vdmML: using XML to represent VDM on the Web.

Brian M. Matthews

Information Technology Department, CLRC Rutherford Appleton Laboratory
Chilton, Didcot, Oxfordshire, OX11 OQX, UK.
bmm@inf.rl.ac.uk

Abstract. In this paper we describe some initial investigations into the use XML to support the use of VDM. This presents a snapshot of ongoing work-in-progress, and thoughts to the future development of such work, including transformation into different formats, proof obligation generation, and integration with other formal and semi-formal methods.

1 Introduction

XML [5] is becoming established as the cornerstone of the new architecture of the World-Wide Web (WWW). XML allows user communities to define their own data formats, yet remain compatible with the rest of the information on the WWW. This allows the universal exchange of information across the web, processable via widely available tools. The software engineering is one which could benefit from the use of XML. Teams are frequently widely distributed and use different tools and need to exchange information with one another in common formats, such as XML can provide. Further there is a need for software engineering information such as specifications and designs to be integrated with other types of information, such as requirements, contracts, programs, and test suites. Again XML supports such interoperability. Thus there is a need for software engineering notations to be formulated in XML.

In this paper we propose an XML format for the VDM-SL, *vdmML*. Specifically, we represent the outer abstract syntax as given in [23]. Representing VDM using XML would give the following benefits.

- It provides a standard structure preserving exchange format between different VDM tools.
- It provides a mechanism for presenting and manipulating VDM using standard WWW tools such as web browsers, and other XML enabled tools.
- It provides a means for exchanging semantically meaningful information with other formats.
- It enables integration with other formats in documents, such as HTML for simple documents, MathML for mathematics, and SVG for diagrams.
- It enables the integration with other notations allowing the use of heterogeneous specifications.
- It eases the development of specifications collaboratively via the WWW via a common exchange format.

– VDM specifications become dynamic resources, enabling the searching for components, their reuse and modification, while allowing the definition of an audit trail capturing their interrelation.
– It provdes for independent validation as a standard format which can be sent to an independent validation service.

Such XML support could form an infrastructure for loose coupling of distributed toolkits such as the Prosper [9, 17], linking specification tools, analysis tools such requirements and traceability tools, and theorem provers.

We demonstrate some of the benefits of using XML through the transformation into different formats, and also discuss further possibilities for the use of XML. In sections 2 and 3 we give a brief introduction to XML, and also MathML, a XML based markup language for representing mathematics which is used in the definition of vdmML. In section 4 we introduce vdmML and discuss its use to express VDM. In section 5 we consider the transformation of VDM into various presentation formats for display. Finally in sections 6 and we discuss some proposals for further work on vdmML and its integration with other formats.

Some work in defining an XML description for VDM has been carried out by John Wordsworth [24]. However, our approach differs as we integrate VDM with MathML and also demonstrate the value of the XML by giving some transformations into other formats. Other related work is John Wordsworth's XML description of Z [24], and Georges Mariano's XML representation of B, within the context of the Bcaml B Parser [12, 13]. These latter two open the possibility of integration with other formats.

2 XML

The Extensible Markup Language (XML) is a profile of the Standard Generalised Markup Language (SGML) [20] developed by the World-Wide Web Consortium (W3C). XML is designed as a lightweight version of SGML for representing documents and data on the WWW. It is a *meta-language* in that it allows users to define their own markup within a generic syntax for markup; by defining new markup in a standard manner, and also formally defining the structure of the markup, interoperability can be maintained whilst allowing user flexibility. The XML activity within W3C has now expanded to a large family of related recommendations. We do not give all the details of the XML family, but note the following relevant points.

