tubular restenosis (18 mm x 1 mm) extending from the proximal to the mid right coronary artery with TIMI 1 flow.

Regression Testing with xml files

XML files that contain **<Recording>** instead of **<Request>** are used for automated testing of the xmlrealiser. A test set is an xml file with a **<Recording>** element representing the test set, containing **<Record>** elements that have a **<Document>** to realise and a **<Realisation>** element for the expected output.

These test files can be produced by an xml server that implements recording of input and output. This is described in Processing instructions (work in progress).

The tests are implemented with JUint using data-driven parameterized tests. There is one JUnit test per file. Each <Record> in the file is tested separately, but this is not apparent from the JUnit display. (There is an add-on at http://code.google.com/p/junitparams/ that might solve this problem.) However, if the test fails the failure trace will contain the test file name, the Recording name and the Record name.

XML test files can be placed in a directory and run together or can be run individually. **testsrc/simplenlg/test/xmlrealiser/Tester.java** is the implementation file. It needs two parameters. These it reads from environment variables of the same name. In Eclipse these can be set as JUnit Run Configuration properties.

LexDBPath – The pathname of the NIH Specialist Lexicon data file.

TestFilePath – Pathname of an individual test xml file, or pathname of a directory. If a directory, all xml files in that directory or sub-directories will be tests to be run.