Transatlantic TUmour MOdel Repositories

D4.2.2
The TUMOR Markup Language (TumorML) Version 1.0 Specification

Project Number:	FP7--IST-247754
Deliverable id:	D4.2.2
Deliverable name:	The TUMOR Markup Language (TumorML) Version 1.0 Specification
Submission Date:	21 May 2012

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	TUMOR
Project Full Name:	Transatlantic TUmour MOdel Repositories
Document id:	D4.2.2
Document name:	The TUMOR Markup Language (TumorML) Version 1.0 Specification
Document type (PU, INT, RE)	PU
Version:	1.2
Submission date:	21 May 2012
Editor: Organisation: Email:	David Johnson University of Oxford david.johnson@cs.ox.ac.uk

Document type PU = public, INT = internal, RE = restricted

ABSTRACT: This deliverable documents the specification for TumorML Version 1.0, a markup language that is to be used in the TUMOR project to wrap up existing cancer models provided by related EC projects (ACGT and ContraCancrum), and to act as a standard format for transmitting models between elements of the TUMOR project infrastructure. The need for TumorML is briefly described, along with a brief discussion on why it has been developed over adopting existing established markup for computational biology such as SBML and CellML. The specification is then described in detail. Finally, the schema code is provided in the appendices in the form of a RelaxNG compact syntax schema, and a corresponding XML Schema Document.

KEYWORD LIST: markup, XML, cancer modelling, specification, schema

MODIFICATION CONTROL			
Version	Date	Status	Author
0.1	9 May 2012	Draft	David Johnson
0.2	10 May 2012	Draft	David Johnson
0.3	11 May 2012	Draft	David Johnson
0.4	14 May 2012	Draft	David Johnson
0.5	16 May 2012	Pre-final	David Johnson
1.0	16 May 2012	Final	David Johnson
1.1	18 May 2012	Final	David Johnson
1.2	20 May 2012	Final	David Johnson

List of Contributors

- All consortium
- David Johnson (UOXF.BL)
- Stelios Sfakianakis (FORTH)
- Vangelis Sakkalis (FORTH)

Contents

1 Executive Summary

The TUMOR project aims at developing a European *clinically oriented* semantic-layered cancer digital model repository from existing EC projects that will be interoperable with the US grid-enabled semantic-layered digital model repository platform at CViT.org (Center for the Development of a Virtual Tumor, Massachusetts General Hospital (MGH), Boston, USA) which is NIH/NCI-caBIG compatible. This interoperable, CViT interfaced, environment will offer a range of services to international cancer modellers, bio-researchers and eventually clinicians aimed at supporting both basic cancer quantitative research and individualized optimization of cancer treatment. This 'Transatlantic' project will therefore be the starting point for an international validation environment that will support joint applications, verification and validation of the clinical relevance of cancer models.

The purpose of this deliverable is to document the specification of *TumorML*, a new markup language to be used by the TUMOR project to enable interoperability between the component parts of the TUMOR infrastructure. This includes transferring models in the 'transatlantic' context where models are imported and exported between the TUMOR repository developed by WP3, and also between the parts of the EU infrastructure, namely the repository and the model execution system developed by WP5. In addition to this, TumorML will provide a standard format for exporting models to external systems that have not been identified or defined within the scope of TUMOR allowing future compatibility.

In deliverable D4.2.1 a review the state-of-the-art markup languages was carried out to determine what existing vocabularies could be used to wrap up and annotate cancer models that will be stored and made available in the TUMOR digital model repository. The current state-of-the-art biological modelling languages were reviewed, including a review a range of XML-based metadata vocabularies that could be integrated into a new markup language for cancer modelling. Based on this review a design for TumorML was described. The work described in this deliverable D4.2.2 used the aforementioned design as a starting point and further refined the design in collaboration with WP3 and WP5 to produce the finalized schema. The full schema expressed in RelaxNG Compact Syntax and as an XML Schema Document (XSD) is found in the appendix of this document.

2 Introduction

The main aim of the TUMOR project is to create a European-based digital repository for clinically oriented cancer models. The repository will store models provided by other EC projects such as the Advancing Clinico Genomic Trials on Cancer (ACGT) and the Clinically Oriented Translational Cancer Multilevel Modeling (ContraCancrum) projects. Biological model repositories are not novel, as demonstrated by existing services provided by the CellML repository, E-Cell, and biomodels.net to name but a few. However, one of the key aims of the TUMOR project is to enable the European cancer model repository to seamlessly interoperate with its US equivalent service that was developed by the Center for the Development of a Virtual Tumor (CViT) project led by the Massachusetts General Hospital (MGH) in Boston, USA. This ‘transatlantic’ link may ultimately allow the US and EU cancer research communities to pool their resources through effective model sharing, and act as a bridge between the two communities to foster further research advancements in cancer.

WP4 of the TUMOR project focuses on developing interoperable interfaces between the two repositories. This will be achieved by the development of a set of Web services to allow the two repositories to communicate with each other, a task led by MGH/Infotech. Secondly, and the focus of this deliverable, the TUMOR project is to develop a simulation markup language specifically targeted at the cancer modelling domain that will act as the standard communication format between elements of the TUMOR infrastructure and eventually for exporting models to external services. The success criteria of the new markup language are to achieve the following two goals.

1. Being able to demonstrate the import and export of models between the two repositories preserving as much metadata as possible, and translating between the US and EU schemas where appropriate.
2. Demonstrating a ‘transatlantic’ compound model linked together via markup describing each component model’s interfaces and their couplings.

In computational biology, there is diverse range of programming and descriptive languages that span across different biological domains and scales. This creates challenges for model reuse and composition, since each model implementation, even if available, may use a completely different technological framework. Combining models may therefore require porting models to a new framework, or re-implementing them, both costly and error prone activities. Before the year 2000, there were no unified efforts towards standardized languages for describing models. Markup languages for computational biology emerged soon after the turn of the millennium with the SBML (Systems Biology Markup Language) and CellML research programmes.

To address the specific domain of cancer modelling, we have developed a markup language, *TumorML*, to describe computational cancer models within TUMOR. The content of this document describes the TumorML specification.

Why do we need a TUMOR Markup Language?

Although generic markup has been established for certain areas of computational biology modelling in SBML and CellML, each has its restrictions when applied to the cancer domain in TUMOR. SBML is developed for the realm of systems biology— a broad ranging domain, but nonetheless a specific kind of modelling for molecular scale processes, and hence mostly applicable only to a subset of the models considered in TUMOR. CellML was developed primarily to describe biological cell function. Neither of these modelling languages fully satisfies the needs and diversity found in clinically-oriented cancer modelling. One other feature that is prevalent throughout is that the state-of-the-art modelling languages are designed to mainly simulate through pure mathematical description. Where an *in silico* approach needs more algorithmic descriptions, for example in agent-based simulations, TumorML aims to enable this by creating a markup that can wrap up existing model implementations in domain-specific elements.

Let us consider the example of the Oncosimulator developed by the ACGT and ConracCancrum projects and described in Stamatakos et al[1]. This model produces its simulations based on a combination of techniques including nondeterministic finite-state automata, the generic Monte Carlo technique, differential equations, general algorithms and complexity theory. CellML being based on declarative mathematics does not support statistical models or models that incorporate random factors, such as in Monte Carlo simulations, nor does it support procedural operations that are required for describing algorithms making it unsuitable for describing the Oncosimulator. SBML in contrast does have greater support for a range of techniques, however tends to be used primarily as an exchange format for systems biology simulation software. Whilst the Oncosimulator does consider molecular scale interactions in its component models, it also considers high-level factors taken from the clinical perspective such as a patient's previous medical history. Such top-down factors are not supported in SBML. TumorML is being developed to overcome these domain-specific limitations, but not as a competitor to either of CellML and SBML, but rather to deal with existing cancer model implementations such as the Oncosimulator.

[1] The "Oncosimulator": a multilevel, clinically oriented simulation system of tumor growth and organism response to therapeutic schemes. Towards the clinical evaluation of in silico oncology, G.S. Stamatakos et al, in Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2007.

3 TumorML specification

Generally speaking, all application-specific markup languages are based on the eXtensible Markup Language (XML) (Bray et al, 2004). XML emerged as a popular choice for computer-based language definition in the late 1990's as it defines a standard syntax on to which other vocabularies can be built. This allows language-specific parsers to reuse the standard XML parsing routines for processing XML documents. XML-based languages are therefore well suited as software-neutral information exchange formats. XML can be thought of as the base alphabet and grammar of a language. What raw XML lacks is semantics or definitions to provide context and application.

This is provided in part by specific vocabularies built on XML, which define element and attribute names and the structural relationships between them. Multiple XML vocabularies can be combined within a single document, enabling the development of various languages targeted at specific narrow domains of discourse that can be incorporated into a compound language.