XML defines markup by the use of markup *tags* which define structured *elements* in the data. These elements can either contain text, or be nested to represent structured data. Thus, an XML representation of some elements from the well-known HTML markup would be:

```
<body>
  <h1> A Heading </h1>
  <p> A paragraph. </p>
</body>
```

This `body` element contains two child elements, `h1` and `p`, delimitted by begin and end tags. Additionally, elements can have attributes, attached to the element's begin tag, thus:

```
<h1 align='center'> A Heading </h1>
```

provides an attribute "`align`" which provides additional information which can be used by an XML processor. If the element has no child text or elements, it can be represented as a single tag, as follows:

```
<img src='picture.gif'/>
```

The structure of a particular class of documents to be used in a particular domain can be defined using a *Document Type Definition* (DTD). This defines the elements and their valid nestings used in a particular class of documents, similar to a context-free grammar, and also the valid attributes of elements and their range of values. The DTD can then be used to validate the conformance of any XML document to that document class. We give relevant parts of the formal syntax for DTDs as they are used. DTDs are inherited from SGML, and are seen as too inflexible, and also incompatible with XML Namespaces (see below). Consequently, a new mechanism for defining classes of XML documents is under development, *XML Schema* [27]. This is a much more powerful mechanism than DTDs allowing for example, stronger data-typing and inheritance. In this paper, because of space considerations, and also because XML Schemas have yet to be completed and are not as yet widely supported, we only consider XML DTDs; future development of the work would include moving to XML Schemas.

XML Namespaces [6] provide a mechanism for using elements defined in different domains within the same XML document by the addition of a *namespace prefix* to element and attribute names to prevent clashes. This provides a simple integration mechanism for different XML data formats.

3 MathML

MathML [16], originally proposed for use with HTML, but now an XML format, defines a general notation for mathematical symbols and expressions using XML. This is intended for use not only for displaying mathematics within web-browsers, but also in web-enabled mathematics processing tools, such as symbolic algebra systems, which can then process the structured data directly . MathML being implemented in tools such as Amaya [1] and Mathematica [14].

MathML has gives two levels of support for mathematics. The *presentation markup* gives a "visually oriented" view on the mathematical notation, concentrating on the physical layout of the mathematical expression on the page, similar to a LaTeX math-mode expression. For example,

$$x^2 + 2x + 1 = 0$$

is represented as:

```
<mrow>
  <mrow>
    <msup> <mi>x</mi> <mn>2</mn> </msup>
    <mo>+</mo>
    <mrow>
      <mn>2</mn>
      <mo> <mchar name='InvisibleTimes'/> </mo>
      <mi>x</mi>
    </mrow>
    <mo>+</mo>
    <mn>1</mn>
  </mrow>
  <mo>=</mo>
  <mn>0</mn>
</mrow>
```

Here, the <mrow> element defines a horizontal linear flow on the page, with <mi> delimitting identifiers, <mo> operators, and <mn> numbers. Further elements define vertical alignment for fractions, and more complex objects, such as integrals and matrices. While giving precise instructions to a rendering engine, this is inherently ambiguous with respect to the meaning of the expression. For example, the <msup> element above is used to typographically represent a superscript and there is no notion that a power operator is intended. This markup is thus is not in general suitable for passing to a mathematics processor, such as symbolic algebra engine or theorem prover.

MathML also provides a *content markup*, which represents the expression in a tree form with an intended semantics. Thus the above expression becomes:

```
<apply>
  <eq/>
  <apply>
    <plus/>
    <apply>
      <power/>
      <ci>x</ci>
      <cn>2</cn>
    </apply>
    <apply>
      <times/>
      <cn>2</cn>
      <ci>x<ci>
    </apply>
    <cn>1</cn>
  </apply>
  <cn>0</cn>
</apply>
```

Here the <apply> element is used repeatedly for the applications of operators in the expression, using the convention that the first child of apply is the operator, and the subsequent the arguments. A set of empty elements is supplied for standard operators, such as <eq/>, <plus/>, <power/> and <times/> above. These built-in operators and constructs approximately cover high-school or first year undergraduate mathematics, with a standard interpretation, and also can be modified and extended by the user to provide alternative semantics.

MathML provides some additional grammatical rules beyond those supplied as default in XML to ensure the well-formedness of such expression trees. The content markup however, relies on the processing engine to provide a default presentation; the user can provide both content and presentation markup of the same expression to be used in the different context.

We propose to use MathML to provide a base framework for expressing VDM; there is little point in reinventing the syntax for common operations and expressions, and they can then be simply extracted from vdmML documents to be passed to a MathML processor. However, MathML lacks the those constructs in VDM which are used for specification of a software system, including datatypes, state, operations and programming constructs. The approach taken here is to provide new markup in a format dedicated to VDM, vdmML, and embed within this language those MathML constructs suitable for expressions.

