The Query Language Position Paper of the XSL Working Group

(Draft 11/18/98)

Status of this Document

This document expresses the XSL working group's position for the upcoming W3C Query Language Workshop. It has been discussed and reviewed by the XSL working group and reflects the majority view expressed by those present at the November 10-12, 1998, face-to-face meeting.

Introduction

The Extensible Stylesheet Language (XSL) has facilities that could serve as a basis for an XML query language. The XSL working group believes that it would be constructive for the W3C to first look in-house for technologies that might seed a W3C-endorsed query language. It is important to the working group that the W3C strive to maximize the reuse of technology within the W3C.

XSL is one of the in-house technologies that the W3C ought to examine. Where XSL and the query language provide the same functionality, it would be beneficial to do so through compatible technologies. The W3C might accomplish this by borrowing from and building on XSL's pattern matching and document transformation capabilities.

It is the consensus of the XSL working group that the W3C should pursue an XML query language that is based on XSL's pattern and transformation facilities. The working group recommends that further development of these facilities remain within the XSL working group and that a coordination group take responsibility for coordinating query requirements with other working groups. The coordination group would either strive to ensure that a single query language meets the requirements of all working groups or that a common query model underlies all W3C query languages.

Query Facilities in XSL

There are two points where the functionality of XSL and the functionality of query languages seem to overlap: both retrieve data from an underlying data source, and both construct a new data source from retrieved data. The following table compares these facilities for SQL and XSL:

	Facilities in SQL	Facilities in XSL
Information Retrieval	SELECT identifies values to retrieve	Select patterns identify nodes to process
	WHERE identifies retrieval constraints	Pattern qualifiers specify node criteria
Construction	Queries return virtual database tables	Stylesheets create new XML documents
	ORDER BY orders returned rows	xsl:sort orders created elements

It is reasonable to expect that a query language for XML will also have these facilities. For example, the XML-QL submission defines a WHERE clause for information retrieval and a CONSTRUCT clause for construction. The XSL specification partitions these facilities out as separate technologies that are not tied to the concerns of print and display, making it possible to reuse the technologies for other endeavors.

Patterns

XSL uses a "pattern" language for information retrieval. The pattern language is a concise syntax for identifying nodes within an XML document. XSL defines two kinds of patterns: match patterns and select patterns. A select pattern retrieves all nodes that meet the criteria that the pattern specifies. This process is known as "selection." A match pattern asks a question about a node to determine whether the pattern identifies the node. For each node in the source document the XSL processor examines each template (see the description of templates below) to determine whether the match pattern in the template identifies the node. In the case where there is more than one template that matches, a single one is selected based on criteria that XSL defines. This process is known as "matching."

Match and select patterns use the same underlying syntax, but match patterns are constrained to a subset of this syntax. Match patterns are used in construction and select patterns are used for information retrieval.

To further clarify the distinction between match patterns and select patterns, consider the example pattern product/sku. As a match pattern this example tests a node to determine whether it is a sku element that is a child of a product element. As a select pattern the example queries a node for the immediate children that are product elements and then queries each of these product elements for the immediate children that are sku elements. The result of applying the match pattern is a Boolean value, while the result of applying the select pattern is a set of sku elements.

The pattern language syntax is modeled on file system path names. An XML document may be thought of as a file system. Each element in the document may be thought of as a directory, and descendant elements may be thought of as subdirectories. Slashes partition element names to name a path through the elements of the document. XSL embellishes this notation with qualifiers. Qualifiers specify constraints on the elements named in the path so that elements may be selected as a function of their content and attributes.

The following examples briefly illustrate XSL’s pattern language:

Pattern	Explanation
`/`	Identifies the root of the document, which contains a single child that is the document's root element.
`category/product`	Identifies product elements that are immediate children of the category element. ('/' selects immediate children.)
`category/product/sku`	Identifies sku elements that are immediate children of product elements that are immediate children of a category element.
`category//sku`	Identifies sku elements found anywhere within the category element. ('//' selects descendant elements.)
`//sku`	Identifies all sku elements found anywhere within the document. (Here '//' is relative to the root.)
`product[attribute(size)="10"]`	Identifies product elements having a size attribute whose value is "10". The brackets enclose a qualifier.
`product[attribute(size)="10"]/sku`	Identifies sku elements that are immediate children of product elements having a size attribute whose value is "10". The brackets enclose a qualifier.

Templates

XSL examines a source document and applies templates to the nodes of the source document to create a result document. XSL's stylesheets use xsl:template elements to define how the result document is constructed. The content of such an element is a template for content that is to be inserted into the result document. Descendents of xsl:template that belong to the XSL namespace control the construction process, while all other descendant elements are copied into the result document. As previously described, the match attribute of xsl:template contains a match pattern that identifies the elements to which the template applies.

