Semantic Extensions to Domain-Specific Markup Languages

Aparna Varde, Elke Rundensteiner, Murali Mani, Mohammed Maniruzzaman and Richard D. Sisson Jr.

Worcester Polytechnic Institute, Worcester, MA, USA.
(aparna | rundenst | mmani | maniruzz | sisson)@wpi.edu

ABSTRACT

A markup language serves as a medium of communication for storing and publishing textual, numeric and other forms of data seamlessly. XML, the eXtensible Markup Language has become the lingua franca in web publishing. It is also a widely accepted standard for databases and document storage. There are many domain-specific markup languages designed using special XML tag sets. Standardization bodies and research communities may extend these to include additional semantics of areas within and related to the domain. This paper outlines the issues to be considered when extending domain-specific markup languages, namely, the motivation for extension, the semantic considerations, the syntactic constraints and other relevant aspects. Illustrating examples are given from the domains of Medicine, Finance and Materials Science. Particular emphasis in these examples is on the extension of the Materials Science Markup Language MatML to include the semantics of one sub-area, namely, the Heat Treating of Materials. The focus of this paper however is not the design of one particular language but rather the generic issues involved in extending domain-specific markup languages.

Keywords: XML, Web Databases, Semantics, Communication Standard, Ontology.

1. INTRODUCTION

XML and Domain-Specific Markup Languages

XML, the eXtensible Markup Language [1] is becoming a widespread standard in web publishing. Developed by the World Wide Web Consortium (W3C), XML is designed to improve the functionality of the web by providing more flexible and adaptable information interpretation. XML is extensible in the sense that it is not a fixed format like HTML [2] (a single predefined markup language). Rather, XML is a meta-language that can be used for the design of customized markup languages. There are many domain-specific markup languages defined that follow the XML syntax and encompass the semantics of the domain. The inclusion of semantic tags in a document makes a document self-describing. Thus this helps in document storage and exchange. A domain-specific markup language becomes a medium of communication for the potential users of the given domain. Potential users include industries, universities, standards bodies, publishers, research groups, domain experts and others. It is important to facilitate storage, retrieval and exchange of information among all these users.

MML: Medical Markup Language

A domain-specific markup language has a certain set of tags that capture the semantics of the domain. An example is the Medical Markup Language (MML) [3] that has been developed in Japan in order to create a set of standards by which medical data, within Japan and hopefully worldwide, can be stored, accessed and exchanged. The following MML module contents are defined at the present time: patient information, health insurance information, diagnosis information, lifestyle information, basic clinic information, particular information at the time of first visit, progress course information, surgery record information and clinical summary information [3]. They are of use to primary care physicians, general surgeons, their patients and related entities. However, specific information, for example opthalmological details such as eye-diseases, spectacle prescriptions and blindness, cannot be stored using these tags. Thus there is need for extension to include the semantics of opthalmology within medicine.

Motivation for Extensions to Markup Languages

Analogous to the medical domain and opthalmology, there are specializations in other domains. Hence, there is often the need to provide a semantic extension to a domain-specific markup language to allow the representation of additional specialized information. An alternative approach for capturing this additional semantics is to define a new markup language for each aspect. For example, rather than extending the general medical tag set with opthalmological details, a new markup language could be defined for opthalmology. However, there is typically basic information about the patient in general medicine [3] that is cross-referenced in opthalmology and vice versa. Also, some data needs to be exchanged among the other specialized fields of that domain. For instance, some opthalmological information is of use to other medical areas such as anesthesia and radiology. If the experts in each area of specialization define their own independent markup language then the cross-referencing of information common to the areas is not facilitated. In order to solve this problem, if some common

tags are re-defined in the extension, then this is inefficient. The common information is stored twice leading to redundancy. It is thus more advisable to extend the existing markup language to include additional semantics.

