A Language for Platform Independent Communication and Storage in Multiobjective Optimisation

Adam Berry
School of Computing
University of Tasmania
Sandy Bay, Tasmania, Australia
Adam.Berry@utas.edu.au

Peter Vamplew
School of Computing
University of Tasmania
Sandy Bay, Tasmania, Australia
Peter.Vamplew@utas.edu.au

ABSTRACT

In real-world multiobjective optimisation, where problem complexity is typically high, systems are increasingly reliant on distributed computation to reduce processing times. While a number of successful distributed techniques have been proposed, they are dependent on high degrees of commonality between communicating processes – requiring homogeneity with respect to platform, environment or implementation language. Such constraints seriously restrict the applicability of distributed optimisers. Consequently, it is important to define a standard independent language for the transmission and storage of agent data in disparate interacting systems. Based on XML, MOO_ML represents such a language.

1. INTRODUCTION

Since real-world problems are typically composed of multiple and often conflicting objectives [1], the conceptual value of multiobjective optimisation (MOO) to decision makers is considerable. However, the practical reality of optimisation is that performance is contingent not just on quality, but also the efficiency of the process. This is a burden for general optimisation since real-world problems are inherently complex, particularly with respect to problem evaluation [2]. Such issues are intensified in multiobjective optimisation, where the number of objectives affects both the processing time required for evaluation and the size and dimensionality of the Pareto front [2]. The net effect is that multiobjective optimisers can slow to such an extent that the utility of the approach is drawn into question.

The pressure to improve time performance has led to an increased focus on distributed computing in the field (see Section 3). Such burgeoning attention is merited - by sharing calculations across multiple processors, the overall execution time must fall (given low intra-processor interaction).

While current implementations of the distributed paradigm do offer significant performance gains, they impose unreasonable commonality requirements: where systems must be homogeneous in platform, environment, algorithm and/or implementation language. Since real-world networks are often ad-hoc and contain disparate components, such a requirement is excessively restrictive – particularly given the costs associated with aligning system elements. Moreover, it is inflexible: a Linux C++ implementation of one optimiser cannot communicate with a Windows C# implementation of another algorithm without significant recoding.

An alternative to requiring commonality is to define a standard, platform-independent format for the storage and transmission of agents. This will accord increased flexibility to distributed systems while reducing realignment costs. Such a format would allow contrasting optimisers, using different algorithms, written in disparate languages, and running on varied hardware and environments to communicate and interact.

The Multi-Objective Optimisation Markup Language (MOO_ML) represents such a format. It is based on the widely adopted XML standard, so creating and parsing MOO_ML documents should be trivial given the plethora of available XML tools, while the text-based XML format also grants the platform independence required. Moreover, MOO_ML itself is generic with respect to all multiobjective optimisation algorithms.

2. MULTIOBJECTIVE OPTIMISATION

The goal of all multiobjective optimisers is to develop an approximation of the Pareto optimal front for a problem that has multiple, potentially conflicting, objectives. In a minimisation problem with n objectives, a is Pareto optimal if, for all possible population members P:

$$\forall p \in P, \forall i \in \{1, 2, .., n\} : f_i(a) \leq f_i(p) \cup (a \text{ incomparable } p)$$

Equation 1 – Pareto Optimality

where f is an objective function that maps a multi-variate solution to a result; a is incomparable with p if it is better than a in one objective but worse in another; and the Pareto front is formed by all Pareto optimal members.

The specifics of generating the approximation front vary between algorithms, but all systems invariably include a representation of a multi-variate solution that is evaluated against the objectives. Therefore the core knowledge of a generic multiobjective agent consists of a set of variable solutions. Given that the majority of multiobjective fronts will always contain more than one solution, it is necessary for any standard, broad-use, storage format to store a set of generic agents.

3. DISTRIBUTED MOO SYSTEMS

The distributed paradigm has only recently drawn attention from the multiobjective research community, despite "extensive research" [3] in single objective evolutionary algorithms (EAs). The early work in the area has largely been focussed around extending the concepts found in the more traditional EAs – namely, the master-slave, island and diffusion (cellular) models [4].