4 vdmML

In order to represent VDM in XML we define a DTD. As this is a context-free grammar, this is similar to the outer abstract syntax as given [23]. Here we discuss some of the features of the DTD without giving all the details.

4.1 Including vdmML in Documents

In order to include fragments of VDM within other documents, with sections of intervening text, or other XML markup-up data, the vdmML provides the <VDM> element, given in the DTD via an element declaration, as follows.

```
<!ELEMENT VDM ANY >
```

This declaration defines a new element, called VDM, together with a *content model* which defines the valid element structures which can occur within the element. In this case, `VDM` has the `ANY` content model means that any valid markup from the vdmML DTD can be placed within this element. This is useful when breaking up VDM notation throughout an XML document for presentation purposes, or within a paper, but to pass a coherent specification in vdmML we use a different mechanism.

In the latter case, the top level element for a VDM specification in vdmML is <Specification>, defined in the DTD via an *element defintion* as follows:

```
<!ELEMENT Specification (DefBlock*) >
```

The content model here defines that a `Specification` element can contain zero or more `DefBlock` elements, as signified by the "`*`" after the element name. Thus similarly to the grammar of VDM, a specification contains a series of definition blocks[1].

Typically, this element will be used as the root element in a document as follows:

```
<?xml version='1.0' ?>
<!DOCTYPE Specification SYSTEM
  "http://www.itd.clrc.ac.uk/Projects/VDMB/VDMML/vdmml.dtd" >
<Specification>
                ...
</Specification>
```

The first line declares that an XML 1.0 document is being used. The *document type declaration* (`<!DOCTYPE ... >` defines the DTD being used, here at a URL, against which the vdmML can be validated. However, frequently, we would expect the VDM specification to be embedded within another markup, such as text, or alternatively it might be supplied without a document type declaration. Then it would be appropriate to define a VDM namespace in the `Specification` element. This would then disambiguate the VDM markup from the other markup in the document. This might be as follows.

```
<vdm:Specification
  xmlns:vdm="http://www.itd.clrc.ac.uk/Projects/VDMB/VDMML"
  xmlns:mml="http://www.w3.org/TR/2000/WD-MathML2-20000328"
>
  ...
</vdm:Specification>
```

Here, two namespace prefixes are defined; "`vdm`" is defined as a prefix for elements from the vdmML domain, and "`mml`" for those elements within the MathML domain, which are embedded within vdmML. These namespace prefixes are arbitrary and can change from document to document. The URL provided with each namespace prefix provides a source for the names in the domain; this will be used by the XML Schema mechanism for validation purposes. The one supplied with MathML is the standard one supplied by the W3C; the one for vdmML refers to the CLRC website [2].

In the rest of this paper, we present both vdmML and MathML markup elements and attributes with these namespace prefixes. This is to clarify which namespace each element is derived from. Such namespace prefixes are omitted in DTD fragments.

4.2 Expressing VDM in XML

Continuing through the DTD, the definition of `Specification` is as follows.

[1] In this paper, we ignore modules in VDM.

[2] Currently this URL is arbitrary and may change, but provides a convenient unique identifier for the namespace.

```
<!ELEMENT Specification (DefBlock*) >
<!ELEMENT DefBlock (TypeDef*
                   | State
                   | ValueDef*
                   | FunDef*
                   | OpnDef*)
>
```

Thus a `Specification` element contains a sequence of `DefBlock` elements, and a `DefBlock` in turn can be either a sequence of `TypeDef` elements, a state element `State`, a sequence of `ValueDef` elements, a sequence of `FunDef` elements or a sequence of `OpnDef` elements; this is similar to the grammar of specifications in the ISO standard. Note that this definition of DefBlock cannot specify that only one State element can occur in a specification; this constraint should be expressible using XML Schema.

The rest of the DTD follows the VDM context free grammar quite closely, and we omit it for brevity. Instead we illustrate our approach through an example of a fragment of VDM.