Extending the Materials Science Markup Language

At the Center for Heat Treating Excellence (CHTE) at Worcester Polytechnic Institute (WPI), an extension is being proposed to a domain-specific markup language MatML. MatML is the Materials Science Markup Language developed by NIST (National Institute of Standards and Technology). It serves as the XML for materials property data [4]. The original MatML elements are bulk details, component details, metadata, graphs and glossary of terms. They have their own sub-elements and attributes. These provide for the storage of information related to the properties of materials such as metals, ceramics and plastics. For example, the chemical composition of a particular alloy would be stored under component details. However the MatML tags are insufficient to capture the semantics of specific sub-areas in Materials Science. The proposed extension [5] introduces into MatML the semantics of one sub-area, namely, the Heat Treating of Materials. Heat Treating [6] involves the controlled heating and cooling of materials to achieve desired mechanical and thermal properties. Quenching [6] is the rapid cooling of the material in a liquid or gas medium. It forms an important step of the Heat Treating operations and is the focus of the proposed extension. There are entities in the Quenching process such as the quenchant or cooling medium. These have properties, e.g., viscosity (the capacity to flow) is a property of the quenchant. The proposed extension to MatML is a "Quenching Element" that provides the XML tags to store these details. The schema and ontology of the actual MatML extension are beyond the scope of this paper. These are explained in [5].

The general issues in extending domain-specific markup languages are discussed in this paper. Several considerations have to be taken into account when extending markup languages. These pertain to the steps in developing the language, the features of the language and the use of XQuery [7] to retrieve information stored using the markup language. These are summarized in the following sections.

Section 2 of this paper includes the outline of the steps in extending a markup language. The languages features are described in Section 3. The use of XML schema constraints in defining a markup language is discussed in Section 4. The considerations involved in the retrieval of information using XQuery are explained in Section 5. The conclusions are stated in Section 6. The acknowledgments are given in Section 7. The references are listed in Section 8.

2. STEPS IN EXTENDING A MARKUP LANGUAGE

Markup language design has several steps analogous to the design of software systems as listed below.

1. **Understanding domain semantics:** It is important to study the domain thoroughly and know the terminology. This helps to determine the tags that are essential to store the data in the domain. Also, it is necessary to be well-acquainted with the existing markup language in the domain [3, 4] to find out where it needs extension.

2. **Modeling the data:** A data model is a generalized, user-defined view of the data related to applications that describes how the data is to be represented for manipulation by humans or computer programs [8]. Techniques such as Entity Relationship (E-R) diagrams [8] are useful in building data models. An E-R diagram is a formal method for arranging data to mimic the behavior of the real world entities represented [8]. This helps to create a picture of the entities in the domain, view their attributes and understand their relationships with each other. Data models set the stage for providing the basis for the markup language extension. Figure 1 shows a subset of an E-R diagram for Heat Treating [5, 6] with reference to terms described in Section 2.

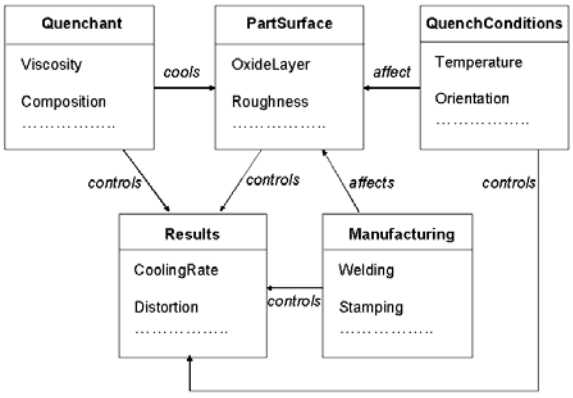

Figure 1: Subset of E-R Diagram for Heat Treating

3. **Conducting interviews:** The needs of the potential users of the markup language must be identified. Hence it is necessary to conduct detailed interviews with them. This helps to identify what entities and attributes need to be included in the extension. Potential users as stated earlier include industries, universities, standards bodies, publishers, research groups, domain experts and others. Often the needs of the potential users are adequately identified by the domain experts. Hence it is generally considered sufficient to interview domain experts.