3.1. THE MASTER-SLAVE MODEL

In the master-slave model [5], a host (master) divides the population into smaller clusters that are evaluated on secondary processors (slaves) and then returned to the master for optimisation (although some master-slave models also allow slaves to perform some evolutionary computation prior to coalescence [6, 7]). This approach aims to reduce time requirements while maintaining the same results as in a single-processor environment (since the optimisation algorithm operates on the whole population). The primary drawback is that the entire population must be repeatedly transported between the master and slaves, thus limiting the practical population size and reducing the speed advantages gained by using such an approach. Moreover, such systems are only as fast as their least efficient slave, since the master must typically wait for all slaves before commencing execution.

3.2. THE ISLAND AND DIFFUSION MODELS

To avoid the communication bottleneck of the master-slave model, the island model [2, 8] assigns an independent self-evaluating population to each processor (island) with genetic diversity maintained via small island migrations. Since this model does not operate on the entire population, it cannot guarantee the performance quality seen in the master-slave algorithm. However, recent preliminary results [2] have indicated that by focussing each sub-population on particular areas of the objective-space, the quality of solutions can be substantially improved.

Diffusion [9] is a more finely grained version of the island model, achieved by increasing the number and connectedness of the islands. It offers a middle ground between the island and master-slave approaches, requiring more communication than the island approach, but better approximating a single cohesive population due to its high connectivity. Thus far research into the application of diffusion to distributed multiobjective optimisers has been limited due to its concordance with parallel, shared-memory machines where the significant communication overhead is of a lower concern [4].

3.3. HOMOGENEITY LIMITATIONS

Irrespective of approach, one key issue has been omitted from discussion in most distributed multiobjective processing literature: homogeneity requirements. This is surprising given the heterogeneous nature of most larger real-world distributed systems [10] and ad-hoc workstation environments [11], where demands for homogeneity impose severe restrictions. It is unreasonable to expect that organisations should be required to make significant software and hardware changes simply to allow system-wide commonality. Moreover, enabling distributed multiobjective systems in heterogeneous environments grants powerful flexibility: each island can operate a unique algorithm, written in the most appropriate language; very-large scale problems, requiring significant computing resources, can operate across the Internet; and small businesses that do not have the funding to purchase hardware or convert software can capitalise on the powers of distributed computing.

4. XML

The virtues of XML have been extolled throughout recent literature and do not require thorough analysis here. However, a brief discussion of the language is necessary, given that it forms the basis of MOO_ML.

The strength of XML lies in its inherent simplicity, flexibility and independence. For all the hyperbole surrounding it, XML does little more than provide a standard format for storing and describing data through the application of user-defined tags. This simplicity, however, means that XML files can be stored in a platform independent manner within a simple text file. Such independence makes XML suitable for distributed multiobjective communication, as systems do not require homogeneity to share information. Furthermore, given the significant support afforded XML, parsing and generating XML documents has become common practice in most major programming languages. XML processing libraries are included as standard within Java 1.4, and a host of freely available, well recognised, parsers exist for other languages (Xerces2 for C++, Perl, COM and Java; Expat for C; and Chilkat XML for C#). Additionally, numerous stand-alone XML applications offer authoring utilities and facilitate the generation of reliable and instantly useable code from standard W3C schemas. As a consequence, any implementation realignment required for the use of MOO_ML should be straightforward and both cost and time effective.

5. INTRODUCING MOO_ML

MOO_ML is a proposed standard language for the storage and communication of data between distributed multiobjective systems. Derived from XML, MOO_ML is a simple, flexible and platform independent language that is designed to store the core content of any multiobjective optimiser. While the focus of MOO_ML is to provide a common medium for communication, it can also be used for the storage of information produced by non-distributed multiobjective systems for use in archiving, population imaging or post processing.

5.1. BREVITY

A danger of basing a language in XML is the potential for document bloat due to the nested nature of tags. This can inhibit the readability of the document, and also slow the writing and parsing of files. Thus, a central focus of MOO_ML is maintaining brevity in content description. Only the core components of a multiobjective system must be stored in a MOO_ML document – consisting of a collection of generic agents, which contain variable solutions to the proposed multiobjective problem. Consequently, the minimum number of tag pairs (base-tags) required for a valid MOO_ML document is defined in Equation 2:

$$\min(tagPairs) = 1 + |P| * (|S| + 1)$$

Equation 2 – Minimum Base-Tags in MOO_ML

where $|P|$ is the population size; and $|S|$ is the total number of unique variable solutions for all objectives.

This represents the smallest possible well-structured combination of tag pairs for an XML representation of a multiobjective population.