An example operation from [2] is as follows.

```
operations

delete_instance (ir:instance_ref)
ext rd schema:schema_def
  wr model : data_def
pre ir in set dom data_def
post mk_db(schema, model) = delete_instance(mk_db(schema ,model ),ir)
```

When expressed in vdmML this becomes the following.

```
<vdm:DefBlock>
  <vdm:Operation id="delete_instance" type="implicit">
    <mml:ci>delete_instance</mml:ci>
    <vdm:Parameters>
      <vdm:Pattern><mml:ci>ir</mml:ci>
      </vdm:Pattern>
      <vdm:Type>
        <vdm:TypeName><mml:ci>instance_ref</mml:ci></vdm:TypeName>
      </vdm:Type>
    </vdm:Parameters>

    <vdm:Externals mode='rd'>
      <mml:ci>schema</mml:ci>
      <vdm:Type>
        <vdm:TypeName><mml:ci>schema_def</mml:ci></vdm:TypeName>
```

```
  </vdm:Type>
</vdm:Externals>
<vdm:Externals mode='wr'>
  <mml:ci>model</mml:ci>
  <vdm:Type>
      <vdm:TypeName><mml:ci>data_def</mml:ci></vdm:TypeName>
  </vdm:Type>
</vdm:Externals>

<vdm:PreCondition>
  <vdm:Expression>
    <mml:apply>
      <mml:in/>
      <mml:ci>ir</mml:ci>
      <mml:apply>
        <mml:csymbol mml:definitionURL="http://www.itd.clrc.ac.uk/Project
          dom
        <mml:csymbol>
        <mml:ci>data_def</mml:ci>
      </mml:apply>
    </mml:apply>
  </vdm:Expression>
</vdm:PreCondition>

<vdm:PostCondition>
  <vdm:Expression>
    <mml:apply>
      <mml:eq/>
      <mml:apply>
        <mml:ci>mk_db</mml:ci>
        <mml:ci>schema</mml:ci>
        <mml:ci>model</mml:ci>
      </mml:apply>
      <mml:apply>
        <mml:ci>delete_instance</mml:ci>
        <mml:apply>
          <mml:ci>mk_db</mml:ci>
          <mml:ci vdm:type='old'>schema</mml:ci>
          <mml:ci vdm:type='old'>model</mml:ci>
        </mml:apply>
        <mml:ci>ir</mml:ci>
      </mml:apply>
    </mml:apply>
  </vdm:Expression>
</vdm:PostCondition>
```

```
  </vdm:Operation>
</vdm:DefBlock>
```

Within this fragment, we find the operation with its parameters, external variable frame declarations, and its pre- and post-condition, represented as vdmML elements, and the . The vdmML version is much more verbose as we are explicitly representing the structural elements of the syntax. We note that:

- The operation has been given a `id` attribute for locating this operation within the document; this can be used for cross-referencing using the `ID` and `IDREF` mechanism built into XML.
- The fact that the operation is implicit is recorded as an attribute.
- The operation name is given using the `<ci>` element from MathML, defined for identifiers in content mode.
- The read/write mode for the external frame is given as an attribute. This allows the DTD to restrict it to the given two values.
- Expressions are given using MathML apply elements, as discussed below.

A prototype conversion tool has been developed which takes the VDM ASCII syntax as defined by the IFAD VDM-SL Toolkit and generates the equivalent vdmML.

4.3 Relationship to MathML

MathML provides vdmML with a base level syntax for identifiers, symbols and expressions.

Identifiers are written as <mml:ci>id</mml:ci>, taken directly from from MathML. In the above example we have added the vdmML attribute vdm:type='old' to some variables representing the pre-state variable in a postcondition, thus signifying that these are semantically related. Such mixing of attributes is accommodated by the XML Namespace mechanism although it does not conform to the MathML DTD; again, XML Schemas should allow for this.

Integers, such as 12, are interpreted as

```
<mml:cn mml:type=''integer''>12</mml:cn>
```

directly from MathML. Similarly, reals such as 12 are interpreted as

```
<mml:cn>12</mml:cn>
```

Real is the default type of number in MathML. MathML also provides boolean constants:

```
<mml:cn mml:type="constant">true</mml:cn>
and
<mml:cn mml:type="constant">false</mml:cn>.
```

For VDM's nil constant for lists etc, we use MathML's `csymbol` element to define new symbols:

```
<mml:csymbol
  mml:definitionURL="http://www.itd.clrc.ac.uk/Projects/VDMB/VDMML">
  nil
</mml:csymbol>
```

The `definitionURL` attribute here refers the user to a reference location where the definitive definition of the symbol can be found in either a machine or human readable form. Here for convenience, we have chosen to use the VDM+B Project web site as the URL for such information[3].

