An XML exchange format for (programming) tasks

Version 0.9

contributors listed in alphabetical order:

Technische Universität Clausthal: Niels Pinkwart, Sven Strickroth
Universität Duisburg-Essen: Michael Striewe
Hochschule Hannover: Sebastian Becker, Oliver Bott
Universität Osnabrück: Helmar Gust, Nadine Werner
Ostfalia Hochschule für Angewandte Wissenschaften: Stefan Bisitz, Stefan Dröschler, Nils Jensen, Uta Priss, Oliver Rod

Introduction

This document specifies syntax and semantics for a standardized “Task Format” - an exchange format for programming exercises/tasks. It formalises the specification of programming exercises, for example, the description, required programming language and suggested tests for evaluating the student submitted code so that exercises written for one tool can be exported and imported into another tool. The main e-assessment tools considered for this format are Praktomat, VIPS, JACK, GATE or Moodle-Grapper WS (inspired by Web-CAT) most of which are included in the eCULT project “ProFormA”.

General Structure

The XML exchange format consists of two parts: The first section shows the description/ specification of a task, including supporting files; and the second section demonstrates a specification of tests which are to be included in the specification of a task under the <tests> tag. Each task can have many tests.

Format specification

The XML file can be exchanged as a standalone file. However, if it refers to several files, it is recommended to combine the XML file and all linked files into a ZIP-archive. The XML file

itself should be stored in the ZIP under the name “/task.xml” so that it can be found by tools supporting this format. The XML file should be well formed and encoded in UTF-8 (RFC 3629). ID’s are globally unique within the XML document. Files should be specified in the smallest scope they apply to.

Task section

XML Specification

The general structure of the XML format is given as follows (this is meant to provide an overview and does not represent a minimal valid document):

```
<task version="0.9" lang="[LANG code]”>
	<description></description>
	<language version=””></language>
	<submission />

	<files />

	<model-solutions />

	<tests />
	<grading-hints />

	<meta-data />
</task>
```

The following code shows the XML Schema for the Task Format:

```
<xs:element name="task">
	<xs:complexType>
		<xs:sequence>
			<xs:element ref="description"/>
			<xs:element ref="language"/>
			<xs:element ref="submission"/>
			<xs:element ref="files"/>
			<xs:element ref="model-solutions"/>
			<xs:element ref="tests"/>
			<xs:element ref="grading-hints"/>
			<xs:element ref="meta-data"/>
		</xs:sequence>
		<xs:attribute name="version" type="xs:string" use="required"/>
		<xs:attribute name="lang" type="xs:string" use="required"/>
```

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

The document root element “task” holds the current version number of the XML Task Format. The only currently valid version is 0.9. The task itself must have an attribute “lang” which specifies the natural language used. The description, title etc should be written in this language. The content of the “lang” attribute must comply with the IETF BCP 47, RFC 4647 and ISO 639-1:2002 standards.

The description part

```
<xs:element name="description"  type="xs:string" />
```

An instance of this element contains the task description as text. A subset of HTML is allowed (see Appendix A).

The language part

```
  <xs:element name="language">
        <xs:complexType>
              <xs:simpleContent>
                    <xs:extension base="xs:string">
                          <xs:attribute name="version" type="xs:string" use="required"/>
                    </xs:extension>
              </xs:simpleContent>
        </xs:complexType>
  </xs:element>
```

An instance of this element contains the programming/modelling/query language to which this task applies. A valid list of values is specified in Appendix B. The “version” attribute specifies which version of the language was used in the creation of the task. (The task is guaranteed to work with that version – any other requirements about version compatibility must be checked externally.) The “version” must be entered as a “point” separated list of up to four unsigned integers.

The submission part

```
  <xs:element name="submission">
        <xs:complexType>
              <xs:choice>
                    <xs:element ref="file-submission"/>
                    <xs:element ref="textarea"/>
              </xs:choice>
```

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

An instance of this element holds exactly one of the following elements: “file-submission” or “textarea”. This specifies if this task requires a file submission or if this task is based on a single textbox where the solution can be entered.