4. **Defining the ontology:** Ontology is the study of what there is, i.e., an inventory of what exists [1, 8].

An ontological commitment is a commitment to an existence claim [1, 8]. Ontology thus serves as the established lingo for the members of the domain. Hence, after understanding the domain and conducting interviews with experts, defining the ontology is imperative in order to proceed with the design. Issues such as synonyms (two or more words having the same meaning) and homographs (one word having multiple meanings) with respect to the domain are crucial here. For example in the financial domain [9], the terms "salary" and "income" mean the same and are synonyms. However, the term "share" can have two connotations in this domain. It could mean "assets belonging to or due to or contributed by an individual person or group", or "any of the equal portions into which the capital stock of a corporation is divided and ownership of which is evidenced by a stock certificate" [9]. Thus "share" is a homograph in the financial domain. In Heat Treating, the terms "part", "probe" and "work-piece" are synonyms, i.e., they refer to the same entity [6]. Terms such as this need to be clarified with reference to the context. This is done through the ontology. Figure 2 is an example of the ontology for our proposed "Quenching Element" extension to MatML [4, 5]. It is a high-level ontology describing the Quenching entities. This is the outcome of discussions with domain experts.

- <**Quenchant**> refers to the cooling medium used in quenching.
- <**PartSurface**> describes the characteristics of the surface of the part material. The features of the material itself are described within the original MatML_doc.
- <**Manufacturing**> specifies the processes used in manufacturing the part, prior to quenching. These could be welding, stamping etc.
- <**QuenchConditions**> stores the details of the quenching process, i.e., the conditions under which the quenching is carried out, e.g. temperature.
- <**Results**> records the outcome of the quenching process in terms of parameters such as cooling rates and heat transfer coefficients at different points.

Figure 2: High-Level Ontology for "Quenching"

5. **Reiterating the ontology:** Once the ontology has been established, it is useful to have additional discussions with domain experts to make the required changes if any. For example, it may seem necessary to create new entities for clarification or remove existing entities to avoid ambiguity. The design of the ontology is reiterated accordingly.

6. **Outlining the initial schema:** The schema provides the structure, i.e., defines the grammar for the language. Once the data model and ontology are formally approved by a team of experts, the first draft of the schema is outlined. This should adhere to the syntax of the original markup language in order to be accommodated as an extension. Features provided by XML, such as constraints should be exploited for a good schema design [1]. Figure 3 shows an example of an initial schema [5]. This is a partial snapshot of the proposed XML schema for the "Quenching Element" [5] as an extension to MatML [4]. The arrow in this figure points to the tag set for the "Results Sub-element" of the "Quenching Element".