For unary and binary operators we use the predefined MathML operators such as `<mml:eq/>`, and `<mml:in/>` in the above example. Built in operators not predefined in MathML are again defined using the `<mml:csymbol>` element with an appropriate definitionURL, such as the `''dom''` unary operator in the above example[4].

These operators are used in conjunction with the apply operator `<mml:apply>` to make expressions in vdmML. More complicated expressions, such as conditionals, or local variables can result in MathML elements containing vdmML elements, reflecting the nesting of expressions. Again, the XML Namespace mechanism can accommodate such mixing of domains.

5 Applying transformations to vdmML

By representing VDM using vdmML, VDM specifications can be passed around the web and processed using standard XML enabled tools such as web browsers, either as a standalone document or embedded in other XML formatted data, such as document formatting. Microsoft Internet Explorer 5 (MSIE5) currently supports XML and other browsers have development versions supporting XML (such as Netscape 6). However, in order to be presented in a human readable format, a stylesheet needs to be applied to it which augments or transforms the XML document into a form presentable by the browser. By supplying different stylesheets to the same XML document, different views can be given, while maintaining their consistency with respect to the underlying data.

The W3C is defining a stylesheet language, XSL, to transform and format XML documents for presentation and to convert XML formats into other possibly non-XML formats. The first part of this is a transformation language XSLT [7] which provides a standard means to provide transformation rules to apply to XML documents, either into another XML format or into a different format altogether. XSLT is itself an XML format.

[3] This URL should refer to a reference site on the Web which provides the definitive definition of the symbols, which this site currently does not provide.

[4] Strictly speaking, all the numeric values and all operators from MathML used vdmML should be interpreted in a standard model for mathematics. As the semantics for VDM are different, especially with respect to undefinedness and LPF, all MathML operators should have a qualifying semantics element redefining them in the semantics of VDM. However, given that they are represented here in the context of vdmML, we feel that it clear what the intended semantics is, and qualifying semantics is superfluous.

In order to demonstrate the usefulness of vdmML, three transformations of vdmML are initially proposed using XSLT scripts to provide different presentations for different circumstances.

– **vdmML to ASCII** This transformation takes a vdmML marked up text and converts to the raw ASCII input format, as used by the IFAD VDM Tools.
– **vdmML to HTML** This transformation takes a vdmML marked up document and converts it into a HTML format suitable for display on a standard Web browser, which is not necessarily an XML enabled.
– **vdmML to LaTeX** This transformation provides suitable LaTeX markup for print quality documents.