The file-submission element

```
<xs:element name="file-submission">
        <xs:complexType>
                <xs:attribute name="max-size" type="xs:positiveInteger" use="optional"/>
                <xs:attribute name="allowed-upload-filename-regexp" type="xs:string"
use="optional" default=".*"/>
                <xs:attribute name="unpack-files-from-archive" type="xs:string" use="optional"
default="false"/>
                <xs:attribute name="unpack-files-from-archive-regexp" type="xs:string"
use="optional" default=".*"/>
        </xs:complexType>
</xs:element>
```

If in place, this element shows that this task requires the upload of a file. The optional attributes of this instance restrict which files are accepted for this task by a conforming e-assessment tool. Four attributes are allowed here: “max-size”, “allowed-upload-filename-regexp”, “unpack-files-from-archive”, and “unpack-files-from-archive-regexp”.

- “max-size” specifies the maximum size of a file in bytes which should be accepted. Systems which have a stronger limit of the file size should print a warning to the importing user. If this attribute is missing, a system default value will be used.
- “allowed-upload-filename-regexp” holds a regular expression of the filenames (only the filename, without path) which the system should accept.
- "unpack-files-from-archive" specifies if uploaded archives (zip/jar) should be unpacked automatically. If it is set to *false,* no extraction takes place and the archive is used as it is.
- In case "unpack-files-from-archive" is set to *true,* “unpack-files-from-archive-regexp” is honoured which holds a regular expression that controls which files are automatically extracted (the filename of the uploaded archive must of course match “allowed-upload-filename-regexp”). Only matching files (the whole path of the zip-items matches with “/” as path separator) are extracted from the archive.

The textarea part

```
<xs:element name="textarea">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="template" minOccurs="0"/>
```

```
                        <xs:element ref="preanswercode" minOccurs="0"/>
                        <xs:element ref="postanswercode" minOccurs="0"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
<xs:element name="template" type="xs:string"/>
<xs:element name="preanswercode" type="xs:string"/>
<xs:element name="postanswercode" type="xs:string"/>
```

If in place, it shows that a task does not require a file upload but a textarea for (smaller-scale) student submitted solutions. This element contains three more elements: “template”, “preanswercode” and “postanswercode” which are optional. The text of the “template” element is used to provide a template or predefined outline of the solution to the students. “preanswercode” and “postanswercode” are only interesting in combination with automatic testing/feedback and are used to build a compilable and runnable program together with the student's solution. For this purpose the text of “preanswercode” is inserted before the student's solution and the text of “postanswercode” is inserted after the student's solution (this also applies to model-solutions which are “equivalent” to student's solutions in this context).

The files part

```
<xs:element name="files">
        <xs:complexType>
                <xs:sequence minOccurs="0" maxOccurs="unbounded">
                        <xs:element ref="file"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

The files element contains 0 or more file elements. A file element is used to attach files to a task. Files can be external or embedded into the XML file.

The file element

```
  <xs:element name="file">
         <xs:complexType>
                 <xs:simpleContent>
                         <xs:extension base="xs:string">
                                 <xs:attribute name="id" type="xs:string" use="required"/>
                                 <xs:attribute name="filename" type="xs:string" use="optional"/>
                                 <xs:attribute name="class" use="required">
                                         <xs:simpleType>
                                                 <xs:restriction base="xs:string">
                                                         <xs:enumeration value="template"/>
                                                         <xs:enumeration value="library"/>
                                                         <xs:enumeration value="workingdata"/>
```

