Multi-level modeling with XML

Jens Gulden

University of Duisburg-Essen
Universitätsstr. 9, 45141 Essen, Germany
jensgulden@acm.org

Abstract. Multi-level modeling offers substantial advantages when expressing complex domain concepts, and it comes with the promise to allow a fine-grained reuse and extensibility of (domain-specific) modeling languages on different levels of conceptual abstractions. A small number of tooling environments have been proposed that support multi-level modeling. However, in order to be able to unleash the potential of multi-level models as widely accepted means to formally define complex concepts, and to realize the promises of reusability and extensibility, a representation format for multi-level models is required, which is easy to understand for modeling experts, and is independent from any specific tooling environment.
This paper proposes design principles of an XML format for representing multi-level models as XML documents. The approach is realized as a lightweight extension to the XML Schema Definition (XSD) standard, using built-in XML language extension mechanisms. The implementation of a multi-level schema validator is provided as a proof of concept.

Keywords: Multi-level modeling · XML · Document format · Exchange format.

1 XML documents with multiple levels of type abstractions

Multi-level modeling is an extension to the well-known object-oriented modeling paradigm. In multi-level modeling, it is possible to define a hierarchy of concepts in which instances of higher-level types can describe types, from which again instances can be derived, and so forth. In other words, objects that are instantiated out of classes can also serve as class-descriptions, and can further be instantiated to objects, which themselves can serve as class-descriptions from which objects can be instantiated. This supports type hierarchies across multiple levels of type-instance relationships. For example, when speaking of *Vehicle*, *Car*, *Volvo*, *Volvo MX 7*, and *Peter's new Volvo MX 7*, in this list, *Car*, *Volvo*, and *Volvo MX 7* both act as instances of the concept directly mentioned before, and as types of the concept directly mentioned afterwards.

Especially when it comes to reconstructing domains of discourse that reflect socio-technical systems with concepts that are described in human understandable natural language, the multi-level modeling paradigm has been evaluated to

enable a potentially large gain in expressivity, and to allow the reuse of modeling language concepts on multiple levels of conceptual abstraction [5].

Some tooling environments support modeling languages with multi-level capabilities, such as the Flexible Meta-Modeling and Execution Language (FMML^x) [5] from the XModeler [3,7] environment, or the Level agnostic Modeling Language (LML) used in the Melanee workbench [1].

However, one assumed reason why the multi-level modeling paradigm has not yet found wider support in science and practice, is the lack of a standardized representation of multi-level models, independent from particular modeling tools, and based on a widespread implementation language. Such a representation could serve as a means of communication among modeling experts to mutually exchange and evaluate multi-level models, and to offer collections of multi-level models in central places. In addition, a standardized representation could serve as an exchange format for multi-level models, and could foster the integration of different multi-level modeling tools to become part of an interoperable tooling environment.

One of the most widespread meta-modeling languages is XML, which provides a standard syntax to represent content, a schema definition and validation mechanism provided by the XML Schema Definition (XSD) language [11], and the query and transformation languages XPath and XSLT [8]. Given this set of existing mechanisms, XML appears suitable to define a language for representing multi-level models in a minimal-invasive, backward compatible way based on an existing standard.

This paper introduces fundamental principles for specifying multi-level models in XML that make use of existing XML technology in the described way. The elaboration provides a backward compatible schema validation mechanism to validate the formal semantics of multi-level models, i. e., to determine whether all entities in a multi-level model are valid according to the multi-level type definitions provided by their ancestor entities. The introduced design principles can serve as a basis for specifying an implementation-independent representation format for multi-level models that is readable by human experts, and an exchange format between different tooling environments for multi-level modeling.

Two central research questions are examined in this paper, which are to be answered as prerequisites for enhancing XML with multi-level modeling capabilities. These research questions are:

Q1: *How can XML be extended in a way that both instance characteristics and type specification characteristics can be represented by XML entity elements?*

Q2: *How can the central mechanism of schema validation of XML documents be adapted to multi-level XML?*

The questions are addressed by extending the existing XML language specification and validation mechanisms to express a core set of multi-level characteristics, which are supported by selected multi-level modeling approaches.