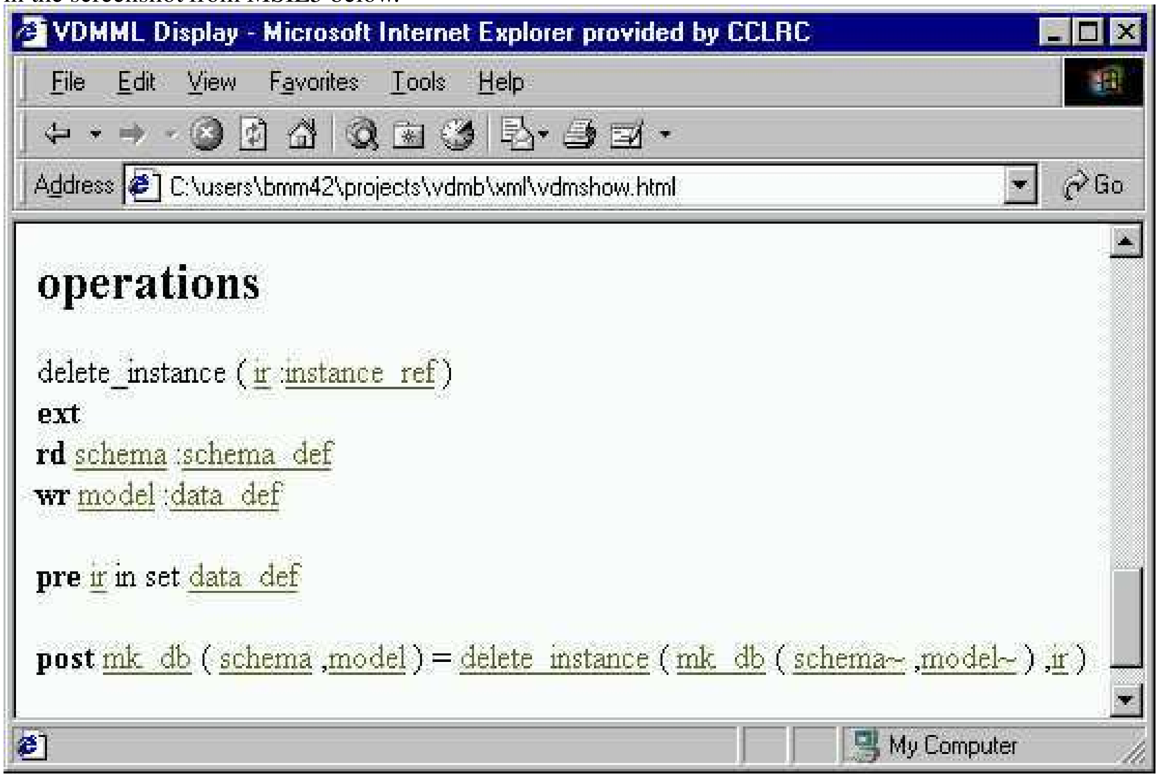
All three transformation scripts can be processed using a standard XSL processor, such as XT [8], either on the server or on the client-side. Alternatively, MSIE5 provides a built-in XSL processor; an XML document can be accessed across the web, together with a reference to the XSL stylesheet and MSIE5 will apply the stlyesheet to the XML, displaying it in using its normal HTML display rules if the output is HTML. Thus this is a suitable demonstration of the vdmML to HTML processing script.

XSLT provides a set of rules for transforming the XML document which are applied by walking over the XML document tree. A fragment of the vdmML to HTML script is given below. This gives the rule which is applied to (implicit) Operation elements. All XSL elements are given with a `xsl` namespace prefix.

```
<xsl:template match="vdm:Operation[@type='implicit']">
  <p>
    <a>
      <xsl:attribute name="src">#<xsl:value-of select="@id"/></xsl:attribute
      <xsl:value-of select="mml:ci"/>
    </a>
    <xsl:apply-templates select="vdm:Parameters"/>
    <xsl:apply-templates select="vdm:Result"/>
    <br/>
    <xsl:if test="vdm:Externals">
      <strong>ext</strong><br/> <xsl:apply-templates select="vdm:Externals"
    </xsl:if>
    <xsl:apply-templates select="vdm:PreCondition"/>
    <xsl:apply-templates select="vdm:PostCondition"/>
    <xsl:apply-templates select="vdm:Exceptions"/>
  </p>
</xsl:template>
```

An XSLT stylesheet consists of a series of templates which match the XML document at specified points and apply the rule to the content of that point. Thus here the template matches the `vdm:Operation` element, with the condition that its `type` attribute is set to `implicit`. On matching, this template will generate a HTML `p` element, an `a` element etc. The embedded `<xsl:apply-templates ... >` elements will apply the templates in the stylesheet to the sub-elements of `vdm:Operation` specified, for example, vdm:Parameters, vdm:PreCondition and vdm:PostCondition. Note that conditionals can be also added using <xsl:if ... >.

The result of applying this stylesheet to the above example operation is displayed in the screenshot from MSIE5 below.



In this transformation, internal link anchors are placed on definitions of identifiers, and internal links are placed on the usage of identifiers. Thus, this transformation automatically produces a version of the document with hyperlinked cross-references.

6 Further Proposals

Representing VDM in XML opens up further possibilities. In this section we discuss some of these as suggestions for future directions of development of vdmML, and further XML support for formal methods.

6.1 Client-side processing

Having delivered VDM to the browser, it can be manipulated using transformations on the structure, using XSLT as discussed above, and also by the using the Document Object Model [10], an API onto the XML structure provided in standard browsers.

This offers the possibility of developing lightweight client-side processing and analysis tools.