```
                                                    <xs:enumeration value="instruction"/>
                                                    <xs:enumeration value="internal"/>
                                                </xs:restriction>
                                            </xs:simpleType>
                                        </xs:attribute>
                                        <xs:attribute ref="type"/>
                                    </xs:extension>
                                </xs:simpleContent>
                            </xs:complexType>
                        </xs:element>
                        <xs:attribute name="type">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="file"/>
                                    <xs:enumeration value="embedded"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
```

The file element includes or links a single file to a task. Each instance/file must have a (task) unique string in its “id” attribute (in order to reference this file within this task) and has to be classified using the “class” attribute with one of the following values:

- “template”: The file is a template for students to be used as a starting point for their solution.
- “library”: The file is a library to be used by students (and might also be required for tests).
- “inputdata”: The file contains data which the algorithm of the students should work with.
- “instruction”: The file contains further instructions for handling the task, e.g. an UML activity diagram.
- “internal”: This file is not visible for students and holds files which are required for processing the task/tests within the system.

The file itself can be embedded into the XML (recommended for shorter plain text files) or can be included in the ZIP-archive (recommended for binary files). If a file is embedded, the “type” attribute must be set to “embedded” and the text content of the element is the file content. Optionally the “filename“ attribute can be set to define the filename (e.g., for Java classes where the class name must be equal to the filename, it can also include a relative path). This attribute mainly applies to embedded files. If no filename is provided, it is up to the system to generate a temporary name. If a file is not embedded, the “type” attribute must be set to “file” and the text content of the element contain the filename within the task ZIP archive.

The model-solutions part

```
<xs:element name="model-solutions">
        <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
                <xs:element ref="model-solution"/>
```

```
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

The model solutions element is used to provide one or more solutions of the task. For each model-solution a new model-solution element is added.

The model-solution element

```
<xs:element name="model-solution">
        <xs:complexType>
                <xs:simpleContent>
                        <xs:extension base="xs:string">
                                <xs:attribute ref="type"/>
                        <xs:attribute name="id" type="xs:string" use="required"/>
                                <xs:attribute name="filename" type="xs:string" use="optional"/>
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
</xs:element>
<xs:attribute name="type">
        <xs:simpleType>
                <xs:restriction base="xs:string">
                        <xs:enumeration value="file"/>
                        <xs:enumeration value="embedded"/>
                </xs:restriction>
        </xs:simpleType>
</xs:attribute>
```

The model-solution element includes or links a single model-solution to a task. Each instance/ solution must have a (task) unique string in its “id” attribute. The model-solution can be directly embedded into the XML file or attached in an extra file within the zip archive (cf. “The file part”).

The tests part

The tests element is used to provide automatic checks and tests for the task. More specific information about the test XML is provided in the second section of this paper.

The grading-hints element

```
<xs:element name="grading-hints" type="xs:anyType"/>
```

An instance of this element holds information on how the creator of the task intended the grading process. This element contains plain text or arbitrary elements in different namespaces. This field is mainly intended to support an exchange of grading ideas and also to allow tasks to be exported and imported again from one system to another.

The meta-data element

```
<xs:element name="meta-data" type="xs:anyType"/>
```

The meta-data element holds a namespace for the meta-data of each system. Because meta-data are already standardized in other systems, it was decided not to formalize them as part of this specification.

Test section

XML Specification

The general structure of the test description is given as follows:

```
<tests>
	<test id=”unique ID” validity=””>
		<title></title>
		<test-type></test-type>
		<test-configuration>
			<software version=””></software>
			<files>
				<file id=””></file>
			</files>
			<code></code>
		</test-configuration>
	  <test-meta-data />
	</test>
</tests>
```

The corresponding XML schema for the test XML structure is:

```
<xs:element name="tests">
	<xs:complexType>
		<xs:sequence minOccurs="0" maxOccurs="unbounded">
			<xs:element ref="test"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>
```

The test element

```
<xs:element name="test">
```

```
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="title"/>
                        <xs:element ref="test-type"/>
                        <xs:element ref="test-configuration"/>
                </xs:sequence>
                <xs:attribute name="validity" use="optional" default=”1.00”>
                        <xs:simpleType>
                                <xs:restriction base="xs:decimal">
                                        <xs:totalDigits value="3"/>
                                        <xs:fractionDigits value="2"/>
                                        <xs:minInclusive value="0"/>
                                        <xs:maxInclusive value="1.00"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="id" use="required" type="xs:string"/>
        </xs:complexType>
</xs:element>
```

The test element has a required attribute “id” and an optional attribute “validity”. The optional attribute “validity” is used by some systems (such as Vips) for tests which only partially verify the solution code.