5.2. VALIDATION

The MOO_ML schema document specifies the order, structure and type of components used in the language, thereby simplifying the validation of MOO_ML files. According to the W3C specification of XML [12], a validation-parser will report any malformed syntax or invalid content. Thus MOO_ML documents with incorrect solution typing, empty agents or poorly defined problem statements can instantly be rejected based simply on schema document validation.

5.3. SELF-DESCRIPTIVENESS

All tags within the proposed MOO_ML standard require, at most, limited documentation to describe their meaning. Tags and attributes have names consistent with the multiobjective literature and their purpose should be apparent to those familiar with the field. This self-descriptiveness increases simplicity and makes the movement from pre-existing, typically ad-hoc, formats less daunting.

5.4. FLEXIBILITY

Though MOO_ML strives for brevity, it is important that flexibility and power is not disregarded. MOO_ML therefore provides a number of optional attributes and tags that are designed to provide further content description when it becomes necessary.

The succinct base-tags are appropriate when certain assumptions can be made about the operation of the distributed system, such as a shared definition of both problem and agent construction. Although these assumptions are largely true for small-scale implementations, they are likely to be invalid when systems are of a larger scale and components are not developed in-house. Thus, MOO_ML facilitates syntactically correct mathematical objective definitions via the application of the Mathematical Markup Language (MathML [13]) and allows the labelling of solutions with variable names.

Beyond their immediate value, objective definitions provide significant scope for future work. By including well-structured, XML-based objective definitions in transmitted documents, implemented algorithms can become generic – parsing the definition into evaluating code, with no need for manual encoding of the problem. The power of such malleability is significant: subtle variations of definitions can be examined with no need to manipulate algorithm code; consistency of definitions can be guaranteed between disparate systems; and generic distributed optimisation services can be made available via the Internet to allow anyone, anywhere, to perform multiobjective optimisation on any mathematically definable problem. Additionally, MathML provides presentation markup that facilitates high quality documentation of problem definitions in TeX and Internet browsers. Thus, the potential application of incorporating both problem and solution into a communicable standard language is large and the development of MathML tools, parsers and evaluators (see [14]) can only build upon such potential.

It is important to note that while the base-tags are well suited to describing components relevant to the generic communication of agents between systems, they are of little practical value to the master-slave model. In this case, it is not the solutions stored by agents that are of primary interest in the communication process, but rather the results that applying those solutions will bring. Consequently, MOO_ML includes the facility for storing objective results in each agent. In addition to the benefits seen in the master-slave model, this feature can reduce potential computation in general by using the MOO_ML document as a result cache.

The final selection of optional MOO_ML components are concerned with documentation and identification. Given that in practice it is likely that many MOO_ML documents will be produced, it is important that outputs can be uniquely identified. As such, the language includes ID and creation-date attributes, which can be used to both track and identify documents through time. Whilst these features will primarily see benefits in post-processing or result-retrieval, they may also find utility in distributed optimisation where receipt of duplicate documents can indicate system errors (such as unnecessary retransmission).

6. FORMALLY DEFINING MOO_ML

This section defines MOO_ML through the use of the W3C XML Schema Document language (XSD) [15]. It is assumed that the reader is familiar with XSD, though accompanying figures and descriptions should lessen the burden for those less accustomed to the technique. Also note that extensive examples and guides for MOO_ML are available online [16].

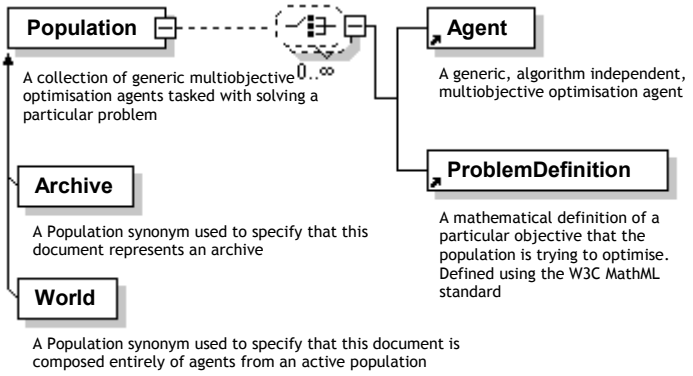
6.1. THE POPULATION ELEMENT.