TumorML has been developed as such an XML-based domain-specific vocabulary and includes elements from existing vocabularies to avoid 'reinventing the wheel.' The candidate vocabularies for reuse were discussed in D4.2.1 and since that time of publication of the initial TumorML design, several decisions were taken in the selection of what can and cannot be reused in this specification. The vocabularies reused, in part, for TumorML version 1.0 are:

- Dublin Core (http://dublincore.org/documents/dces/) – For basic curation of models and model implementations, we reuse a subset of the Dublin Core Metadata Element Set (DCMES). Dublin Core was described in the markup review document D4.2.1.
- xCard (http://tools.ietf.org/html/rfc6351) – To describe entities that create the model entries in the TUMOR repository, we use XML representation of vCards. RDF vCard is described in the markup review document D4.2.1, however the xCard RFC standard proposal was only published in August 2011 after the markup review was carried out.
- BibTeXML (http://bibtexml.sourceforge.net) – Whilst not a standardized vocabulary, BibTeXML is an XML representation of the BibTeX format for bibliographic references. In this first version of TumorML we reuse the document categories with a view to implementing the full BibTeX reference structure in future versions. BibTeXML was not described in the markup review.
- JSDL (http://www.ggf.org/documents/GFD.56.pdf) – A subset of JSDL execution requirements is used in TumorML for describing the basic requirements for executing TumorML models. JSDL was described in the markup review document D4.2.1.
- xMML (http://dx.doi.org/10.1109/eScienceW.2011.9) – The Multiscale Modeling Language, MML, and its XML version, xMML, proposes a standard way of describing multiscale models and most importantly how to couple models of different scales. TumorML reuses a simplified part of the xMML coupling markup to represent the coupling of computational models, in particular the concepts of declaring instances of models and coupling parameters of a instantiated compound model are used. xMML was not described in the markup review as the authors' results were published after the last deliverable was completed.

This description of the XML schema specification for TumorML follows.

Common Definitions

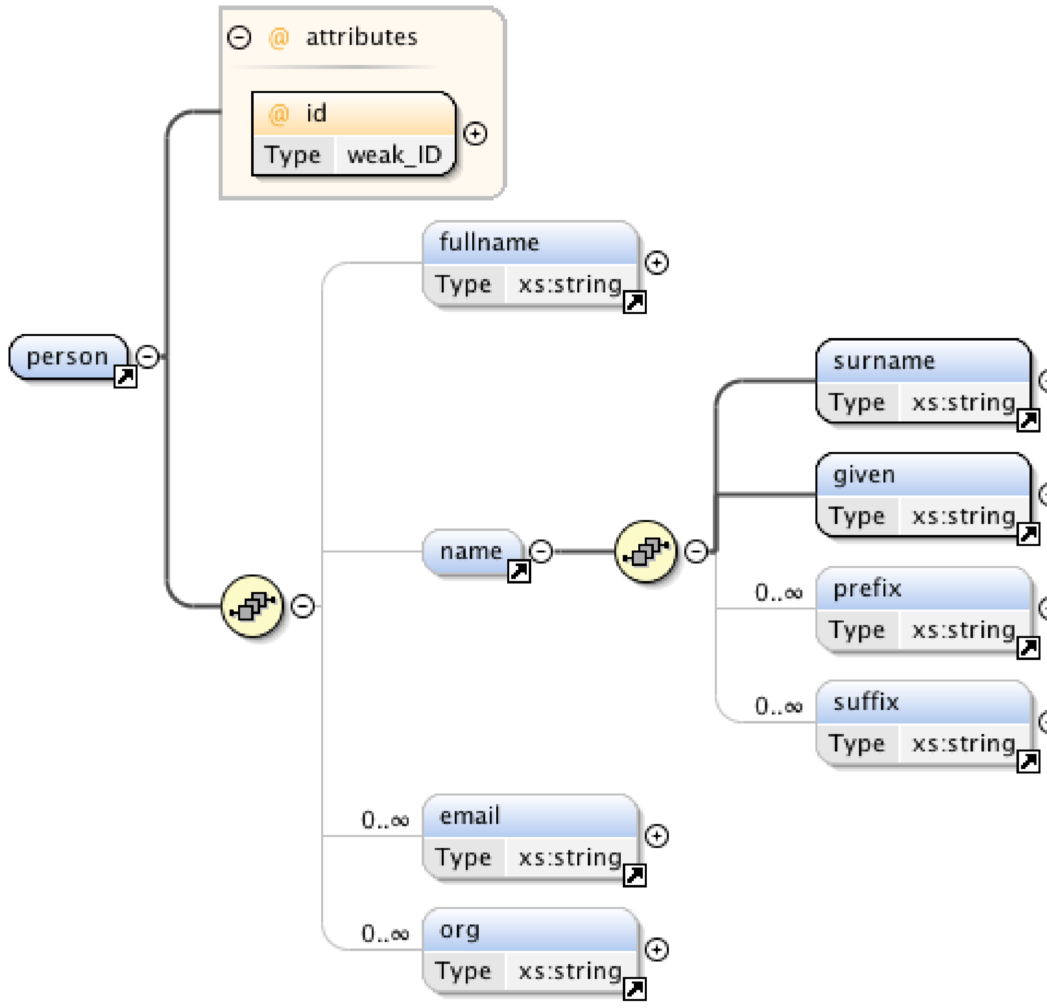
Throughout TumorML, there are several predefined sets of markup that are reused in difference parts of the specification. These are **`weak_ID`** type, the **`<person>`** element, the **`dc_elements`** element pattern consisting of Dublin Core elements and the **`<reference>`** element.

IDs in TumorML

`weak_ID` is derived from the XSD type **`NCName`** that represents a non-colonized name used as a base for the XSD type **`ID`**. The purpose of **`weak_ID`** is to provide the most flexible identifier without having to make it unique within the TumorML document, as its primary use is for referring to repository database records. As the way records are identified in the repository database is currently undefined, this allows maximum flexibility that can be further constrained in a later version of TumorML. In addition to this, there is no rule that declares that an ID for one type of element cannot be identical to that of another element. ID does not enforce this as the **`ID`** constraint enforces uniqueness global to the document root.

The ***`person`*** element

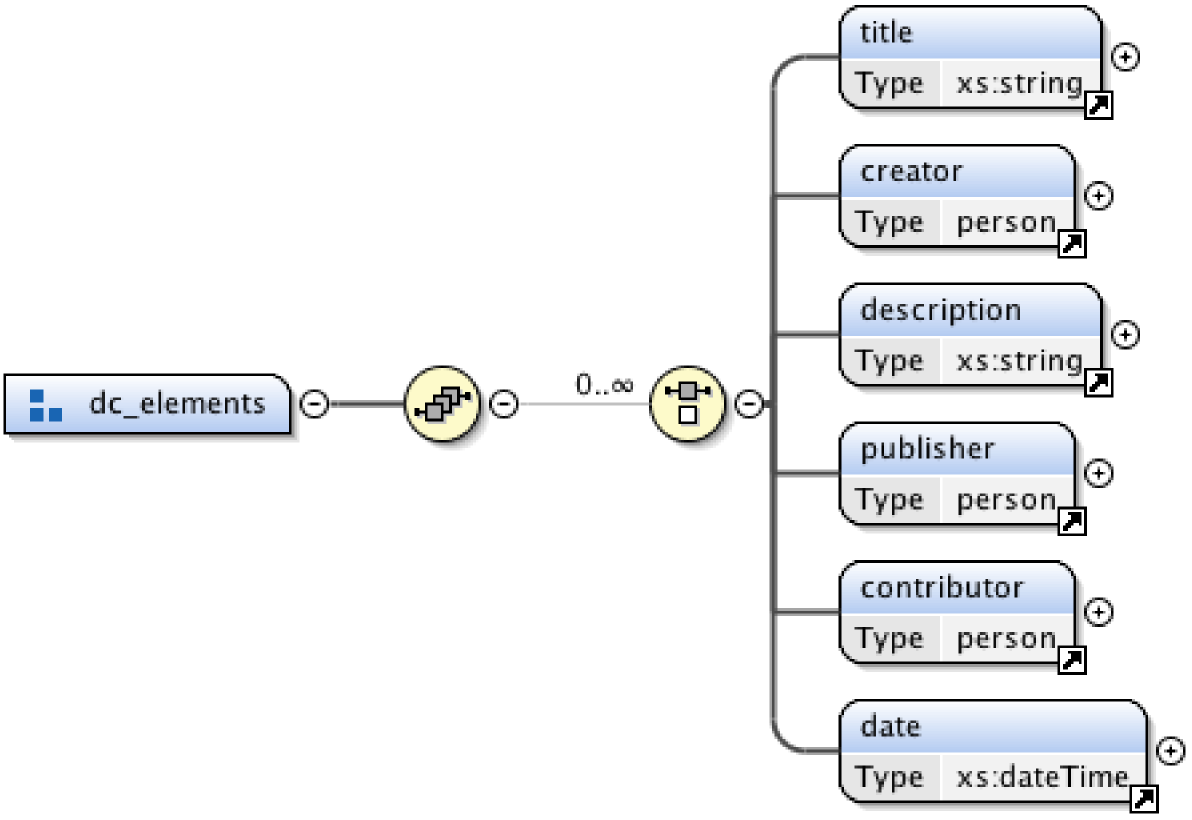
The **`<person>`** element is used to describe stakeholders, typically users that have some role in a TumorML document. A **`<person>`** element is expressed using a subset of the xCard standard.