The title element

```
<xs:element name="title" type="xs:string"/>
```

The title element is used to provide a short and clear name for the test that can be displayed to students as part of their results. It should be noted that the title does not have a language attribute because it is assumed that the title is written in the same natural language as specified for the task itself.

The test-type element

```
<xs:element name="test-type" type="xs:string"/>
```

Examples of values are: java-syntax, java-junittest. A list of allowed entries is specified in Appendix C.

The test-configuration part

```
<xs:element name="test-configuration">
        <xs:complexType mixed="true">
                <xs:sequence>
                        <xs:element ref="software"/>
                        <xs:choice>
                                <xs:element ref="files"/>
```

```
                                <xs:element ref="code"/>
                        </xs:choice>
                        <xs:any namespace="##other" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

The test-configuration contains all parameters which are needed for configuring this specific test. The test-configuration should either contain a files part or a code element which contains the actual code of the test. It has sub-elements which are required, however, each test can also have elements of its own namespace for test-type specific configuration options.

The software element

```
<xs:element name="software">
        <xs:complexType>
                <xs:simpleContent>
                        <xs:extension base="xs:string">
                                <xs:attribute name="version" use="required" type="xs:string"/>
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

The software element determines the framework used for executing the test. It should be specified according to the list of test types in Appendix B. The version attribute contains the version number of the framework. It specifies which version was used when the test was written. (It is up to the interpreting software to determine version compatibility).

The files part

```
<xs:element name="files">
        <xs:complexType>
                <xs:sequence minOccurs="0" maxOccurs="unbounded">
                        <xs:element ref="file"/>
                </xs:sequence>
        </xs:complexType>
<xs:element>
```

Several files can be specified via file elements.

The file element

```
<xs:element name="file">
```

```
        <xs:complexType>
                <xs:simpleContent>
                        <xs:extension base="xs:string">
                                <xs:attribute name="id" type="xs:string" use="required"/>
                                <xs:attribute name="filename" type="xs:string" use="optional"/>
                        </xs:extension>
                </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

The file element includes or links a single file to a test. Each instance/file must have a (task) unique string in its “id” attribute (in order to reference this file within this task). Files should always be defined in the smallest scope possible (for example, if a file is only used by one test, it should be defined there and not in the task’s global files section). Defining files in the smallest context improves the interchangeability of single tests. The “filename” attribute is defined as in subsection “The file element” of the “Task section”.

The code element

```
<xs:element name="code" type="xs:string"/>
```

The code element contains the plain text code for testing. The code is executed by the framework specified for the test.

The test-meta-data element

<xs:element name=”test-meta-data” type=”xs:anyType”/>

The test-meta-data element holds a namespace for test-specific meta-data of each system. This is particularly useful for attributes that are required for ex- and import in one system but which are not relevant for other systems.

Appendix A: Subset of HTML

<!-- … -->, a, b, blockquote, br, p, sup, sub, center, div, dl, dd, dt, em, font, h1, h2, h3, h4, h5, h6, hr, img, li, ol, strong, pre, span, table, tbody, td, tr, th, tt and ul.
However, for div, b, br, dd, dt, dl, blockquote, p, sup, sub, h1, ...h6, li, ol, hr, strong, pre, span, table, tbody, td, tr, th, tt and ul there are no attributes allowed in order to avoid problems with different layouts.

Appendix B: List of programming languages

- java
- SQL
- prolog

Appendix C: List of test types

- java-compilation
- java-junit
- java-checkstyle
- java-code-coverage-emma
- java-findbugs
- java-pmd
- dejagnu
- anonymity (heuristics for checking that students have not included their names in the code)