The *Population* element (Figure 1) represents the root node of any MOO_ML document. Note that the *Population*, *Archive* and *World* tags are synonymous and can be used interchangeably in the root node to further clarify the purpose of the document. Thus, the content of any MOO_ML document can be broadly classified without needing to resort to external documentation. The inclusion of *Variables* and *Objectives* attributes facilitate further in-parser agent and objective validation.

6.2. THE AGENT ELEMENT.

The *Agent* element (Figure 2) defines a generic multiobjective agent that can contain solutions and results (see Section 6.3). If the document need only contain a single agent then this element can theoretically act as the root, though it is preferable to use a single member population, as the *Agent* element has no facility for additional documentation or objective definitions.

Diagram



Children: Agent*; ObjectiveDefinition*

Attributes

Name	Type	Description
ID	xs:string	An identifier for this population
Variables	xs:positiveInteger	The number of unique variables to be solved
Objectives	xs:positiveInteger	The number of objectives to be optimised
Creation	xs:dateTime	The creation date of this particular document

Source

```
<xs:element name="Population">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="Agent"/>
      <xs:element ref="ObjectiveDefinition"/>
    </xs:choice>
    <xs:attribute name="ID" type="xs:string"/>
    <xs:attribute name="Variables" type="xs:positiveInteger"/>
    <xs:attribute name="Objectives" type="xs:positiveInteger"/>
    <xs:attribute name="Creation" type="xs:dateTime"/>
  </xs:complexType>
</xs:element>
<xs:element name="Archive" substitutionGroup="Population"/>
<xs:element name="World" substitutionGroup="Population"/>
```

Figure 1 – Defining the Population Element and Population Synonyms

Diagram

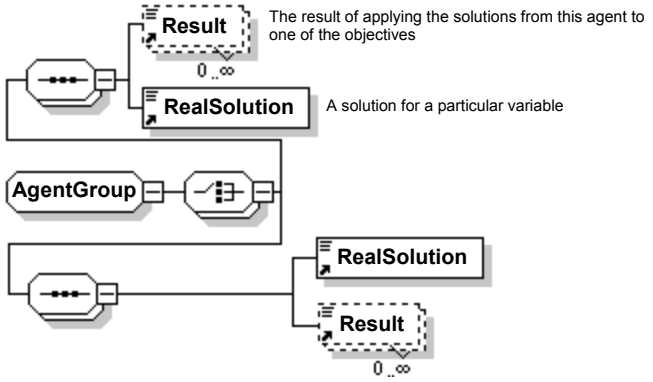
Used By: **Element**: Population

Source

```
<xs:element name="Agent">
  <xs:complexType>  <xs:group ref="AgentGroup"/>  </xs:complexType>
</xs:element>
```

Figure 2 – Defining the Agent Element

Diagram



Note: Though complex in appearance, the structure of the *AgentGroup* is simply an unordered collection of *Results* and *Solutions*, requiring at least one *Solution* element.

Children: Result*; RealSolution+ | **Used By**: **Element**: Agent

Source

```
<xs:group name="AgentGroup">
  <xs:choice>
    <xs:sequence>
      <xs:element ref="RealSolution"/>
      <xs:element ref="Result" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:sequence>
      <xs:element ref="Result" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="RealSolution"/>
    </xs:sequence>
  </xs:choice>
</xs:group>
```

Figure 3 – Defining the AgentGroup

Type	xs:decimal	Used By	**Group**: AgentGroup
Attributes	**Name** ObjName	**Type** xs:string	**Description** The name of the objective for the corresponding result
Source	<xs:element name="**Result**"> <xs:complexType> <xs:simpleContent> <xs:extension base="xs:decimal"> <xs:attribute name="**ObjName**" type="xs:string"/> </xs:extension> </xs:simpleContent> </xs:complexType> </xs:element>		