The four sub-elements describe a person's full name, name broken down into constituent parts, email address, and organisation. A person also has an **id** for lookup in the repository.

Dublin Core elements

dc_elements represents the markup pattern for a set of Dublin Core elements. In this specification we have not exhaustively reproduced the whole element set, but rather just the following:

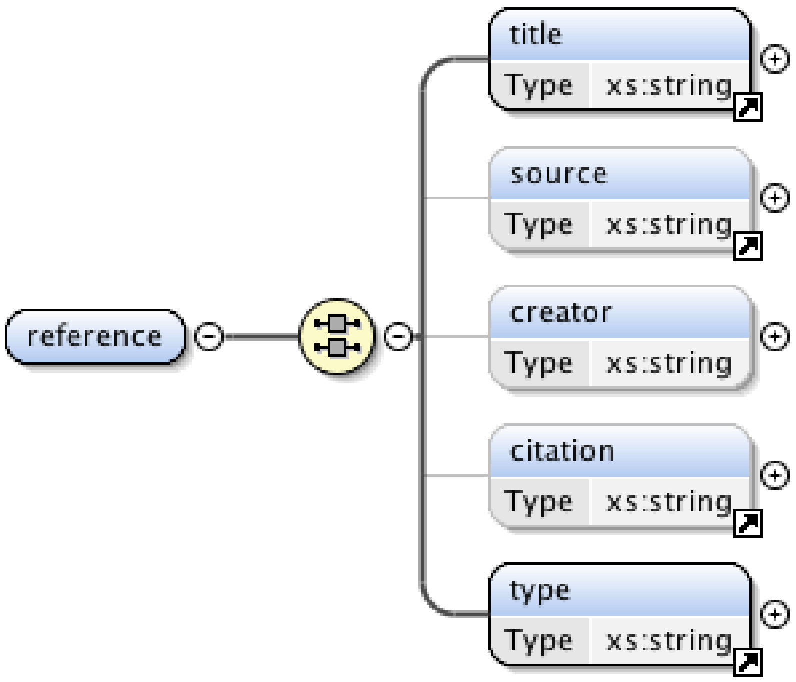


- **<title>** – the name of the resource given by the creator or publisher. This element's content is expressed as a **string**. This element is mandatory.
- **<creator>** – the person or organisation responsible for creating the intellectual content of the resource at hand. This element's content is expressed as a <**person>**. This element is optional.
- **<description>**– a textual description of the resource. This may be an abstract or a textual description for non-textual content. This element's content is expressed as a **string**. This element is optional.
- **<publisher>** – the person or organisation responsible for making the intellectual content of the resource at hand available. Note that this may be a different entity to the creator. This element's content is expressed as a <**person>**. This element is optional.
- **<contributor>** – persons or organisations that, in addition to the creator, may have contributed to creating the intellectual content of the resource at hand. This element's content is expressed as a <**person>**. This element is optional and can consist of multiple entries.

- **`<date>`** – the date the resource was published (specific to this particular version of the resource). The date is assumed to be a timestamp expressed in the XSD type **`dateTime`**. This element is mandatory.

The ***`reference`*** element

Finally, the **`<reference>`** element is used to describe bibliographic references of models and related components of model descriptions. These should be used to correctly attribute work that has been done elsewhere that the model description may be based on.



The elements reflect metadata relating to external references:

- **`<title>`** – The title of the referenced resource. This element is mandatory.
- **`<source>`** – The source URL of a referenced resource. This element is optional.
- **`<creator>`** – The author of the referenced resource. This element is optional.
- **`<citation>`** – Full text citation in any form. This element is optional.
- **`<type>`** – Categorization of the resource. The categories are taken from the BibTeX file format and include the following categories: *article, book, booklet, manual, technical report, Masters thesis, PhD thesis, in book, in collection, proceedings, in proceedings, conference, unpublished, and miscellaneous*. This element is mandatory.

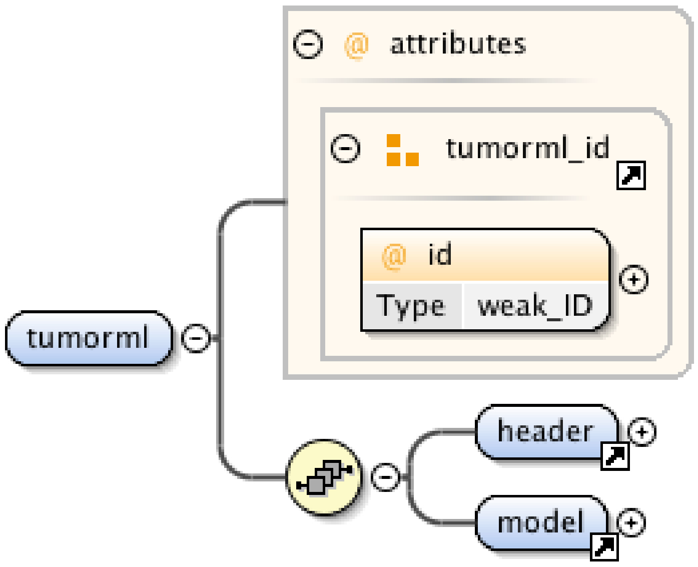
TumorML root element

A well-formed TumorML document, like all well-formed XML documents, must begin with an XML declaration to specify the XML version and character encoding used within the rest of the document.

The root element of every TumorML document is <**`tumorml>`**. The document is split into two main sections

- **`<header>`** – A header containing metadata about the model.
- **`<model>`** – A description of the model itself in terms of interface and implementation.

The top-level structure of TumorML is depicted below.



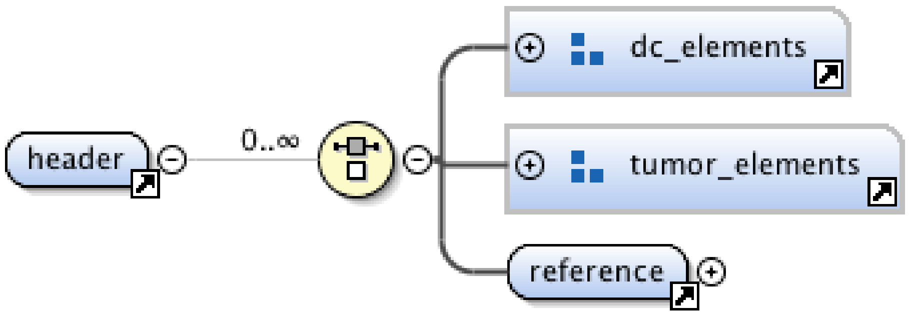
The following example shows the basic structure of a TumorML document omitting child elements that are described in the following sections:

```
<?xml version="1.0" encoding="UTF-8"?>
<tumorml xmlns="http://www.tumor-project.eu/tumorml/1.0" id="0001">
      <header>
            ...
      </header>
      <model>
            ...
      </model>
</tumorml>
```

Each TumorML document includes an optional **`id`** attribute that is used by the TUMOR repository to lookup specific model records.

The *header* element

The **`<header>`** element contains a list of metadata elements to aid in publishing, search and retrieval of TumorML models, and a list of external references to attribute. There may only be one **`<header>`** per TumorML document. The ordering of the elements is not constrained, however it is recommended that they appear in the following sequence for clarity. The header consists of a set of general document curation elements taken from Dublin Core, a set of TUMOR repository specific elements, and a list of references where applicable.



Dublin Core elements

The first set of metadata elements are inherited from the Dublin Core Metadata Element Set (DCMES), Version 1.1, and defined in the **`dc_elements`** pattern described previously.

TUMOR repository metadata elements

This second set of metadata elements are TUMOR specific in order to enable more cancer domain specific search for models in the TUMOR repository.

- **`<math>`** - The mathematical type that the cancer simulation model is based on. This element is optional and can take the values '*discrete*', '*continuous*', or '*hybrid*'.
- **`<biocomplexityDirection>`** - The biocomplexity direction of the model. This element is optional and can take the values '*topDown*', '*bottomUp*' or '*middleOut*'.
- **`<cancer>`** - The type of cancer described by the model. This element is optional and can take the values '*glioma*', '*nephroblastoma*', '*lung*', or '*breast*'.
- **`<materialization>`** - The materialization of the cancer. This element is optional and can take the values '*solid*' or '*liquid*'.
- **`<homogeneity>`** - The homogeneity of the tumour. This element is optional and can take the values '*homogeneous'* or '*non-homogeneous*'.
- **`<imageBasedDetectability>`** - Whether or not the tumour can be detected in an image. This element is optional and can take the values '*imageable'* or '*non-imageable*'.
- **`<freeGrowth>`** - Provides a Boolean value of whether the simulated tumor includes a free growth parameter. This element is optional.
- **`<treatmentIncluded>`** - A Boolean value of whether or not simulated tumour includes simulation of treatments. This element is optional.