The key idea of the proposed approach lies in merging XML document instances with XML schema definitions, and add a notion of delayed instantiation

semantics, also known as "intrinsic" features [5] or "potency" [1], to the schema definition language. Both extensions together allow to express the static semantics of multi-level models in XML, i. e., they allow to create XML models in which any modeled entity can simultaneously act as an instance and a type declaration, and entities of any higher abstraction level can influence the type characteristics of lower-level instances. The paper proposes a lean set of syntactical enhancements to XML for expressing multi-level models, and it comes with the implementation of a schema validator which converts a multi-level XML document to an XML Schema Definition (XSD), that is subsequently used to validate the model document using a regular XSD validator.

The remainder of the paper is structured as follows: Sect. 2 elaborates modifications to the original XML declaration mechanisms that allow for the implementation of multi-level model characteristics in XML, and it introduces the prototype implementation of a multi-level schema validator. An example that demonstrates the presented approach is shown in Sect. 3. The final Sect. 4 concludes the work.

2 Extending XML with multi-level entities

2.1 Language design

To represent multi-level models in an XML document format, a number of essential requirements guide the design of the representation language. The central requirements and design decisions proposed for MXML, which is the prototypical language developed here, are stated in the following.

Req. 1 *Combine type definitions and instance entity definitions*
This is the most important requirement to be fulfilled for realizing multi-level modeling with XML. In XML, instances and types can be mixed by allowing a `<xsd:complexType> ... </xsd:complexType>` element to occur as an optional child element in any XML element of a multi-level XML document. Like any cleanly defined XML dialect, XSD is defined in its own namespace. Mixing it with content from other namespaces will thus not influence the processing of any other document content, which is an inherent extensibility feature of XML.

Req. 2 *Provide multi-level features: abstraction levels, intrinsic features*
Additional information that is required for specifying multi-level models is provided through the use of attributes with the `ml:` namespace prefix. They integrate with existing document content in the same orthogonal manner as inserted XSD schema fragments. The attributes currently suggested as multi-level extensions to any XML entity element are:
`ml:level` *(optional)* The absolute abstraction level of the entity. If a meta-entity is specified using `ml:of`, this defaults to level-1 of the meta-class.
`ml:of` *(optional)* The meta-entity of the entity. If not specified, the entity is considered to instantiate the default XML entity meta-type.

`ml:abstract` *(optional)* Flag to indicate whether the entity should be considered abstract. In this case, any attributes, children elements, or text-content will be ignored by the schema validator and the entity serves as a class declaration only.

The following attributes are suggested as extensions to the `<xsd:element>` and `<xsd:attribute>` elements:

`ml:intrinsic` *(optional)* Absolute level on which the declaration should be instantiated.

`ml:potency` *(optional)* Relative level, counting from the current, on which the declaration should be instantiated.

The former two attributes are mutually exclusive. If none of these attributes is provided, the type declaration element, like in conventional XSD, takes effect immediately during the next instantiation, i. e., it becomes effective for the element it is declared in.

The `ml:intrinsic` / `ml:potency` attributes are representatives for fundamental multi-level model characteristics that provide delayed instantation. This set of attributes can be refined in later versions to also cover *range* and *leap potency*, with a more fine-grained specification of *min*, *max*, and *depth* values.

Req. 3 *Design with minimal invasive changes to existing standards*
In order to provide a lean integration into existing XML standards, which allows an easy adaption and also keeps the implementation more efficient through reusability, the design should opt for as few modifications to the original XML language mechanisms as possible. This requirement is fulfilled in the proposed solution by reusing parts of the XSD language for specifying type characteristics of multi-level entities. When validating, XSD fragments from a multi-level document will be transformed to a standard XSD document, which is then used for validation.