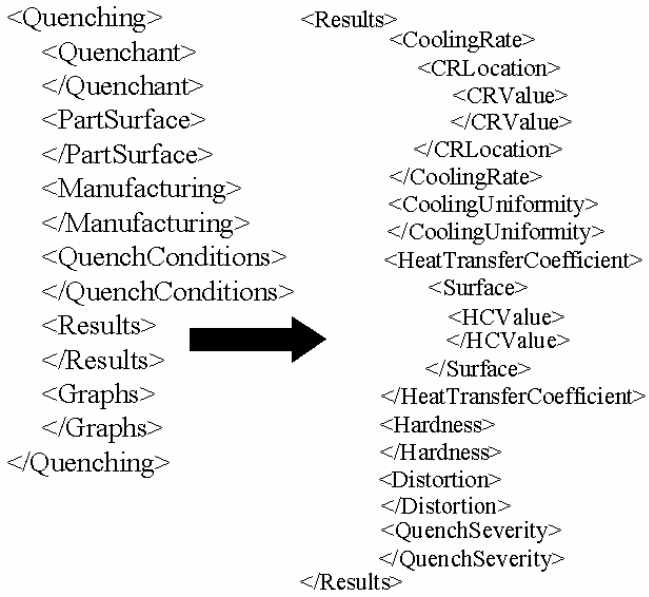

Figure 3: Partial Snapshot of MatML Extension

7. **Revising the schema based on critical reviews:** The initial schema serves as the medium of communication between the designers and the potential users of the markup language. This is subject to further changes until the domain experts are satisfied that this adequately represents their needs. Schema revision may involve several iterations, some of which are the outcome of discussions with standards bodies such as NIST for MatML [4]. Thus this stage goes beyond communication with domain experts. In order for the proposed extension to be accepted as a standard for communication worldwide and be incorporated into the existing markup language, it is important to have it thoroughly reviewed by standards bodies.

3. LANGUAGE FEATURES

The markup language extension needs to be powerful enough to incorporate the following features.

1. **Avoid redundancy:** Data stored using the original markup language should not be duplicated in the

extension. For example, material properties such as thermal conductivity [6] are already stored using the original MatML [4] and should not be stored again using the Quenching extension [5].

2. **Make information non-ambiguous:** This refers, for example, to the concepts of synonyms and homographs described earlier. The markup language needs to be clear and precise to avoid confusion while reading the stored information.

3. **Provide easy interpretability of data:** The markup language should be such that readers are able to understand and interpret stored information without much reference to related documentation. For example, in science and engineering domains [3, 6], the details of the input conditions of a performed experiment should be stored close to its results in order to enhance readability.

4. **Capture domain constraints in the schema:** There may be certain requirements imposed by the domain that need to be captured in the schema using XML constraints [1, 10]. A simple example is the primary key constraint. A primary key serves to uniquely identify an entity [1, 7]. In addition, XML provides a choice constraint [10] that allows the declaration of mutually exclusive elements. For example, in the financial domain [9] a person could be either an "insolvent" (bankrupt) or an "asset-holder", but not both. Thus these two terms are mutually exclusive. Other XML constraints are sequence constraints to declare a list of elements in order, and occurrence constraints that define the minimum and maximum occurrences of an element [10]. The markup language extension needs to make use of these as needed in order to adequately represent the domain semantics. Constraints are discussed in detail in the next section.

4. XML SCHEMA CONSTRAINTS

Constraints are mechanisms in XML that enable the storage of information adhering to specific rules such as enforcing order and declaring mutually exclusive elements [10]. Some of these constraints relevant to extending domain-specific markup languages are described below with examples from the extension of the Materials Science Markup Language, MatML [4] to include Heat Treating semantics [5].

1. **Sequence Constraint:** This constraint is used to declare a list of elements (or sub-elements) to occur in a particular order. Enclosing the concerned elements within <xsd: sequence> tags provides this constraint [10]. Figure 4 shows an example of a sequence constraint as applicable to the MatML schema extension [5]. This indicates that the sub-element "QuenchConditions" must appear before the sub-element "Results". This is required by the domain to enhance readability. The whole extension captures one instance of a Quenching process. The sub-element "QuenchConditions" denotes the input conditions of the process, while "Results" denotes the observations. The input conditions of the Quenching process affect the observations. It is thus necessary for a user to read the input conditions first in order to understand how they have an impact on the corresponding observations. Thus the sub-element "QuenchConditions" is stored before "Results".

```
<xsd:element name="Quenching">
  <xsd:complexType>
    <xsd:sequence>
      ……..
      <xsd:element name="QuenchConditions">
       …..
      </xsd:element>
      <xsd:element name="Results"/>
       …..
      </xsd:element>
       ……
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 4: Sequence Constraint Example

2. **Disjunction Constraint:** This is used to declare mutually exclusive elements, i.e., the elements that are such that only one of them can exist. It is declared using <xsd:choice> in the schema [5, 10]. For example, in Materials Science, a part can be manufactured by either Casting or Powder Metallurgy but not both [6]. Thus the corresponding sub-elements "Casting" and PowderMetallurgy" are mutually exclusive and are enclosed within <xsd:choice> tags as shown. This is illustrated in Figure 5.