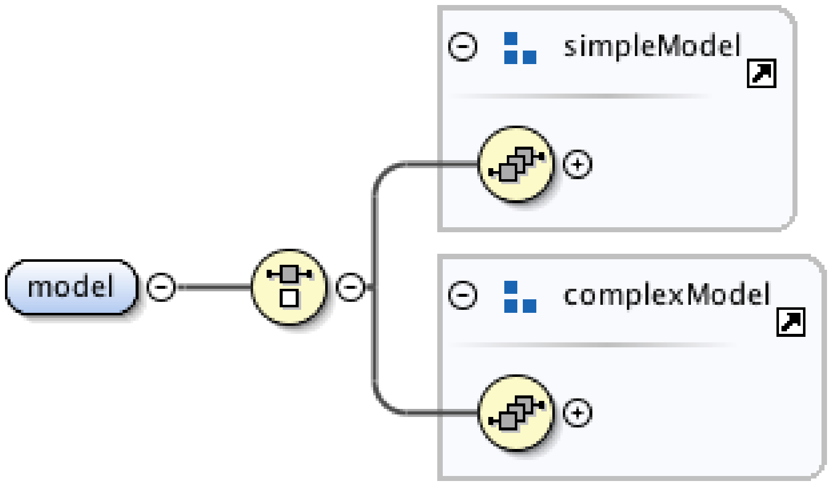
References list

Finally a list of any external references can be included in the header. The **`<reference>`** element contains the following list of metadata described earlier. Any number of references can be included.

The *`model`* element

The **`<model>`** element contains the description of a computational model, including its input and output interface, included implementations, and where applicable, compound sub-model topologies. A definition of a `model` serves as a description of the standard way to interface with the `model` via exposed parameters (both inputs and outputs), and as a container for any descriptions of model implementations. A `model` is treated like a 'black box' where how the model is implemented is of no concern, but where how to communicate data to and from the model, as well as how to execute an implementation of a model, allows us to wrap up and publish existing models with TumorML.

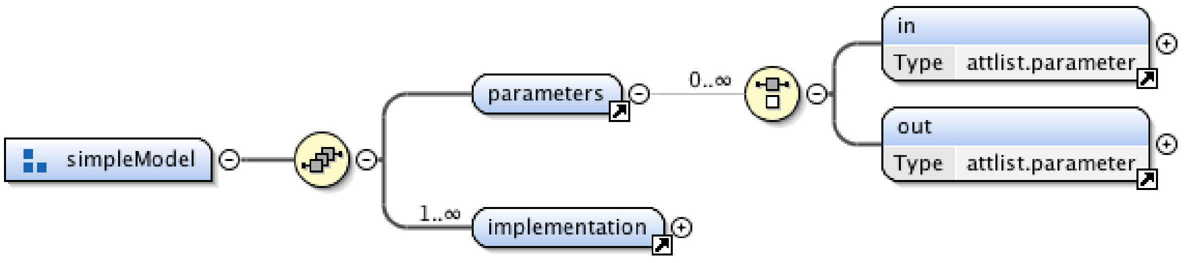
As described earlier, models can be either Simple Models or Complex Models, and this is reflected in the XML schema by having a choice of two patterns that can be enclosed within the **`<model>`** tag.



Simple Models

A simple model contains two mandatory sections inside the **`<model>`** element:

- **`<parameters>`** - The list of the input and output parameters. There may only be one parameter list declared per model.
- **`<implementation>`** - Descriptions of any implementations of the model. An implementation description contains some basic metadata about the published implementation, a file package manifest, and some basic requirements for running a simulation using the implementation. A model may have multiple implementations.



The ***`parameters`*** element

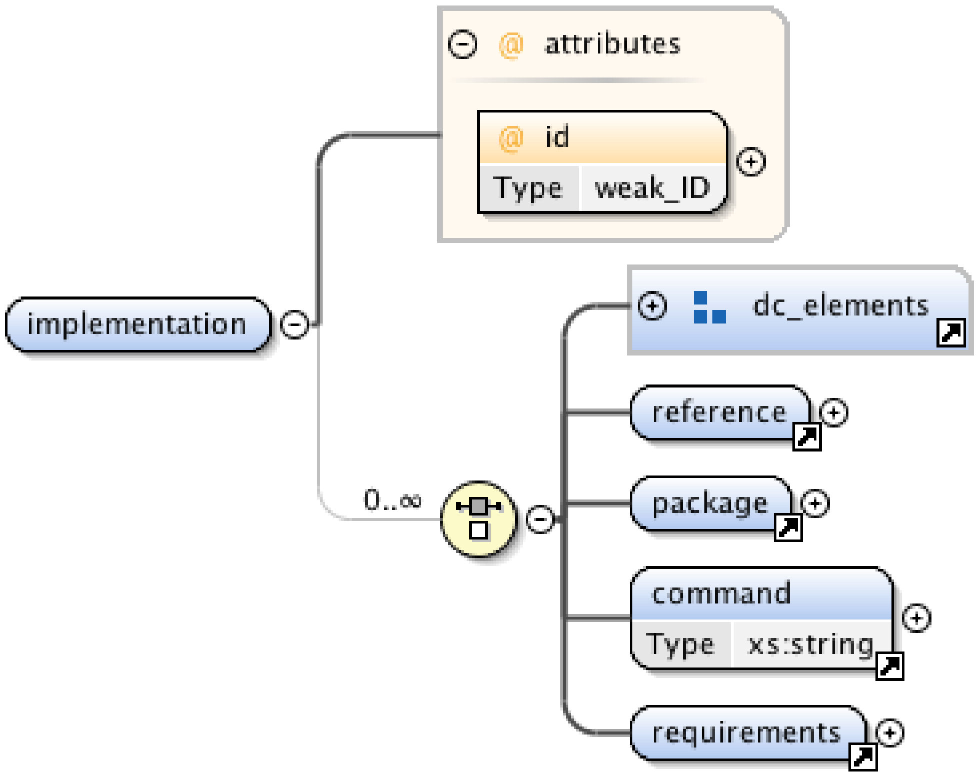
The list of parameters consists of an unordered declaration of each input and output parameter that is made externally visible by the model that is required to run a simulation of the model. Each parameter is enclosed in either **`<in>`** or **`<out>`** tags (input and output parameters respectively) holds the following metadata.

- **`name`** – A unique name as an identifier. This attribute is mandatory.
- **`label`** – A textual label describing in brief the parameter. This attribute is optional.
- **`optional`** – A Boolean value of whether or not this parameter is strictly required by the model. If the parameter is optional, a default value must be provided to allow for a value to be substituted when no value is provided at run-time. This attribute is optional, however if it is not provided it is assumed that the value is true.
- **`<description>`** – A longer text field to hold a more detailed description of the parameter than required in the label. This element is optional.
- **`<value>`** - Constraints put on the value that this parameter can take. This element is mandatory.

The constraints put on each **`<value>`** are as follows.

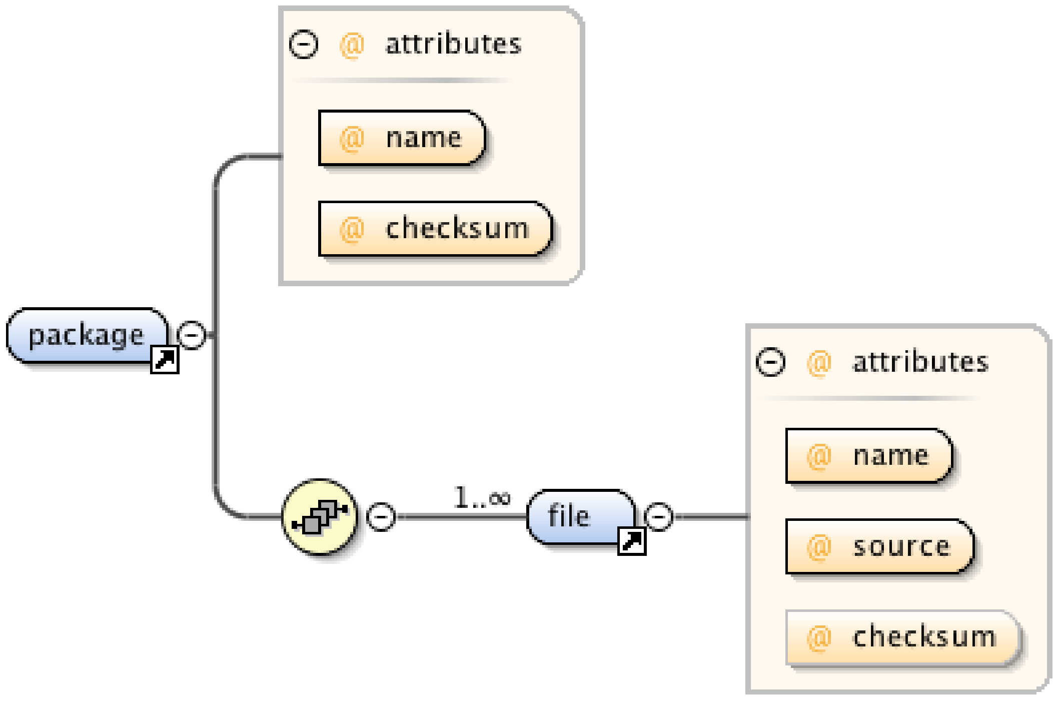
- **`type`** – What basic command-line data type this value is expressed in. Types are restricted to: '*int*', '*double*', '*filename*', and '*string*'. This attribute is mandatory.
- **`default`** – If applicable, the default value for this parameter if no value is specified at runtime. This element is optional.
- **`<range>`** – A specification of the minimum and maximum of the range of valid values, including whether or not the range is inclusive and exclusive of the boundary values specified. Attributes are provided for each respectively in **`min`**, **`max`**, and **`inclusive`**. This element is optional.
- **`<unit>`** - A specification of what quantifiable unit the value takes. This can be used to validate parameters connected in the case of compound/complex models. Unit declarations are composed of two attributes, **prefix** and **units**. **`prefix`** denotes SI prefixes listed at http://physics.nist.gov/cuu/Units/prefixes.html. **`units`** denotes SI base units listed at http://physics.nist.gov/cuu/Units/current.html. This element is optional.