Req. 4 *Implement with recursion to existing schema validation*
To implement a schema validator for MXML documents, the approach should reuse a standard XSD schema validator. This is desirable both in order to formally prove the backward compatibility of the approach, as well as for an efficient development of the MXML schema validator. The suggested approach realizes this by using a two-phase validation process, in which first the MXML document is transformed to a temporary XSD schema document and a plain XML representation, and then an existing XSD validator is invoked which applies the schema to the plain XML.

A few additional conventions apply to this proposed language design. If multiple element instances of the same name contain schema definitions, the definition statements are syntactically merged together in the same way as if they were all given in only one schema definition. No element instance has priority over another in specifying schema definitions. It may, however, turn out to be a good notation style to locate an element's schema definition in only the first element by convention, which can be expected to increase the readability and maintainability of manually edited MXML documents.

On the top nesting level of the element hierarchy in MXML documents, any regular XSD fragment may be placed. This allows to include auxiliary non-multi-level schema declarations, e. g., an enumeration type for defining the value range of an attribute used by a multi-level entity.

2.2 Syntax proposal

A syntax for interweaving XML instance content with XSD schema declarations in MXML is proposed in the following. According to common grammar notation conventions, parts enclosed in `[...]` are optional, parts in `{...}` can occur zero to any times, and a vertical bar | indicates a choice between either the left- or the right-hand side. The syntax of any XML element is redefined as follows:

```
<element { attrN="..." } [ ml:level="level" ] [ ml:of="parent" ]
                                               [ ml:abstract="[true]" ] >
  [ <xsd:complexType>
       ... schema declarations ...
    </xsd:complexType> ]
  { <child {attrN="..."} > ... </child> }
  [ text content ]
</element>
```

This means, any XML element can at first act as a regular element instance with the ability to have attributes, nested children elements, and text content. In addition, any XML element can also contain a schema definition fragment, which defines the formal semantics of instances of the element. The proposed syntax of XSD schema definition fragments to be included in any multi-level XML element is shown in the following:

```
<xsd:complexType>
    <xsd:sequence>
      { <xsd:element { attrN="..." }
                     [ ml:intrinsic="level" | ml:potency="levels" ] > ...
       </xsd:element> }
    </xsd:sequence>
  { <xsd:attribute { attrN="..." }
                   [ ml:intrinsic="level" | ml:potency="levels" ] > ...
    </xsd:attribute> }
</xsd:complexType>
```

The structure-descriptor `<xsd:sequence> ... </xsd:sequence>` around the children element type declarations is included for compatibility with XSD, but does not carry meaning in the current version of MXML. Any structure-descriptor will internally be treated equivalent to `<xsd:choice minOccurs="0" maxOccurs="unbounded"> ... </xsd:choice>`, which means that children elements may occur in any order and for an arbitrary number of times. This is syntactically compatible to any available structure-descriptor in XSD, i. e., children elements can be arranged in any structure, but the desired structure must be enforced by convention and will not formally be validated.

2.3 Schema validation

As a proof of concept, a schema validator that takes into account the multi-level nature of MXML documents has been developed.* According to Req. 4, the implementation of the schema validator is kept as lean as possible and reuses existing XSD schema validation functionality.

The multi-level approach allows documents to mix instance content and schema definitions. Existing XML validation approaches, however, assume that instance content (`.xml`) and schema definitions (`.xsd`) are stored in separate documents. To trace back multi-level document validation to traditional schema validation, it is thus required to separate the schema information in a multi-level document from the instance content. This is done by using two XSLT transformations that take as input an MXML document, and temporarily generate one XML instance document and one XML Schema Definition document out of it. The instance document can then be validated against the schema document using an existing XSD validator.

The documents involved in an MXML application and their interrelationships are shown in Fig. 1. To validate an MXML document `doc.xml`, the XSLT transformation `mlschema.xsl` is applied to it, which generates a temporary XML Schema Definition document `doc-schema.xsd`. The `mlinstance.xsl` transformation is applied to generate a plain XML instance document without multi-level specification parts. This document is then validated against the generated schema.