For example, XSLT could be used to generate proof obligations on the fly, and present them to the user. Further, given a vdmML enabled front-end to a theorem prover, such proof obligations could then be passed on to the theorem prover for analysis. Results from the theorem prover, in the form of proof trees, could also be passed back to the user in XML format for display. Thus vdmML could support a web-based distributed VDM support environment.

6.2 Integration with other Web formats

By using vdmML, VDM documents can be seemlessly integrated with other existing XML based formats on the web. Above we have discussed using MathML and HTML (or the XML based variant of HTML, XHTML [25]) for mathematics and documents. Other existing formats include:

– *Scaleable Vector Graphics* (SVG [21]) for representing diagrams using XML, which could provide a graphical description of VDM, via an XSLT transformation.
– *Digital Signatures* (DSig [11]) for authentication of VDM specifications, turning them into legal contracts.
– *Resource Description Format* (RDF [18]) for providing contextual information ("metadata") about VDM documents as web resources. This could be used for example for cataloguing and searching VDM repositories, for version control, and for providing supporting material.

By using these technologies, VDM specifications can become flexible and powerful web-based resources within a distributed software development support environment.

6.3 Exchange with other methods

vdmML also opens the possibility of exchange with other formats. As mentioned above, there has already been preliminary work on representing both Z and B using XML. XML provides a means for representing a mixed document with specifications in each which can be extracted simply to be processed in the support system of each tool. Further, XSLT could be used to implement transformations between VDM and other languages, such as the transformations proposed to integrate VDM and B [3, 15, 4], within the web client. Thus XML could support heterogeneous development, with each formal method being used where it is most appropriate and the consistency of the development ensured by transformation.

Further, vdmML could support the integration of formal and informal methods. For example, there are already proposed XML interchange formats for UML [19, 26]. As with formal methods above, XML could support integrated document with both VDM and UML, but distinguishable. Again, transformation could support the formal translation of UML notions into VDM and vice-versa. SVG can be used to present UML graphically in the browser.

7 Towards a Formal Methods Markup Language

A more ambitious longer term aim might be to develop a single XML based markup language for representing all formal methods data which might be called Formal Methods Markup Language (FMML). This could be seen as similar to the MathML effort which tries to provide sufficient expressive power to represent the all of mathematics, but also allows individual domains to modify or extend the notions to the specialised semantics of that domain. Thus FMML might provide generic elements for, for example, specifications, modules, refinements, proof rules and theories, and proofs, but for each particular variant formal method, (or class of formal methods such as model based methods like VDM, B and Z), specialised notions would be introduced, such as the state or operation with its post-condition.

The benefits of a single FMML, above and beyond the benefits mentioned in the introduction, might include the following.

- It enables the exchange of formal methods data between different formal methods.
- It enables the exchange of formal methods data between different formal methods tools.
- It provides support for heterogeneous specifications and developments within a single framework.
- It provides a single interchange format between different theorem proving systems.
- It provides a single generic formatting mechanism into other document formats (TeX, RTF etc) for printing.

However to use FMML to integrate languages we need to tackle larger problems. Different languages have different constructs and means of representing those constructs. Different languages have very different semantics. If FMML would seek to integrate for example, model-based languages, process-algebras and algebraic specification languages, then a wide variety of different structures and semantics would need to be accommodated. How these differences are handled would be the major challenge of this work. It is not proposed to produce a single universal specification language: such efforts have been undertaken before and are beyond the scope of FMML. What FMML would provide is a universal syntax for exchange and integration.

XML Schema with its flexible and modular structure and its inheritance mechanism might well provide a suitable vehicle for describing such a single language. The effort also might take inspiration from the Text Encoding Initiative (TEI) [22], a long running project to electronically encode literary texts using SGML. What it interesting for FMML is its ”pizza model” style of giving DTDs. TEI provides a base DTD providing basic structures common to all TEI documents. However, particular classes of texts (plays, poetry etc) provide a ”topping” DTD to provide structures particular to that class. This model could be used to provide base structures (specification, module, expression, statement, rule, proof for instance), and specific DTD extensions given for given formal methods.

8 Conclusion

The work presented in this paper is a prelimary snapshot of work in progress together with some thoughts for future work. Nevertheless, it demonstrates the utility of presenting VDM in XML, and forms the springboard for further work in this area, to bring some of these promising ideas into reality.