Every Simple Model must include at least one executable implementation. The details of each implementation are wrapped in individual **`<implementation>`** elements.



The ***`implementation`*** element

Each **`<implementation>`** includes an in internal ID for looking up specific implementations within a single model description, some standard metadata in the form of Dublin Core elements and a set of references, a file package description, a command-line that executes the executable, and a set of basic computational requirements.



The ***package*** element

Each **<package>** contains a package name and checksum, and a list of files. Each file also has a name, a full-path source relative to the package root, and again, a checksum.

The ***command*** element

The **<command>** element specifies a single command-line instruction that should begin executing the model implementation, where it is up to the implementation's author to ensure that their files are packaged in such a way that only one such command needs to be issued to begin a simulation run. Where normally a model implementation might require multiple sub-programs to be executed, the author should provide a script that automates these multiple stages.

To link the **<command>** declaration with the **<parameters>** list, arguments of the command-line should map to the input parameter names where applicable, and the model implementation should design its execution interface accordingly. In TumorML we use the parameter substitution operator commonly found in scripting languages where a named parameter is prefixed with a dollar sign '$'. For example, given a parameter list with inputs `inputA` and `inputB`:

```
<parameters>
     <in name="inputA" optional="false">
          <value type="double" />
     </in>
     <in name="inputB" optional="true">
          <value type="int" default="1" />
     </in>
</parameters>
```

a valid **<command>** might be:

```
<command>mymodel.exe $inputA $inputB</command>
```

At run-time, the execution system will substitute values given as parameters to the command-line. Where optional parameters are given, a default value is expected so that the appropriate parameter can still be substituted.

```
<command>mymodel.exe $inputA 1</command>
```

For example, given the previous example, if `inputB` were not to provide a value at run-time, the default value would be substituted in the command-line as above.

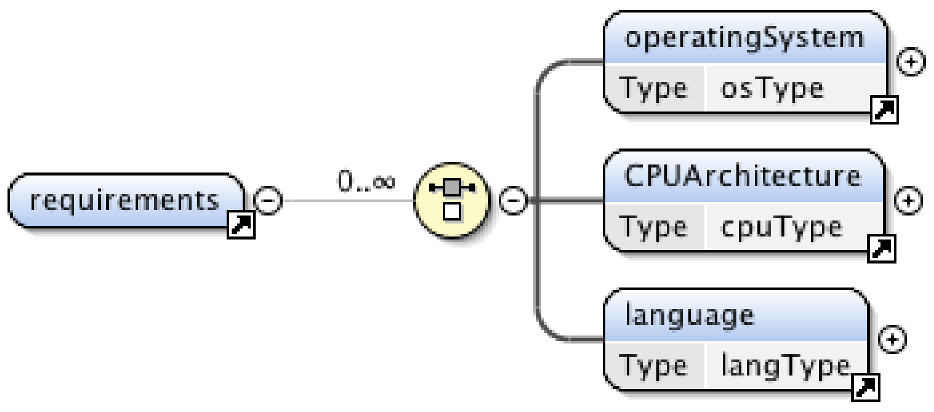
A special case for model execution in TUMOR is where implementations have been designed to read in the parameters list as raw XML. In these cases, the command-line is modified to use a keyword argument to indicate that a parameter file is to be read by the model execution command. The parameter file is used to specify run-time values. For example, we may have a file `input_parameters.xml` with the contents:

```
<parameters>
     <in name="inputA" optional="false">
          <value type="double">0.5</value>
     </in>
</parameters>
```

This file indicates that the input parameter `inputA` has the value 0.5. The corresponding command specification to execute this:

```
<command>mymodel.exe $XML_PARAMETERS</command>
```

When the **`$XML_PARAMETERS`** keyword is present in the command-line, it indicates that instead of mapping parameters to command-line arguments through using the parameter substitution operator, the executable binary or script on the command-line is able to read in a parameters file such as the above example. In this case, any execution system will ensure that the XML parameters fed into the model implementation corresponds to the parameters specified in the TumorML model description, and any other parameter substitution operators used on the **`<command>`** specification are ignored.

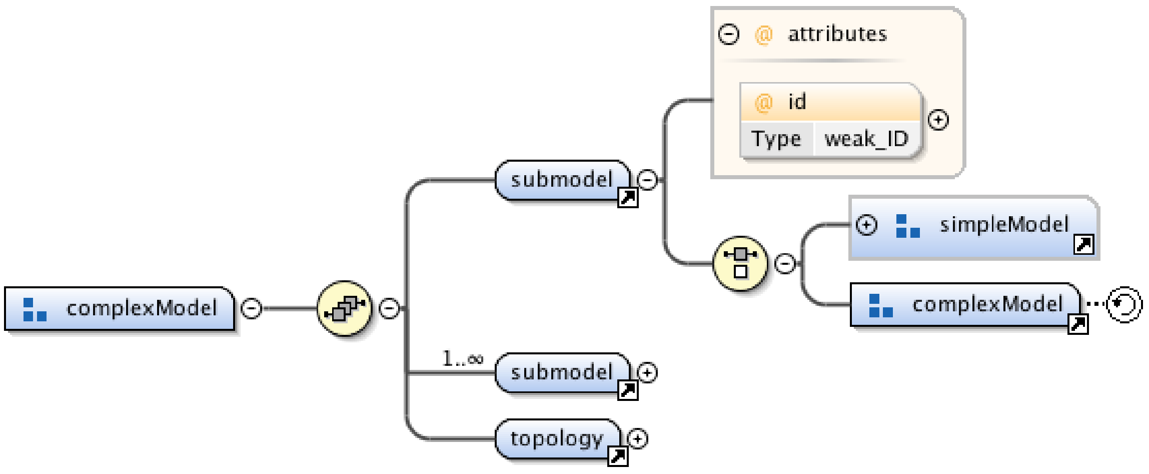


The **`<requirements>`** element contains metadata relating to the execution requirements of the model implementation. Here we specify three requirements that are taken from JSDL: operating system, CPU architecture and programming languages. This information will be used by any execution system to determine the correct platform, including any language interpreters and/or compilers, to execute the model implementation.

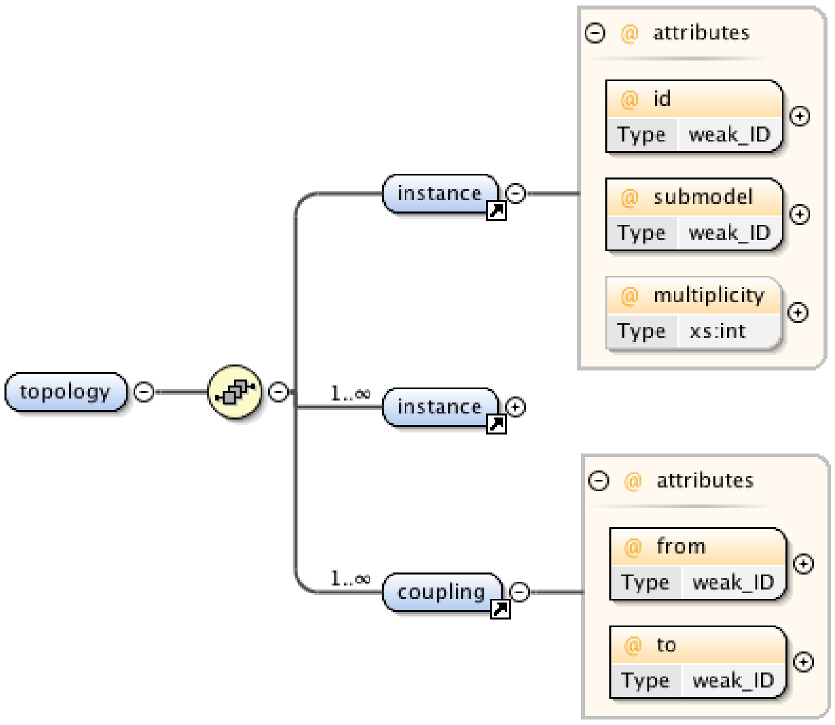
Complex Models

Complex Models represent sets of models coupled together via their computational interfaces. Like with Simple Models, Complex Models are defined within the **`<model>`** element, however have a different structure.