A fundamental advantage of this validation approach is that it proves the backward compatibility of MXML by construction, because the relationships between the original MXML document and the temporary non-MXML instance and schema documents are formally defined by the `mlschema.xsl` and `mlinstance.xsl` transformations. One disadvantage, however, is that in case a validation error occurs, it might be challenging to trace an error message that was issued for the transformed documents, back to the origin of an error in the MXML document. A solution to this would be to supplement the schema validator with another XSLT transformation that checks additional validity constraints on the MXML document. An example of such a constraint would be to ensure that an entity on level `m` must not contain intrinsic attributes with an instantiation level greater than `m`. This option is not implemented yet, which is why the transformation `mlvalidate.xsl` is displayed with a dashed border in Fig. 1.

2.4 Related work

The fundamental idea behind multi-level modeling is based on the combined notion of entities carrying instance properties and type declarations at the same time. This kind of entities has been discussed in several works on extensions

*The schema validator implementation and the example artifacts are available at `https://www.wi-inf.uni-due.de/FGFrank/download/mxml-0.1.0.zip`.

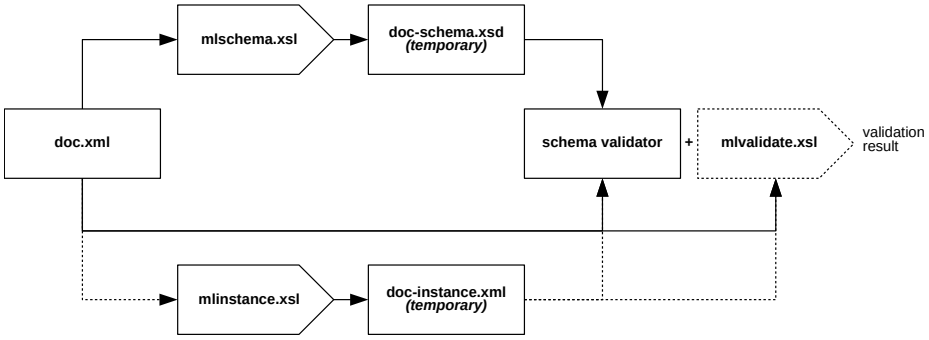

Fig. 1. Documents architecture and validation process

to the object-oriented modeling paradigm, e. g., by [10] which introduces the concept of "Power Types" as a kind of entities with both objects and class characteristics, and [6] which proposes the use of "Clabjects".

Modeling approaches that incorporate the notion of such entities into a multi-level modeling language are proposed by, e. g., [2] and [5]. Corresponding tooling support that offers functionality for creating and using multi-level models has been provided, e. g., by the MELANEE modeling environment [1], XMODELER [3], and METADEPTH [4]. While in principle comparable to each other, these approaches each follow proprietary conceptualizations with respect to the languages they provide and the implementations of the underlying tooling support. This is exactly the reason why a unified document representation and a tool-independent exchange format is useful to communicate about essential concepts, and to provide interoperability among tools.

An approach which combines the XML Metadata Interchange (XMI) format with ECORE representations of meta-models is MULTECORE [9]. This is an attractive alternative to using plain XML for expressing multi-level models, because it integrates into the widespread ECLIPSE MODELING FRAMEWORK (EMF). However, MULTECORE achieves downward compatibility to ECORE by storing representations of each abstraction level individually in one ECORE model. The extension of ECORE to a multi-level architecture is achieved by organizing multiple single-level models to appear as one multi-level model. This is contrary to the goal of this work, which aims at reflecting multi-level characteristics with internal language extensions to XML.

One should be aware that the term "multi-level XML document" is sometimes used in XML teaching literature with a different meaning. Several web pages about XML basics use the term "multi-level" to refer to XML documents with a nesting depth of children elements greater than one. That means, whenever an element that is a direct child of the document root element has at least one more child element nested inside, the XML document is considered to be "multi-level". This, of course, has nothing to do with the notion of multiple abstraction levels for type descriptions as it is addressed in this work.