Acknowledgements

The author would like to thank his colleagues on the UK EPSRC funded project “VDM+B” for their support. Also, John Wordsworh and Georges Mariano for their help in preparing this paper, and John Fitzgerald for his helpful comments.

References

1. *Amaya Editor/Browser*, http://www.w3.org/Amaya, 2000.
2. J.C. Bicarregui and B.M. Matthews, *The Specification and Proof of an EXPRESS to SQL “Compiler”* in Proof in VDM: Case Studies, ed. J.C. Bicarregui, Springer, 1998.
3. J.C. Bicarregui, B.M. Matthews, B. Ritchie and S. Agerholm *Investigating the integration of two formal methods* Formal Aspects of Computing 10(6), 1998.
4. J.C.Bicarregui et.al. *Supporting co-use of VDM and B by translation* submitted to the 2nd VDM workshop, York, Sept 2000.
5. T. Bray, J Paoli, C.M. Sperberg-Mcqueen, *Extensible Markup Laguage (XML) 1.0*, W3C Recommendation, http://www.w3.org/TR/1998/REC-xml-19980210, February 1998.
6. T. Bray, D. Hollander, A. Layman *Namespaces in XML* World Wide Web Consortium Recommendation REC-xml-names-19990114, http://www.w3.org/TR/1999/REC-xml-names-19990114, January 1999
7. James Clark, *XSL Transformations (XSLT), Version 1.0* W3C Recommendation, http://www.w3.org/TR/1999/REC-xslt-19991116, 16 November 1999.
8. James Clark, *XT*, http://www.jclark.com/xml/xt.html
9. L. A. Dennis, G. Collins, M. Norrish, R. Boulton, K. Slind, G. Robinson, M. Gordon, and T. Melham,*The PROSPER Toolkit*. Proceedings of TACAS 2000 LNCS 1785, 2000.
10. *Document Object Model (DOM) Level 1 Specification Version 1.0* W3C Recommendation REC-DOM-Level-1-19981001, http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001, 1998
11. *XML-Signature Syntax and Processing* W3C Working Draft http://www.w3.org/TR/2000/WD-xmldsig-core-20000711/, 2000.
12. Georges Mariano, *Bcaml : B parser module*, http://www3.inrets.fr/Public/ESTAS/Mariano.Georges/annonce/index.html, July 2000.
13. Georges Mariano, private communication, July 2000.
14. Mathematica, http://www.wolfram.com, 2000.
15. B.M. Matthews, B. Ritchie and J.C. Bicarregui, *Synthesising structure from flat specifications*, in D. Bert (ed), *Proceedings of the 2nd International B Conference* LNCS 1393, Springer-Verlag, 1998.
16. N. Poppelier, R. Miner, P. Ion, D. Carlisle, *Mathematical Markup Language (MathML) version 2.0*, W3C Working Draft, http://www.w3.org/TR/2000/WD-MathML2-2000-328, March 2000.
17. The Prosper Project, http://www.dcs.gla.ac.uk/prosper/, 2000.
18. Resource Description Framework (RDF) homepage, http://www.w3.org/RDF/
19. J. Suzuki, Y. Yamamoto, *Making UML Models interoperable with UXF* Papers from UML’98, LNCS 1618, 1999.

20. *Information processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*. First edition, ISO (International Organization for Standardization). ISO 8879:1986(E), 1986.
21. *Scalable Vector Graphics (SVG) 1.0 Specification* W3C Candidate Recommendation http://www.w3.org/TR/2000/CR-SVG-20000802/, 2 August 2000
22. Text Encoding Initiative homepage, http://quirk.oucs.ox.ac.uk/TEI/index.html
23. *Vienna Development Method – Specification Language* International Standard ISO/IEC 13817-1, 1996.
24. John Wordsworth, private communication, July 2000.
25. *XHTML 1.0: The Extensible HyperText Markup Language A Reformulation of HTML 4 in XML 1.0* W3C Recommendation http://www.w3.org/TR/2000/REC-xhtml1-20000126 2000
26. *XML Metadata Interchange (XMI)*, the Object Management Group ftp://ftp.omg.org/pub/docs/formal/00-06-01.pdf, 2000.
27. W3C XML Schema activity page http://www.w3.org/XML/Schema.html