- **`<submodel>`** - Each complex model consists of at least two **`<submodel>`** elements. Each sub model has an **`id`** and can either be a Simple Model or another Complex Model. This recursive pattern puts no constraint on the number of levels of complexity a model may be composed of.
- **`<topology>`** - A description of how the sub models are connected via their interfaces is enclosed in the **`<topology>`** element. This element is mandatory.



The **`<topology>`** describes how sub models are connected, and this is specified as a list of named model instances and a graph of connected instance parameters.



Each **`<instance>`** has its own ID, references a sub model ID, and possesses a multiplicity that defines the number of runtime instances of the model that should be present. The

`<coupling>` is specified **`from`** and **`to`** references to instance ID and parameter ID combinations in the form:

```
coupling-from ::= instance-id "." parameter-name
coupling-to ::= instance-id "." parameter-name
```

For example, if we have an **`<instance>`** with ID *`A`* and the referenced model has an <**`out>`** parameter *`p2`*, with an **`<instance>`** *`B`* and its referenced model has an **`<in>`** parameter *`p1`* a valid coupling might be *`from A.p2 to B.p1`*, i.e.

```
<submodel name="modelA">
     …
     <parameters>
          <in name="p1"><value type="double" /></in>
          <out name="p2"><value type="double" /></out>
     </parameters>
     …
</submodel>
<topology>
     <instance id="A" submodel="modelA" />
     <instance id="B" submodel="modelA" />
     <coupling from="A.p2" to="B.p1"
</topology>
```

There may only be one **`<topology>`** element per model where there must be two or more **`<instance>`** elements and at least one **`<coupling>`** element. All attributes of **`<instance>`** and **`<coupling>`** are mandatory.

4 Appendix I - Abbreviations and acronyms

ACGT – Advancing Clinico Genomic Trials on Cancer

caBIG – cancer Biomedical Informatics Grid

ContraCancrum - Clinically Oriented Translational Cancer Multilevel Modelling

CViT – Center for the Development of a Virtual Tumor

DMR – Digital Model Repository

EC – European Commission

JSDL – Job Submission Description Language

MathML – Mathematical Markup Language

MGH – Massachusetts General Hospital

MML – Multiscale Moeling Language

NCI – National Cancer Institute

NIH – National Institute of Health

RDF – Resource Description Framework

SBML – Systems Biology Modelling Language

TUMOR – Transatlantic Tumor Model Repositories

W3C – World Wide Web Consortium

WSDL – Web Services Description Language

XML – Extensible Markup Language

xMML – XML Multiscale Modeling Language

XSD – XML Schema Document

5 Appendix II – TumorML v1.0 RelaxNG Schema

```
# RELAX NG schema for TumorML 1.0
# Author: David Johnson
# Created with oXygen XML editor

namespace tumorml = "http://www.tumor-project.eu/tumorml/1.0"

# TUMOR repository metadata value types

mathType =
      string "discrete"
    | string "continuous"
    | string "hybrid"
biocomplexityDirectionType =
      string "topDown"
    | string "bottomUp"
    | string "middleOut"
cancerType =
      string "glioma"
    | string "nephroblastoma"
    | string "lung"
    | string "breast"
materializationType =
      string "solid"
    | string "liquid"
homogeneityType =
      string "homogeneous"
    | string "non-homogeneous"
imageBasedDetectabilityType =
      string "imageable"
    | string "non-imageable"

# JSDL types
# n.b. Restricted due to DB schema restrictions

osType =
      string "unknown"
    | string "linux"
    | string "winNT"
    | string "macOS"
    | string "winXP"
    | string "winVista"
    | string "win7"
    | string "solaris"
    | string "sunOS"
    | string "javaVM"
cpuType =
      string "sparc"
    | string "powerpc"
    | string "x86"
    | string "x86_32"
    | string "x86_64"
    | string "parisc"
    | string "mips"
    | string "ia64"
    | string "arm"
    | string "other"
langType =
      string "c"
    | string "cpp"
    | string "java"
    | string "python"
    | string "perl"
    | string "fortran"
    | string "bash"
    | string "multiple"

commandLineDataTypes =
      string "int"
    | string "double"
    | string "filename"
    | string "string"

# some simple Dublin Core elements
# n.b. Simplified due to DB schema restrictions

dc_elements = (
    element title { text }
  & element creator { person }?
  & element description { text }?
  & element publisher { person }?
  & element contributor { person }*
  & element date { xsd:dateTime }
)

# Until we define our own ID regex and incorporate Schematron rules for
uniqueness,
# we will use regular XSD unqualified names as IDs and assume no semantic
checks.

weak_ID = xsd:NCName

# some simple xCard (vCard) elements describing an individual person,
including a TUMOR user ID
# n.b. Simplified due to DB schema restrictions

person = element person {
    attribute id { weak_ID }
  , element fullname { text }?
  , element name {
        element surname { text }
      , element given { text }
      , element prefix { text }*
      , element suffix { text }*
    }?
  , element email { text }*
  , element org { text }*
}

# TUMOR repository specific metadata
# n.b. Based on TUMOR DB schema
tumor_elements = (
    element math { mathType | text }?
  & element biocomplexityDirection { biocomplexityDirectionType | text }?
  & element cancer { cancerType | text }?
  & element materialization { materializationType | text }?
  & element homogeneity { homogeneityType | text }?
  & element imageBasedDetectability { imageBasedDetectabilityType | text }?
  & element freeGrowth { xsd:boolean }?
  & element treatmentIncluded { xsd:boolean }?
)

# External references
# n.b. Based on TUMOR DB schema

# bibType is based on BibTeX
bibType = string "article"
    | string "book"
    | string "booklet"
    | string "manual"
    | string "techreport"
    | string "mastersthesis"
    | string "phdthesis"
    | string "inbook"
    | string "incollection"
    | string "proceedings"
    | string "inproceedings"
    | string "conference"
    | string "unpublished"
    | string "misc"

reference = element reference {
    element title { text }
  & element source { text }?
  & element creator { text }?
  & element citation { text }?
  & element type { bibType | text }
}

# SI unit prefixes based on NIST Reference
# http://physics.nist.gov/cuu/Units/prefixes.html

siUnitPrefixTypes =
    string "yotta"
  | string "zetta"
  | string "exa"
  | string "peta"
  | string "tera"
  | string "giga"
  | string "mega"
  | string "kilo"
  | string "hecto"
  | string "deka"
  | string "deci"
  | string "centi"
  | string "milli"
  | string "micro"
  | string "nano"
  | string "pico"
  | string "femto"
  | string "atto"
  | string "zepto"
  | string "yocto"

# SI base units based on NIST Reference
# http://physics.nist.gov/cuu/Units/current.html

siBaseUnitTypes =
    string "meter"
  | string "kilogram"
  | string "second"
  | string "ampere"
  | string "kelvin"
  | string "mole"
  | string "candela"

# Some extra units accepted for use but not part of SI based on NIST
Reference
# http://physics.nist.gov/cuu/Units/outside.html + gram

siExtraUnitTypes =
    string "minute"
  | string "hour"
  | string "day"
  | string "gram"

# Combined lists of unit types used in TumorML

siUnitTypes = siBaseUnitTypes | siExtraUnitTypes
```

```
# Each TumorML model has an ID to lookup in the repository

start = tumorml
tumorml_id = attribute id { weak_ID }

# Basic structure is a model ID, header containing metadata, followed by
the model description

tumorml = element tumorml {
    tumorml_id
  , header
  , model
}

# The header contains metadata as:
# * Dublin Core elements
# * TUMOR specific elements
# * Bibliographical references (optional)

header = element header {
    dc_elements
  & tumor_elements
  & reference*
}

# The model description contains either:
# 1) a simple model as
#      * One or more implementations
#      * A input/output parameters profile
# or
# 2) a complex model as
#      * Multiple submodels
#      * A model instance topology

model = element model {
    simpleModel
  | complexModel
}

simpleModel = (
    parameters
  , implementation+
)

complexModel = (
    submodel
  , submodel+
  , topology
)

# Each implementation contains:
#  * Metadata as Dublin Core and any number of bibliographic references
(optional)
#  * A package file name
#  * A command line to initiate model execution
#  * A list of runtime environment requirements

implementation = element implementation {
    attribute id { weak_ID }
  & dc_elements
  & reference*
  & package
  & command
  & requirements
}

# A package consists of a package name and a list of files

package = element package {
    attribute name { text }
  & attribute checksum { text }
  & file+
}

# A file has a name, a full path relative to the package root, and a
checksum (optional)

file = element file {
    attribute name { text } # replace with a regex describing valid
filenames
  & attribute source { text } # replace with a regex describing valid paths
  & attribute checksum { text }?
}

# A command is a simple string for now. Will replace with a suitable regex

command = element command { text }

# The requirements include:
#   * Operating system (optional)
#   * CPU architecture (optional)
#   * Languages that need to be supported (optional, can be multiple)

requirements = element requirements{
    element operatingSystem{ osType }?
  & element CPUArchitecture{ cpuType }?
  & element language { langType }*
}

# Each submodel has a unique ID and:
#   * All the elements of a simple model
# or
#   * All the elements of a complex model

submodel = element submodel {
    (
        attribute id { weak_ID }
      , simpleModel
    )
  | complexModel
}

# A topology declares a 2 or more instances and at least one coupling of
instance parameters together

topology = element topology {
    instance, instance+
  , coupling+
}

# An instance has a unique ID, references a submodel by ID and has a
multiplicity (currently fixed)
```

```
instance = element instance {
    attribute id { weak_ID }
  & attribute submodel { weak_ID }
  & attribute multiplicity { xsd:int }?
}

# A coupling refers to a specific parameter of two different submodels
#  * from submodel.parameter
#  * to submodel.parameter

coupling = element coupling {
    attribute from { weak_ID }
  & attribute to { weak_ID }
}

# The parameters element defines at least on input parameter and multiple
output parameters
# n.b. There must be at least one input parameter

parameters = element parameters {
    element in { attlist.parameter }+
  & element out { attlist.parameter }*
}

# Parameters consist of:
#  * A unique ID/name
#  * A label (optional)
#  * A description  (optional)
#  * Whether or not this parameter is optional (optional)
#  * A value descriptor (optional)
#      * The descriptor contains a datatype, optional default value, and
optional value range
#  * A unit constraint, based on SI prefix and base units (optional)
#
# regardless of whether they are inputs or outputs

attlist.parameter = (
    attribute name { weak_ID }
  & attribute label { text }?
  & attribute optional { xsd:boolean }?
  & element description { text }?
  & element value {
      attribute type { commandLineDataTypes }
    & attribute \default { text }?
    & element range {
        element min {
            attribute inclusive { xsd:boolean }?
          & text
        }?
      & element max {
            attribute inclusive { xsd:boolean }?
          & text
        }?
    }
    & element unit {
        attribute prefix { siUnitPrefixTypes }?
      & attribute units { siUnitTypes }
    }?
  }
)
```