3 Example

The following listing shows an MXML document that specifies ingredients of a cooking recipe. A general type `Ingredient` is refined into the types `VegetarianIngredient` and `Spice`, the latter of which gets further refined into `Salt` and `Pepper`.

```
<ingredients xmlns:ml="urn:ml" xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <ingredient ml:abstract="yes">
    <xs:complexType>
      <xs:sequence>
        <!-- <xs:element ref="name"/> (implicit by language) -->
        <xs:element name="price" type="xs:string" ml:potency="-1"/>
        <xs:element name="tasteDescription" type="xs:string" ml:potency="2"/>
      </xs:sequence>
      <xs:attribute name="healthy" type="xs:boolean" ml:potency="1"/>
    </xs:complexType>
  </ingredient>

  <vegetarianIngredient ml:of="ingredient" healthy="true">
    <xs:complexType>
      <xs:sequence>
        <xs:attribute name="healthy" type="xs:boolean"/>
      </xs:sequence>
    </xs:complexType>
  </vegetarianIngredient>

  <spice ml:of="ingredient">
    <xs:complexType>
      <xs:attribute name="appetizing" type="xs:boolean"/>
      <xs:attribute name="sudatory" type="xs:boolean"/> <!-- makes you sweat -->
      <xs:attribute name="digestive" type="xs:boolean"/>
    </xs:complexType>
  </spice>

  <pepper ml:of="spice" color="red">
    <xs:complexType>
      <xs:attribute name="color" type="pepperColor"/>
    </xs:complexType>
    <tasteDescription>very tasty!</tasteDescription>
  </pepper>

  <!-- this is a regular level-agnostic auxiliary XML Schema definition -->
  <xs:simpleType name="pepperColor" final="restriction">
    <xs:restriction base="xs:string">
      <xs:enumeration value="green" />
      <xs:enumeration value="black" />
      <xs:enumeration value="white" />
      <xs:enumeration value="red" />
    </xs:restriction>
  </xs:simpleType>

  <salt ml:of="spice">
    <xs:complexType>
      <xs:attribute name="avgLifetime" type="xs:double"/>
    </xs:complexType>
    <tasteDescription>very tasty!</tasteDescription>
  </salt>

  <pepper ml:of="spice" color="green"/>

</ingredients>
```

This document can be validated using the `run-validator.xml` script from the example implementation package, which produces both a temporary `.xsd` file and a temporary `.xml` file. The former is subsequently applied as the validating schema to the latter, using a standard XSD schema validator. When run with the example, the output of the `run-validator.xml` script is empty, which indicates that no error during validation has been found and the MXML document is considered to be formally valid.

4 Conclusion and future work

This work has suggested fundamental design principles to extend XML with multi-level modeling capabilities. It includes a lean language implementation that integrates well with existing XML concepts and tools. As the language is independent from any tooling environment, it can potentially serve as an exchange format for experts to communicate about multi-level models, build shared libraries of multi-level models, and achieve interoperability between different multi-level modeling tools on the technical level.

Two research questions had initially been asked, the first about how XML can be extended in a way that both instance and type characteristics of multi-level model entities can be represented by XML elements (*Q1*), and the second on how a schema validation mechanism for multi-level XML documents can be provided, based on the existing XSD language (*Q2*). Both questions have been answered by designing an XML dialect which fulfills 4 main requirements: *Req. 1* demanded for the ability to combine type definitions and instance entity definitions in the representation format. This has been achieved by interweaving regular XML content with XSD fragments. *Req. 2* demanded for multi-level type characteristics, such as delayed instantiation, to be expressable by the language, which has been achieved by defining attributes using the `ml:` namespace prefix for describing multi-level characteristics of XML entities. These attributes extend the semantics of `<xsd:element>` and `<xsd:attribute>` declarations. This lean set of proposed language extensions also adheres to *Req. 3*, which demanded for applying minimal invasive changes to the existing XML language mechanisms. The last requirement *Req. 4* suggested to reuse existing schema validation components to implement a multi-level schema validator, which has been realized by a prototypical schema validator implementation (see Sect. 2.3).