Figure 4 – Defining the Result Element

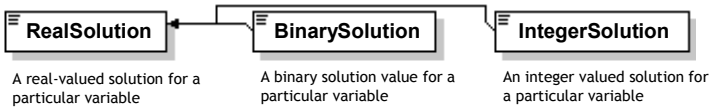
Diagram			
Type	RealSolutionType; BinarySolutionType; IntegerSolutionType	Used By	**Group**: AgentGroup
Attributes	**Name** VarName	**Type** xs:string	**Description** The name of the variable being solved
Source	<xs:element name="**RealSolution**" type="**RealSolutionType**"/> <xs:element name="**BinarySolution**" type="**BinarySolutionType**" substitutionGroup="**RealSolution**"/> <xs:element name="**IntegerSolution**" type="**IntegerSolutionType**" substitutionGroup="**RealSolution**"/> <xs:complexType name="**RealSolutionType**"> <xs:simpleContent> <xs:extension base="xs:decimal"> <xs:attribute name="**VarName**" type="xs:string"/> </xs:extension> </xs:simpleContent> </xs:complexType> <xs:complexType name="**BinarySolutionType**"> <xs:simpleContent> <xs:restriction base=" **RealSolutionType** "> <xs:pattern value="[01]+"/> </xs:restriction> </xs:simpleContent> </xs:complexType> <xs:complexType name="**IntegerSolutionType**"> <xs:simpleContent> <xs:restriction base=" **RealSolutionType** "> <xs:pattern value="[+-]?[0-9]+"/> </xs:restriction> </xs:simpleContent> </xs:complexType>		

Figure 5 – Defining Solutions

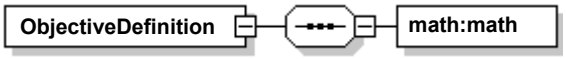
Diagram			
Children	**Element:** math (refer to MathML XSD)	Used By	**Element**: Population
Attributes	**Name** ObjName ObjType	**Type** xs:string xs:enumeration	**Description** The name of the objective that the definition describes Specifies if the objective is a "Constraint", "Min" or "Max" problem
Source	<xs:element name="ObjectiveDefinition"> <xs:complexType> <xs:sequence> <xs:element ref="math:math"/> </xs:sequence> <xs:attribute name="ObjName" type="xs:string"/> <xs:attribute name="ObjType" type="ObjectiveType" use="required"/> </xs:complexType> </xs:element> <xs:simpleType name="ObjectiveType"> <xs:restriction base="xs:string"> <xs:enumeration value="Min"/> <xs:enumeration value="Max"/> <xs:enumeration value="Constraint"/> </xs:restriction> </xs:simpleType>		

Figure 6 – Defining the ObjectiveDefinition Element

Source	<xs:schema **targetNamespace**="http://www.comp.utas.edu.au/moo" xmlns:**xs**="http://www.w3.org/2001/XMLSchema" xmlns:**math**=http://www.w3.org/1998/Math/MathML xmlns="http://www.comp.utas.edu.au/moo" elementFormDefault="qualified" attributeFormDefault="unqualified"> <xs:import namespace=http://www.w3.org/1998/Math/MathML schemaLocation="http://www.w3.org/Math/XMLSchema/mathml2/mathml2.xsd"/>

Figure 7 – The Namespace Declaration

6.3. THE AGENTGROUP.

The *AgentGroup* (Figure 3) represents the set of results and solutions that define an *Agent*. Note that an *Agent* is only well formed if it contains at least one solution and is invalid if the number and type of solutions and results do not correspond to the *Variables* and *Objectives* root attributes or the *ObjectiveDefinition* (should they exist). If all results or solutions lack labelling attributes (*ObjName* or *VarName*) then it is assumed that the ordering of the elements correspond to the ordering of the objectives or variables in the *ObjectiveDefinition* or, if no *ObjectiveDefinition* is provided, the optimiser. Duplicate attribute values or partial inclusion of naming labels within a single *Agent* are invalid and will cause the document to fail during parsing.

6.4. THE RESULT ELEMENT

The optional *Result* element (Figure 4) contains the numeric result of applying the solutions stored within the agent to a single objective.

6.5. SOLUTION ELEMENTS

In MOO_ML, a solution to a problem variable can be a *RealSolution*, *IntegerSolution*, or *BinarySolution* (Figure 5). A *RealSolution* can contain any real valued number; an *IntegerSolution* is restricted to integers; and the *BinarySolution* is capable only of housing binary digits.

6.6. THE OBJECTIVE DEFINITION ELEMENT

An *ObjectiveDefinition* (Figure 6) represents a unique objective in the optimisation problem, with syntax derived from MathML (inclusive of both content and presentation layers). Thus MOO_ML can represent almost any mathematically definable problem.