```
<xsd:element name="Manufacturing">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element ref="Casting"/>
      <xsd:element ref="PowderMetallurgy"/>
    </xsd:choice>
    ……
  </xsd:complexType>
</xsd:element>
```

Figure 5: Disjunction Constraint Example

3. **Key Constraint:** A key constraint is analogous to a primary key in relational databases [8]. It is used to declare an attribute to be a primary

key. This implies that the attribute must have unique values and cannot have empty or null values. This is indicated in the schema by declaring the corresponding attribute as type "xsd:ID" and declaring its use as "required" [10]. An example of this is shown in Figure 6. This denotes that for the element "Quenchant" which refers to the cooling medium used in a Quenching process, its "id" is crucial since it serves to uniquely identify the medium [5]. In other words, in storing the details of the Quenching process, it is required that the id or name of the cooling medium be stored [6]. This is because the purpose of conducting these experiments is to characterize the quenchants. Thus this is enforced as a constraint.

```
<xsd:element name="Quenchant">
  <xsd:complexType>
    <xsd:attribute name = "id"  type ="xsd:ID"  use ="required"/>

    ……..
  </xsd:complexType>
</xsd:element>
```

Figure 6: Key Constraint Example

4. **Occurrence Constraint:** This constraint is used to declare the minimum and maximum permissible occurrences of an element. It is written as "minOccurs = x" and "maxOccurs = y" where "x" and "y" denote the minimum and maximum occurrences respectively [10]. A "maxOccurs" value of "unbounded" implies that there is no limit on the number of times this element can occur within a schema. A "minOccurs" value greater than "0" implies that it is necessary to include this element within the schema at least once. Figure 7 shows an example of this constraint [5]. With reference to this example, it is clear that the "Cooling Rate" element must occur at least 8 times and that there is no upper bound on the number of times it can occur. This is because in the domain, the value of cooling rate must be stored at least at 8 points in order to adequately capture the details of the Quenching process. However cooling rate values may be recorded at hundreds or even thousands of points and thus there is no upper limit on the number of values that can be stored for cooling rate [6]. In the case of graphs, however, it is not necessary that at least one graph be stored. It is essential though to keep the number of graphs stored less than three per instance of the process [5]. This is required as per the domain. Generally the two graphs stored in Quenching are the cooling rate curve and heat transfer coefficient curve. In addition, a cooling curve may be stored [6]. A cooling curve is a plot of temperature (T) versus time (t) during Quenching or rapid cooling. A cooling rate curve is a plot of cooling rate (dT/dt) versus time (t). A heat transfer coefficient curve is a plot of heat transfer coefficient (hc) versus temperature (T), where a heat transfer coefficient characterizes a Quenching process.

```
<xsd:element name="Cooling Rate" minOccurs="8"
  maxOccurs="unbounded">
    ……..
</xsd:element>

<xsd:element name="Graphs" minOccurs="0"
  maxOccurs="3">
    ……..
</xsd:element>
```

Figure 7: Occurrence Constraint Example

5. RETRIEVAL USING XQUERY

XQuery is a query language for XML [7] developed by the World Wide Web Consortium (W3C). XQuery can be used to retrieve XML data. Hence it can query information stored using a domain-specific markup language that has been designed with XML tags. It is thus advisable to design the extension to the markup language to facilitate retrieval using XQuery. A few suggestions for doing this are as follows.

1. **Encourage users to store data in a case-sensitive manner:** XQuery is case-sensitive [7]. Hence it is useful to place emphasis on case when storing the data using the domain-specific markup language and its extension. This helps to obtain correct retrieval of information.

2. **Use tags to enhance querying efficiency:** In many domains, it is possible to anticipate a typical user query. For example, in Heat Treating, a user is very likely to retrieve the details of a quenchant in terms of its name, type and manufacturer without requesting information about the quenchant properties. Thus it is advisable to add a level of abstraction around the name-related tags and the property-related tags. This is done using additional tags such as <NameDetails> and <PropertyDetails> [5]. The XQuery expression to retrieve information for a name-related query can then be constructed such that it gets the name details in a single traversal of the path, namely, <NameDetails> </NameDetails>. In the absence of this abstraction, the XQuery expression to get these details would have contained a greater number of tags, with additional levels of nesting. Thus introducing abstraction by anticipating typical user queries

enhances the efficiency of querying. An example of this abstraction is shown in Figure 8.