The presented work is intended to elaborate fundamental design principles for representing multi-level models with XML. It does not claim to offer a complete exchange format for tool integration, which would require a set of significantly longer standardization documents. As a consequence, several limitations apply to the current elaboration of MXML. At first, the range of supported multi-level features is limited to a set of concepts provided by selected multi-level modeling approaches. Extending the range of supported approaches will be subject to future work. On the implementation level, the next step to improve the schema validator will be to support the use of the `ml:intrinsic` attribute to specify an absolute intrinsic level attached to `<xsd:element>` and `<xsd:attribute>`

declarations. Currently, only relative level distances specified by `ml:potency` are supported.

Behavior-related aspects, e. g., the adaptation of the XPath query language [8] or the Object Constraint Language (OCL) [12] to multi-level concepts, are out of the scope of this article and also subject to future work.

Future work should also cover the extension of MXML to a unified representation of language features from all representative multi-level modeling approaches. This could be done by assembling meta-models of the respective languages, and create a unified meta-model for MXML that incorporates all individual language approaches. To exemplify the integration capabilities of the exchange format, a set of importers and exporters should be provided for existing multi-level modeling tools, e. g., in the form of model transformations that convert between the internal representation formats of the tools and MXML.

References

1. Colin Atkinson and Ralph Gerbig. Flexible deep modeling with Melanee. In Stefanie Betz Ulrich Reimer, editor, *Modellierung 2016, 2.-4. März 2016, Karlsruhe – Workshopband*, volume 255, pages 117–122, Bonn, 2016. Ges. f. Informatik.
2. Colin Atkinson and Thomas Kühne. The essence of multilevel metamodeling. In Martin Gogolla and Cris Kobryn, editors, *UML '01 Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, pages 19–33, London, 2001. Springer UK.
3. Tony Clark and James Willans. Software language engineering with XMF and XModeler. In Marjan Mernik, editor, *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*, pages 311–340. IGI Global, 2012.
4. Juan de Lara and Esther Guerra. Deep meta-modelling with metadepth. In Jan Vitek, editor, *Objects, Models, Components, Patterns*, pages 1–20, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
5. Ulrich Frank. Multi-level modeling – toward a new paradigm of conceptual modeling and information systems design. *Business & Information Systems Engineering (BISE)*, 6(3), 2014.
6. C. A. González-Pérez and B. Henderson-Sellers. *Metamodelling for software engineering*. John Wiley, Chichester, UK and Hoboken, NJ, 2008.
7. Jens Gulden and Ulrich Frank. MEMOCenterNG – a full-featured modeling environment for organization-modeling and model-driven software development. In Pnina Soffer and Erik Proper, editors, *Proceedings of the CAiSE Forum 2010 Hammamet, Tunisia, June 9-11, 2010*, volume 592 of *CEUR Workshop Proceedings*, pages 76–83. CEUR, 2010. ISSN 1613-0073.
8. Michael Kay. *XSLT 2.0 and XPath 2.0 Programmer's Reference*. Wrox Press Ltd., Birmingham, UK, UK, 4th edition, 2008.
9. Fernando Macías, Adrian Rutle, and Volker Stolz. MultEcore: Combining the best of fixed-level and multilevel metamodelling. In *MULTI@MoDELS*, volume 1722 of *CEUR Workshop Proceedings*, pages 66–75. CEUR-WS.org, 2016.
10. James Odell. Power types. *Journal of Object Oriented Programming*, 1994.
11. Eric van der Vlist. *XML Schema*. O'Reilly, Cambridge, 2002.
12. Jos Warmer and Anneke Kleppe. *The Object Constraint Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2003.