6.7. DEFINING NAMESPACES

Figure 7 illustrates the namespace declarations for MOO_ML. While most are standard, the *targetNamespace* is significant as it specifies the location of the MOO_ML schema for third-party use.

7. FUTURE WORK

The most pressing future work lies in applying MOO_ML to pre-existing distributed MOO systems. It is only once such integration is made that the utility of the language can be gauged, while it presents the best avenue for highlighting possible areas of extension and refinement. Upcoming papers will examine the performance of MOO_ML-based highly heterogeneous distributed systems – where there is divergence in platform, environment and optimisation algorithm. Later work will be focussed on the use of MOO_ML to enable shared archiving as an alternative to pre-existing distributed models and the development of MOO_ML parser libraries for Java, C++ and C# [16].

8. CONCLUSION

MOO_ML represents a movement from ad-hoc storage methods to a platform-independent standard for generic multiobjective optimisation. The importance of this approach is most evident in distributed systems, where it will grant considerable flexibility by removing the need for restrictive system homogeneity. Furthermore, by grounding MOO_ML in XML, the complexities of document parsing and validation are reduced, while lessening language migration burdens. To validate such claims, upcoming work should examine the use of MOO_ML in areas where distributed computing is of advantage.

9. ACKNOWLEDGEMENTS

The authors would like to acknowledge: Trixie Berry, Michael Berry, Pauline Mak, Ian Lewis, Chloe Skilbeck, the School of Computing and the makers of XMLSpy.

REFERENCES

[1] Coello Coello, C.A., "A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques". Knowledge and Information Systems, 1999. 1(3): p. 129-156.

[2] Deb, K., Zope, P., and Jain, A. "Distributed Computing of Pareto-Optimal Solutions with Evolutionary Algorithms". In Evolutionary Multi-Criterion Optimisation, Second International Conference. 2003.

[3] Cantu-Paz, E., "A Survey of Parallel Genetic Algorithms" (Tech Report). 1997, Illinois GA Lab.

[4] Zydallis, J., Van Veldhuizen, D., and Lamont, G. "Using Parallel Concepts in Multiobjective Evolutionary Algorithms". In The Second Workshop on Multiobjective Problem Solving from Nature. 2002. Granada, Spain.

[5] Makinen, R., et al. "Parallel Genetic Solution for Multiobjective MDO". In Parallel CFD. 1996.

[6] Toscano Pulido, G. and Coello Coello, C. "The Micro Genetic Algorithm 2: Towards On-Line Adaptation in Evolutionary Multiobjective Optimization". In Evolutionary Multiobjective Optimisation. 2003. Faro, Portugal.

[7] Toro, D., Ortega, J., and Diaz, A., "PSFGA: A Parallel Genetic Algorithm for Multiobjective Optimization". In Euromicro Workshop on Parallel, Distributed and Network-based Processing, 2002.

[8] Horii, H., Miki, M., Koizumi, T. and Tsujiuchi, N., "Asynchronous Migration of Island Parallel GA for Multi-Objective Optimization Problem". In 4th Asia-Pacific Conference on Simulated Evolution and Learning. 2002.

[9] Rowe, J., Vinsen, K., and Marvin, N. "Parallel GAs for Multiobjective Functions". In Second Nordic Workshop on Genetic Algorithms and their Applications. 1996. Vassa, Finland.

[10] Vorapanya, A., "Large-Scale Distributed Services" (PhD). University of Florida. 2000.

[11] Al-Yamani, A., Sait, S., and Youssef, H., "Parallelizing Tabu Search on a Cluster of Heterogeneous Workstations". Journal of Heuristics, 2002. 8(3): p. 277-304.

[12] Bray, T., et al. (Eds), "Extensible Markup Language (XML) 1.0" (Third Edition), http://www.w3.org/TR/REC-xml, 2001

[13] Carlisle, D., et al., "Mathematical Markup Language (MathML) Version 2.0", http://www.w3.org/TR/MathML2, 2001

[14] Ion, P.E., "MathML 2.0 Implementation and Interoperability Report (Draft)", http://www.w3.org/Math/iandi, 2001

[15] Thompson, H., et al. (Eds), "XML Schema Recommendation (Parts 1 and 2)", http://www.w3.org/TR/xmlschema-1 and http://www.w3.org/TR/xmlschema-2, 2001

[16] Berry, A. and Vamplew, P., "MOOnline", http://www.comp.utas.edu.au/moo, 2004