```
<Quenchant>
   <NameDetails>
      <Manufacturer> </Manufacturer>
      <Type>
         <Subtype> </Subtype>
      </Type>
   </NameDetails>
   <PropertyDetails>
      ……..
   </PropertyDetails>
</Quenchant>
```

Figure 8: Abstraction for Readability

6. CONCLUSIONS

Several aspects of extending domain-specific markup languages have been discussed in this paper. These include the motivation for extension, the steps involved in the process of extension, the features of the language and the retrieval considerations with a query language such as XQuery. An extension to a domain-specific markup language MatML has been proposed at CHTE, WPI. This extension captures the semantics of Heat Treating, a sub-area in Materials Science. Discussions with NIST, that developed MatML, are ongoing to incorporate the proposed extension as a standard. Most of the examples in this paper are from the proposed MatML extension. A few other examples are from the medical and financial domains. The focus of this paper has thus been the issues in extending domain-specific markup languages.

7. ACKNOWLEDGMENTS

The authors thank the members of the Database Systems Research Group (DSRG) in the Department of Computer Science at WPI for their feedback regarding this work. We also thank the Quenching Research Team in the Department of Materials Science for their co-operation in designing the Heat Treating extension to MatML. The support and encouragement of the Center for Heat Treating Excellence (CHTE) and its member companies for our work is gratefully acknowledged.

8. REFERENCES

[1] K. Yokota, T. Kunishima and B. Liu "Semantic Extensions of XML for Advanced Applications", **IEEE Australian Computer Science Communications Proceedings of Workshop on Information Technology for Virtual Enterprises**, Vol. 23, No. 6, January 2001, pp. 49 – 57.

[2] D.J. Bouvier, *"Versions and Standards of HTML",* **ACM SIGAPP Applied Computing Review**, Vol. 3, No. 2, October 1995, pp. 9 – 15.

[3] J. Guo, K. Araki, K. Tanaka, J. Sato, M. Suzuki, A. Takada, T. Suzuki, Y. Nakashima and H. Yoshihara, "The Latest MML (Medical Markup Language) Version 2.3 --- XML based Standard for Medical Data Exchange/ Storage", **Journal of Medical Systems**, Vol. 27, No. 4, August 2003, pp. 357 – 366.

[4] E.F. Begley, "MatML Version 3.0 Schema", NIST 6939, **National Institute of Standards and Technology Report**, USA, January 2003.

[5] A. Varde, E. Rundensteiner, M. Mani, M. Maniruzzaman and R. Sisson Jr., "Augmenting MatML with Heat Treating Semantics", **ASM International's Symposium on Web-Based Materials Property Databases**, October 2004, To Appear.

[6] G. Totten, C. Bates and N. Clinton, **Handbook of Quench Technology and Quenchants**, ASM International, 1993.

[7] S. Boag, M. Fernández, D. Florescu, J. Robie and J. Simeon, "XQuery 1.0: An XML Query Language" **W3C Working Draft**, November 2003.

[8] R. Ramakrishnan and J. Gehrke, **Database Management Systems**, McGraw Hill Companies. 2000.

[9] J. Seward and D. Logue, "Handbook of Modern Finance", **WG&L Financial**, 2004.

[10] S. Davidson, W. Fan, C. Hara, J. Qin, "Propagating XML Constraints to Relations", **International Conference on Data Engineering,** March 2003, pp. 543 – 552.