The following examples briefly illustrate XSL's templates:

Template	Explanation

`<xsl:template match="sku">`
`    SKU: <xsl:value-of/>`
`</xsl:template>`
	For each instance of the sku element, constructs text consisting of the word "SKU:" followed by the textual content of the sku element.

`<xsl:template match="//product">`
`  NAME: <xsl:value-of select="attribute(name)"/>`
`  <xsl:apply-templates select="sku"/>`
`</xsl:template>`
	When used in combination with the previous template, constructs the name of each product and all SKUs associated with each product, where the name is drawn from the value of a product element's name attribute.

`<xsl:template match="category">`
`  <SKUS CATEGORY="{attribute(name)}">`
`    <xsl:apply-templates select="product/sku"/>`
`  </SKUS>`
`</xsl:template>`
	When used in combination with the first template, constructs one SKUS element for each category element, gives the element a name equal to the value of the name attribute of the associated category element, and lists all of the SKUs contained within the category element's child product elements.

These examples demonstrate the use of XSL patterns and templates to retrieve information from a document and to construct a result document that provides a virtual view of the source document. Consider the last of the above example templates. This template may be thought of as a query for all SKUs and the categories to which the SKUs belong, where the resulting SKUs are grouped by category.

Comparing Stylesheets and Queries

Stylesheets and XML queries have significant overlap because both may yield XML documents. However, the emphasis placed on certain features tends to differ between a query language and a stylesheet language.

For example, a query language generally places more emphasis on:

- more complex information retrieval
- the naming and reuse of intermediate query results
- the ability to perform joins, whether between documents or within a document
- the ability to retreive information from multiple documents

While a stylesheet language normally places more importance on:

- handling recursive and heterogeneous XML data
- controlling whitespace
- numbering and other positional features
- supporting multiple output documents

In addition to these transformational requirements, a stylesheet language also defines a set of formatting capabilities. XSL clearly demarcates the boundary between transformation and formatting by separating these concerns into two different XML namespaces and by defining formatting objects so that neither patterns nor templates are dependent on them.

Why Start with XSL?

Suppose someone has an XML document and needs to create another XML document from it. If both XSL and the query language are capable of generating XML from XML, the person has a bit of a dilemma. Either technology would suffice. Let's say the person decides to go with XSL. A few weeks or months down the road this person may find that the query language was the proper choice and now must replace all occurrences of XSL stylesheets with XML queries. Substitute "person" with "W3C working group" and it becomes easy to see the dilemmas we could be creating for the W3C in the future.

This scenario suggests that the W3C should at least attempt to ensure that it recommends compatible technologies for similar functionality. Here are some reasons for borrowing technology from XSL:

1. When there are fewer standards for a given task, vendor support is less divided among the standards, and vendor products are more interoperable.
2. The fewer technologies users have to learn, the easier and faster it is for users to learn new products, and the less time and money companies have to spend educating users.
3. The W3C can get a head start by starting with related technologies that it already espouses.
4. XSL uses separate technologies for information retrieval and document construction, which allows the information retrieval mechanism to be used in places where construction is not required.

The W3C should be able to accrue the above benefits by using XSL technologies as the foundation of the query language and by building on this foundation to satisfy requirements that exceed XSL query requirements.

XSL Query Requirements

Regardless of the query language that the W3C defines, XSL may benefit by using the language or some subset of it. To allow the XSL working group to have this option, the group suggests that the query language should satisfy the requirements that XSL's pattern language satisfies. These requirements follow:

1. The query language should be at least as expressive as XSL's pattern language. That is, the query language should be able to identify any set of nodes that the pattern language is capable of identifying.
2. The query language should be usable for node selection. ("Selection" is defined above.)
3. The query language should be usable for node matching. ("Matching" is defined above.)
4. The query language should be able to perform relative navigation. For example, XSL's ancestor() term may appear in select patterns to identify the ancestor of the node under query.

This paper does not attempt to identify the details of XSL's query requirements, since XSL's pattern language already embodies these requirements, and since the above references to the pattern language should convey the requirements.

Summary of Position

The XSL working group believes that it has already developed technologies that are relevant to a query language for XML. These technologies are XSL templates and XSL patterns. The working group does not have a clear picture of how a query language effort could best leverage from XSL technologies, but the working group is willing to make the following assertions:

1. The query language should use XSL's patterns as the basis for information retrieval.
2. The query language should use XSL's templates as the basis for materializing query results.
3. The query language should be a least as expressive as XSL is currently.
4. Development of the pattern and transformation languages should remain in the XSL working group.
5. A coordination group should ensure either that a single query language satisfies all working group requirements or that all W3C query languages share an underlying query model.

The most fundamental concern of the working group is that the W3C should strive for compatibility among technologies that have overlapping functionality. Compatibility broadens vendor support, ensures interoperability, minimizes user learning curves, and promotes reuse within the W3C.