6 Appendix III – TumorML v1.0 XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  XML schema for TumorML 1.0
  Author: David Johnson
  Created with oXygen XML editor, converted from RelaxNG compact syntax
schema using Trang.
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.tumor-project.eu/tumorml/1.0"
elementFormDefault="qualified">
  <!-- TUMOR repository metadata value types -->
  <xs:simpleType name="mathType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="discrete"/>
      <xs:enumeration value="continuous"/>
      <xs:enumeration value="hybrid"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="biocomplexityDirectionType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="topDown"/>
      <xs:enumeration value="bottomUp"/>
      <xs:enumeration value="middleOut"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="cancerType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="glioma"/>
      <xs:enumeration value="nephroblastoma"/>
      <xs:enumeration value="lung"/>
      <xs:enumeration value="breast"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="materializationType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="solid"/>
      <xs:enumeration value="liquid"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="homogeneityType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="homogeneous"/>
      <xs:enumeration value="non-homogeneous"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="imageBasedDetectabilityType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="imageable"/>
      <xs:enumeration value="non-imageable"/>
    </xs:restriction>
  </xs:simpleType>
  <!--
    JSDL types
    n.b. Restricted due to DB schema restrictions
  -->
  <xs:simpleType name="osType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="unknown"/>
      <xs:enumeration value="linux"/>
      <xs:enumeration value="winNT"/>
      <xs:enumeration value="macOS"/>
      <xs:enumeration value="winXP"/>
      <xs:enumeration value="winVista"/>
      <xs:enumeration value="win7"/>
      <xs:enumeration value="solaris"/>
      <xs:enumeration value="sunOS"/>
      <xs:enumeration value="javaVM"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="cpuType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="sparc"/>
      <xs:enumeration value="powerpc"/>
      <xs:enumeration value="x86"/>
      <xs:enumeration value="x86_32"/>
      <xs:enumeration value="x86_64"/>
      <xs:enumeration value="parisc"/>
      <xs:enumeration value="mips"/>
      <xs:enumeration value="ia64"/>
      <xs:enumeration value="arm"/>
      <xs:enumeration value="other"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="langType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="c"/>
      <xs:enumeration value="cpp"/>
      <xs:enumeration value="java"/>
      <xs:enumeration value="python"/>
      <xs:enumeration value="perl"/>
      <xs:enumeration value="fortran"/>
      <xs:enumeration value="bash"/>
      <xs:enumeration value="multiple"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="commandLineDataTypes">
    <xs:restriction base="xs:string">
      <xs:enumeration value="int"/>
      <xs:enumeration value="double"/>
      <xs:enumeration value="filename"/>
      <xs:enumeration value="string"/>
    </xs:restriction>
  </xs:simpleType>
  <!--
    some simple Dublin Core elements
    n.b. Simplified due to DB schema restrictions
  -->
  <xs:group name="dc_elements">
    <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="title"/>
        <xs:element ref="creator"/>
        <xs:element ref="description"/>
        <xs:element ref="publisher"/>
        <xs:element ref="contributor"/>
        <xs:element ref="date"/>
      </xs:choice>
    </xs:sequence>
  </xs:group>
  <xs:element name="title" type="xs:string"/>
  <xs:element name="creator" type="person"/>
  <xs:element name="description" type="xs:string"/>
  <xs:element name="publisher" type="person"/>
  <xs:element name="contributor" type="person"/>
  <xs:element name="date" type="xs:dateTime"/>
  <!--
    Until we define our own ID regex and incorporate Schematron rules for
uniqueness,
    we will use regular XSD unqualified names as IDs and assume no semantic
checks.
  -->
  <xs:simpleType name="weak_ID">
    <xs:restriction base="xs:NCName"/>
  </xs:simpleType>
  <!--
    some simple xCard (vCard) elements describing an individual person,
including a TUMOR user ID
    n.b. Simplified due to DB schema restrictions
  -->
  <xs:complexType name="person">
    <xs:sequence>
      <xs:element ref="person"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" ref="fullname"/>
        <xs:element minOccurs="0" ref="name"/>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="email"/>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="org"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="weak_ID"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="fullname" type="xs:string"/>
  <xs:element name="name">
    <xs:complexType>
```

```
      <xs:sequence>
        <xs:element ref="surname"/>
        <xs:element ref="given"/>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="prefix"/>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="suffix"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="surname" type="xs:string"/>
  <xs:element name="given" type="xs:string"/>
  <xs:element name="prefix" type="xs:string"/>
  <xs:element name="suffix" type="xs:string"/>
  <xs:element name="email" type="xs:string"/>
  <xs:element name="org" type="xs:string"/>
  <!--
    TUMOR repository specific metadata
    n.b. Based on TUMOR DB schema
  -->
  <xs:group name="tumor_elements">
    <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="math"/>
        <xs:element ref="biocomplexityDirection"/>
        <xs:element ref="cancer"/>
        <xs:element ref="materialization"/>
        <xs:element ref="homogeneity"/>
        <xs:element ref="imageBasedDetectability"/>
        <xs:element ref="freeGrowth"/>
        <xs:element ref="treatmentIncluded"/>
      </xs:choice>
    </xs:sequence>
  </xs:group>
  <xs:element name="math" type="xs:string"/>
  <xs:element name="biocomplexityDirection" type="xs:string"/>
  <xs:element name="cancer" type="xs:string"/>
  <xs:element name="materialization" type="xs:string"/>
  <xs:element name="homogeneity" type="xs:string"/>
  <xs:element name="imageBasedDetectability" type="xs:string"/>
  <xs:element name="freeGrowth" type="xs:boolean"/>
  <xs:element name="treatmentIncluded" type="xs:boolean"/>
```

```
<!--
  External references
  n.b. Based on TUMOR DB schema
-->
<!-- bibType is based on BibTeX -->
<xs:simpleType name="bibType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="article"/>
    <xs:enumeration value="book"/>
    <xs:enumeration value="booklet"/>
    <xs:enumeration value="manual"/>
    <xs:enumeration value="techreport"/>
    <xs:enumeration value="mastersthesis"/>
    <xs:enumeration value="phdthesis"/>
    <xs:enumeration value="inbook"/>
    <xs:enumeration value="incollection"/>
    <xs:enumeration value="proceedings"/>
    <xs:enumeration value="inproceedings"/>
    <xs:enumeration value="conference"/>
    <xs:enumeration value="unpublished"/>
    <xs:enumeration value="misc"/>
  </xs:restriction>
</xs:simpleType>
<xs:element name="reference">
  <xs:complexType>
    <xs:all>
      <xs:element ref="title"/>
      <xs:element minOccurs="0" ref="source"/>
      <xs:element minOccurs="0" name="creator" type="xs:string"/>
      <xs:element minOccurs="0" ref="citation"/>
      <xs:element ref="type"/>
    </xs:all>
  </xs:complexType>
</xs:element>
<xs:element name="source" type="xs:string"/>
<xs:element name="citation" type="xs:string"/>
<xs:element name="type" type="xs:string"/>
<!--
  SI unit prefixes based on NIST Reference
  http://physics.nist.gov/cuu/Units/prefixes.html
```

```
-->
<xs:simpleType name="siUnitPrefixTypes">
  <xs:restriction base="xs:string">
    <xs:enumeration value="yotta"/>
    <xs:enumeration value="zetta"/>
    <xs:enumeration value="exa"/>
    <xs:enumeration value="peta"/>
    <xs:enumeration value="tera"/>
    <xs:enumeration value="giga"/>
    <xs:enumeration value="mega"/>
    <xs:enumeration value="kilo"/>
    <xs:enumeration value="hecto"/>
    <xs:enumeration value="deka"/>
    <xs:enumeration value="deci"/>
    <xs:enumeration value="centi"/>
    <xs:enumeration value="milli"/>
    <xs:enumeration value="micro"/>
    <xs:enumeration value="nano"/>
    <xs:enumeration value="pico"/>
    <xs:enumeration value="femto"/>
    <xs:enumeration value="atto"/>
    <xs:enumeration value="zepto"/>
    <xs:enumeration value="yocto"/>
  </xs:restriction>
</xs:simpleType>
<!--
  SI base units based on NIST Reference
  http://physics.nist.gov/cuu/Units/current.html
-->
<xs:simpleType name="siBaseUnitTypes">
  <xs:restriction base="xs:string">
    <xs:enumeration value="meter"/>
    <xs:enumeration value="kilogram"/>
    <xs:enumeration value="second"/>
    <xs:enumeration value="ampere"/>
    <xs:enumeration value="kelvin"/>
    <xs:enumeration value="mole"/>
    <xs:enumeration value="candela"/>
  </xs:restriction>
</xs:simpleType>
```

```
  <!--
    Some extra units accepted for use but not part of SI based on NIST
Reference
    http://physics.nist.gov/cuu/Units/outside.html + gram
  -->
  <xs:simpleType name="siExtraUnitTypes">
    <xs:restriction base="xs:string">
      <xs:enumeration value="minute"/>
      <xs:enumeration value="hour"/>
      <xs:enumeration value="day"/>
      <xs:enumeration value="gram"/>
    </xs:restriction>
  </xs:simpleType>
  <!-- Combined lists of unit types used in TumorML -->
  <xs:simpleType name="siUnitTypes">
    <xs:union memberTypes="siBaseUnitTypes siExtraUnitTypes"/>
  </xs:simpleType>
  <!-- Each TumorML model has an ID to lookup in the repository -->
  <xs:attributeGroup name="tumorml_id">
    <xs:attribute name="id" use="required" type="weak_ID"/>
  </xs:attributeGroup>
  <!-- Basic structure is a model ID, header containing metadata, followed
by the model description -->
  <xs:element name="tumorml">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="header"/>
        <xs:element ref="model"/>
      </xs:sequence>
      <xs:attributeGroup ref="tumorml_id"/>
    </xs:complexType>
  </xs:element>
  <!--
    The header contains metadata as:
    * Dublin Core elements
    * TUMOR specific elements
    * Bibliographical references (optional)
  -->
  <xs:element name="header">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
```

```
        <xs:group ref="dc_elements"/>
        <xs:group ref="tumor_elements"/>
        <xs:element ref="reference"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <!--
    The model description contains either:
    1) a simple model as
        * One or more implementations
        * A input/output parameters profile
    or
    2) a complex model as
        * Multiple submodels
        * A model instance topology
  -->
  <xs:element name="model">
    <xs:complexType>
      <xs:choice>
        <xs:group ref="simpleModel"/>
        <xs:group ref="complexModel"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:group name="simpleModel">
    <xs:sequence>
      <xs:element ref="parameters"/>
      <xs:element maxOccurs="unbounded" ref="implementation"/>
    </xs:sequence>
  </xs:group>
  <xs:group name="complexModel">
    <xs:sequence>
      <xs:element ref="submodel"/>
      <xs:element maxOccurs="unbounded" ref="submodel"/>
      <xs:element ref="topology"/>
    </xs:sequence>
  </xs:group>
  <!--
    Each implementation contains:
     * Metadata as Dublin Core and any number of bibliographic references
(optional)
```

```
     * A package file name
     * A command line to initiate model execution
     * A list of runtime environment requirements
  -->
  <xs:element name="implementation">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:group ref="dc_elements"/>
        <xs:element ref="reference"/>
        <xs:element ref="package"/>
        <xs:element ref="command"/>
        <xs:element ref="requirements"/>
      </xs:choice>
      <xs:attribute name="id" use="required" type="weak_ID"/>
    </xs:complexType>
  </xs:element>
  <!-- A package consists of a package name and a list of files -->
  <xs:element name="package">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="file"/>
      </xs:sequence>
      <xs:attribute name="name" use="required"/>
      <xs:attribute name="checksum" use="required"/>
    </xs:complexType>
  </xs:element>
  <!-- A file has a name, a full path relative to the package root, and a
checksum (optional) -->
  <xs:element name="file">
    <xs:complexType>
      <xs:attribute name="name" use="required"/>
      <xs:attribute name="source" use="required"/>
      <xs:attribute name="checksum"/>
    </xs:complexType>
  </xs:element>
  <!-- A command is a simple string for now. Will replace with a suitable
regex -->
  <xs:element name="command" type="xs:string"/>
  <!--
    The requirements include:
     * Operating system (optional)
     * CPU architecture (optional)
     * Languages that need to be supported (optional, can be multiple)
  -->
  <xs:element name="requirements">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="operatingSystem"/>
        <xs:element ref="CPUArchitecture"/>
        <xs:element ref="language"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="operatingSystem" type="osType"/>
  <xs:element name="CPUArchitecture" type="cpuType"/>
  <xs:element name="language" type="langType"/>
  <!--
    Each submodel has a unique ID and:
     * All the elements of a simple model
    or
     * All the elements of a complex model
  -->
  <xs:element name="submodel">
    <xs:complexType>
      <xs:choice>
        <xs:group ref="simpleModel"/>
        <xs:group ref="complexModel"/>
      </xs:choice>
      <xs:attribute name="id" type="weak_ID"/>
    </xs:complexType>
  </xs:element>
  <!-- A topology declares a 2 or more instances and at least one coupling
of instance parameters together -->
  <xs:element name="topology">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="instance"/>
        <xs:element maxOccurs="unbounded" ref="instance"/>
        <xs:element maxOccurs="unbounded" ref="coupling"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!-- An instance has a unique ID, references a submodel by ID and has a
multiplicity (currently fixed) -->
  <xs:element name="instance">
    <xs:complexType>
      <xs:attribute name="id" use="required" type="weak_ID"/>
      <xs:attribute name="submodel" use="required" type="weak_ID"/>
      <xs:attribute name="multiplicity" type="xs:int"/>
    </xs:complexType>
  </xs:element>
  <!--
    A coupling refers to a specific parameter of two different submodels
     * from submodel.parameter
     * to submodel.parameter
  -->
  <xs:element name="coupling">
    <xs:complexType>
      <xs:attribute name="from" use="required" type="weak_ID"/>
      <xs:attribute name="to" use="required" type="weak_ID"/>
    </xs:complexType>
  </xs:element>
  <!--
    The parameters element defines at least on input parameter and multiple
output parameters
    n.b. There must be at least one input parameter
  -->
  <xs:element name="parameters">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="in"/>
        <xs:element ref="out"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="in" type="attlist.parameter"/>
  <xs:element name="out" type="attlist.parameter"/>
  <!--
    Parameters consist of:
     * A unique ID/name
     * A label (optional)
     * A description  (optional)
     * Whether or not this parameter is optional (optional)
     * A value descriptor (optional)
         * The descriptor contains a datatype, optional default value, and
optional value range
     * A unit constraint, based on SI prefix and base units (optional)


    regardless of whether they are inputs or outputs
  -->
  <xs:complexType name="attlist.parameter">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="description"/>
      <xs:element ref="value"/>
    </xs:choice>
    <xs:attribute name="name" use="required" type="weak_ID"/>
    <xs:attribute name="label"/>
    <xs:attribute name="optional" type="xs:boolean"/>
  </xs:complexType>
  <xs:element name="value">
    <xs:complexType>
      <xs:all>
        <xs:element ref="range"/>
        <xs:element minOccurs="0" ref="unit"/>
      </xs:all>
      <xs:attribute name="type" use="required"
type="commandLineDataTypes"/>
      <xs:attribute name="default"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="range">
    <xs:complexType>
      <xs:all>
        <xs:element minOccurs="0" ref="min"/>
        <xs:element minOccurs="0" ref="max"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="min">
    <xs:complexType mixed="true">
      <xs:attribute name="inclusive" type="xs:boolean"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="max">
```

```
    <xs:complexType mixed="true">
      <xs:attribute name="inclusive" type="xs:boolean"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="unit">
    <xs:complexType>
      <xs:attribute name="prefix" type="siUnitPrefixTypes"/>
      <xs:attribute name="units" use="required" type="siUnitTypes"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```